aws

開發人員指南

Amazon DynamoDB

API 版本 2012-08-10

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

Amazon DynamoDB: 開發人員指南

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

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

什麼是 Amazon DynamoDB？ 1
特性 2
無伺服器 2
NoSQL 2
全受管 2
任何規模下的個位數毫秒效能 2
使用案例 2
功能 3
全域資料表的多主動複寫 3
ACID 交易 4
變更資料擷取 4
次要索引 4
服務整合 4
無伺服器整合 4
匯入與匯出資料至 Amazon S3 5
零 ETL 整合 5
快取 5
安全 5
恢復能力 6
全域資料表 6
持續備份與時間點復原 (PITR) 6
隨需備份與還原 6
存取 DynamoDB 7
定價 7
開始使用 7
DynamoDB 入門 8
初次使用者資源 8
Amazon DynamoDB 初次使用者的其他最佳實務 10
AWS CLI 資源 10
程式設計資源 10
存取 DynamoDB 11
使用主控台 11
使用 AWS CLI 12
使用 API 14

使用 NoSQL Workbench 14
IP 位址範圍 15
網路通訊協定第 6 版 (IPv6) 的雙堆疊端點 15
先決條件 16
設定 DynamoDB 17
設定 DynamoDB (Web 服務) 17
設定 DynamoDB 本機版 (可下載版本) 20
步驟 1：建立資料表 41
步驟 2：寫入資料 93
步驟 3：讀取資料 125
步驟 4：更新資料 152
步驟 5：查詢資料 184
步驟 6：（選用） 清除 222
後續步驟 240
Console-to-Code 241
運作方式 241
搭配 DynamoDB 使用 Console-to-Code 的優勢 242
範例使用案例 242
開始使用 242
運作方式 243
速查表 243
初始 設定 243
SDK 或 CLI 243
基本動作 244
命名規則 245
服務配額基本概念 246
其他資訊 247
核心元件 248
資料表、項目與屬性 248
主索引鍵 251
次要索引 252
DynamoDB Streams 255
DynamoDB API 256
控制平面 257
資料平面 257
DynamoDB Streams 259

交易 ... 259
支援的資料類型和命名規則 ... 260
命名規則 ... 260
資料類型 ... 261
資料類型描述項 ... 265
DynamoDB 資料表類別 ... 266
DynamoDB 的分割區與資料分配 ... 266
資料分佈：分割區索引鍵 ... 267
資料分佈：分割區索引鍵與排序索引鍵 ... 268
了解如何從 SQL 過渡到 NoSQL ... 270
關聯式或 NoSQL？ ... 271
存取與驗證 ... 272
建立資料表 ... 275
取得資料表的資訊 ... 277
將資料寫入資料表 ... 279
從資料表讀取資料 ... 283
管理索引 ... 291
修改資料表中的資料 ... 296
從資料表刪除資料 ... 299
移除資料表 ... 301
Amazon DynamoDB 學習資源與工具 ... 302
寫程式和視覺化工具 ... 302
規範性指引 ... 302
知識中心 ... 304
部落格文章、儲存庫和指南 ... 304
資料建模和設計模式 ... 305
培訓課程 ... 305
讀取和寫入 ... 307
DynamoDB 讀取一致性 ... 307
最終一致讀取 ... 307
高度一致性讀取 ... 307
全域資料表讀取一致性 ... 308
讀取和寫入操作 ... 308
讀取操作耗用量 ... 308
寫入操作耗用量 ... 309
DynamoDB 輸送容量 ... 311

隨需模式 311
佈建模式 311
DynamoDB 隨需容量模式 312
讀取請求單位與寫入請求單位 313
初始輸送量與擴展特性 313
DynamoDB 隨需資料表的最大輸送量 313
佈建容量模式 315
讀取與寫入容量單位 316
選擇初始輸送量設定 316
DynamoDB 自動擴展 317
使用自動擴展管理輸送容量 318
預留容量 346
暖輸送量 347
檢查資料表的暖輸送量 348
提高資料表的暖輸送量 351
建立暖輸送量較高的資料表 358
暖輸送量應用情境 368
高載容量與調適型容量 370
高載容量 370
調適型容量 371
切換容量模式 372
佈建容量模式至隨需容量模式 372
隨需容量模式至佈建容量模式 373
使用 DynamoDB 編寫程式 375
DynamoDB 的 AWS SDK 支援概觀 375
AWS 支援帳戶型端點的 SDK 377
使用 DynamoDB 的程式設計介面 378
更高階程式設計界面 384
執行程式碼範例 440
低階 API 447
使用 Python 編寫程式 453
關於 Boto 453
Boto 文件 454
用戶端和資源層 454
使用 batch_writer 457
其他程式碼範例 458

工作階段和執行緒安全 458
Config 459
錯誤處理 463
日誌 465
事件勾點 466
分頁和分頁程式 467
等待程式 469
使用 JavaScript 編寫程式 469
關於 適用於 JavaScript 的 AWS SDK 470
適用於 JavaScript 的 AWS SDK V3 470
JavaScript 文件 470
抽象層 470
Marshall 公用程式函數 473
讀取項目 473
條件式寫入 475
分頁 476
Config 478
等待程式 480
錯誤處理 481
日誌 483
考量事項 483
使用 進行程式設計 AWS SDK for Java 2.x 484
關於 AWS SDK for Java 2.x 485
開始使用 486
適用於 Java 2.x 的 SDK 文件 494
支援的介面 495
其他程式碼範例 508
同步和非同步程式設計 509
HTTP 用戶端 509
Config 511
錯誤處理 516
AWS 請求 ID 517
日誌 517
分頁 519
資料類別註釋 521
錯誤處理 521

錯誤元件 ... 521
交易性錯誤 ... 522
錯誤訊息和錯誤碼 ... 522
應用程式中的錯誤處理 ... 527
錯誤重試和指數退避 ... 527
批次操作和錯誤處理 ... 528
使用 AWS SDKs ... 529
使用 DynamoDB ... 531
處理資料表 ... 531
資料表上的基本操作 ... 531
選擇 DynamoDB 資料表類別時的注意事項 ... 540
標籤和標籤 ... 541
使用全域資料表 ... 546
一致性模式 ... 546
帳戶組態 ... 546
核心概念 ... 548
相同帳戶全域資料表 ... 551
多帳戶全域資料表 ... 608
全域資料表計費 ... 628
全域資料表版本 ... 630
全域資料表最佳實務 ... 640
處理項目 ... 641
項目大小和格式 ... 642
讀取項目 ... 643
寫入項目 ... 644
傳回值 ... 646
批次操作 ... 647
原子計數器 ... 650
條件式寫入 ... 650
使用表達式 ... 655
存留時間 (TTL) ... 693
查詢資料表 ... 722
掃描資料表 ... 730
PartiQL 查詢語言 ... 737
使用項目：Java ... 784
處理項目：.NET ... 814

使用索引 846
全域次要索引 849
本機次要索引 938
搭配交易使用 986
運作方式 987
將 IAM 搭配交易使用 995
範例程式碼 998
使用串流 1002
選項 1003
使用 Kinesis Data Streams 1004
使用 DynamoDB Streams 1020
使用 DAX 的記憶體內加速 1098
DAX 使用案例 1099
DAX 使用須知 1100
運作方式 1100
DAX 如何處理請求 1102
項目快取 1103
查詢快取 1104
DAX 叢集元件 1105
節點 1105
叢集 1106
區域與可用區域 1107
參數群組 1107
安全群組 1108
叢集 ARN 1108
叢集端點 1108
節點端點 1108
子網路群組 1109
事件 1109
Maintenance window (維護時段) 1109
建立 DAX 叢集 1111
為 DAX 建立可存取 DynamoDB 的 IAM 服務角色 1111
DAX 和 IPv6 1113
使用 AWS CLI 1115
使用主控台 1120
一致性模式 1125

DAX 叢集節點之間的一致性 1125
DAX 項目快取行為 1126
DAX 查詢快取行為 1129
強烈一致和交易讀取 1130
負快取 1130
寫入策略 1130
使用 DAX 用戶端開發 1133
教學課程：執行範例應用程式 1134
修改現有應用程式以使用 DAX 1195
管理 DAX 叢集 1196
管理 DAX 叢集的 IAM 許可 1196
擴展 DAX 叢集 1199
自訂 DAX 叢集設定 1200
配置 TTL 設定 1201
DAX 的標記支援 1203
AWS CloudTrail 整合 1204
刪除 DAX 叢集 1204
監控 DynamoDB Accelerator 1204
DAX 監控工具 1205
使用 CloudWatch 進行監控 1206
使用 記錄 DAX 操作 AWS CloudTrail 1227
DAX T3/T2 爆量執行個體 1227
DAX T2 執行個體系列 1227
DAX T3 執行個體系列 1228
DAX 存取控制 1228
DAX 的 IAM 服務角色 1229
允許 DAX 叢集存取的 IAM 政策 1231
案例研究：存取 DynamoDB 和 DAX 1232
存取 DynamoDB，但不可使用 DAX 存取 1233
存取 DynamoDB 和 DAX 1235
透過 DAX 存取 DynamoDB，但不直接存取 DynamoDB 1241
DAX 靜態加密 1243
使用 啟用靜態加密 AWS 管理主控台 1245
DAX 傳輸中加密 1246
使用 DAX 的服務連結角色 1247
DAX 的服務連結角色許可 1247

建立 DAX 的服務連結角色 1248
編輯 DAX 的服務連結角色 1249
刪除 DAX 的服務連結角色 1249
跨 AWS 帳戶存取 DAX 1250
設定 IAM 1251
設定 VPC 1254
修改 DAX 用戶端以允許跨帳戶存取權 1256
DAX 叢集調整大小指南 1260
概觀 1261
預估流量 1261
負載測試 1262
建立資料模型 1264
處理項目收集 1265
使用物品集合組織資料來加快查詢 1266
資料建模基礎 1267
單一資料表設計 1267
多資料表設計 1269
資料建模建置區塊 1271
複合排序索引鍵 1272
多租用 1273
稀鬆索引 1274
生存時間 1275
封存生存時間 1276
垂直分割 1277
寫入碎片 1279
資料建模結構描述設計套件 1281
先決條件 1281
社交網路 1282
遊戲設定檔 1291
投訴管理系統 1299
週期性付款 1316
裝置狀態更新 1321
線上商店 1334
關聯式模型 1357
傳統關聯式資料庫模型 1357
DynamoDB 如何免除 JOIN 操作的需求 1359

DynamoDB 交易如何消除寫入程序的負荷 1359
首要步驟 1361
範例 1363
移轉至 DynamoDB 1374
移轉原因 1374
移轉時的考量 1375
運作方式 1376
移轉工具 1377
選擇移轉策略 1378
離線移轉 1379
混合移轉 1381
線上 - 以 1:1 方式移轉每個資料表 1382
線上 - 使用自訂暫存資料表移轉 1383
NoSQL Workbench 1386
下載 1387
資料模型建立工具 1388
建立新模型 1388
匯入現有的模型 1390
編輯現有的模型 1390
新增範例資料 1391
新增和驗證存取模式 1391
從 CSV 匯入 1392
面向 1393
彙總檢視 1394
匯出模型 1394
遞交資料模型 1395
操作建置器 1396
連線至資料集 1396
建置操作 1398
複製資料表 1408
匯出至 CSV 1409
範例資料模型 1410
員工資料模型 1410
開發論壇資料模型 1411
音樂資料庫資料模型 1411
滑雪渡假村資料模型 1411

信用卡優惠資料模型 1412
書籤資料模型 1412
版本歷史記錄 1413
備份與還原 1418
時間點備份 1419
開始之前 1419
啟用時間點復原 1420
隨需備份 1424
運作方式 1424
備份資料表 1427
還原資料表 1429
刪除資料表備份 1435
使用 IAM 1436
備份的帳單 1443
運作方式 1443
DynamoDB 備份帳單範例 1444
還原 1447
使用時間點復原還原資料表 1447
還原 DynamoDB 資料表至某個時間點 1449
使用 AWS 備份 1454
運作方式 1455
建立備份 1457
複製備份 1459
還原資料表 1460
刪除備份 1461
與 DynamoDB AWS Backup 相比， 管理的隨需備份 1461
程式碼範例 1463
基本概念 1466
Hello DynamoDB 1467
了解基本概念 1477
動作 1634
案例 2037
使用 DAX 加速讀取 2039
進階全域次要索引案例 2048
建置應用程式以將資料提交至 DynamoDB 資料表 2051
比較多個值與單一屬性 2052

有條件地更新項目的 TTL 2074
連線至本機執行個體 2081
計數表達式運算子 2082
建立 REST API 以追蹤 COVID-19 資料 2115
建立傳訊應用程式 2115
建立無伺服器應用程式來管理相片 2116
建立具有全域次要索引的資料表 2120
建立啟用暖輸送量的資料表 2124
建立 Web 應用程式以追蹤 DynamoDB 資料 2134
建立 websocket 聊天應用程式 2136
建立具有 TTL 的項目 2136
建立和管理 MRSC 全域資料表 2143
建立和管理示範 MREC 的全域資料表 2163
使用 PartiQL DELETE 刪除資料 2172
偵測映像中的 PPE 2178
使用 PartiQL INSERT 插入資料 2179
從瀏覽器調用 Lambda 函式 2183
管理全域次要索引 2184
管理資源型政策 2187
監控 DynamoDB 效能 2189
執行進階查詢操作 2190
執行清單操作 2225
執行映射操作 2257
執行集合操作 2281
使用多批 PartiQL 陳述式查詢資料表 2312
使用 PartiQL 查詢資料表 2374
使用全域次要索引查詢資料表 2429
使用 begins_with 條件查詢資料表 2438
使用日期範圍查詢資料表 2444
使用複雜篩選條件表達式查詢資料表 2452
使用動態篩選條件表達式查詢資料表 2460
使用篩選條件表達式和限制查詢資料表 2469
查詢具有巢狀屬性的資料表 2473
查詢具有分頁的資料表 2483
查詢具有高度一致性讀取的資料表 2493
使用 PartiQL SELECT 查詢資料 2498

查詢 TTL 項目 2503
使用日期和時間模式查詢資料表 2507
儲存 EXIF 和其他映像資訊 2521
設定屬性型存取控制 2522
了解更新表達式順序 2525
更新資料表的暖輸送量設定 2556
更新項目的 TTL 2562
使用 PartiQL UPDATE 更新資料 2566
使用 API Gateway 來調用 Lambda 函式 2572
使用 Step Functions 調用 Lambda 函式 2574
使用文件模型 2575
使用高階物件持久性模型 2591
使用原子計數器操作 2600
使用條件式操作 2623
使用表達式屬性名稱 2653
使用排程事件來調用 Lambda 函式 2680
使用本機次要索引 2682
使用串流和存留時間 2685
使用全域資料表和多區域複寫最終一致性 (MREC) 2689
使用資源標記 2692
使用資料表加密 2695
無伺服器範例 2697
使用 DynamoDB 觸發條件調用 Lambda 函式 2697
使用 DynamoDB 觸發條件報告 Lambda 函式的批次項目失敗 2706
AWS 社群貢獻 2717
建置和測試無伺服器應用程式 2718
安全 2720
AWS 受管政策 2721
AWS 受管政策 2721
AWS 受管政策：AmazonDynamoDBFullAccess_v2 2721
AmazonDynamoDBReadOnlyAccess 2722
AWS 受管政策的 DynamoDB 更新 2723
資源型政策 2724
建立資料表 2725
連接資源型政策 2731
將政策附加至串流 2736

移除資源型政策 2738
跨帳戶存取權 2739
封鎖公開存取 2741
API 操作 2743
IAM 授權 2750
範例 2751
考量事項 2758
最佳實務 2759
屬性型存取控制 2760
為什麼要使用 ABAC？ 2762
條件索引鍵 2762
考量事項 2762
在 DynamoDB 中啟用 ABAC 2763
使用 ABAC 2765
範例使用案例 2767
疑難排解 2780
資料保護 2781
靜態加密 2782
強化 DynamoDB 連線安全性 2805
IAM 2807
身分和存取權管理 2807
使用條件 2834
法規遵循驗證 2859
恢復能力 2859
基礎設施安全性 2860
使用 VPC 端點 2860
AWS PrivateLink 適用於 DynamoDB 2869
Amazon VPC 端點的類型 2870
使用 AWS PrivateLink for Amazon DynamoDB 時的考量事項 2871
建立 Amazon VPC 端點 2871
存取 Amazon DynamoDB 介面端點 2871
從 DynamoDB 介面端點存取 DynamoDB 資料表及控制 API 操作 2872
更新內部部署 DNS 組態 2874
建立 Amazon VPC 端點政策 2876
搭配 AWS 管理主控台 私有存取使用 DynamoDB 端點 2877
AWS PrivateLink 適用於 DynamoDB Streams 2878

AWS PrivateLink for DAX ... 2884
組態與漏洞分析 ... 2886
安全最佳實務 ... 2886
預防性安全最佳實務 ... 2887
偵測性安全最佳實務 ... 2889
監控和記錄 ... 2892
監控計畫 ... 2892
效能基準 ... 2892
整合服務 ... 2893
自動化監控工具 ... 2893
監控指標 ... 2893
如何使用 DynamoDB 指標？ ... 2894
在 CloudWatch 主控台中檢視指標 ... 2895
在 中檢視指標 AWS CLI ... 2895
指標和維度 ... 2897
在 DynamoDB 中建立 CloudWatch 警示 ... 2924
記錄操作 ... 2927
CloudTrail 中的 DynamoDB 資訊 ... 2928
了解資料 DynamoDB 日誌檔案項目 ... 2932
Contributor Insights ... 2951
運作方式 ... 2951
開始使用 ... 2960
使用 IAM ... 2970
最佳實務 ... 2976
NoSQL 設計 ... 2976
NoSQL 與 RDBMS 對比 ... 2977
兩個主要概念 ... 2977
一般方法 ... 2977
NoSQL Workbench ... 2978
DynamoDB Well-Architected Lens ... 2979
成本最佳化 ... 2979
進行 Amazon DynamoDB Well-Architected Lens 檢閱 ... 3021
Amazon DynamoDB Well-Architected Lens 的支柱 ... 3022
分割區索引鍵設計 ... 3023
分佈工作負載 ... 3024
寫入碎片 ... 3025

有效率地上傳資料 3026
排序索引鍵設計 3028
版本控制 3028
次要索引 3029
一般準則 3030
稀鬆索引 3032
聚合 3034
GSI 過載 3040
GSI 碎片 3042
建立一個複本 3046
大型項目 3047
壓縮 3047
垂直分割 3048
使用 Amazon S3 3048
時間序列資料 3049
時間序列資料的設計模式 3049
時間序列資料表範例 3049
多對多關係 3050
相鄰清單 3051
具體化圖形 3052
查詢及掃描 3056
掃描效能 3057
避免尖峰 3057
平行掃描 3059
資料表設計 3060
使用全域資料表 3060
關鍵事實 3061
關於 MREC 的重點說明 3062
關於 MRSC 的重點說明 3063
使用案例 3064
寫入模式 3065
DynamoDB 的路由策略 3070
疏散程序 3074
輸送容量規劃 3077
準備作業核對清單 3078
結論和資源 3084

控制平面 3085
大量資料操作 3085
條件式批次更新 3086
高效大量操作 3090
處理並行更新 3091
樂觀鎖定 3092
使用交易進行估算鎖定 3095
分散式鎖定 3097
選擇並行控制策略 3100
帳單與用量報告 3101
輸送容量 3104
串流 3107
儲存 3107
備份與還原 3108
資料傳輸 3111
CloudWatch 3111
DAX 3112
將 DynamoDB 資料表從一個帳戶移轉至另一帳戶 3113
使用 AWS Backup 進行跨帳戶備份與還原以移轉資料表 3114
使用匯出至 S3 並從 S3 匯入的方式移轉資料表 3116
DAX 規範指引 3118
評估 DAX 的適用性 3118
設定 DAX 用戶端 3120
設定 DAX 叢集 3121
調整 DAX 叢集大小 3126
部署叢集 3131
叢集作業 3132
監控 DAX 3135
將 DynamoDB 與其他 AWS 服務搭配使用 3137
與 Amazon Cognito 整合 3137
與 Amazon Redshift 整合 3139
使用 CMK 時的跨帳戶整合考量 3140
與 Amazon Redshift 的零 ETL 整合 3144
使用 COPY 將資料從 DynamoDB 載入 Amazon Redshift 3152
與 Amazon EMR 整合 3153
概觀 3154

教學課程：使用 Amazon DynamoDB 和 Apache Hive 3154
在 Hive 中建立外部資料表 3163
處理 HiveQL 陳述式 3166
查詢 DynamoDB 中的資料 3167
在 Amazon DynamoDB 之中複製和貼入資料 3169
效能調校 3182
與 S3 整合 3187
從 Amazon S3 匯入 3187
匯出至 Amazon S3 3206
與 Amazon SageMaker 資料湖倉整合 3231
與 Amazon SageMaker 資料湖倉的零 ETL 整合 3231
與 Amazon OpenSearch Service 整合 3233
運作方式 3233
建立整合 3234
後續步驟 3234
處理中斷變更 3234
與 OpenSearch Service 的零 ETL 整合 3237
與 Amazon EventBridge 整合 3240
運作方式 3240
透過主控台建立整合 3241
後續步驟 3243
與 Amazon MSK 整合 3243
運作方式 3244
整合範例 3244
後續步驟 3250
整合最佳實務 3251
建立快照 3251
變更資料擷取 3251
搭配 DynamoDB 使用代理式 AI 3253
DynamoDB 的生成式 AI 使用案例 3253
DynamoDB 的生成式 AI 部落格 3254
利用 DynamoDB 與 OpenSearch Service 的零 ETL 整合 3254
LangGraph 檢查點存放區 3255
主要功能 3255
先決條件 3255
安裝 3256

基本使用 3256
生產組態 3257
DynamoDB 資料表組態 3258
所需的 IAM 許可 3259
非同步用量 3260
清除 3260
錯誤處理 3261
其他資源 3261
配額和條件限制 3262
執行配額管理任務 3262
存取 DynamoDB 配額 3262
在主控台中檢視目前的配額 3263
使用 AWS CLI 檢視目前的配額 3263
請求提高配額 3265
配額 3266
讀取/寫入輸送量 3266
預留容量 346
表格 3269
全域資料表 3269
次要索引 3270
每份資料表的預估次要索引屬性 3270
DynamoDB Streams 3271
從 Amazon S3 匯入 3271
資料表匯出至 Amazon S3 3271
備份與還原 3271
Contributor Insights 3271
Constraints 3272
讀取/寫入容量模式 3272
次要索引 3270
分割區索引鍵和排序索引鍵 3273
命名規則 3274
資料類型 3275
項目 3275
屬性 3276
表達式參數 3276
DynamoDB 交易 3277

DynamoDB Streams 3278
DynamoDB Accelerator (DAX) 3278
API 特定限制 3279
DynamoDB 靜態加密 3281
API 參考 3282
疑難排解 3283
內部伺服器錯誤 3283
調查內部伺服器錯誤 3283
將內部伺服器錯誤的影響降至最低 3284
提升營運覺察 3284
Latency (延遲) 3288
限流 3291
限流診斷 3293
解決指南 3299
GSI 寫入限流與背壓 3319
CloudWatch 限流度量指標 3321
附錄 3323
使用 DynamoDB 對 SSL/TLS 連線建立問題進行疑難排解 3323
測試您的應用程式或服務 3323
測試您的用戶端瀏覽器 3324
更新您的軟體應用程式用戶端 3324
更新您的用戶端瀏覽器 3324
手動更新您的憑證套件 3325
在 DynamoDB 中使用的資料表和資料範例 3325
範例資料檔案 3326
建立範例資料表和上傳資料 3339
建立範例資料表和上傳資料 - Java 3339
建立範例資料表和上傳資料 - .NET 3349
使用 的範例應用程式 適用於 Python (Boto3) 的 AWS SDK 3362
步驟 1：在本機上部署及測試 3363
步驟 2：檢查資料模型和實作詳細資訊 3367
步驟 3：在生產環境中部署 3376
步驟 4：清除資源 3384
DynamoDB 中的保留字 3385
AWS 適用於 Java 的 SDK 1.x 範例 3398
DAX 與 Java 開發套件第 1 版 3398

修改適用於 Java 1.x 的開發套件的現有應用程式來使用 DAX 3410
使用適用於 Java 1.x 的開發套件查詢全域次要索引 3415
AWS 適用於 Go 的 SDK 1.x 範例 3418
Go 和 DAX 3419
AWS 適用於 Node.js 的 SDK 2.x 範例 3421
Node.js 和 DAX 3421
文件歷史紀錄 3432
舊版更新 3456
舊版功能 3479
全域資料表 2017.11.29 版 (舊版) 3479
運作方式 3480
最佳實務和要求 3484
建立全域資料表 (2017.11.29 版) 3487
監控全域資料表 3491
對全域資料表使用 IAM 3492
舊版低階 DynamoDB API (2011-12-05) 3495
BatchGetItem 3496
BatchWriteItem 3503
CreateTable 3509
DeleteItem 3516
DeleteTable 3522
DescribeTables 3526
GetItem 3530
ListTables 3534
PutItem 3536
Query 3542
Scan 3555
UpdateItem 3571
UpdateTable 3579
舊版 DynamoDB 條件式參數 3583
AttributesToGet 3585
AttributeUpdates 3586
ConditionalOperator 3588
預期 3589
KeyConditions 3593
QueryFilter 3596

ScanFilter 3598
使用舊式參數撰寫條件 3600
..... mmmdcix

什麼是 Amazon DynamoDB？

Amazon DynamoDB 是無伺服器、全受管的分散式 NoSQL 資料庫，具有任何規模的單一位數毫秒效能。

DynamoDB 能協助您克服關聯式資料庫在擴展與營運上的複雜性。DynamoDB 專為在各種規模下需維持一致效能的作業型工作負載所設計並最佳化。例如，無論您有 1000 萬或 1 億使用者，DynamoDB 都會為購物車使用案例提供一致的單一位數毫秒效能。於 2012 年推出的 DynamoDB，持續協助您擺脫關聯式資料庫，同時降低成本並提升大規模效能。

各種規模、產業及地區的客戶皆使用 DynamoDB 建置現代化的無伺服器應用程式，能從小規模起步並擴展至全球。DynamoDB 可彈性擴展以支援幾乎任意大小的資料表，同時維持一致的個位數毫秒效能與高可用性。

在 Amazon Prime Day 等活動期間，DynamoDB 為多個高流量的 Amazon 系統與服務提供支援，包括 Alexa、Amazon.com 網站及所有 Amazon 履行中心。在這類活動中，DynamoDB API 已處理數兆次來自 Amazon 系統與服務的呼叫。DynamoDB 持續為數百位客戶提供資料表服務，其尖峰流量可達每秒超過 50 萬次請求。另有數百位客戶的資料表容量超過 200 TB，DynamoDB 每小時可處理超過 10 億次請求。

主題

- DynamoDB 的主要特性
- DynamoDB 使用案例
- DynamoDB 的主要功能
- 服務整合
- 安全
- 恢復能力
- 存取 DynamoDB
- DynamoDB 定價
- DynamoDB 入門

DynamoDB 的主要特性

無伺服器

使用 DynamoDB，您無需佈建伺服器，也不必修補、管理、安裝、維護或操作任何軟體。DynamoDB 提供零停機維護。DynamoDB 無版本區分 (主要版、次要版或修補版)，且無需設定維護時段。

DynamoDB 的隨需容量模式採用依用量計費定價，針對讀寫請求只需依實際使用量付費。使用隨需容量模式時，DynamoDB 會自動調整資料表容量以因應負載變化，並在無須人工管理的情況下維持效能。此模式亦可縮減至零，因此當資料表無流量時，您不需支付輸送量費用，且不會發生冷啟動。

NoSQL

作為 NoSQL 資料庫，DynamoDB 專為提供優於傳統關聯式資料庫的效能、可擴展性、可管理性與彈性而設計。為支援多樣化的使用案例，DynamoDB 同時支援鍵值資料模型與文件資料模型。

與關聯式資料庫不同，DynamoDB 不支援 JOIN 運算子。建議您對資料模型進行取消正規化，以降低查詢往返次數與所需處理資源。作為 NoSQL 資料庫，DynamoDB 提供強大的讀取一致性與 ACID 交易，可支援企業級應用程式的建置。

全受管

作為全受管資料庫服務，DynamoDB 代為處理管理資料庫的繁瑣作業，讓您得以專注於為客戶創造價值。它負責處理設定、組態、維護、高可用性、硬體佈建、安全性、備份、監控等各項作業。因此，當您建立 DynamoDB 資料表後，系統可立即投入生產工作負載。DynamoDB 持續提升可用性、可靠性、效能、安全性與功能，且無需升級或停機。

任何規模下的個位數毫秒效能

DynamoDB 專為提升關聯式資料庫效能與可擴展性而設計，能在任何規模下提供個位數毫秒效能。為達成此規模與效能，DynamoDB 針對高效能工作負載進行最佳化，並提供可促進高效資料庫運作的 API。為確保大規模效能，DynamoDB 省略了在擴展時效率低落或效能不佳的功能，例如 JOIN 操作。無論您有 100 或 1 億使用者，DynamoDB 都會為您的應用程式提供一致的單一位數毫秒效能。

DynamoDB 使用案例

各種規模、產業及地區的客戶皆使用 DynamoDB 建置現代化的無伺服器應用程式，能從小規模起步並擴展至全球。DynamoDB 特別適用於在任何規模下需維持一致效能且幾乎不需操作負擔的使用案例。以下列出幾種可使用 DynamoDB 的典型使用案例：

- 金融服務應用程式 – 假設您是一家金融服務公司，正在建置如即時交易與路由、貸款管理、權杖生成及交易分類帳等應用程式。使用 DynamoDB 全域資料表，您的應用程式可以回應事件， AWS 區域 並以快速的本機讀取和寫入效能提供所選流量。

  DynamoDB 適用於對可用性要求極高的應用程式。它免除了為提升儲存容量或輸送量、進行版本控制與授權所需的手動擴展執行個體負擔。

  您可使用 DynamoDB 交易，透過單一請求在一或多個資料表間達成原子性、一致性、隔離性與耐久性 (ACID)。(ACID) 交易特別適用於涉及金融交易處理或訂單履行的工作負載。DynamoDB 可在工作負載增減時即時調整，讓您能依市場條件 (例如交易時段) 高效擴展資料庫。
- 遊戲應用程式 – 若您是遊戲開發商，可將 DynamoDB 應用於遊戲平台的各個層面，例如遊戲狀態、玩家資料、工作階段歷程與排行榜。選擇 DynamoDB，可同時獲得可擴展性、一致效能，以及無伺服器架構帶來的操作便利。DynamoDB 特別適合支援成功遊戲所需的橫向擴充架構。它可快速調整遊戲的輸送量，無論擴增或縮減 (可縮減至零，且無冷啟動)。無論您是針對尖峰流量橫向擴充，還是在遊戲使用量低時向後擴展，此可擴展性都會最佳化架構的效率。
- 串流應用程式 – 媒體與娛樂公司使用 DynamoDB 作為內容與內容管理服務的中繼資料索引，或用於提供近即時的體育統計資訊。他們亦使用 DynamoDB 執行使用者監看清單與書籤服務，並處理每日數十億筆客戶事件以生成推薦結果。這些客戶受惠於 DynamoDB 所提供的可擴展性、效能與彈性。DynamoDB 會隨工作負載變化自動擴展，使串流媒體應用可支援各種需求層級。

若想進一步了解各產業客戶如何運用 DynamoDB，請參閱 Amazon DynamoDB 客戶與這是我的架構。

DynamoDB 的主要功能

全域資料表的多主動複寫

全域資料表提供跨所選 資料的多重主動複寫 AWS 區域 ，可用性為 99.999%。全域資料表提供全受管解決方案，可部署多區域、多主動資料庫，而無需自行建置或維護複寫機制。使用全域資料表，您可以指定資料表的可用 AWS 區域 位置。DynamoDB 會即時將資料變更複寫至所有相關資料表。

全域部署的應用程式可在所選區域中以本機方式存取資料，以達成個位數毫秒的讀寫效能。由於全域資料表採多主動設計，因此無需指定主資料表。這表示在跨區域執行容錯移轉時，不會發生複雜或延遲的移轉，也不會導致資料庫停機。

ACID 交易

DynamoDB 專為關鍵任務級工作負載設計。它支援 (ACID) 交易，以滿足需要複雜商業邏輯的應用程式需求。DynamoDB 提供原生的伺服端交易支援，簡化開發人員在單一或多個資料表間進行協調式「全有或全無」變更的體驗。

事件驅動架構的變更資料擷取

DynamoDB 支援近乎即時地串流項目層級的變更資料擷取 (CDC) 紀錄。它為變更資料擷取 (CDC) 提供兩種串流模型：DynamoDB Streams 與 Kinesis Data Streams for DynamoDB。每當應用程式在資料表中建立、更新或刪除項目時，串流便會以近乎即時的方式記錄各項目層級變更的時間順序。這使得 DynamoDB Streams 特別適合事件驅動架構的應用程式，用以擷取並處理變更。

次要索引

DynamoDB 提供建立全域次要索引與本機次要索引的選項，使您能以替代索引鍵查詢資料表資料。使用這些次要索引，您可以透過非主索引鍵的屬性來存取資料，最大化資料存取的彈性。

服務整合

DynamoDB 與數個 廣泛整合 AWS 服務 ，協助您從資料中獲得更多價值、消除無差別的繁重工作，以及大規模操作工作負載。一些範例包括： AWS CloudFormation、Amazon CloudWatch、Amazon S3、 AWS Identity and Access Management (IAM) 和 AWS Auto Scaling。以下章節說明您可透過 DynamoDB 進行的部分服務整合：

無伺服器整合

為了建構端對端無伺服器應用程式，DynamoDB 與多項無伺服器 AWS 服務進行原生整合。例如，您可以將 DynamoDB 與 AWS Lambda 整合以建立觸發條件，這些觸發條件是自動回應 DynamoDB Streams 事件的程式碼片段。透過觸發條件，您可以建立對 DynamoDB 資料表中資料變更即時回應的事件驅動應用程式。為最佳化成本，您可以篩選由 Lambda 自 DynamoDB 串流處理的事件。

下列清單列出數個與 DynamoDB 進行無伺服器整合的範例：

- AWS AppSync 用於建立 GraphQL API
- Amazon API Gateway 用於建立 REST API。
- Lambda 適用於無伺服器運算。

- Amazon Kinesis Data Streams 用於變更資料擷取 (CDC)

匯入與匯出資料至 Amazon S3

將 DynamoDB 與 Amazon S3 整合可讓您輕鬆將資料匯出至 Amazon S3 儲存貯體，用於分析與機器學習。DynamoDB 支援完整資料表匯出與增量匯出，可在指定期間匯出變更、更新或刪除的資料。您也可以從 Amazon S3 匯入資料至新的 DynamoDB 資料表。

零 ETL 整合

DynamoDB 支援與 Amazon Redshift 的零 ETL 整合，並可透過 OpenSearch 擷取管道與 Amazon DynamoDB 搭配使用。這些整合可讓您在 DynamoDB 資料表上執行複雜分析，並運用進階搜尋功能。例如，您可以在 DynamoDB 資料上執行全文搜尋、向量搜索與語意搜尋。零 ETL 整合不會對在 DynamoDB 上執行的生產工作負載造成影響。

快取

DynamoDB Accelerator (DAX) 是專為 DynamoDB 所設計的全受管、高可用性快取服務。DAX 可提供高達 10 倍的效能提升，將延遲從毫秒縮短至微秒，並支援每秒數百萬次請求。DAX 會自動處理將記憶體內加速新增至 DynamoDB 資料表所需的所有繁重作業，無需您手動管理快取失效、資料載入或叢集維運。

安全

DynamoDB 使用 IAM 來協助您安全控制對 DynamoDB 資源的存取。透過 IAM，您可以集中管理權限，控制哪些 DynamoDB 使用者可存取哪些資源。您可以使用 IAM 來控制能通過身分驗證 (登入) 和授權使用資源的 (具有許可) 的人員。由於 DynamoDB 採用 IAM，存取 DynamoDB 時無需使用者名稱或密碼。由於無須管理複雜的密碼輪替政策，因此可簡化您的整體安全防護。透過 IAM，您也可以啟用精細層級存取控制，在屬性層級提供授權。您也可以定義資源型政策，並搭配 IAM Access Analyzer 與封鎖公開存取 (BPA) 功能，以簡化原則管理。

預設情況下，DynamoDB 會對所有靜態客戶資料進行加密。靜態加密透過使用儲存在 AWS Key Management Service (AWS KMS) 中的加密金鑰，來提升資料的安全性。您可以透過靜態加密，建立符合嚴格加密合規和法規要求，而且對安全性要求甚高的應用程式。當您存取加密的資料表，DynamoDB 會以透明方式解密資料表資料。您不必變更任何代碼或應用程式來使用或管理加密的資料表。DynamoDB 會持續提供與預期相同的單一位數毫秒延遲，而且所有 DynamoDB 查詢都能順暢地處理加密的資料。

您可以指定 DynamoDB 是否應使用 AWS 擁有的金鑰（預設加密類型）AWS 受管金鑰、 或客戶受管金鑰來加密使用者資料。使用 AWS擁有的 KMS 金鑰進行的預設加密不需額外付費。若需用戶端加密，您可使用AWS 資料庫加密 SDK。

DynamoDB 亦符合多項合規標準，包括 HIPAA、PCI DSS 與 GDPR，協助您符合法規要求。

恢復能力

預設情況下，DynamoDB 會自動在三個可用區域間複寫您的資料，以確保高耐用性與 99.99% 可用性 SLA。DynamoDB 亦提供進階功能，協助您達成業務持續性與災難復原目標。

DynamoDB 提供下列功能，以支援您的資料復原能力與備份需求：

功能

- 全域資料表
- 持續備份與時間點復原 (PITR)
- 隨需備份與還原

全域資料表

DynamoDB 全域資料表提供 99.999% 可用性 SLA，並具備多區域復原能力。這有助於您建立具復原能力的應用程式，並針對最低復原時間點目標 (RTO) 與復原點目標 (RPO) 進行最佳化。全域資料表亦可與 AWS Fault Injection Service (AWS FIS) 整合，以在全域工作負載上進行故障注入測試。例如，您可暫停全域資料表的複寫至任一複本資料表。

持續備份與時間點復原 (PITR)

連續備份可讓您達到每秒級精細度，並能啟動時間點復原。使用時間點復原，您可將資料表還原至過去 35 天內任一秒的狀態。您可以將復原期間設為 1 至 35 天。

持續備份與啟動時間點還原不會佔用佈建容量。此功能也不會影響應用程式的效能或可用性。

隨需備份與還原

隨需備份與還原可讓您建立資料表完整備份，以支援長期保留及符合法規需求。備份不會影響資料表效能，且可支援任意規模的資料表。透過AWS Backup 整合，您可以使用 AWS Backup 自動排程、複製、標記和管理 DynamoDB 隨需備份的生命週期。使用 AWS Backup，您可以跨帳戶和區域複製隨需備份，並將較舊的備份轉換為冷儲存，以最佳化成本。

存取 DynamoDB

您可透過 AWS 管理主控台、DynamoDB 專用 NoSQL Workbench、 AWS Command Line Interface 或 DynamoDB API 操作 DynamoDB。

如需詳細資訊，請參閱存取 DynamoDB。

DynamoDB 定價

DynamoDB 針對資料表的讀取、寫入及儲存收費，並對啟用的選用功能額外計費。DynamoDB 提供兩種容量模式：隨需與佈建，分別對應不同的讀寫計費方式。

DynamoDB 亦屬於永遠免費方案，可提供 25 GB 儲存空間。永遠免費方案亦包含 25 個佈建寫入與 25 個佈建讀取容量單位 (WCU、RCU)，可支援每月 2 億次請求。

如需詳細資訊，請參閱 Amazon DynamoDB 定價。

DynamoDB 入門

若您首次使用 DynamoDB，建議先閱讀以下主題：

- DynamoDB 入門 – 引導您完成 DynamoDB 設定、建立範例資料表及上傳資料。本主題也提供使用 AWS 管理主控台、 AWS CLI NoSQL Workbench 和 DynamoDB APIs 執行一些基本資料庫操作的相關資訊。
- DynamoDB 核心元件 – 說明 DynamoDB 的基本概念。
- 使用 DynamoDB 進行設計和架構的最佳實務 – 提供有關 NoSQL 設計、DynamoDB Well-Architected Lens、資料表設計及多項其他 DynamoDB 功能的建議。這些最佳實務可協助您在使用 DynamoDB 時最大化效能，並降低輸送量成本。

我們也建議您參閱下列教學課程，這些課程說明了完整的端到端流程，協助您熟悉 DynamoDB 的操作與概念。您可使用永遠免費方案功能完成這些教學課程。

- 使用 Amazon DynamoDB 建立和查詢 NoSQL 資料表
- 使用 NoSQL 鍵 - 值資料存放區建置應用程式

若需關於移轉至 DynamoDB 的資源、工具與策略資訊，請參閱移轉至 DynamoDB。若要閱讀最新的部落格與白皮書，請參閱 Amazon DynamoDB 資源。

DynamoDB 入門

您將在下列各節中了解如何連線、建立和管理 DynamoDB 資料表。

開始之前，您應該熟悉 Amazon DynamoDB 中的基本概念。您可以在 什麼是 Amazon DynamoDB？ 中取得快速概觀，並在 Amazon DynamoDB 的核心元件 中更深入了解。然後請繼續前往先決條件。

> **Note**
>
> 註冊時 AWS，您可以使用 AWS 免費方案開始使用 DynamoDB。如果您尚未超過 Amazon DynamoDB 的免費方案優惠，完成本節範例無需支付任何費用。如果已經超過，則從您建立資料表到刪除資料表的這段時間，將產生標準 DynamoDB 使用費。
> 如果您不想要註冊免費方案帳戶，則可以在電腦上設定 DynamoDB 本機版 (可下載版本)。可下載版本可讓您在本機開發和測試應用程式，而無需註冊 AWS 帳戶或存取 DynamoDB Web 服務。

主題

- 為初次使用者提供的 Amazon DynamoDB 資源
- 存取 DynamoDB
- 先決條件
- 設定 DynamoDB
- 步驟 1：在 DynamoDB 建立資料表
- 步驟 2：將資料寫入 DynamoDB 資料表
- 步驟 3：從 DynamoDB 資料表讀取資料
- 步驟 4：更新 DynamoDB 資料表中的資料
- 步驟 5：查詢 DynamoDB 資料表中的資料
- 步驟 6：(選用) 刪除 DynamoDB 資料表以清除資源
- 繼續了解 DynamoDB
- 使用 Console-to-Code 產生 Amazon DynamoDB 的基礎設施程式碼

為初次使用者提供的 Amazon DynamoDB 資源

建議您在開始前先閱讀以下章節，並在需要時用做參考。

Example 服務特色和定價 提供 DynamoDB、常見使用案例、服務重點和定價的一般產品概觀。	Example DynamoDB 資源 向您介紹 服務、資料建模的概念以及核心功能和功能的影片、教學課程和方案指引。
Example 開始使用 有關設定 DynamoDB、建立範例資料表和上傳資料的資訊。	Example DynamoDB 基本課程 教導 DynamoDB 基礎知識的免費數位課程，包括資料表設計、資料類型和基本操作。
Example DynamoDB 模組 簡短的重點影片教學課程，說明重要的 DynamoDB 概念和功能。	Example DynamoDB 程式碼範例儲存庫 各種程式設計語言的實用 DynamoDB 程式碼範例。
Example 免費 DynamoDB 訓練 AWS 提供免費的數位培訓課程，涵蓋 DynamoDB 概念、功能和最佳實務。	Example 適用於 DynamoDB 的 NoSQL Workbench 提供資料建模、資料視覺化和查詢開發功能的統一視覺化工具。
Example DynamoDB 設計模式 具有實際程式碼範例的不同使用案例的最佳實務和資料建模範例。	Example 實際操作教學課程 中的Step-by-step教學課程 AWS 管理主控台 ，引導您完成常見的 DynamoDB 任務。
Example 遷移至 DynamoDB 將資料庫遷移至 DynamoDB 的程序、工具和策略概觀。	Example AWS 適用於 DynamoDB 的 Well-Architected Lens

	使用 DynamoDB 設計和操作可靠、安全、高效且經濟實惠應用程式的架構最佳實務。

Amazon DynamoDB 初次使用者的其他最佳實務

完成前面的章節之後，請閱讀這些章節：

- DynamoDB 輸送容量

  概述 DynamoDB 兩種可用的輸送量模式，以及為應用程式選擇適當容量模式的考量因素。隨需模式是大多數 DynamoDB 工作負載的預設和建議輸送量選項。

- 最佳實務

  識別並解決問題，以在使用 DynamoDB 時達到最高效能並將成本降至最低。

AWS CLI 資源

如果您想要使用 AWS 命令列界面 (AWS CLI)，您可以使用這些文件來協助您開始使用：

- AWS CLI 文件

  本節提供下載 AWS CLI、在系統上進行 AWS CLI 工作，以及提供 AWS 登入資料的相關資訊。

- AWS CLI DynamoDB 文件

  此個別文件涵蓋 AWS CLI DynamoDB 命令的所有 ，包括語法和範例。

程式設計資源

您可利用各種熱門的程式設計語言編寫應用程式語言以使用 DynamoDB API。以下是一些可用資源：

- 適用於 Amazon Web Services 的工具

  AWS 提供多種支援 DynamoDB 的軟體開發套件 (SDKs)。您可以使用 Java、.NET、PHP、Ruby 和其他語言來編寫 DynamoDB 程式碼。這些 SDK 可將您的請求格式化為 DynamoDB、剖析回應並提供重試邏輯和錯誤處理，協助大幅簡化您的應用程式開發。

- DynamoDB API 參考

如果您不想使用 AWS SDKs，可以直接使用 DynamoDB API 與 DynamoDB 互動。這份文件涵蓋了所有 DynamoDB API 操作，包括語法和範例。您可以在本節中找到建立與驗證請求與處理回應的疑難排解秘訣和資訊。

存取 DynamoDB

您可以使用 AWS 管理主控台、 AWS Command Line Interface (AWS CLI) 或 DynamoDB API 存取 Amazon DynamoDB。

主題

- 使用主控台
- 使用 AWS CLI
- 使用 API
- 使用適用於 DynamoDB 的 NoSQL Workbench
- IP 位址範圍
- 網路通訊協定第 6 版 (IPv6) 的雙堆疊端點

使用主控台

您可以在 https://console.aws.amazon.com/dynamodb/home：// 存取 AWS 管理主控台 適用於 Amazon DynamoDB 的 。

以下是您可以在 DynamoDB 主控台中執行的一些動作：

- 管理資料表：建立、更新和刪除資料表。容量計算機可協助預估容量需求。
- 與資料互動：檢視、新增、更新和刪除資料表中的項目。管理存留時間 (TTL) 設定。
- 監控和分析：檢視儀表板、監控和設定警示，以及分析 DynamoDB 資料表的指標和警示。
- 最佳化和擴展：管理次要索引、串流、觸發條件、預留容量和其他可強化 DynamoDB 使用量的進階功能。

DynamoDB 主控台提供管理 DynamoDB 資源的完整介面。我們鼓勵您存取主控台並與其互動，以進一步了解。

使用 AWS CLI

您可以使用 AWS Command Line Interface (AWS CLI) 從命令列控制多個 AWS 服務，並透過指令碼將其自動化。您可以使用 AWS CLI 進行臨機操作，例如建立資料表。其也可以用於在公用程式指令碼中嵌入 Amazon DynamoDB 操作。

您必須先取得存取金鑰 ID 和私密存取金鑰，才能 AWS CLI 搭配 DynamoDB 使用 。如需詳細資訊，請參閱授予程式設計存取權。

如需 中可用於 DynamoDB 的所有命令的完整清單 AWS CLI，請參閱 AWS CLI 命令參考。

下載和設定 AWS CLI

AWS CLI 您可以在 https：//http://aws.amazon.com/cli 取得 。它可在 Windows、macOS，或 Linux 上執行。下載 之後 AWS CLI，請依照下列步驟進行安裝和設定：

1. 前往《AWS Command Line Interface 使用者指南》。
2. 請遵循安裝 AWS CLI 及設定 AWS CLI 的說明進行。

AWS CLI 搭配 DynamoDB 使用

命令列格式包含 DynamoDB 操作名稱，隨後接著該操作的參數。 AWS CLI 支援參數值以及 JSON 的速記語法。

例如，以下命令會建立名為 Music 的資料表。分割區索引鍵為 Artist，而排序索引鍵為 SongTitle。(為確保易讀性，本節的長命令以分行顯示。)

```
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
```

以下命令會為資料表新增新的項目。這些範例混合使用速記語法和 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
```

在命令列上難以編寫有效的 JSON。但是 AWS CLI 能夠讀取 JSON 檔案。例如，請試想下列 JSON 程式碼片段，其存放在名為 key-conditions.json 的檔案中。

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

您現在可以使用 AWS CLI發行 `Query` 請求。在此範例中，key-conditions.json 檔案的內容會做為 `--key-conditions` 參數使用。

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

AWS CLI 搭配 DynamoDB local 使用

AWS CLI 也可以與在電腦上執行的 DynamoDB 本機 （ 可下載版本 ） 互動。若要啟用此功能，請為每個命令新增下列參數：

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

下列範例使用 AWS CLI 列出本機資料庫中的資料表。

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

若 DynamoDB 使用的連接埠號碼並非預設值 (8000)，請相應地修改 `--endpoint-url` 的數值。

> **Note**
>
> AWS CLI 無法使用 DynamoDB 本機 （ 可下載版本 ） 做為預設端點。因此，您必須為每個命令指定 `--endpoint-url`。

使用 API

您可以使用 AWS 管理主控台 和 AWS Command Line Interface 以互動方式使用 Amazon DynamoDB。但是，若要充分利用 DynamoDB，您可以使用 AWS 軟體開發套件撰寫應用程式的程式碼。

AWS SDKs 在 Java、瀏覽器中的 JavaScript、.NET、Node.js、PHP、Python、Ruby、C++、Go、Android 和 iOS 中提供 DynamoDB 的廣泛支援。

您必須先取得 AWS 存取金鑰 ID 和私密存取金鑰，才能搭配 DynamoDB 使用 AWS SDKs。如需詳細資訊，請參閱設定 DynamoDB (Web 服務)。

如需使用 AWS SDKs 進行 DynamoDB 應用程式程式設計的高階概觀，請參閱 使用 DynamoDB 和 AWS SDKs程式設計。

使用適用於 DynamoDB 的 NoSQL Workbench

您也可以透過下載並使用 DynamoDB 專用 NoSQL Workbench 來存取 DynamoDB。

Amazon DynamoDB 專用 NoSQL Workbench 是用於現代資料庫開發和操作的跨平台用戶端 GUI 應用程式。適用於 Windows、macOS 和 Linux。NoSQL Workbench 是視覺化開發工具，提供了資料模型建立、資料視覺化和查詢開發功能，協助您設計、建立、查詢及管理 DynamoDB 資料表。NoSQL

Workbench 現在包含 DynamoDB 本機版做為安裝程序的選用部分，可讓您更輕鬆地在 DynamoDB 本機版中建立資料模型。若要深入了解 DynamoDB 本機版及其要求，請參閱 設定 DynamoDB 本機版 (可下載版本)。

> **Note**
>
> 適用於 DynamoDB 的 NoSQL Workbench 目前不支援使用雙重驗證 (2FA) 設定的 AWS 登入。

建立資料模型

借助 DynamoDB 專用 NoSQL Workbench，您可以使用滿足您應用程式資料存取模式的現有資料模型，來建置新的資料模型或設計模型。您也可以在程序結束時，匯入及匯出設計好的資料模型。如需詳細資訊，請參閱使用 NoSQL Workbench 建立資料模型。

建立操作

NoSQL Workbench 提供強大的圖形使用者介面供您開發及測試查詢。您可以使用 operation builder (操作建置器) 來檢視、探索及查詢即時資料集。您也可以使用結構式操作建置器來建立及執行資料平面操作。支援投射及條件表達式，並讓您使用多種語言產生範本程式碼。如需詳細資訊，請參閱使用 NoSQL Workbench 探索資料集與建立操作。

IP 位址範圍

Amazon Web Services (AWS) 會以 JSON 格式發布目前的 IP 位址範圍。若要檢視目前範圍，請下載 ip-ranges.json。如需詳細資訊，請參閱《 AWS 一般參考》中的 AWS IP 位址範圍。

若要尋找您可以用來存取 DynamoDB 資料表和索引的 IP 位址範圍，請在 ip-ranges.json 檔案中搜尋下列字串：`"service": "DYNAMODB"`。

> **Note**
>
> IP 位址範圍不適用於 DynamoDB Streams 或 DynamoDB Accelerator (DAX)。

網路通訊協定第 6 版 (IPv6) 的雙堆疊端點

DynamoDB 提供與 IPv4 和 IPv6 相容的雙堆疊端點。端點命名慣例為：

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

如需 DynamoDB 端點和區域可用性的完整清單，請參閱《 AWS 一般參考指南》中的 Amazon DynamoDB 端點和配額主題。

如需設定 AWS CLI 以使用雙堆疊端點的詳細資訊，請參閱《 AWS 命令列界面指南》中的設定 以使用所有 AWS 服務的雙堆疊端點一節。

如需設定 SDK 用戶端以使用雙堆疊端點的詳細資訊，請參閱 AWS SDKs雙堆疊和 FIPS 端點主題。

在搭配 IPv6 使用 DynamoDB 之前，您必須更新用於 IP 地址篩選的 IAM 使用者角色或資源型政策，以包含 IPv6 地址範圍。未考慮 IPv6 地址的 IP 地址篩選政策可能會導致存取問題。如需詳細資訊，請參閱 AWS Identity and Access Management 指南中的 IP 地址條件運算子一節。

先決條件

開始 Amazon DynamoDB 教學課程之前，請先了解您可以在 存取 DynamoDB 中存取 DynamoDB 的方式。然後透過 Web 服務或 設定 DynamoDB 中的本機下載版本來設定 DynamoDB。之後請繼續前往步驟 1：在 DynamoDB 建立資料表。

> **Note**
>
> - 如果您計劃只透過 與 DynamoDB 互動 AWS 管理主控台，則不需要 AWS 存取金鑰。完成註冊 AWS 中的步驟，然後繼續前往 步驟 1：在 DynamoDB 建立資料表。
> - 如果您不想要註冊免費方案帳戶，則可以設定 DynamoDB 本機版 (可下載版本)。然後繼續步驟 1：在 DynamoDB 建立資料表。
> - 在 Linux 和 Windows 的終端機中使用 CLI 命令時存在差異。以下指南介紹針對 Linux 終端機 (包括 macOS) 格式化的命令，以及針對 Windows CMD 格式化的命令。請選擇最適合您正在使用的終端機應用程式的命令。

設定 DynamoDB

除了 Amazon DynamoDB Web 服務之外， AWS 還提供可在電腦上執行的可下載版 DynamoDB。可下載的版本對於開發和測試您的程式碼來說很有幫助。它可讓您在本機撰寫及測試應用程式，而不用存取 DynamoDB Web 服務。

本節的主題說明如何設定 DynamoDB (可下載版本) 和 DynamoDB Web 服務。

主題

- 設定 DynamoDB (Web 服務)
- 設定 DynamoDB 本機版 (可下載版本)

設定 DynamoDB (Web 服務)

使用 Amazon DynamoDB Web 服務：

1. 註冊 AWS。
2. 取得 AWS 存取金鑰 （ 用於以程式設計方式存取 DynamoDB)。

> **Note**
>
> 如果您計劃只透過 與 DynamoDB 互動 AWS 管理主控台，則不需要 AWS 存取金鑰，您可以直接跳到 使用主控台。

3. 設定您的憑證 (用來以程式設計方式存取 DynamoDB)。

註冊 AWS

若要使用 DynamoDB 服務，您必須擁有 AWS 帳戶。如果您還沒有帳戶，系統會在您註冊時提示您建立帳戶。除非您使用註冊的任何 AWS 服務，否則您無需付費。

註冊 AWS

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

授予程式設計存取權

您必須先擁有程式設計存取權，才能以程式設計方式或透過 AWS Command Line Interface (AWS CLI) 存取 DynamoDB。如果您打算只使用 DynamoDB 主控台，則不需要程式設計存取權。

如果使用者想要與 AWS 外部互動，則需要程式設計存取 AWS 管理主控台。授予程式設計存取權的方式取決於正在存取的使用者類型 AWS。

若要授予使用者程式設計存取權，請選擇下列其中一個選項。

哪個使用者需要程式設計存取權？	到	根據
IAM	（建議）使用主控台登入資料做為臨時登入資料，以簽署對 AWS CLI、 AWS SDKs程式設計請求。 AWS APIs	請依照您要使用的介面所提供的指示操作。 • 如需 AWS CLI，請參閱AWS Command Line Interface 《使用者指南》中的登入以進行 AWS 本機開發。 • AWS SDKs，請參閱 AWS SDKs 和工具參考指南中的登入以進行 AWS 本機開發。
人力資源身分 (IAM Identity Center 中管理的使用者)	使用暫時登入資料簽署對 AWS CLI、 AWS SDKs程式設計請求。 AWS APIs	請依照您要使用的介面所提供的指示操作。 • 如需 AWS CLI，請參閱AWS Command Line Interface 《使用者指南》中的設定 AWS CLI 要使用 AWS IAM Identity Center的 。

哪個使用者需要程式設計存取權？	到	根據
		• AWS SDKs、工具和 AWS APIs，請參閱 AWS SDKs 和工具參考指南中的 IAM Identity Center 身分驗證。
IAM	使用暫時登入資料簽署對 AWS CLI、 AWS SDKs程式設計請求。 AWS APIs	遵循《IAM 使用者指南》中將臨時登入資料與 AWS 資源搭配使用的指示。
IAM	(不建議使用) 使用長期憑證簽署對 AWS CLI、 AWS SDKs 或 AWS APIs程式設計請求。	請依照您要使用的介面所提供的指示操作。 • 如需 AWS CLI，請參閱AWS Command Line Interface 《 使用者指南》中的使用 IAM 使用者憑證進行身分驗證。 • AWS SDKs和工具，請參閱 AWS SDKs和工具參考指南中的使用長期憑證進行身分驗證。 • 對於 AWS APIs，請參閱《IAM 使用者指南》中的管理 IAM 使用者的存取金鑰。

設定您的憑證

您必須先設定登入資料以啟用應用程式的授權 AWS CLI，才能以程式設計方式或透過 存取 DynamoDB。

有幾種方式可以執行此作業。例如，您可以手動建立憑證檔案，存放您的存取金鑰 ID 與私密存取金鑰。您也可以使用 AWS CLI 命令`aws configure`自動建立 檔案。或者，您也可以使用環境變數。如需設定登入資料的詳細資訊，請參閱程式設計特定的 AWS SDK 開發人員指南。

若要安裝和設定 AWS CLI，請參閱 使用 AWS CLI。

與其他 DynamoDB 服務整合

您可以將 DynamoDB 與許多其他 AWS 服務整合。如需詳細資訊，請參閱下列內容：

- 將 DynamoDB 與其他 AWS 服務搭配使用
- CloudFormation 適用於 DynamoDB
- 搭配 DynamoDB 使用 AWS Backup
- AWS Identity and Access Management (IAM) 和 DynamoDB
- 搭配 Amazon DynamoDB 使用 AWS Lambda

設定 DynamoDB 本機版 (可下載版本)

可下載版 Amazon DynamoDB 能讓您在本機開發及測試應用程式，不用存取 DynamoDB Web 服務。而且資料庫在您的電腦上可獨自運作。當您準備好要在生產環境中部署您的應用程式時，您可移除程式碼中的本機端點，並將其指向 DynamoDB Web 服務。

擁有這個本機版本可讓您節省輸送量、資料儲存體和數據傳輸費用。此外，您在開發應用程式時，不需要網際網路連線。

DynamoDB 本機版可以作為 Apache Maven 依存項目 (需要 JRE) 或 Docker 映像檔下載。

如果您偏好使用 Amazon DynamoDB Web 服務，請參閱 設定 DynamoDB (Web 服務)。

主題

- 在本機電腦上部署 DynamoDB
- DynamoDB 本機版使用須知
- DynamoDB 本機版的版本歷史記錄
- DynamoDB 本機版中的遙測功能

在本機電腦上部署 DynamoDB

> **Note**
>
> - DynamoDB 本機版提供三種版本：v3.x (目前)、v2.x (舊版) 和 v1.x (已棄用)。
> - DynamoDB v3.x 建議用於本機測試和開發用途。

- 從 DynamoDB 本機版 V2.x 移轉至 V3.x 需要將匯入陳述式從 `com.amazonaws.services.dynamodbv2` 更新為 `software.amazon.dynamodb`，並更新 Maven 使用者的 Maven 相依性。
- 如果您要將使用適用於 Java v1.x 之 SDK 的應用程式移轉至適用於 Java 2.x 的 SDK，請遵循AWS 適用於 Java 2.x 的 SDK 的步驟。

下載 DynamoDB 本機版

請依照這些步驟在您的電腦上安裝並執行 DynamoDB。

在電腦上設定 DynamoDB

1. 從下列其中一個位置免費下載 DynamoDB 本機版。

下載連結	檢查總和
.tar.gz | .zip	.tar.gz.sha256 | .zip.sha256

> **Important**
>
> 若要在您的電腦上執行 DynamoDB v2.6.0 或更高版本，您必須有 Java 執行階段環境 (JRE) 第 17.x 版或更新版本。應用程式無法在舊版的 JRE 上執行。

2. 在您下載封裝後，請解壓縮內容，並將解壓縮的目錄複製到您選擇的位置。
3. 若要在您的電腦上啟動 DynamoDB，請開啟命令提示視窗，導覽至您解壓縮 `DynamoDBLocal.jar` 的目錄，然後輸入下列命令。

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

> **Note**
>
> 如果您要使用 Windows PowerShell，請務必括住參數名稱或完整名稱及值，如下所示：`java -D"java.library.path=./DynamoDBLocal_lib" -jar DynamoDBLocal.jar`

DynamoDB 會處理傳入的請求，直到您將其停止。若要停止 DynamoDB，請在命令提示字元中按 Ctrl+C。
DynamoDB 預設使用連接埠 8000。如果無法使用連接埠 8000，此命令就會擲出例外狀況。如需完整的 DynamoDB 執行時期選項清單 (包括 -port)，請輸入此命令。

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

4. 以程式設計方式或透過 AWS Command Line Interface (AWS CLI) 存取 DynamoDB 之前，您必須先設定您的憑證，以便啟用您的應用程式授權。可下載版 DynamoDB 需要所有憑證才能運作，如下列範例所示。

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

   您可以使用 AWS CLI 的 `aws configure` 命令來設定憑證。如需詳細資訊，請參閱 使用 AWS CLI。

5. 開始寫入應用程式。若要使用 存取在本機執行的 DynamoDB AWS CLI，請使用 `--endpoint-url` 參數。例如，使用下列命令來列出 DynamoDB 資料表。

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

將 DynamoDB 本機版執行為 Docker 映像檔

Amazon DynamoDB 的可下載版本作為 Docker 映像檔提供。如需詳細資訊，請參閱 dynamodb-local。若要查看您目前的 DynamoDB 本機版的版本，請輸入下列命令：

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

如需使用 DynamoDB local 作為建置在 AWS Serverless Application Model (AWS SAM) 上之 REST 應用程式的一部分的範例，請參閱用於管理訂單的 SAM DynamoDB 應用程式。此範例應用程式示範如何使用 DynamoDB 本機版進行測試。

如果您想要執行同時使用 DynamoDB 本機容器的多容器應用程式，請使用 Docker Compose 定義並執行應用程式中的所有服務，包括 DynamoDB 本機版。

使用 Docker Compose 安裝並執行 DynamoDB 本機版：

1. 下載並安裝 Docker Desktop。
2. 將以下程式碼複製到一個檔案中並儲存為 `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
```

如果您想讓應用程式和 DynamoDB 本機版位於不同的容器中，請使用以下 yaml 檔案。

```
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
```

這個 docker-compose.yml 指令碼會建立一個 app-node 容器和一個 dynamodb-local 容器。指令碼會在 app-node 容器中執行命令，該命令使用 dynamodb-local 連線至 AWS CLI 容器，並說明帳戶和資料表限制。

若要搭配您自己的應用程式影像使用，請將下列範例中的 image 數值取代為您應用程式的數值：

```
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**
>
> YAML 指令碼需要您指定 AWS 存取金鑰和 AWS 私密金鑰，但它們不需要是您存取 DynamoDB 本機的有效 AWS 金鑰。

3. 執行下列命令列命令：

```
docker-compose up
```

將 DynamoDB 本機版執行為 Apache Maven 相依性

> **Note**
>
> 如果您要將使用適用於 Java v1.x 之 SDK 的應用程式移轉至適用於 Java 2.x 的 SDK，請遵循AWS 適用於 Java 2.x 的 SDK 的步驟。

請按照以下步驟操作，在您的應用程式中將 Amazon DynamoDB 用為依存項目。

部署 DynamoDB 為 Apache Maven 儲存庫

1. 下載並安裝 Apache Maven。如需詳細資訊，請參閱下載 Apache Maven 和安裝 Apache Maven。
2. 將 DynamoDB Maven 儲存庫新增至您應用程式的專案物件模型 (POM) 檔案。

```
<!--Dependency:-->
<dependencies>
   <dependency>
      <groupId>software.amazon.dynamodb</groupId>
      <artifactId>DynamoDBLocal</artifactId>
      <version>3.3.0</version>
   </dependency>
</dependencies>
```

與 Spring Boot 3 和/或 Spring Framework 6 搭配使用的範例範本：

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

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

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

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

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

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

> **Note**
>
> 您也可以使用 Maven 中央存儲庫 URL。

在 中執行 DynamoDB 本機 AWS CloudShell

AWS CloudShell 是以瀏覽器為基礎的預先驗證 Shell，您可以直接從 啟動 AWS 管理主控台。您可以從幾種不同的方式導覽至 AWS CloudShell AWS 管理主控台 。如需詳細資訊，請參閱 入門 AWS CloudShell。

請依照下列步驟，在 AWS CloudShell 的任何位置執行 DynamoDB 本機 AWS 管理主控台。

在 的 AWS CloudShell 中執行 DynamoDB 本機 AWS 管理主控台

1. AWS CloudShell 從主控台界面啟動，選擇可用的 AWS 區域，並切換到您偏好的 shell，例如 Bash、PowerShell 或 Z shell。
2. 若要選擇 AWS 區域，請前往選取區域選單，然後選取支援的 AWS 區域 。(可用區域會反白顯示。)
3. 從 中 AWS 管理主控台，選擇下列其中一個選項 AWS CloudShell 來啟動 ：
    a. 在導覽列上，選擇 AWS CloudShell 圖示。
    b. 在搜尋方塊中，輸入 CloudShell 一詞，然後選擇 CloudShell。
    c. 在最近造訪的小工具中，選擇 CloudShell。
    d. 從主控台工具列中，選擇 CloudShell。
4. 若要在 中執行 DynamoDB 本機 AWS CloudShell ，您可以使用`dynamodb-local`別名。您可以指定變更 DynamoDB 本機版設定的其他命令列選項。關於可用的選項，請參閱 the section called “使用須知”。

> **Note**
>
> 若要在背景執行 DynamoDB 本機， AWS CloudShell 請在 中使用 執行 DynamoDB 本機`dynamodb-local &`。

5. 若要 AWS CloudShell 使用 存取在本機執行的 DynamoDB AWS CLI，請使用 `--endpoint-url` 參數。例如，使用下列命令來列出 DynamoDB 資料表：

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

如需示範設定和使用 DynamoDB 本機版的多種方法的範例專案，包括下載 JAR 檔案、將其作為 Docker 映像檔執行，以及將其用作 Maven 相依性，請參閱 DynamoDB 本機版範例 Java 專案。

DynamoDB 本機版使用須知

除端點外，使用可下載版 Amazon DynamoDB 執行的應用程式，也應該能使用 DynamoDB Web 服務。但是在本機使用 DynamoDB 時，您應該清楚下列事項：

- 如果您使用 `-sharedDb` 選項，DynamoDB 會建立單一資料庫檔案，名為 shared-local-instance.db。連線至 DynamoDB 的每個程式都能存取此檔案。如果您刪除此檔案，就會遺失所有存放在其中的資料。
- 如果您省略 `-sharedDb`，則資料庫檔案會命名為 myaccesskeyid_region.db，其 AWS 存取金鑰 ID 和 AWS 區域會顯示在您的應用程式組態中。如果您刪除此檔案，就會遺失所有存放在其中的資料。
- 如果您使用 `-inMemory` 選項，DynamoDB 完全不會寫入任何資料庫檔案。相反地，所有資料都會寫入記憶體，並且在您終止 DynamoDB 時不儲存資料。
- 如果您使用 `-inMemory` 選項，則也必須使用 `-sharedDb` 選項。
- 如果您使用 `-optimizeDbBeforeStartup` 選項，就必須也指定 `-dbPath` 參數，以便 DynamoDB 能找到它的資料庫檔案。
- DynamoDB AWS SDKs 需要您的應用程式組態指定存取金鑰值和 AWS 區域值。除非您使用的是 `-sharedDb` 或 `-inMemory` 選項，否則 DynamoDB 會使用這些數值來命名本機資料庫檔案。這些值不必是可在本機執行的有效 AWS 值。不過，您可能會發現使用有效的值很方便，因為稍後只要變更您使用的端點，就可以在雲端執行程式碼。
- DynamoDB 本機版一律對 `billingModeSummary.` 傳回 Null
- DynamoDB 本機版 `AWS_ACCESS_KEY_ID` 只能包含字母 (A–Z、a–z) 和數字 (0–9)。
- DynamoDB 本機版不支援時間點復原 (PITR)。

主題

- 命令列選項
- 設定區域端點
- 可下載版 DynamoDB 和 DynamoDB Web 服務之間的差異

命令列選項

您可以使用下列命令列選項搭配可下載版 DynamoDB 使用：

- `-corsvalue`：啟用 JavaScript 跨來源資源共用 (CORS) 支援。您必須提供逗號分隔的特定網域「允許」清單。`-cors` 的預設設定是星號 (*)，意為允許公開存取。
- `-dbPath value`：DynamoDB 寫入其資料庫檔案的目錄。如果您不指定此選項，檔案會寫入目前的目錄。您不能同時指定 `-dbPath` 和 `-inMemory`。
- `-delayTransientStatuses`：導致 DynamoDB 對部分操作造成延遲。DynamoDB (可下載版本) 幾乎可以立即執行部分任務，例如在資料表和索引上建立/更新/刪除操作。但是，DynamoDB 服務需要較多時間處理這些任務。設定此參數有助於在您電腦上執行的 DynamoDB 將 DynamoDB Web 服務的行為模擬得更逼真。(此參數目前只會造成 CREATING 或 DELETING 狀態的全域次要索引延遲)。
- `-help`：列印用量摘要及選項。
- `-inMemory`：DynamoDB 在記憶體中執行，而非使用資料庫檔案。當您停止 DynamoDB 時，不會儲存任何資料。您不能同時指定 `-dbPath` 和 `-inMemory`。
- `-optimizeDbBeforeStartup`：先最佳化基礎資料庫資料表，再啟動您電腦上的 DynamoDB。當您使用此參數時，也必須指定 `-dbPath`。
- `-port value`：DynamoDB 用來與您的應用程式進行通訊的連接埠號碼。如果您不指定此選項，預設連接埠為 `8000`。

> **Note**
>
> DynamoDB 預設使用連接埠 8000。如果無法使用連接埠 8000，此命令就會擲出例外狀況。您可以使用 `-port` 選項來指定不同的連接埠號碼。如需完整的 DynamoDB 執行時期選項清單 (包括 `-port`)，請輸入此命令：
>
> ```
> java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar
> -help
> ```

- `-sharedDb`：若您指定 `-sharedDb`，則 DynamoDB 會使用單一資料庫檔案，而非為每個憑證和區域使用不同的檔案。
- `-disableTelemetry`：指定時，DynamoDB 本機版將不會傳送任何遙測資料。
- `-version` – 列印 DynamoDB 本機版的版本。

設定區域端點

根據預設， AWS SDKs 和工具會使用 Amazon DynamoDB Web 服務的端點。若要使用開發套件和工具搭配可下載版 DynamoDB，您必須指定區域端點：

`http://localhost:8000`

AWS Command Line Interface

您可以使用 AWS Command Line Interface (AWS CLI) 與可下載的 DynamoDB 互動。

若要存取在本機執行的 DynamoDB，請使用 `--endpoint-url` 參數。以下是使用 AWS CLI 列出您電腦上 DynamoDB 中資料表的範例。

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

> **Note**
>
> AWS CLI 無法使用可下載的 DynamoDB 版本做為預設端點。因此，您必須`--endpoint-url`使用每個 AWS CLI 命令來指定 。

AWS SDKs

您指定端點的方式，視您使用的程式設計語言和 AWS 軟體開發套件而定。下列各節說明如何執行此作業：

- Java：設定 AWS 區域和端點 (DynamoDB 本機支援適用於 Java V1 和 V2 的 AWS SDK)
- CodeSamples.Java.RegionAndEndpoint .NET：設定 AWS 區域和端點

可下載版 DynamoDB 和 DynamoDB Web 服務之間的差異

可下載版 DynamoDB 僅用於開發和測試。相較之下，DynamoDB Web 服務則是受管服務，具可擴展性、可用性及耐用性的特色，因此適用於生產。

可下載版 DynamoDB 和 Web 服務之間的差異如下：

- AWS 區域 用戶端層級 AWS 帳戶 不支援 和 distinct。
- 可下載版 DynamoDB 會忽略佈建輸送量設定，即使 `CreateTable` 操作需要它們。您可為 `CreateTable` 指定任何您想要的佈建讀取和寫入輸送量數字，即使這些數字不予使用。您可以每天呼叫 `UpdateTable`，次數不限。但是會忽略任何佈建輸送量值的變更。
- `Scan` 操作以循序方式執行。不支援平行掃描。忽略 `Segment` 操作的 `TotalSegments` 與 `Scan` 參數。
- 資料表資料的讀取和寫入操作速度只受限於您電腦的速度。`CreateTable`、`UpdateTable` 和 `DeleteTable` 操作會立即發生，且資料表狀態一律為 ACTIVE。只變更資料表或全域次要索引之佈建輸送量設定的 `UpdateTable` 操作會立即發生。如果 `UpdateTable` 操作建立或刪除任何全域次要索引，則這些索引會先轉換到一般狀態 (例如分別為 CREATING 和 DELETING)，再變成 ACTIVE 狀態。資料表在這段時間內仍維持 ACTIVE。
- 讀取操做為最終一致。但是，由於您電腦上執行的 DynamoDB 本機版速度的緣故，大部分的讀取會是高度一致。
- 不會追蹤項目集合指標和項目集合大小。操作回應中會傳回 Null，而不是項目集合指標。
- 在 DynamoDB 中，每個結果集傳回的資料有 1 MB 的限制。DynamoDB Web 服務和可下載版本都會執行此限制。但在查詢索引時，DynamoDB 服務只會計算投影索引鍵和屬性的大小。反之，可下載版 DynamoDB 會計算整個項目的大小。
- 如果您使用的是 DynamoDB Streams，建立碎片的速率可能不同。在 DynamoDB Web 服務中，碎片建立行為有部分會受到資料表分割區活動的影響。當您在本機執行 DynamoDB 時，不會分割資料表。無論哪一種情況，碎片都只是暫時存在，所以您的應用程式不應該依賴碎片行為。
- 可下載 DynamoDB 版本不會對交易 API 丟出 `TransactionConflictExceptions`。建議您使用 Java 模擬架構，在 DynamoDB 處理常式中模擬 `TransactionConflictExceptions`，以測試您的應用程式如何回應相衝突的交易。
- 在 DynamoDB Web 服務中，無論是透過主控台或 存取 AWS CLI，資料表名稱都會區分大小寫。名為 `Authors` 和名為 `authors` 的資料表可同時存在，視為不同的資料表。在可下載的版本中，資料表名稱不區分大小寫，因為嘗試建立這樣的兩份資料表會造成錯誤。
- DynamoDB 的可下載版本不支援標記。
- 可下載的 DynamoDB 版本會忽略 ExecuteStatement 中的 Limit 參數。

DynamoDB 本機版的版本歷史記錄

下表說明 DynamoDB 本機版每個版本的重要變更。

版本	變更	描述	Date
3.3.0	新增全域次要索引的多屬性索引鍵支援	• 新增全域次要索引的多屬性索引鍵支援 • 將 SDK Java 版本更新為從 2.41.0 到 2.41.7 的最新公有版本	2026 年 1 月 19 日
3.2.0	已修正多個 Kotlin 版本的相容性問題	• 在 DescribeStream API 上新增對 ShardFilter 參數的支援 • 將 SDK Java 版本更新至 2.33.0 至 2.41.0 的最新公有版本 • 修正 UpdateTable 上的 DeletionProtection 錯誤	2026 年 1 月 9 日
3.1.0	提升 PartiQL 查詢效能，包括 Joda-time 相依性	• 將 SDK Java 版本更新為最新公有版本 (從 2.25.50 到 2.33.0) • 在 Pom.xml 檔案中包含 Joda-time 相依性 • 提升 PartiQL 查詢效能 • 升級相依性以修正多個 CVE 漏洞問題	2025 年 9 月 14 日
3.0.0	從 AWS SDK Java V1 遷移至 V2	• 從 AWS SDK Java V1 遷移至 V2	2025 年 7 月 17 日

版本	變更	描述	Date
		• 將套件結構從 `com.amazonaws.services.dynamodbv2` 更新為 `software.amazon.dynamodb.services` • 已移除 AWS SDK Java V1 相依性	
2.6.0	在 DynamoDB API 中支援資料表 ARN 做為資料表名稱 效能修正和安全性更新	• 新增支援在數個 DynamoDB API 使用資料表 ARN 作為資料表名稱 • 修正高效能機器上的 `CreateStreamTable` 錯誤，例如 Mac M3 • 升級相依性以修正漏洞問題 (CVE-2022-49043、CVE-2024-56732、CVE-2020-29582、CVE-2025-21502、CVE-2024-50602、CVE-2025-24970、CVE-2025-25193)	2025 年 3 月 13 日

版本	變更	描述	Date
2.5.4	升級到 Jetty 相依性	• 從 Jetty 12.0.8 升級到 Jetty 12.0.14 (解決 CVE-2024-6763、CVE-2024-8184 、CVE-2024 -47535) 緩解 (CVE-2024-21634)	2024 年 12 月 12 日
2.5.3	將 Log4j Core 中的 Jackson 相依性升級至 2.17.x (解決 CVE-2022-1471)	• 將 Log4j Core 中的 Jackson 相依性升級至 2.17.x (解決 CVE-2022-1471)，以解決 SnakeYAML 函式庫中的關鍵安全漏洞 (這是過渡相依性)	2024 年 11 月 6 日
2.5.2	修正更新資料表工作流程的錯誤	• 修正更新資料表嘗試將資料表從「Billing Mode Ondemand」更新為「Provisioned With GSI」時的工作流程錯誤	2024 年 6 月 20 日
2.5.1	針對 OndemandT hroughPut 功能中發生的錯誤提供修補程式	• 修正幾個與 OndemandT hroughPut 相關的錯誤	2024 年 6 月 5 日

版本	變更	描述	Date
2.5.0	支援為隨需資料表、ReturnValuesOnConditionCheckFailure 、BatchExecuteStatement 和 ExecuteTransaction Request 設定最大輸送量	• 將遙測新增至嵌入模式 • 修正 ConditionalCheckException 的 SDKv2 轉譯	2024 年 5 月 28 日
2.4.0	支援 ReturnValuesOnConditionCheckFailure - 嵌入模式	• 針對在多個串流操作的 TrimmedDataAccessException 進行嵌入模式修正 • 修正嵌入模式中 SDKv2 的例外狀況轉譯	2024 年 4 月 17 日
2.3.0	Jetty 和 JDK 升級	• 升級到 Jetty 12.0.2 • 升級到 JDK 17 • 將 ANTLR4 升級到 4.10.1	2024 年 3 月 14 日
2.2.0	新增支援資料表刪除保護和 ReturnValuesOnConditionCheckFailure 參數	• 新增支援資料表刪除保護 • 新增支援 ReturnValuesOnConditionCheckFailure • 新增支援 -version 旗標	2023 年 12 月 14 日

版本	變更	描述	Date
2.1.0	SQLLite Native Libraries for Maven 專案的支援和新增遙測功能	• 在 DynamoDB 本機版上新增遙測功能 • 動態複製 SQLLite Native Libraries for Maven 專案 • 已移除 Maven 依存項目中的 io.github.ganadist.sqlite4java 程式庫 • 將 GoogleGuava 升級至 32.1.1-jre	2023 年 10 月 23 日
2.0.0	從 javax 移轉到 jakarta 命名空間和 JDK11 支援	• 從 javax 移轉到 jakarta 命名空間和 JDK11 支援 • 修復在伺服器啟動時處理無效存取和私密金鑰的問題 • 透過更新相依關係來修復 Maven 找到的漏洞	2023 年 7 月 5 日
1.25.1	將 Log4j Core 中的 Jackson 相依性升級至 2.17.x (解決 CVE-2022-1471)	將 Log4j Core 中的 Jackson 相依性升級至 2.17.x (解決 CVE-2022-1471)，以解決 SnakeYAML 函式庫中的關鍵安全漏洞 (這是過渡相依性)	2024 年 11 月 6 日

版本	變更	描述	Date
1.25.0	新增支援資料表刪除保護和 `ReturnValuesOnConditionCheckFailure` 參數	• 新增支援資料表刪除保護 • 新增支援 ReturnValuesOnConditionCheckFailure • 新增支援 -version 旗標	2023 年 12 月 18 日
1.24.0	SQLLite Native Libraries for Maven 專案的支援和新增遙測功能	• 在 DynamoDB 本機版上新增遙測功能 • 動態複製 SQLLite Native Libraries for Maven 專案 • 已移除 Maven 依存項目中的 io.github.ganadist.sqlite4java 程式庫 • 將 GoogleGuava 升級至 32.1.1-jre	2023 年 10 月 23 日
1.23.0	在伺服器啟動時處理無效存取和私密金鑰	• 修復在伺服器啟動時處理無效存取和私密金鑰的問題 • 透過更新相依關係來修復 Maven 找到的漏洞	2023 年 6 月 28 日
1.22.0	對於 PartiQL 限制操作的支援	• 最佳化 PartiQL 的 IN 子句 • 限制操作支援 • Maven 專案的 M1 支援	2023 年 6 月 8 日

版本	變更	描述	Date
1.21.0	支援每筆交易 100 個動作	• 將每筆交易的動作從 25 提升到 100 • 升級 Docker 映像檔 Open JDK 到 11 • 修復 BatchExecuteStatement 中有重複項目時引發異常的奇偶校驗	2023 年 1 月 26 日
1.20.0	新增對 M1 Mac 的支援	• 新增對 M1 Mac 的支援 • 將 Jetty 相依性升級至 9.4.48.v20220622	2022 年 9 月 12 日
1.19.0	已升級 PartiQL 剖析器	已升級 PartiQL 剖析器和其他相關程式庫	2022 年 7 月 27 日
1.18.0	已升級 log4j-core 和 jackson-core	已升級 log4j-core 至 2.17.1 版和 jackson-core 2.10.x 到 2.12.0 版	2022 年 1 月 10 日
1.17.2	已升級 log4j-core	已升級 log4j-core 相依性到 2.16 版	2021 年 1 月 16 日
1.17.1	已升級 log4j-core	已更新 log4j-core 相依性以修補零時差漏洞，以防止遠端程式碼之執行 - Log4Shel	2021 年 1 月 10 日

版本	變更	描述	Date
1.17.0	Javascript Web Shell 已作廢	• 將 AWS SDK 相依性更新為適用於 Java 的 AWS SDK 1.12.x • Javascript Web Shell 已作廢	2021 年 1 月 8 日

DynamoDB 本機版中的遙測功能

在 AWS，我們根據我們從與客戶互動中學到的內容來開發和啟動服務，並使用客戶意見回饋來迭代產品。遙測功能這項額外的資訊能幫助我們進一步了解客戶的需求、診斷問題，並且提供各項功能來改善客戶體驗。

DynamoDB 本機版會收集遙測資訊，例如一般使用指標、系統和環境資訊以及錯誤。如需所收集遙測類型的詳細資訊，請參閱 收集的資訊類型。

DynamoDB 本機版不會收集個人資訊，例如使用者姓名或電子郵件地址。同時也不會擷取敏感的專案層級資訊。

身為客戶，您可以全權掌控是否開啟遙測功能，並且可以隨時變更您的設定。如果遙測功能保持開啟，DynamoDB 本機版會在背景傳送遙測資料，不需任何額外的客戶互動。

使用命令列選項關閉遙測功能

您可以在使用選項 `-disableTelemetry` 啟動 DynamoDB 本機版時，使用命令列選項關閉遙測功能。如需詳細資訊，請參閱命令列選項。

關閉單一工作階段的遙測功能

在 macOS 和 Linux 作業系統中，您可以關閉單一工作階段的遙測功能。若要關閉目前工作階段的遙測功能，請執行下列命令，將環境變數 DDB_LOCAL_TELEMETRY 設定為 `false`。針對每個新的終端或工作階段重複此命令。

```
export DDB_LOCAL_TELEMETRY=0
```

在所有工作階段中關閉您的設定檔的遙測功能

當您在作業系統上執行 DynamoDB 本機版時，執行下列命令即可關閉所有工作階段的遙測功能。

關閉 Linux 中的遙測功能

1. 執行：

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

2. 執行：

```
source ~/.profile
```

關閉 macOS 中的遙測功能

1. 執行：

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

2. 執行：

```
source ~/.profile
```

關閉 Windows 中的遙測功能

1. 執行：

```
setx DDB_LOCAL_TELEMETRY 0
```

2. 執行：

```
refreshenv
```

使用 Maven 專案嵌入的 DynamoDB 本機版關閉遙測功能

您可以使用 Maven 專案嵌入的 DynamoDB 本機版來關閉遙測功能。

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

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

收集的資訊類型

- 使用情形資訊：一般遙測，例如伺服器啟動/停止，以及呼叫的 API 或操作。
- 系統和環境資訊：Java 版本、作業系統 (Windows、Linux 或 macOS)、DynamoDB 本機版執行所在的環境 (例如，單機版 JAR、Docker 容器或作為 Maven 依存項目)，以及用量屬性的雜湊值。

進一步了解

DynamoDB local 收集的遙測資料遵循 AWS 資料隱私權政策。如需詳細資訊，請參閱下列內容：

- AWS 服務條款
- 資料隱私權常見問答集

步驟 1：在 DynamoDB 建立資料表

在此步驟中，您可以在 Amazon DynamoDB 中建立 `Music` 資料表。此資料表具有下列詳細資訊：

- 分割區索引鍵：`Artist`
- 排序索引鍵：`SongTitle`

如需資料表操作的詳細資訊，請參閱 在 DynamoDB 中使用資料表和資料。

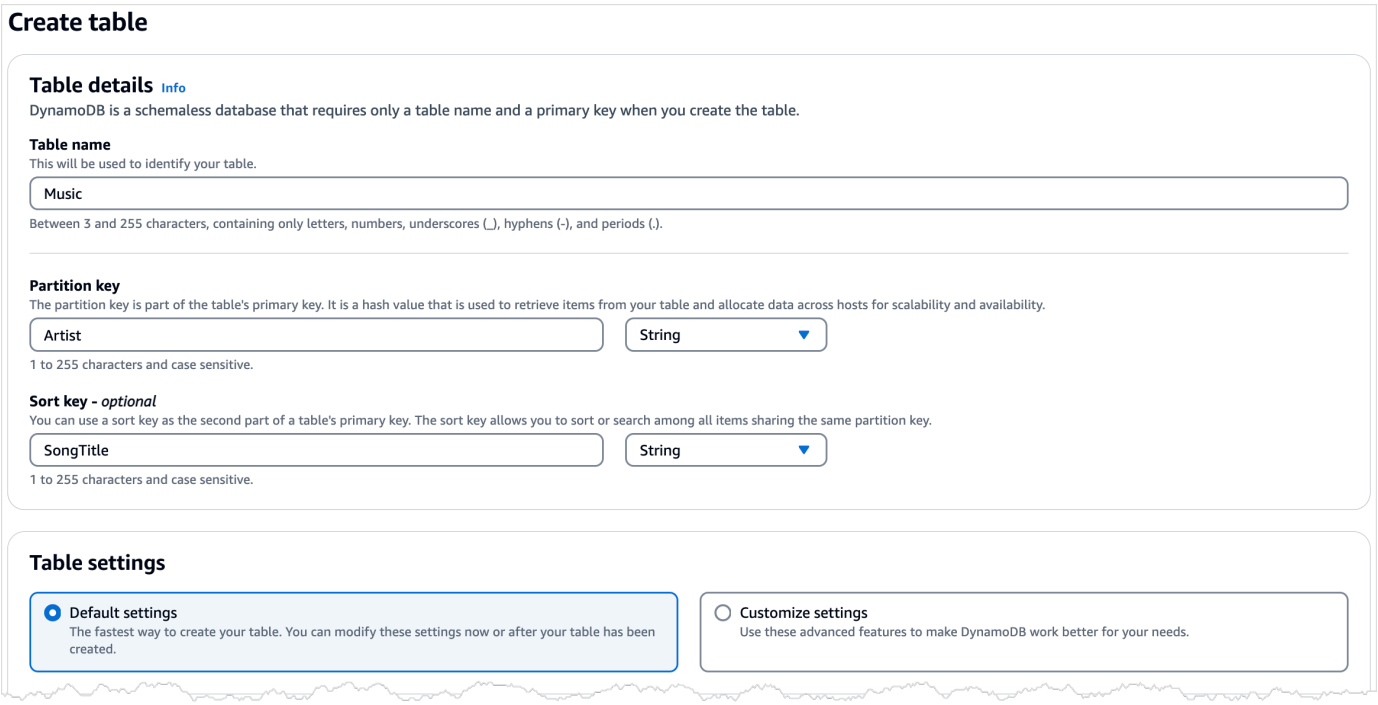
> **Note**
>
> 開始之前，請確定您已遵循 先決條件 中的步驟。

AWS 管理主控台

若要使用 DynamoDB 主控台建立新的 `Music` 資料表：

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。

2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 選擇 Create Table (建立資料表)。
4. 輸入資料表詳細資訊 (如下所示)：
   a. 對於 Table name (資料表名稱)，請輸入 **`Music`**。
   b. 對於 Partition key (分割區索引鍵)，請輸入 **`Artist`**。
   c. 對於排序索引鍵，請輸入 **`SongTitle`**。
5. 對於資料表設定，請保留預設設定的預設選擇。
6. 選擇建立資料表以建立資料表。



7. 一旦資料表處於 `ACTIVE` 狀態，建議您執行下列步驟在資料表啟用 DynamoDB 的時間點備份：
   a. 選擇資料表名稱以開啟資料表。
   b. 選擇備份。
   c. 在時間點復原 (PITR) 區段中選擇編輯。
   d. 在編輯時間點復原設定頁面選擇啟用時間點復原。
   e. 選擇儲存變更。

AWS CLI

下列 AWS CLI 範例會使用 建立新的`Music`資料表`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
```

使用 `create-table` 傳回以下範例結果。

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

請注意，TableStatus 欄位的值會設定為 CREATING。

若要確認 DynamoDB 已完成建立 Music 資料表，請使用 describe-table 命令。

Linux

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

Windows CMD

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

此命令會傳回下列結果。當 DynamoDB 已完成建立資料表時，TableStatus 欄位的值會設定為 ACTIVE。

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

一旦資料表進入 ACTIVE 狀態，根據最佳實務，應執行以下命令，以在資料表上啟用 DynamoDB 的時間點備份：

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
```

此命令會傳回下列結果。

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

> **Note**
>
> 啟用具備時間點復原的連續備份會有成本影響。如需有關定價的詳細資訊，請參閱 Amazon DynamoDB 定價。

AWS 開發套件

以下程式碼範例示範如何使用 AWS SDK 建立 DynamoDB 資料表。

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

```
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

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

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

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

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

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

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
{tableName}. {ex.Message}");
            throw;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateTable。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 CreateTable。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

等待資料表變成作用中的程式碼。

```
//! 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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 CreateTable。

CLI

AWS CLI

範例 1：建立具有標籤的資料表

下列 create-table 範例使用指定的屬性和索引鍵結構描述，建立名為 MusicCollection 的資料表。此資料表使用佈建輸送量，並使用預設 AWS 擁有的 CMK 進行靜態加密。該命令也會運用索引鍵 Owner 和值 blueTeam，將標籤套用至資料表。

```
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
```

輸出：

```
{
    "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"
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 2：在隨需模式下建立資料表

以下範例使用隨需模式 (而非佈建的輸送量模式)，建立名為 MusicCollection 的資料表。這適用於工作負載無法預測的資料表。

```
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
```

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 3：建立資料表，並使用客戶管理的 CMK 進行加密

下列範例建立名為 MusicCollection 的資料表，並使用客戶管理的 CMK 將其加密。

```
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
```

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 4：建立具有本機次要索引的資料表

以下範例使用指定的屬性和索引鍵結構描述，建立名為 MusicCollection 的資料表，其中包含名為 AlbumTitleIndex 的本機次要索引。

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 5：建立具有全域次要索引的資料表

以下範例會建立名為 GameScores 的資料表，以及名為 GameTitleIndex 的全域次要索引。基礎資料表具有分割區索引鍵 UserId 和排序索引鍵 GameTitle，可讓您有效率地找到個別使用者在特定遊戲中的最佳分數，而 GSI 具有分割區索引鍵 GameTitle 和排序索引鍵 TopScore，可讓您快速找到特定遊戲的整體最高分數。

```
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
                }
            }
        ]"
```

輸出：

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

```
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 6：建立具有多個全域次要索引的資料表

以下範例會建立名為 GameScores，且具有兩個全域次要索引的資料表。GSI 結構描述是透過檔案傳遞，而不是在命令列上傳遞。

```
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
```

gsi.json 的內容：

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 7：建立已啟用串流的資料表

下列範例建立名為 GameScores，並啟用 DynamoDB Streams 的資料表。每個項目的新舊映像都會寫入串流。

```
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
```

輸出：

```
{
    "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"
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 8：建立已啟用僅限索引鍵串流的資料表

下列範例建立名為 GameScores，並啟用 DynamoDB Streams 的資料表。只有修改過的項目索引鍵屬性才會寫入串流。

```
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
```

輸出：

```
{
    "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
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的 DynamoDB Streams 的變更資料擷取。

範例 9：建立具有 Standard Infrequent Access 類別的資料表

下列範例會建立名為 GameScores 的資料表，並指派 Standard-Infrequent Access (DynamoDB Standard-IA) 資料表類別。此資料表類別針對主要成本的儲存進行最佳化。

```
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
```

輸出：

```
{
    "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
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表類別。

範例 10：建立已啟用刪除保護的資料表

下列範例會建立名為 GameScores，並啟用刪除保護的資料表。

```
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
```

輸出：

```
{
    "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
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用刪除保護。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 CreateTable。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 CreateTable。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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 "";
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 CreateTable。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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;
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 CreateTable。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 CreateTable。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

```
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

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

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 CreateTable。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立資料表。

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 CreateTable。

PowerShell

Tools for PowerShell V4

範例 1：此範例會建立名為 Thread 的資料表，其主索引鍵包含 'ForumName' (索引鍵類型雜湊) 和 'Subject' (索引鍵類型範圍)。用來建構資料表的結構描述，可以使用 -Schema 參數，依照所示或指定方式輸送至每個 cmdlet。

```
$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
```

輸出：

```
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  : {}
```

範例 2：此範例會建立名為 Thread 的資料表，其主索引鍵包含 'ForumName' (索引鍵類型雜湊) 和 'Subject' (索引鍵類型範圍)。也會定義本機次要索引。本機次要索引的索引鍵會從資料表 (ForumName) 上的主要雜湊索引鍵自動設定。用來建構資料表的結構描述，可以使用 -Schema 參數，依照所示或指定方式輸送至每個 cmdlet。

```
$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
```

輸出：

```
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}
```

範例 3：此範例示範如何使用單一管道建立名為 Thread 的資料表，該資料表的主索引鍵包含 'ForumName' (索引鍵類型雜湊) 和 'Subject' (索引鍵類型範圍) 以及本機次要索引。如果管道或 -Schema 參數未提供新的 TableSchema 物件，Add-DDBKeySchema 和 Add-DDBIndexSchema 會為您建立新的 TableSchema 物件。

```
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
```

輸出：

```
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}
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 CreateTable。

Tools for PowerShell V5

範例 1：此範例會建立名為 Thread 的資料表，其主索引鍵包含 'ForumName' (索引鍵類型雜湊) 和 'Subject' (索引鍵類型範圍)。用來建構資料表的結構描述，可以使用 -Schema 參數，依照所示或指定方式輸送至每個 cmdlet。

```
$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
```

輸出：

```
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  : {}
```

範例 2：此範例會建立名為 Thread 的資料表，其主索引鍵包含 'ForumName' (索引鍵類型雜湊) 和 'Subject' (索引鍵類型範圍)。也會定義本機次要索引。本機次要索引的索引鍵會從資料表 (ForumName) 上的主要雜湊索引鍵自動設定。用來建構資料表的結構描述，可以使用 -Schema 參數，依照所示或指定方式輸送至每個 cmdlet。

```
$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
```

輸出：

```
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}
```

範例 3：此範例示範如何使用單一管道建立名為 Thread 的資料表，該資料表的主索引鍵包含 'ForumName' (索引鍵類型雜湊) 和 'Subject' (索引鍵類型範圍) 以及本機次要索引。如果管道或 -Schema 參數未提供新的 TableSchema 物件，Add-DDBKeySchema 和 Add-DDBIndexSchema 會為您建立新的 TableSchema 物件。

```
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
```

輸出：

```
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}
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 CreateTable。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立用於存放電影資料的資料表。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 CreateTable。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
# 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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 CreateTable。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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))
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 CreateTable。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 CreateTable。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 CreateTable。

如需更多的 DynamoDB 範例，請參閱 使用 AWS SDKs DynamoDB 程式碼範例。

在建立新的資料表之後，請繼續 步驟 2：將資料寫入 DynamoDB 資料表。

步驟 2：將資料寫入 DynamoDB 資料表

在此步驟中，您會將數個項目插入至在 步驟 1：在 DynamoDB 建立資料表 中建立的 `Music` 資料表。

如需寫入操作的詳細資訊，請參閱 寫入項目。

AWS 管理主控台

請遵循下列步驟，使用 DynamoDB 主控台將資料寫入至 `Music` 資料表。

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 在資料表頁面選擇 Music 資料表。
4. 選擇探索資料表項目。
5. 在已傳回項目區段中選擇建立項目。
6. 在建立項目頁面上執行下列動作，將項目新增至資料表：
    a. 選擇 Add new attribute (新增屬性)，接著選擇 Number (數字)。
    b. 對於屬性名稱，請輸入 **`Awards`**。
    c. 重複此程序，建立類型 String (字串) 的 **`AlbumTitle`**。
    d. 為您的項目輸入下列值：
        i. 針對 Artist (藝人)，輸入 **`No One You Know`**。
        ii. 針對 SongTitle (歌曲名稱)，輸入 **`Call Me Today`**。
        iii. 針對 AlbumTitle (專輯名稱)，輸入 **`Somewhat Famous`**。
        iv. 針對 Awards (獎項)，輸入 **`1`**。
7. 選擇 Create item (建立項目)。
8. 重複此程序並利用下列值建立另一個項目：
    a. 針對 Artist (藝人)，輸入 **`Acme Band`**。
    b. 針對 SongTitle (歌曲名稱)，輸入 **`Happy Day`**。
    c. 針對 AlbumTitle (專輯名稱)，輸入 **`Songs About Life`**。
    d. 針對 Awards (獎項)，輸入 **`10`**。
9. 再執行一次，建立另一個具有與上一步驟相同 Artist (藝術家) 但其他屬性值不同的項目：
    a. 針對 Artist (藝人)，輸入 **`Acme Band`**。
    b. 針對 SongTitle (歌曲名稱)，輸入 **`PartiQL Rocks`**。
    c. 針對 AlbumTitle (專輯名稱)，輸入 **`Another Album Title`**。
    d. 針對 Awards (獎項)，輸入 **`8`**。

AWS CLI

下列 AWS CLI 範例會在 Music 資料表中建立新的數個項目。您可以透過 DynamoDB API 或 PartiQL (一種適用於 DynamoDB 的 SQL 相容查詢語言) 進行此操作。

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'}"
```

如需有關使用 PartiQL 寫入資料的詳細資訊，請參閱 PartiQL Insert 陳述式。

如需有關 DynamoDB 中支援之資料類型的詳細資訊，請參閱資料類型。

如需有關如何以 JSON 表示 DynamoDB 資料類型的詳細資訊，請參閱屬性值。

AWS 開發套件

下列程式碼範例示範如何使用 AWS SDK 將項目寫入 DynamoDB 資料表。

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/// <summary>
/// Adds a new item to the table.
/// </summary>
/// <param name="newMovie">A Movie object containing informtation for
/// the movie to add to the table.</param>
```

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

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

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

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 PutItem。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

}
```

此範例中使用的公用程式函數。

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

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 PutItem。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

等待資料表變成作用中的程式碼。

```
//! 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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 PutItem。

CLI

AWS CLI

範例 1：將項目新增至資料表

下列put-item 範例會將新項目加入至 MusicCollection 資料表。

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

item.json 的內容：

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

輸出：

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

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的寫入項目。

範例 2：有條件地覆寫資料表中的項目

下列 put-item 範例只有在現有項目具有值為 Greatest Hits 的 AlbumTitle 屬性時，才會覆寫 MusicCollection 資料表中的現有項目。命令會傳回項目的上一個值。

```
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
```

item.json 的內容：

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

names.json 的內容：

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

values.json 的內容：

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

輸出：

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

如果索引鍵已存在，您應該會看到下列輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的寫入項目。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 PutItem。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 PutItem。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 DynamoDbClient 將項目放入資料表。

```
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);
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 PutItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 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;
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 PutItem。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

將項目放入資料表。

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

使用 DynamoDB 文件用戶端將項目放入資料表。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 PutItem。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
```

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

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

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

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 PutItem。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        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);
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 PutItem。

PowerShell

Tools for PowerShell V4

範例 1：建立新的項目，或將現有項目取代為新項目。

```
$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
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 PutItem。

Tools for PowerShell V5

範例 1：建立新的項目，或將現有項目取代為新項目。

```
$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
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 PutItem。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 PutItem。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 PutItem。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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,
    })
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 PutItem。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 PutItem。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
    }
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 PutItem。

如需更多的 DynamoDB 範例，請參閱 使用 AWS SDKs DynamoDB 程式碼範例。

將資料寫入至您的資料表之後，請繼續 步驟 3：從 DynamoDB 資料表讀取資料。

步驟 3：從 DynamoDB 資料表讀取資料

在此步驟中，您將讀回在 步驟 2：將資料寫入 DynamoDB 資料表 中建立的其中一個項目。您可以使用 DynamoDB 主控台或 AWS CLI ，透過指定 `Artist`和 從`Music`資料表讀取項目`SongTitle`。

如需 DynamoDB 中讀取操作的詳細資訊，請參閱 讀取項目。

AWS 管理主控台

請遵循下列步驟，使用 DynamoDB 主控台從 `Music` 資料表讀取資料。

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 在資料表頁面選擇 Music 資料表。
4. 選擇探索資料表項目。
5. 在已傳回項目區段檢視資料表中存放的項目清單，依 `Artist` 和 `SongTitle` 排序。清單中的第一個項目包含名為 Acme Band 的 Artist，以及 SongTitle PartiQL Rocks。

AWS CLI

下列 AWS CLI 範例會從 讀取項目`Music`。您可以透過 DynamoDB API 或 PartiQL (一種適用於 DynamoDB 的 SQL 相容查詢語言) 進行此操作。

DynamoDB API

> **Note**
>
> DynamoDB 的預設行為是最終一致讀取。`consistent-read` 參數在下面用來展示高度一致性讀取。

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
\"}}"
```

使用 `get-item` 傳回以下範例結果。

```
{
    "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'"
```

使用 PartiQL `Select` 陳述式傳回以下範例結果。

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

如需有關使用 PartiQL 讀取資料的詳細資訊，請參閱 PartiQL Select 陳述式。

AWS 開發套件

下列程式碼範例示範如何使用 AWS SDK 從 DynamoDB 資料表讀取項目。

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
```

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

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

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

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 GetItem。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 GetItem。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 GetItem。

CLI

AWS CLI

範例 1：讀取資料表中的項目

下列 `get-item` 範例從 `MusicCollection` 資料表擷取項目。資料表具有 hash-and-range 主索引鍵 (`Artist` 和 `SongTitle`)，因此您必須指定這兩個屬性。該命令也會請求操作所取用讀取容量的相關資訊。

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

`key.json` 的內容：

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的讀取項目。

範例 2：使用一致讀取來讀取項目

以下範例使用高度一致性讀取，從 MusicCollection 資料表擷取項目。

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

key.json 的內容：

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

輸出：

```
{
    "Item": {
        "AlbumTitle": {
```

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的讀取項目。

範例 3：擷取特定的項目屬性

以下範例使用投影表達式，僅擷取所需項目的三個屬性。

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

names.json 的內容：

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

輸出：

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
        "Title": {
```

```
            "S": "Book 102 Title"
        },
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的讀取項目。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 GetItem。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 GetItem。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 DynamoDbClient 從資料表取得項目。

```
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);
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 GetItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 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;
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 GetItem。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

從資料表取得項目。

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

使用 DynamoDB 文件用戶端從資料表取得項目。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 GetItem。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

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

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 GetItem。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 GetItem。

PowerShell

Tools for PowerShell V4

範例 1：使用分割區索引鍵 SongTitle 和排序索引鍵 Artist 傳回 DynamoDB 項目。

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

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

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 GetItem。

Tools for PowerShell V5

範例 1：使用分割區索引鍵 SongTitle 和排序索引鍵 Artist 傳回 DynamoDB 項目。

```
$key = @{
```

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

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

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 GetItem。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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"]
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 GetItem。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 GetItem。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 GetItem。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 GetItem。

如需更多的 DynamoDB 範例，請參閱 使用 AWS SDKs DynamoDB 程式碼範例。

若要更新資料表中的資料，請繼續步驟 4：更新 DynamoDB 資料表中的資料。

步驟 4：更新 DynamoDB 資料表中的資料

在此步驟中，您會更新已在 步驟 2：將資料寫入 DynamoDB 資料表 建立的項目。您可以使用 DynamoDB 主控台或 AWS CLI ，透過指定 `Artist`、 和更新的 `SongTitle`來更新`Music`資料表中`AlbumTitle`項目的 `AlbumTitle`。

如需寫入操作的詳細資訊，請參閱 寫入項目。

AWS 管理主控台

您可以使用 DynamoDB 主控台，更新 `Music` 資料表中的資料。

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 從資料表清單中，選擇 Music (音樂) 資料表。
4. 選擇探索資料表項目。
5. 在已傳回項目中，針對具有 Acme Band Artist 和 Happy Day SongTitle 的項目列執行下列動作：
   a. 將游標放在名為 Songs About Life 的 AlbumTitle 上。
   b. 選擇編輯圖示。
   c. 在編輯字串快顯視窗中輸入 **`Songs of Twilight`**。
   d. 選擇儲存。

> **Tip**
>
> 或者若要更新項目，請在已傳回項目區段中執行下列動作：
>
> 1. 選擇 Artist 名為 Acme Band 且 SongTitle 名為 Happy Day 的項目列。
> 2. 從動作下拉式清單中，選擇編輯項目。
> 3. 針對 AlbumTitle 輸入 **`Songs of Twilight`**。
> 4. 選擇儲存與關閉。

AWS CLI

下列 AWS CLI 範例會更新 Music 資料表中的項目。您可以透過 DynamoDB API 或 PartiQL (一種適用於 DynamoDB 的 SQL 相容查詢語言) 進行此操作。

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
```

使用 update-item 傳回以下範例結果，因為已指定 return-values ALL_NEW。

```
{
    "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
 *"
```

使用 Update 陳述式傳回以下範例結果，因為已指定 RETURNING ALL NEW *。

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

如需有關使用 PartiQL 更新資料的詳細資訊，請參閱 PartiQL Update 陳述式。

AWS 開發套件

下列程式碼範例示範如何使用 AWS SDK 更新 DynamoDB 資料表中的項目。

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
```

```
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

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

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 UpdateItem。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

}
```

此範例中使用的公用程式函數。

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

```
}
```

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 UpdateItem。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

等待資料表變成作用中的程式碼。

```
//! 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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 UpdateItem。

CLI

AWS CLI

範例 1：更新資料表中的項目

下列 update-item 範例會更新 MusicCollection 資料表中的項目。它會新增屬性 (Year) 並修改 AlbumTitle 屬性。更新後出現的項目的所有屬性，都會在回應中傳回。

```
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
```

key.json 的內容：

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

expression-attribute-names.json 的內容：

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

expression-attribute-values.json 的內容：

```
{
    ":y":{"N": "2015"},
```

```
    ":t":{"S": "Louder Than Ever"}
}
```

輸出：

```
{
    "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
        ]
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的寫入項目。

範例 2：有條件地更新項目

下列範例會更新 MusicCollection 資料表中的項目，但前提是現有項目尚未具備 Year 屬性。

```
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)"
```

key.json 的內容：

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

expression-attribute-names.json 的內容：

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

expression-attribute-values.json 的內容：

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

如果項目已具備 Year 屬性，DynamoDB 會傳回下列輸出。

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的寫入項目。

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 UpdateItem。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 UpdateItem。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 DynamoDbClient 更新資料表中的項目。

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

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 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;
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateItem。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
    name: String,
    updateVal: String,
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)
```

```
    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] =
        AttributeValueUpdate {
            value = AttributeValue.S(updateVal)
            action = AttributeAction.Put
        }

    val request =
        UpdateItemRequest {
            tableName = tableNameVal
            key = itemKey
            attributeUpdates = updatedValues
        }

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 UpdateItem。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 UpdateItem。

PowerShell

Tools for PowerShell V4

範例 1：使用分割區索引鍵 SongTitle 和排序索引鍵 Artist，將 DynamoDB 項目的類型屬性設定為 'Rap'。

```
$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
```

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 UpdateItem。

Tools for PowerShell V5

範例 1：使用分割區索引鍵 SongTitle 和排序索引鍵 Artist，將 DynamoDB 項目的類型屬性設定為 'Rap'。

```
$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
```

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 UpdateItem。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用更新表達式更新項目。

```
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"]
```

使用包含算術運算的更新表達式更新項目。

```
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"]
```

僅在項目符合特定條件時更新項目。

```
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"]
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 UpdateItem。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 UpdateItem。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 UpdateItem。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 UpdateItem。

如需更多的 DynamoDB 範例，請參閱 使用 AWS SDKs DynamoDB 程式碼範例。

若要查詢 Music 資料表中的資料，請繼續 步驟 5：查詢 DynamoDB 資料表中的資料。

步驟 5：查詢 DynamoDB 資料表中的資料

在此步驟中，您會指定 Music，來查詢您已在 the section called “步驟 2：寫入資料” 寫入至 Artist 資料表的資料。這將顯示與分割區索引鍵：Artist 相關聯的所有歌曲。

如需查詢操作的詳細資訊，請參閱 在 DynamoDB 中查詢資料表。

AWS 管理主控台

請遵循下列步驟，使用 DynamoDB 主控台來查詢 Music 資料表中的資料。

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 從資料表清單中，選擇 Music (音樂) 資料表。
4. 選擇探索資料表項目。
5. 在掃描或查詢項目中確認已選取查詢。
6. 針對 Partition key (分割區索引鍵)，輸入 **`Acme Band`**，然後選擇 Run (執行)。

AWS CLI

下列 AWS CLI 範例會查詢 `Music` 資料表中的項目。您可以透過 DynamoDB API 或 PartiQL (一種適用於 DynamoDB 的 SQL 相容查詢語言) 進行此操作。

DynamoDB API

您可以藉由使用 `query` 並提供分割區索引鍵，透過 DynamoDB API 查詢項目。

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\"}}"
```

使用 `query` 傳回所有與此特定 `Artist` 相關聯的歌曲。

```
{
    "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

您可以藉由使用 `Select` 陳述式並提供分割區索引鍵，透過 PartiQL 查詢項目。

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'"
```

以此方式使用 Select 陳述式會傳回所有與此特定 Artist 相關聯的歌曲。

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

如需有關使用 PartiQL 查詢資料的詳細資訊，請參閱 PartiQL Select 陳述式。

AWS 開發套件

以下程式碼範例示範如何使用 AWS SDK 查詢 DynamoDB 資料表。

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "title",
                "year",
            },
            ConsistentRead = true,
```

```
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 Query。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 Query。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! 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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 Query。

CLI

AWS CLI

範例 1：查詢資料表

下列 query 範例會查詢 MusicCollection 資料表中的項目。資料表具有 hash-and-range 主索引鍵 (Artist 和 SongTitle)，但此查詢只會指定雜湊索引鍵值。命令傳回名為 "No One You Know" 的歌手演唱曲目標題。

```
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
```

expression-attributes.json 的內容：

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用 DynamoDB 中查詢。

範例 2：使用高度一致性讀取查詢資料表，並以遞減順序遍歷索引

下列範例會執行與第一個範例相同的查詢，但會以相反順序傳回結果，並且使用高度一致性讀取。

```
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
```

expression-attributes.json 的內容：

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用 DynamoDB 中查詢。

範例 3：篩選出特定結果

下列範例會查詢 MusicCollection，但會在 AlbumTitle 屬性中排除具有特定值的結果。請注意，這不會影響 ScannedCount 或 ConsumedCapacity，因為僅在項目被讀取後才會套用篩選條件。

```
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
```

values.json 的內容：

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

names.json 的內容：

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

輸出：

```
{
    "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
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用 DynamoDB 中查詢。

範例 4：僅擷取項目計數

下列範例會擷取符合查詢的項目計數，但不會自行擷取任何項目。

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

expression-attributes.json 的內容：

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用 DynamoDB 中查詢。

範例 5：查詢索引

下列範例查詢的是本機次要索引 AlbumTitleIndex。查詢會從已投射至本機次要索引的基底資料表，傳回所有屬性。請注意，查詢本機次要索引或全域次要索引時，也需要使用 table-name 參數提供基底資料表的名稱。

```
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
```

expression-attributes.json 的內容：

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

輸出：

```
{
    "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
            }
```

```
        }
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用 DynamoDB 中查詢。

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 Query。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 Query。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 DynamoDbClient 查詢資源表。

```
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;
   }
}
```

使用 DynamoDbClient 和次要索引查詢資料表。

```
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);
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 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;
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request =
```

```
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 Query。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 Query。

PowerShell

Tools for PowerShell V4

範例 1：調用查詢，傳回具有指定之 SongTitle 和 Artist 的 DynamoDB 項目。

```
$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
```

輸出：

```
Name                           Value
----                           -----
```

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 Query。

Tools for PowerShell V5

範例 1：調用查詢，傳回具有指定之 SongTitle 和 Artist 的 DynamoDB 項目。

```
$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
```

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 Query。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用索引鍵條件表達式查詢項目。

```
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"]
```

查詢項目並投影以傳回資料子集。

```
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"]
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 Query。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

尋找在指定年份製作的電影。

```
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![])
    }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的Query。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 Query。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 Query。

如需更多的 DynamoDB 範例，請參閱 使用 AWS SDKs DynamoDB 程式碼範例。

若要為您的資料表建立全域次要索引，請繼續步驟 6：(選用) 刪除 DynamoDB 資料表以清除資源。

步驟 6：(選用) 刪除 DynamoDB 資料表以清除資源

如果您不再需要針對此教學課程建立的 Amazon DynamoDB 資料表，則可以將其刪除。此步驟協助確保您不會為了未使用的資源而付費。您可以使用 DynamoDB 主控台或 AWS CLI 來刪除您在 中建立的Music資料表步驟 1：在 DynamoDB 建立資料表。

如需 DynamoDB 中資料表操作的詳細資訊，請參閱 在 DynamoDB 中使用資料表和資料。

AWS 管理主控台

若要使用主控台刪除 Music 資料表：

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 選擇資料表清單中 Music 資料表旁的核取方塊。
4. 選擇 刪除。

AWS CLI

下列 AWS CLI 範例使用 刪除Music資料表delete-table。

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

AWS 開發套件

下列程式碼範例示範如何使用 AWS SDK 刪除 DynamoDB 資料表。

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
 deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
 table {tableName}. {ex.Message}");
```

```
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
 {tableName}. {ex.Message}");
            return false;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteTable。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DeleteTable。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DeleteTable。

CLI

AWS CLI

刪除資料表

下列 `delete-table` 範例會刪除 `MusicCollection` 資料表。

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

輸出：

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
```

```
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的刪除資料表。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DeleteTable。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DeleteTable。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DeleteTable。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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;
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 DeleteTable。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 DeleteTable。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DeleteTable。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 DeleteTable。

PowerShell

Tools for PowerShell V4

範例 1：刪除指定的資料表。在繼續操作之前，系統會提示您確認。

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

範例 2：刪除指定的資料表。在繼續操作之前，系統不會提示您進行確認。

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 DeleteTable。

Tools for PowerShell V5

範例 1：刪除指定的資料表。在繼續操作之前，系統會提示您確認。

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

範例 2：刪除指定的資料表。在繼續操作之前，系統不會提示您進行確認。

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 DeleteTable。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DeleteTable。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
# 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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 DeleteTable。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DeleteTable。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 DeleteTable。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 DeleteTable。

如需更多的 DynamoDB 範例，請參閱 使用 AWS SDKs DynamoDB 程式碼範例。

繼續了解 DynamoDB

如需使用 Amazon DynamoDB 的詳細資訊，請參閱下列主題：

- 在 DynamoDB 中使用資料表和資料
- 在 DynamoDB 使用項目和屬性
- 在 DynamoDB 中查詢資料表
- 在 DynamoDB 中使用全域次要索引
- 搭配交易使用
- 使用 DynamoDB Accelerator (DAX) 的記憶體內加速
- 使用 DynamoDB 和 AWS SDKs程式設計

使用 Console-to-Code 產生 Amazon DynamoDB 的基礎設施程式碼

Amazon Q Developer 的 Console-to-Code 功能可將手動資料表建立步驟轉換為可重現的自動化程式碼，以簡化 Amazon DynamoDB 的基礎設施管理。此功能可協助開發人員在其環境中有效率地擴展資料庫資源組態。如需詳細資訊，請參閱AWS 服務 使用 Amazon Q Developer Console-to-Code 自動化。

Console-to-Code 會擷取詳細的 DynamoDB 資料表組態，包括分割區索引鍵、排序索引鍵、佈建的輸送量設定和次要索引，並將它們轉換為精確的基礎設施即程式碼範本。使用生成式 AI，此工具可確保產生的程式碼維持主控台工作流程期間建立的參數相容性。

開發人員可以產生多種格式的 DynamoDB 基礎設施程式碼，例如：

- AWS TypeScript、Python 和 Java 中的雲端開發套件 (AWS CDK)
- CloudFormation 在 YAML 或 JSON 中

這種方法可讓團隊：

- 標準化資料庫資源管理
- 實作版本控制的基礎設施
- 減少手動組態錯誤

Amazon DynamoDB 的 Console-to-Code 適用於所有商業 AWS 區域，提供強大的解決方案，可將手動組態轉換為自動化、可重現的基礎設施程式碼。

運作方式

搭配 DynamoDB 使用 Console-to-Code 時，程序通常涉及：

1. 在主控台中設計原型 – 使用 DynamoDB 主控台來建立和設定資源，例如資料表。如需詳細資訊，請參閱連接到 Amazon DynamoDB。
2. 錄製動作 – Console-to-Code 會在您執行動作時記錄這些動作。
3. 程式碼產生 – 此功能使用 Amazon Q Developer 的生成式 AI 功能，將您的主控台動作轉換為您偏好格式的可重複使用的程式碼。
4. 程式碼自訂 – 接著您可以複製或下載此程式碼，並進一步針對您的生產工作負載進行自訂。

搭配 DynamoDB 使用 Console-to-Code 的優勢

簡化自動化

只要按一下，即可將手動 DynamoDB 資料表建立和組態轉換為可重複使用的程式碼。

最佳實務

產生的程式碼遵循可靠部署的 AWS 引導式最佳實務。

橋接主控台與程式碼

您不需要選擇使用 AWS 管理主控台 或 Infrastructure-as-Code (IaC)。您可以同時使用這兩種方法。

加速開發

快速開始自動化程式碼，還能針對生產用途進一步自訂。

範例使用案例

- 建立具備特定屬性、索引鍵和容量設定的 DynamoDB 資料表
- 設定全域次要索引和本機次要索引
- 設定 DynamoDB 資料表的自動擴展政策
- 建立備份與還原組態
- 建立和管理 DynamoDB 串流

開始使用

開始將 Console-to-Code 與 DynamoDB 搭配使用：

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodbv2/ 開啟 DynamoDB 主控台。
2. 開始透過主控台介面建立或修改 DynamoDB 資源。
3. 使用 Console-to-Code 功能，以您偏好的格式產生動作的程式碼。
4. 複製或下載產生的程式碼，並根據您的特定需求進行自訂。

如需如何使用 Console-to-Code 的詳細資訊和詳細說明，請參閱《Amazon Q Developer 使用者指南》中的 AWS 服務 使用 Amazon Q Developer Console-to-Code 自動化。

Amazon DynamoDB：運作方式

下列各節提供 Amazon DynamoDB 服務元件及其互動方式概觀。

主題

- DynamoDB 速查表
- Amazon DynamoDB 的核心元件
- DynamoDB API
- Amazon DynamoDB 中支援的資料類型和命名規則
- DynamoDB 資料表類別
- DynamoDB 的分割區與資料分配
- 了解如何從 SQL 過渡到 NoSQL
- Amazon DynamoDB 學習資源與工具

DynamoDB 速查表

此備忘單提供使用 Amazon DynamoDB 及其各種 AWS SDKs快速參考。

初始 設定

1. 註冊。 AWS
2. 取得 AWS 存取金鑰，以程式設計方式存取 DynamoDB。
3. 設定您 DynamoDB 憑證。

另請參閱:

- 設定 DynamoDB (Web 服務)
- DynamoDB 入門
- 核心元件的基本概述

SDK 或 CLI

選擇您偏好的 SDK，或設定 AWS CLI。

> **Note**
>
> 當您 AWS CLI 在 Windows 上使用 時，不在引號內的反斜線 (\) 會被視為歸位字元。此外，您必須逸出其他引號內的任何引號和大括號。如需範例，請參閱下一章節「建立資料表」的 Windows 標籤。

另請參閱:

- AWS CLI 搭配 DynamoDB
- DynamoDB 入門 - 步驟 2

基本動作

本節提供基本 DynamoDB 任務的程式碼。如需這些任務的詳細資訊，請參閱 DynamoDB 和 AWS SDKs 入門。

建立資料表

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
```

將項目寫入資料表

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

從資料表讀取項目

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

從資料表刪除項目

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

查詢資料表

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

刪除資料表

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

列出資料表名稱

```
aws dynamodb list-tables
```

命名規則

- 所有名稱都必須使用 UTF-8 編碼並區分大小寫。
- 資料表名稱與索引名稱長度必須介於 3 到 255 個字元之間，而且只能包含下列字元：
  - `a-z`
  - `A-Z`
  - `0-9`

- _ (底線)
- - (連字號)
- . (點號)

- 屬性名稱至少必須為一個字元長，而且大小不能超過 64KB。

如需詳細資訊，請參閱命名規則。

服務配額基本概念

讀取和寫入單位

- 讀取容量單位 (RCU) – 每秒一個高度一致性讀取，或每秒兩個最終一致讀取，適用於大小上限為 4 KB 的項目。
- 寫入容量單位 (WCU) – 每秒一個寫入，適用於大小上限為 1 KB 的項目。

資料表限制

- 資料表大小 – 資料表大小沒有任何實際限制。就項目數或位元組數而言，資料表是沒有限制的。
- 資料表數量 – 對於任何 AWS 帳戶，每個 AWS 區域的初始配額為 2，500 個資料表。
- 查詢和掃描的頁面大小限制 – 每個查詢或掃描的每頁限制為 1 MB。若您在資料表上的查詢參數或掃描操作產生超過 1 MB 的資料，DynamoDB 會傳回初始相符項目。同時也會傳回一個 `LastEvaluatedKey` 屬性，您可以在新的請求中使用該屬性來讀取下一頁。

索引

- 本機次要索引 (LSI) – 您可以定義最多五個本機次要索引。當索引必須與基礎資料表保持高度一致時，LSI 就特別有用。
- 全域次要索引 (GSIs) – 每個資料表有 20 個全域次要索引的預設配額。
- 每份資料表投影次要索引屬性 – 您最多可以投影 100 個屬性到資料表所有的區域和全域次要索引。這只適用於使用者指定的投影屬性。

分割區索引鍵

- 分割區索引鍵值的長度下限為 1 個位元組。長度上限為 2048 個位元組。
- 資料表或次要索引中，不同的分割區索引鍵值數目沒有實際限制。

- 排序索引鍵值的長度下限為 1 個位元組。長度上限為 1024 個位元組。
- 一般而言，每個分割區索引鍵值的相異排序索引鍵值數目沒有實際限制。例外狀況是有次要索引的資料表。

如需次要索引、分割區索引鍵設計和排序索引鍵設計的詳細資訊，請參閱最佳實務。

常用資料類型限制

- String (字串) – 字串長度受到項目大小上限 400 KB 的限制。字串是 UTF-8 二進位編碼的 Unicode。
- 數字 – 數字的精準度最多可達 38 位數，可為正數、負數或零。
- 二進位 – 二進位長度受到項目大小上限 400 KB 的限制。使用二進制屬性的應用程式必須先以 base64 編碼資料，再傳送到 DynamoDB。

如需支援的完整資料類型清單，請參閱資料類型。如需詳細資訊，請參閱服務配額。

項目、屬性和表達式參數

DynamoDB 的項目大小上限是 400 KB，包括屬性名稱二進位長度 (UTF-8 長度) 和屬性值二進位長度 (UTF-8 長度)。屬性名稱算作大小限制的一部分。

清單、映射或集合中值數目不限，只要含有值的項目符合 400 KB 項目大小限制。

針對表達式參數，任何表達式字串的長度上限為 4 KB。

如需項目大小、屬性和表達式參數的詳細資訊，請參閱服務配額。

其他資訊

- 安全性
- 監控和記錄
- 使用串流
- 備份和時間點復原
- 與其他 AWS 服務整合
- API 參考
- 架構中心：資料庫最佳實務
- 教學課程影片
- DynamoDB 論壇

Amazon DynamoDB 的核心元件

在 DynamoDB 中，資料表、項目與屬性都是您會用到的核心元件。資料表是項目的集合，而每個項目則是屬性的集合。DynamoDB 使用主索引鍵來唯一識別資料表中的每個項目。您可以使用 DynamoDB Streams 來擷取 DynamoDB 資料表中的資料修改事件。

DynamoDB 中有其限制。如需詳細資訊，請參閱Amazon DynamoDB 中的配額。

以下影片將為您介紹資料表、項目與屬性。

資料表、項目與屬性

資料表、項目與屬性

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"}

以下是基本 DynamoDB 元件：

- 資料表：與其他資料庫系統類似，DynamoDB 會將資料存放在資料表中。資料表是資料的集合。例如，您可以使用名為 People 的資料表範例，來存放朋友、家人或其他任何人的相關個人聯絡資訊。您也可以使用 Cars 資料表來存放各人駕駛之車輛的相關資訊。
- 項目：每個資料表包含零或多個項目。項目是可從所有其他項目唯一識別的一組屬性。在 People 資料表中，每個項目代表一個人。在 Cars 資料表中，每個項目代表一輛車。DynamoDB 中的項目與其他資料庫系統中的資料列、紀錄或元組有許多相似之處。在 DynamoDB 中，可以存放在資料表中的項目數不限。

- 屬性：每個項目是由一或多個屬性所組成。屬性是一種基本資料元素，不必再進一步細分。例如，People 資料表中的項目包含名為 PersonID、LastName、FirstName 等屬性。在 Department 資料表中，項目可能會有 DepartmentID、Name、Manager 等屬性。DynamoDB 中的屬性與其他資料庫系統中的欄位或資料行有許多相似之處。

下圖顯示一個名為 People 的資料表，其中包含一些範例項目與屬性。

```
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"
}
```

People 資料表的注意事項如下：

- 資料表中的每個項目都有唯一識別符或主索引鍵，可區分該項目與資料表中的所有其他項目。在 People 資料表中，主索引鍵是由一個屬性 (PersonID) 所組成。
- 除了主索引鍵之外，People 資料表沒有結構描述，這表示您不需要事先定義屬性或其資料類型。每個項目可以有其專屬的不同屬性。
- 大多數屬性為純量，亦即只能有一個值。字串與數字是常見的純量範例。
- 有些項目有巢狀屬性 (Address)。DynamoDB 支援巢狀屬性，最多 32 層深。

以下是另一個名為 Music 的範例資料表，您可以用來追蹤音樂收藏。

```
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"
}
```

Music 資料表的注意事項如下：

- Music 的主索引鍵是由兩個屬性 (Artist 與 SongTitle) 所組成。資料表中的每個項目必須有這兩個屬性。Artist 與 SongTitle 的組合可區分資料表中的每個項目與所有其他項目。
- 除了主索引鍵之外，Music 資料表沒有結構描述，這表示您不需要事先定義屬性或其資料類型。每個項目可以有其專屬的不同屬性。
- 其中一個項目有巢狀屬性 (PromotionInfo)，包含其他的巢狀屬性。DynamoDB 支援巢狀屬性，最多 32 層深。

如需詳細資訊，請參閱 在 DynamoDB 中使用資料表和資料。

主索引鍵

當您建立資料表時，除了資料表名稱，您還必須指定資料表的主索引鍵。主索引鍵可唯一識別資料表中的每個項目，因此兩個項目不能具有相同的索引鍵。

DynamoDB 支援兩種不同類型的主索引鍵：

- 分割區索引鍵：簡易主索引鍵，由一個屬性 (稱為分割區索引鍵) 所組成。

  DynamoDB 使用分割區索引鍵值作為內部雜湊函數的輸入。雜湊函數的輸出決定要存放項目的分割區 (DynamoDB 的內部實體儲存體)。

在只有一個分割區索引鍵的資料表中，沒有兩個項目的分割區索引鍵值是相同的。

所以 資料表、項目與屬性 中描述的 People 資料表，是具有簡單主索引鍵 (PersonID) 的資料表範例。您可以藉由提供 People 資料表中任何項目的 PersonId 值，來直接存取該項目。

- 分割區索引鍵與排序索引鍵：稱為複合主鍵，這種類型的索引鍵是由兩個屬性組成。第一個屬性是分割區索引鍵，第二個屬性是排序索引鍵。

  DynamoDB 使用分割區索引鍵值作為內部雜湊函數的輸入。雜湊函數的輸出決定要存放項目的分割區 (DynamoDB 的內部實體儲存體)。具有相同分割區索引鍵值的所有項目會存放在一起，並依排序索引鍵值排序。

  在具有一個分割區金鑰與一個排序索引鍵的資料表中，兩個項目可能會有相同的分割區索引鍵值。不過，這兩個項目必須具有不同的排序索引鍵值。

  資料表、項目與屬性 中描述的 Music 資料表，是具有複合主鍵 (Artist 和 SongTitle) 的資料表範例。如果您提供 Music 資料表中任何項目的 Artist 與 SongTitle 值，即可直接存取該項目。

  複合主鍵可讓您更有彈性地查詢資料。例如，如果您只提供 Artist 值，則 DynamoDB 會擷取該演出者的所有歌曲。您可以提供一個 Artist 值與一段範圍的 SongTitle 值，來擷取特定演出者的歌曲子集。

> **Note**
>
> 項目的分割區索引鍵也稱為其雜湊屬性。雜湊屬性一詞衍生自 DynamoDB 中內部雜湊函數的用法，可將資料項目根據其分割區索引鍵值平均分佈到所有分割區。
> 項目的排序索引鍵也稱為其範圍屬性。範圍屬性一詞衍生自 DynamoDB 存放項目的方式，具有相同分割區索引鍵的項目會實際緊密相鄰，並依排序索引鍵值排序。

每個主索引鍵屬性必須是純量 (亦即只能保留一個值)。主索引鍵屬性允許的資料類型僅限於字串、數字或二進位。其他非索引鍵屬性則沒有此限制。

次要索引

您可以在資料表上建立一或多個次要索引。次要索引可讓您在除了使用主索引鍵查詢外，也可使用備用索引鍵查詢資料表中的資料。DynamoDB 不需要您使用索引，但可讓您的應用程式在查詢資料時更具靈活性。在資料表上建立次要索引之後，您可以從索引讀取資料，方法與從資料表讀取十分相似。

DynamoDB 支援兩種索引：

- 全域次要索引：一種含分割區索引鍵或排序索引鍵的索引，這些索引鍵可與資料表上的索引鍵不同。在全域次要索引中，主索引鍵值無需唯一。
- 本機次要索引：是一種與資料表擁有相同分割區索引鍵但不同排序索引鍵的索引。

在 DynamoDB 中，全域次要索引 (GSI) 是涵蓋整個資料表的索引，可讓您查詢所有分割區索引鍵。本機次要索引 (LSI) 是一種與基礎資料表共用相同分割區索引鍵，但具有不同排序索引鍵的索引。

DynamoDB 中的每個資料表配額為 20 個全域次要索引 (預設配額) 與 5 個本機次要索引。

在上述範例 Music 資料表中，您可以依 Artist (分割區索引鍵) 或依 Artist 與 SongTitle (分割區索引鍵與排序索引鍵) 查詢資料項目。如果您也想要依 Genre 與 AlbumTitle 查詢資料，該怎麼辦？ 若要執行此作業，您可以在 Genre 與 AlbumTitle 上建立索引，然後查詢索引，方法與查詢 Music 資料表十分相似。

下圖顯示 Music 資料表範例，其中包含一個名為 GenreAlbumTitle 的新索引。在此索引中，Genre 是分割區索引鍵，而 AlbumTitle 是排序索引鍵。

音樂資料表	GenreAlbumTitle

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

<pre>{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The
 Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,</pre>
	<pre>{
	    "Genre": "Country",
	    "AlbumTitle": "Somewhat Famous",
	    "Artist": "No One You Know",
	    "SongTitle": "Somewhere Down The
	 Road"
	}</pre>

音樂資料表	GenreAlbumTitle

<pre>{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts
 Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": {
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        },
        "TourDates": {
            "Seattle": "20150622",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}</pre>
	<pre>{
	    "Genre": "Rock",
	    "AlbumTitle": "The Buck Starts
	 Here",
	    "Artist": "The Acme Band",
	    "SongTitle": "Still In Love"
	}</pre>

<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>

GenreAlbumTitle 索引的注意事項如下：

- 每個索引都屬於一個資料表，稱為索引的基礎資料表。在上述範例中，Music 是 GenreAlbumTitle 索引的基礎資料表。
- DynamoDB 會自動維護索引。當您新增、更新或刪除基礎資料表中的項目時，DynamoDB 會在屬於該資料表的任何索引中新增、更新或刪除對應的項目。
- 建立索引時，您可以指定要從基礎資料表複製或投影到索引的屬性。DynamoDB 至少會將索引鍵屬性從基礎資料表投影到索引。`GenreAlbumTitle` 即為一例，其中只有索引鍵屬性會從 `Music` 資料表投影到索引。

您可以查詢 GenreAlbumTitle 索引，尋找特定內容類型的所有專輯 (例如所有 Rock 專輯)。您也可以查詢此索引，尋找特定內容類型中具有特定專輯標題的所有專輯 (例如標題開頭字母為 H 的所有 Country 專輯)。

如需詳細資訊，請參閱 在 DynamoDB 中使用次要索引來改善資料存取。

DynamoDB Streams

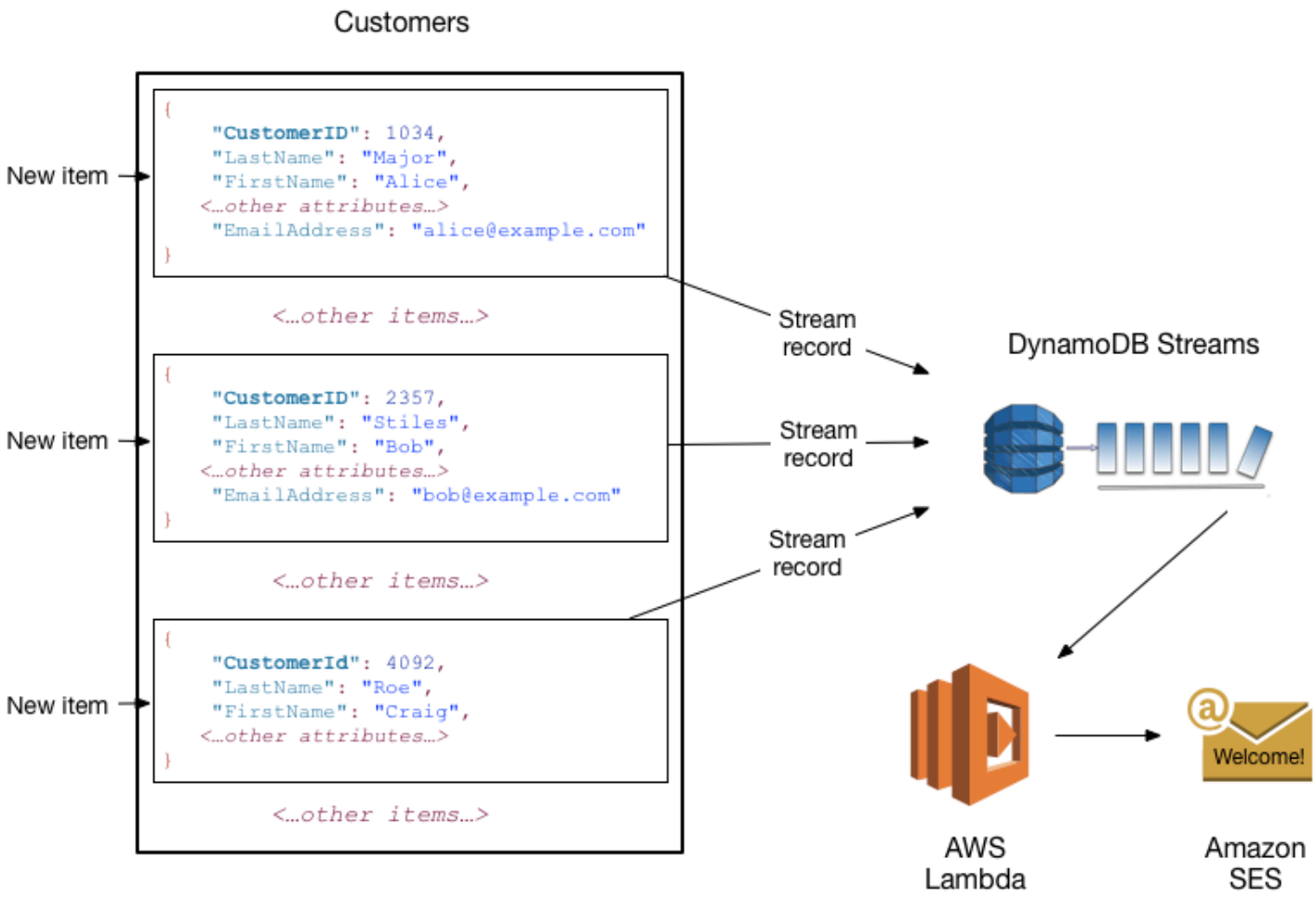
DynamoDB Streams 是選用功能，可擷取 DynamoDB 資料表中的資料修改事件。這些事件的相關資料會依事件出現的順序，近乎即時地出現在串流中。

每個事件是以串流紀錄表示。如果您在資料表上啟用串流，只要發生下列其中一個事件，DynamoDB Streams 就會寫入一個串流紀錄：

- 新增項目至資料表：串流會擷取整個項目的影像，包括其所有屬性。
- 項目已更新：串流會擷取項目中已修改之任何屬性的「之前」與「之後」影像。
- 從資料表刪除項目：串流會擷取整個項目的影像，再加以刪除。

每個串流紀錄也會包含資料表的名稱、事件時間戳記與其他中繼資料。串流紀錄的存留期為 24 小時，之後會自動從串流移除。

您可以將 DynamoDB Streams 與 搭配使用 AWS Lambda 來建立觸發，只要串流中出現感興趣的事件，就會自動執行程式碼。例如，以含有公司客戶資訊的 Customers 資料表為例。假設您想要傳送「歡迎」電子郵件給每個新客戶。您可以在該資料表上啟用串流，然後將串流與 Lambda 函式建立關聯。Lambda 函式會在每次出現新的串流紀錄時執行，但只會處理 Customers 資料表的新增項目。針對具有 `EmailAddress` 屬性的任何項目，Lambda 函式會調用 Amazon Simple Email Service (Amazon SES) 來將電子郵件傳送至該地址。



> **Note**
>
> 在此範例中，最後一個客戶 Craig Roe 不會收到電子郵件，因為他沒有 `EmailAddress`。

除了觸發之外，DynamoDB Streams 還支援強大的解決方案，例如區域內和跨 AWS 區域的資料複寫、DynamoDB 資料表中資料的具體化視觀表、使用 Kinesis 具體化視觀表的資料分析等等。

如需詳細資訊，請參閱DynamoDB Streams 的變更資料擷取。

DynamoDB API

若要使用 Amazon DynamoDB，您的應用程式必須使用一些簡單的 API 操作。以下是這些操作的摘要，並依類別分組。

> **Note**
>
> 如需 API 操作的完整清單，請參閱 Amazon DynamoDB API 參考。

主題

- 控制平面
- 資料平面
- DynamoDB Streams
- 交易

控制平面

控制平面操作可讓您建立及管理 DynamoDB 資料表。它們也可讓您使用索引、串流，以及相依於資料表的其他物件。

- `CreateTable`：建立新的資料表。您可以選擇性地建立一或多個次要索引，並為資料表啟用 DynamoDB Streams。
- `DescribeTable`：傳回資料表的相關資訊，例如其主索引鍵結構描述、輸送量設定、索引資訊。
- `ListTables`：傳回您所有的資料表名稱清單。
- `UpdateTable`：修改資料表或其索引的設定、在資料表上建立或移除新的索引，或修改資料表的 DynamoDB Streams 設定。
- `DeleteTable`：從 DynamoDB 移除資料表及其所有相依物件。

資料平面

資料平面操作可讓您對資料表中的資料執行建立、讀取、更新與刪除 (也稱為 CRUD) 動作。部分資料平面操作也可讓您從讀取資料。

您可以使用 PartiQL：一種適用於 Amazon DynamoDB 的 SQL 相容查詢語言 來執行這些 CRUD 操作，也可以使用 DynamoDB 的傳統 CRUD API 將每個操作分隔為不同的 API 呼叫。

PartiQL - 一種 SQL 相容查詢語言

- `ExecuteStatement`：讀取資料表中的多個項目。您也可以寫入或更新資料表中的單一項目。在寫入或更新單一項目時，您必須指定主索引鍵屬性。

- BatchExecuteStatement：從資料表寫入、更新或讀取多個項目。這比 ExecuteStatement 的效率高上好幾倍，因為您的應用程式只需要往返網路一次即可寫入或讀取項目。

傳統 API

建立資料

- PutItem：將單一項目寫入資料表。您必須指定主索引鍵屬性，但不必指定其他屬性。
- BatchWriteItem：最多可將 25 個項目寫入資料表。這比呼叫 PutItem 的效率高上好幾倍，因為您的應用程式只需要往返網路一次即可寫入項目。

讀取資料

- GetItem：從資料表擷取單一項目。您必須指定所需項目的主索引鍵。您可以擷取整個項目，或只擷取其屬性子集。
- BatchGetItem：最多可從一或多個資料表擷取 100 個項目。這比呼叫 GetItem 的效率高上好幾倍，因為您的應用程式只需要往返網路一次即可讀取項目。
- Query：擷取所有具有特定分割區索引鍵的項目。您必須指定分割區索引鍵值。您可以擷取整個項目，或只擷取其屬性子集。您可以選擇性地將條件套用至排序索引鍵值，只擷取具有相同分割區索引鍵的資料子集。您可以在資料表上使用此操作，只要該資料表同時具有分割區索引鍵與排序索引鍵。您也可以在索引上使用此操作，只要該索引同時具有分割區索引鍵與排序索引鍵。
- Scan：擷取指定資料表或索引中的所有項目。您可以擷取整個項目，或只擷取其屬性子集。您可以選擇性地套用篩選條件，只傳回您感興趣的值並捨棄其餘值。

更新資料

- UpdateItem：修改項目中的一或多個屬性。您必須指定要修改之項目的主索引鍵。您可以新增屬性，以及修改或移除現有的屬性。您也可以執行條件式更新，只在符合使用者定義的條件時，更新才會成功。您可以選擇性地實作原子計數器，遞增或遞減數字屬性，而不會影響其他寫入請求。

刪除資料

- DeleteItem：從資料表刪除單一項目。您必須指定要刪除之項目的主索引鍵。
- BatchWriteItem：最多可從一或多個資料表刪除 25 個項目。這比呼叫 DeleteItem 的效率高上好幾倍，因為您的應用程式只需要往返網路一次即可刪除項目。

> **Note**
>
> 您可以使用 `BatchWriteItem` 建立資料和刪除資料。

DynamoDB Streams

DynamoDB Streams 操作可讓您在資料表上啟用或停用串流，並允許存取串流中包含的資料修改紀錄。

- `ListStreams`：傳回您所有的串流清單，或只傳回特定資料表的串流。
- `DescribeStream`：傳回串流的相關資訊，例如其 Amazon Resource Name (ARN)，以及您的應用程式可開始讀取前幾個串流紀錄的位置。
- `GetShardIterator`：傳回碎片疊代運算，這是您的應用程式從串流擷取紀錄所使用的資料結構。
- `GetRecords`：使用指定的碎片疊代運算，擷取一或多個串流紀錄。

交易

交易提供了不可分割性、一致性、隔離性和耐久性 (ACID)，讓您能夠輕鬆地維持應用程式的資料正確度。

您可以使用 PartiQL：一種適用於 Amazon DynamoDB 的 SQL 相容查詢語言 來執行交易操作，也可以使用 DynamoDB 的傳統 CRUD API 將每個操作分隔為不同的 API 呼叫。

PartiQL - 一種 SQL 相容查詢語言

- `ExecuteTransaction`：允許對資料表內和跨資料表的多個項目進行保證全有或全無變更 CRUD 操作的批次操作。

傳統 API

- `TransactWriteItems`：允許對資料表內和跨資料表的多個項目進行保證全有或全無變更的 `Put`、`Update` 和 `Delete` 操作的批次操作。
- `TransactGetItems`：允許 `Get` 操作來從一或多個資料表擷取多個項目的批次操作。

Amazon DynamoDB 中支援的資料類型和命名規則

本節說明 Amazon DynamoDB 命名規則與 DynamoDB 支援的各種資料類型。這些資料類型有所限制。如需詳細資訊，請參閱 資料類型。

主題

- 命名規則
- 資料類型
- 資料類型描述項

命名規則

DynamoDB 中的資料表、屬性與其他物件必須具有名稱。名稱應該具有意義且簡潔，例如 Products、Books 與 Authors 等都是一目了然的名稱。

以下是 DynamoDB 的命名規則：

- 所有名稱都必須使用 UTF-8 編碼並區分大小寫。
- 資料表名稱與索引名稱長度必須介於 3 到 255 個字元之間，而且只能包含下列字元：
  - `a-z`
  - `A-Z`
  - `0-9`
  - `_` (底線)
  - `-` (破折號)
  - `.` (點號)
- 屬性名稱至少必須為一個字元長，而且大小不能超過 64KB。屬性名稱盡可能簡潔是公認的最佳實務。這有助於減少使用的讀取請求單位，因為屬性名稱包含在儲存體和輸送量使用量的計量中。

  以下為例外狀況。這些屬性名稱絕對不能超過 255 個字元。

  - 次要索引分割區索引鍵名稱
  - 次要索引排序索引鍵名稱
  - 任何使用者指定的投影屬性名稱 (僅適用於本機次要索引)

保留字與特殊字元

DynamoDB 有一份保留字與特殊字元清單。如需完整清單，請參閱 DynamoDB 中的保留字。此外，下列字元在 DynamoDB 中具有特殊意義：# (井字號) 與 : (冒號)。

雖然 DynamoDB 可讓您將這些保留字與特殊字元用於名稱，但建議您避免這樣做，因為每次在表達式中使用這些名稱，都必須定義預留位置變數。如需詳細資訊，請參閱 DynamoDB 中的表達式屬性名稱 (別名)。

資料類型

針對資料表中的屬性，DynamoDB 支援許多不同的資料類型。其可分類如下：

- 純量類型：純量類型只能代表一個值。純量類型包括數字、字串、二進位、布林值與 Null。
- 文件類型：文件類型可代表具有巢狀屬性的複雜結構，如 JSON 文件中所示。文件類型為清單與映射。
- 集合類型：集合類型可代表多個純量值。集合類型包括字串集、數字集與二進位集。

當您建立資料表或次要索引時，您必須指定每個主索引鍵屬性 (分割區索引鍵與排序索引鍵) 的名稱與資料類型。此外，您必須將每個主索引鍵屬性定義為字串、數字或二進位類型。

DynamoDB 是 NoSQL 資料庫且沒有結構描述。這表示除了主索引鍵屬性之外，您不需要在建立資料表時定義任何屬性或資料類型。相較之下，關聯式資料庫需要您在建立資料表時定義每個資料行的名稱與資料類型。

以下是每個資料類型的說明，以及 JSON 格式的範例。

純量類型

純量類型包括數字、字串、二進位、布林值與 Null。

Number

數字可以是正數、負數或零。數字精確度最高可達 38 位數。超過此值會導致例外狀況。若您需要比 38 位數更高的精確度，則可以使用字串。

- 正數範圍：1E-130 到 9.99999999999999999999999999999999999999E+125
- 負數範圍：-9.99999999999999999999999999999999999999E+125 到 -1E-130

在 DynamoDB 中，數字會以變數長度表示。前後的零會截去。

所有數字都會以字串形式跨網路傳送到 DynamoDB，以提高語言與程式庫之間的相容性。不過，DynamoDB 會將其視為數學運算的數字類型屬性。

您可以使用數字資料類型來代表日期或時間戳記。一個做法是使用 epoch 時間，也就是自 1970 年 1 月 1 日 00:00:00 UTC 起經過的秒數。例如，epoch 時間 `1437136300` 代表 2015 年 7 月 17 日下午 12:31:40 UTC。

如需詳細資訊，請參閱 http://en.wikipedia.org/wiki/Unix_time。

String

字串是 UTF-8 二進位編碼的 Unicode。如果屬性未用作為索引或資料表的索引鍵，則字串的最小長度可以是零，而且受到最大 DynamoDB 項目大小限制 400 KB 的限制。

下列其他限制適用於定義為類型字串的主索引鍵屬性：

- 針對簡易主索引鍵，第一個屬性值 (分割區索引鍵) 的長度上限為 2048 位元組。
- 針對複合主鍵，第二個屬性值 (排序索引鍵) 的長度上限為 1024 位元組。

DynamoDB 會收集並比較使用基本 UTF-8 字串編碼位元組的字串。例如，"a" (0x61) 大於 "A" (0x41)，而 "¿" (0xC2BF) 大於 "z" (0x7A)。

您可以使用字串資料類型來代表日期或時間戳記。一個做法是使用 ISO 8601 字串，如下列範例所示：

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

如需詳細資訊，請參閱 http://en.wikipedia.org/wiki/ISO_8601。

> **Note**
>
> 與傳統的關聯式資料庫不同，DynamoDB 原生不支援日期和時間資料類型。使用 Unix Epoch 時間可能很有用，可用於將日期和時間資料儲存為數字資料類型。

二進位

二進位類型屬性可存放任何二進位資料，例如壓縮文字、加密資料或影像。每當 DynamoDB 比較二進位值時，都會將二進位資料的每個位元組視為不帶正負號。

如果屬性未用作為索引或資料表的索引鍵，則二進制屬性的長度可以是零，而且受到最大 DynamoDB 項目大小限制 400 KB 的限制。

如果您將主索引鍵屬性定義為二進位類型屬性，則會有以下其他限制：

- 針對簡易主索引鍵，第一個屬性值 (分割區索引鍵) 的長度上限為 2048 位元組。
- 針對複合主鍵，第二個屬性值 (排序索引鍵) 的長度上限為 1024 位元組。

您的應用程式必須將二進位值編碼為 base64 編碼格式，再傳送到 DynamoDB。收到這些值時，DynamoDB 會將資料解碼為不帶正負號的位元組陣列，並用作二進制屬性長度。

下列範例顯示使用 base64 編碼文字的二進制屬性。

```
dGhpcyB0ZXh0IGlzIGJhc2U2NC1lbmNvZGVk
```

Boolean

布林值類型屬性可存放 `true` 或 `false`。

Null

Null 代表具有未知或未定義狀態的屬性。

文件類型

文件類型為清單與映射。這些資料類型可彼此互相巢狀來代表複雜的資料結構，最高可達 32 層深。

清單或映射中的值數目不限，只要含有值的項目符合 DynamoDB 項目大小限制 (400 KB)。

如果屬性不用於資料表或索引鍵，則屬性值可以是空字串或空的二進位值。屬性值不得為空的集合 (字串集合、數字集合或二進位集合)。然而，系統允許空的清單和映射。清單和映射中允許空字串和二進位值。如需詳細資訊，請參閱屬性。

清單

清單類型屬性可存放一組排序的值。清單會以方括號括住：`[ ... ]`

清單類似於 JSON 陣列。清單元素中可存放的資料類型不限，而且清單元素中的元素不必屬於相同類型。

下列範例顯示一個清單，其中包含兩個字串與一個數字。

```
FavoriteThings: ["Cookies", "Coffee", 3.14159]
```

> **Note**
>
> DynamoDB 可讓您使用清單中的個別元素，即使這些元素的巢狀結構很深也一樣。如需詳細資訊，請參閱在 DynamoDB 中使用表達式。

Map

映射類型屬性可存放一組未排序的名稱/值對。映射會以大括弧括住：`{ ... }`

映射類似於 JSON 物件。映射元素中可存放的資料類型不限，而且映射中的元素不必屬於相同類型。

映射很適合用來將 JSON 文件存放到 DynamoDB 中。下列範例顯示一個映射，其中包含字串、數字與含有另一個映射的巢狀清單。

```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
        "Telephone",
        {
            Pens: { Quantity : 3},
            Pencils: { Quantity : 2},
            Erasers: { Quantity : 1}
        }
    ]
}
```

> **Note**
>
> DynamoDB 可讓您使用映射中的個別元素，即使這些元素的巢狀結構很深也一樣。如需詳細資訊，請參閱在 DynamoDB 中使用表達式。

集合

DynamoDB 支援代表數字集合、字串集合或二進位值集合的類型。集合內的所有元素必須屬於相同類型。例如，數字集合只能包含數字，字串集合只能包含字串。

集合中的值數目不限，只要含有值的項目符合 DynamoDB 項目大小限制 (400 KB)。

集合內的每個值必須是唯一的。集合內的值順序不會保留。因此，您的應用程式不得依賴集合內元素的任何特定順序。DynamoDB 不支援空集合，但集合中允許存在空字串和二進位值。

下列範例顯示字串集、數字集與二進位集：

```
["Black", "Green", "Red"]

[42.2, -19, 7.5, 3.14]

["U3Vubnk=", "UmFpbnk=", "U25vd3k="]
```

資料類型描述項

低階 DynamoDB API 通訊協定使用資料類型描述項做為字符，告知 DynamoDB 如何解譯每項屬性。

下列為 DynamoDB 資料類型描述項的完整清單：

- **S**：字串
- **N**：數字
- **B**：二進位
- **BOOL**：布林值
- **NULL**：Null
- **M**：映射
- **L**：清單
- **SS**：字串集合
- **NS**：數字集合
- **BS**：二進位集合

DynamoDB 資料表類別

DynamoDB 提供兩種資料表類別，旨在協助您最佳化成本。預設值為 DynamoDB 標準資料表類別，建議大多數工作負載使用。DynamoDB 標準：不常存取 (DynamoDB 標準-IA) 資料表類別針對以儲存為主要成本的資料表進行最佳化。例如，儲存不常存取資料的資料表 (例如應用程式日誌、舊社交媒體貼文、電子商務訂單歷史記錄以及過去遊戲成就) 都是適合標準-IA 資料表類別的選項。如需定價詳細資訊，請參閱 Amazon DynamoDB 定價。

每個 DynamoDB 資料表都與一個資料表類別相關聯 (依 DynamoDB Standard 預設)。與資料表相關聯的所有次要索引都使用相同的資料表類別。每個資料表類別針對資料儲存以及讀取和寫入要求提供不同的定價。您可以根據資料表的儲存體和輸送量使用模式，為資料表選取最具成本效益的資料表類別。

資料表類別的選擇不是永久性的，您可以使用 、 AWS 管理主控台 AWS CLI 或 AWS SDK 變更此設定。DynamoDB 也支援使用 管理單一區域資料表和全域資料表 AWS CloudFormation 的資料表類別。若要進一步了解如何選取資料表類別，請參閱 選擇 DynamoDB 資料表類別時的注意事項。

DynamoDB 的分割區與資料分配

Amazon DynamoDB 將資料存放在分割區中。分割區是資料表的儲存配置，由固態硬碟 (SSDs) 提供支援，並自動複寫至 AWS 區域內的多個可用區域。分割區管理完全是由 DynamoDB 處理，您永遠不需要自行管理分割區。

當您建立資料表時，資料表的初始狀態為 `CREATING`。在此階段期間，DynamoDB 會配置足夠的分割區給資料表，讓它可以處理您的佈建輸送量需求。您可以在資料表狀態變更為 `ACTIVE` 之後，開始寫入及讀取資料表資料。

DynamoDB 會在下列情況下配置額外的分割區給資料表：

- 如果您將資料表的佈建輸送量設定，增加到超過現有分割區所能支援的設定。
- 如果現有的分割區容量將滿且需要更多儲存空間。

分割區管理會在背景自動進行，而且對您的應用程式是透明的。您的資料表會全程可供使用，並完整支援您的佈建輸送量需求。

如需詳細資訊，請參閱分割區索引鍵設計。

DynamoDB 中的全域次要索引也是由分割區所組成。全域次要索引中的資料會與其基礎資料表中的資料分開存放，但索引分割區的運作方式與資料表分割區相同。

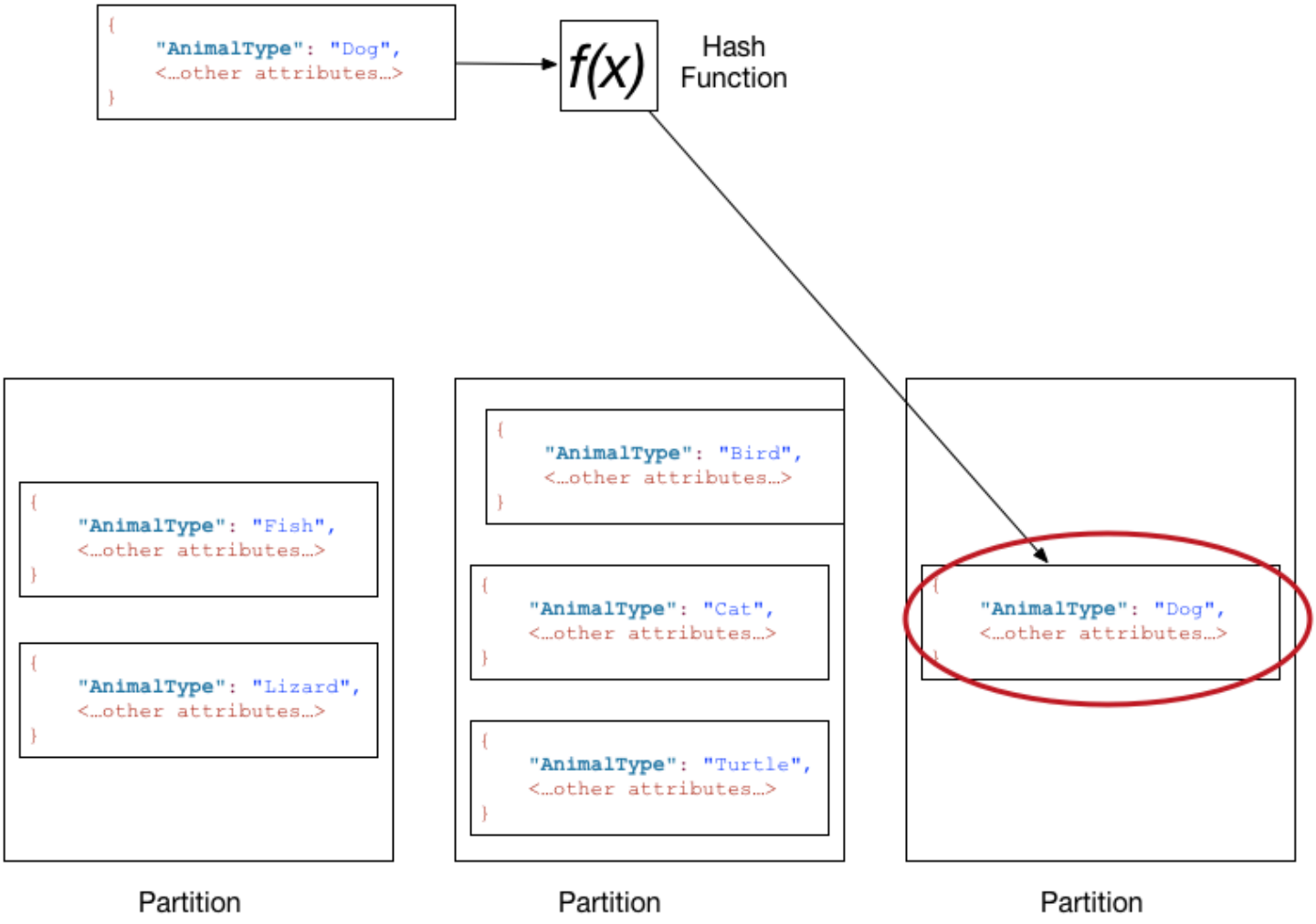
資料分佈：分割區索引鍵

如果您的資料表具有簡易主索引鍵 (僅限分割區索引鍵)，DynamoDB 會根據每個項目的分割區索引鍵值來存放與擷取項目。

為了將項目寫入資料表，DynamoDB 使用分割區索引鍵值作為內部雜湊函數的輸入。雜湊函數的輸出值決定要存放項目的分割區。

若要讀取資料表的項目，您必須指定項目的分割區索引鍵值。DynamoDB 會使用此值作為雜湊函數的輸入，產生能夠找到此項目的分割區。

下圖顯示一個名為 Pets 的資料表，其橫跨多個分割區。該資料表的主索引鍵是 AnimalType (僅顯示此索引鍵屬性)。DynamoDB 使用其雜湊函數來判斷新項目的存放位置，在此例中是依據字串 Dog 的雜湊值。請注意，項目不會依序存放。每個項目的位置取決於其分割區索引鍵的雜湊值。



> **Note**
>
> DynamoDB 已經過最佳化，可將項目一致分佈到資料表的分割區，不論有多少分割區。建議您選擇相較於資料表中的項目數，可擁有大量相異值的分割區索引鍵。

資料分佈：分割區索引鍵與排序索引鍵

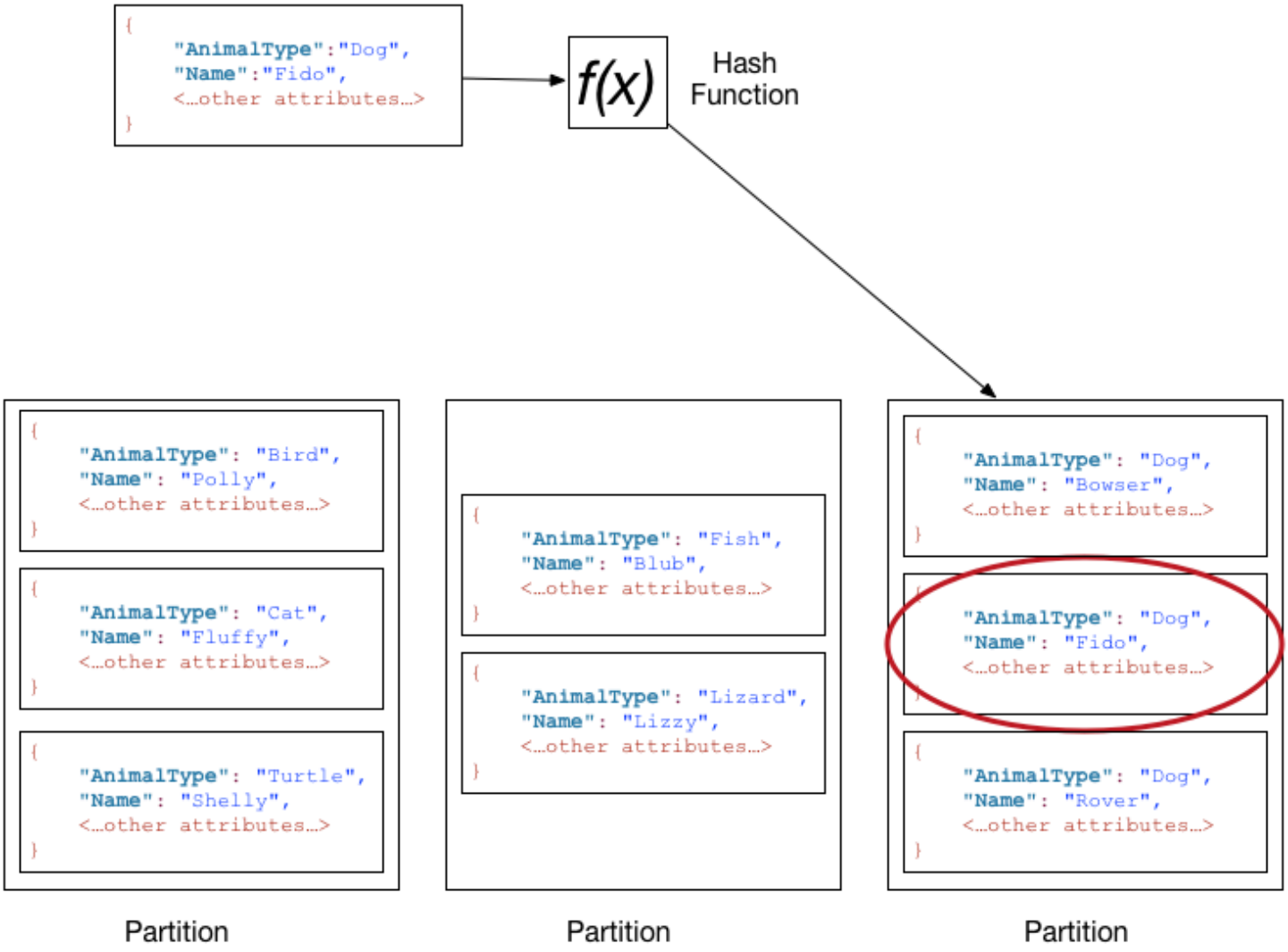
如果資料表具有複合主鍵 (分割區索引鍵和排序索引鍵)，則 DynamoDB 會以與 資料分佈：分割區索引鍵 中所述相同的方式計算分割區索引鍵的雜湊值。系統傾向將具有相同分割區索引鍵值的項目存放在相鄰位置，並依排序索引鍵屬性的值排序。具有相同分割區索引鍵屬性值的一組項目稱為「項目集合」。項目集合經過最佳化，可高效擷取集合中指定範圍的項目。若資料表未設定本機次要索引，DynamoDB 會自動將項目集合分配至所需數量的分割區，以存放資料並維持讀寫輸送量。

為了將項目寫入資料表，DynamoDB 會計算分割區索引鍵的雜湊值，來決定哪個分割區應該包含項目。在該分割區中，多個項目可以具有相同的分割區索引鍵值。因此，DynamoDB 使用相同的分割區索引鍵將項目儲存在其他項目中，按排序索引鍵遞增排列。

若要讀取資料表的項目，您必須指定項目的分割區索引鍵值和排序索引鍵值。DynamoDB 會計算分割區索引鍵的雜湊值，產生能夠找到此項目的分割區。

如果您想要的項目具有相同的分割區索引鍵值，您可以在單一操作 (`Query`) 中從資料表讀取多個項目。DynamoDB 會傳回具有該分割區索引鍵值的所有項目。您可以選擇性地將條件套用至排序索引鍵，讓它只傳回特定值範圍內的項目。

假設 Pets 資料表具有複合主鍵，其中包含 AnimalType (分割區索引鍵) 與 Name (排序索引鍵)。下圖顯示 DynamoDB 寫入一個項目，其分割區索引鍵值為 Dog，排序索引鍵值為 Fido。



若要從 Pets 資料表讀取同樣的項目，DynamoDB 會計算 Dog 的雜湊值，產生存放這些項目的分割區。然後，DynamoDB 會掃描排序索引鍵屬性值，直到找到 Fido。

若要讀取 AnimalType 為 Dog 的所有項目，您可以發出 `Query` 操作，而不需要指定排序索引鍵條件。根據預設，項目會依存放順序 (即依排序索引鍵的遞增順序) 傳回。您也可以改為請求遞減順序。

若只要查詢其中的一些 Dog 項目，您可以將條件套用至排序索引鍵 (例如僅限 Dog 項目，其中 Name 開始的字母介於 A 到 K 的範圍內)。

> **Note**
>
> 在 DynamoDB 資料表中，每個分割區索引鍵值的相異排序索引鍵值數目沒有上限。如果您需要在 Pets 資料表中存放數十億個 Dog 項目，則 DynamoDB 會分配足夠的儲存空間來自動處理此需求。

了解如何從 SQL 過渡到 NoSQL

若您是應用程式開發人員，您可能已有使用關聯式資料庫管理系統 (RDBMS) 和結構式查詢語言 (SQL) 的經驗。開始使用 Amazon DynamoDB 後，您可能會發現許多相似點，但也有許多相異點。NoSQL 一詞用來說明具高可用性、可擴展性且高效能最佳化的非關聯式資料庫系統。NoSQL 資料庫 (例如 DynamoDB) 與關聯式模型不同，它會使用另一種模型來管理資料，例如鍵/值對或文件儲存。如需詳細資訊，請參閱何謂 NoSQL？。

Amazon DynamoDB 支援 PartiQL，其為一種與 SQL 相容的開放原始碼查詢語言，讓您能以高效率的方式查詢資料，不論資料存放在何處或以何種格式存放。使用 PartiQL，您可以輕鬆處理來自關聯式資料庫的結構化資料、開放資料格式的半結構化和巢套資料，甚至是 NoSQL 或文件資料庫中允許不同行不同屬性的無固定結構資料。如需詳細資訊，請參閱 PartiQL 查詢語言。

以下章節說明常見的資料庫任務，並比較和比對 SQL 陳述式及其相對應的 DynamoDB 操作。

> **Note**
>
> 本節中的 SQL 範例與 MySQL RDBMS 相容。
> 本節中的 DynamoDB 範例會顯示 DynamoDB 操作的名稱，並以 JSON 格式呈現該操作的參數。

主題

- 在關聯式 (SQL) 資料庫與 NoSQL 之間進行取捨
- 從資料表刪除資料時，關聯式 (SQL) 資料庫與 DynamoDB 的差異
- 建立資料表時，關聯式 (SQL) 資料庫與 DynamoDB 之間的差異
- 從關聯式 (SQL) 資料庫與 DynamoDB 取得資料表資訊的差異
- 寫入資料表時，關聯式 (SQL) 資料庫與 DynamoDB 的差異
- 讀取資料表資料時，關聯式 (SQL) 資料庫與 DynamoDB 的主要差異
- 管理索引時，關聯式 (SQL) 資料庫與 DynamoDB 的差異
- 修改資料表資料時，關聯式 (SQL) 資料庫與 DynamoDB 的差異
- 從資料表刪除資料時，關聯式 (SQL) 資料庫與 DynamoDB 的差異
- 移除資料表時，關聯式 (SQL) 資料庫與 DynamoDB 的差異

在關聯式 (SQL) 資料庫與 NoSQL 之間進行取捨

當今應用程式的要求比以往更加嚴苛。例如，一款線上遊戲剛開始可能只有少數使用者和極少量的資料。但是，如果遊戲成功了，該遊戲可能容易超過基礎資料庫管理系統的資源。Web 應用程式常擁有數百、數千，甚至數百萬名並行使用者，並且每天產生數 TB 或以上的新資料。這種應用程式的資料庫每秒必須處理成千上萬筆讀取和寫入。

Amazon DynamoDB 正適合這類工作負載。身為開發人員，您可以從小規模開始，並隨著您的應用程式變得愈來愈熱門而逐漸增加使用率。DynamoDB 可無縫擴展，能夠處理極大量的資料和使用者。

如需傳統關聯式資料庫建模以及如何針對 DynamoDB 進行調整的詳細資訊，請參閱 在 DynamoDB 中製作關聯式資料模型的最佳實務。

下表顯示關聯式資料庫管理系統 (RDBMS) 與 DynamoDB 之間的一些概要性差異。

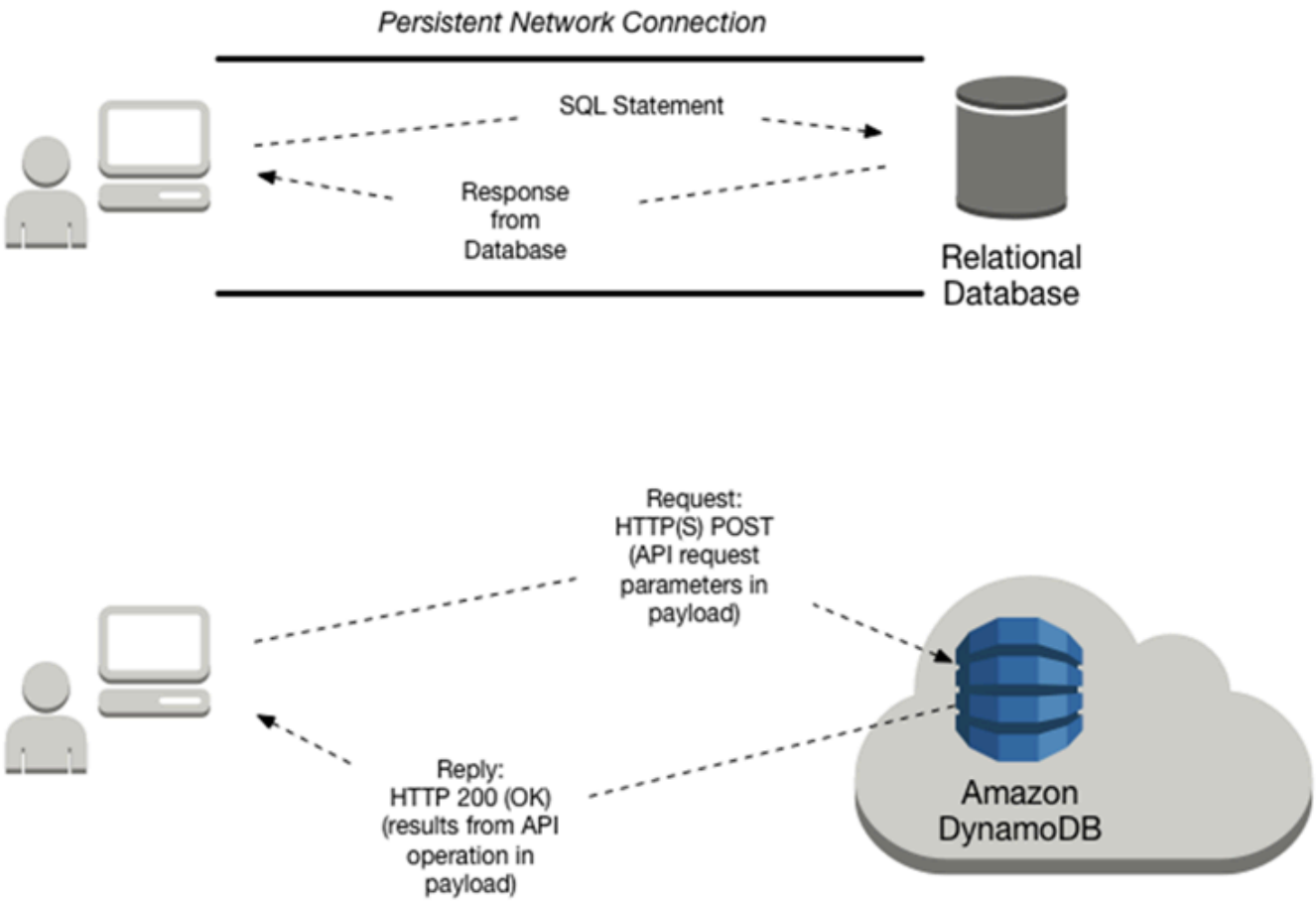
特性	關聯式資料庫管理系統 (RDBMS)	Amazon DynamoDB
最佳工作負載	臨機操作查詢、資料倉儲、OLAP (線上分析處理)。	Web 規模應用程式，包括社群網路、遊戲、媒體共用及物聯網 (IoT)。
資料模型	關聯式模型需要定義良好的結構描述，所有資料皆會標準化成資料表、資料列及資料行。此外，資料表、資料行、索引和其他資料庫元素間也都會定義所有關聯性。	DynamoDB 不具結構描述。每個資料表都必須具備一個主索引鍵，以唯一識別各個資料項目，但其他非索引鍵屬性則沒有類似的限制條件。DynamoDB 可以管理結構化或半結構化資料，包括 JSON 文件。
資料存取	SQL 是存放和擷取資料的標準。關聯式資料庫提供豐富的工具組，可簡化資料庫驅動之應用程式的開發，但這些工具全都使用 SQL。	您可以使用 AWS 管理主控台、 AWS CLI或 NoSQL WorkBench 來使用 DynamoDB，並執行臨機操作。SQL 相容查詢語言 PartiQL 讓您能在 DynamoDB 中選取、插入、更新和刪除資料。應用程式可以使用 AWS

特性	關聯式資料庫管理系統 (RDBMS)	Amazon DynamoDB
		軟體開發套件 (SDKs)，使用以物件為基礎、以文件為中心或低階介面來使用 DynamoDB。
效能	關聯式資料庫經儲存最佳化，因此效能一般取決於磁碟子系統。開發人員和資料庫管理員必須最佳化查詢、索引及資料表結構，才能達到尖峰效能。	DynamoDB 經運算最佳化，因此效能主要是基礎硬體和網路延遲的功能。作為受管服務，DynamoDB 會將您和您的應用程式與這些實作細節隔開，讓您可專心設計及建置穩固且高效能的應用程式。
擴展	使用更快的硬體可以更容易地向上擴展。資料庫資料表也能在分散式系統中橫跨多部主機，但這需要額外投資。關聯式資料庫在檔案的數目和大小方面皆具有大小上限，而對最大擴展能力有所限制。	DynamoDB 專為使用分散式硬體叢集橫向擴充而設計。這項設計可讓輸送量增加，卻不會增加延遲。客戶可指定其輸送量需求，DynamoDB 便會配置充足的資源以符合這些需求。每個資料表的項目數目沒有上限，該資料表的總大小也沒有上限。

從資料表刪除資料時，關聯式 (SQL) 資料庫與 DynamoDB 的差異

在應用程式可以存取資料庫之前，必須先進行驗證，以確保允許應用程式使用資料庫。必須對其進行授權，以便應用程式執行其具有許可的動作。

下列圖表說明用戶端與關聯式資料庫及 Amazon DynamoDB 的互動。



下表提供用戶端互動任務的詳細資訊。

特性	關聯式資料庫管理系統 (RDBMS)	Amazon DynamoDB
存取資料庫的工具	多數關聯式資料庫皆提供命令列介面 (CLI)，讓您可輸入臨機操作 SQL 陳述式並立即查看結果。	在大多數的案例中，您會撰寫應用程式程式碼。您也可以使用 AWS 管理主控台、 AWS Command Line Interface (AWS CLI) 或 NoSQL Workbench 將臨機操作請求傳送至 DynamoDB 並檢視結果。SQL 相容查詢語言 PartiQL 讓您能在 DynamoDB 中選取、插入、更新和刪除資料。

特性	關聯式資料庫管理系統 (RDBMS)	Amazon DynamoDB
連線到資料庫	應用程式會建立並維持與資料庫的網路連線。當應用程式完成時，它會終止連線。	DynamoDB 是一項 Web 服務，與該服務之間的互動是無狀態的。應用程式不需要維持耐久性網路連線。而是使用 HTTP(S) 請求及回應與 DynamoDB 互動。
身分驗證	應用程式在經過驗證前無法連線到資料庫。RDBMS 可自行執行身分驗證，也可將此任務交由主機作業系統或目錄服務執行。	每一個發送至 DynamoDB 的請求都必須附有密碼編譯簽章，此簽章會驗證該特定請求。AWS SDKs 提供建立簽章和簽署請求所需的所有邏輯。如需詳細資訊，請參閱《》中的簽署 AWS API 請求AWS 一般參考。
授權	應用程式只能執行獲得授權的動作。資料庫管理員或應用程式擁有者可使用 SQL GRANT 及 REVOKE 陳述式控制對資料庫物件 (例如資料表)、資料 (例如資料表中的資料列) 的存取，或控制發行特定 SQL 陳述式的能力。	在 DynamoDB 中，授權由 AWS Identity and Access Management (IAM) 處理。您可以撰寫 IAM 政策來授予 DynamoDB 資源 (例如資料表) 相關許可，然後允許使用者和角色使用該政策。IAM 也針對 DynamoDB 資料表中的個別資料項目提供精細的存取控制。如需詳細資訊，請參閱Amazon DynamoDB 的 Identity and Access Management。

特性	關聯式資料庫管理系統 (RDBMS)	Amazon DynamoDB
傳送請求	應用程式會針對想要執行的每項資料庫操作各發出一個 SQL 陳述式。在收到 SQL 陳述式後，RDBMS 會檢查其語法，建立執行操作的計畫，然後執行計畫。	應用程式會向 DynamoDB 傳送 HTTP(S) 請求。該請求包含要執行之 DynamoDB 操作的名稱及參數。DynamoDB 會立即執行請求。
接收回應	RDBMS 會傳回 SQL 陳述式的結果。若發生錯誤，RDBMS 會傳回錯誤狀態及訊息。	DynamoDB 會傳回含有操作結果的 HTTP(S) 回應。若發生錯誤，DynamoDB 會傳回 HTTP 錯誤狀態及訊息。

建立資料表時，關聯式 (SQL) 資料庫與 DynamoDB 之間的差異

資料表是關聯式資料庫及 Amazon DynamoDB 中的基本資料結構。關聯式資料庫管理系統 (RDBMS) 需要您在建立資料表的同時定義其結構描述。反之，DynamoDB 資料表不具結構描述；除了主索引鍵之外，您不需要在建立資料表時定義任何額外屬性或資料類型。

下一節將比較使用 SQL 建立資料表的方式，以及使用 DynamoDB 建立資料表的方式。

主題

- 使用 SQL 建立資料表
- 使用 DynamoDB 建立資料表

使用 SQL 建立資料表

在 SQL 中，您會使用 `CREATE TABLE` 陳述式建立資料表，如下列範例所示。

```
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)
);
```

此資料表的主索引鍵包含 Artist 和 SongTitle。

您必須定義資料表所有資料行、資料類型及主索引鍵。(若有需要，您之後可以使用 ALTER TABLE 陳述式變更這些定義)。

許多 SQL 實作可讓您為資料表定義儲存規格，做為 CREATE TABLE 陳述式的一部分。除非您另外指示，否則會以預設儲存設定建立資料表。在生產環境中，資料庫管理員可協助判斷最佳儲存參數。

使用 DynamoDB 建立資料表

使用 CreateTable 操作建立佈建模式資料表，並依下方所示指定參數：

```
{
    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
    }
}
```

此資料表的主索引鍵包含 Artist (分割區索引鍵) 和 SongTitle (排序索引鍵)。

您必須提供下列參數給 `CreateTable`：

- `TableName`：資料表的名稱。
- `KeySchema`：用於主索引鍵的屬性。如需詳細資訊，請參閱 資料表、項目與屬性 及 主索引鍵。
- `AttributeDefinitions`：索引鍵結構描述屬性的資料類型。
- `ProvisionedThroughput (for provisioned tables)`：此資料表所需的每秒讀取及寫入數。DynamoDB 會保留足夠的儲存和系統資源，以便始終能滿足您的輸送量需求。若有需要，您之後可以使用 `UpdateTable` 操作進行變更。因為儲存配置全由 DynamoDB 管理，因此您不需要指定資料表的儲存需求。

從關聯式 (SQL) 資料庫與 DynamoDB 取得資料表資訊的差異

您可以確認資料表是否已按照您的規格建立。關聯式資料庫中會顯示資料表的所有結構描述。Amazon DynamoDB 資料表不具結構描述，因此只會顯示主索引鍵屬性。

主題

- 取得 SQL 資料表的相關資訊
- 取得 DynamoDB 中資料表的相關資訊

取得 SQL 資料表的相關資訊

多數關聯式資料庫管理系統 (RDBMS) 可讓您說明資料表的結構，即欄、資料類型、主索引鍵定義等。SQL 沒有此操作的標準做法。但是，許多資料庫系統皆會提供 `DESCRIBE` 命令。以下是 MySQL 的範例。

```
DESCRIBE Music;
```

這會傳回您資料表的結構，內含所有資料行名稱、資料類型及大小。

```
+------------+-------------+------+-----+---------+-------+
| 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    |       |
+------------+-------------+------+-----+---------+-------+
```

此資料表的主索引鍵包含 Artist 和 SongTitle。

取得 DynamoDB 中資料表的相關資訊

DynamoDB 具有類似的 `DescribeTable` 操作。唯一的參數是資料表名稱。

```
{
    TableName : "Music"
}
```

`DescribeTable` 的回覆如下所示。

```
{
  "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 也會傳回有關資料表索引、佈建輸送量設定、約略的項目計數及其他中繼資料的資訊。

寫入資料表時，關聯式 (SQL) 資料庫與 DynamoDB 的差異

關聯式資料庫資料表含有資料的資料列。資料列由欄所組成。Amazon DynamoDB 資料表包含項目。項目由屬性組成。

本節說明如何將一個資料列 (或項目) 寫入資料表。

主題

- 將資料寫入 SQL 資料表
- 將資料寫入 DynamoDB 中的資料表

將資料寫入 SQL 資料表

關聯式資料庫中的資料表為二維資料結構，由資料列和資料行組成。某些資料庫管理系統也支援半結構化資料，通常使用原生 JSON 或 XML 資料類型。但是，實作細節會因廠商而有所不同。

在 SQL 中，您會使用 INSERT 陳述式新增列至資料表。

```
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}'
);
```

此資料表的主索引鍵包含 Artist 和 SongTitle。您必須指定這些資料行的值。

> **Note**
>
> 此範例使用 Tags 資料行存放 Music 資料表中歌曲的半結構化資料。Tags 資料行已定義為 TEXT 類型，可在 MySQL 中存放多達 65,535 個字元。

將資料寫入 DynamoDB 中的資料表

在 Amazon DynamoDB 中，您可以使用 DynamoDB API 或 PartiQL (SQL 相容查詢語言) 將項目新增至資料表。

DynamoDB API

借助 DynamoDB API，您可以使用 `PutItem` 操作將項目新增至資料表。

```
{
    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
        }
    }
}
```

此資料表的主索引鍵包含 Artist 和 SongTitle。您必須指定這些屬性的值。

以下為此 `PutItem` 範例的一些重要須知：

- DynamoDB 會使用 JSON 提供文件的原生支援。這讓 DynamoDB 適合存放半結構化資料，例如 Tags。您也可以從 JSON 文件擷取及運用資料。
- Music (音樂) 資料表除了主索引鍵 (Artist 和 SongTitle) 之外，沒有任何預先定義的屬性。
- 多數 SQL 資料庫都以交易為導向。當您發出 `INSERT` 陳述式時，資料都不會永久修改，直到您發出 `COMMIT` 陳述式為止。若使用 Amazon DynamoDB，當 DynamoDB 以 HTTP 200 狀態代碼 (OK) 回覆時，`PutItem` 操作會永久生效。

下列是一些其他的 `PutItem` 範例。

```
{
    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**
>
> 除了 PutItem 之外，DynamoDB 還支援 BatchWriteItem 操作，可讓您同時寫入多個項目。

PartiQL for DynamoDB

借助 PartiQL，您可以使用 PartiQL Insert 陳述式，使用 ExecuteStatement 操作將項目新增至資料表，

```
INSERT into Music value {
    'Artist': 'No One You Know',
    'SongTitle': 'Call Me Today',
    'AlbumTitle': 'Somewhat Famous',
    'Year' : '2015',
    'Genre' : 'Acme'
}
```

此資料表的主索引鍵包含 Artist 和 SongTitle。您必須指定這些屬性的值。

> **Note**
>
> 如需使用 Insert 和 ExecuteStatement 的程式碼範例，請參閱 適用於 DynamoDB 的 PartiQL Insert 陳述式。

讀取資料表資料時，關聯式 (SQL) 資料庫與 DynamoDB 的主要差異

使用 SQL，您可用 SELECT 陳述式從資料表擷取一或多個資料列。並可用 WHERE 子句決定傳回給您的資料。

這與使用 Amazon DynamoDB 不同，其提供下列操作來讀取資料：

- ExecuteStatement 會從資料表檢索單一或多個項目。BatchExecuteStatement 以單一操作從不同資料表檢索多個項目。這些操作都使用 PartiQL，這是一種與 SQL 相容的查詢語言。
- GetItem：從資料表擷取單一項目。因為可供直接存取項目的實體位置，所以此為讀取單一項目最有效率的方式。(DynamoDB 還會提供 BatchGetItem 操作，允許您在單一操作中執行多達 100 個 GetItem 呼叫。)
- Query：擷取具有特定分割區索引鍵的所有項目。在那些項目中，您可以將條件套用至排序索引鍵，並只擷取部分資料。Query 供您快速且有效率地存取存放資料的分割區。(如需詳細資訊，請參閱 DynamoDB 的分割區與資料分配。)
- Scan：擷取指定資料表中的所有項目。(此操作不應用於大型資料表，因為會使用大量系統資源)。

> **Note**
>
> 使用關聯式資料庫，您可用 SELECT 陳述式聯結多個資料表的資料，然後傳回結果。聯結是關聯式模型的基礎。若要確保聯結可有效率地執行，資料庫及其應用程式應持續調整其效能。DynamoDB 是不支援資料表聯結的非關聯式 NoSQL 資料庫。相反地，應用程式會一次從一個資料表讀取資料。

下列章節說明讀取資料的不同使用案例，以及如何使用關聯式資料庫和 DynamoDB 執行這些任務。

主題

- 使用主索引鍵讀取項目的差異
- 查詢資料表的差異
- 掃描資料表的差異

使用主索引鍵讀取項目的差異

資料庫常見的一種存取模式是從資料表讀取單一項目。您必須指定您想要讀取之項目的主索引鍵。

主題

- 在 SQL 中使用項目的主索引鍵讀取項目
- 在 DynamoDB 中使用項目的主索引鍵讀取項目

在 SQL 中使用項目的主索引鍵讀取項目

在 SQL 中，您會使用 SELECT 陳述式從資料表擷取資料。您可以在結果中請求一或多個資料行 (或使用 * 運算子請求所有資料行)。WHERE 子句則可決定要傳回的資料列。

下列為從 Music 資料表擷取單一資料列的 SELECT 陳述式。WHERE 子句會指定主索引鍵值。

```
SELECT *
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

您可以修改此查詢，只擷取部分資料行。

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

請注意，此資料表的主索引鍵包含 Artist 和 SongTitle。

在 DynamoDB 中使用項目的主索引鍵讀取項目

在 Amazon DynamoDB 中，您可以使用 DynamoDB API 或 PartiQL (SQL 相容查詢語言) 讀取資料表中的項目。

DynamoDB API

借助 DynamoDB API，您可以使用 PutItem 操作將項目新增至資料表。

DynamoDB 提供 GetItem 操作，可依主索引鍵擷取項目。GetItem 的效率極高，因為它可供您直接存取項目的實體位置。(如需詳細資訊，請參閱 DynamoDB 的分割區與資料分配。)

根據預設，GetItem 會傳回整個項目及其所有屬性。

```
{
    TableName: "Music",
```

```
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    }
}
```

您可以新增 ProjectionExpression 參數，以只傳回某些屬性。

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    },
    "ProjectionExpression": "AlbumTitle, Year, Price"
}
```

請注意，此資料表的主索引鍵包含 Artist 和 SongTitle。

DynamoDB GetItem 操作非常有效率。它使用主索引鍵值判斷所指項目的確切儲存位置，然後直接從該位置擷取項目。在使用主索引鍵值擷取項目的案例中，SQL SELECT 陳述式一樣具有效率。

SQL SELECT 陳述式可支援多種查詢和資料表掃描。DynamoDB 使用其 Query 和 Scan 操作提供類似功能，這些操作會在 查詢資料表的差異 和 掃描資料表的差異 中說明。

SQL SELECT 陳述式可執行資料表聯結，可讓您同時從多個資料表擷取資料。在資料庫資料表標準化且資料表間的關聯性明確時，聯結最為有效。但是，若您在單一 SELECT 陳述式中聯結太多資料表，應用程式的效能可能會受到影響。您可以使用資料庫複寫、具體化視觀表，或查詢重寫來因應這項問題。

DynamoDB 是非關聯式資料庫，不支援資料表聯結。若您正在將現有應用程式從關聯式資料庫移轉至 DynamoDB，必須將您的資料模型去標準化，才不需聯結。

PartiQL for DynamoDB

借助 PartiQL，您可以使用 PartiQL Select 陳述式，使用 ExecuteStatement 操作讀取資料表的項目。

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

請注意，此資料表的主索引鍵包含 Artist 和 SongTitle。

> **Note**
>
> PartiQL Select 陳述式也可以用來查詢或掃描 DynamoDB 資料表

如需使用 `Select` 和 `ExecuteStatement` 的程式碼範例，請參閱 適用於 DynamoDB 的 PartiQL Select 陳述式。

查詢資料表的差異

資料庫常見的另一種存取模式是根據您的查詢條件，從資料表讀取多個項目。

主題

- 使用 SQL 查詢資料表
- 在 DynamoDB 中查詢資料表

使用 SQL 查詢資料表

使用 SQL 時，`SELECT` 陳述式可讓您查詢索引鍵資料行、非索引鍵資料行或任何組合。`WHERE` 子句則可決定要傳回的資料列，如下列範例所示。

```
/* Return a single song, by primary key */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today';
```

```
/* Return all of the songs by an artist */

SELECT * FROM Music
WHERE Artist='No One You Know';
```

```
/* Return all of the songs by an artist, matching first part of title */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE 'Call%';
```

```
/* Return all of the songs by an artist, only if the price is less than 1.00 */

SELECT * FROM Music
WHERE Artist='No One You Know'
AND Price < 1.00;
```

請注意，此資料表的主索引鍵包含 Artist 和 SongTitle。

在 DynamoDB 中查詢資料表

在 Amazon DynamoDB 中，您可以使用 DynamoDB API 或 PartiQL (SQL 相容查詢語言) 查詢資料表中的項目。

DynamoDB API

若使用 Amazon DynamoDB，您可以使用 `Query` 操作以類似方式擷取資料。`Query` 操作可供您快速且有效率地存取存放資料的實體位置。如需詳細資訊，請參閱DynamoDB 的分割區與資料分配。

您可以對任何資料表或次要索引使用 `Query`。您必須為分割區索引鍵值指定相等條件，並可選擇性地為已定義的排序索引鍵屬性提供另一個條件。

`KeyConditionExpression` 參數指定您想要查詢的索引鍵值。您可以使用選用的 `FilterExpression`，在結果傳回之前從中移除特定的項目。

在 DynamoDB 中，您必須使用 `ExpressionAttributeValues` 作為表達式參數 (例如 `KeyConditionExpression` 及 `FilterExpression`) 中的預留位置。這與在關聯式資料庫中使用繫結變數類似，您會在執行時期將實際值代入 `SELECT` 陳述式。

請注意，此資料表的主索引鍵包含 Artist 和 SongTitle。

下列是一些 DynamoDB `Query` 範例。

```
// Return a single song, by primary key

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and SongTitle = :t",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call Me Today"
    }
```

```
}
```

```
// Return all of the songs by an artist

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a",
    ExpressionAttributeValues: {
        ":a": "No One You Know"
    }
}
```

```
// Return all of the songs by an artist, matching first part of title

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and begins_with(SongTitle, :t)",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call"
    }
}
```

```
// Return all of the songs by an artist, only if the price is less than 1.00

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a",
    FilterExpression: "Price < :p",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":p": 1.00
    }
}
```

> **Note**
>
> FilterExpression 會在Query讀取相符項目後套用，因此不會減少使用的讀取容量。如果可能，請將您的資料建模，讓範圍條件在排序索引鍵KeyConditionExpression上使用，以實現有效率的查詢。如需詳細資訊，請參閱在 DynamoDB 中查詢資料表。

PartiQL for DynamoDB

若使用 PartiQL，您可以藉由使用 `ExecuteStatement` 操作和 `Select` 陳述式對分割區索引鍵進行查詢。

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know'
```

以此方式使用 `SELECT` 陳述式會傳回所有與此特定 `Artist` 相關聯的歌曲。

如需使用 `Select` 和 `ExecuteStatement` 的程式碼範例，請參閱 適用於 DynamoDB 的 PartiQL Select 陳述式。

掃描資料表的差異

在 SQL 中，沒有 `SELECT` 子句的 `WHERE` 陳述式會傳回資料表中的每個資料列。在 Amazon DynamoDB 中，`Scan` 操作會執行同樣的操作。在這兩種案例中，您可以擷取所有項目或部分項目。

無論是使用 SQL 或 NoSQL 資料庫，建議您謹慎使用掃描，因為掃描會使用大量系統資源。有時候適合掃描 (例如掃描小型資料表) 或必須掃描 (例如大量匯出資料)。但一般來說，您應將應用程式設計為避免執行掃描。如需詳細資訊，請參閱在 DynamoDB 中查詢資料表。

> **Note**
>
> 執行大量匯出也會為每個分割區建立至少 1 個檔案。每個檔案的所有項目都來自該特定分割區的雜湊金鑰空間。

主題

- 使用 SQL 掃描資料表
- 掃描 DynamoDB 中的資料表

使用 SQL 掃描資料表

使用 SQL 時，您可以使用不指定 `WHERE` 子句的 `SELECT` 陳述式來掃描資料表，並擷取其中的所有資料。您可以在結果中請求一或多個資料行。或者，您可以使用萬用字元 (*) 請求所有資料行。

以下為使用 `SELECT` 陳述式的範例。

```
/* 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;
```

掃描 DynamoDB 中的資料表

在 Amazon DynamoDB 中，您可以使用 DynamoDB API 或 PartiQL (SQL 相容查詢語言) 對資料表執行掃描。

DynamoDB API

借助 DynamoDB API，可使用 `Scan` 操作，存取資料表中的每個項目或次要索引，以傳回一或多個項目和項目屬性。

```
// Return all of the data in the table
{
    TableName:  "Music"
}
```

```
// Return all of the values for Artist and Title
{
    TableName:  "Music",
    ProjectionExpression: "Artist, Title"
}
```

`Scan` 操作也提供了 `FilterExpression` 參數，您可以用它來捨棄不想要顯示在結果中的項目。`FilterExpression` 會在掃描完後並在結果傳回給您前套用。(此動作不建議與大型資料表一起使用。即使只傳回少量的符合項目，您仍須為整個 `Scan` 支付費用。)

PartiQL for DynamoDB

借助 PartiQL，您可以藉由使用 `ExecuteStatement` 操作並使用 `Select` 陳述式傳回資料表的所有內容。

```
SELECT AlbumTitle, Year, Price
FROM Music
```

請注意，此陳述式會傳回「Music」資料表中的所有項目。

如需使用 Select 和 ExecuteStatement 的程式碼範例，請參閱 適用於 DynamoDB 的 PartiQL Select 陳述式。

管理索引時，關聯式 (SQL) 資料庫與 DynamoDB 的差異

索引提供您替代的查詢模式，可加速查詢。本節會比較及比對 SQL 及 Amazon DynamoDB 中索引的建立及使用。

無論您是使用關聯式資料庫或 DynamoDB，都應明智地建立索引。每當寫入資料表時，資料表的所有索引都必須更新。在大型資料表的大量寫入環境中，這可能會使用大量系統資源。在唯讀或大部分讀取的環境中，這不算是問題。但是，您應確認應用程式確實使用了索引，而非只是占用空間。

主題

- 建立索引時，關聯式 (SQL) 資料庫與 DynamoDB 的差異
- 查詢與掃描索引時，關聯式 (SQL) 資料庫與 DynamoDB 的差異

建立索引時，關聯式 (SQL) 資料庫與 DynamoDB 的差異

比較 SQL 中的 CREATE INDEX 陳述式及 Amazon DynamoDB 中的 UpdateTable 操作。

主題

- 使用 SQL 建立索引
- 在 DynamoDB 中建立索引

使用 SQL 建立索引

在關聯式資料庫中，索引是一種資料結構，可讓您快速查詢資料表中的不同資料行。您可以使用 CREATE INDEX SQL 陳述式對現有的資料表新增索引，並指定要編製索引的資料行。在索引建立後，您可以如常地查詢資料表中的資料，但資料庫現在已可使用索引在快速尋找資料表中的指定資料列，而不用掃描整個資料表。

在您建立索引後，資料庫會為您維護。每當您修改資料表中的資料時，索引就會自動修改，以反映資料表的變更。

在 MySQL 中，您會建立如下的索引。

```
CREATE INDEX GenreAndPriceIndex
ON Music (genre, price);
```

在 DynamoDB 中建立索引

在 DynamoDB 中，您可以建立及使用次要索引來達到類似目的。

DynamoDB 中的索引與關聯式資料庫中的索引不同。在您建立次要索引時，必須指定其索引鍵屬性，即一個分割區索引鍵和一個排序索引鍵。在建立次要索引後，您便可以 Query 或 Scan 該索引，正如您對資料表所做的那樣。DynamoDB 沒有查詢最佳化程式，因此只有在您 Query 或 Scan 次要索引時才會使用此類索引。

DynamoDB 支援兩種不同的索引：

- 全域次要索引：索引的主索引鍵可以是資料表中任兩個屬性。
- 本機次要索引：索引的分割區索引鍵必須與資料表的分割區索引鍵相同。但是，排序索引鍵可以是其他任何屬性。

DynamoDB 會確認次要索引中的資料最終與其資料表一致。您可以對資料表或本機次要索引請求高度一致的 Query 或 Scan 操作。但是，全域次要索引只支援最終一致性。

您可以使用 UpdateTable 操作並指定 GlobalSecondaryIndexUpdates，對現有的資料表新增全域次要索引。

```
{
    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
```

```
                }
            }
        }
    ]
}
```

您必須提供下列參數給 UpdateTable：

- TableName：會與索引相關聯的資料表。
- AttributeDefinitions：索引之索引鍵結構描述屬性的資料類型。
- GlobalSecondaryIndexUpdates：您想要建立之索引的相關詳細資訊：
  - IndexName：索引的名稱。
  - KeySchema：用於索引主索引鍵的屬性。
  - Projection：資料表中複製到索引的屬性。在此案例中，ALL 表示會複製所有屬性。
  - ProvisionedThroughput (for provisioned tables)：此索引所需的每秒讀取及寫入數。(這和資料表的佈建輸送量設定是分開的。)

部分此操作會涉及將資料表中的資料回填到新的索引。在回填過程中，資料表仍可使用。但是，在索引的 Backfilling 屬性從 true 變更為 false 之前，索引都尚未就緒。您可以使用 DescribeTable 操作來檢視此屬性。

查詢與掃描索引時，關聯式 (SQL) 資料庫與 DynamoDB 的差異

比較在 SQL 中使用 SELECT 陳述式及在 Amazon DynamoDB 中使用 Query 及 Scan 操作查詢及掃描索引。

主題

- 在 SQL 中查詢及掃描索引
- 在 DynamoDB 中查詢及掃描索引

在 SQL 中查詢及掃描索引

在關聯式資料庫中，您不會直接使用索引。而是透過發出 SELECT 陳述式查詢資料表，然後查詢最佳化器就可使用任何索引。

查詢最佳化器是一項關聯式資料庫管理系統 (RDBMS) 元件，可評估可用的索引，並判斷是否可用其加速查詢。若索引可用來加速查詢，RDBMS 便會先存取索引，然後用它尋找資料表中的資料。

以下為一些可使用 GenreAndPriceIndex 改善效能的 SQL 陳述式。我們假設 Music 資料表具有足夠的資料，可讓查詢最佳化器決定使用此索引，而非單純地掃描整個資料表。

```
/* 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;
```

在 DynamoDB 中查詢及掃描索引

在 DynamoDB 中，您可以直接在索引上執行 `Query` 和 `Scan` 操作，與您在資料表上的操作方式相同。您可以使用 DynamoDB API 或 PartiQL (SQL 相容查詢語言) 來查詢或掃描索引。您必須同時指定 `TableName` 和 `IndexName`。

下列是關於 DynamoDB 中 GenreAndPriceIndex 的部分查詢。(此索引的索引鍵結構描述包含 Genre 和 Price)。

DynamoDB API

```
// All of the rock songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre",
    ExpressionAttributeValues: {
        ":genre": "Rock"
    },
};
```

此範例使用 `ProjectionExpression` 指出您只想要在結果中顯示部分屬性，而非所有屬性。

```
// 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"
};
```

下列是對 GenreAndPriceIndex 執行的掃描。

```
// Return all of the data in the index

{
    TableName:  "Music",
    IndexName: "GenreAndPriceIndex"
}
```

PartiQL for DynamoDB

使用 PartiQL，您可以使用 PartiQL `Select` 陳述式對索引執行查詢和掃描。

```
// 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
```

下列是對 GenreAndPriceIndex 執行的掃描。

```
// Return all of the data in the index

SELECT *
FROM Music.GenreAndPriceIndex
```

> **Note**
>
> 如需使用 Select 的程式碼範例，請參閱 適用於 DynamoDB 的 PartiQL Select 陳述式。

修改資料表資料時，關聯式 (SQL) 資料庫與 DynamoDB 的差異

SQL 語言會提供 UPDATE 陳述式來修改資料。Amazon DynamoDB 使用 UpdateItem 操作來完成類似的任務。

主題

- 修改 SQL 資料表中的資料
- 修改 DynamoDB 中資料表的資料

修改 SQL 資料表中的資料

在 SQL 中，您會使用 UPDATE 陳述式修改一或多列。SET 子句可為一或多個資料行指定新的值，WHERE 子句則可決定要修改的資料列。下列是 範例。

```
UPDATE Music
SET RecordLabel = 'Global Records'
WHERE Artist = 'No One You Know' AND SongTitle = 'Call Me Today';
```

若沒有任何資料列符合 WHERE 子句，UPDATE 陳述式便沒有任何效果。

修改 DynamoDB 中資料表的資料

在 DynamoDB 中，您可以使用 DynamoDB API 或 PartiQL (SQL 相容查詢語言) 修改單一項目。如果您想要修改多個項目，則必須使用多個操作。

DynamoDB API

若使用 DynamoDB API，您可以使用 UpdateItem 操作修改單一項目。

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
```

```
    },
    UpdateExpression: "SET RecordLabel = :label",
    ExpressionAttributeValues: {
        ":label": "Global Records"
    }
}
```

您必須指定要修改之項目的 Key 屬性，以及 UpdateExpression 以指定屬性值。UpdateItem 的運作方式類似「upsert」操作。如果項目存在資料表中，則會進行更新，如果不存在，則會新增 (插入) 新項目。

UpdateItem 支援條件式寫入，即只有在特定 ConditionExpression 評估為 true 時，操作才會成功。例如，若歌曲的價格未大於或等於 2.00，下列 UpdateItem 操作就不會執行更新。

```
{
    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 也支援原子計數器，或可增加或減少的 Number 類型屬性。原子計數器與 SQL 資料庫中的順序產生器、身分欄位或自動遞增欄位在許多方面都非常類似。

下列是 UpdateItem 操作的範例，它會初始化新屬性 (Plays) 來追蹤歌曲的播放次數。

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = :val",
    ExpressionAttributeValues: {
        ":val": 0
```

```
    },
    ReturnValues: "UPDATED_NEW"
}
```

ReturnValues 參數會設為 UPDATED_NEW，這會傳回任何更新屬性的新值。在此案例中，它會傳回 0 (零)。

只要有人播放此歌曲，我們就可以使用下列 UpdateItem 操作將 Plays 增加 1。

```
{
    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

借助 PartiQL，您可以使用 PartiQL Update 陳述式，使用 ExecuteStatement 操作修改資料表中的項目。

此資料表的主索引鍵包含 Artist 和 SongTitle。您必須指定這些屬性的值。

```
UPDATE Music
SET RecordLabel ='Global Records'
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

您也可以一次修改多個欄位，如以下範例所示。

```
UPDATE Music
SET RecordLabel = 'Global Records'
SET AwardsWon = 10
WHERE Artist ='No One You Know' AND SongTitle='Call Me Today'
```

Update 也支援原子計數器，或可增加或減少的 Number 類型屬性。原子計數器與 SQL 資料庫中的順序產生器、身分欄位或自動遞增欄位在許多方面都非常類似。

以下為 Update 陳述式的示例，用以初始化新的屬性 (Plays) 來追蹤歌曲的播放次數。

```
UPDATE Music
SET Plays = 0
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

當某人播放此歌曲時，我們即可使用下列 Update 陳述式將 Plays 增加 1。

```
UPDATE Music
SET Plays = Plays + 1
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

> **Note**
>
> 如需使用 Update 和 ExecuteStatement 的程式碼範例，請參閱 適用於 DynamoDB 的 PartiQL Update 陳述式。

從資料表刪除資料時，關聯式 (SQL) 資料庫與 DynamoDB 的差異

在 SQL 中，DELETE 陳述式會從資料表中移除一或多個資料列。Amazon DynamoDB 會使用 DeleteItem 操作一次刪除一個項目。

主題

- 從 SQL 資料表刪除資料
- 從 DynamoDB 中的資料表刪除資料

從 SQL 資料表刪除資料

在 SQL 中，您可使用 DELETE 陳述式刪除一或多個資料列。WHERE 子句則可決定您想要修改的資料列。下列是 範例。

```
DELETE FROM Music
WHERE Artist = 'The Acme Band' AND SongTitle = 'Look Out, World';
```

您可修改 WHERE 子句來刪除多個資料列。例如，您可以刪除特定演出者的所有歌曲，如下所示。

```
DELETE FROM Music WHERE Artist = 'The Acme Band'
```

從 DynamoDB 中的資料表刪除資料

在 DynamoDB 中，您可以使用 DynamoDB API 或 PartiQL (SQL 相容查詢語言) 刪除單一項目。如果您想要修改多個項目，則必須使用多個操作。

DynamoDB API

若使用 DynamoDB API，您會使用 `DeleteItem` 操作從資料表中刪除資料，一次刪除一個項目。您必須指定項目的主索引鍵值。

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    }
}
```

> **Note**
>
> 除了 `DeleteItem` 之外，Amazon DynamoDB 還支援 `BatchWriteItem` 操作，可讓您同時刪除多個項目。

`DeleteItem` 支援條件式寫入，即只有在特定 `ConditionExpression` 評估為 true 時，操作才會成功。例如，下列 `DeleteItem` 操作只會刪除具有 RecordLabel 屬性的項目。

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    },
   ConditionExpression: "attribute_exists(RecordLabel)"
}
```

PartiQL for DynamoDB

借助 PartiQL，您可以使用 `Delete` 陳述式透過 `ExecuteStatement` 操作從資料表刪除資料，一次刪除一個項目。您必須指定項目的主索引鍵值。

此資料表的主索引鍵包含 Artist 和 SongTitle。您必須指定這些屬性的值。

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks'
```

您也可以為操作指定其他條件。以下 DELETE 操作僅刪除具有超過 11 個 Awards (獎項) 的項目。

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks' AND Awards > 11
```

> **Note**
>
> 如需使用 DELETE 和 ExecuteStatement 的程式碼範例，請參閱 適用於 DynamoDB 的 PartiQL Delete 陳述式。

移除資料表時，關聯式 (SQL) 資料庫與 DynamoDB 的差異

在 SQL 中，您可使用 DROP TABLE 陳述式移除資料表。在 Amazon DynamoDB 中，您可以使用 DeleteTable 操作。

主題

- 移除 SQL 資料表
- 移除 DynamoDB 中的資料表

移除 SQL 資料表

當您不再需要某個資料表，並想要永久將它捨棄時，您會在 SQL 中使用 DROP TABLE 陳述式。

```
DROP TABLE Music;
```

資料表捨棄後，便無法復原。(某些關聯式資料庫的確可讓您復原 DROP TABLE 操作，但這屬於廠商專屬的功能，並未廣泛實作)。

移除 DynamoDB 中的資料表

在 DynamoDB 中，DeleteTable 是類似的操作。在下列範例中，資料表會永久刪除。

```
{
```

```
    TableName: "Music"
}
```

Amazon DynamoDB 學習資源與工具

您可以使用下列其他資源，協助您了解及使用 DynamoDB。

主題

- 寫程式和視覺化工具
- 規範性指引文章
- 知識中心文章
- 部落格文章、儲存庫和指南
- 資料建模和設計模式簡報
- 培訓課程

寫程式和視覺化工具

您可以使用下列程式撰寫和視覺化工具來搭配 DynamoDB 使用：

- Amazon DynamoDB 專用 NoSQL Workbench：一種統一的視覺化工具，可協助您設計、建立、查詢和管理 DynamoDB 資料表。其提供資料建模、資料視覺化和查詢開發功能。
- Dynobase：此桌面工具可讓您輕鬆查看和使用 DynamoDB 資料表、建立應用程式程式碼，以及透過即時驗證來編輯紀錄。
- DynamoDB 工具箱：由 Jeremy Daly 開發的專案，提供實用工具，協助進行資料建模及在 JavaScript 與 Node.js 中操作。
- DynamoDB Streams 處理器：此簡易工具可以簡化使用 DynamoDB Streams 的作業。

規範性指引文章

AWS 方案指引提供經過時間測試的策略、指南和模式，以協助加速您的專案。這些資源是由 AWS 技術專家和 AWS 合作夥伴的全球社群所開發，以其多年協助客戶實現業務目標的經驗為基礎。

資料建模與移轉

- DynamoDB 階層式資料模型

- 使用 DynamoDB 建立資料模型
- 使用 將 Oracle 資料庫遷移至 DynamoDB AWS DMS

全域資料表

- 使用 Amazon DynamoDB 全域資料表

無伺服器

- 使用 實作無伺服器 saga 模式 AWS Step Functions

SaaS 架構

- 透過單一控制平面管理多個 SaaS 產品的租用戶
- 在 SaaS 架構中使用 C# 和 AWS CDK 進行筒倉模型的租用戶上線

資料保護與資料移動

- 設定對 Amazon DynamoDB 的跨帳戶存取權
- 適用於 DynamoDB 的完整資料表複製選項
- AWS 的資料庫災難復原策略

Miscellaneous (其他)

- 協助在 DynamoDB 中強制執行標記

規範性指引影片逐步解說

- 使用無伺服器架構建立資料管道
- Novartis - Buying Engine：人工智慧採購入口網站
- Veritiv：啟用 Insights 以預測 AWS 資料湖上的銷售需求
- mimik：混合邊緣雲端利用 AWS 來支援邊緣微服務網格
- 使用 Amazon DynamoDB 變更資料擷取

如需 DynamoDB 的其他規範性指引文章和影片，請參閱規範性指引。

知識中心文章

AWS 知識中心文章和影片涵蓋了我們從 AWS 客戶收到的最常見問題和請求。以下是關於與 DynamoDB 相關之特定工作的最新知識中心文章：

成本最佳化

- 如何使用 Amazon DynamoDB 最佳化成本？

限流和延遲

- 如何疑難排解 Amazon DynamoDB 資料表的高延遲問題？
- 為什麼我的 DynamoDB 資料表會被限流？
- 為什麼我的隨選 DynamoDB 資料表會被限流？

分頁

- 如何在 DynamoDB 中實作分頁

交易

- 為什麼我的 `TransactWriteItems` API 調用在 DynamoDB 中失敗

疑難排解

- 如何解決 DynamoDB 自動擴展的問題？
- 如何疑難排解 DynamoDB 中的 HTTP 4XX 錯誤

如需 DynamoDB 的其他文章和影片，請參閱知識中心文章。

部落格文章、儲存庫和指南

除了 DynamoDB 開發人員指南之外，還有許多有用的資源可協助使用 DynamoDB。以下是使用 DynamoDB 的精選部落格文章、儲存庫和指南：

- AWS 各種 AWS SDK 語言的 DynamoDB 程式碼範例儲存庫：Node.js、Java、Python、.Net、Go 和 Rust。

- DynamoDB 說明書：由 Alex DeBrie 提供的全面指南，教導使用 DynamoDB 進行資料模型的策略導向方法。
- DynamoDB 指南：由 Alex DeBrie 提供的入門指南，逐步引導 DynamoDB NoSQL 資料庫的基本概念和進階功能。
- 如何透過 20 個簡單步驟從 RDBMS 轉換至 DynamoDB：Jeremy Daly 提供的學習資料建模實用步驟清單。
- DynamoDB JavaScript DocumentClient 速查表：此速查表可協助您在 Node.js 或 JavaScript 環境中開始使用 DynamoDB 建置應用程式。
- DynamoDB 核心概念影片：此播放清單涵蓋許多 DynamoDB 的核心概念。

資料建模和設計模式簡報

您可以使用以下關於資料建模和設計模式的資源，以協助您充分利用 DynamoDB：

- AWS re：Invent 2019：使用 DynamoDB 建立資料模型
  - 觀看 Alex DeBrie 的演講，開始認識 DynamoDB 資料建模的原則。
- AWS re：Invent 2020：DynamoDB 的資料建模 – 第 1 部分
- AWS re：Invent 2020：DynamoDB 的資料建模 – 第 2 部分
- AWS re：Invent 2017：進階設計模式
- AWS re：Invent 2018：進階設計模式
- AWS re：Invent 2019：進階設計模式
  - Jeremy Daly 分享了他在本次會議中提到的 12 個重要結論。
- AWS re:Invent 2020：DynamoDB 進階設計模式 (第 1 部分)
- AWS re：Invent 2020：DynamoDB 進階設計模式 – 第 2 部分
- Twitch 上的 DynamoDB 辦公時間

> **Note**
>
> 每個會議涵蓋不同的使用案例和範例。

培訓課程

有許多不同的培訓課程和教育選項，可讓您進一步了解 DynamoDB。以下是一些目前的範例：

- 使用 Amazon DynamoDB 開發 – 由 設計 AWS ，可將您從初學者帶到使用 Amazon DynamoDB 資料建模開發真實世界應用程式的專家。
- DynamoDB 深入課程：Pluralsight 課程。
- Amazon DynamoDB：建置 NoSQL 資料庫驅動的應用程式 – 來自 edX 上託管 AWS 的訓練和認證團隊的課程。

DynamoDB 讀取和寫入

DynamoDB 讀取和寫入是指從資料表擷取資料 (讀取) 並在資料表中插入、更新或刪除資料 (寫入) 的操作。當您使用 DynamoDB 時，請務必了解讀取和寫入的概念，因為這些概念會直接影響應用程式的效能和成本。

本主題提供有關適用於 DynamoDB 的不同讀取一致性類型的詳細資訊。本主題也說明您可能執行的不同讀取和寫入操作的單位耗用量。

主題

- DynamoDB 讀取一致性
- DynamoDB 讀取和寫入操作

DynamoDB 讀取一致性

Amazon DynamoDB 會從資料表、本機次要索引 (LSI)、全域次要索引 (GSI) 和串流讀取資料。如需詳細資訊，請參閱Amazon DynamoDB 的核心元件。資料表和 LSI 都提供兩個讀取一致性選項：最終一致 (預設) 和高度一致性讀取。GSI 和串流的所有讀取都是最終一致讀取。

當您的應用程式將資料寫入 DynamoDB 資料表並收到 HTTP 200 回應 (OK) 時，代表已成功寫入並會永久保存。DynamoDB 提供專供讀取隔離，可確保讀取操作一律傳回某個項目已確認的值。讀取將永遠不會從最終沒有成功的寫入中呈現對項目的視圖。專供讀取隔離不會防止在讀取操作後立即修改項目。

最終一致讀取

最終一致是所有讀取操作的預設讀取一致性模式。當向 DynamoDB 資料表或索引發出最終一致讀取時，回應可能無法反映出最近完成的寫入操作結果。如果您在短時間後重複讀取請求，應該最終會傳回最近的資料作為回應。最終一致讀取支援資料表、本機次要索引和全域次要索引。另外請注意，從 DynamoDB 串流的讀取也是最終一致讀取。

最終一致讀取的成本是高度一致性讀取的一半。如需詳細資訊，請參閱 Amazon DynamoDB 定價。

高度一致性讀取

讀取操作如 `GetItem`、`Query` 與 `Scan` 提供 `ConsistentRead` 參數。當您將 `ConsistentRead` 設定為是時，DynamoDB 會傳回具有最新資料的回應，以反映所有先前寫入操作均更新成功。高度一致性讀取僅支援資料表和本機次要索引。全域次要索引或 DynamoDB 串流不支援高度一致性讀取。

全域資料表讀取一致性

DynamoDB 也支援用於多重作用中和多區域複寫的全域資料表。全域資料表由不同 AWS 區域中的多個複本資料表組成。對任何複本資料表中任何項目所做的任何變更，都會複寫到相同全域資料表中的所有其他複本，通常一秒內即可完成，並且是最終一致。如需詳細資訊，請參閱一致性模式。

DynamoDB 讀取和寫入操作

DynamoDB 讀取操作可讓您透過指定分割區索引鍵值，以及選擇性的排序索引鍵值，從資料表擷取一或多個項目。使用 DynamoDB 寫入操作，您可以在資料表中插入、更新或刪除項目。本主題說明這兩個操作的容量單位耗用量。

主題

- 讀取的容量單位耗用量
- 寫入操作的容量單位耗用量

讀取的容量單位耗用量

DynamoDB 讀取請求可以是高度一致的、最終一致的，也可以是交易式。

- 一個項目最多 4 KB 的高度一致性讀取請求需要一個讀取單位。
- 一個項目最多 4 KB 的最終一致讀取請求需要一半讀取單位。
- 一個項目最多 4 KB 的交易讀取請求需要兩個讀取單位。

若要進一步了解 DynamoDB 讀取一致性模式，請參閱 DynamoDB 讀取一致性。

讀取的項目大小會向上四捨五入到下一個 4 KB 倍數。例如，讀取 3,500 個位元組的項目，將會使用與讀取 4 KB 項目相同的輸送量。

如果您需要讀取大於 4KB 的項目，DynamoDB 需要額外的讀取請求單位。所需的讀取容量單位總數取決於項目大小，以及您需要最終一致或高度一致性讀取。例如，如果您的項目大小為 8 KB，則需要 2 個讀取單位來維持一個高度一致性讀取。如果您選擇最終一致讀取，則需要 1 個讀取單位，對於交易讀取請求則需要 4 個讀取單位。

以下說明 DynamoDB 讀取操作如何使用讀取單位：

- GetItem：讀取資料表中的單一項目。若要求出 `GetItem` 會使用的讀取單位數目，請取得項目的大小，然後將其向上四捨五入到下一個 4 KB 界限。如果您已指定高度一致性讀取，則這是所需的容量單位數目。如果是最終一致讀取 (預設)，請將此數字除以二。

  例如，如果您讀取的項目大小為 3.5 KB，則 DynamoDB 會將項目大小捨入為 4 KB。如果您讀取 10 KB 項目，則 DynamoDB 會將項目大小捨入為 12 KB。

- BatchGetItem：從一或多個資料表讀取最多 100 個項目。DynamoDB 會將批次中的每個項目作為個別 `GetItem` 請求處理。因此 DynamoDB 會先將每個項目的大小，向上四捨五入至下一個 1 KB 界限，然後計算總大小。結果不一定與所有項目的總大小相同。例如，如果 `BatchGetItem` 讀取兩個大小為 1.5 KB 和 6.5 KB 的項目，DynamoDB 會將大小計算為 12 KB (4 KB + 8 KB)。DynamoDB 不會將大小計算為 8 KB (1.5 KB + 6.5 KB)。

- Query：讀取具有相同分割區索引鍵值的多個項目。傳回的所有項目都會視為單一讀取操作，而 DynamoDB 會計算所有項目的總大小。DynamoDB 接著會將該大小四捨五入到下一個 4 KB 邊界。例如，假設您的查詢傳回 10 個項目，其合併大小是 40.8 KB。DynamoDB 會將操作的項目大小四捨五入為 44 KB。如果查詢傳回各為 64 位元組的 1500 個項目，則累積大小為 96 KB。

- Scan：讀取資料表中的所有項目。DynamoDB 會考量所評估項目的大小，而非掃描所傳回項目的大小。如需 Scan 操作的詳細資訊，請參閱 掃描 DynamoDB 中的資料表。

> **Important**
>
> 如果您對某個不存在的項目執行讀取操作，DynamoDB 仍會耗用讀取輸送量，如上所述。對於 `Query`/`Scan` 操作，即使沒有資料，您仍需根據讀取一致性和搜尋來提供請求的分割區數量支付額外的讀取輸送量費用。

針對傳回項目的任何操作，您可請求要擷取的部分屬性。不過，這樣做不會影響項目大小計算。此外，`Query` 和 `Scan` 也可以傳回項目計數，而非屬性值。取得項目計數會使用相同數量的讀取單位，而且會進行相同的項目大小計算，這是因為 DynamoDB 必須讀取每個項目，才能遞增計數。

寫入操作的容量單位耗用量

一個寫入單位代表一個大小上限為 1 KB 的項目寫入。如果您需要寫入大於 1 KB 的項目，DynamoDB 需要使用額外的寫入單位。交易寫入請求需要 2 個寫入單位才能執行一個大小上限 1 KB 的項目的寫入。所需的寫入請求單位總數取決於項目大小。例如，如果您的項目大小為 2 KB，您需要 2 個寫入單位來維持一個寫入請求，如果是交易寫入請求，則需要 4 個寫入單位。

寫入的項目大小會向上四捨五入到下一個 1 KB 倍數。例如，寫入 500 個位元組的項目，將會使用與寫入 1 KB 項目相同的輸送量。

以下內容說明 DynamoDB 寫入操作如何使用寫入單位：

- PutItem：將單一項目寫入資料表。如果資料表中已有具有相同主索引鍵的項目，則操作會取代該項目。若要計算佈建輸送量使用，則有關的項目大小是兩者中較大的一個。
- UpdateItem：修改資料表中的單一項目。DynamoDB 會考量項目在更新前後所顯示的大小。使用的佈建輸送量會反映這些項目大小中較大的一個。即使您只更新一小部分項目的屬性，`UpdateItem` 還是會使用完整數量的佈建輸送量 (「之前」和「之後」項目大小中較大的一個)。
- DeleteItem：從資料表中移除單一項目。佈建輸送量使用是根據所刪除項目的大小。
- BatchWriteItem：最多可將 25 個項目寫入一或多個資料表。DynamoDB 會將批次中的每個項目作為個別 `PutItem` 或 `DeleteItem` 請求處理 (不支援更新)。因此 DynamoDB 會先將每個項目的大小，向上四捨五入至下一個 1 KB 界限，然後計算總大小。結果不一定與所有項目的總大小相同。例如，如果 `BatchWriteItem` 寫入兩個大小為 500 位元組和 3.5 KB 的項目，DynamoDB 會將大小計算為 5 KB (1 KB + 4 KB)。DynamoDB 不會將大小計算為 4 KB (500 位元組 + 3.5 KB)。

針對 `PutItem`、`UpdateItem` 和 `DeleteItem` 操作，DynamoDB 會將項目大小捨入至下一個 1 KB。例如，如果您放置或刪除 1.6 KB 項目，則 DynamoDB 會將項目向上四捨五入到 2 KB。

`PutItem`、`UpdateItem` 和 `DeleteItem` 允許條件式寫入，其中您可以指定必須評估為 true，操作才會繼續的表達式。如果表達式計算結果為 false，則 DynamoDB 仍然會從資料表使用寫入容量單位：使用的寫入容量單位數量取決於項目的大小。此項目可以是資料表中的現有項目，或您嘗試建立或更新的新項目。例如，假設現有項目為 300 KB。您嘗試建立或更新的新項目為 310 KB。新項目使用的寫入容量單位將為 310 KB。

DynamoDB 輸送容量

本節說明 DynamoDB 資料表可用的兩種輸送量模式，並提供為應用程式選擇合適容量模式時的考量重點。資料表的輸送量模式決定其容量管理方式。輸送量模式也會決定資料表讀取與寫入操作的計費方式。在 Amazon DynamoDB 中，您可為資料表選擇隨需模式或佈建模式，以符合不同的工作負載需求。

主題

- 隨需模式
- 佈建模式
- DynamoDB 隨需容量模式
- DynamoDB 佈建容量模式
- 認識 DynamoDB 暖輸送量
- DynamoDB 高載容量與調適型容量
- 切換 DynamoDB 容量模式時的注意事項

隨需模式

Amazon DynamoDB 隨需模式是一種無伺服器輸送量選項，可簡化資料庫管理，並自動擴展以支援最嚴苛的應用程式需求。DynamoDB 隨需模式讓您能建立資料表，而無需擔心容量規劃、使用量監控或擴展政策設定。DynamoDB 隨需模式提供按請求付費的定價機制，讓您僅需為實際用量付費。若為隨需模式資料表，不需要指定您預期應用程式將進行的讀取和寫入輸送量。

隨需模式是大多數 DynamoDB 工作負載的預設和建議輸送量選項。DynamoDB 會自動管理輸送量與擴展機制，以支援從小型工作負載擴展至每秒數百萬次請求。您可依需求讀寫 DynamoDB 資料表，無需手動管理輸送容量。如需詳細資訊，請參閱DynamoDB 隨需容量模式。

佈建模式

使用佈建模式時，您必須指定應用程式所需的每秒讀取與寫入次數。系統會依您佈建的每小時讀取與寫入容量收費，而非根據實際用量。這有助於您控制 DynamoDB 的使用，以維持或低於已定義的請求率，藉此獲得可預測的成本。

若您的工作負載穩定且成長可預期，並能可靠預測應用程式容量需求，則可選擇佈建容量模式。如需詳細資訊，請參閱DynamoDB 佈建容量模式。

DynamoDB 隨需容量模式

Amazon DynamoDB 隨需模式提供真正的無伺服器資料庫體驗，可自動擴展以滿足最嚴苛的工作負載，無需容量規劃。隨需模式簡化設定流程、免除容量管理與監控，並提供快速自動擴展。採用按需付費 (pay-per-request) 定價時，您無需擔心閒置容量，因為只需為實際使用的輸送量付費。系統會根據每個讀取或寫入請求向您收費，因此成本將直接反映實際使用量。

當您選擇隨需模式，DynamoDB 會在您的工作負載上升或下降到任何先前曾達到的流量程度時，立即因應您的工作負載。當工作負載的流量達到新峰值時，DynamoDB 會自動擴展以滿足更高的輸送量需求。隨需模式是預設且建議使用的輸送量選項，因為它簡化了現代無伺服器應用程式的建置，使應用程式能從小規模起步，並可擴展至每秒數百萬次請求。當隨需資料表完成橫向擴充後，未來即可立即再次達到相同輸送量，且不會受到限流。若資料表未產生任何流量，使用隨需模式時將不會產生輸送量費用。如需隨需模式的擴展屬性詳細資訊，請參閱 初始輸送量與擴展特性。

採用隨需模式的資料表可提供與 DynamoDB 佈建模式相同的個位數毫秒級延遲、服務水準協議 (SLA) 及安全性。

> **Note**
>
> 根據預設，DynamoDB 會防止意外或失控的資源耗用。若要將帳戶中所有資料表的讀取與寫入輸送量上限擴展至超過 40,000，您可以申請增加該配額。超出預設資料表輸送量配額的請求將受到限流。如需詳細資訊，請參閱輸送量預設配額。

此外，您也可以為個別隨需資料表及全域次要索引設定每秒最大讀取、寫入 (或兩者) 輸送量上限。透過設定輸送量，您可以控制資料表層級的用量與成本範圍，防止資源耗用意外激增，並避免過度使用以維持可預期的成本管理。超出資料表輸送量上限的請求將受到限流。您可依應用程式需求，隨時調整資料表的最大輸送量設定。如需詳細資訊，請參閱DynamoDB 隨需資料表的最大輸送量。

若要開始使用，請建立或更新資料表，並將其設定為隨需模式。如需詳細資訊，請參閱DynamoDB 資料表上的基本操作。

您可在 24 小時滾動期間內，最多將資料表從佈建容量模式切換至隨需模式四次。您可隨時將資料表從隨需模式切換回佈建容量模式。

如需了解如何在讀取與寫入容量模式間切換，請參閱 切換 DynamoDB 容量模式時的注意事項。如需隨需資料表配額的詳細資訊，請參閱 讀取/寫入輸送量。

主題

- 讀取請求單位與寫入請求單位
- 初始輸送量與擴展特性
- DynamoDB 隨需資料表的最大輸送量

讀取請求單位與寫入請求單位

DynamoDB 會根據應用程式在資料表上執行的讀取與寫入作業，依讀取請求單位與寫入請求單位計費。

一個讀取請求單位代表：對於大小上限為 4 KB 的項目，每秒可執行一次高度一致性讀取，或兩次最終一致讀取。如需有關 DynamoDB 讀取一致性模式的詳細資訊，請參閱 DynamoDB 讀取一致性。

一個寫入請求單位代表：對於大小上限為 1 KB 的項目，每秒可執行一次寫入作業。

如需讀取與寫入單位消耗方式的詳細資訊，請參閱 DynamoDB 讀取和寫入操作。

初始輸送量與擴展特性

DynamoDB 資料表會自動使用隨需容量模式以因應您應用程式的流量。新建立的隨需資料表可支援每秒最高 4,000 次寫入與 12,000 次讀取。隨需容量模式會立即因應，最高達到資料表峰值流量的兩倍。例如，假設應用程式的流量模式介於每秒 25,000 到 50,000 高度一致性讀取之間；50,000 次每秒讀取為先前的流量峰值。隨需容量模式可立即支援每秒高達 100,000 次的持續讀取流量。若應用程式能持續每秒 100,000 次讀取，該峰值將成為新的基準峰值。該基準峰值可讓後續流量達到每秒最高 200,000 次讀取。

若工作負載在資料表上的流量超過先前峰值兩倍，DynamoDB 會隨流量成長自動配置更多容量，以確保不發生限流。此容量配置可確保工作負載不會發生限流情況。但是，如果在 30 分鐘之內超過先前峰值的兩倍以上，還是會發生限流降速。例如，若應用程式的流量介於每秒 25,000 至 50,000 高度一致性讀取之間，則 50,000 次每秒讀取為已達到的峰值。建議您在每秒超過 100,000 次讀取前，先預熱資料表，或將流量提升分散於至少 30 分鐘內進行。如需有關資料表預熱的詳細資訊，請參閱 認識 DynamoDB 暖輸送量。

若工作負載的峰值流量維持在先前峰值的兩倍以內，DynamoDB 將不會施加 30 分鐘的限流限制。若您的尖峰流量超過先前峰值的兩倍，請確保此流量成長發生在上次達到峰值的 30 分鐘之後。

DynamoDB 隨需資料表的最大輸送量

對於隨需資料表，您可選擇在個別資料表及其相關的全域次要索引 (GSI) 上，指定每秒最大讀取、寫入 (或兩者) 輸送量。指定最大隨需輸送量有助於將資料表層級的使用量與成本維持在可控範圍內。根據預

設，最大輸送量設定不會套用，且您的隨需輸送量速率會受限於帳戶中所有資料表合計 40,000 個資料表層級的讀寫輸送量AWS 服務配額。如有需要，您可以申請提高服務配額上限。

當您設定隨需資料表的最大輸送量時，超過設定上限的輸送量請求將會受到限流處理。您可依應用程式需求，隨時修改資料表層級的輸送量設定。

以下列出數個可受益於設定隨需資料表最大輸送量的常見使用案例：

- 輸送量成本最佳化 – 為隨需資料表設定最大輸送量，可進一步提升成本的可預測性與可管理性。此外，隨需模式提供更高彈性，可支援流量模式與預算各異的工作負載。
- 防止過度使用 – 透過設定最大輸送量，您可避免隨需資料表因非最佳化程式碼或異常程序導致讀寫耗用量意外激增。此資料表層級設定可防止組織於特定時間內耗用過多資源。
- 保護下游服務 – 客戶應用程式可能同時採用無伺服器與傳統伺服器架構。架構中的無伺服器元件可快速擴展以因應需求。然而，容量固定的下游元件可能因此不堪負荷。實施隨需資料表的最大輸送量設定，可避免大量事件傳遞至多個下游元件並產生非預期副作用。

您可為新建或現有的單一區域資料表、全域資料表及 GSI 設定隨需模式的最大輸送量。您亦可於資料表還原或從 Amazon S3 匯入資料的工作流程中設定最大輸送量。

您可透過 DynamoDB 主控台、AWS CloudFormation、 AWS CLI或 DynamoDB API 來設定隨需資料表的最大輸送量。

> **Note**
>
> 隨需資料表的最大輸送量採「盡力而為」政策，應視為目標值，而非保證的請求上限。由於高載容量，您的工作負載可能暫時超出設定的最大輸送量。在部分情況下，DynamoDB 會利用高載容量以處理超出資料表最大輸送量設定的讀寫請求。使用高載容量，未預期的讀取或寫入請求在經過限流之後可能會成功。

主題

- 使用隨需模式最大輸送量的考量事項
- 請求限流與 CloudWatch 指標

使用隨需模式最大輸送量的考量事項

當您在隨需模式中設定資料表的最大輸送量時，請注意以下事項：

- 您可針對任一隨需資料表的讀寫輸送量上限，或該表中個別的全域次要索引，微調設定以符合特定需求。
- 您可透過 Amazon CloudWatch 監控並分析 DynamoDB 資料表層級的使用指標，以判定隨需模式的適當最大輸送量設定。如需詳細資訊，請參閱DynamoDB 指標和維度。
- 當您在任一全域資料表複本上設定最大讀寫 (或兩者) 輸送量時，系統會自動將相同設定套用至所有複本資料表。全域資料表中的複本資料表與次要索引必須具有相同的寫入輸送量設定，才能確保資料正確複寫。如需詳細資訊，請參閱全域資料表的最佳實務。
- 可指定的最小讀取或寫入最大輸送量為每秒一個請求單位。
- 指定的最大輸送量必須低於任一隨需資料表或其全域次要索引的預設輸送量配額。

請求限流與 CloudWatch 指標

若您的應用程式超出隨需資料表設定的讀寫輸送量上限，DynamoDB 會開始對這些請求進行限流。當 DynamoDB 限流讀取或寫入時，會將 `ThrottlingException` 傳回至呼叫者。接著，您可視需要採取相應的行動。例如，您可提高或停用資料表的最大輸送量設定，或等待短暫間隔後再重試請求。

為簡化監控資料表或全域次要索引之最大輸送量，CloudWatch 提供以下指標：OnDemandMaxReadRequestUnits 和 OnDemandMaxWriteRequestUnits。

DynamoDB 佈建容量模式

在 DynamoDB 中建立新的佈建資料表時，必須指定其佈建輸送容量。這代表資料表可支援的讀取與寫入輸送量。系統會依您佈建的每小時讀取與寫入容量收費，而非根據實際用量。

隨著應用程式的資料與存取需求變化，您可能需要調整資料表的輸送量設定。您可以使用自動擴展來自動調整資料表的佈建容量，以回應流量的變動。DynamoDB 自動擴展會使用 Application Auto Scaling 中的擴展政策。若要在 DynamoDB 中設定自動擴展，除了目標使用率百分比外，還需設定讀寫容量的最小與最大值。Application Auto Scaling 會建立和管理 CloudWatch 警示，當指標偏離目標就會觸發擴展事件，自動擴展會監控資料表活動，並依預設閾值自動調整容量設定。當耗用容量連續兩分鐘超過設定的目標使用率時，自動擴展即會觸發。CloudWatch 警示在觸發自動擴展前，可能會有短暫數分鐘的延遲。如需詳細資訊，請參閱使用 DynamoDB Auto Scaling 功能自動管理輸送容量。

如果使用了 DynamoDB Auto Scaling 功能，則會自動調整輸送量設定來回應實際工作負載。您也可使用 UpdateTable 操作，手動調整資料表的輸送容量。例如，若您需要將現有資料存放區中的資料大量匯入新的 DynamoDB 資料表，可以選擇這麼做。您可以建立具有大型寫入輸送量設定的資料表，然後在大量資料載入完成後減少此設定。

> **Note**
>
> 根據預設，DynamoDB 會防止意外或失控的資源耗用。若要將帳戶中所有資料表的讀取與寫入輸送量上限擴展至超過 40,000，您可以申請增加該配額。超出預設資料表輸送量配額的請求將受到限流。如需詳細資訊，請參閱輸送量預設配額。

您可在 24 小時滾動期間內，最多將資料表從佈建容量模式切換至隨需模式四次。您可隨時將資料表從隨需模式切換回佈建容量模式。

如需了解如何在讀取與寫入容量模式間切換，請參閱 切換 DynamoDB 容量模式時的注意事項。

主題

- 讀取容量單位和寫入容量單位
- 選擇初始輸送量設定
- DynamoDB 自動擴展
- 使用 DynamoDB Auto Scaling 功能自動管理輸送容量
- DynamoDB 預留容量

讀取容量單位和寫入容量單位

對於佈建模式資料表，您可依容量單位指定輸送量需求。這些單位代表應用程式每秒需讀取或寫入的資料量。您稍後可以視需要修改這些設定，或讓 DynamoDB Auto Scaling 功能自動進行修改。

對於大小上限為 4 KB 的項目，一個讀取容量單位 (RCU) 代表每秒一次高度一致性讀取，或每秒兩次最終一致讀取。如需有關 DynamoDB 讀取一致性模式的詳細資訊，請參閱 DynamoDB 讀取一致性。

一個寫入容量單位 (WCU) 代表每秒可對大小上限為 1 KB 的項目執行一次寫入。如需了解不同讀取與寫入操作的詳細資訊，請參閱 DynamoDB 讀取和寫入操作。

選擇初始輸送量設定

每個應用程式在資料庫讀寫需求上各不相同。在決定 DynamoDB 資料表的初始輸送量設定時，請考慮以下事項：

- 預期的讀取與寫入請求率 – 您應預估應用程式每秒需要執行的讀取與寫入次數。
- 項目大小 – 若部分項目體積足夠小，則可使用單一容量單位進行讀取或寫入。較大的項目需要多個容量單位。透過預估資料表中項目的平均大小，您可以為資料表的佈建輸送量設定更精確的參數。

- 讀取一致性需求 – 讀取容量單位以高度一致性讀取作為基準，其消耗的資料庫資源為最終一致讀取的兩倍。您應該判斷應用程式需要高度一致性讀取，還是可以放寬這項需求並改為執行最終一致讀取。根據預設，DynamoDB 中的讀取操作為最終一致性。若有需要，您可以為這些操作請求高度一致性讀取。

例如，假設您希望每秒從資料表讀取 80 個項目。這些項目的大小為 3 KB，且您需要高度一致性讀取。在此情況下，每次讀取需消耗一個佈建的讀取容量單位。若要計算此數值，請將操作的項目大小除以 4 KB。接著，將結果四捨五入至最接近的整數，如下例所示：

- 3 KB/4 KB = 0.75，或 1 個讀取容量單位

因此，若要每秒從資料表讀取 80 個項目，請將資料表的佈建讀取輸送量設定為 80 個讀取容量單位，如下例所示：

- 每個項目 1 個讀取容量單位 × 每秒 80 個讀取 = 80 個讀取容量單位

現在假設您希望每秒向資料表寫入 100 個項目，且每個項目的大小為 512 位元組。在此情況下，每次寫入需消耗一個佈建的寫入容量單位。若要計算此數值，請將操作的項目大小除以 1 KB。接著，將結果四捨五入至最接近的整數，如下例所示：

- 512 位元組/1 KB = 0.5 或 1 個寫入容量單位

若要每秒向資料表寫入 100 個項目，請將資料表的佈建寫入輸送量設定為 100 個寫入容量單位：

- 每個項目 1 個寫入容量單位 × 每秒 100 個寫入 = 100 個寫入容量單位

DynamoDB 自動擴展

DynamoDB 自動擴展會主動管理資料表與全域次要索引的佈建輸送容量。透過自動擴展，您可以定義讀取與寫入容量單位的範圍 (上限與下限)。您也可以定義該範圍內的目標使用率百分比。DynamoDB Auto Scaling 功能會嘗試維持您的目標使用率，即使您的應用程式工作負載有所增減。

透過 DynamoDB Auto Scaling，資料表或全域次要索引可增加其佈建的讀取與寫入容量，來處理突然增加的流量，而不需要進行請求限流。當工作負載降低時，DynamoDB Auto Scaling 可降低輸送量，讓您無須支付未使用的佈建容量。

> **Note**
>
> 如果您使用 AWS 管理主控台 建立資料表或全域次要索引，預設會啟用 DynamoDB Auto Scaling。
> 您可以隨時使用 主控台、 AWS CLI或其中一個 AWS SDKs 來管理自動擴展設定。如需詳細資訊，請參閱使用 DynamoDB Auto Scaling 功能自動管理輸送容量。

使用率

使用率可協助您判斷是否超過佈建容量；若是，應降低資料表容量以節省成本。相反地，使用率也可協助您判斷是否存在佈建容量不足的情況。在此情況下，您應提高資料表容量，以防止在突發高流量期間出現請求限流的情況。如需詳細資訊，請參閱 Amazon DynamoDB 自動擴展：在任何規模下的效能與成本最佳化。

若您使用 DynamoDB 自動擴展功能，則也必須設定目標使用率百分比。自動擴展會依據此百分比作為目標，向上或向下調整容量。建議將目標使用率設定為 70%。如需詳細資訊，請參閱使用 DynamoDB Auto Scaling 功能自動管理輸送容量。

使用 DynamoDB Auto Scaling 功能自動管理輸送容量

許多資料庫工作負載本來就具週期性且難以事先預測。例如，假設有一個社交聯網應用程式，其中大部分使用者會在日間活動。資料庫必須能夠處理日間活動，但夜間則不需要同樣多的輸送量。又例如，請設想一下正被快速採用的新行動遊戲應用程式。如果遊戲變得太熱門，可能會超過可用的資料庫資源，而導致效能變慢及客戶不滿意。這類工作負載通常需要手動介入來擴展或縮減資料庫資源，以回應不斷改變的用量。

Amazon DynamoDB Auto Scaling 使用 AWS Application Auto Scaling 服務代表您動態調整佈建輸送量容量，以回應實際的流量模式。這使資料表或全域次要索引 (GSI) 能增加其佈建讀寫容量，在流量突增時處理請求而不被限流。當工作負載降低時，Application Auto Scaling 可降低輸送量，讓您無須為未使用的佈建容量付費。

> **Note**
>
> 如果您使用 AWS 管理主控台 建立資料表或全域次要索引，預設會啟用 DynamoDB Auto Scaling。您可隨時修改自動擴展設定。如需詳細資訊，請參閱AWS 管理主控台 搭配 DynamoDB Auto Scaling 使用。

刪除資料表或全域資料表複本時，任何相關聯的可擴展目標、擴展政策或 CloudWatch 警示都不會跟著自動刪除。

使用 Application Auto Scaling，您就可以為資料表或全域次要索引建立擴展政策。調整規模政策可指定您要擴展讀取容量或寫入容量 (或兩者)，也可為資料表或索引指定最大與最小佈建容量單位設定。

調整規模政策也包含目標使用率：即在某個時間點耗用的佈建輸送量百分比。Application Auto Scaling 使用目標追蹤演算法，向上或向下調整資料表 (或索引) 的佈建輸送量，以此回應實際的工作負載，讓實際容量使用率保持或接近目標使用率。

DynamoDB 會以一分鐘為週期，輸出耗用的佈建輸送量。當耗用容量連續兩分鐘超過設定的目標使用率時，自動擴展即會觸發。CloudWatch 警示在觸發自動擴展前，可能會有短暫數分鐘的延遲。此延遲可確保 CloudWatch 指標的準確評估。若耗用輸送量峰值間隔超過一分鐘，自動擴展可能不會觸發。同理，當連續 15 個資料點低於目標使用率時，會觸發縮減規模事件。無論哪種情況，自動擴展觸發後，會調用 UpdateTable API。接著，系統可能需要數分鐘時間來更新資料表或索引的佈建容量。在此期間，所有超出資料表先前佈建容量的請求都會受到限流。

Important

您無法調整資料點數目以觸發底層警示 (但此數值未來可能會變更)。

您可將自動擴展目標使用率值設定在 20% 至 90% 之間，做為您的讀寫容量。

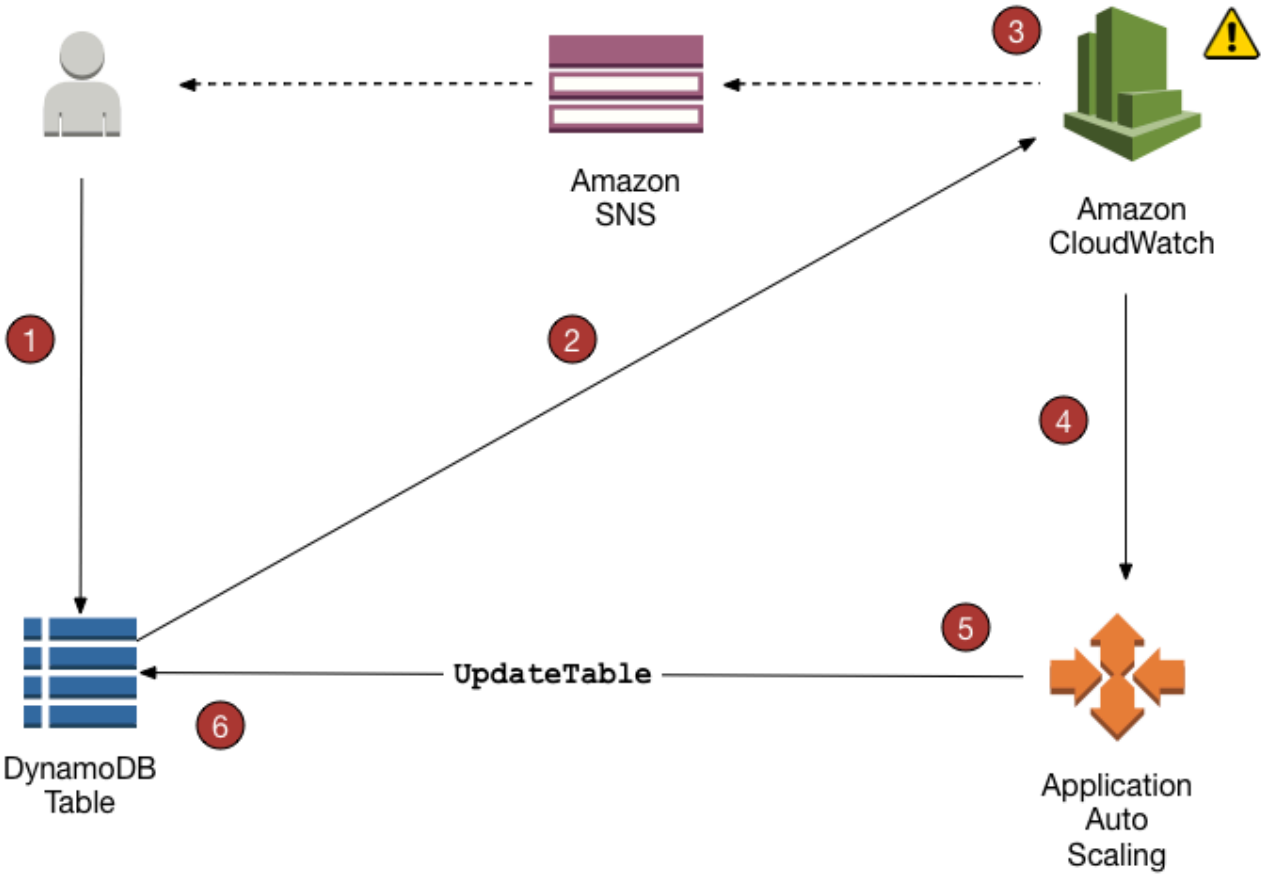
Note

除了資料表外，DynamoDB Auto Scaling 功能也支援全域次要索引。每個全域次要索引都有自己的佈建輸送容量，與其基礎資料表的佈建輸送容量無關。當您為全域次要索引建立擴展政策時，Application Auto Scaling 會調整索引的佈建輸送量設定，以確保其實際使用率保持或接近您想要的使用比率。

DynamoDB Auto Scaling 功能的運作方式

> **Note**
>
> 若要快速開始使用 DynamoDB Auto Scaling 功能，請參閱 AWS 管理主控台 搭配 DynamoDB Auto Scaling 使用。

下圖提供 DynamoDB Auto Scaling 功能如何管理資料表輸送容量的高層級概觀。



下列步驟摘要說明上圖所示的自動擴展程序：

1. 您可以為 DynamoDB 資料表建立 Application Auto Scaling 政策。
2. DynamoDB 會將耗用的容量指標發布至 Amazon CloudWatch。
3. 若資料表的使用容量持續一段特定時間都超過目標使用率 (或低於目標)，Amazon CloudWatch 會觸發警示。您可以在主控台檢視警示，並使用 Amazon Simple Notification Service (Amazon SNS) 接收通知。

4. CloudWatch 警示會調用 Application Auto Scaling 來評估擴展政策。
5. Application Auto Scaling 發出 `UpdateTable` 請求來調整資料表的佈建輸送量。
6. DynamoDB 處理 `UpdateTable` 請求，並動態增加 (或減少) 資料表的佈建輸送容量，以便接近您的目標使用率。

若要了解 DynamoDB Auto Scaling 功能的運作方式，請假設您有一個名為 `ProductCatalog` 的資料表。該資料表不常大量載入資料，因此不會產生太多寫入活動。不過，它會發生大量讀取活動，並會隨著時間變化。透過監控 `ProductCatalog` 的 Amazon CloudWatch 指標，您判斷出資料表需要 1,200 個讀取容量單位 (以避免 DynamoDB 在活動尖峰時限流讀取請求)。您也判斷出當讀取流量在最低點時，`ProductCatalog` 至少需要 150 個讀取容量單位。如需防止限流的詳細資訊，請參閱 在 Amazon DynamoDB 中針對限流問題進行疑難排解。

在 150 到 1,200 個讀取容量單位範圍內，您決定 `ProductCatalog` 資料表的適當目標使用率為 70%。目標使用率是以使用容量單位與佈建容量單位的比率，以百分比表示。Application Auto Scaling 使用其目標追蹤演算法，確保視需要調整 `ProductCatalog` 的佈建讀取容量，讓使用率維持在或接近 70%。

> **Note**
>
> 只有在實際工作負載持續幾分鐘保持很高 (或很低) 的狀態時，DynamoDB Auto Scaling 功能才會修改佈建輸送量設定。Application Auto Scaling 目標追蹤演算法會設法長期將目標使用率保持在或接近您選擇的數值。
> 資料表的內建高載容量可應付短期遽增的活動。如需詳細資訊，請參閱高載容量。

若要為 `ProductCatalog` 資料表啟用 DynamoDB Auto Scaling 功能，您可以建立擴展政策。此政策會指定以下內容：

- 要管理的資料表或全域次要索引
- 要管理的容量類型 (讀取容量或寫入容量)
- 佈建輸送量設定的上限與下限
- 您的目標使用率

當您建立擴展政策時，Application Auto Scaling 會代您建立一組 Amazon CloudWatch 警示。每組警示皆代表您佈建輸送量設定的上限與下限。當資料表的實際使用率持續一段時間偏離您的目標使用率時，就會觸發這些 CloudWatch 警示。

當其中一個 CloudWatch 警示觸發時，Amazon SNS 會傳送通知給您 (若已啟用)。然後 CloudWatch 警示會調用 Application Auto Scaling，再由其通知 DynamoDB 適當地將 `ProductCatalog` 資料表的佈建容量調高或調低。

在擴展事件期間，每個記錄的組態項目 AWS Config 都會收費。當發生擴展事件時，系統會針對每個讀取與寫入自動擴展事件建立四個 CloudWatch 警示：ProvisionedCapacity 警示 (ProvisionedCapacityLow、ProvisionedCapacityHigh) 以及 ConsumedCapacity 警示 (AlarmHigh、AlarmLow)。這樣總共會產生八個警示。因此， 會為每個擴展事件 AWS Config 記錄八個組態項目。

> **Note**
>
> 您也可以設定 DynamoDB 擴展的排程，使其在特定時間執行。在此了解基本步驟。

使用須知

開始使用 DynamoDB Auto Scaling 功能之前，您應該注意下列事項：

- DynamoDB Auto Scaling 功能可根據您的自動擴展政策，視需要經常增加讀取容量或寫入容量。所有 DynamoDB 配額仍然有效，如 Amazon DynamoDB 中的配額 所述。
- DynamoDB Auto Scaling 功能不會阻止您手動修改佈建輸送量設定。這些手動調整不會影響與 DynamoDB Auto Scaling 相關的任何現有 CloudWatch 警示。
- 如果您為具有一或多個全域次要索引的資料表啟用 DynamoDB Auto Scaling，強烈建議您也將自動擴展統一套用至這些索引。這將有助於確保改善資料表的寫入和讀取效能，並有助於避免限流。您可以在 AWS 管理主控台中選取 Apply same settings to global secondary indexes (將相同的設定套用至全域次要索引) 來啟用自動擴展的功能。如需詳細資訊，請參閱在現有資料表上啟用 DynamoDB Auto Scaling 功能。
- 刪除資料表或全域資料表複本時，任何相關聯的可擴展目標、擴展政策或 CloudWatch 警示都不會跟著自動刪除。
- 為現有資料表建立 GSI 時，不會為 GSI 啟用 Auto Scaling。建置 GSI 時，您必須手動管理容量。GSI 上的回填完成並達到活動狀態後，Auto Scaling 的操作將恢復正常。

AWS 管理主控台 搭配 DynamoDB Auto Scaling 使用

當您使用 AWS 管理主控台 建立新的資料表時，預設會為該資料表啟用 Amazon DynamoDB 自動擴展。您也可以使用主控台，啟用現有資料表的自動擴展、修改自動擴展設定或停用自動擴展。

> **Note**
>
> 如需設定向內擴展和向外擴展冷卻時間等更進階的功能，請使用 AWS Command Line Interface (AWS CLI) 來管理 DynamoDB 自動擴展。如需詳細資訊，請參閱使用 AWS CLI 管理 DynamoDB 自動擴展。

主題

- 開始之前：授予 DynamoDB Auto Scaling 功能的使用者許可
- 建立啟用 Auto Scaling 的新資料表
- 在現有資料表上啟用 DynamoDB Auto Scaling 功能
- 在主控台上檢視自動擴展活動
- 修改或停用 DynamoDB Auto Scaling 設定

開始之前：授予 DynamoDB Auto Scaling 功能的使用者許可

在 AWS Identity and Access Management (IAM) 中， AWS 受管政策`DynamoDBFullAccess`提供使用 DynamoDB 主控台所需的許可。不過，如需 DynamoDB Auto Scaling 功能，則使用者需要額外的權限。

> **Important**
>
> 須有 `application-autoscaling:*` 許可才能刪除已啟用自動擴展功能的資料表。 AWS 受管政策`DynamoDBFullAccess`包含這類許可。

若要為使用者設定 DynamoDB 主控台存取和 DynamoDB Auto Scaling，請建立角色，接著將 AmazonDynamoDBFullAccess 政策新增至該角色。然後將角色指派給使用者。

建立啟用 Auto Scaling 的新資料表

> **Note**
>
> DynamoDB Auto Scaling 需要有服務連結角色 (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`)，其會代表您執行自動擴展動作。系統會自動建立此角色。如需詳細資訊，請參閱《Application Auto Scaling 使用者指南》中的適用於 Application Auto Scaling 的服務連結角色。

在啟用 Auto Scaling 的情況下建立新資料表

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 選擇 Create Table (建立資料表)。
3. 在建立資料表頁面上，輸入資料表名稱和主索引鍵詳細資訊。
4. 若選擇預設設定，系統會自動在新資料表中啟用自動擴展。

   否則，請選擇自訂設定，然後依下列步驟設定資料表的自訂參數：

   a. 在資料表類別中，請維持 DynamoDB Standard 的預設選項。
   b. 在讀取/寫入容量設定中，維持佈建為預設選項，然後依下列步驟操作：

      i. 在讀取容量中，請確認自動擴展已設定為開啟。
      ii. 在寫入容量中，請確定自動擴展已設定為開啟。
      iii. 在讀取容量與寫入容量中，請為資料表設定所需的擴展策略，並可選擇性地套用至所有全域次要索引。

      - 容量單位下限：輸入自動擴展範圍的下限。
      - 容量單位上限：輸入自動擴展範圍的上限。
      - 目標使用率：輸入資料表的目標使用率百分比。

   > **Note**
   >
   > 如果您為新資料表建立全域次要索引，則建立時索引的容量將與基礎資料表的容量相同。建立資料表後，您可以在資料表的設定中變更索引的容量。

5. 選擇 Create Table (建立資料表)。系統會根據您指定的自動擴展參數建立資料表。

在現有資料表上啟用 DynamoDB Auto Scaling 功能

> **Note**
>
> DynamoDB Auto Scaling 需要有服務連結角色 (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`)，其會代表您執行自動擴展動作。系統會自動建立此角色。如需詳細資訊，請參閱 Application Auto Scaling 的服務連結角色。

啟用現有資料表的 DynamoDB Auto Scaling

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇要啟用自動擴展的資料表，然後依下列步驟操作：
   a. 前往其他設定分頁。
   b. 在 Read/write capacity (讀取/寫入容量) 區段中，選擇 Edit (編輯)。
   c. 在 Capacity mode (容量模式) 區段中，選擇 Provisioned (佈建)。
   d. 在 Table capacity (資料表容量) 區段，將 Read capacity (讀取容量)、Write capacity (寫入容量)，或兩者的 Auto scaling (自動擴展) 設定為 On (開啟)。針對這些項目，請為資料表設定所需的擴展政策，並選擇性地設定資料表的所有全域次要索引。
      - 容量單位下限：輸入自動擴展範圍的下限。
      - 容量單位上限：輸入自動擴展範圍的上限。
      - 目標使用率：輸入資料表的目標使用率百分比。
      - 對所有全域次要索引使用相同的容量讀取/寫入容量設定-選擇全域次要索引是否應該使用與基礎資料表相同的自動擴展政策。

      > **Note**
      > 為獲得最佳效能，我們建議您啟用 Use the same read/write capacity settings for all global secondary indexes (針對全部全域次要索引使用相同的讀取/寫入容量設定)。此選項允許 DynamoDB Auto Scaling 功能以統一方式擴展基礎資料表上的所有全域次要索引。這包括現有的全域次要索引，以及您未來為此資料表建立的任何其他索引。
      > 啟用此選項後，您就無法在個別全域次要索引上設定擴展政策。
4. 當您滿意設定後，請選擇 Save (儲存)。

在主控台上檢視自動擴展活動

當應用程式將讀取和寫入流量推送至資料表時，DynamoDB Auto Scaling 功能會動態修改資料表的輸送量設定。Amazon CloudWatch 會追蹤所有 DynamoDB 資料表和次要索引的佈建和耗用容量、限流事件、延遲和其他指標。

若要在 DynamoDB 主控台中檢視這些指標，請選擇要使用的資料表，然後選擇 Monitor (監控) 索引標籤。若要建立資料表指標的可自訂檢視，請選取 View all in CloudWatch (在 CloudWatch 檢視全部)。

修改或停用 DynamoDB Auto Scaling 設定

您可以使用 AWS 管理主控台 來修改 DynamoDB 自動擴展設定。若要這樣做，請前往 Additional settings (其他設定) 索引標籤，然後選擇 Edit (編輯) 中的 Read/write capacity (讀取/寫入容量) 一節。如需這些設定的詳細資訊，請參閱 在現有資料表上啟用 DynamoDB Auto Scaling 功能。

使用 AWS CLI 管理 DynamoDB 自動擴展

您可以使用 AWS Command Line Interface (AWS CLI) 管理 Amazon DynamoDB 自動擴展 AWS 管理主控台，而不是使用 。本節中的教學課程會示範如何安裝和設定 AWS CLI 來管理 DynamoDB Auto Scaling 功能。在此教學課程中，您將執行下列操作：

- 建立名為 `TestTable` 的 DynamoDB 資料表。初始輸送量設定是 5 個讀取容量單位和 5 個寫入容量單位。
- 為 `TestTable` 建立 Application Auto Scaling 政策。該政策會設法將使用寫入容量與佈建寫入容量之間的目標比率保持在 50%。此指標範圍介於 5 到 10 個寫入容量單位之間。(Application Auto Scaling 無法調整超出此範圍的輸送量。)
- 運行 Python 程式來將寫入流量推送至 `TestTable`。當目標比率持續超過 50% 時，Application Auto Scaling 會通知 DynamoDB 提高 `TestTable` 的輸送量，以便維持 50% 的目標使用率。
- 確認 DynamoDB 已成功調整 `TestTable` 的佈建寫入容量。

> **Note**
>
> 您也可以設定 DynamoDB 擴展的排程，使其在特定時間執行。在此了解基本步驟。

主題

- 開始之前
- 步驟 1：建立 DynamoDB 資料表
- 步驟 2：註冊可擴展的目標
- 步驟 3：建立擴展政策
- 步驟 4：將寫入流量導向 TestTable

- 步驟 5：檢視 Application Auto Scaling 動作
- (選用) 步驟 6：清除

開始之前

開始這些教學課程之前，請完成以下任務。

安裝 AWS CLI

若您尚未執行此作業，您必須安裝及設定 AWS CLI。若要執行此作業，請遵循《AWS Command Line Interface 使用者指南》中的這些說明：

- 安裝 AWS CLI
- 設定 AWS CLI

安裝 Python

本教學課程的一部分會要求您執行 Python 程式 (請參閱 步驟 4：將寫入流量導向 TestTable)。如果您還沒有安裝 Python 程式，則可以下載 Python。

步驟 1：建立 DynamoDB 資料表

在此步驟中，您會使用 AWS CLI 來建立 `TestTable`。主索引鍵包含 `pk` (分割區索引鍵) 和 `sk` (排序索引鍵)。這些屬性的類型皆為 `Number`。初始輸送量設定是 5 個讀取容量單位和 5 個寫入容量單位。

1. 使用下列 AWS CLI 命令來建立資料表。

```
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. 若要檢查資料表的狀態，請使用以下命令。

```
aws dynamodb describe-table \
```

```
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

當資料表狀態為 ACTIVE 時，即可供使用。

步驟 2：註冊可擴展的目標

接下來，您可以使用 Application Auto Scaling 將資料表寫入容量註冊為可擴展的目標。這可讓 Application Auto Scaling 調整 TestTable 的佈建寫入容量，但僅在 5 至 10 個容量單位的範圍內。

> **Note**
>
> DynamoDB Auto Scaling 需要有服務連結角色 (AWSServiceRoleForApplicationAutoScaling_DynamoDBTable)，其會代表您執行自動擴展動作。系統會自動建立此角色。如需詳細資訊，請參閱《Application Auto Scaling 使用者指南》中的適用於 Application Auto Scaling 的服務連結角色。

1. 輸入下列命令來註冊可擴展的目標。

```
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. 若要確認註冊，請使用下列命令。

```
aws application-autoscaling describe-scalable-targets \
    --service-namespace dynamodb \
    --resource-id "table/TestTable"
```

> **Note**
>
> 您也可以針對全域次要索引註冊可擴展的目標。例如，對於全域次要索引 (test-index)，資源 ID 和可擴展的維度引數會適當地更新。
>
> ```
> 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
```

步驟 3：建立擴展政策

在此步驟中，您會建立 TestTable 的擴展政策。此政策會定義 Application Auto Scaling 可在哪些詳細資訊下調整資料表的佈建輸送量，以及在執行此動作時要採取的動作。您可以將此政策與您在上一個步驟中定義的可擴展目標建立關聯 (TestTable 資料表的寫入容量單位)。

該政策包含下列元素：

- PredefinedMetricSpecification：允許 Application Auto Scaling 調整的指標。對於 DynamoDB，下列值是 PredefinedMetricType 的有效值：
  - DynamoDBReadCapacityUtilization
  - DynamoDBWriteCapacityUtilization
- ScaleOutCooldown：提高佈建輸送量的每個 Application Auto Scaling 事件之間的最短時間 (以秒為單位)。此參數可讓 Application Auto Scaling 持續 (但不積極) 增加輸送量，以便回應真實的工作負載。ScaleOutCooldown 的預設設定為 0。
- ScaleInCooldown：減少佈建輸送量的每個 Application Auto Scaling 事件之間的最短時間 (以秒為單位)。此參數可讓 Application Auto Scaling 以逐漸且可預測的方式降低輸送量。ScaleInCooldown 的預設設定為 0。
- TargetValue：Application Auto Scaling 可確保將耗用容量與佈建容量的比率保持在此值或接近此數值。您能以百分比的形式定義 TargetValue。

> **Note**
>
> 為了進一步了解 TargetValue 如何運作，請假設您資料表的佈建輸送量設定為 200 個寫入容量單位。您決定為此資料表建立擴展政策，並將 TargetValue 設為 70 %。
> 現在，假設您開始將寫入流量導向該資料表，那實際的寫入輸送量就會是 150 個容量單位。耗用與佈建比率現在為 (150/200)，也就是 75%。此比率超過您的目標，因此 Application Auto Scaling 會將佈建的寫入容量提高至 215，使得比率成為 (150/215) (也就是 69.77 百分比)；盡可能靠近但不超過 TargetValue。

對於 TestTable，您將 TargetValue 設為 50%。Application Auto Scaling 將資料表佈建的輸送量調整在 5–10 個容量單位的範圍內 (請參閱 步驟 2：註冊可擴展的目標)，使耗用與佈建的比率維持在或接近 50%。您可以將 ScaleOutCooldown 和 ScaleInCooldown 的數值設為 60 秒。

1. 使用下列內容建立名為 scaling-policy.json 的檔案。

```
{
    "PredefinedMetricSpecification": {
        "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60,
    "TargetValue": 50.0
}
```

2. 使用下列 AWS CLI 命令來建立政策。

```
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. 請注意，在輸出中，Application Auto Scaling 已建立兩個 Amazon CloudWatch 警示：擴展目標範圍的上限和下限各一個警示。

4. 使用下列 AWS CLI 命令來檢視擴展政策的詳細資訊。

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --policy-name "MyScalingPolicy"
```

5. 在輸出中，確認政策設定符合 步驟 2：註冊可擴展的目標 和 步驟 3：建立擴展政策 中的規格。

步驟 4：將寫入流量導向 TestTable

現在，您可以將資料寫入 TestTable 來測試擴展政策。為此，您須執行 Python 程式。

1. 使用下列內容建立名為 bulk-load-test-table.py 的檔案。

```
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. 請輸入下列命令來執行程式。

   ```
   python bulk-load-test-table.py
   ```

   TestTable 的佈建寫入容量非常低 (5 個寫入容量單位)，因此程式偶爾會因寫入限流而停滯。這是預期的行為。

   讓程式繼續執行，您則繼續下一個步驟。

步驟 5：檢視 Application Auto Scaling 動作

在此步驟中，您可以檢視代表您啟動的 Application Auto Scaling 動作。您也可以確認 Application Auto Scaling 已更新 TestTable 的佈建寫入容量。

1. 輸入下列命令來檢視 Application Auto Scaling 動作。

   ```
   aws application-autoscaling describe-scaling-activities \
       --service-namespace dynamodb
   ```

在 Python 程式執行時，偶爾重新執行此命令。(調用擴展政策需要幾分鐘的時間。) 您最終應該會看到下列輸出。

```
...
{
    "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"
},
...
```

這表示 Application Auto Scaling 已對 DynamoDB 發出 `UpdateTable` 請求。

2. 輸入下列命令，確認 DynamoDB 已增加資料表的寫入容量。

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

`WriteCapacityUnits` 應該從 5 擴展至 10。

(選用) 步驟 6：清除

在此教學課程中，您已建立數個資源。如果不再需要這些資源，就可以將其刪除。

1. 刪除 `TestTable` 的擴展政策。

```
aws application-autoscaling delete-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
```

```
    --policy-name "MyScalingPolicy"
```

2. 取消註冊可擴展的目標。

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits"
```

3. 刪除 TestTable 資料表。

```
aws dynamodb delete-table --table-name TestTable
```

使用 AWS SDK 在 Amazon DynamoDB 資料表上設定自動擴展

除了使用 AWS 管理主控台 和 AWS Command Line Interface (AWS CLI)，您還可以撰寫與 Amazon DynamoDB Auto Scaling 互動的應用程式。本節包含可用來測試此功能的兩個 Java 程式：

- EnableDynamoDBAutoscaling.java
- DisableDynamoDBAutoscaling.java

啟用資料表的 Application Auto Scaling

以下程式顯示 DynamoDB 資料表 (TestTable) 自動擴展政策的設定範例。它會繼續如下：

- 此程式會將寫入容量單位註冊為 TestTable 的可擴展目標。此指標範圍介於 5 到 10 個寫入容量單位之間。
- 建立可擴展目標之後，程式會建置目標追蹤組態。該政策會設法將使用寫入容量與佈建寫入容量之間的目標比率保持在 50%。
- 程式接著會根據目標追蹤組態來建立擴展政策。

> **Note**
>
> 手動移除資料表或全域資料表複本時，不會自動移除任何相關聯的可擴展目標、擴展政策或 CloudWatch 警示。

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

此程式需要您提供有效 Application Auto Scaling 服務連結角色的 Amazon Resource Name (ARN)。(例如：`arn:aws:iam::122517410325:role/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`。) 在下列程式中，將 `SERVICE_ROLE_ARN_GOES_HERE` 取代為實際 ARN。

```
package com.amazonaws.codesamples.autoscaling;

import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClientBuilder;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.MetricType;
import com.amazonaws.services.applicationautoscaling.model.PolicyType;
import
 com.amazonaws.services.applicationautoscaling.model.PredefinedMetricSpecification;
import com.amazonaws.services.applicationautoscaling.model.PutScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.RegisterScalableTargetRequest;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;
import
 com.amazonaws.services.applicationautoscaling.model.TargetTrackingScalingPolicyConfiguratio

public class EnableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = (AWSApplicationAutoScalingClient)
 AWSApplicationAutoScalingClientBuilder
   .standard().build();

 public static void main(String args[]) {
```

```
 ServiceNamespace ns = ServiceNamespace.Dynamodb;
 ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
 String resourceID = "table/TestTable";

 // Define the scalable target
 RegisterScalableTargetRequest rstRequest = new RegisterScalableTargetRequest()
   .withServiceNamespace(ns)
   .withResourceId(resourceID)
   .withScalableDimension(tableWCUs)
   .withMinCapacity(5)
   .withMaxCapacity(10)
   .withRoleARN("SERVICE_ROLE_ARN_GOES_HERE");

 try {
  aaClient.registerScalableTarget(rstRequest);
 } catch (Exception e) {
  System.err.println("Unable to register scalable target: ");
  System.err.println(e.getMessage());
 }

 // Verify that the target was created
 DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceIds(resourceID);
 try {
  DescribeScalableTargetsResult dsaResult =
aaClient.describeScalableTargets(dscRequest);
  System.out.println("DescribeScalableTargets result: ");
  System.out.println(dsaResult);
  System.out.println();
 } catch (Exception e) {
  System.err.println("Unable to describe scalable target: ");
  System.err.println(e.getMessage());
 }

 System.out.println();

 // Configure a scaling policy
 TargetTrackingScalingPolicyConfiguration targetTrackingScalingPolicyConfiguration
= new TargetTrackingScalingPolicyConfiguration()
   .withPredefinedMetricSpecification(
     new PredefinedMetricSpecification()
       .withPredefinedMetricType(MetricType.DynamoDBWriteCapacityUtilization))
```

```
    .withTargetValue(50.0)
    .withScaleInCooldown(60)
    .withScaleOutCooldown(60);

  // Create the scaling policy, based on your configuration
  PutScalingPolicyRequest pspRequest = new PutScalingPolicyRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID)
    .withPolicyName("MyScalingPolicy")
    .withPolicyType(PolicyType.TargetTrackingScaling)

 .withTargetTrackingScalingPolicyConfiguration(targetTrackingScalingPolicyConfiguration);

  try {
   aaClient.putScalingPolicy(pspRequest);
  } catch (Exception e) {
   System.err.println("Unable to put scaling policy: ");
   System.err.println(e.getMessage());
  }

  // Verify that the scaling policy was created
  DescribeScalingPoliciesRequest dspRequest = new DescribeScalingPoliciesRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID);

  try {
   DescribeScalingPoliciesResult dspResult =
 aaClient.describeScalingPolicies(dspRequest);
   System.out.println("DescribeScalingPolicies result: ");
   System.out.println(dspResult);
  } catch (Exception e) {
   e.printStackTrace();
   System.err.println("Unable to describe scaling policy: ");
   System.err.println(e.getMessage());
  }

 }

}
```

停用資料表的 Application Auto Scaling

下列程式會反轉先前的程序。它會移除自動擴展政策，然後取消註冊可擴展目標。

Java v2

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExceptio
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeregisterScalableTargetReques
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class DisableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <policyName>\s
```

```
            Where:
               tableId - The table Id value (for example, table/Music).\s
               policyName - The name of the policy (for example, $Music5-scaling-
policy).

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

        ApplicationAutoScalingClient appAutoScalingClient =
 ApplicationAutoScalingClient.builder()
            .region(Region.US_EAST_1)
            .build();

        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
 ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        String tableId = args[0];
        String policyName = args[1];

        deletePolicy(appAutoScalingClient, policyName, tableWCUs, ns, tableId);
        verifyScalingPolicies(appAutoScalingClient, tableId, ns, tableWCUs);
        deregisterScalableTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
    }

    public static void deletePolicy(ApplicationAutoScalingClient
 appAutoScalingClient, String policyName, ScalableDimension tableWCUs,
 ServiceNamespace ns, String tableId) {
        try {
            DeleteScalingPolicyRequest delSPRequest =
 DeleteScalingPolicyRequest.builder()
                .policyName(policyName)
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deleteScalingPolicy(delSPRequest);
            System.out.println(policyName +" was deleted successfully.");

        } catch (ApplicationAutoScalingException e) {
```

```
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    // Verify that the scaling policy was deleted
    public static void verifyScalingPolicies(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        DescribeScalingPoliciesRequest dscRequest =
DescribeScalingPoliciesRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceId(tableId)
            .build();

        DescribeScalingPoliciesResponse response =
appAutoScalingClient.describeScalingPolicies(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }

    public static void deregisterScalableTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        try {
            DeregisterScalableTargetRequest targetRequest =
DeregisterScalableTargetRequest.builder()
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deregisterScalableTarget(targetRequest);
            System.out.println("The scalable target was deregistered.");

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    public static void verifyTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
```

```
        DescribeScalableTargetsRequest dscRequest =
 DescribeScalableTargetsRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceIds(tableId)
            .build();

        DescribeScalableTargetsResponse response =
 appAutoScalingClient.describeScalableTargets(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }
}
```

Java v1

```
package com.amazonaws.codesamples.autoscaling;

import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import
 com.amazonaws.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DeregisterScalableTargetRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;

public class DisableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = new
 AWSApplicationAutoScalingClient();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
```

```
 ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
 String resourceID = "table/TestTable";

 // Delete the scaling policy
 DeleteScalingPolicyRequest delSPRequest = new DeleteScalingPolicyRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID)
   .withPolicyName("MyScalingPolicy");

 try {
  aaClient.deleteScalingPolicy(delSPRequest);
 } catch (Exception e) {
  System.err.println("Unable to delete scaling policy: ");
  System.err.println(e.getMessage());
 }

 // Verify that the scaling policy was deleted
 DescribeScalingPoliciesRequest descSPRequest = new
DescribeScalingPoliciesRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID);

 try {
  DescribeScalingPoliciesResult dspResult =
aaClient.describeScalingPolicies(descSPRequest);
  System.out.println("DescribeScalingPolicies result: ");
  System.out.println(dspResult);
 } catch (Exception e) {
  e.printStackTrace();
  System.err.println("Unable to describe scaling policy: ");
  System.err.println(e.getMessage());
 }

 System.out.println();

 // Remove the scalable target
 DeregisterScalableTargetRequest delSTRequest = new
DeregisterScalableTargetRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID);
```

```
  try {
   aaClient.deregisterScalableTarget(delSTRequest);
  } catch (Exception e) {
   System.err.println("Unable to deregister scalable target: ");
   System.err.println(e.getMessage());
  }

  // Verify that the scalable target was removed
  DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceIds(resourceID);

  try {
   DescribeScalableTargetsResult dsaResult =
 aaClient.describeScalableTargets(dscRequest);
   System.out.println("DescribeScalableTargets result: ");
   System.out.println(dsaResult);
   System.out.println();
  } catch (Exception e) {
   System.err.println("Unable to describe scalable target: ");
   System.err.println(e.getMessage());
  }

 }

}
```

DynamoDB 預留容量

對於使用標準資料表類別的佈建容量資料表，DynamoDB 提供為讀取與寫入容量購買預留容量的功能。預留容量購買是一項協議，您需在協議期間支付最低佈建輸送容量，以換取優惠價格。

> **Note**
>
> 您無法為複寫寫入容量單位 (rWCU) 購買預留容量。預留容量僅適用於其購買所在的區域。使用 DynamoDB 標準 IA 資料表類別或隨需容量模式的資料表也無法使用預留容量。

預留容量以 100 個 WCUs 或 100 個 RCUs 為單位購買。最小可購買的預留容量方案為 100 個容量單位 (讀取或寫入)。DynamoDB 預留容量可採一年期承諾，或在特定區域中提供三年期承諾。一年期方案可節省最高 54% 的費用，三年期方案最高可節省 77%。如需了解購買時機與方式的詳細資訊，請參閱 Amazon DynamoDB 預留容量。

> **Note**
>
> 您可以使用 管理主控台，為寫入容量單位 (WCUs) 和讀取容量單位 (RCUs) AWS 購買最多 1，000，000 個預留容量單位。如果您想要在單一購買中購買超過 1，000，000 個佈建容量單位，或具有作用中的預留容量，並想要購買額外的預留容量，這會導致超過 1，000，000 個作用中的佈建容量單位，請遵循 Amazon DynamoDB 預留容量中「如何購買預留容量」一節所述的程序。

購買 DynamoDB 預留容量時，您需支付一次性的部分預付款，並可享有已承諾佈建用量的優惠每小時費率。無論實際用量如何，您仍需支付整體承諾佈建用量的費用，因此節省金額與實際使用情況密切相關。超出已購預留容量的佈建部分將依標準佈建容量費率計費。透過事先預留您的讀取與寫入容量單位，您就可以比佈建容量成本省下明顯更多費用。

您無法販售、取消或將預留容量轉移至其他區域或帳戶。

認識 DynamoDB 暖輸送量

暖輸送量指 DynamoDB 資料表可即時支援的讀取與寫入操作次數。根據預設，這些值適用於所有資料表與全域次要索引 (GSI)，並反映其根據歷史用量調整的擴展程度。若您使用隨需模式，或將佈建輸送量更新為這些值，應用程式即可即時處理達該值上限的請求。

當用量增加時，DynamoDB 會自動調整暖輸送量值。您也可以在需要時主動增加這些值，這對於即將推出的尖峰事件特別有用，例如產品推出或銷售。對於預先規劃的尖峰活動，若 DynamoDB 資料表的請求率可能提升 10 倍、100 倍或更高，您可評估當前暖輸送量是否足以應付預期流量。若不足，您可提升暖輸送量值，而不需變更輸送量設定或計費模式。此程序稱為預熱資料表，讓您可設定資料表可即時支援的基準值。這可確保應用程式能自請求高峰發生之時起即處理更高的請求率。一旦增加，就無法減少暖輸送量值。

您可提升讀取、寫入或兩者的暖輸送量值。此值可套用於新建或現有的單一區域資料表、全域資料表與 GSIs。對全域資料表而言，此功能適用於 2019.11.21 版 (目前)，您設定的暖輸送量將自動套用至全域資料表的所有複本資料表。您可隨時預熱任意數量的 DynamoDB 資料表。完成預熱所需時間取決於設定的值與資料表或索引的大小。您可同時提交多個預熱請求，且不會干擾任何資料表操作。您可將資料

表預熱至該區域帳戶的資料表或索引配額上限。使用 Service Quotas 主控台檢查目前的限制，並視需要提高配額。

根據預設，所有資料表與次要索引皆可免費使用暖輸送量值。然而，若您主動提升預設暖輸送量值以預熱資料表，將需支付相關請求的費用。如需詳細資訊，請參閱 Amazon DynamoDB 定價。

如需暖輸送量的詳細資訊，請參閱以下主題。

主題

- 檢查 DynamoDB 資料表的目前暖輸送量
- 提高現有 DynamoDB 資料表的暖輸送量
- 建立暖輸送量較高的新 DynamoDB 資料表
- 了解不同情境下的 DynamoDB 暖輸送量

檢查 DynamoDB 資料表的目前暖輸送量

使用下列 AWS CLI 和 AWS 主控台指示來檢查資料表或索引目前的暖輸送量值。

AWS 管理主控台

若要透過 DynamoDB 主控台檢查 DynamoDB 資料表的暖輸送量：

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 在資料表頁面上，選取目標資料表。
4. 選取其他設定以檢視目前的暖輸送量值。此值顯示為每秒讀取單位與每秒寫入單位。

AWS CLI

下列 AWS CLI 範例示範如何檢查 DynamoDB 資料表的暖輸送量。

1. 在 DynamoDB 資料表上執行 `describe-table` 操作。

```
aws dynamodb describe-table --table-name GameScores
```

2. 系統會傳回類似下列的回應。WarmThroughput 設定會顯示為 ReadUnitsPerSecond 和 WriteUnitsPerSecond。當暖輸送量值正在更新時，Status 會顯示為 UPDATING；當設定完成後，則顯示為 ACTIVE。

```
{
    "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"
```

```
        }
    }
}
```

提高現有 DynamoDB 資料表的暖輸送量

檢查完 DynamoDB 資料表目前的暖輸送量值後，請依下列步驟進行更新：

AWS 管理主控台

若要透過 DynamoDB 主控台檢查 DynamoDB 資料表的暖輸送量值：

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 在資料表頁面上，選取目標資料表。
4. 在暖輸送量欄位中，選取編輯。
5. 在編輯暖輸送量頁面上，選擇提高暖輸送量。
6. 調整每秒讀取單位與每秒寫入單位。這兩個設定決定資料表可立即處理的輸送量上限。
7. 選取儲存。
8. 當請求處理完成後，暖輸送量欄位會更新每秒讀取單位和每秒寫入單位的最新值。

> **Note**
>
> 更新暖輸送量值為非同步作業。當更新完成時，`Status` 會從 `UPDATING` 變更為 `ACTIVE`。

AWS CLI

下列 AWS CLI 範例示範如何更新 DynamoDB 資料表的暖輸送量值。

1. 在 DynamoDB 資料表上執行 `update-table` 操作。

```
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. 系統會傳回類似下列的回應。WarmThroughput 設定會顯示為 ReadUnitsPerSecond 和 WriteUnitsPerSecond。當暖輸送量值正在更新時，Status 會顯示為 UPDATING；當設定完成後，則顯示為 ACTIVE。

```
{
    "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 開發套件

下列 SDK 範例示範如何更新 DynamoDB 資料表的暖輸送量設定。

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;
  }
}
```

建立暖輸送量較高的新 DynamoDB 資料表

建立 DynamoDB 資料表時，可依下列步驟調整暖輸送量設定。這些步驟同樣適用於建立全域資料表或次要索引。

AWS 管理主控台

若要透過主控台建立 DynamoDB 資料表並設定暖輸送量：

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 選取建立資料表。
3. 選取資料表名稱、分割區索引鍵和排序索引鍵 (選用)。
4. 在資料表設定中，選取自訂設定。
5. 在暖輸送量欄位中，選取增加暖輸送量。
6. 調整每秒讀取單位與每秒寫入單位。這兩個設定定義了資料表可即時處理的最大輸送量。
7. 完成其他資料表詳細設定後，選取建立資料表。

AWS CLI

下列 AWS CLI 範例說明如何建立具有自訂暖輸送量值的 DynamoDB 資料表。

1. 執行 `create-table` 以建立下列 DynamoDB 資料表。

```
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. 系統會傳回類似下列的回應。WarmThroughput 設定會顯示為 ReadUnitsPerSecond 和 WriteUnitsPerSecond。當暖輸送量值正在更新時，Status 會顯示為 UPDATING；當設定完成後，則顯示為 ACTIVE。

```
{
    "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 開發套件

下列 SDK 範例示範如何建立具自訂暖輸送量設定的 DynamoDB 資料表。

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;
  }
}
```

了解不同情境下的 DynamoDB 暖輸送量

以下列出在使用 DynamoDB 暖輸送量時可能遇到的多種情境。

主題

- 暖輸送量與非均勻存取模式

- 佈建資料表的暖輸送量
- 隨需資料表的暖輸送量
- 已設定最大輸送量的隨需資料表暖輸送量

暖輸送量與非均勻存取模式

資料表的暖輸送量可能為每秒 30,000 個讀取單位與每秒 10,000 個寫入單位，但在達到此上限前，仍可能在讀取或寫入操作中遭遇限流。這通常是因為存在熱分割區造成。雖然 DynamoDB 可持續擴展以支援幾乎無限制的輸送量，但每個分割區的限制為每秒 1,000 個寫入單位及每秒 3,000 個讀取單位。若應用程式對資料表少部分分割區發送過多流量，即便尚未達到資料表的暖輸送量上限，也可能遭遇限流。建議遵循 DynamoDB 最佳實務，以確保系統具備無縫可擴展性並避免產生熱分割區。

佈建資料表的暖輸送量

假設某個佈建資料表的暖輸送量為每秒 30,000 個讀取單位與每秒 10,000 個寫入單位，但目前的佈建輸送量僅為 4,000 個 RCU 和 8,000 個 WCU。您可透過更新佈建輸送量設定，立即將資料表的佈建輸送量擴展至 30,000 個 RCU 或 10,000 個 WCU。當您將佈建輸送量提升至超過上述數值時，暖輸送量會自動調整為新的更高值，因為這代表系統已建立新的峰值輸送量。例如，若將佈建輸送量設定為 50,000 RCU，暖輸送量將隨之提升至每秒 50,000 個讀取單位。

```
"ProvisionedThroughput":
    {
        "ReadCapacityUnits": 4000,
        "WriteCapacityUnits": 8000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

隨需資料表的暖輸送量

新建立的隨需資料表初始暖輸送量為每秒 12,000 個讀取單位與每秒 4,000 個寫入單位。資料表可立即支援持續流量，最高可達上述層級。當請求量超過每秒 12,000 個讀取單位或 4,000 個寫入單位時，暖輸送量會自動提升至更高等級。

```
"WarmThroughput":
```

```
    {
        "ReadUnitsPerSecond": 12000,
        "WriteUnitsPerSecond": 4000
    }
```

已設定最大輸送量的隨需資料表暖輸送量

假設某個隨需資料表的暖輸送量為每秒 30,000 個讀取單位，但最大輸送量設定為 5,000 個讀取請求單位 (RRU)。在此情境中，資料表的輸送量將受限於您設定的 5,000 個 RRU 上限。任何超出該上限的輸送量請求都會遭到限流。不過，您可依應用程式需求，隨時調整該資料表的專屬最大輸送量。

```
"OnDemandThroughput":
    {
        "MaxReadRequestUnits": 5000,
        "MaxWriteRequestUnits": 4000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

DynamoDB 高載容量與調適型容量

為了將因輸送量異常導致的限流降至最低，DynamoDB 會利用高載容量因應突發用量峰值。DynamoDB 使用調適型容量以自動調整資源，應對不均勻的存取模式。

高載容量

DynamoDB 透過高載容量，為您的輸送量佈建提供一些靈活性。當未充分使用可用輸送量時，DynamoDB 會保留部分閒置容量，供後續輸送量高載期間應對用量峰值之用。使用高載容量，未預期的讀取或寫入請求在經過限流之後可能會成功。

DynamoDB 目前最多可保留 5 分鐘 (300 秒) 的未使用讀寫容量。在偶發的高載讀取或寫入活動中，這些額外容量單位可能迅速被耗盡，速度甚至快於您為資料表設定的每秒佈建輸送容量。

DynamoDB 也可能在不事先通知的情況下為背景維護和其他任務使用高載容量。

請注意，高載容量的詳細資訊可能會在將來有所變更。

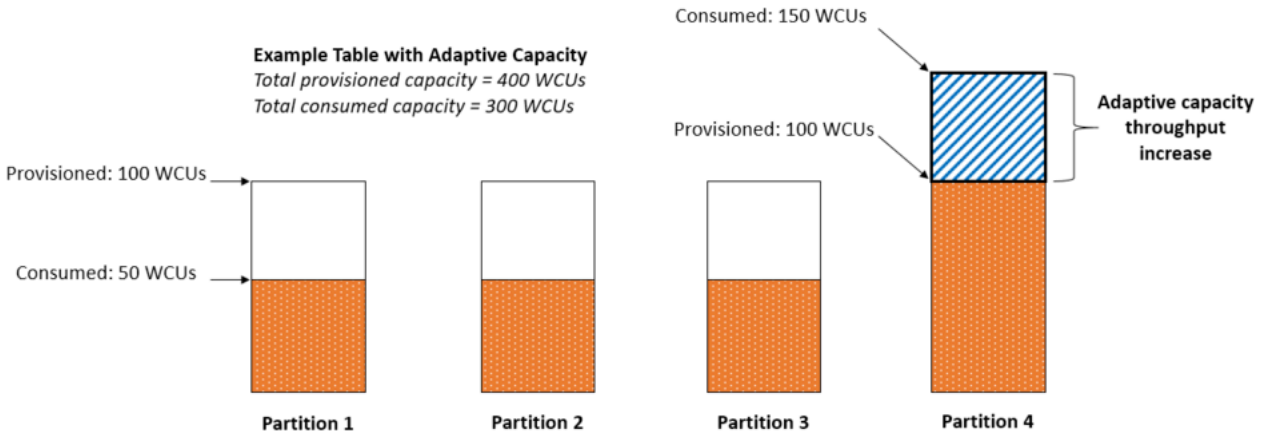
調適型容量

DynamoDB 會自動將資料分配至多個分割區，這些分割區儲存於 AWS 雲端多部伺服器上。並非所有情況下都能均勻分配讀寫作業。當資料存取不平衡時，與其他分割區相比，「經常性」分割區可以接收更高量的讀取和寫入流量。由於分割區的讀取與寫入作業皆獨立管理，若單一分割區接收超過 3,000 次讀取作業或 1,000 次以上的寫入作業，即會觸發限流。調適型容量用於自動增加能接收更多流量的分割區輸送容量。

DynamoDB 調適型容量使您的應用程式能夠繼續讀取和寫入經常性分割區而不受限制，加強適應不均衡的存取模式，前提是流量不會超過您資料表的總佈建的容量或分割區最大容量。調適型容量用於自動並立即增加能接收更多流量的分割區輸送容量。

下圖說明調適型容量的運作方式。此範例資料表已佈建 400 個單位 (WCU) 並在四個分割區之間平均共用，因此每個分割區每秒最多可承受 100 個 WCU。分割區 1、2 和 3 各接收每秒 50 個 WCU 的寫入流量。分割區 4 每秒接收 150 個 WCU。此經常性分割區可在仍有未使用的高載容量時接受寫入流量，但最終分割區會限流每秒超過 100 個 WCU 的流量。

DynamoDB 的調適型容量機制會自動提高分割區 4 的容量，使其能承受每秒 150 WCU 的高工作負載而不被限流。



所有 DynamoDB 資料表都會自動啟用調適型容量，不需額外費用。您不需要明確啟用或停用此功能。

隔離經常存取的項目

若您的應用程式導致一個或多個項目的流量過大，則調適型容量將重新平衡您的分割區，以使經常存取的項目不會駐留在同一分割區上。這種經常存取項目的隔離，可減少因您的工作負載於單一分割區超過輸送量配額而導致需要求限流的可能性。只要項目集合不是由排序索引鍵單調增加或減少所追蹤的流量，您也可以依排序索引鍵將項目集合分成區段。

如果您的應用程式於單一項目經常有高流量，調適型容量可能重新平衡您的資料，讓分割區僅包含單一經常存取的項目。如果是這種情況，DynamoDB 可為分割區中該單一項目的主索引鍵帶來最多 3,000 個 RCU 和 1,000 個 WCU 的輸送量。資料表上存在本機次要索引時，調適型容量不會在多個資料表分割區中分割項目集合。

切換 DynamoDB 容量模式時的注意事項

建立 DynamoDB 資料表時，您必須選取隨需或佈建容量模式。

您可在 24 小時滾動期間內，最多將資料表從佈建容量模式切換至隨需模式四次。您可隨時將資料表從隨需模式切換回佈建容量模式。

主題

- 從佈建容量模式切換至隨需容量模式
- 從隨需容量模式切換至佈建容量模式

從佈建容量模式切換至隨需容量模式

在佈建容量模式中，您可根據預期的應用程式需求設定讀取與寫入容量。當您將資料表從佈建的模式更新為隨需模式時，不需要指定您預期應用程式將進行的讀取和寫入輸送量。DynamoDB 隨需容量模式針對讀取與寫入請求提供「按請求付費」的簡單計價方式，讓您僅需依實際用量付費，輕鬆平衡成本與效能。您可視需要為個別隨需資料表及其關聯的全域次要索引設定最大讀取、寫入 (或兩者) 輸送量，以協助控制成本與用量上限。如需設定特定資料表或索引之最大輸送量的詳細資訊，請參閱 DynamoDB 隨需資料表的最大輸送量。

當您將資料表從佈建容量模式切換為隨需容量模式時，DynamoDB 會對資料表及分割區結構進行多項調整。此程序需要幾分鐘的時間。在切換期間，您資料表提供的輸送量將與先前佈建的寫入容量單位與讀取容量單位一致。

隨需容量模式的初始輸送量

如果您最近首次將現有資料表切換為隨需容量模式，該資料表將具有以下既有峰值設定，即使先前並未以隨需容量模式處理流量。

以下為可能的情境範例：

- 任何設定為低於 4,000 WCU 與 12,000 RCU，且未曾佈建更高容量的佈建資料表。當此資料表首次切換為隨需容量模式時，DynamoDB 會橫向擴充，以即時支援每秒至少 4,000 個寫入單位與 12,000 個讀取單位。

- 佈建為 8,000 WCU 與 24,000 RCU 的資料表。當此資料表切換為隨需容量模式時，將可隨時維持每秒至少 8,000 個寫入單位與 24,000 個讀取單位。
- 設定為 8,000 個 WCU 和 24,000 RCU 的佈建資料表，在持續期間每秒消耗 6,000 個寫入單位和 18,000 個讀取單位。當此資料表切換為隨需容量模式時，將可維持每秒至少 8,000 個寫入單位與 24,000 個讀取單位。先前的流量可能會進一步允許資料表在不限流的情況下，維持更高層級的流量。
- 先前設定為 10,000 個 WCU 和 10,000 RCU 的佈建資料表，但目前佈建 10 個 RCU 和 10 個 WCU。當此資料表切換為隨需容量模式時，將可維持每秒至少 10,000 個寫入單位與 10,000 個讀取單位。

自動擴展設定

當您將資料表從佈建的模式更新為隨需模式時：

- 如果您使用的是主控台，將會刪除所有的自動擴展設定 (如果有的話)。
- 如果您使用的是 AWS CLI 或 AWS SDK，則會保留所有自動擴展設定。當您再次將資料表更新為佈建的計費模式時，可以套用這些設定。

在 DynamoDB 主控台中批次編輯容量模式

您可使用 DynamoDB 主控台批次編輯多個資料表，將其從佈建容量模式切換至隨需容量模式。若要批次編輯容量模式：

1. 在 DynamoDB 主控台中開啟資料表頁面。
2. 選取要更新為隨需容量模式的資料表核取方塊。
3. 在動作功能表中選取更新至隨需容量模式。

此批次作業可讓您有效率地將多個資料表切換至隨需容量模式，而不必逐一更新。

從隨需容量模式切換至佈建容量模式

從隨需模式切換回佈建容量模式時，則該資料表提供的輸送量將與原先設定為隨需容量模式時達到的先前峰值一致。

管理容量

在將資料表從隨需模式更新為佈建的模式時，請考量下列的事項：

- 如果您使用的是 AWS CLI 或 AWS 開發套件，請使用 Amazon CloudWatch 查看歷史用量 (`ConsumedWriteCapacityUnits` 和 `ConsumedReadCapacityUnits`指標） 來判斷新的輸送量設定，以選擇資料表和全域次要索引的正確佈建容量設定。

  > **Note**
  >
  > 如果您將全域資料表切換為佈建的模式，決定新的輸送量設定時，請針對基礎資料表和全域次要索引，檢視涵跨其所有區域複本的最大耗用量。

- 若您要從隨需容量模式切換回佈建容量模式，請務必將初始佈建單位設定得足以支援資料表或索引在轉換期間的容量需求。

管理 Auto Scaling

當您將資料表從隨需模式更新為佈建的模式時：

- 若您使用主控台，建議依下列預設值啟用自動擴展功能。
  - 目標使用率：70%
  - 佈建容量下限：5 個單位
  - 佈建容量上限：區域最大值
- 如果您使用的是 AWS CLI 或 SDK，則會保留您之前的自動擴展設定 （如果有的話）。

使用 DynamoDB 和 AWS SDKs程式設計

本節涵蓋開發人員相關的主題。如果您要改為執行程式碼範例，請參閱執行此開發人員指南中的程式碼範例。

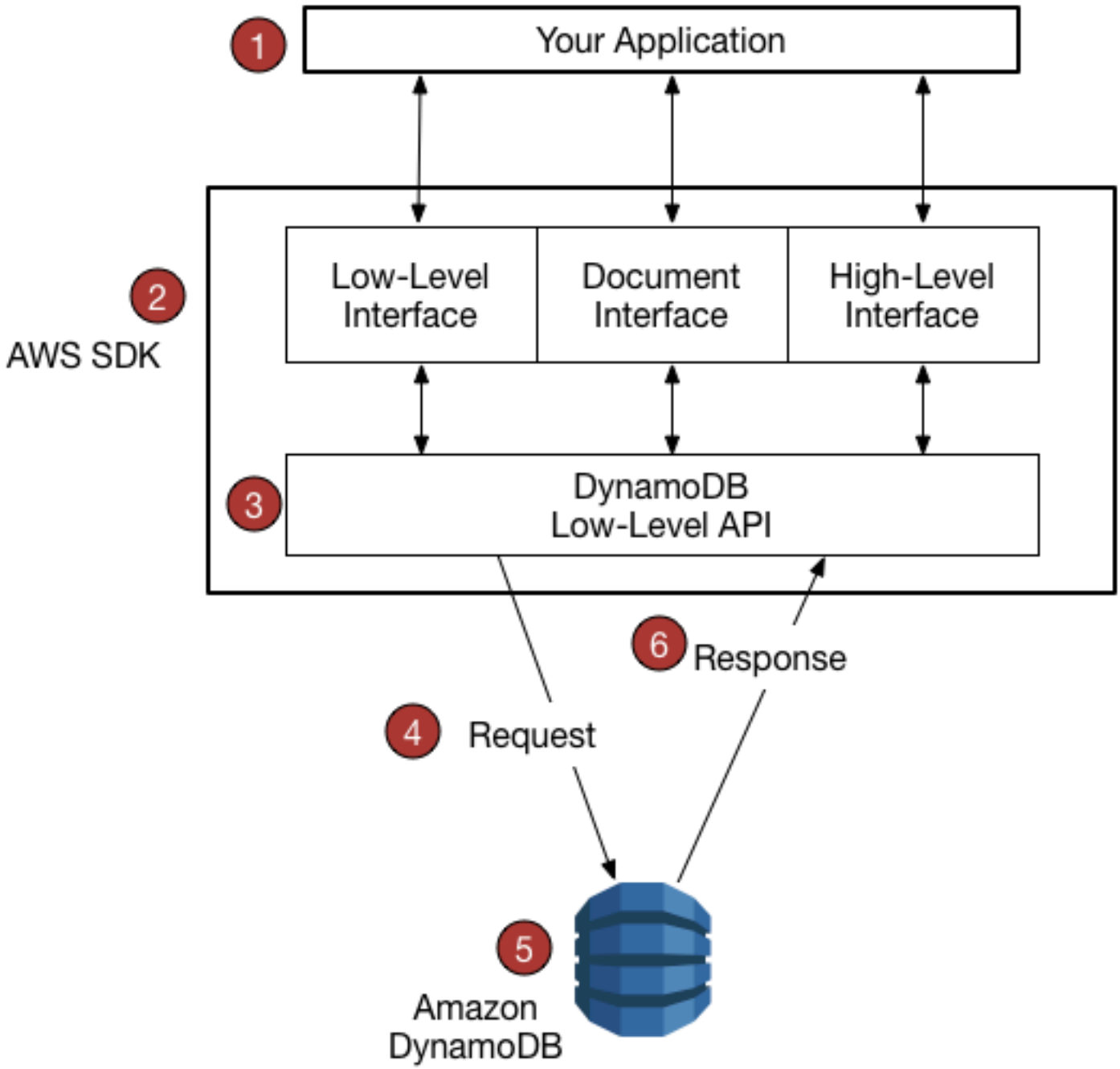
> **Note**
>
> 2017 年 12 月， AWS 開始遷移所有 Amazon DynamoDB 端點以使用 Amazon Trust Services (ATS) 發行的安全憑證的程序。如需詳細資訊，請參閱使用 DynamoDB 對 SSL/TLS 連線建立問題進行疑難排解。

主題

- DynamoDB 的 AWS SDK 支援概觀
- 使用 Python 和 Boto3 為 Amazon DynamoDB 編寫程式
- 使用 JavaScript 編寫 Amazon DynamoDB 程式
- 使用 程式設計 DynamoDB AWS SDK for Java 2.x
- 使用 DynamoDB 時發生錯誤
- 搭配 AWS SDK 使用 DynamoDB

DynamoDB 的 AWS SDK 支援概觀

下圖提供使用 AWS SDKs 的 Amazon DynamoDB 應用程式程式設計的高階概觀。



1. 您可以使用適用於程式設計語言的 AWS 開發套件撰寫應用程式。
2. 每個 AWS SDK 提供一或多個程式設計界面，以使用 DynamoDB。可用的特定界面取決於您使用的程式設計語言和 AWS SDK。選項包括：
   - 使用 DynamoDB 的低階介面
   - 使用 DynamoDB 的文件介面
   - 使用 DynamoDB 的物件持久性介面
   - 高階介面

3. SDK AWS 建構 HTTP(S) 請求，以搭配低階 DynamoDB API 使用。
4. SDK 會將請求 AWS 傳送至 DynamoDB 端點。
5. DynamoDB 會執行請求。如果請求成功，DynamoDB 會傳回 HTTP 200 回應代碼 (OK)。如果請求不成功，DynamoDB 會傳回 HTTP 錯誤代碼和錯誤訊息。
6. AWS SDK 會處理回應並將其傳播回您的應用程式。

每個 AWS SDKs都為您的應用程式提供重要的服務，包括下列項目：

- 格式化 HTTP(S) 請求和序列化請求參數。
- 為每個請求產生密碼編譯簽章。
- 將請求轉寄至 DynamoDB 端點，並接收來自 DynamoDB 的回應。
- 從這些回應中提取結果。
- 在出現錯誤的情況下實作基本重試邏輯。

您不需要為上述任何任務編寫程式碼。

> **Note**
>
> 如需 AWS SDKs的詳細資訊，包括安裝指示和文件，請參閱適用於 Amazon Web Services 的工具。

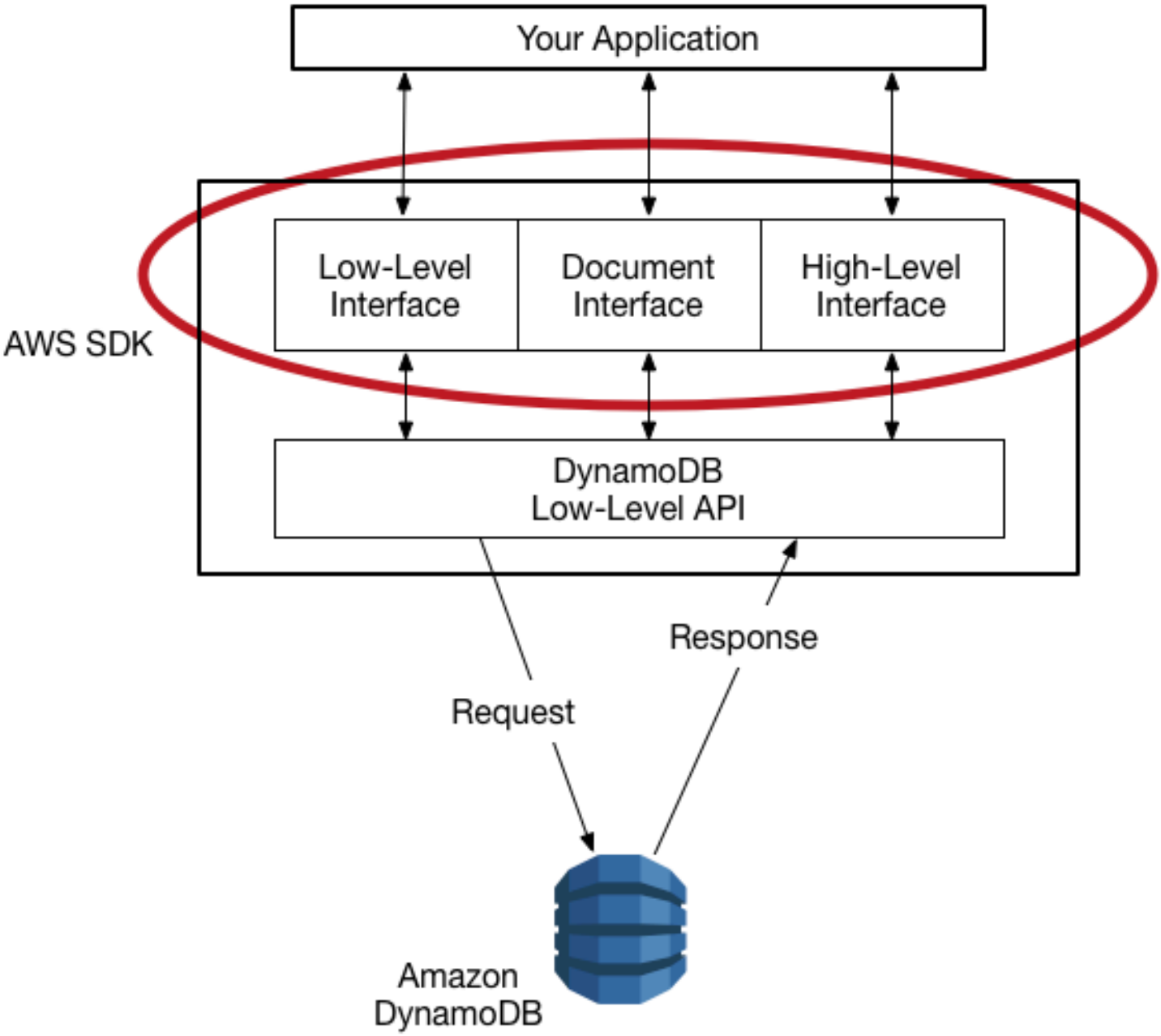
AWS 支援帳戶型端點的 SDK

AWS 已在 AWS 2024 年 9 月 4 日推出適用於 DynamoDB 帳戶型端點的 SDK 支援，從適用於 Java 的 AWS SDK V1 開始。這些新端點 AWS 有助於確保高效能和可擴展性。更新的 SDK 會自動使用格式為 `https://(account-id).ddb.(region).amazonaws.com` 的新端點。

如果您使用 SDK 用戶端的單一執行個體向多個帳戶提出請求，您的應用程式重複使用連線的機會會較少。 AWS 建議修改您的應用程式，以連接至每個 SDK 用戶端執行個體的較少帳戶。另一種方法是將 SDK 用戶端設定為使用 `ACCOUNT_ID_ENDPOINT_MODE` 設定繼續使用區域端點，如 AWS SDK 和工具參考指南中所述。

使用 DynamoDB 的程式設計介面

每個 AWS 軟體開發套件提供一或多個程式化介面，以便與 Amazon DynamoDB 搭配使用。這些介面範圍從簡單的低階 DynamoDB 包裝函式到物件導向的持久性層。可用的界面會根據您使用的 AWS SDK 和程式設計語言而有所不同。



下一節會以 適用於 Java 的 AWS SDK 為範例來重點介紹一些可用的介面。(並非全部介面都可用於所有 AWS 軟體開發套件。)

主題

- 使用 DynamoDB 的低階介面
- 使用 DynamoDB 的文件介面
- 使用 DynamoDB 的物件持久性介面

使用 DynamoDB 的低階介面

每個語言特定的 AWS SDK 都為 Amazon DynamoDB 提供低階介面，方法與低階 DynamoDB API 請求非常相似。

在某些情況下，您需要識別使用 資料類型描述項 的屬性資料類型，例如，適用於字串的 S 或適用於數字的 N。

> **Note**
>
> 每種語言特定的 AWS 軟體開發套件都提供低階介面。

下列 Java 程式使用 適用於 Java 的 AWS SDK的低階介面。

低階介面範例

```
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);
        }
    }
}
```

使用 DynamoDB 的文件介面

AWS SDKs提供文件界面，可讓您對資料表和索引執行資料平面操作（建立、讀取、更新、刪除）。使用文件介面時，您不需要指定 資料類型描述項。資料類型是由資料本身的語義隱含的。這些 AWS SDKs也提供方法，可輕鬆將 JSON 文件轉換為原生 Amazon DynamoDB 資料類型。

> **Note**
>
> 文件界面可在適用於 Java、.NET、Node.js 和 JavaScript SDK AWS SDKs 中使用。

下列 Java 程式使用 適用於 Java 的 AWS SDK的文件介面。程式會建立代表 `Music` 資料表的 `Table` 物件，然後要求該物件使用 `GetItem` 來擷取歌曲。然後程式會列印歌曲發行的年份。

`software.amazon.dynamodb.document.DynamoDB` 類別會實作 DynamoDB 文件介面。請注意 DynamoDB 作為低階用戶端 (`AmazonDynamoDB`) 包裝函式的方式。

文件介面範例

```
package com.amazonaws.codesamples.gsg;

import software.amazon.dynamodb.AmazonDynamoDB;
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import software.amazon.dynamodb.document.DynamoDB;
import software.amazon.dynamodb.document.GetItemOutcome;
import software.amazon.dynamodb.document.Table;

public class MusicDocumentDemo {

    public static void main(String[] args) {

        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");
        GetItemOutcome outcome = table.getItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Call Me Today");

        int year = outcome.getItem().getInt("Year");
        System.out.println("The song was released in " + year);

    }
}
```

使用 DynamoDB 的物件持久性介面

AWS SDKs提供物件持久性界面，您不會在其中直接執行資料平面操作。反之，您要建立代表 Amazon DynamoDB 資料表和索引中項目的物件，並且僅與這些物件互動。這允許您編寫以物件為中心的程式碼，而不是以資料庫為中心的程式碼。

> **Note**
>
> 適用於 Java 和 .NET AWS SDKs 中提供物件持久性界面。如需詳細資訊，請參閱適用於 DynamoDB 的 適用於 DynamoDB 的更高階程式設計界面。

物件持久性介面

```
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();
    }
}
```

適用於 DynamoDB 的更高階程式設計界面

AWS SDKs 為應用程式提供使用 Amazon DynamoDB 的低階界面。這些用戶端類別和方法直接對應至低階 DynamoDB API。不過，當需要將複雜的資料類型映射至資料庫資料表中的項目時，許多開發人員會遭遇到網路斷線的感覺，或阻抗不符。使用低階資料庫界面，開發人員必須撰寫讀取或寫入物件資

料至資料庫資料表的方法，反之亦然。物件類型和資料庫表格每個組合所需的額外程式碼數量似乎非常大。

為了簡化開發，適用於 Java 和 .NET AWS SDKs 提供額外的介面，具有更高層級的抽象。DynamoDB 的較高階界面可讓您定義程式中的物件與存放這些物件資料的資料庫表格之間的關係。定義此映射之後，您可以呼叫簡單的物件方法 (例如 `save`、`load` 或 `delete`)，也可以代表您自動叫用基礎低階 DynamoDB 操作。這允許您編寫以物件為中心的程式碼，而不是以資料庫為中心的程式碼。

DynamoDB 的高階程式設計界面可在適用於 Java 和 .NET AWS SDKs 中使用。

Java

- Java 1.x：DynamoDBMapper
- Java 2.x：DynamoDB 增強型用戶端

.NET

- 在 DynamoDB 中使用 .NET 文件模型
- 使用 .NET 物件持久性模型和 DynamoDB

Java 1.x：DynamoDBMapper

> **Note**
>
> 適用於 Java 的 SDK 有兩個版本：1.x 和 2.x。1.x 版本已於 2024 年 1 月 12 日宣布終止支援。該版本將於 2025 年 12 月 31 日到期並終止支援。我們強烈建議您在新的開發作業使用 2.x 版本。

適用於 Java 的 AWS SDK 提供 `DynamoDBMapper`類別，可讓您將用戶端類別映射至 Amazon DynamoDB 資料表。若要使用 `DynamoDBMapper`，請定義 DynamoDB 資料表中項目與程式碼中其對應物件執行個體之間的關係。`DynamoDBMapper` 類別也讓您執行各種建立、讀取、更新和對項目進行刪除 (CRUD) 操作，以及執行查詢和掃描資料表。

主題

- DynamoDBMapper Class
- 適用於 Java 的 DynamoDBMapper 支援的資料類型

- 適用於 DynamoDB 的 Java 註釋
- DynamoDBMapper 的選用組態設定
- 含版本編號的 DynamoDB 及樂觀鎖定
- 在 DynamoDB 中映射任意資料
- DynamoDBMapper 範例

> **Note**
>
> DynamoDBMapper 類別不允許您建立、更新或刪除資料表。若要執行這些任務，請改為使用低階適用於 Java 的開發套件介面。

適用於 Java 的開發套件提供一組註釋類型，讓您可以將類別映射至資料表。例如，考量其 Id 為分割區索引鍵的 ProductCatalog 資料表。

```
ProductCatalog(Id, ...)
```

您可以將用戶端應用程式中的類別映射至 ProductCatalog 資料表，如下列 Java 程式碼所示。此程式碼定義名為 CatalogItem 的純舊 Java 物件 (POJO)，而此物件使用註釋將物件欄件映射至 DynamoDB 屬性名稱。

Example

```
package com.amazonaws.codesamples;

import java.util.Set;

import software.amazon.dynamodb.datamodeling.DynamoDBAttribute;
import software.amazon.dynamodb.datamodeling.DynamoDBHashKey;
import software.amazon.dynamodb.datamodeling.DynamoDBIgnore;
import software.amazon.dynamodb.datamodeling.DynamoDBTable;

@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
```

```
    private Set<String> bookAuthors;
    private String someProp;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) {this.id = id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() {return title; }
    public void setTitle(String title) { this.title = title; }

    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN; }

    @DynamoDBAttribute(attributeName="Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
    public String getSomeProp() { return someProp; }
    public void setSomeProp(String someProp) { this.someProp = someProp; }
}
```

在上述程式碼中，@DynamoDBTable 註釋會將 CatalogItem 類別映射至 ProductCatalog 資料表。您可以將個別類別執行個體存放為資料表中的項目。在類別定義中，@DynamoDBHashKey 註釋會將 Id 屬性映射至主索引鍵。

類別屬性預設會映射至資料表中的相同名稱屬性。Title 和 ISBN 屬性會映射至資料表中的相同名稱屬性。

DynamoDB 屬性的名稱符合類別中所宣告屬性的名稱時，@DynamoDBAttribute 註釋是選用項目。它們不同時，請搭配使用此註釋與 attributeName 參數，指定此屬性所對應的 DynamoDB 屬性。

在上述範例中，將 @DynamoDBAttribute 註釋新增至每個屬性，確保屬性名稱完全符合前個步驟中所建立的資料表，並與本指南的其他程式碼範例中所使用的屬性名稱一致。

類別定義可以有未映射至資料表中任何屬性的屬性。您可以新增 @DynamoDBIgnore 註釋來識別這些屬性。在上述範例中，SomeProp 屬性會標上 @DynamoDBIgnore 註釋。當您將 CatalogItem 執行個體上傳至資料表時，DynamoDBMapper 執行個體不會包含 SomeProp 屬性。此外，當您從資料表中擷取項目時，映射器不會傳回此屬性。

在您定義映射類別之後，可以使用 DynamoDBMapper 方法，將該類別的執行個體寫入至 Catalog 資料表中的對應項目。以下程式碼範例會示範此技術。

```
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);
```

下列程式碼範例示範如何擷取項目以及存取它的一些屬性：

```
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 提供直觀且自然的方式來處理 Java 內的 DynamoDB 資料。它也提供數項內建功能；例如，樂觀鎖定、ACID 交易、自動產生的分割區索引鍵和排序索引鍵值，以及物件版本控制。

DynamoDBMapper Class

DynamoDBMapper 類別是 Amazon DynamoDB 的進入點。它提供 DynamoDB 端點的存取，也讓您可以存取多個資料表中的資料。它也讓您執行各種建立、讀取、更新和對項目進行刪除 (CRUD) 操作，以及執行查詢和掃描資料表。此類別提供下列使用 DynamoDB 的方法。

如需對應的 Javadoc 文件，請參閱《適用於 Java 的 AWS SDK API 參考》中的 DynamoDBMapper。

主題

- save
- load
- delete
- query
- queryPage
- 掃描
- scanPage
- parallelScan
- batchSave
- batchLoad
- batchDelete
- batchWrite
- transactionWrite
- transactionLoad
- count
- generateCreateTableRequest
- createS3Link
- getS3ClientCache

save

將指定的物件儲存至資料表。您要儲存的物件是此方法的唯一必要參數。您可以使用 `DynamoDBMapperConfig` 物件來提供選用的組態參數。

如果具有相同主索引鍵的項目不存在，則此方法會在資料表中建立新的項目。如果具有相同主索引鍵的項目存在，則會更新現有項目。如果分割區索引鍵和排序索引鍵的類型是字串，並且標註 `@DynamoDBAutoGeneratedKey`，則會將隨機全域唯一識別符 (UUID) 授予它們 (若未初始化)。標註 `@DynamoDBVersionAttribute` 的版本欄位會遞增一。此外，如果更新版本欄位或產生索引鍵，則會因這項操作而更新傳入的物件。

根據預設，只會更新對應至已映射類別屬性的屬性。項目上的任何其他現有屬性則不受影響。不過，如果您指定 `SaveBehavior.CLOBBER`，則可以強制完全覆寫項目。

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER).build();

mapper.save(item, config);
```

如果您已啟用版本控制，則用戶端與伺服器端項目版本必須相符。不過，如果使用 SaveBehavior.CLOBBER 選項，則版本不需要相符。如需版本控制的詳細資訊，請參閱「含版本編號的 DynamoDB 及樂觀鎖定」。

load

從資料表擷取項目。您必須提供要擷取之項目的主索引鍵。您可以使用 DynamoDBMapperConfig 物件來提供選用的組態參數。例如，您可以選擇性地請求高度一致性讀取，確保此方法只擷取最新的項目數值，如下列 Java 陳述式所示。

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT).build();

CatalogItem item = mapper.load(CatalogItem.class, item.getId(), config);
```

DynamoDB 預設會傳回數值最終一致的項目。如需 DynamoDB 最終一致性模式的資訊，請參閱 DynamoDB 讀取一致性。

delete

刪除資料表中的項目。您必須傳入已映射類別的物件執行個體。

如果您已啟用版本控制，則用戶端與伺服器端項目版本必須相符。不過，如果使用 SaveBehavior.CLOBBER 選項，則版本不需要相符。如需版本控制的詳細資訊，請參閱「含版本編號的 DynamoDB 及樂觀鎖定」。

query

查詢資料表或次要索引。

假設您有一個存放論壇主題回覆的 Reply 資料表。每個對話主旨都可以有零個以上的回覆。Reply 資料表的主索引鍵包含 Id 和 ReplyDateTime 欄位；其中，Id 是分割區索引鍵，而 ReplyDateTime 是主索引鍵的排序索引鍵。

```
Reply ( Id, ReplyDateTime, ... )
```

假設您已建立 Reply 類別與 DynamoDB 中對應 Reply 資料表之間的映射。下列 Java 程式碼使用 DynamoDBMapper 來尋找特定對話主旨在過去兩週的所有回覆。

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);
```

查詢會傳回 Reply 物件的集合。

query 方法預設會傳回「延遲載入」集合。它一開始只會傳回一頁的結果，然後視需要進行服務呼叫來取得下一頁。若要取得所有相符的項目，請逐一查看 latestReplies 集合。

請注意，在集合上呼叫 size() 方法將會載入所有結果，以提供準確的計數。這可能會導致耗用大量已佈建的輸送量，而在非常大型的資料表上可能甚至會耗盡 JVM 中的所有記憶體。

若要查詢索引，您必須先將索引建模為映射器類別。假設 Reply 資料表有一個名為 PostedBy-Message-Index 的全域次要索引。此索引的分割區索引鍵是 PostedBy，而排序索引鍵是 Message。索引中項目的類別定義將會如下。

```
@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.
}
```

@DynamoDBTable 註釋指出此索引與 Reply 資料表建立關聯。@DynamoDBIndexHashKey 註釋表示索引的分割區索引鍵 (PostedBy)，而 @DynamoDBIndexRangeKey 表示索引的排序索引鍵 (Message)。

您現在可以使用 DynamoDBMapper 來查詢索引，並擷取特定使用者所張貼的訊息子集。如果資料表和索引之間沒有衝突的映射，且映射器中已建立映射，則不需要指定索引名稱。映射器將根據主索引鍵和排序索引鍵推論。下列程式碼會查詢全域次要索引。因為全域次要索引支援最終一致讀取，但不支援高度一致性讀取，所以必須指定 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);
```

查詢會傳回 PostedByMessage 物件的集合。

queryPage

查詢資料表或次要索引，並傳回單頁的相符結果。與使用 query 方法相同，您必須指定分割區索引鍵值以及套用至排序索引鍵屬性的查詢篩選條件。不過，queryPage 只會傳回第一「頁」的資料；亦即，符合 1 MB 的資料量

掃描

掃描整個資料表或次要索引。您可以選擇性地指定 FilterExpression 來篩選結果集。

假設您有一個存放論壇主題回覆的 Reply 資料表。每個對話主旨都可以有零個以上的回覆。Reply 資料表的主索引鍵包含 Id 和 ReplyDateTime 欄位；其中，Id 是分割區索引鍵，而 ReplyDateTime 是主索引鍵的排序索引鍵。

```
Reply ( Id, ReplyDateTime, ... )
```

如果您已將 Java 類別映射至 Reply 資料表，則可以使用 DynamoDBMapper 來掃描資料表。例如，下列 Java 程式碼會掃描整個 Reply 資料表，而且只會傳回特定一年的回覆。

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);
```

scan 方法預設會傳回「延遲載入」集合。它一開始只會傳回一頁的結果，然後視需要進行服務呼叫來取得下一頁。若要取得所有相符的項目，請逐一查看 replies 集合。

請注意，在集合上呼叫 size() 方法將會載入所有結果，以提供準確的計數。這可能會導致耗用大量已佈建的輸送量，而在非常大型的資料表上可能甚至會耗盡 JVM 中的所有記憶體。

若要掃描索引，您必須先將索引建模為映射器類別。假設 Reply 資料表有一個名為 PostedBy-Message-Index 的全域次要索引。此索引的分割區索引鍵是 PostedBy，而排序索引鍵是 Message。此索引的映射器類別顯示在 query 區段。它使用 @DynamoDBIndexHashKey 和 @DynamoDBIndexRangeKey 註釋來指定索引的分割區索引鍵和排序索引鍵。

下列程式碼範例掃描 PostedBy-Message-Index。它不會使用掃描篩選條件，因此會將索引中的所有項目都傳回給您。

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false);
```

```
    List<PostedByMessage> iList =  mapper.scan(PostedByMessage.class, scanExpression);
    Iterator<PostedByMessage> indexItems = iList.iterator();
```

scanPage

掃描資料表或次要索引，並傳回單頁的相符結果。與使用 scan 方法相同，您可以選擇性地指定 FilterExpression 來篩選結果集。不過，scanPage 只會傳回第一「頁」的資料；亦即，符合 1 MB 內的資料量。

parallelScan

執行整個資料表或次要索引的平行掃描。您可以指定資料表的一些邏輯區段，以及掃描表達式來篩選結果。parallelScan 會將掃描任務分到多個工作者 (一個邏輯區段一個工作者)；工作者會平行處理資料，並傳回結果。

下列 Java 程式碼範例會對 Product 資料表執行平行掃描。

```
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

使用 AmazonDynamoDB.batchWriteItem 方法的一或多個呼叫，將物件儲存至一或多個資料表。此方法不提供交易保證。

下列 Java 程式碼會將兩個項目 (書籍) 儲存至 ProductCatalog 資料表。

```
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

使用一或多個資料表的主索引鍵，以從中擷取多個項目。

下列 Java 程式碼會從兩個不同的資料表擷取兩個項目。

```
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

使用 AmazonDynamoDB.batchWriteItem 方法的一或多個呼叫，刪除一或多個資料表中的物件。此方法不提供交易保證。

下列 Java 程式碼會刪除 ProductCatalog 資料表中的兩個項目 (書籍)。

```
Book book1 = mapper.load(Book.class, 901);
Book book2 = mapper.load(Book.class, 902);
mapper.batchDelete(Arrays.asList(book1, book2));
```

batchWrite

使用 AmazonDynamoDB.batchWriteItem 方法的一或多個呼叫，將物件儲存至一或多個資料表，並刪除物件。此方法未提供交易保證或支援版本控制 (條件式放置或刪除)。

下列 Java 程式碼會將新項目寫入至 Forum 資料表、將新項目寫入至 Thread 資料表，並刪除 ProductCatalog 資料表中的項目。

```
// 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

使用 AmazonDynamoDB.transactWriteItems 方法的一個呼叫，將物件儲存至一或多個資料表，並從中刪除物件。

如需交易特定例外的清單，請參閱 TransactWriteItems 錯誤。

如需 DynamoDB 交易與提供之不可部分完成性、一致性、隔離性和耐久性 (ACID) 保證的相關資訊，請參閱 Amazon DynamoDB Transactions。

> **Note**
>
> 此方法不支援下列項目：
>
> - DynamoDBMapperConfig.SaveBehavior。

以下 Java 程式碼會以交易的方式，將新項目寫入每個 Forum 和 Thread 資料表。

```
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

使用 AmazonDynamoDB.transactGetItems 方法的一個呼叫，載入一或多個資料表中的物件。

如需交易特定例外的清單，請參閱 TransactGetItems 錯誤。

如需 DynamoDB 交易與提供之不可部分完成性、一致性、隔離性和耐久性 (ACID) 保證的相關資訊，請參閱 Amazon DynamoDB Transactions。

以下 Java 程式碼會以交易的方式，將從每個 Forum 和 Thread 資料表載入一個項目。

```
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

評估指定的掃描表達式，並傳回相符項目的計數。不傳回任何項目資料。

generateCreateTableRequest

剖析代表 DynamoDB 資料表的 POJO 類別，並傳回該資料表的 CreateTableRequest。

createS3Link

建立 Amazon S3 中物件的連結。您必須指定儲存貯體名稱和索引鍵名稱，以唯一識別儲存貯體中的物件。

若要使用 `createS3Link`，映射器類別必須定義 getter 和 setter 方法。下列程式碼範例透過將新的屬性和 getter/setter 方法新增至 `CatalogItem` 類別來說明這種情況。

```
@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;
    }

...
}
```

下列 Java 程式碼定義要寫入至 `Product` 資料表的新項目。此項目包含產品映像的連結；映像資料會上傳至 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);
```

`S3Link` 類別提供許多其他方法來操控 Amazon S3 中的物件。如需詳細資訊，請參閱 Javadocs for `S3Link`。

getS3ClientCache

傳回基礎 S3ClientCache 來存取 Amazon S3。S3ClientCache 是 AmazonS3Client 物件的智慧映射。如果您有多個用戶端， S3ClientCache可協助您依 AWS 區域整理用戶端，並可隨需建立新的 Amazon S3 用戶端。

適用於 Java 的 DynamoDBMapper 支援的資料類型

本節說明 Amazon DynamoDB 中支援的基本 Java 資料類型、集合和任意資料類型。

Amazon DynamoDB 支援下列基本 Java 資料類型和基本包裝函式類別。

- `String`
- `Boolean`, `boolean`
- `Byte`, `byte`
- `Date` (作為 ISO_8601 毫秒精確性字串，已轉移為 UTC)
- `Calendar` (作為 ISO_8601 毫秒精確性字串，已轉移為 UTC)
- `Long`, `long`
- `Integer`, `int`
- `Double`, `double`
- `Float`, `float`
- `BigDecimal`
- `BigInteger`

> **Note**
>
> - 如需有關 DynamoDB 命名規則和各種支援之資料類型的詳細資訊，請參閱 Amazon DynamoDB 中支援的資料類型和命名規則。
> - 空白的二進位值由 DynamoDBMapper 支援。
> - AWS SDK for Java 2.x支援空白字串值。
>
>   在適用於 Java 的 AWS SDK 1.x 中，DynamoDBMapper 支援讀取空的字串屬性值，但它不會寫入空的字串屬性值，因為這些屬性會從請求中刪除。

DynamoDB 支援 Java Set、List 和 Map 集合類型。下表摘要說明如何將這些 Java 類型映射至 DynamoDB 類型。

Java 類型	DynamoDB 類型
所有數字類型	N (數字類型)
Strings	S (字串類型)
Boolean	BOOL (布林類型)，0 或 1。
ByteBuffer	B (二進位類型)
Date	S (字串類型)。Date 值會以 ISO-8601 格式字串存放。
Set 集合類型	SS (字串集) 類型、NS (數字集) 類型或 BS (二進位集) 類型。

`DynamoDBTypeConverter` 介面可讓您將自己的任意資料類型映射至 DynamoDB 原生支援的資料類型。如需詳細資訊，請參閱在 DynamoDB 中映射任意資料。

適用於 DynamoDB 的 Java 註釋

本節說明可用於將類別和屬性映射至 Amazon DynamoDB 中資料表和屬性的註釋。

如需對應的 Javadoc 文件，請參閱《適用於 Java 的 AWS SDK API 參考》中的註釋類型摘要。

> **Note**
>
> 在下列註釋中，只需要 `DynamoDBTable` 和 `DynamoDBHashKey`。

主題

- DynamoDBAttribute
- DynamoDBAutoGeneratedKey
- DynamoDBAutoGeneratedTimestamp
- DynamoDBDocument

- DynamoDBHashKey
- DynamoDBIgnore
- DynamoDBIndexHashKey
- DynamoDBIndexRangeKey
- DynamoDBRangeKey
- DynamoDBTable
- DynamoDBTypeConverted
- DynamoDBTyped
- DynamoDBVersionAttribute

DynamoDBAttribute

將屬性映射至資料表屬性。每個類別屬性預設會映射至同名的項目屬性。不過，如果名稱不同，您可以使用此註釋將屬性 (property) 映射至屬性 (attribute)。在下列 Java 程式碼片段中，`DynamoDBAttribute` 會將 `BookAuthors` 屬性映射至資料表中的 `Authors` 屬性名稱。

```
@DynamoDBAttribute(attributeName = "Authors")
public List<String> getBookAuthors() { return BookAuthors; }
public void setBookAuthors(List<String> BookAuthors) { this.BookAuthors =
 BookAuthors; }
```

將物件儲存至資料表時，`DynamoDBMapper` 使用 `Authors` 做為屬性名稱。

DynamoDBAutoGeneratedKey

將分割區索引鍵或排序索引鍵屬性標示為自動產生。儲存這些屬性時，`DynamoDBMapper` 會產生隨機 UUID。只有字串屬性才能標示為自動產生的索引鍵。

以下是示範使用自動產生索引鍵參數的範例。

```
@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());
    }
}
```

DynamoDBAutoGeneratedTimestamp

自動產生時間戳記。

```
@DynamoDBAutoGeneratedTimestamp(strategy=DynamoDBAutoGenerateStrategy.ALWAYS)
public Date getLastUpdatedDate() { return lastUpdatedDate; }
public void setLastUpdatedDate(Date lastUpdatedDate) { this.lastUpdatedDate =
 lastUpdatedDate; }
```

或者，可以透過提供策略屬性來定義自動產生策略。預設值為 ALWAYS。

DynamoDBDocument

指出類別可以序列化為 Amazon DynamoDB 文件。

例如，假設您要將 JSON 文件映射至 Map (M) 類型的 DynamoDB 屬性。下列程式碼範例定義包含 Map 類型之巢狀屬性 (Pictures) 的項目。

```
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; }

     }
}
```

然後您就可儲存新的 ProductCatalog 項目，並具有 Pictures，如下列範例所示。

```
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);
```

產生的 ProductCatalog項目看起來將會如下 (JSON 格式)。

```
{
  "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"
  }
}
```

DynamoDBHashKey

將類別屬性映射至資料表的分割區索引鍵。此屬性必須是純量字串、數字或二進位類型的其中之一。此屬性不能是集合類型。

假設您的 ProductCatalog 資料表將 Id 作為主索引鍵。下列 Java 程式碼定義 CatalogItem 類別，並使用 @DynamoDBHashKey 標籤將其 Id 屬性映射至 ProductCatalog 資料表的主索引鍵。

```
@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.
}
```

DynamoDBIgnore

指出應該忽略相關聯屬性的 DynamoDBMapper 執行個體。將資料儲存至資料表時，DynamoDBMapper 不會將此屬性儲存至資料表。

套用至 getter 方法或非模組化屬性的類別欄位。若該註釋是直接套用至類別欄位，則必須在相同的類別宣告相對應的 getter 和 setter。

DynamoDBIndexHashKey

將類別屬性映射至全域次要索引的分割區索引鍵。此屬性必須是純量字串、數字或二進位類型的其中之一。此屬性不能是集合類型。

如果您需要對全域次要索引進行 Query，請使用此註釋。您必須指定索引名稱 (globalSecondaryIndexName)。如果類別屬性的名稱與索引的分割區索引鍵不同，您也必須指定該索引屬性的名稱 (attributeName)。

DynamoDBIndexRangeKey

將類別屬性映射至全域次要索引或本機次要索引的排序索引鍵。此屬性必須是純量字串、數字或二進位類型的其中之一。此屬性不能是集合類型。

如果您需要對本機次要索引或全域次要索引進行 Query，並且想要使用索引的排序索引鍵來縮小您結果的範圍，請使用此註釋。您必須指定索引名稱 (globalSecondaryIndexName 或 localSecondaryIndexName)。如果類別屬性的名稱與索引的排序索引鍵不同，您也必須指定該索引屬性的名稱 (attributeName)。

DynamoDBRangeKey

將類別屬性映射至資料表的排序索引鍵。此屬性必須是純量字串、數字或二進位類型的其中之一。它不能是集合類型。

如果主索引鍵是複合 (分割區索引鍵和排序索引鍵)，您可以使用此標籤，將類別欄位映射至排序索引鍵。例如，假設您的 Reply 資料表存放論壇主題回覆。每個對話都可以有許多回覆。因此，此資料表的主索引鍵是 ThreadId 和 ReplyDateTime。分割區索引鍵是 ThreadId，而排序索引鍵是 ReplyDateTime。

下列 Java 程式碼定義 Reply 類別，並將它映射至 Reply 資料表。它會使用 @DynamoDBHashKey 和 @DynamoDBRangeKey 標籤來識別映射至主索引鍵的類別屬性。

```
@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.
}
```

DynamoDBTable

識別 DynamoDB 中的目標資料表。下列 Java 程式碼定義 Developer 類別，並將它映射至 DynamoDB 中的 People 資料表。

```
@DynamoDBTable(tableName="People")
public class Developer { ...}
```

@DynamoDBTable 註釋可以予以繼承。任何繼承自 Developer 類別的新類別也會映射至 People 資料表。例如，假設您建立繼承自 Lead 類別的 Developer 類別。因為您已將 Developer 類別映射至 People 資料表，所以也會將 Lead 類別物件存放至相同的資料表。

@DynamoDBTable 也可以予以覆寫。任何繼承自 Developer 類別的新類別預設會映射至相同的 People 資料表。不過，您可以覆寫此預設映射。例如，如果您建立繼承自 Developer 類別的類別，則可以新增 @DynamoDBTable 註釋，明確地將它映射至另一個資料表，如下列 Java 程式碼範例所示。

```
@DynamoDBTable(tableName="Managers")
public class Manager extends Developer { ...}
```

DynamoDBTypeConverted

將屬性標示為使用自訂類型轉換器的註釋。可以標註於使用者定義的註釋，以將其他屬性傳遞給 DynamoDBTypeConverter。

DynamoDBTypeConverter 介面可讓您將自己的任意資料類型映射至 DynamoDB 原生支援的資料類型。如需詳細資訊，請參閱在 DynamoDB 中映射任意資料。

DynamoDBTyped

覆寫標準屬性類型繫結的註釋。如果套用標準類型的預設屬性繫結，則該類型不需要註釋。

DynamoDBVersionAttribute

識別用於存放樂觀鎖定版本編號的類別屬性。DynamoDBMapper 在儲存新的項目時會將版本編號指派給此屬性，並在每次您更新項目時予以遞增。僅支援數字純量類型。如需資料類型的詳細資訊，請參閱資料類型。如需版本控制的詳細資訊，請參閱「含版本編號的 DynamoDB 及樂觀鎖定」。

DynamoDBMapper 的選用組態設定

當您建立 DynamoDBMapper 執行個體時，會有特定預設行為；您可以使用 DynamoDBMapperConfig 類別來覆寫這些預設值。

下列程式碼片段會使用自訂設定來建立 DynamoDBMapper：

```
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);
```

如需詳細資訊，請參閱《適用於 Java 的 AWS SDK API 參考》中的 DynamoDBMapperConfig。

您可以針對 DynamoDBMapperConfig 執行個體使用下列引數：

- DynamoDBMapperConfig.ConsistentReads 列舉值：
  - EVENTUAL：映射器執行個體使用最終一致讀取請求。
  - CONSISTENT：映射器執行個體使用高度一致性讀取請求。您可以搭配使用這個選用設定與 load、query 或 scan 操作。高度一致性讀取具有效能和帳單隱憂；如需詳細資訊，請參閱 DynamoDB 產品詳細資訊頁面。

  如果您未指定映射器執行個體的讀取一致性設定，則預設值為 EVENTUAL。

> **Note**
>
> 此值僅適用於 DynamoDBMapper 的 query、querypage、load 和 batch load 操作。

- DynamoDBMapperConfig.PaginationLoadingStrategy 列舉值：控制映射器執行個體如何處理分頁資料清單，例如 query 或 scan 的結果：
  - LAZY_LOADING：映射器執行個體會在可能時載入資料，並將所有載入的結果保留在記憶體中。
  - EAGER_LOADING：映射器執行個體會在初始化清單時立即載入資料。
  - ITERATION_ONLY：您只能使用從清單中讀取的迭代器。在迭代期間，清單會先清除所有先前的結果，再載入下一頁，因此清單最多會將一頁的已載入結果保留在記憶體中。這也表示清單只能重複使用一次。處理大型項目時，建議使用此策略，以減少記憶體負擔。

  如果您未指定映射器執行個體的分頁載入策略，則預設值為 LAZY_LOADING。
- DynamoDBMapperConfig.SaveBehavior 列舉值：指定映射器執行個體在儲存操作期間應該如何處理屬性：
  - UPDATE：在儲存操作期間，會更新所有建模屬性，未建模屬性則不受影響。基本數字類型 (byte、int、long) 設定為 0。物件類型設定為 Null。
  - CLOBBER：在儲存操作期間，清除和取代所有屬性 (包含未建模屬性)。透過刪除並重新建立項目來完成這項操作。使用版本控制的欄位限制條件也會予以忽略。

  如果您未指定映射器執行個體的儲存行為，則預設值為 UPDATE。

  > **Note**
  >
  > DynamoDBMapper 交易操作不支援 DynamoDBMapperConfig.SaveBehavior 列舉。

- DynamoDBMapperConfig.TableNameOverride 物件：指示映射器執行個體忽略類別之 DynamoDBTable 註釋所指定的資料表名稱，並改為使用您提供的不同資料表名稱。在執行時期將資料分割為多個資料表時，這十分有用。

您可以視需要覆寫每個操作的 DynamoDBMapper 預設組態物件。

含版本編號的 DynamoDB 及樂觀鎖定

樂觀鎖定是一種策略，確保您要更新 (或刪除) 的用戶端項目與 Amazon DynamoDB 中的項目相同。如果您使用此策略，則會保護其他項目的寫入不會覆寫資料庫寫入，反之亦然。

使用樂觀鎖定，每個項目都有做為版本編號的屬性。如果您從資料表中擷取項目，則應用程式會記錄該項目的版本編號。您可以更新項目，但只有在伺服器端的版本編號尚未變更時。若版本不相符，表示已有其他人在您之前進行修改。更新嘗試失敗，因為您有此項目的過期版本。如果發生這種情況，請檢索

項目然後嘗試進行更新以重試一次。樂觀鎖定預防您意外覆寫其他人所作的變更。同時也預防其他人意外覆寫您所作的變更。

雖然您可以實作自己的樂觀鎖定策略，但 適用於 Java 的 AWS SDK 提供 `@DynamoDBVersionAttribute`註釋。在資料表的映射類別中，您指定一個屬性來存放版本編號，並使用此註釋進行標示。當您儲存物件時，DynamoDB 資料表中的對應項目會有屬性可存放版本編號。當您第一次儲存物件時，`DynamoDBMapper` 會指派版本編號，並在每次更新項目時自動遞增版本編號。只有在用戶端物件版本與 DynamoDB 資料表中項目的對應版本編號相符時，您的更新或刪除請求才會成功。

如果發生下列情況，則會擲回 `ConditionalCheckFailedException`：

- 您使用含 `@DynamoDBVersionAttribute` 的樂觀鎖定，而且伺服器上的版本值與用戶端上的值不同。
- 您可以搭配使用 `DynamoDBMapper` 與 `DynamoDBSaveExpression`，以在儲存資料時指定自己的條件式限制條件，而且這些限制條件會失敗。

> **Note**
>
> - DynamoDB 全域資料表會在並行更新間使用「最後寫入者獲勝」核對機制。如果您使用的是全域資料表，最後寫入者政策獲勝。因此，在此情況下，鎖定政策不會如預期般運作。
> - `DynamoDBMapper` 交易寫入操作不支援同一物件的 `@DynamoDBVersionAttribute` 標註和條件表達式。如果交易寫入內的物件以 `@DynamoDBVersionAttribute` 標註，且具有條件表達式，則會拋出 SdkClientException。

例如，下列 Java 程式碼定義有數個屬性的 `CatalogItem` 類別。`Version` 屬性會標上 `@DynamoDBVersionAttribute` 註釋。

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;}
}
```

您可以將 @DynamoDBVersionAttribute 註釋套用至基本包裝函式類別所提供的可為 Null 類型，而基本包裝函式類別提供可為 Null 類型 (例如 Long 和 Integer)。

樂觀鎖定對這些 DynamoDBMapper 方法的影響如下：

- save：針對新項目，DynamoDBMapper 會指派初始版本編號 1。如果您擷取項目，請更新其一或多個屬性並嘗試儲存變更，只有在用戶端的版本編號與伺服器端的相符時，儲存操作才會成功。DynamoDBMapper 會自動遞增版本編號。
- delete：delete 方法會採用物件作為參數，而 DynamoDBMapper 會在刪除項目之前執行版本檢查。如果請求中指定 DynamoDBMapperConfig.SaveBehavior.CLOBBER，則可以停用版本檢查。

  DynamoDBMapper 內的樂觀鎖定內部實作，使用 DynamoDB 所提供的條件式更新和條件式刪除支援。

- `transactionWrite` —
  - `Put`：針對新項目，`DynamoDBMapper` 會指派初始版本編號 1。如果您擷取項目，請更新其一或多個屬性並嘗試儲存變更，只有在用戶端的版本編號與伺服器端的相符時，Put 操作才會成功。`DynamoDBMapper` 會自動遞增版本編號。
  - `Update`：針對新項目，`DynamoDBMapper` 會指派初始版本編號 1。如果您擷取項目，請更新其一或多個屬性並嘗試儲存變更，只有在用戶端的版本編號與伺服器端的相符時，update 操作才會成功。`DynamoDBMapper` 會自動遞增版本編號。
  - `Delete`：`DynamoDBMapper` 會在刪除項目之前執行版本檢查。只有用戶端與伺服器端的版本號碼相符時，delete 操作才會成功。
  - `ConditionCheck`：不支援 `ConditionCheck` 操作的 `@DynamoDBVersionAttribute` 註釋。以 `@DynamoDBVersionAttribute` 標註 `ConditionCheck` 項目時，會拋出 SdkClientException。

停用樂觀鎖定

若要停用樂觀鎖定，您可以將 `DynamoDBMapperConfig.SaveBehavior` 列舉值從 `UPDATE` 變更為 `CLOBBER`。您可以建立跳過版本檢查的 `DynamoDBMapperConfig` 執行個體來執行這項操作，並將此執行個體用於所有請求。如需 `DynamoDBMapperConfig.SaveBehavior` 和其他選用 `DynamoDBMapper` 參數的資訊，請參閱「DynamoDBMapper 的選用組態設定」。

您也只能設定特定操作的鎖定行為。例如，下列 Java 程式碼片段使用 `DynamoDBMapper` 來儲存型錄項目。指定 `DynamoDBMapperConfig.SaveBehavior` 的方式是將選用 `DynamoDBMapperConfig` 參數新增至 `save` 方法。

> **Note**
>
> transactionWrite 方法不支援 DynamoDBMapperConfig.SaveBehavior 組態。不支援停用 transactionWrite 的樂觀鎖定。

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));
```

在 DynamoDB 中映射任意資料

除了支援的 Java 類型之外 (請參閱 適用於 Java 的 DynamoDBMapper 支援的資料類型)，您還可以使用應用程式中未直接映射至 Amazon DynamoDB 類型的類型。若要映射這些類型，您必須提供將複雜類型轉換為 DynamoDB 支援之類型的實作 (反之亦然)，並使用 `@DynamoDBTypeConverted` 註釋來標註複雜類型存取子方法。轉換器程式碼會在儲存或載入物件時轉換資料。它也用於使用複雜類型的所有操作。請注意，在查詢和掃描操作期間比較資料時，會針對 DynamoDB 中所存放的資料進行比較。

例如，請考慮下列可定義本身為 `CatalogItem` 之 `Dimension` 屬性的 `DimensionType` 類別。此屬性會將項目維度存放為高度、寬度和厚度。假設您決定將這些項目維度存放為 DynamoDB 中的字串 (例如 8.5x11x.05)。下列範例提供轉換器程式碼，以將 `DimensionType` 物件轉換為字串以及將字串轉換為 `DimensionType`。

> **Note**
>
> 此程式碼範例假設您已根據 在 DynamoDB 中建立資料表，以及載入程式碼範例的資料 一節的說明將資料載入 DynamoDB 的帳戶。
> 如需執行下列範例的逐步說明，請參閱「Java 程式碼範例」。

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;
        }
    }
}
```

DynamoDBMapper 範例

適用於 Java 的 AWS SDK 提供 DynamoDBMapper 類別，可讓您將用戶端類別映射至 DynamoDB 資料表。若要使用 DynamoDBMapper，請定義 DynamoDB 資料表中項目與程式碼中其對應物件執行個體之間的關係。DynamoDBMapper 類別也讓您執行各種建立、讀取、更新和對項目進行刪除 (CRUD) 操作，以及執行查詢和掃描資料表。

若要進一步了解如何使用 DynamoDBMapper，請參閱《適用於 Java 1.x 的 AWS SDK 開發人員指南》中使用適用於 Java 的 AWS SDK 的 DynamoDB 範例。

Java 2.x：DynamoDB 增強型用戶端

DynamoDB 增強型用戶端是適用於 Java 的 AWS SDK 2 (v2) 版中的高階程式庫。提供將客戶端類別映射到 DynamoDB 資料表的直接方式。您要在自己的程式碼中定義資料表及其對應模型類別之間的關係。定義關係之後，您可以直觀地對 DynamoDB 中的資料表或項目執行各種建立、讀取、更新或刪除 (CRUD) 操作。

如需有關如何將增強型用戶端與 DynamoDB 搭配使用的詳細資訊，請參閱適用於 Java 的 AWS SDK 2.x 中的使用 DynamoDB 增強型用戶端。

在 DynamoDB 中使用 .NET 文件模型

適用於 .NET 的 AWS SDK 提供文件模型類別，包裝一部分低階 Amazon DynamoDB 操作，可進一步簡化您的程式碼編寫。在文件模型中，主類別為 `Table` 和 `Document`。`Table` 類別提供資料操作方法，例如 `PutItem`、`GetItem` 和 `DeleteItem`。該類別也提供 `Query` 和 `Scan` 方法。`Document` 類別則代表資料表中的單一項目。

上述文件模型類別可在 `Amazon.DynamoDBv2.DocumentModel` 命名空間中取得。

> **Note**
>
> 您無法使用文件模型建立、更新和刪除資料表。但文件模型支援大部分的常見資料操作。

主題

- 支援的資料類型

支援的資料類型

文件模型會支援一組原始的 .NET 資料類型和集合資料類型。模型目前支援下列基本資料類型。

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`

- `float`
- `Guid`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

下表摘要說明上述 .NET 類型與 DynamoDB 類型的映射。

.NET 基本類型	DynamoDB 類型
所有數字類型	N (數字類型)
所有字串類型	S (字串類型)
MemoryStream, byte[]	B (二進位類型)
bool	N (數字類型)。0 代表 false，1 代表 true。
DateTime	S (字串類型)。`DateTime` 值會以 ISO-8601 格式字串存放。
Guid	S (字串類型)。
集合類型 (List、HashSet 和陣列)	BS (二進位集) 類型、SS (字串集) 類型和 NS (數字集) 類型。

適用於 .NET 的 AWS SDK 會定義將 DynamoDB 的布林值、null 值、清單和映射類型映射到 .NET 文件模型 API 的類型：

- 使用 `DynamoDBBool` 作為布林類型。

- 使用 DynamoDBNull 作為 null 類型。
- 使用 DynamoDBList 作為清單類型。
- 使用 Document 作為映射類型。

> **Note**
>
> - 支援空白的二進位值。
> - 支援讀取空白字串值。寫入 DynamoDB 時，字串 Set 類型的屬性值會支援空白字串的屬性值。List 或 Map 類型中包含的字串類型的空白字串屬性值和空白字串值會從寫入請求中捨棄

使用 .NET 物件持久性模型和 DynamoDB

適用於 .NET 的 AWS SDK 提供物件持久性模型，可讓您將用戶端類別映射至 Amazon DynamoDB 資料表。然後每個物件執行個體會映射至相對應資料表中的某個項目。為了將用戶端物件儲存至資料表，物件持久性模型會提供 DynamoDBContext 類別 (即 DynamoDB 的進入點)。此類別可讓您連線至 DynamoDB，繼而存取資料表、執行各種 CRUD 操作以及執行查詢。

物件持久性模型提供了一組屬性，可將用戶端類別映射到資料表，並將屬性/欄位映射至資料表屬性。

> **Note**
>
> 物件持久性模型不提供用於建立、更新或刪除資料表的 API。它只提供資料操作。您只能使用適用於 .NET 的 AWS SDK 低階 API 來建立、更新和刪除資料表。

以下範例會示範物件持久性模型的運作方式。它會從 ProductCatalog 資料表開始作業。它將 Id 作為主索引鍵。

```
ProductCatalog(Id, ...)
```

假設您的 Book 類別具有 Title、ISBN 以及 Authors 屬性。您可以透過新增物件持久性模型所定義的屬性，將 Book 類別映射至 ProductCatalog 資料表中，如下列 C# 程式碼範例所示。

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; }
}
```

在上述範例中，DynamoDBTable 屬性會將 Book 類別映射至 ProductCatalog 資料表。

物件持久性模型支援類別屬性與資料表屬性之間的明確映射和預設映射。

- 明確映射：若要將屬性映射至主索引鍵，您必須使用 DynamoDBHashKey 和 DynamoDBRangeKey 物件持久性模型屬性。此外，對於非主索引鍵屬性，如果類別中的屬性名稱和要對應的資料表屬性不相同，則必須明確新增 DynamoDBProperty 屬性來定義映射。

  在上述範例中，Id 屬性會映射至具有相同名稱的主索引鍵，BookAuthors 屬性則會映射至 ProductCatalog 資料表中的 Authors 屬性。
- 預設映射：依預設，物件持久性模型會將類別屬性映射至資料表中具有相同名稱的屬性。

  在上述範例中，屬性 Title 和 ISBN 會映射至 ProductCatalog 資料表中具有相同名稱的屬性。

您不必映射每個類別屬性。您可以新增 DynamoDBIgnore 屬性來識別這些屬性。當您將 Book 執行個體儲存至資料表時，DynamoDBContext 不會包含 CoverPage 屬性。在擷取書籍執行個體時，它也不會傳回此屬性。

您可以映射 .NET 基本類型的屬性，如 int 和字串。只要提供適當的轉換器將任意資料映射至其中一種 DynamoDB 類型，您也可以映射任意資料類型。若要進一步了解如何映射任意類型，請參閱 使用 適用於 .NET 的 AWS SDK 物件持久性模型使用 DynamoDB 映射任意資料。

物件持久性模型支援樂觀鎖定。在更新操作期間，這可確保您擁有即將更新項目的最新副本。如需詳細資訊，請參閱使用 DynamoDB 和 適用於 .NET 的 AWS SDK 物件持久性模型的樂觀鎖定。

如需詳細資訊，請參閱下列主題。

主題

- 支援的資料類型
- .NET 物件持久性模型的 DynamoDB 屬性
- .NET 物件持久性模型的 DynamoDBContext 類別
- 使用 DynamoDB 和 適用於 .NET 的 AWS SDK 物件持久性模型的樂觀鎖定
- 使用 適用於 .NET 的 AWS SDK 物件持久性模型使用 DynamoDB 映射任意資料

支援的資料類型

物件持久性模型會支援一組基本的 .NET 資料類型、集合和任意資料類型。模型目前支援下列基本資料類型。

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

物件持久性模型也支援 .NET 集合類型。`DynamoDBContext` 能夠轉換具體的集合類型和簡單的純舊 CLR 物件 (POCO)。

下表摘要說明上述 .NET 類型與 DynamoDB 類型的映射。

.NET 基本類型	DynamoDB 類型
所有數字類型	N (數字類型)
所有字串類型	S (字串類型)
MemoryStream, byte[]	B (二進位類型)
bool	N (數字類型)。0 代表 false，1 代表 true。
集合類型	BS (二進位集) 類型、SS (字串集) 類型和 NS (數字集) 類型。
DateTime	S (字串類型)。DateTime 值會以 ISO-8601 格式字串存放。

物件持久性模型也支援任意資料類型。不過，您必須提供轉換器程式碼，才能將複雜類型映射至 DynamoDB 類型。

> **Note**
>
> - 支援空白的二進位值。
> - 支援讀取空白字串值。寫入 DynamoDB 時，字串 Set 類型的屬性值會支援空白字串的屬性值。List 或 Map 類型中包含的字串類型的空白字串屬性值和空白字串值會從寫入請求中捨棄

.NET 物件持久性模型的 DynamoDB 屬性

本節說明物件持久性模型所提供的屬性，讓您可以將類別和屬性映射至 DynamoDB 資料表和屬性。

> **Note**
>
> 在下列屬性中，只有 DynamoDBTable 和 DynamoDBHashKey 是必要屬性。

DynamoDBGlobalSecondaryIndexHashKey

將類別屬性映射至全域次要索引的分割區索引鍵。如果您需要對全域次要索引進行 Query，請使用此屬性。

DynamoDBGlobalSecondaryIndexRangeKey

將類別屬性映射至全域次要索引的排序索引鍵。如果您需要對全域次要索引進行 Query，並且想要使用索引的排序索引鍵來縮小您結果的範圍，請使用此屬性。

DynamoDBHashKey

將類別屬性映射至資料表主索引鍵的分割區索引鍵。主索引鍵屬性不能是集合類型。

下列 C# 程式碼範例會映射 Book 類別至 ProductCatalog 資料表，並映射 Id 屬性至資料表主索引鍵的分割區索引鍵。

```
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }

    // Additional properties go here.
}
```

DynamoDBIgnore

指出應該忽略的相關聯屬性。如果您不想儲存任何類別屬性，可以新增此屬性來指示 DynamoDBContext 將物件儲存到資料表時不包含此屬性。

DynamoDBLocalSecondaryIndexRangeKey

將類別屬性映射至本機次要索引的排序索引鍵。如果您需要對本機次要索引進行 Query，並且想要使用索引的排序索引鍵來縮小您結果的範圍，請使用此屬性。

DynamoDBProperty

將類別屬性映射至資料表屬性。如果類別屬性映射至具有相同名稱的資料表屬性，則不需要指定此屬性。不過，如果名稱不同，您可以使用此標籤提供映射。在下列 C# 陳述式中，DynamoDBProperty 會將 BookAuthors 屬性映射至資料表中的 Authors 屬性。

```
[DynamoDBProperty("Authors")]
```

```
public List<string> BookAuthors { get; set; }
```

當儲存物件資料至相應的資料表時，DynamoDBContext 會使用此映射資訊來建立 Authors 屬性。

DynamoDBRenamable

指定類別屬性的替代名稱。如果您正在撰寫自訂轉換器，以將任意資料映射至 DynamoDB 資料表 (其中類別屬性的名稱與資料表屬性不同)，則此功能非常有用。

DynamoDBRangeKey

將類別屬性映射至資料表主索引鍵的排序索引鍵。如果資料表具有複合主鍵 (分割區索引鍵和排序索引鍵)，則必須在類別映射中同時指定 DynamoDBHashKey 和 DynamoDBRangeKey 屬性。

例如，範例資料表 Reply 有一個由 Id 分割區索引鍵和 Replenishment 排序索引鍵組成的主索引鍵。下列 C# 程式碼範例將 Reply 類別映射至 Reply 資料表。類別定義也會指出其中兩個屬性映射至主索引鍵。

```
[DynamoDBTable("Reply")]
public class Reply
{
   [DynamoDBHashKey]
   public int ThreadId { get; set; }
   [DynamoDBRangeKey]
   public string Replenishment { get; set; }

   // Additional properties go here.
}
```

DynamoDBTable

識別 DynamoDB 中類別要映射的目標資料表。例如，下列 C# 程式碼範例將 Developer 類別映射至 DynamoDB 中的 People 資料表。

```
[DynamoDBTable("People")]
public class Developer { ...}
```

此屬性可被繼承或覆寫。

- DynamoDBTable 屬性可被繼承。在上述範例中，如果新增新從 Developer 類別繼承的新類別 Lead，它也會映射至 People 資料表。Developer 和 Lead 物件都存放在 People 資料表中。

- DynamoDBTable 屬性也可被寫。在下列 C# 程式碼範例中，Manager 類別繼承自 Developer 類別。但是，明確新增 DynamoDBTable 屬性會將類別映射到另一個資料表 (Managers)。

```
[DynamoDBTable("Managers")]
public class Manager : Developer { ...}
```

您可以新增選用參數 LowerCamelCaseProperties，以便請求 DynamoDB 在儲存物件至資料表時，將屬性名稱的第一個字母變成小寫，如下列 C# 範例所示。

```
[DynamoDBTable("People", LowerCamelCaseProperties=true)]
public class Developer
{
    string DeveloperName;
    ...
}
```

在儲存 Developer 類別的執行個體時，DynamoDBContext 會儲存 DeveloperName 屬性作為 developerName。

DynamoDBVersion

標識用於儲存項目版本編號的類別屬性。如需版本控制的詳細資訊，請參閱「使用 DynamoDB 和 適用於 .NET 的 AWS SDK 物件持久性模型的樂觀鎖定」。

.NET 物件持久性模型的 DynamoDBContext 類別

DynamoDBContext 類別是 Amazon DynamoDB 資料庫的進入點。它可讓您連線至 DynamoDB，繼而存取各種資料表中的資料、執行各種 CRUD 操作以及執行查詢。此 DynamoDBContext 類別提供下列方法。

主題

- CreateMultiTableBatchGet
- CreateMultiTableBatchWrite
- CreateBatchGet
- CreateBatchWrite
- 刪除
- Dispose

- ExecuteBatchGet
- ExecuteBatchWrite
- FromDocument
- FromQuery
- FromScan
- GetTargetTable
- Load
- Query
- Save
- Scan
- ToDocument
- 指定 DynamoDBContext 的選用參數

CreateMultiTableBatchGet

建立由多個個別 BatchGet 物件組成的 MultiTableBatchGet 物件。其中的每個 BatchGet 物件都可用來從單一 DynamoDB 資料表中擷取項目。

若要從資料表擷取項目，請使用 ExecuteBatchGet 方法，將 MultiTableBatchGet 物件作為參數傳遞。

CreateMultiTableBatchWrite

建立由多個個別 BatchWrite 物件組成的 MultiTableBatchWrite 物件。其中的每個 BatchWrite 物件可用來寫入或刪除單一 DynamoDB 資料表中的項目。

若要寫入資料表，請使用 ExecuteBatchWrite 方法，將 MultiTableBatchWrite 物件作為參數傳遞。

CreateBatchGet

建立 BatchGet 物件，您可以使用該物件從資料表中擷取多個項目。

CreateBatchWrite

建立 BatchWrite 物件，您可以使用該物件將多個項目放入資料表中，或從資料表中刪除多個項目。

刪除

刪除資料表中的項目。這個方法需要您希望刪除之項目的主索引鍵。您可以提供主索引鍵值或包含主索引鍵值的用戶端物件，作為此方法的參數。

- 如果指定用戶端物件作為參數，且已啟用樂觀鎖定，則只有在物件的用戶端和伺服器端版本相符時，才能成功刪除項目。
- 如果只指定主索引鍵值作為參數，無論您是否已啟用樂觀鎖定，刪除都會成功。

> **Note**
>
> 若要在背景執行此操作，請改用 `DeleteAsync` 方法。

Dispose

處置所有受管和非受管的資源。

ExecuteBatchGet

從一或多個資料表讀取資料，處理 `MultiTableBatchGet` 中的所有 `BatchGet` 物件。

> **Note**
>
> 若要在背景執行此操作，請改用 `ExecuteBatchGetAsync` 方法。

ExecuteBatchWrite

在一或多個資料表中寫入或刪除資料，處理 `MultiTableBatchWrite` 中的所有 `BatchWrite` 物件。

> **Note**
>
> 若要在背景執行此操作，請改用 `ExecuteBatchWriteAsync` 方法。

FromDocument

若為 `Document` 執行個體，`FromDocument` 方法會傳回用戶端類別的執行個體。

如果您想使用文件模型類別以及物件持久性模型來執行任何資料操作，這會很有幫助。如需 提供之文件模型類別的詳細資訊 適用於 .NET 的 AWS SDK，請參閱 在 DynamoDB 中使用 .NET 文件模型。

假設您有名為 doc 的 Document 物件，且其中包含 Forum 項目的表示法。(若要查看如何建構此物件，請參閱本主題後續內容中的 ToDocument 方法說明。) 您可以使用 FromDocument 從 Document 擷取 Forum 項目，如下列 C# 程式碼範例所示。

Example

```
forum101 = context.FromDocument<Forum>(101);
```

> **Note**
>
> 如果您的 Document 物件會實作 IEnumerable 介面，您可以使用 FromDocuments 方法。這允許您逐一查看 Document 中的所有類別執行個體。

FromQuery

以 QueryOperationConfig 物件中定義的查詢參數執行 Query 操作。

> **Note**
>
> 若要在背景執行此操作，請改用 FromQueryAsync 方法。

FromScan

以 ScanOperationConfig 物件中定義的掃描參數執行 Scan 操作。

> **Note**
>
> 若要在背景執行此操作，請改用 FromScanAsync 方法。

GetTargetTable

擷取指定類型的目標資料表。如果您正在撰寫自訂轉換器，將任意資料映射至 DynamoDB 資料表，而且需要判斷哪個資料表與自訂資料類型相關聯，則此功能非常有用。

Load

從資料表擷取項目。這個方法只需要您希望擷取之項目的主索引鍵。

DynamoDB 預設會傳回數值最終一致的項目。如需最終一致性模式的資訊，請參閱 DynamoDB 讀取一致性。

Load 或 LoadAsync 方法會呼叫 GetItem 操作，這需要您指定資料表的主索引鍵。由於 GetItem 會忽略 IndexName 參數，您無法使用索引分割區或排序索引鍵載入項目。因此您必須使用資料表主索引鍵來載入項目。

> **Note**
>
> 若要在背景執行此操作，請改用 LoadAsync 方法。若要檢視使用 LoadAsync 方法在 DynamoDB 資料表執行高階 CRUD 操作的範例，請參閱下列範例。

```
/// <summary>
/// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
/// table.
/// </summary>
public class HighLevelItemCrud
{
    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();
        DynamoDBContext context = new DynamoDBContext(client);
        await PerformCRUDOperations(context);
    }

    public static async Task PerformCRUDOperations(IDynamoDBContext context)
    {
        int bookId = 1001; // Some unique value.
        Book myBook = new Book
        {
            Id = bookId,
            Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
            Isbn = "111-1111111001",
            BookAuthors = new List<string> { "Author 1", "Author 2" },
        };
```

```
            // Save the book to the ProductCatalog table.
            await context.SaveAsync(myBook);

            // Retrieve the book from the ProductCatalog table.
            Book bookRetrieved = await context.LoadAsync<Book>(bookId);

            // Update some properties.
            bookRetrieved.Isbn = "222-2222221001";

            // Update existing authors list with the following values.
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);

            // Try to retrieve deleted book. It should return null.
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
            }
        }
    }
```

Query

根據您提供的查詢參數查詢資料表。

您只能查詢具有複合主鍵 (分割區索引鍵和排序索引鍵) 的資料表。查詢時，必須指定分割區索引鍵和套用至排序索引鍵的條件。

假設您有用戶端 Reply 類別映射至 DynamoDB 中的 Reply 資料表。下列 C# 程式碼範例會查詢 Reply 資料表，找出過去 15 天所張貼的論壇主題回覆。Reply 資料表的主索引鍵具有 Id 分割區索引鍵和 ReplyDateTime 排序索引鍵。

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);
```

這會傳回 Reply 物件的集合。

Query 方法會傳回「延遲載入」IEnumerable 集合。它一開始只會傳回一頁的結果，然後視需要進行服務呼叫來取得下一頁。若要取得所有相符的項目，您只需要逐一查看 IEnumerable。

如果資料表具有簡易主索引鍵 (分割區索引鍵)，您便不能使用 Query 方法。不過，您可以使用 Load 方法並提供分割區索引鍵來擷取項目。

> **Note**
>
> 若要在背景執行此操作，請改用 QueryAsync 方法。

Save

將指定的物件儲存至資料表。如果輸入物件中指定的主索引鍵不存在於資料表中，該方法會將新項目新增至資料表中。如果主索引鍵存在，該方法會更新現有項目。

如果您已設定樂觀鎖定，則只有在用戶端和項目的伺服器端版本相符時才能成功更新項目。如需詳細資訊，請參閱使用 DynamoDB 和 適用於 .NET 的 AWS SDK 物件持久性模型的樂觀鎖定。

> **Note**
>
> 若要在背景執行此操作，請改用 SaveAsync 方法。

Scan

執行整個資料表掃描。

您可以指定掃描條件來篩選掃描結果。您可以根據資料表中的任何屬性來評估條件。假設您有用戶端 Book 類別映射至 DynamoDB 中的 ProductCatalog 資料表。下列 C# 範例會掃描資料表，並且只會傳回價格低於 0 的所有書籍項目。

Example

```
IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
                    new ScanCondition("Price", ScanOperator.LessThan, price),
                    new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
      );
```

Scan 方法會傳回「延遲載入」IEnumerable 集合。它一開始只會傳回一頁的結果，然後視需要進行服務呼叫來取得下一頁。若要取得所有相符的項目，您只需要逐一查看 IEnumerable。

出於效能考量，您應查詢資料表並避免執行資料表掃描。

> **Note**
>
> 若要在背景執行此操作，請改用 ScanAsync 方法。

ToDocument

從類別執行個體傳回 Document 文件模型類別的執行個體。

如果您想使用文件模型類別以及物件持久性模型來執行任何資料操作，這會很有幫助。如需 提供之文件模型類別的詳細資訊 適用於 .NET 的 AWS SDK，請參閱 在 DynamoDB 中使用 .NET 文件模型。

假設您有用戶端類別映射至範例 Forum 資料表。然後，您可以使用 DynamoDBContext 從 Forum 資料表中獲取項目作為 Document 物件，如下列 C# 程式碼範例所示。

Example

```
DynamoDBContext context = new DynamoDBContext(client);

Forum forum101 = context.Load<Forum>(101); // Retrieve a forum by primary key.
Document doc = context.ToDocument<Forum>(forum101);
```

指定 DynamoDBContext 的選用參數

使用物件持久性模型時，您可以指定 DynamoDBContext 的下列選用參數。

- **ConsistentRead**：當使用 Load、Query 或 Scan 操作來擷取資料時，您可以新增此選用參數來請求資料的最新數值。
- **IgnoreNullValues**：此參數會通知 DynamoDBContext 在 Save 操作期間忽略屬性上的 Null 數值。如果此參數為 false (或未設定)，則會將 Null 數值轉譯為刪除特定屬性的指令。
- **SkipVersionCheck**：此參數會通知 DynamoDBContext 不要在儲存或刪除項目時比較版本。如需版本控制的詳細資訊，請參閱「使用 DynamoDB 和 適用於 .NET 的 AWS SDK 物件持久性模型的樂觀鎖定」。
- **TableNamePrefix**：使用特定字串作為所有資料表名稱的字首。如果此參數為 null (或未設定)，則不會使用字首。
- DynamoDBEntryConversion – 指定用戶端使用的轉換結構描述。您可以將此參數設定為版本 V1 或 V2。V1 是預設版本。

  根據您設定的版本，此參數的行為會變更。例如：

  - 在 V1 中，bool 資料類型會轉換為 N 數字類型，其中 0 代表 false，1 代表 true。在 V2 中，bool 會轉換為 BOOL。
  - 在 V2 中，清單和陣列不會與 HashSet 一起分組。數值、字串類型和二進位類型的清單和陣列會轉換為 L (清單) 類型，可以傳送空白清單來更新清單。這與 V1 不同，其中空白清單不會透過線路傳送。

    在 V1 中，清單、HashSet 和陣列等集合類型都視為相同。清單、HashSet 和數值陣列會轉換為 NS (數字集) 類型。

  下列範例會將轉換結構描述版本設定為 V2，變更 .NET 類型和 DynamoDB 資料類型之間的轉換行為。

```
var config = new DynamoDBContextConfig
{
    Conversion = DynamoDBEntryConversion.V2
};
var contextV2 = new DynamoDBContext(client, config);
```

下列 C# 範例會指定 ConsistentRead 和 SkipVersionCheck 這兩個上述選用參數來建立新的 DynamoDBContext。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context =
       new DynamoDBContext(client, new DynamoDBContextConfig { ConsistentRead = true,
 SkipVersionCheck = true});
```

DynamoDBContext 包含這些選用參數，以及您使用此內容傳送的每個請求。

您可以使用 DynamoDBContext 為個別操作指定參數，而不是在 DynamoDBContext 層級設定這些參數，如下列 C# 程式碼範例所示。此範例會載入特定的書籍項目。DynamoDBContext 的 Load 方法會指定 ConsistentRead 和 SkipVersionCheck 等選用參數。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context = new DynamoDBContext(client);
Book bookItem = context.Load<Book>(productId,new DynamoDBContextConfig{ ConsistentRead
 = true, SkipVersionCheck = true });
```

在此例中，DynamoDBContext 只有在傳送 Get 請求時才會包含這些參數。

使用 DynamoDB 和 適用於 .NET 的 AWS SDK 物件持久性模型的樂觀鎖定

物件持久性模型中的樂觀鎖定支援可確保應用程式的項目版本與伺服器端的項目版本相同，然後再更新或刪除項目。假設您要擷取待更新的項目。不過，在您傳回更新之前，某些其他應用程式會更新相同的項目。現在應用程式的項目有過時的副本。如果沒有樂觀鎖定，您執行的任何更新都會覆寫其他應用程式所做的更新。

物件持久性模型的樂觀鎖定功能提供了 DynamoDBVersion 標籤，可讓您用來啟用樂觀鎖定。若要使用此功能，請在類別中新增一項屬性，以便儲存版本編號。您可以將 DynamoDBVersion 屬性加入該屬性。當您第一次儲存物件時，DynamoDBContext 會指派版本編號，並在每次更新項目時遞增值。

只有在用戶端物件版本與伺服器端項目的對應版本編號相符時，您的更新或刪除請求才會成功。如果應用程式有過時的副本，它必須先從伺服器取得最新版本，才能更新或刪除該項目。

下列 C# 程式碼範例會定義 Book 類別，該類別具有將其映射至 ProductCatalog 資料表的物件持久性屬性。類別中的 VersionNumber 屬性裝飾了會儲存版本編號數值的 DynamoDBVersion 屬性。

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**
>
> 您只可將 DynamoDBVersion 屬性套用至可為 null 的數字基本類型 (如 int?)。

樂觀鎖定對 DynamoDBContext 操作的影響如下：

- 對於新項目，DynamoDBContext 會指派初始版本編號 0。如果您擷取現有項目，請更新其一或多個屬性並嘗試儲存變更，只有在用戶端的版本編號與伺服器端的相符時，儲存操作才會成功。DynamoDBContext 會遞增版本號碼。您不需要設定版本編號。
- Delete 方法提供可將主索引鍵值或物件作為參數的多載，如下列 C# 程式碼範例所示。

  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);
  ```

如果提供物件作為參數，則只有在物件版本符合對應的伺服器端項目版本時才能成功刪除。不過，如果提供主索引鍵值作為參數，DynamoDBContext 不知道任何版本編號，其會在沒有檢查是哪個版本的情況下刪除項目。

請注意，物件持久性模型程式碼中的樂觀鎖定內部實作會使用 DynamoDB 中的條件式更新和條件式刪除 API 動作。

停用樂觀鎖定

若要停用樂觀鎖定，您可以使用 SkipVersionCheck 組態屬性。您可以在建立 DynamoDBContext 時設定此屬性。在這種情況下，您使用內容發起的任何請求都會停用樂觀鎖定。如需詳細資訊，請參閱指定 DynamoDBContext 的選用參數。

您可以停用特定操作的樂觀鎖定，而不是在內容層級設定屬性，如下列 C# 程式碼範例所示。此範例會使用內容來刪除書籍項目。Delete 方法可設定選用 SkipVersionCheck 屬性為 true，停用版本檢查。

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 });
```

使用 適用於 .NET 的 AWS SDK 物件持久性模型使用 DynamoDB 映射任意資料

除了支援的 .NET 類型之外 (請參閱 支援的資料類型)，您還可以使用應用程式中未直接映射至 Amazon DynamoDB 類型的類型。只要您提供轉換器將資料從任意類型轉換為 DynamoDB 類型，物件持久性模型就支援儲存任意類型的資料，反之亦然。轉換器程式碼會在物件的儲存和載入期間轉換資料。

您可以在用戶端建立任何類型。不過，存放在資料表中的資料是 DynamoDB 類型之一；在查詢和掃描期間，所做的任何資料比較都會與存放在 DynamoDB 中的資料進行比較。

下列 C# 程式碼範例會定義具有 Id、Title、ISBN 以及 Dimension 屬性的 Book 類別。Dimension 屬性具有 DimensionType，可說明 Height、Width 和 Thickness 屬性。範例程式碼提供了轉換器方法 ToEntry 和 FromEntry 來轉換 DimensionType 與 DynamoDB 字串類型之間的資料。例如，在儲存 Book 執行個體時，轉換器會建立一個書籍 Dimension 字串，例如 "8.5x11x.05"。在擷取書籍時，它會將字串轉換為 DimensionType 執行個體。

此範例會將 Book 類型映射為 ProductCatalog 資料表。它儲存了一個範例 Book 執行個體、擷取、更新其維度並再次儲存更新的 Book。

如需測試下列範例的逐步說明，請參閱 .NET 程式碼範例。

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;
        }
    }
}
```

執行此開發人員指南中的程式碼範例

AWS SDKs 以下列語言為 Amazon DynamoDB 提供廣泛的支援：

- Java
- 瀏覽器中的 JavaScript
- .NET
- Node.js
- PHP
- Python
- Ruby
- C++
- Go
- Android
- iOS

此開發人員指南中的程式碼範例使用下列程式設計語言，提供 DynamoDB 操作的更深入說明：

- Java 程式碼範例
- .NET 程式碼範例

在開始此練習之前，您需要建立 AWS 帳戶、取得存取金鑰和私密金鑰，並在電腦上設定 AWS Command Line Interface (AWS CLI)。如需詳細資訊，請參閱設定 DynamoDB (Web 服務)。

> **Note**
>
> 如果您使用可下載的 DynamoDB 版本，則需要使用 AWS CLI 來建立資料表和範例資料。您也需要使用每個 AWS CLI 命令指定 `--endpoint-url` 參數。如需詳細資訊，請參閱設定區域端點。

在 DynamoDB 中建立資料表，以及載入程式碼範例的資料

如需在 DynamoDB 中建立資料表、在範例資料集中載入、查詢資料以及更新資料的基礎知識，請參閱下文。

- 步驟 1：在 DynamoDB 建立資料表
- 步驟 2：將資料寫入 DynamoDB 資料表
- 步驟 3：從 DynamoDB 資料表讀取資料
- 步驟 4：更新 DynamoDB 資料表中的資料

Java 程式碼範例

主題

- Java：設定您的 AWS 登入資料
- Java：設定 AWS 區域和端點

此開發人員指南包含 Java 程式碼片段及可立即執行的程式。您可以在以下章節中找到這些程式碼範例：

- 在 DynamoDB 使用項目和屬性
- 在 DynamoDB 中使用資料表和資料

- 在 DynamoDB 中查詢資料表
- 掃描 DynamoDB 中的資料表
- 在 DynamoDB 中使用次要索引來改善資料存取
- Java 1.x：DynamoDBMapper
- DynamoDB Streams 的變更資料擷取

您可以搭配使用 AWS Toolkit for Eclipse 與 Eclipse 來快速開始使用。除了功能完整的 IDE 之外，您還可以取得 適用於 Java 的 AWS SDK 具有自動更新的 ，以及用於建置 AWS 應用程式的預先設定範本。

執行 Java 程式碼範例 (使用 Eclipse)

1. 下載並安裝 Eclipse IDE。
2. 下載並安裝 AWS Toolkit for Eclipse。
3. 啟動 Eclipse，然後在 Eclipse 選單上，選擇 File (檔案)、New (新建)，再選擇 Other (其他)。
4. 在 Select a wizard (選取協助程式) 中，選擇 AWS，選擇 AWS Java Project (AWS Java 專案)，再選擇 Next (下一步)。
5. 在建立 AWS Java 中，執行下列動作：

   a. 在專案名稱 中，輸入您的專案名稱。

   b. 在 Select Account (選取帳戶) 中，從清單選擇您的憑證描述檔。

      如果這是您第一次使用 AWS Toolkit for Eclipse，請選擇設定 AWS 帳戶來設定您的 AWS 登入資料。
6. 選擇 Finish (完成) 建立專案。
7. 從 Eclipse 選單，選擇 File (檔案)、New (新建)，再選擇 Class (類別)。
8. 在 Java Class (Java 類別) 的 Name (名稱) 中，輸入您的類別名稱 (使用與您要執行之程式碼範例相同的名稱)，然後選擇 Finish (完成) 建立類別。
9. 將程式碼範例從文件頁面複製到 Eclipse 編輯器。
10. 若要執行程式碼，請在 Eclipse 選單上選擇 Run (執行)。

Java 軟體開發套件提供可與 DynamoDB 搭配使用的安全執行緒用戶端。根據最佳實務，您的應用程式應該建立一個用戶端，並在執行緒之間重複使用該用戶端。

如需更多資訊，請參閱適用於 Java 的 AWS SDK。

> **Note**
>
> 此指南中的程式碼範例適用於最新版的 適用於 Java 的 AWS SDK。
> 如果您使用的是 AWS Toolkit for Eclipse，則可以為適用於 Java 的 SDK 設定自動更新。若要在 Eclipse 中執行此作業，請前往 Preferences (偏好設定)，然後依次選擇 AWS 工具組、適用於 Java 的 AWS SDK、Download new SDK automatically (自動下載新的開發套件)。

Java：設定您的 AWS 登入資料

適用於 Java 的 SDK 需要您在執行時間提供 AWS 登入資料給應用程式。本指南中的程式碼範例假設您使用的是 AWS 登入資料檔案，如 適用於 Java 的 AWS SDK 開發人員指南中的設定 AWS 登入資料中所述。

以下是名為 的 AWS 登入資料檔案範例`~/.aws/credentials`，其中波狀符號字元 (~) 代表您的主目錄。

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java：設定 AWS 區域和端點

依預設，程式碼範例會存取美國西部 (奧勒岡) 區域的 DynamoDB。您可以修改 `AmazonDynamoDB` 屬性來變更區域。

以下程式碼範例會執行個體化新的 `AmazonDynamoDB`。

```
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import com.amazonaws.regions.Regions;
...
// This client will default to US West (Oregon)
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
.withRegion(Regions.US_WEST_2)
.build();
```

您可以使用 `withRegion` 方法，對任何可用區域中的 DynamoDB 執行程式碼。如需完整清單，請參閱 Amazon Web Services 一般參考 中的 AWS 區域與端點。

如果您想要使用 DynamoDB 在電腦本機執行程式碼範例，請設定端點如下：

AWS SDK V1

```
AmazonDynamoDB client =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
new AwsClientBuilder.EndpointConfiguration("http://localhost:8000", "us-west-2"))
.build();
```

AWS SDK V2

```
DynamoDbClient client = DynamoDbClient.builder()
    .endpointOverride(URI.create("http://localhost:8000"))
    // The region is meaningless for local DynamoDb but required for client builder
 validation
    .region(Region.US_EAST_1)
    .credentialsProvider(StaticCredentialsProvider.create(
    AwsBasicCredentials.create("dummy-key", "dummy-secret")))
    .build();
```

.NET 程式碼範例

主題

- .NET：設定您的 AWS 登入資料
- .NET：設定 AWS 區域和端點

此指南包含 .NET 程式碼片段與可立即執行的程式。您可以在以下章節中找到這些程式碼範例：

- 在 DynamoDB 使用項目和屬性
- 在 DynamoDB 中使用資料表和資料
- 在 DynamoDB 中查詢資料表
- 掃描 DynamoDB 中的資料表
- 在 DynamoDB 中使用次要索引來改善資料存取
- 在 DynamoDB 中使用 .NET 文件模型
- 使用 .NET 物件持久性模型和 DynamoDB
- DynamoDB Streams 的變更資料擷取

您可以將 適用於 .NET 的 AWS SDK 與 Toolkit for Visual Studio 搭配使用，以快速開始使用。

執行 .NET 程式碼範例 (使用 Visual Studio)

1. 下載並安裝 Microsoft Visual Studio。
2. （選用）下載並安裝 Toolkit for Visual Studio。
3. 設定您的 AWS 登入資料。在共用登入資料檔案 () 中設定 AWS 登入資料設定檔`~/.aws/credentials`。如需詳細資訊，請參閱《適用於 .NET 的 AWS SDK 開發人員指南》中的設定 AWS 憑證。
4. 啟動 Visual Studio。選擇 File (檔案)、New (新增)、Project (專案)。
5. 搜尋主控台應用程式，選取以 .NET 為目標的 C# 範本，然後選擇下一步。設定您的專案名稱和位置，然後選擇建立。
6. 將適用於 DynamoDB NuGet 的 AWS SDK 套件新增至您的專案：
   a. 在方案總管中，開啟您專案的內容 (右鍵) 選單，然後選擇 Manage NuGet Packages (管理 NuGet 套件)。
   b. 在 NuGet 套件管理員中，選擇 Browse (瀏覽)。
   c. 在搜尋方塊中輸入 **`AWSSDK.DynamoDBv2`**，並等候搜尋完成。
   d. 選擇 AWSSDK.DynamoDBv2，然後選擇 Install (安裝)。
7. 在您的 Visual Studio 專案中，開啟 `Program.cs`。將內容取代為您要執行的文件頁面中的程式碼範例。
8. 若要執行程式碼，請在 Visual Studio 工具列中選擇 Start (啟動)。

適用於 .NET 的 SDK 提供使用 DynamoDB 的安全執行緒用戶端。根據最佳實務，您的應用程式應該建立一個用戶端，並在執行緒之間重複使用該用戶端。

如需詳細資訊，請參閱適用於 .NET 的AWS 軟體開發套件。

> **Note**
>
> 此指南中的程式碼範例適用於最新版的 適用於 .NET 的 AWS SDK。

.NET：設定您的 AWS 登入資料

適用於 .NET 的 SDK 需要您在執行時間提供 AWS 登入資料給應用程式。本指南中的程式碼範例假設您使用 SDK 存放區來管理 AWS 登入資料檔案，如《 適用於 .NET 的 AWS SDK 開發人員指南》中的使用 SDK 存放區中所述。

Toolkit for Visual Studio 支援來自任意數目帳戶的多組憑證。每個集合都稱為描述檔。Visual Studio 會將項目新增至專案的 App.config 檔案，讓您的應用程式可以在執行時間找到 AWS 登入資料。

下列範例顯示預設 App.config 檔案，當您使用 Toolkit for Visual Studio 建立新的專案時，就會生成此檔案。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
    <add key="AWSProfileName" value="default"/>
    <add key="AWSRegion" value="us-west-2" />
 </appSettings>
</configuration>
```

在執行時間，程式會使用 AWSProfileName 項目指定的default一組 AWS 登入資料。 AWS 登入資料本身會以加密形式保存在 SDK 存放區中。Toolkit for Visual Studio 提供圖形化使用者介面來管理所有來自於 Visual Studio 的憑證。如需詳細資訊，請參閱《AWS Toolkit for Visual Studio 使用者指南》中的指定憑證。

> **Note**
>
> 依預設，程式碼範例會存取美國西部 (奧勒岡) 區域的 DynamoDB。您可以修改 App.config 檔案中的 AWSRegion 項目來變更區域。您可以將 AWSRegion 設定為 DynamoDB 可用的任何區域。如需完整清單，請參閱 Amazon Web Services 一般參考 中的 AWS 區域與端點。

.NET：設定 AWS 區域和端點

依預設，程式碼範例會存取美國西部 (奧勒岡) 區域的 DynamoDB。您可以修改 App.config 檔案中的 AWSRegion 項目來變更區域。或者，您亦可以修改 AmazonDynamoDBClient 屬性來變更區域。

以下程式碼範例會執行個體化新的 AmazonDynamoDBClient。這會修改用戶端，讓程式碼對不同區域中的 DynamoDB 執行。

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// This client will access the US East 1 region.
clientConfig.RegionEndpoint = RegionEndpoint.USEast1;
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

如需區域的完整清單，請參閱 Amazon Web Services 一般參考 中的 AWS 區域與端點。

如果您想要使用 DynamoDB 在電腦本機執行程式碼範例，請設定端點如下：

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// Set the endpoint URL
clientConfig.ServiceURL = "http://localhost:8000";
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

DynamoDB 低階 API

Amazon DynamoDB 低階 API 為 DynamoDB 的協定層級介面。在此層級中，每個 HTTP(S) 要求都必須依照正確的格式，並附上有效的數位簽章。

AWS SDKs 會代表您建構低階 DynamoDB API 請求，並處理來自 DynamoDB 的回應。您如此即可專注於應用程式的邏輯，而非低階的結節。但您仍可從了解低階 DynamoDB API 運作方式的基本知識中獲益。

如需低階 DynamoDB API 的詳細資訊，請參閱《Amazon DynamoDB API 參考》。

> **Note**
>
> DynamoDB Streams 有自己的低階 API，與 DynamoDB 的低階 API 不同，且完全由 AWS SDKs 支援。
> 如需詳細資訊，請參閱DynamoDB Streams 的變更資料擷取。如需低階 DynamoDB Streams API，請參閱《Amazon DynamoDB Streams API 參考》。

低階 DynamoDB API 使用 JavaScript 物件表示法 (JSON) 作為接線協定格式。JSON 以階層顯示資料，以便同時傳達資料值和資料結構。名稱/值對以 `name:value` 格式定義。資料階層由成對的巢狀括住之名稱與值加以定義。

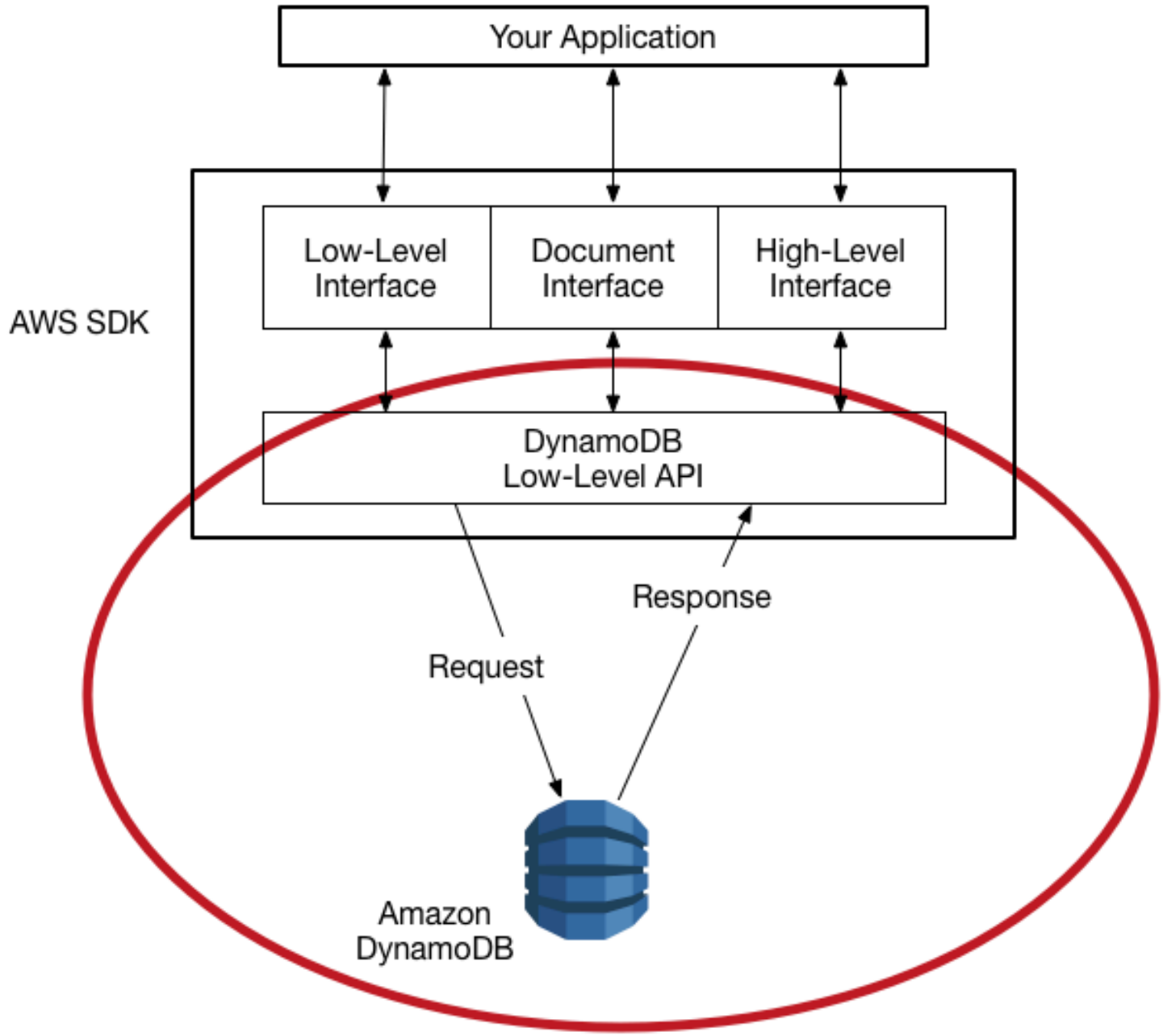
DynamoDB 只會使用 JSON 作為傳輸協定，而非儲存格式。 AWS SDKs 使用 JSON 將資料傳送至 DynamoDB，而 DynamoDB 會以 JSON 回應。DynamoDB 不會以 JSON 格式永久存放資料。

> **Note**
>
> 如需 JSON 的詳細資訊，請參閱 `JSON.org` 網站上的 JSON 簡介。

主題

- 要求格式

- 回應格式
- 資料類型描述項
- 數值資料
- 二進位資料



要求格式

DynamoDB 低階 API 接受 HTTP(S) POST 請求作為輸入。 AWS 軟體開發套件會為您建構這些請求。

假設您有一個名為 Pets 的資料表，其索引鍵結構描述由 AnimalType (分割區索引鍵) 及 Name (排序索引鍵) 所組成。這些屬性的類型皆為 string。若要從 擷取項目Pets，軟體 AWS 開發套件會建構下列請求。

```
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"}
    }
}
```

請注意以下此要求的相關事宜：

- Authorization 標頭包含 DynamoDB 驗證請求所需之資訊。如需詳細資訊，請參閱 中的簽署 AWS API 請求和簽署第 4 版簽署程序Amazon Web Services 一般參考。
- X-Amz-Target 標頭包含 DynamoDB 操作的名稱：GetItem。(此也會同時附上低階 API 版本，在此案例中為 20120810。)
- 要求的承載 (主體) 包含操作的參數 (JSON 格式)。若是 GetItem 操作，參數為 TableName 與 Key。

回應格式

收到請求時，DynamoDB 會處理該請求並會傳回回應。針對前述的要求，HTTP(S) 回應承載會包含操作的結果，如下列範例中所示。

```
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"}
    }
}
```

此時， AWS 軟體開發套件會將回應資料傳回至您的應用程式，以供進一步處理。

> **Note**
>
> 若 DynamoDB 無法處理請求，其會傳回 HTTP 錯誤碼及訊息。 AWS 軟體開發套件會以例外狀況的形式，將這些傳播到您的應用程式。如需詳細資訊，請參閱使用 DynamoDB 時發生錯誤。

資料類型描述項

低階 DynamoDB API 協定需要每個屬性都要附有一個資料類型的描述項。資料類型描述項是告知 DynamoDB 如何解譯每項屬性的字符。

「要求格式」與「回應格式」中的範例，展示如何使用資料類型描述項的範例。`GetItem` 要求指定 `Pets` 索引鍵結構描述屬性 (`AnimalType` 與 `Name`) 為 `S`，即類型為 `string`。`GetItem` 回應包含具有 `string` (`S`)、`number` (`N`)、`map` (`M`) 以及 `list` (`L`) 類型屬性的 Pets 項目。

下列為 DynamoDB 資料類型描述項的完整清單：

- **`S`**：字串
- **`N`**：數字
- **`B`**：二進位
- **`BOOL`**：布林值
- **`NULL`**：Null
- **`M`**：映射
- **`L`**：清單
- **`SS`**：字串集合
- **`NS`**：數字集合
- **`BS`**：二進位集合

下表顯示每個資料類型描述項的正確 JSON 格式。請注意，數字會以字串表示以保留精確度，而布林值和 null 則使用其原生 JSON 類型。

描述項	JSON format (JSON 格式)	備註
`S`	`{"S": "Hello"}`	值是 JSON 字串。
`N`	`{"N": "123.45"}`	值是字串，而不是 JSON 號碼。這可保留跨語言的精確度。
`B`	`{"B": "dGhpcyBpcyBhIHRlc3Q="}`	值是 base64 編碼字串。

描述項	JSON format (JSON 格式)	備註
BOOL	`{"BOOL": true}`	值是 JSON 布林值 (`true` 或 `false`)，而不是字串。
NULL	`{"NULL": true}`	值是`true`表示 null 的 JSON 布林值。
M	`{"M": {"Name": {"S": "Joe"}}}`	值是屬性名稱值對的 JSON 物件。
L	`{"L": [{"S": "Red"}, {"N": "5"}]}`	值是屬性值的 JSON 陣列。
SS	`{"SS": ["Red", "Blue"]}`	值是字串的 JSON 陣列。
NS	`{"NS": ["1", "2.5"]}`	值是數字字串的 JSON 陣列。
BS	`{"BS": ["U3Vubnk=", "UmFpbnk="]}`	值是 base64 編碼字串的 JSON 陣列。

> **Note**
>
> 如需 DynamoDB 資料類型的詳細說明，請參閱 資料類型。

數值資料

不同的程式語言提供不同程度的 JSON 支援。在某些情況下，您可能會決定要使用第三方的程式庫進行 JSON 文件之驗證及剖析。

某些第三方程式庫會依據 JSON 數字類型建置，提供其本身的類型，例如 `int`、`long` 或 `double`。但 DynamoDB 中的原生數字資料類型，無法精確地映射到這些其他資料類型，所以這些類型的差異可能會導致衝突。此外，許多 JSON 程式庫不會處理固定精確度的數值，而是會自動針對包含小數點的數字，推斷為雙精確度資料類型。

為解決這些問題，DynamoDB 提供了不會造成資料遺失的單一數字類型。為避免不必要的隱含轉換為雙精確度值，DynamoDB 使用字串來進行數值資料傳輸。此方法提供了更新屬性值的彈性，同時還能維持適當的排序語意，例如將值 "01"、"2" 及 "03" 依適當的順序排列。

若數字精確度對您的應用程式來說很重要，則應在將其傳遞至 DynamoDB 之前，先將數值轉換為字串。

二進位資料

DynamoDB 支援二進制屬性。但 JSON 原本並不支援二進位資料的編碼。若要在要求中傳送二進位資料，您必須將其編碼為 Base64 格式。在接收到請求時，DynamoDB 會將 Base64 資料解碼回二進位。

DynamoDB 使用的 base64 編碼結構描述會在 Internet Engineering Task Force (IETF) 網站上的 RFC 4648 說明。

使用 Python 和 Boto3 為 Amazon DynamoDB 編寫程式

本指南為想要透過 Python 使用 Amazon DynamoDB 的程式設計人員提供指導。了解不同的抽象層、組態管理、錯誤處理、控制重試政策、管理保持連線等。

主題

- 關於 Boto
- 使用 Boto 文件
- 了解用戶端和資源抽象層
- 使用資料表資源 batch_writer
- 探索用戶端和資源層的其他程式碼範例
- 了解用戶端和資源物件如何與工作階段和執行緒互動
- 自訂 Config 物件
- 錯誤處理
- 日誌
- 事件勾點
- 分頁和分頁程式
- 等待程式

關於 Boto

您可以使用適用於 Python 的官方 AWS SDK 從 Python 存取 DynamoDB，通常稱為 Boto3。Boto (發音為 boh-toh) 的名稱來自亞馬遜河原生的淡水海豚。Boto3 程式庫是程式庫的第三個主要版本，於

2015 年首次發行。Boto3 程式庫相當大，因為它支援所有 AWS 服務，而不只是 DynamoDB。此指導僅針對 Boto3 中與 DynamoDB 相關的部分。

Boto 由 維護並發佈 AWS 為 GitHub 上託管的開放原始碼專案。分為兩個套件：Botocore 和 Boto3。

- Botocore 提供低階功能。在 Botocore 中，您會找到用戶端、工作階段、憑證、組態和例外類別。
- Boto3 建置在 Botocore 之上，提供更高階、更具 Python 風格的介面。具體而言，它公開 DynamoDB 資料表做為資源，並提供比服務導向的低階用戶端介面更簡單、更優雅的介面。

由於這些專案託管在 GitHub 上，因此您可以檢視原始程式碼、追蹤未解決的問題，或提交您自己的問題。

使用 Boto 文件

使用下列資源開始使用 Boto 文件：

- 從 Quickstart 區段開始，該區段提供安裝套件的堅實起點。如需安裝 Boto3 的說明，請前往此處 (Boto3 通常可在 等 AWS 服務內自動使用 AWS Lambda)。
- 之後，請專注於文件的 DynamoDB 指南。它說明如何執行基本的 DynamoDB 活動：建立和刪除資料表、操作項目，以及執行批次操作、查詢和掃描。其範例使用資源介面。當您看到 `boto3.resource('dynamodb')`，表示您正在使用較高階的資源介面。
- 在本指南之後，您可以檢閱 DynamoDB 參考。此登陸頁面提供您可使用的類別和方法的完整清單。在頂端，您會看到 `DynamoDB.Client` 類別，提供所有控制平面和資料平面操作的低階存取。在底部則可查看 `DynamoDB.ServiceResource` 類別，這是更高階且具有 Python 風格的介面。您可以使用它來建立資料表、跨資料表執行批次操作，或取得資料表特定動作的 `DynamoDB.ServiceResource.Table` 執行個體。

了解用戶端和資源抽象層

您將使用的兩個介面是用戶端介面和資源介面。

- 低階用戶端介面提供對基礎服務 API 的 1 對 1 映射。DynamoDB 提供的每個 API 都可以透過用戶端取得。這表示用戶端介面可以提供完整的功能，但使用起來通常更冗長且複雜。
- 較高階的資源介面不提供基礎服務 API 的 1 對 1 映射。不過，它提供可讓您更方便存取服務的方法，例如 `batch_writer`。

以下是使用用戶端介面插入項目的範例。請注意所有的值如何以映射形式傳遞，其中鍵代表其類型 (字串為「S」，數字為「N」)，值則為字串。這稱為 DynamoDB JSON 格式。

```
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 ...
    }
)
```

以下是使用資源介面的相同 `PutItem` 操作。資料類型是隱含的：

```
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 ...
    }
)
```

如有需要，您可以使用 boto3 提供的 `TypeSerializer` 和 `TypeDeserializer` 類別，在一般 JSON 和 DynamoDB JSON 之間轉換：

```
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()
    }
```

以下是如何使用用戶端介面執行查詢。它將查詢表達為 JSON 建構，使用需要變數替換的 KeyConditionExpression 字串來處理任何潛在的關鍵字衝突：

```
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',
    }
)
```

使用資源介面的相同查詢操作可以縮短和簡化：

```
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')
)
```

最後，假設您想要取得資料表的大致大小 (這是保留在資料表上的中繼資料，大約每 6 小時更新一次)。使用用戶端介面，您可以執行 `describe_table()` 操作，並從傳回的 JSON 結構提取答案：

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.describe_table(TableName='YourTableName')
size = response['Table']['TableSizeBytes']
```

透過資源介面，資料表會隱含地執行描述操作，並直接將資料顯示為屬性：

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')
size = table.table_size_bytes
```

> **Note**
>
> 考慮使用用戶端或資源界面進行開發時，請注意，根據資源文件，不會將新功能新增至資源界面：「 AWS Python SDK 團隊不打算將新功能新增至 boto3 中的資源界面。在 boto3 的生命週期期間，現有的介面將繼續運作。客戶可以透過用戶端介面尋找存取較新服務功能。」

使用資料表資源 batch_writer

`batch_writer` 是一種便利的方法，僅適用於較高階的資料表資源。DynamoDB 支援批次寫入操作，在一個網路請求中最多允許 25 個放置或刪除操作。像這樣的批次處理可最大限度減少網路往返，從而提高效率。

透過低階用戶端程式庫，您可以使用 `client.batch_write_item()` 操作來執行批次。您必須手動將工作分成 25 個批次。在每次操作之後，您也必須請求獲得未處理項目的清單 (有些寫

入操作可能會成功，有些則可能會失敗)。然後，您必須將這些未處理的項目再次傳遞至稍後的 batch_write_item() 操作。有大量的樣板程式碼。

Table.batch_writer 方法會建立內容管理員，用於以批次方式寫入物件。它會顯示一個介面，看起來像是您一次寫入一個項目，但其實在內部會分批緩衝和傳送項目。它也會隱含地處理未處理的項目重試。

```
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)
```

探索用戶端和資源層的其他程式碼範例

您也可以參考下列程式碼範例儲存庫，以探索在使用用戶端和資源的情況下，各種函數的使用方式：

- 官方 AWS 單一動作程式碼範例。
- 官方 AWS 案例導向程式碼範例。
- 社群維護的單一動作程式碼範例。

了解用戶端和資源物件如何與工作階段和執行緒互動

資源物件不具備執行緒安全性，不應跨執行緒或程序共用。如需詳細資訊，請參閱資源指南。

相反地，用戶端物件通常具備執行緒安全性，但特定進階功能除外。如需更多詳細資訊，請參閱用戶端指南。

工作階段物件不具備執行緒安全性。因此，每次在多執行緒環境中建立用戶端或資源時，應先建立新的工作階段，然後從工作階段建立用戶端或資源。如需詳細資訊，請參閱工作階段指南。

當您呼叫 boto3.resource() 時，您會隱含地使用預設工作階段。這有助於撰寫單執行緒程式碼。撰寫多執行緒程式碼時，您會想要先為每個執行緒建構新的工作階段，然後從該工作階段擷取資源：

```
# Explicitly create a new Session for this thread
session = boto3.Session()
dynamodb = session.resource('dynamodb')
```

自訂 Config 物件

建構用戶端或資源物件時，您可以傳遞選用的已命名參數來自訂行為。名為 `config` 的參數會解鎖各種功能。這是 `botocore.client.Config` 的執行個體，而 Config 參考文件會顯示它公開供您控制的所有內容。組態指南提供良好的概觀。

> **Note**
>
> 您可以修改在工作階段層級、 AWS 組態檔案內或作為環境變數的許多行為設定。

逾時的組態

自訂組態的一個用途是調整聯網行為：

- connect_timeout (浮點數或整數) – 嘗試建立連線時擲出逾時例外狀況的秒數。預設值為 60 秒。
- read_timeout (浮點數或整數) – 嘗試從連線讀取時擲出逾時例外狀況前的秒數。預設值為 60 秒。

對 DynamoDB 而言，60 秒的逾時太長了。這表示暫時性網路故障會導致用戶端延遲一分鐘，然後才可以再試一次。下列程式碼會將逾時縮短為一秒：

```
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)
```

如需逾時的更多討論，請參閱調整延遲感知 DynamoDB 應用程式的 AWS Java SDK HTTP 請求設定。請注意，Java SDK 具有比 Python 更多的逾時組態。

保持連線的組態

如果您使用的是 botocore 1.27.84 或更新版本，您也可以控制 TCP Keep-Alive：

- tcp_keepalive (bool) - 啟用 TCP Keep-Alive 通訊端選項，如果設定為 `True` (預設為 `False`)，則在建立新連線時使用。從 botocore 1.27.84 開始才能使用。

將 TCP Keep-Alive 設定為 `True` 可減少平均延遲。以下是當您有正確的 botocore 版本時，將 TCP Keep-Alive 設定為 true 的範例程式碼：

```
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 與 HTTP Keep-Alive 不同。使用 TCP Keep-Alive 時，基礎作業系統會透過通訊端連線傳送小型封包，以保持連線並立即偵測任何下降。使用 HTTP Keep-Alive 時，會重複使用基礎通訊端上建置的 Web 連線。boto3 一律會啟用 HTTP Keep-Alive。

閒置連線可以保持運作的時間有限制。如果您有閒置連線，但希望下一個請求使用已建立的連線，請考慮定期傳送請求 (每分鐘傳送一次)。

重試的組態

組態也會接受稱為重試的字典，您可以在其中指定所需的重試行為。當 SDK 收到錯誤且錯誤為暫時性類型時，會在 SDK 中重試。如果在內部重試錯誤 (重試最終產生成功的回應)，從呼叫程式碼的角度就不會看到錯誤，只是延遲稍微增加。以下是您可以指定的值：

- max_attempts – 一個整數，表示對單一請求進行的重試嘗試次數上限。例如，若將此值設定為 2，初始請求後最多可重試請求兩次。若將此值設為 0，則在初始請求後不會嘗試任何重試。
- total_max_attempts – 一個整數，代表對單一請求所嘗試的總數上限。這包括初始請求，因此值 1 表示不會重試任何請求。如果同時提供 `total_max_attempts` 和 `max_attempts`，則

`total_max_attempts` 具有優先權。`total_max_attempts` 優先於 `max_attempts`，因為它映射到 `AWS_MAX_ATTEMPTS` 環境變數和 `max_attempts` 組態檔案值。

- mode – 一個字串，代表應該使用的重試模式 botocore 類型。有效的值如下：
  - legacy – 預設模式。第一次重試等待 50 毫秒，然後使用基本係數為 2 的指數退避。對於 DynamoDB，會執行最多總計 10 次嘗試 (除非以上述覆寫)。

    > **Note**
    >
    > 使用指數退避時，最後一次嘗試將等待近 13 秒。

  - 標準 – 具名標準，因為它更符合其他 AWS SDKs。第一次重試將等待從 0 毫秒到 1,000 毫秒的隨機時間。如果需要再次重試，會挑選從 0 毫秒到 1,000 毫秒的另一個隨機時間，並乘以 2。如果需要額外的重試，會執行相同的隨機挑選乘以 4，依此類推。每次等待上限為 20 秒。相較於 `legacy` 模式，此模式在偵測到更多失敗條件時會執行重試。對於 DynamoDB，會執行最多總共 3 次嘗試 (除非以上述覆寫)。
  - adaptive - 實驗性重試模式，包含標準模式的所有功能，但新增了自動用戶端限流。透過自適應速率限制，SDKs可以降低傳送請求的速率，以更好地適應 AWS 服務的容量。這是一種暫時性模式，其行為可能會變更。

您可以在重試指南中找到這些重試模式的擴展定義，也可以在 SDK 參考的重試行為主題中找到。

以下是明確使用 `legacy` 重試政策的範例，總共最多 3 個請求 (2 次重試)：

```
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)
```

由於 DynamoDB 是高度可用、低延遲的系統，因此建議您可以比內建重試政策所允許的程度，加快重試速度。您可以實作自己的重試政策，方法是將最大嘗試次數設定為 0，自行擷取例外狀況，並從自己的程式碼適當重試，而不是依賴 boto3 進行隱含重試。

如果您管理自己的重試政策，您會想要區分限流和錯誤：

- 限流 (由 ProvisionedThroughputExceededException 或 ThrottlingException 表示) 表示運作狀態良好的服務，通知您已超過 DynamoDB 資料表或分割區的讀取或寫入容量。每經過一毫秒，就能獲得更多的讀取或寫入容量，因此您可以快速重試 (例如每 50 毫秒) 以嘗試存取該新發布的容量。使用限流時，您不需要特別使用指數退避，因為傳回限流對 DynamoDB 很輕量，而且不會向您收取每次請求的費用。指數退避會將較長的延遲指派給已等待最長時間的用戶端執行緒，以統計方式將 p50 和 p99 向外延伸。
- 錯誤 (由 InternalServerError 或 ServiceUnavailable 等表示) 表示服務有暫時性問題。可能涉及整個資料表，也可能只是您正在讀取或寫入的分割區。發生錯誤時，您可以在重試之前暫停更長的時間 (例如 250 毫秒或 500 毫秒)，並使用抖動來交錯重試。

集區連線上限的組態

最後，組態可讓您控制連線集區大小：

- max_pool_connections (整數) – 在連線集區中保留的最大連線數。如果未指定任何值，則會使用預設值 10。

此選項控制要保留在集區中以供重複使用的 HTTP 連線數目上限。每個工作階段會保留不同的集區。如果您預期針對同一工作階段建置的用戶端或資源有 10 個以上的執行緒，您應該考慮將此值提高，讓執行緒不必等待其他正在使用集區連線的執行緒。

```
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
```

錯誤處理

AWS Boto3 中並未靜態定義所有 服務例外狀況。這是因為 AWS 服務的錯誤和例外狀況差異很大，可能會有所變更。Boto3 將所有服務例外狀況包裝為 `ClientError`，並以結構化 JSON 公開詳細資訊。例如，錯誤回應的結構如下所示：

```
{
    '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
    }
}
```

下列程式碼會擷取任何 `ClientError` 例外狀況，並查看 `Error` 內的 `Code` 字串值，以決定要採取的動作：

```
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
```

部分 (但並非全部) 例外狀況代碼已具體化為最高階類別。您可以選擇直接處理這些例外狀況。使用用戶端介面時，這些例外狀況會在您的用戶端上動態填入，而您使用用戶端執行個體來擷取這些例外狀況，如下所示：

```
except ddb_client.exceptions.ProvisionedThroughputExceededException:
```

使用資源介面時，您必須使用 `.meta.client` 從資源周遊到基礎用戶端來存取例外狀況，如下所示：

```
except ddb_resource.meta.client.exceptions.ProvisionedThroughputExceededException:
```

若要檢閱具體化例外狀況類型的清單，您可以動態產生清單：

```
ddb = boto3.client("dynamodb")
print([e for e in dir(ddb.exceptions) if e.endswith('Exception') or
 e.endswith('Error')])
```

使用條件表達式執行寫入操作時，您可以請求假如表達式失敗，應在錯誤回應中傳回項目的值。

```
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'])
```

如需進一步了解錯誤處理和例外狀況：

- 有關錯誤處理的 boto3 指南提供了有關錯誤處理技巧的詳細資訊。
- 程式設計錯誤的 DynamoDB 開發人員指南區段列出了您可能會遇到的錯誤。
- API 參考中的常見錯誤區段。
- 每個 API 操作的文件會列出呼叫可能產生的錯誤 (例如 BatchWriteItem)。

日誌

boto3 程式庫與 Python 的內建記錄模組整合，以追蹤工作階段期間發生的情況。若要控制記錄層級，您可以設定記錄模組：

```
import logging

logging.basicConfig(level=logging.INFO)
```

這樣會設定根記錄器來記錄 INFO 和更高層級的訊息。系統會忽略嚴重程度低於層級的記錄訊息。記錄層級包括 DEBUG、INFO、WARNING、ERROR 和 CRITICAL。預設值為 WARNING。

boto3 中的記錄器是階層式的。程式庫使用幾個不同的記錄器，每個記錄器對應至程式庫的不同部分。您可以個別控制每個記錄器的行為：

- boto3：boto3 模組的主要記錄器。
- botocore：botocore 套件的主要記錄器。
- botocore.auth：用於記錄請求的 AWS 簽章建立。
- botocore.credentials：用於記錄憑證擷取和重新整理的程序。
- botocore.endpoint：用於在透過網路傳送請求之前記錄建立請求。
- botocore.hooks：用於記錄程式庫中觸發的事件。
- botocore.loaders：用於載入部分 AWS 服務模型時的記錄。
- botocore.parsers：用於在剖析之前記錄 AWS 服務回應。
- botocore.retryhandler：用於記錄 AWS 服務請求重試的處理 （舊版）。
- botocore.retries.standard：用於記錄 AWS 服務請求重試的處理 （標準或適應模式）。
- botocore.utils：用於記錄程式庫中的其他活動。
- botocore.waiter：用於記錄等待程式的功能，在達到特定狀態之前輪詢 AWS 服務。

其他程式庫也會記錄。在內部，boto3 會使用第三方 urllib3 來處理 HTTP 連線。當延遲很重要時，您可以監看其日誌，以確保您的集區在 urllib3 建立新連線或關閉閒置連線時獲得充分利用。

- urllib3.connectionpool：用於記錄連線集區處理事件。

下列程式碼片段將大部分記錄設定為 INFO，DEBUG 記錄則用於端點和連線集區活動：

```
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)
```

事件勾點

Botocore 在執行的各個階段期間發出事件。您可以為這些事件註冊處理常式，以便每次發出事件時，都會呼叫您的處理常式。這可讓您擴充 botocore 的行為，而不必修改其內部。

例如，假設您希望每次在應用程式中對任何 DynamoDB 資料表呼叫 PutItem 操作時加以記錄。您可以在 'provide-client-params.dynamodb.PutItem' 事件上註冊，以便每次在關聯的工作階段上調用 PutItem 操作時擷取和記錄。範例如下：

```
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 ...
```

```
    }
)
```

在處理常式中，您甚至可以透過程式設計方式操作參數來變更行為：

```
params['TableName'] = "NewTableName"
```

如需事件的詳細資訊，請參閱事件的 botocore 文件和事件的 boto3 文件。

分頁和分頁程式

有些請求，例如查詢和掃描，會限制單一請求上傳回的資料大小，並要求您重複提出請求以提取後續分頁。

您可以使用 `limit` 參數控制每個分頁要讀取的項目數量上限。例如，如果您想要最後 10 個項目，您可以使用 `limit` 僅擷取最後 10 個項目。請注意，限制是指套用任何篩選之前，應該從資料表讀取多少。篩選後無法指定您想要的確切 10 個項目；您只能控制預先篩選的計數，並在實際擷取 10 個項目時在用戶端檢查。無論限制為何，每個回應的大小上限一律為 1 MB。

如果回應包含 `LastEvaluatedKey`，則表示回應因為達到計數或大小限制而結束。索引鍵是針對回應評估的最後一個鍵。您可以擷取此 `LastEvaluatedKey` 並將其傳遞至後續呼叫做為 `ExclusiveStartKey`，以讀取該起點的下一個區塊。沒有 `LastEvaluatedKey` 傳回該項目時，表示沒有更多符合查詢或掃描的項目。

以下是一個簡單的範例 (使用資源介面，但用戶端介面具有相同的模式)，每個頁面和迴圈最多讀取 100 個項目，直到所有項目都已讀取為止。

```
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']
```

為了方便起見，boto3 可以使用分頁程式為您執行此操作。不過，它僅適用於用戶端介面。以下是改寫為使用分頁程式的程式碼：

```
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)
```

如需詳細資訊，請參閱分頁程式指南和 DynamoDB.Paginator.Query 的 API 參考。

> **Note**
>
> 分頁程式也有自己的組態設定，名為 `MaxItems`、`StartingToken` 和 `PageSize`。若要使用 DynamoDB 進行分頁，您應該忽略這些設定。

等待程式

等待程式提供在繼續之前等待某些項目完成的能力。目前，僅支援等待建立或刪除資料表。在背景中，等待程式操作會每 20 秒為您檢查一次，最多 25 次。您可以自行執行此操作，但在撰寫自動化時，使用等待程式是一種簡練的方式。

此程式碼說明如何等待特定資料表建立完成：

```
# 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']
```

如需詳細資訊，請參閱等待程式指南和等待程式參考。

使用 JavaScript 編寫 Amazon DynamoDB 程式

本指南為想要搭配 JavaScript 使用 Amazon DynamoDB 的程式設計人員提供指導。了解 適用於 JavaScript 的 AWS SDK、可用的抽象層、設定連線、處理錯誤、定義重試政策、管理保持連線等。

主題

- 關於 適用於 JavaScript 的 AWS SDK
- 使用 適用於 JavaScript 的 AWS SDK V3
- 存取 JavaScript 文件
- 抽象層
- 使用 marshall 公用程式函數
- 讀取項目
- 條件式寫入
- 分頁
- 指定組態
- 等待程式
- 錯誤處理
- 日誌
- 考量事項

關於 適用於 JavaScript 的 AWS SDK

適用於 JavaScript 的 AWS SDK 可讓您 AWS 服務 使用瀏覽器指令碼或 Node.js 存取 。本文件著重於最新版本的 SDK (V3)。V3 適用於 JavaScript 的 AWS SDK 由 維護 AWS 為 GitHub 上託管的開放原始碼專案。問題和功能請求是公開的，您可以在 GitHub 儲存庫的問題頁面上存取它們。

JavaScript V2 類似於 V3，但包含語法差異。V3 更模組化，可更輕鬆地提供較小的相依性，並具有一級 TypeScript 支援。我們建議您使用最新版的 SDK。

使用 適用於 JavaScript 的 AWS SDK V3

您可以使用 Node Package Manager 將 SDK 新增至 Node.js 應用程式。以下範例示範如何新增最常用於 DynamoDB 的 SDK 套件。

- `npm install @aws-sdk/client-dynamodb`
- `npm install @aws-sdk/lib-dynamodb`
- `npm install @aws-sdk/util-dynamodb`

安裝套件會將參考新增至 package.json 專案檔案的相依性區段。您可以選擇使用較新的 ECMAScript 模組語法。如需這兩種方法的進一步詳細資訊，請參閱考量一節。

存取 JavaScript 文件

使用下列資源開始使用 JavaScript 文件：

- 存取開發人員指南以取得核心 JavaScript 文件。安裝指示位於設定區段。
- 存取 API 參考文件，以探索所有可用的類別和方法。
- 適用於 JavaScript 的 SDK 支援 DynamoDB AWS 服務 以外的許多 。使用下列程序來尋找 DynamoDB 的特定 API 涵蓋範圍：

  1. 從服務中，選擇 DynamoDB 和程式庫。這會記錄低階用戶端。
  2. 選擇 lib-dynamodb。這會記錄高階用戶端。兩個用戶端代表您可以選擇使用的兩個不同的抽象層。如需抽象層的詳細資訊，請參閱以下章節。

抽象層

適用於 JavaScript V3 的 SDK 具有低階用戶端 (`DynamoDBClient`) 和高階用戶端 (`DynamoDBDocumentClient`)。

主題

- 低階客戶端 (DynamoDBClient)
- 高階用戶端 (DynamoDBDocumentClient)

低階客戶端 (**DynamoDBClient**)

低階用戶端對於基礎線路通訊協定不提供額外的抽象概念。它可讓您完全控制通訊的各個層面，但由於沒有抽象概念，因此您必須使用 DynamoDB JSON 格式執行提供項目定義等操作。

如以下範例所示，使用此格式的資料類型必須明確陳述。S 表示字串值，N 表示數值。線路上的數字一律會以標記為數字類型的字串傳送，以確保精確度不會遺失。低階 API 呼叫的命名模式例如 `PutItemCommand` 和 `GetItemCommand`。

下列範例使用低階用戶端，搭配利用 DynamoDB JSON 定義的 `Item`：

```
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();
```

高階用戶端 (**DynamoDBDocumentClient**)

高階 DynamoDB 文件用戶端提供內建的便利功能，例如不需要手動封送資料，以及允許使用標準 JavaScript 物件直接讀取和寫入。lib-dynamodb 文件提供優點清單。

若要執行個體化 DynamoDBDocumentClient，請建構低階 DynamoDBClient，然後使用 DynamoDBDocumentClient 包裝它。函數命名慣例在兩個套件之間略有不同。例如，低階使用 PutItemCommand，而高階使用 PutCommand。不同的名稱允許兩組函數在相同的內容中共存，讓您可以在相同的指令碼中混合兩者。

```
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();
```

當您使用 GetItem、Query 或 Scan 等 API 操作讀取項目時，使用模式是一致的。

使用 marshall 公用程式函數

您可以自行使用低階用戶端和封送或取消封送資料類型。公用程式套件 util-dynamodb 具有 marshall() 公用程式函數，可接受 JSON 並產生 DynamoDB JSON，以及反向執行的 unmarshall() 函數。下列範例使用低階用戶端搭配由 marshall() 呼叫處理的資料封送。

```
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();
```

讀取項目

若要從 DynamoDB 資料表讀取項目，請使用 GetItem API 操作。與 PutItem 命令類似，您可以選擇使用低階用戶端或高階文件用戶端。以下範例示範如何使用高階文件用戶端擷取項目。

```
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();
```

使用 Query API 操作讀取多個項目。您可以使用低階用戶端或文件用戶端。以下範例使用高階文件用戶端。

```
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();
```

條件式寫入

DynamoDB 寫入操作可以指定邏輯條件表達式，必須評估為 true 才能繼續寫入。如果條件未評估為 true，寫入操作會產生例外狀況。條件表達式可以檢查項目是否已存在，或其屬性是否符合特定限制條件。

```
ConditionExpression = "version = :ver AND size(VideoClip) < :maxsize"
```

當條件式表達式失敗時，您可以使用 ReturnValuesOnConditionCheckFailure 來請求錯誤回應包含不符合條件的項目，以推斷問題所在。如需詳細資訊，請參閱使用 Amazon DynamoDB 在高並行情況下處理條件式寫入錯誤。

```
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;
      }
  }
```

其他程式碼範例顯示 JavsScript SDK V3 用量的其他層面，可在 JavaScript SDK V3 文件和 DynamoDB-SDK-Examples GitHub 儲存庫下取得。

分頁

主題

- 使用 paginateScan 便利方法

Scan 或 Query 等讀取請求可能會傳回資料集中的多個項目。如果您使用 Limit 參數執行 Scan 或 Query，一旦系統讀取到該項目數時，就會傳送部分回應，而且您需要分頁才能擷取其他項目。

對於每個請求，系統最多只會讀取 1 MB 的資料。如果您包含 Filter 表達式，系統仍將從磁碟讀取最多 1 MB 的資料，但會傳回該 MB 中符合篩選條件的項目。篩選操作可能從頁面傳回 0 個項目，但在搜尋窮盡之前仍需要進一步分頁。

您應該在回應中尋找 LastEvaluatedKey，並在後續請求中使用它做為 ExclusiveStartKey 參數，以繼續資料擷取。這可做為書籤，如下列範例所述。

> **Note**
>
> 在第一次迭代時，該範例會傳遞 null lastEvaluatedKey 做為 ExclusiveStartKey，這是允許的。

使用 LastEvaluatedKey 的範例：

```
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);
});
```

使用 **paginateScan** 便利方法

開發套件提供稱為 paginateScan 和 paginateQuery 的便利方法，可在幕後為您執行此操作並重複提出請求。使用標準 Limit 參數指定每個請求要讀取的項目數量上限。

```
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**
>
> 除非資料表很小，否則不建議採用定期執行完整資料表掃描的存取模式。

指定組態

主題

- 逾時的組態
- 保持連線的組態
- 重試的組態

設定 DynamoDBClient 時，您可以將組態物件傳遞至建構函數，以指定各種組態覆寫。例如，如果呼叫內容還不知道，您可以指定要連線的區域，或指定要使用的端點 URL。如果您想要針對 DynamoDB 本機版執行個體進行開發，這非常有用。

```
const client = new DynamoDBClient({
  region: "eu-west-1",
  endpoint: "http://localhost:8000",
});
```

逾時的組態

DynamoDB 使用 HTTPS 進行用戶端-伺服器通訊。您可以透過提供 NodeHttpHandler 物件來控制 HTTP 層的某些層面。例如，您可以調整金鑰逾時值 connectionTimeout 和 requestTimeout。connectionTimeout 是用戶端在嘗試建立連線後放棄連線時所等待的最長持續時間，以毫秒為單位。

requestTimeout 定義傳送請求後用戶端等待回應的時間長度，也以毫秒為單位。兩者的預設值都是零，表示逾時已停用，而且如果未收到回應，用戶端等待的時間將沒有極限。您應該將逾時設定為合理值，以便在發生網路問題時，請求會因逾時而發出錯誤，並可啟動新的請求。例如：

```
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**
>
> 提供的範例使用 Smithy 匯入。Smithy 是一種用於定義服務和 SDK 的語言，屬於開放原始碼，並由 AWS維護。

除了設定逾時值之外，您還可以設定通訊端數量上限，以增加每個原始伺服器的並行連線數量。開發人員指南包含設定 maxSockets 參數的詳細資訊。

保持連線的組態

使用 HTTPS 時，第一個請求一律需要一些來回通訊才能建立安全連線。HTTP Keep-Alive 允許後續請求重複使用已建立的連線，使請求更有效率並降低延遲。JavaScript V3 預設會啟用 HTTP Keep-Alive。

閒置連線可以保持運作的時間有限制。如果您有閒置連線，但希望下一個請求使用已建立的連線，請考慮每分鐘傳送定期請求。

> **Note**
>
> 請注意，在較舊的 SDK V2 中，保持連線預設為關閉，這表示每個連線都會在使用後立即關閉。如果使用 V2，您可以覆寫此設定。

重試的組態

當 SDK 收到錯誤回應，且 SDK 判斷該錯誤是可恢復的，例如限流例外狀況或暫時服務例外狀況時，將會再次重試。身為呼叫者的您不會察覺到這一切，但您可能會注意到請求需要更長的時間才能成功。

根據預設，適用於 JavaScript V3 的 SDK 會在放棄錯誤並將其傳遞至呼叫內容之前，發出總共 3 次請求。您可以調整這些重試次數和頻率。

DynamoDBClient 建構函數接受會限制將要嘗試多少次的 maxAttempts 設定。以下範例會將該值從預設值 3 提高為總計 5 次。如果您將其設定為 0 或 1，表示您不希望任何自動重試，並想要自行在擷取區塊內處理任何可恢復的錯誤。

```
const client = new DynamoDBClient({
  maxAttempts: 5,
});
```

您也可以使用自訂重試策略來控制重試的時間。若要這樣做，請匯入 util-retry 公用程式套件，並建立自訂退避函數，根據目前的重試計數計算重試之間的等待時間。

以下範例顯示，如果第一次嘗試失敗，最多嘗試 5 次，延遲為 15、30、90 和 360 毫秒。自訂退避函數 calculateRetryBackoff 會接受重試嘗試次數 (從第一次重試的 1 開始) 來計算延遲，並傳回該請求應等待的毫秒數。

```
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.
  ),
});
```

等待程式

DynamoDB 用戶端包含兩個有用的等待程式函數，當您希望程式碼等待資料表修改完成後才繼續執行時，可在建立、修改或刪除資料表時使用。例如，您可以部署資料表、呼叫 waitUntilTableExists 函數，而且程式碼會封鎖，直到資料表變成 ACTIVE 為止。等待程式每 20 秒會以 describe-table 輪詢一次 DynamoDB 服務。

```
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 功能只會在可以執行會顯示資料表狀態 ACTIVE 的 describe-table 命令時傳回控制項。這可確保您可以使用 waitUntilTableExists 等待建立完成，以及新增 GSI 索引等修改，這可能需要一些時間來套用，資料表才會返回 ACTIVE 狀態。

錯誤處理

在這裡的早期範例中，我們概括地發現了所有錯誤。不過，在實際的應用程式中，區分各種錯誤類型並實作更精確的錯誤處理非常重要。

DynamoDB 錯誤回應包含中繼資料，包括錯誤的名稱。您可以擷取錯誤，然後比對可能的錯誤情況字串名稱，以判斷如何繼續。對於伺服器端錯誤，您可以利用 instanceof 運算子搭配 @aws-sdk/client-dynamodb 套件匯出的錯誤類型，有效率地管理錯誤處理。

請務必注意，這些錯誤只有在所有重試都用盡之後才會顯示。如果重試錯誤，且最終接續成功呼叫，從程式碼的角度來看並沒有錯誤，只是延遲稍微增加了。重試會在 Amazon CloudWatch 圖表中顯示為失敗的請求，例如限流或錯誤請求。如果用戶端達到重試計數上限，則會放棄並產生例外狀況。這是用戶端表達其不會重試的方式。

以下是擷取錯誤並根據傳回的錯誤類型採取行動的程式碼片段。

```
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.
      }
    }
}
```

請參閱《DynamoDB 開發人員指南》中的常見 the section called “錯誤處理” 錯誤字串。您可以在該 API 呼叫的文件中找到任何特定 API 呼叫的確切錯誤，例如查詢 API 文件。

錯誤中繼資料包含其他屬性，視錯誤而定。對於 `TimeoutError`，中繼資料包含已嘗試的次數和 `totalRetryDelay`，如下所示。

```
{
  "name": "TimeoutError",
  "$metadata": {
    "attempts": 3,
    "totalRetryDelay": 199
  }
}
```

如果您管理自己的重試政策，您會想要區分限流和錯誤：

- 限流 (由 `ProvisionedThroughputExceededException` 或 `ThrottlingException` 表示) 表示運作狀態良好的服務，通知您已超過 DynamoDB 資料表或分割區的讀取或寫入容量。每經過一毫秒，就能獲得更多讀取或寫入容量，因此您可以快速重試，例如每 50 毫秒，以嘗試存取該新發布的容量。

  使用限流時，您不需要特別使用指數退避，因為傳回限流對 DynamoDB 很輕量，而且不會向您收取每次請求的費用。指數退避會將較長的延遲指派給已等待最長時間的用戶端執行緒，以統計方式將 p50 和 p99 向外延伸。
- 錯誤 (由 `InternalServerError` 或 `ServiceUnavailable` 等表示) 表示服務的暫時性問題，可能是整個資料表，也可能是您正在讀取或寫入的分割區。發生錯誤時，您可以在重試之前暫停更長的時間，例如 250 毫秒或 500 毫秒，並使用抖動來交錯重試。

日誌

開啟記錄功能，以取得 SDK 正在執行之作業的詳細資訊。您可以在 DynamoDBClient 上設定參數，如以下範例所示。更多日誌資訊會顯示在主控台中，並包含中繼資料，例如狀態碼和耗用容量。如果您於本機上在終端機視窗中執行程式碼，日誌會顯示在該處。如果您在 中執行程式碼 AWS Lambda，且已設定 Amazon CloudWatch logs，則會在該處寫入主控台輸出。

```
const client = new DynamoDBClient({
  logger: console
});
```

您也可以連接到內部 SDK 活動，並在發生特定事件時執行自訂記錄。以下範例使用用戶端的 middlewareStack 攔截從 SDK 傳送的每個請求，並在發生時記錄該請求。

```
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 提供強大的勾點，用於觀察和控制 SDK 行為。如需詳細資訊，請參閱 部落格簡介模組化 Middleware Stack 適用於 JavaScript 的 AWS SDK。

考量事項

在專案 適用於 JavaScript 的 AWS SDK 中實作 時，以下是一些需要考慮的進一步因素。

模組系統

開發套件支援兩個模組系統：CommonJS 和 ES (ECMAScript)。CommonJS 使用 require 函數，而 ES 則使用 import 關鍵字。

1. CommonJS – const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");

2. ES (ECMAScript – `import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";`

   專案類型指定要使用的模組系統，並在 package.json 檔案的類型區段中指定。預設為 CommonJS。使用 `"type": "module"` 來指示 ES 專案。如果您有使用 CommonJS 套件格式的現有 Node.JS 專案，您仍然可以使用 .mjs 副檔名來命名函數檔案，以新增具有更現代化 SDK V3 匯入語法的函數。這將允許把程式碼檔案視為 ES (ECMAScript)。

非同步操作

您將看到許多使用回呼的程式碼範例，並承諾處理 DynamoDB 操作的結果。使用現代 JavaScript 不再需要這種複雜性，開發人員可以利用更簡潔且可讀取的非同步/等待語法進行非同步操作。

Web 瀏覽器執行時期

使用 React 或 React Native 進行建置的 Web 和行動開發人員可以在其專案中使用適用於 JavaScript 的 SDK。使用舊版的 SDK V2，Web 開發人員必須將完整的 SDK 載入瀏覽器，並參考託管於 https://sdk.amazonaws.com/js/ 的 SDK 映像。

使用 V3 時，您可以使用 Webpack 將所需的 V3 用戶端模組和所有必要的 JavaScript 函數綁定到單一 JavaScript 檔案中，並將其新增至 HTML 頁面 <head> 中的指令碼標籤，如 SDK 文件的瀏覽器指令碼入門一節所述。

DAX 資料平面操作

適用於 JavaScript V3 的 SDK 支援 Amazon DynamoDB Streams Accelerator (DAX) 資料平面操作。

使用 程式設計 DynamoDB AWS SDK for Java 2.x

此程式設計指南為想要搭配 Java 使用 Amazon DynamoDB 的程式設計人員提供指導。本指南涵蓋不同的概念，包括抽象層、組態管理、錯誤處理、控制重試政策和管理保持連線。

主題

- 關於 AWS SDK for Java 2.x
- 開始使用 AWS SDK for Java 2.x
- 檢閱 AWS SDK for Java 2.x 文件
- 支援的介面
- 其他程式碼範例
- 同步和非同步程式設計

- HTTP 用戶端
- 設定 HTTP 用戶端
- 錯誤處理
- AWS 請求 ID
- 日誌
- 分頁
- 資料類別註釋

關於 AWS SDK for Java 2.x

您可以使用 官方從 Java 存取 DynamoDB 適用於 Java 的 AWS SDK。適用於 Java 的 SDK 有兩個版本：1.x 和 2.x。1.x 版本已於 2024 年 1 月 12 日宣布終止支援。它將在 2024 年 7 月 31 日進入維護模式，並將於 2025 年 12 月 31 日終止支援。對於新開發，我們強烈建議您使用 2.x，該版本於 2018 年首次發行。本指南專門針對 2.x，且僅著重於與 DynamoDB 相關的 SDK 部分。

如需 AWS SDKs，請參閱《AWS SDK 和工具參考指南》中的 SDK 和工具維護政策和 AWS SDKs 和工具版本支援矩陣。 AWS SDKs

AWS SDK for Java 2.x 是 1.x 程式碼基底的主要重寫。適用於 Java 2.x 的 SDK 支援現代 Java 功能，例如 Java 8 中引入的非封鎖 I/O。適用於 Java 2.x 的 SDK 也新增了對可插入 HTTP 用戶端實作的支援，以提供更多的網路連線彈性和組態選項。

適用於 Java 1.x 的 SDK 和適用於 Java 2.x 的 SDK 之間的明顯變更是使用新的套件名稱。Java 1.x SDK 使用 `com.amazonaws` 套件名稱，而 Java 2.x SDK 使用 `software.amazon.awssdk`。同樣地，Java 1.x SDK 的 Maven 成品使用 `com.amazonaws` `groupId`，而 Java 2.x SDK 成品則使用 `software.amazon.awssdk` `groupId`。

> **Important**
>
> 適用於 Java 的 AWS SDK 1.x 有一個名為 的 DynamoDB 套件`com.amazonaws.dynamodbv2`。套件名稱中的「v2」不表示其適用於 Java 2 (J2SE)。相反地，「v2」表示套件支援 DynamoDB 低階 API 的第二個版本，而非低階 API 的原始版本。

支援 Java 版本

AWS SDK for Java 2.x 提供長期支援 (LTS) Java 版本的完整支援。

開始使用 AWS SDK for Java 2.x

下列教學課程說明如何使用 Apache Maven 來定義適用於 Java 2.x 的 SDK 的相依性。本教學課程也說明如何撰寫連線至 DynamoDB 的程式碼，以列出可用的 DynamoDB 資料表。本指南中的教學課程是以《AWS SDK for Java 2.x 開發人員指南》中的 AWS SDK for Java 2.x入門教學課程為基礎。我們已編輯本教學課程，以呼叫 DynamoDB 而非 Amazon S3。

為了完成本教學，您需要執行以下各項：

- 步驟 1：設定本教學課程
- 步驟 2：建立專案
- 步驟 3：撰寫程式碼
- 步驟 4：建置並執行應用程式

步驟 1：設定本教學課程

在開始本教學課程之前，您需要以下項目：

- 存取 DynamoDB 的許可。
- 使用 設定單一登入存取的 Java AWS 服務 開發環境 AWS 存取入口網站。

若要設定本教學課程，請遵循 AWS SDK for Java 2.x 開發人員指南中設定概觀中的指示。設定具有單一登入存取設定的開發環境 (針對 Java SDK)，並擁有作用中的 AWS 存取入口網站工作階段後，請繼續本教學課程的步驟 2。

步驟 2：建立專案

若要建立本教學課程的專案，請執行 Maven 命令，該命令會提示您輸入如何設定專案。輸入並確認所有輸入後，Maven 會建立 `pom.xml` 檔案和建立 stub Java 檔案來完成建置專案。

1. 開啟終端機或命令提示視窗，然後導覽至您選擇的目錄，例如您的 `Desktop` 或 `Home` 資料夾。
2. 在終端輸入下列命令，然後按下 Enter 鍵。

```
mvn archetype:generate \
   -DarchetypeGroupId=software.amazon.awssdk \
   -DarchetypeArtifactId=archetype-app-quickstart \
   -DarchetypeVersion=2.22.0
```

3. 針對每個提示，輸入第二欄中列出的值。

提示詞	要輸入的值
`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. 輸入最後一個值後，Maven 會列出您所做的選擇。若要確認，請輸入 Y。或者輸入 N，然後重新輸入您的選擇。

Maven 會根據您輸入的 `artifactId` 值建立名為 `getstarted` 的專案資料夾。在 `getstarted` 資料夾中，尋找名為 `README.md` 的檔案以供檢閱、`pom.xml` 檔案和 `src` 目錄。

Maven 會建置下列目錄樹狀目錄。

```
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
```

下列顯示 pom.xml 專案檔案的內容。

pom.xml

dependencyManagement 區段包含 AWS SDK for Java 2.x的相依性，而 dependencies 區段具有 DynamoDB 的相依性。指定這些相依性會強制 Maven 在 Java 類別路徑中包含相關的 .jar 檔案。根據預設， AWS 軟體開發套件不會包含所有 的所有類別 AWS 服務。對於 DynamoDB，如果您使用低階介面，則應具有對 dynamodb 成品的相依性。或者，如果您使用高階介面，則為 dynamodb-enhanced 成品。如果您未包含相關的相依性，則程式碼無法編譯。由於 maven.compiler.source 和 maven.compiler.target 屬性中的 1.8 值，專案使用 Java 1.8。

```
<?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>
```

步驟 3：撰寫程式碼

下列程式碼顯示 Maven 建立的 App 類別。main 方法是應用程式的進入點，它會建立 Handler 類別的執行個體，然後呼叫其 sendRequest 方法。

App 類別

```
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");
     }
 }
```

Maven 建立的 DependencyFactory 類別包含建置和傳回 DynamoDbClient 執行個體的 dynamoDbClient 原廠方法。DynamoDbClient 執行個體使用 Apache 型 HTTP 用戶端的執行個體。這是因為您在 Maven 提示您使用哪個 HTTP 用戶端時指定了 apache-client。

下列程式碼顯示 DependencyFactory 類別。

DependencyFactory 類別

```
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 類別包含您程式的主要邏輯。App 類別中建立了 Handler 的執行個體時，DependencyFactory 會提供 DynamoDbClient 服務用戶端。您的程式碼使用 DynamoDbClient 執行個體來呼叫 DynamoDB。

Maven 會產生具有 *TODO* 註解的下列 Handler 類別。教學課程中的下一個步驟會以程式碼取代 *TODO* 註解。

Handler 類別，Maven 產生的

```
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.
    }
}
```

若要填入邏輯，請使用下列程式碼取代 Handler 類別的整個內容。sendRequest 方法已填入，並新增必要的匯入。

Handler 類別，已實作

下列程式碼使用 DynamoDbClient 執行個體來擷取現有資料表的清單。如果指定帳戶和 AWS 區域內存在資料表，則程式碼會使用 Logger 執行個體來記錄這些資料表的名稱。

```
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));
        }
```

```
    }
}
```

步驟 4：建置並執行應用程式

在您建立專案並包含完整的 Handler 類別之後，請建置並執行應用程式。

1. 請確定您有一個作用中的 AWS IAM Identity Center 工作階段。若要確認，請執行 AWS Command Line Interface (AWS CLI) 命令 aws sts get-caller-identity 並檢查回應。如果您沒有作用中的工作階段，請參閱使用 AWS CLI登入以取得指示。
2. 開啟終端機或命令提示視窗，然後導覽至您的專案目錄 getstarted。
3. 使用下列命令來建置專案：

```
mvn clean package
```

4. 若要部署應用程式，請執行下列命令：

```
mvn exec:java -Dexec.mainClass="org.example.App"
```

檢視檔案之後，請刪除物件，然後刪除儲存貯體。

成功

如果您的 Maven 專案能建置且執行時沒有錯誤，恭喜您！您已成功使用適用於 Java 的 SDK 2.x 建置了第一個 Java 應用程式。

清除

若要清除您在本教學課程中建立的資源，請刪除專案資料夾 getstarted。

檢閱 AWS SDK for Java 2.x 文件

AWS SDK for Java 2.x 開發人員指南涵蓋 SDK 在一切 AWS 服務中的所有層面。建議您檢閱下列主題：

- 從 1.x 版移轉至 2.x – 包含 1.x 和 2.x 之間差異的詳細說明。本主題也包含如何使用兩個主要版本的指示，以並排方式顯示。
- 適用於 Java 2.x SDK 的 DynamoDB 指南 – 說明如何執行基本的 DynamoDB 操作：建立資料表、操作項目和擷取項目。這些範例使用低階介面。Java 有數個介面，如下節所述：支援的介面。

> **Tip**
>
> 檢閱這些主題之後，請將 AWS SDK for Java 2.x API 參考加入書籤。它涵蓋所有項目 AWS 服務，我們建議您將其用作主要 API 參考。

支援的介面

AWS SDK for Java 2.x 支援下列界面，視您想要的抽象程度而定。

本節主題

- 低階介面
- 高階介面
- 文件介面
- 使用 Query 範例比較各介面

低階介面

低階介面提供對基礎服務 API 的 1 對 1 映射。每個 DynamoDB API 都可以透過此介面使用。這表示低階介面可以提供完整的功能，但通常使用起來更冗長且複雜。例如，您必須使用 `.s()` 函數來保留字串，並使用 `.n()` 函數來保留數字。下列 PutItem 範例使用低階介面插入項目。

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

高階介面

中的高階界面 AWS SDK for Java 2.x 稱為 DynamoDB 增強型用戶端。此介面提供更道地的程式碼撰寫體驗。

增強型用戶端提供一種在用戶端資料類別和旨在存放該資料的 DynamoDB 資料表之間映射的方法。您要在自己的程式碼中定義資料表及其對應模型類別之間的關係。然後，您可以依賴 SDK 來管理資料類型處理。如需增強型用戶端的詳細資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的 DynamoDB 增強型用戶端 API。

下列 PutItem 範例使用高階介面。在此範例中，名為 `YourItem` 的 `DynamoDbBean` 會建立 `TableSchema`，使其能夠直接使用為 `putItem()` 呼叫的輸入。

```
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;
        }
    }
}
```

適用於 Java 的 AWS SDK 1.x 有自己的高階界面，通常由其主要類別 參考。DynamoDBMapper AWS SDK for Java 2.x 會以名為 的個別套件 （ 和 Maven 成品 ） 發佈`software.amazon.awssdk.enhanced.dynamodb`。Java 2.x SDK 通常由其主類別 DynamoDbEnhancedClient 參考。

使用不可變資料類別的高階介面

DynamoDB 增強型用戶端 API 的映射功能也適用於不可變的資料類別。不可變類別只有取得器，且需要 SDK 用來建立類別執行個體的建置器類別。Java 中的不可變性是一種常用的樣式，開發人員可用來建立沒有副作用的類別。這些類別在複雜多執行緒應用程式中的行為更可預測。不可變類別會使用 `@DynamoDbImmutable` 註釋，其採用建置器類別做為其輸入，而不是使用 High-level interface example 中所示的 `@DynamoDbBean` 註釋。

下列範例採用建置器類別 DynamoDbEnhancedClientImmutablePutItem 做為輸入來建立資料表結構描述。然後，此範例提供結構描述做為 PutItem API 呼叫的輸入。

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

下列範例顯示不可變的資料類別。

```
@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); }
    }
}
```

使用不可變資料類別和第三方樣板產生程式庫的高階介面

不可變的資料類別 (如上例所示) 需要一些樣板程式碼。例如，除了 Builder 類別之外，對資料類別的取得器和設定器邏輯。例如 Project Lombok 等第三方程式庫可協助您產生該類型的樣板程式碼。減少大部分樣板程式碼可協助您限制使用不可變資料類別和 AWS SDK 所需的程式碼數量。這進一步提高了生產力與程式碼的可讀性。如需詳細資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的使用 Lombok 等第三方程式庫。

下列範例示範 Project Lombok 如何簡化使用 DynamoDB 增強型用戶端 API 所需的程式碼。

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

下列範例顯示不可變資料類別的不可變資料物件。

```
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 類別使用 Project Lombok 和 AWS SDK 提供的下列註釋：

- @Builder – 為 Project Lombok 提供的資料類別產生複雜的建置器 API。
- @DynamoDbImmutable – 將DynamoDbImmutable類別識別為 AWS 開發套件提供的 DynamoDB 可映射實體註釋。
- @Value – @Data 的不可變變體。根據預設，所有欄位都是私有和最終欄位，不會產生設定器。Project Lombok 提供此註釋。

文件介面

AWS SDK for Java 2.x 文件界面不需要指定資料類型描述項。資料類型是由資料本身的語義隱含的。此文件界面類似於 文件界面 適用於 Java 的 AWS SDK 1.x，但具有重新設計的界面。

以下 Document interface example 顯示使用文件介面表示的 PutItem 呼叫。此範例也使用 EnhancedDocument。若要使用增強型文件 API 針對 DynamoDB 資料表執行命令，您必須先將資料表與文件資料表結構描述建立關聯，才能建立 DynamoDBTable 資源物件。文件資料表結構描述建置器需要主索引鍵和屬性轉換器提供者。

您可以使用 AttributeConverterProvider.defaultProvider() 來轉換預設類型的文件屬性。您可以使用自訂 AttributeConverterProvider 實作變更整體預設行為。您也可以變更單一屬性的轉換器。AWS SDK 和工具參考指南提供有關如何使用自訂轉換器的更多詳細資訊和範例。其主要用途是用於沒有預設轉換器的網域類別屬性。使用自訂轉換器，您可以向 SDK 提供寫入或讀取至 DynamoDB 所需的資訊。

```
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)");
    }

}
```

若要將原生 Amazon DynamoDB 資料類型與 JSON 文件彼此轉換，您可以使用下列公用程式方法：

- EnhancedDocument.fromJson(String json) – 從 JSON 字串建立新的 EnhancedDocument 實體。
- EnhancedDocument.toJson() – 建立文件的 JSON 字串表示法，您可以在應用程式中使用，就像任何其他 JSON 物件一樣。

使用 **Query** 範例比較各介面

本節顯示使用各種介面表示的相同 Query 呼叫。若要微調這些查詢的結果，請注意下列事項：

- DynamoDB 以一個特定的分割區索引鍵值為目標，因此您必須完全指定分割區索引鍵。
- 若要讓查詢目標僅購物車項目，排序索引鍵具有使用 begins_with 的索引鍵條件表達式。
- 我們使用 limit() 將查詢限制為最多傳回 100 個項目。
- 我們將 scanIndexForward 設定為 false。結果會依 UTF-8 位元組的順序傳回，這通常表示會先傳回數量最低的購物車項目。透過將 scanIndexForward 設定為 false，我們把順序反轉，並先傳回數量最高的購物車項目。
- 我們會套用篩選條件，以移除不符合條件的任何結果。篩選的資料會耗用讀取容量，無論項目是否符合篩選條件。

Example **Query** 使用低階介面

下列範例使用 keyConditionExpression 查詢名為 YourTableName 的資料表。這會將查詢限制為特定的分割區索引鍵值，以及以特定字首值開頭的排序索引鍵值。這些金鑰條件會限制從 DynamoDB 讀取的資料量。最後，查詢會對使用 filterExpression 從 DynamoDB 擷取的資料套用篩選條件。

```
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`** 使用文件介面

下列範例會使用文件介面查詢名為 YourTableName 的資料表。

```
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`** 使用高階介面

下列範例會使用 DynamoDB 增強型用戶端 API 查詢名為 YourTableName 的資料表。

```
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;
        }
    }
}
```

使用不可變資料類別的高階介面

當您使用高階不可變資料類別執行 Query 時，程式碼會與高階介面範例相同，但實體類別 YourItem 或 YourImmutableItem 的建構除外。如需詳細資訊，請參閱 PutItem 範例。

使用不可變資料類別和第三方樣板產生程式庫的高階介面

當您使用高階不可變資料類別執行 Query 時，程式碼會與高階介面範例相同，但實體類別 YourItem 或 YourImmutableLombokItem 的建構除外。如需詳細資訊，請參閱 PutItem 範例。

其他程式碼範例

如需如何搭配適用於 Java 2.x 的 SDK 使用 DynamoDB 的其他範例，請參閱下列程式碼範例儲存庫：

- 官方 AWS 單一動作程式碼範例
- 社群維護的單一動作程式碼範例

- 官方 AWS 案例導向程式碼範例

同步和非同步程式設計

為 AWS SDK for Java 2.x 提供同步和非同步用戶端 AWS 服務，例如 DynamoDB。

`DynamoDbClient` 和 `DynamoDbEnhancedClient` 類別提供同步方法，這些方法會封鎖您的執行緒執行，直到用戶端收到服務的回應。如果您不需要非同步操作，此用戶端是與 DynamoDB 互動最直接的方式。

`DynamoDbAsyncClient` 和 `DynamoDbEnhancedAsyncClient` 類別提供非同步方法，這些方法會立即傳回，將控制權交還給呼叫端執行緒，無需等待回應。非封鎖用戶端具有優勢，可在幾個執行緒中用於獲得高並行，以最少的運算資源有效率地處理 I/O 請求。這可改善輸送量和回應能力。

AWS SDK for Java 2.x 使用原生支援進行非封鎖 I/O。 適用於 Java 的 AWS SDK 1.x 必須模擬非封鎖 I/O。

同步方法會在有可用回應時傳回結果，您需要一個方法在回應準備好時取得回應。中的非同步方法會適用於 Java 的 AWS SDK 傳回`CompletableFuture`物件，其中包含未來非同步操作的結果。當您對這些 `CompletableFuture` 物件呼叫 `get()` 或 `join()` 時，您的程式碼會封鎖，直到結果可用為止。如果您在提出請求的同時呼叫這些函數，則行為類似於純同步呼叫。

如需非同步程式設計的詳細資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的使用非同步程式設計。

HTTP 用戶端

為了支援每個用戶端，有一個 HTTP 用戶端來處理與 AWS 服務的通訊。您可以插入替代 HTTP 用戶端，選擇具有最符合您應用程式特性的用戶端。有些更輕量；有些則有更多組態選項。

有些 HTTP 用戶端僅支援同步使用，有些僅支援非同步使用。如需可協助您為工作負載選取最佳 HTTP 用戶端的流程圖，請參閱《AWS SDK for Java 2.x 開發人員指南》中的 HTTP 用戶端建議。

下列清單顯示一些可能的 HTTP 用戶端：

主題

- Apache 型 HTTP 用戶端
- URLConnection 型 HTTP 用戶端
- Netty 型 HTTP 用戶端

- AWS CRT 型 HTTP 用戶端

Apache 型 HTTP 用戶端

`ApacheHttpClient` 類別支援同步服務用戶端。這是同步使用的預設 HTTP 用戶端。如需有關 `ApacheHttpClient` 類別的資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的設定 Apache 型 HTTP 用戶端。

`URLConnection` 型 HTTP 用戶端

`UrlConnectionHttpClient` 類別是同步用戶端的另一個選項。它比 Apache 型 HTTP 用戶端載入更快，但功能較少。如需有關設定 `UrlConnectionHttpClient` 類別的資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的設定 URLConnection 型 HTTP 用戶端。

Netty 型 HTTP 用戶端

`NettyNioAsyncHttpClient` 類別支援非同步用戶端。這是非同步使用的預設選擇。如需有關 `NettyNioAsyncHttpClient` 類別的資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的設定 Netty 型 HTTP 用戶端。

AWS CRT 型 HTTP 用戶端

AWS 通用執行期 (CRT) 程式庫中較新的 `AwsCrtHttpClient` 和 `AwsCrtAsyncHttpClient`類別是支援同步和非同步用戶端的更多選項。相較於其他 HTTP 用戶端， AWS CRT 提供：

- 更快速的 SDK 啟動時間
- 記憶體佔用空間較小
- 延遲時間縮短
- 連線運作狀態管理
- DNS 負載平衡

如需有關設定 `AwsCrtHttpClient`和 `AwsCrtAsyncHttpClient`類別的資訊，請參閱《 AWS SDK for Java 2.x 開發人員指南》中的設定 AWS CRT 型 HTTP 用戶端。

AWS CRT 型 HTTP 用戶端不是預設值，因為這會破壞現有應用程式的回溯相容性。不過，對於 DynamoDB，我們建議您將 AWS CRT 型 HTTP 用戶端用於同步和非同步用途。

如需 AWS CRT 型 HTTP 用戶端的簡介，請參閱《 AWS 開發人員工具部落格》中的宣布 AWS CRT HTTP 用戶端的可用性 AWS SDK for Java 2.x。

設定 HTTP 用戶端

設定用戶端時，您可以提供各種組態選項，包括：

- 設定 API 呼叫不同層面的逾時。
- 啟用 TCP Keep-Alive。
- 遇到錯誤時控制重試政策。
- 指定執行攔截器執行個體可以修改的執行屬性。執行攔截器可以編寫程式碼來攔截 API 請求和回應的執行。這可讓您即時執行發布指標和修改傳輸中請求等任務。
- 新增或操作 HTTP 標頭。
- 啟用用戶端效能指標的追蹤。使用此功能可協助您收集應用程式中的服務用戶端指標並在 Amazon CloudWatch 中分析輸出。
- 指定要用於排程任務的替代執行器服務，例如非同步重試嘗試和逾時任務。

您可以提供 `ClientOverrideConfiguration` 物件給服務用戶端 `Builder` 類別以控制組態。您會在下列各節的一些程式碼範例中看到此情況。

`ClientOverrideConfiguration` 提供標準組態選擇。不同的可插入 HTTP 用戶端也有實作特定的組態可能性。

本節主題

- 逾時組態
- RetryMode
- DefaultsMode
- Keep-Alive 組態

逾時組態

您可以調整用戶端組態，以控制與服務呼叫相關的各種逾時。與其他 AWS 服務相比，DynamoDB 提供較低的延遲。因此，您可能想要將這些屬性調整為較低的逾時值，以便在發生聯網問題時快速檢錯。

您可以在 DynamoDB 用戶端上使用 `ClientOverrideConfiguration`，或針對基礎 HTTP 用戶端實作變更詳細的組態選項，以自訂延遲相關行為。

您可以使用 `ClientOverrideConfiguration` 來設定下列具影響力的屬性：

- `apiCallAttemptTimeout` – 在放棄和逾時之前，等待 HTTP 請求完成一次嘗試的時間長度。
- `apiCallTimeout` – 用戶端完全執行 API 呼叫所需的時間量。這包括由所有 HTTP 請求組成的請求處理常式執行，包括重試。

AWS SDK for Java 2.x 提供一些逾時選項的預設值，例如連線逾時和通訊端逾時。SDK 不提供 API 呼叫逾時或個別 API 呼叫嘗試逾時的預設值。如果未在 `ClientOverrideConfiguration` 中設定這些逾時，則 SDK 會使用通訊端逾時值，而非整體 API 呼叫逾時。通訊端的預設逾時值為 30 秒。

RetryMode

另一個與您應該考慮的逾時組態相關的組態是 `RetryMode` 組態物件。此組態物件包含重試行為的集合。

適用於 Java 2.x 的 SDK 支援下列重試模式：

- `legacy` – 如果您未明確變更，預設的重試模式。此重試模式專屬於 Java SDK。它的特徵是最多 3 次重試，或對於 DynamoDB 等服務有 8 次重試。
- `standard` – 命名為「標準」，因為它更符合其他 AWS SDKs。此模式會等待從 0 毫秒到 1,000 毫秒的隨機時間量進行第一次重試。如果需要重試，則此模式會挑選從 0 毫秒到 1,000 毫秒的另一個隨機時間，並將其乘以 2。如果需要額外的重試，則會執行相同的隨機挑選值乘以 4，依此類推。每次等待上限為 20 秒。相較於 `legacy` 模式，此模式在偵測到時會執行重試的失敗條件更多。對於 DynamoDB，除非您使用 numRetries 覆寫，否則它最多會執行總共 3 次嘗試。
- `adaptive` – 以 `standard` 模式為基礎，並動態限制 AWS 請求率，使成功率最大化。這可能會因為請求延遲而發生。當可預測延遲很重要時，我們不建議使用自適應重試模式。

您可以在 AWS SDK 和工具參考指南的重試行為主題中找到這些重試模式的擴展定義。

重試政策

所有 `RetryMode` 組態都有以一或多個 `RetryCondition` 組態為基礎的 `RetryPolicy`。對於 DynamoDB SDK 用戶端實作的重試行為而言，`TokenBucketRetryCondition` 特別重要。此條件會限制 SDK 使用權杖儲存貯體演算法進行的重試次數。根據選取的重試模式，限流例外狀況可能會也可能不會從 `TokenBucket` 中減去權杖。

當用戶端遇到可重試的錯誤，例如限流例外狀況或暫時性伺服器錯誤時，SDK 會自動重試請求。您可以控制這些重試的次數和速度。

設定用戶端時，您可以提供支援下列參數的 `RetryPolicy`：

- numRetries – 請求視為失敗之前應套用的重試次數上限。無論您使用的重試模式為何，預設值都是 8。

  > **Warning**
  >
  > 請務必在適當考量後變更此預設值。

- backoffStrategy – 要套用至重試的 BackoffStrategy，預設策略為 FullJitterBackoffStrategy。此策略會根據目前的次數或重試次數、基本延遲和最大退避時間，在額外重試之間執行指數延遲。然後，它會新增抖動以提供一點隨機性。無論重試模式為何，指數延遲中使用的基本延遲為 25 毫秒。
- retryCondition – RetryCondition 決定要不要重試請求。根據預設，它會重試一組特定的 HTTP 狀態碼，以及其認為可重試的例外狀況。在大多數情況下，預設組態應已足夠。

下列程式碼提供替代的重試政策。它總共指定 5 次重試 (總共 6 個請求)。第一次重試應該在延遲大約 100 毫秒之後進行，每次額外的重試都會將該時間以指數方式倍增，最多延遲一秒鐘。

```
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

ClientOverrideConfiguration 和 RetryMode 不管理的逾時屬性通常透過指定 DefaultsMode 來隱含設定。

AWS SDK for Java 2.x (2.17.102 版或更新版本） 推出對 的支援DefaultsMode。此功能為常見的可配置設定提供一組預設值，例如 HTTP 通訊設定、重試行為、服務區域端點設定，以及可能的任何 SDK 相關組態。使用此功能時，您可以取得針對常見使用案例量身打造的新組態預設值。

預設模式會跨所有 AWS SDKs標準化。適用於 Java 2.x 的 SDK 支援下列預設模式：

- `legacy` – 提供預設設定，這些設定會因 AWS SDK 而異，而且在 `DefaultsMode` 建立之前已存在。
- `standard` – 提供大多數案例的預設非最佳化設定。
- `in-region` – 以標準模式為基礎，並包含為 AWS 服務 在相同 內呼叫的應用程式量身打造的設定 AWS 區域。
- `cross-region` – 以標準模式為基礎，並為在不同區域中呼叫 AWS 服務 的應用程式提供具有高逾時值的設定。
- `mobile` – 以標準模式為基礎，並為具有較高延遲的行動應用程式提供量身打造的高逾時設定。
- `auto` – 以標準模式為基礎，並包含實驗性功能。SDK 會嘗試探索執行時期環境，自動判斷適當的設定。自動偵測是以啟發式為基礎，不提供 100% 的準確性。如果無法判斷執行時期環境，則會使用標準模式。自動偵測可能會查詢執行個體中繼資料和使用者資料，這可能會導致延遲。如果啟動延遲對您的應用程式至關重要，我們建議您改為明確選擇 `DefaultsMode`。

您可以透過下列方式設定預設模式：

- 直接在用戶端上，透過 `AwsClientBuilder.Builder#defaultsMode(DefaultsMode)`。
- 在組態設定檔上，透過 `defaults_mode` 設定檔檔案屬性。
- 全域，透過 `aws.defaultsMode` 系統屬性。
- 全域，透過 `AWS_DEFAULTS_MODE` 環境變數。

> **Note**
>
> 對於 `legacy` 以外的任何模式，附加的預設值可能會隨著最佳實務的演進而變更。因此，如果您使用的是 `legacy` 以外的模式，建議您在升級 SDK 時執行測試。

AWS SDK 和工具參考指南中的智慧組態預設會以不同的預設模式提供組態屬性及其預設值的清單。

您可以根據應用程式的特性和應用程式互動的 AWS 服務 來選擇預設模式值。

這些值的設定考量了 的廣泛選擇 AWS 服務 。對於 DynamoDB 資料表和應用程式都部署在一個區域中的典型 DynamoDB 部署，`in-region` 預設模式在 `standard` 預設模式之間最為相關。

Example DynamoDB SDK 用戶端組態已針對低延遲呼叫進行調校

下列範例會將逾時調整為預期低延遲 DynamoDB 呼叫的較低值。

```
DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.builder()
    .defaultsMode(DefaultsMode.IN_REGION)
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder())
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(3))
        .apiCallAttemptTimeout(Duration.ofMillis(500))
        .build())
    .build();
```

個別 HTTP 用戶端實作可能會為您提供對逾時和連線使用行為的更精細控制。例如，對於以 AWS CRT 為基礎的用戶端，您可以啟用 `ConnectionHealthConfiguration`，讓用戶端主動監控已使用連線的運作狀態。如需詳細資訊，請參閱《 AWS SDK for Java 2.x 開發人員指南》中的 AWS CRT 型 HTTP 用戶端的進階組態。

Keep-Alive 組態

啟用保持連線可以透過重複使用連線來減少延遲。有兩種不同類型的保持連線：HTTP Keep-Alive 和 TCP Keep-Alive。

- HTTP Keep-Alive 會嘗試維護用戶端和伺服器之間的 HTTPS 連線，以便稍後的請求可以重複使用該連線。這會略過稍後請求的重型 HTTPS 身分驗證。HTTP Keep-Alive 預設為在所有用戶端上啟用。
- TCP Keep-Alive 請求基礎作業系統透過通訊端連線傳送小型封包，以提供通訊端保持運作的額外保證，並立即偵測任何封包丟失。這可確保稍後的請求不會花時間嘗試使用被捨棄的通訊端。預設會針對所有用戶端停用 TCP Keep-Alive。下列程式碼範例示範如何在每個 HTTP 用戶端上啟用它。針對所有非 CRT 型 HTTP 用戶端啟用時，實際的 Keep-Alive 機制取決於作業系統。因此，您必須透過作業系統設定額外的 TCP Keep-Alive 值，例如逾時和封包數量。您可以在 Linux 或 macOS 上使用 `sysctl`，或在 Windows 上使用登錄值來執行此操作。

Example在 Apache 型 HTTP 用戶端上啟用 TCP Keep-Alive

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(ApacheHttpClient.builder().tcpKeepAlive(true))
    .build();
```

`URLConnection` 型 HTTP 用戶端

任何使用 `URLConnection` 型 HTTP 用戶端的同步用戶端 `HttpURLConnection` 都沒有啟用保持連線的機制。

Example在 Netty 型 HTTP 用戶端上啟用 TCP Keep-Alive

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(NettyNioAsyncHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Example在 AWS CRT 型 HTTP 用戶端上啟用 TCP Keep-Alive

使用 AWS CRT 型 HTTP 用戶端，您可以啟用 TCP 保持連線並控制持續時間。

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(AwsCrtHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

使用非同步 DynamoDB 用戶端時，您可以啟用 TCP Keep-Alive，如下列程式碼所示。

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

錯誤處理

處理例外狀況時， AWS SDK for Java 2.x 會使用執行時間 （ 未勾選 ） 例外狀況。

涵蓋所有 SDK 例外狀況的基本例外狀況是 SdkServiceException，其延伸自 Java 未檢查的 RuntimeException。如果您發現這種情況，您將會擷取到 SDK 擲回的所有例外狀況。

SdkServiceException 具有名為 AwsServiceException 的子類別。此子類別表示與 AWS 服務通訊時的任何問題。它有一個名為 DynamoDbException 的子類別，表示與 DynamoDB 通訊時發生問題。如果您發現這種情況，您將會擷取到與 DynamoDB 相關的所有例外狀況，但沒有其他 SDK 例外狀況。

在 DynamoDbException 下有更具體的例外狀況類型。其中一些例外狀況類型適用於控制平面操作，例如 TableAlreadyExistsException。其他則適用於資料平面操作。以下是常見的資料平面例外狀況範例：

- ConditionalCheckFailedException：您已指定評估為 false 的條件。例如，您可能已嘗試對項目執行條件式更新，但屬性的實際值不符合條件中的預期值。以此方式失敗的請求不會重試。

其他情況則未定義特定的例外狀況。例如，當您的請求受到限流時，可能會擲回特定 ProvisionedThroughputExceededException，而在其他情況下則擲回更通用的 DynamoDbException。在任何一種情況下，您都可以檢查 isThrottlingException() 是否傳回 true，以判斷限流是否造成例外狀況。

根據您的應用程式需求，您可以擷取所有 AwsServiceException 或 DynamoDbException 執行個體。不過，在不同的情況下，您通常需要不同的行為。處理條件檢查失敗的邏輯與處理限流的邏輯不同。定義您要處理的例外路徑，並務必測試替代路徑。這可協助您確保可以處理所有相關案例。

如需可能遇到的常見錯誤清單，請參閱 使用 DynamoDB 時發生錯誤。另請參閱《Amazon DynamoDB API 參考》中的常見錯誤。API 參考也為每個 API 操作 (例如 Query 操作) 提供可能的確切錯誤。如需處理例外狀況的資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的 AWS SDK for Java 2.x例外狀況處理。

AWS 請求 ID

每個請求都包含一個請求 ID，如果您使用 AWS 支援 來診斷問題，這對於提取該 ID 很有幫助。衍生自 SdkServiceException 的每個例外狀況都有可用來擷取請求 ID 的 requestId() 方法。

日誌

使用 SDK 提供的日誌記錄，對於從用戶端程式庫擷取任何重要訊息以及更深入的偵錯目的都很有用。記錄器是階層式的，SDK 使用 software.amazon.awssdk 做為其根記錄器。您可以使用 TRACE、DEBUG、INFO、WARN、ERROR、ALL 或 OFF 其中之一來設定層級。設定的層級會套用至該記錄器，並沿著記錄器階層向下套用。

對於其記錄， AWS SDK for Java 2.x 使用適用於 Java 的 Simple Logging Façade (SLF4J)。這可做為其他記錄器周圍的抽象層，您可以使用它來插入您偏好的記錄器。如需有關插入記錄器的說明，請參閱 SLF4J 使用者手冊。

每個記錄器都有特定的行為。根據預設，Log4j 2.x 記錄器會建立 ConsoleAppender，其會將日誌事件附加至 System.out，並預設為 ERROR 日誌層級。

SLF4J 輸出中包含的 SimpleLogger 記錄器預設為 System.err，並預設為 INFO 日誌層級。

對於任何生產部署，建議您將 WARN 的階層設定為 software.amazon.awssdk，以便從 SDK 的用戶端程式庫擷取任何重要訊息，同時限制輸出數量。

如果 SLF4J 在類別路徑上找不到支援的記錄器 (無 SLF4J 繫結)，則會預設為無操作實作。此實作會導致將訊息記錄於 System.err，說明 SLF4J 在 classpath 上找不到記錄器實作。若要避免這種情況，您必須新增記錄器實作。若要這樣做，您可以在 Apache Maven pom.xml 中新增對成品的相依性，例如 org.slf4j.slf4j-simple 或 org.apache.logging.log4j.log4j-slf4j2-imp。

如需有關如何在 SDK 中設定記錄的資訊，包括將記錄相依性新增至應用程式組態，請參閱《適用於 Java 的 AWS SDK 開發人員指南》中的使用適用於 Java 2.x 的 SDK 記錄。

如果您使用 Apache Log4j 2 記錄器，Log4j2.xml 檔案中的下列組態顯示如何調整記錄行為。此組態會將根記錄器層級設定為 WARN。階層中的所有記錄器都會繼承此記錄層級，包括 software.amazon.awssdk 記錄器。

根據預設，輸出會移至 System.out。在下列範例中，我們仍會覆寫預設輸出 Log4j 附加元件，以套用量身打造的 Log4j PatternLayout。

Log4j2.xml 組態檔案的 範例

下列組態會在所有記錄器階層的 ERROR 和 WARN 層級將訊息記錄到主控台。

```
<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 請求 ID 記錄

當發生問題時，您可以在例外狀況中找到請求 ID。不過，如果您想要未產生例外狀況的請求 ID，則可以使用記錄功能。

software.amazon.awssdk.request 記錄器會在 DEBUG 層級輸出請求 ID。下列範例延伸先前的 configuration example，將根記錄器層級保持在 ERROR、將 software.amazon.awssdk 保持在層級 WARN，以及將 software.amazon.awssdk.request 保持在層級 DEBUG。設定這些層級有助於擷取請求 ID 和其他請求相關詳細資訊，例如端點和狀態碼。

```
<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>
```

以下為日誌輸出的範例：

```
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
```

分頁

有些請求，例如 Query 和 Scan，會限制單一請求傳回的資料大小，並要求您重複提出請求以提取後續頁面。

您可以使用 Limit 參數控制每個頁面要讀取的項目數量上限。例如，您可以使用 Limit 參數僅擷取最後 10 個項目。此限制指定套用任何篩選之前要從資料表讀取的項目數量。如果您在篩選後只想要 10 個項目，則無法指定該項目數。您只能控制預先篩選的計數，並在實際擷取 10 個項目時在用戶端檢查。無論限制為何，回應的大小上限一律為 1 MB。

API 回應中可能包含 LastEvaluatedKey。這表示回應因為達到計數限制或大小限制而結束。此金鑰是針對該回應評估的最後一個金鑰。透過直接與 API 互動，您可以擷取此 LastEvaluatedKey，並將其傳遞至後續呼叫做為 ExclusiveStartKey，以便從該起點讀取下一個區塊。如果沒有傳回任何 LastEvaluatedKey，表示沒有更多符合 Query 或 Scan API 呼叫的項目。

下列範例使用低階介面，根據 keyConditionExpression 參數將項目限制為 100。

```
QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
        .expressionAttributeValues(Map.of(
                ":pk_val", AttributeValue.fromS("123"),
                ":sk_val", AttributeValue.fromN("1000")))
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName(TABLE_NAME);

while (true) {
    QueryResponse queryResponse = DYNAMODB_CLIENT.query(queryRequestBuilder.build());

    queryResponse.items().forEach(item -> {
        LOGGER.info("item PK: [" + item.get("pk") + "] and SK: [" + item.get("sk") +
 "]");
    });

    if (!queryResponse.hasLastEvaluatedKey()) {
        break;
    }
    queryRequestBuilder.exclusiveStartKey(queryResponse.lastEvaluatedKey());
}
```

AWS SDK for Java 2.x 可以透過提供自動分頁方法，進行多次服務呼叫，以自動為您取得後續頁面的結果，簡化與 DynamoDB 的互動。這樣可以簡化您的程式碼，但它也移除了手動讀取頁面時所能保持對資源用量的一些控制。

透過使用 DynamoDB 用戶端中可用的 Iterable 方法，例如 QueryPaginator 和 ScanPaginator，SDK 會負責處理分頁。這些方法的傳回類型是自訂可迭代物件，可用來逐一查看所有頁面。SDK 會在內部為您處理服務呼叫。您可以使用 Java 串流 API 處理 QueryPaginator 的結果，如下列範例所示。

```
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();
```

資料類別註釋

Java SDK 提供數個註釋，您可以將這些註釋放在資料類別的屬性上。這些註釋會影響 SDK 如何與屬性互動。透過新增註釋，您可以讓屬性做為隱含原子計數器、維護自動產生的時間戳記值，或追蹤項目版本號碼。如需詳細資訊，請參閱資料類別註釋。

使用 DynamoDB 時發生錯誤

本節說明執行時期錯誤和其處理方式。其中也會說明 Amazon DynamoDB 特有的錯誤訊息和程式碼。如需適用於所有 AWS 服務的常見錯誤清單，請參閱存取管理

主題

- 錯誤元件
- 交易性錯誤
- 錯誤訊息和錯誤碼
- 應用程式中的錯誤處理
- 錯誤重試和指數退避
- 批次操作和錯誤處理

錯誤元件

當您的程式傳送請求時，DynamoDB 會嘗試進行處理。如果請求成功，則 DynamoDB 會傳回 HTTP 成功狀態碼 (200 OK) 以及所請求操作的結果。

如果請求不成功，DynamoDB 會傳回錯誤。每個錯誤都會有三個元件：

- HTTP 狀態碼 (例如 400)。

- 例外狀況名稱 (例如 `ResourceNotFoundException`)。
- 錯誤訊息 (例如 `Requested resource not found: Table: tablename not found`)。

AWS SDKs 負責將錯誤傳播到您的應用程式，以便您可以採取適當的動作。例如，在 Java 程式中，您可以撰寫 `try-catch` 邏輯來處理 `ResourceNotFoundException`。

如果您不是使用 AWS SDK，則需要從 DynamoDB 剖析低階回應的內容。以下是這類回應的範例。

```
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"}
```

交易性錯誤

如需有關交易性錯誤的資訊，請參閱 DynamoDB 中的交易衝突處理

錯誤訊息和錯誤碼

以下是 DynamoDB 依 HTTP 狀態碼進行分組並傳回的例外狀況清單。確定要重試嗎？為是，則可以重新提交相同的請求。確定要重試嗎？為否，則需要先修正使用者端上的問題，再提交新的請求。

HTTP 狀態碼 400

HTTP `400` 狀態碼指出請求發生問題，例如身分驗證失敗、遺失必要參數，或超過資料表的佈建輸送量。您必須先修正應用程式中的問題，再重新提交請求。

AccessDeniedException

訊息：Access denied. (存取遭拒。)

用戶端未正確地簽署請求。如果您使用的是 AWS 軟體開發套件，系統會自動為您簽署請求；如果不是，則請移至AWS 一般參考中的 Signature 第 4 版簽署程序。

OK to retry? (確定要重試嗎？) 否

ConditionalCheckFailedException

訊息：The conditional request failed. (條件式請求失敗。)

您已指定評估為 false 的條件。例如，您可能已嘗試對項目執行條件式更新，但屬性的實際值不符合條件中的預期值。

OK to retry? (確定要重試嗎？) 否

IncompleteSignatureException

訊息：The request signature does not conform to standards. (請求簽章不符合 AWS 標準。)

請求簽章未包含所有必要元件。如果您使用的是 AWS 開發套件，系統會自動為您簽署請求；否則，請前往 中的 Signature 第 4 版簽署程序AWS 一般參考。

OK to retry? (確定要重試嗎？) 否

ItemCollectionSizeLimitExceededException

訊息：Collection size exceeded. (已超過集合大小。)

針對具有本機次要索引的資料表，具有相同分割區索引鍵值的項目群組已超過 10 GB 的大小上限。如需項目集合的詳細資訊，請參閱 本機次要索引中的項目集合。

OK to retry? (確定要重試嗎？) 是

LimitExceededException

訊息：Too many operations for a given subscriber. (指定訂閱用戶的操作太多。)

並行控制平面操作太多。在 CREATING、DELETING 或 UPDATING 狀態中，資料表和索引的累積數量不能超過 500。

OK to retry? (確定要重試嗎？) 是

MissingAuthenticationTokenException

訊息：Request must contain a valid (registered) AWS Access Key ID. (請求必須包含有效 (已註冊) AWS 存取金鑰 ID。)

請求未包含必要授權標頭，或其格式不正確。請參閱 DynamoDB 低階 API。

OK to retry? (確定要重試嗎？) 否

ProvisionedThroughputExceededException

訊息：You exceeded your maximum allowed provisioned throughput for a table or for one or more global secondary indexes. To view performance metrics for provisioned throughput vs. consumed throughput, open the Amazon CloudWatch console. (您已超過資料表或一或多個全域次要索引的最大允許佈建輸送量。若要檢視佈建輸送量與使用輸送的效能指標，請開啟 Amazon CloudWatch 主控台。)

錯誤包含 `ThrottlingReason` 欄位清單，提供為何發生限流的特定內容，並遵循格式 `ResourceType+OperationType+LimitType` (例如 `TableReadProvisionedThroughputExceeded`) 和受影響資源的 ARN。這可協助您識別要限流的資源 (資料表或索引)、觸發限流的操作類型 (讀取或寫入)，以及超過的特定限制 (在此案例中為佈建容量)。如需診斷和解決限流問題的詳細資訊，請參閱 限流診斷。

範例：您的請求率太高。DynamoDB AWS SDKs 會自動重試收到此例外狀況的請求。除非您的重試佇列太大無法完成，否則您的請求最後會成功。請使用 錯誤重試和指數退避 來減少請求的頻率。

OK to retry? (確定要重試嗎？) 是

ReplicatedWriteConflictException

訊息：另一個區域中的請求正在修改此請求中的一或多個項目。

範例：針對多區域強式一致性 (MRSC) 全域資料表中的項目請求寫入操作，該資料表目前正由另一個區域中的請求修改。

OK to retry? (確定要重試嗎？) 是

RequestLimitExceeded

訊息：輸送量超過帳戶的目前輸送量限制。若要請求提高限制，請聯絡 AWS Support，網址為 https：//https://aws.amazon.com/support。

錯誤包含 `ThrottlingReason` 欄位清單，提供為何發生限流的特定內容，並遵循格式 `ResourceType+OperationType+LimitType` (例如 `TableWriteAccountLimitExceeded`

或 IndexReadAccountLimitExceeded) 和受影響資源的 ARN。這可協助您識別正在限流的資源 (資料表或索引)、觸發限流的操作類型 (讀取或寫入)，以及已超過帳戶層級的服務配額。如需診斷和解決限流問題的詳細資訊，請參閱 限流診斷。

範例：隨選請求率超過允許的帳戶輸送量且資料表無法進一步擴展。

OK to retry? (確定要重試嗎？) 是

ResourceInUseException

訊息：The resource which you are attempting to change is in use. (正在使用您嘗試變更的資源。)

範例：您已嘗試重新建立現在資料表，或刪除目前處於 CREATING 狀態的資料表。

OK to retry? (確定要重試嗎？) 否

ResourceNotFoundException

訊息：Requested resource not found. (找不到已註冊的資源。)

範例：所請求的資料表不存在，或太早處於 CREATING 狀態。

OK to retry? (確定要重試嗎？) 否

ThrottlingException

訊息：Rate of requests exceeds the allowed throughput. (請求率超過允許的輸送量。)

這個例外狀況會傳回為 AmazonServiceException 回應，並具備 THROTTLING_EXCEPTION 狀態碼。如果您執行控制平面 API 操作太快，可能會傳回此例外狀況。

對於使用隨需的資料表，如果您的請求率太高，任何資料平面 API 操作都可能會傳回此例外狀況。若要進一步了解隨需擴展，請參閱 初始輸送量與擴展特性。

錯誤包含 ThrottlingReason 欄位清單，提供為何發生限流的特定內容，並遵循格式 ResourceType+OperationType +LimitType (例如 TableReadKeyRangeThroughputExceeded 或 IndexWriteMaxOnDemandThroughputExceeded) 和受影響資源的 ARN。此資訊可協助您識別要限流的資源 (資料表或索引)、觸發限流的操作類型 (讀取或寫入)，以及超過的特定限制 (分割區限制或隨需最大輸送量)。如需診斷和解決限流問題的詳細資訊，請參閱 限流診斷。

OK to retry? (確定要重試嗎？) 是

UnrecognizedClientException

訊息：The Access Key ID or security token is invalid. (存取金鑰 ID 或安全字符無效。)

請求簽章不正確。最可能的原因是無效的 AWS 存取金鑰 ID 或私密金鑰。

OK to retry? (確定要重試嗎？) 是

ValidationException

訊息：Varies, depending upon the specific error(s) encountered (會根據發生的特定錯誤而不同)

可能會因數個原因而發生此錯誤，例如必要參數遺失、值超出範圍，或資料類型不符。錯誤訊息包含導致錯誤之請求特定部分的詳細資訊。

OK to retry? (確定要重試嗎？) 否

HTTP 狀態碼 5xx

HTTP 5xx 狀態碼指出 AWS必須解決的問題。這可能是暫時性錯誤，在此情況下，您可以重試請求，直到成功為止。如果不是，請前往 AWS Service Health Dashboard，確認服務是否發生任何操作問題。

如需詳細資訊，請參閱如何解決 Amazon DynamoDB 中的 HTTP 5xx 錯誤？

InternalServerError (HTTP 500)

DynamoDB 無法處理您的請求。

OK to retry? (確定要重試嗎？) 是

> **Note**
>
> 處理項目時，您可能會發生內部伺服器錯誤。在資料表生命週期期間，這些是預期錯誤。可以立即重試任何失敗的請求。
> 當您在寫入作業上收到狀態碼 500 時，作業可能已成功或失敗。如果寫入操作是 `TransactWriteItem` 請求，則可以重試操作。如果寫入作業是單一項目寫入要求，例如 `PutItem`、`UpdateItem`，或 `DeleteItem`，那麼您的應用程序應該在重試操作之前讀取項目的狀態，和/或使用 DynamoDB 條件表達式 CLI 範例，以確保項目在重試後保

持正確的狀態，無論先前的作業是成功還是失敗。如果冪等性是寫入操作的要求，請使用 `TransactWriteItem`，它通過自動指定 `ClientRequestToken` 來消除多次嘗試執行相同動作的歧義。

ServiceUnavailable (HTTP 503)

DynamoDB 目前無法使用。(這應該是暫時狀態)。

OK to retry? (確定要重試嗎？) 是

應用程式中的錯誤處理

若要讓您的應用程式順暢執行，您需要新增邏輯來截獲並回應錯誤。一般方式包含使用 `try-catch` 區塊或 `if-then` 陳述式。

AWS SDKs 會執行自己的重試和錯誤檢查。如果您在使用其中一個 AWS SDKs時發生錯誤，錯誤代碼和描述可協助您進行疑難排解。

您應該也會在回應中看到 `Request ID`。如果您需要與 AWS Support 合作來診斷問題， `Request ID`會很有幫助。

錯誤重試和指數退避

網路上有許多元件 (例如 DNS 伺服器、交換器、負載平衡器和其他項目) 可以在指定請求之生命週期中的任何階段產生錯誤。一般在網路環境中處理這些錯誤回應的技術，就是在用戶端應用程式中實作重試。這技術會提高應用程式的可靠性。

每個 AWS SDK 會自動實作重試邏輯。您可以視需要修改重試參數。例如，請考慮使用需要快速檢錯策略 (即發生錯誤時不允許重試) 的 Java 應用程式。透過 適用於 Java 的 AWS SDK，您可以使用 `ClientConfiguration`類別並提供 `maxErrorRetry`的值`0`來關閉重試。如需詳細資訊，請參閱程式設計語言的 AWS SDK 文件。

如果您不是使用 AWS SDK，您應該重試接收伺服器錯誤的原始請求 (5xx)。不過，用戶端錯誤 (4xx，非 `ThrottlingException` 或 `ProvisionedThroughputExceededException`) 指出您需要先修訂請求本身更正問題，然後再試一次。如需解決特定限流案例的詳細建議，請參閱 DynamoDB 限流疑難排解一節。

除了簡單的重試之外，每個 AWS SDK 還實作指數退避演算法，以更好地控制流程。指數退避的背後概念是，針對連續錯誤回應，讓重試之間的等待時間漸進拉長。例如，最多 50 毫秒再進行第一次重

試、最多 100 毫秒再進行第二次重試、最多 200 毫秒再進行第三次重試，以此類推。不過，在一分鐘之後，如果請求未成功，則問題可能是請求大小超過佈建輸送量，而不是請求率。設定大約一分鐘停止的最大重試次數。如果請求未成功，請調查您的佈建輸送量選項。

> **Note**
>
> AWS SDKs 實作自動重試邏輯和指數退避。

大多數指數退避演算法會使用抖動 (隨機延遲)，以防止連續衝突。因為您在這些情況下並未嘗試避免這種衝突，所以不需要使用此亂數。不過，如果您使用並行用戶端，抖動有助於讓請求更快取得成功。如需詳細資訊，請參閱關於指數退避和抖動的部落格文章。

批次操作和錯誤處理

DynamoDB 低階 API 支援讀取和寫入的批次操作。`BatchGetItem` 從一或多個資料表讀取項目，`BatchWriteItem` 在一或多個資料表中放置或刪除項目。在其他非批次 DynamoDB 操作中，這些批次操作會實作為包裝函式。換言之，`BatchGetItem` 會針對批次中的每個項目調用 `GetItem` 一次。同樣地，`BatchWriteItem` 會針對批次中的每個項目適當地調用 `DeleteItem` 或 `PutItem`。

批次操作可以容忍批次中個別請求的失敗。例如，請考慮使用 `BatchGetItem` 請求來讀取五個項目。即使部分基礎 `GetItem` 請求失敗，這也不會導致整個 `BatchGetItem` 操作失敗。但若全部五個讀取操作均失敗，則整個 `BatchGetItem` 便會失敗。

批次操作會傳回失敗個別請求的資訊，讓您可以診斷問題，並重試操作。針對 `BatchGetItem`，會在回應的 `UnprocessedKeys` 值中傳回有問題的資料表和主索引鍵。針對 `BatchWriteItem`，在 `UnprocessedItems` 中會傳回類似的資訊。

最可能的失敗讀取或失敗寫入原因是限流。針對 `BatchGetItem`，批次請求中的一或多個資料表沒有足夠的已佈建讀取容量可支援操作。針對 `BatchWriteItem`，一或多個資料表沒有足夠的已佈建寫入容量。

如果 DynamoDB 傳回任何未處理的項目，您應該對這些項目重試批次操作。不過，強烈建議您使用指數退避演算法。如果您立即重試批次操作，則基於個別資料表上的限流，基礎讀取或寫入請求仍然可能會失敗。如果您使用指數退避來延遲批次操作，則批次中的個別請求較可能會成功。

> **Note**
>
> AWS SDKs提供更高層級的用戶端，可自動處理未處理的項目重試，因此您不需要自行實作此重試邏輯。例如：

- Java – 適用於 Java 的 AWS SDK v2 中的 DynamoDB 增強型用戶端和 v1 中的 DynamoDBMapper 都會在執行批次操作時自動重試未處理的項目。
- Python – boto3 Table 資源會隱含地`batch_writer`處理未處理的項目重試，以進行批次寫入操作。如需詳細資訊，請參閱the section called “使用 batch_writer”。

如果您使用的是低階用戶端或不提供此行為的 SDK，您必須自行實作上述的重試邏輯。

搭配 AWS SDK 使用 DynamoDB

AWS 軟體開發套件 (SDKs) 適用於許多熱門的程式設計語言。每個 SDK 都提供 API、程式碼範例和說明文件，讓開發人員能夠更輕鬆地以偏好的語言建置應用程式。

SDK 文件	代碼範例
適用於 C++ 的 AWS SDK	適用於 C++ 的 AWS SDK 程式碼範例
AWS CLI	AWS CLI 程式碼範例
適用於 Go 的 AWS SDK	適用於 Go 的 AWS SDK 程式碼範例
適用於 Java 的 AWS SDK	適用於 Java 的 AWS SDK 程式碼範例
適用於 JavaScript 的 AWS SDK	適用於 JavaScript 的 AWS SDK 程式碼範例
適用於 Kotlin 的 AWS SDK	適用於 Kotlin 的 AWS SDK 程式碼範例
適用於 .NET 的 AWS SDK	適用於 .NET 的 AWS SDK 程式碼範例
適用於 PHP 的 AWS SDK	適用於 PHP 的 AWS SDK 程式碼範例
AWS Tools for PowerShell	AWS Tools for PowerShell 程式碼範例
適用於 Python (Boto3) 的 AWS SDK	適用於 Python (Boto3) 的 AWS SDK 程式碼範例
適用於 Ruby 的 AWS SDK	適用於 Ruby 的 AWS SDK 程式碼範例

SDK 文件	代碼範例
適用於 Rust 的 AWS SDK	適用於 Rust 的 AWS SDK 程式碼範例
適用於 SAP ABAP 的 AWS SDK	適用於 SAP ABAP 的 AWS SDK 程式碼範例
適用於 Swift 的 AWS SDK	適用於 Swift 的 AWS SDK 程式碼範例

如需 DynamoDB 專屬範例，請參閱使用 AWS SDKs DynamoDB 程式碼範例。

> **可用性範例**
>
> 找不到所需的內容嗎？ 請使用本頁面底部的提供意見回饋連結申請程式碼範例。

使用資料表、項目、查詢、掃描和索引

本節提供在 Amazon DynamoDB 中使用資料表、項目、查詢和更多內容的詳細資訊。

主題

- 在 DynamoDB 中使用資料表和資料
- 全域資料表：多重作用中的多區域複寫
- 在 DynamoDB 使用項目和屬性
- 在 DynamoDB 中使用次要索引來改善資料存取
- 管理 DynamoDB 交易的複雜工作流程
- 使用 Amazon DynamoDB 變更資料擷取

在 DynamoDB 中使用資料表和資料

本節說明如何使用 AWS Command Line Interface (AWS CLI) 和 AWS SDKs 在 Amazon DynamoDB 中建立、更新和刪除資料表。

> **Note**
>
> 您也可以使用 AWS 管理主控台來執行這些相同的任務。如需詳細資訊，請參閱使用主控台。

本節也會使用 DynamoDB Auto Scaling 或手動設定佈建輸送量來提供輸送容量的詳細資訊。

主題

- DynamoDB 資料表上的基本操作
- 選擇 DynamoDB 資料表類別時的注意事項
- 將標籤新增至 DynamoDB 中的資源

DynamoDB 資料表上的基本操作

與其他資料庫系統類似，Amazon DynamoDB 會將資料存放在資料表中。您可使用幾個基本操作來管理您的資料表。

主題

- 建立資料表
- 說明資料表
- 更新資料表
- 刪除資料表
- 使用刪除保護
- 列出資料表名稱
- 說明佈建輸送量配額

建立資料表

使用 `CreateTable` 操作在 Amazon DynamoDB 中建立資料表。若要建立資料表，您必須提供以下資訊：

- 資料表名稱 名稱必須符合 DynamoDB 命名規則，且目前 AWS 帳戶和區域必須是唯一的。例如，您可以在美國東部 (維吉尼亞北部) 建立 `People` 資料表，在歐洲 (愛爾蘭) 建立另一個 `People` 資料表。不過，這兩個資料表彼此必須完全不同。如需詳細資訊，請參閱 Amazon DynamoDB 中支援的資料類型和命名規則。
- 主索引鍵。主索引鍵可以包含一個屬性 (分割區索引鍵) 或兩個屬性 (分割區索引鍵和排序索引鍵)。您需要提供屬性名稱、資料類型和每個屬性的角色：HASH (適用於分割區索引鍵) 和 RANGE (適用於排序索引鍵)。如需詳細資訊，請參閱 主索引鍵。
- 輸送量設定 (針對佈建的資料表)。如果使用佈建的模式，您必須指定資料表的讀取和寫入初始輸送量設定。您稍後可以修改這些設定，或讓 DynamoDB Auto Scaling 功能來管理設定。如需詳細資訊，請參閱DynamoDB 佈建容量模式及使用 DynamoDB Auto Scaling 功能自動管理輸送容量。

範例 1：建立隨需資料表

使用隨需模式來建立相同的資料表 `Music`。

```
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
```

CreateTable 操作會傳回資料表的中繼資料，如下所示。

```
{
    "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**
>
> 呼叫隨需資料表上的 DescribeTable 時，讀取容量單位與寫入容量單位設為 0。

範例 2：建立佈建資料表

下列 AWS CLI 範例示範如何建立資料表 (Music)。主索引鍵包含 Artist (分割區索引鍵) 和 SongTitle (排序索引鍵)，而且它們每一個的資料類型都是 String。此資料表的輸送量上限是 10 個讀取容量單位和 5 個寫入容量單位。

```
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
```

CreateTable 操作會傳回資料表的中繼資料，如下所示。

```
{
    "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
    }
}
```

TableStatus 元素指出資料表的目前狀態 (CREATING)。根據您指定的 ReadCapacityUnits 和 WriteCapacityUnits 值，可能需要一些時間才能建立資料表。這些項目的較大值需要 DynamoDB 為資料表配置更多資源。

範例 3：使用 DynamoDB 標準的不常存取資料表類別建立資料表

若要建立相同的 Music 資料表，使用 DynamoDB 標準：不常存取資料表類別。

```
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
```

CreateTable 操作會傳回資料表的中繼資料，如下所示。

```
{
    "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
    }
}
```

說明資料表

若要檢視資料表的詳細資訊，請使用 DescribeTable 操作。您必須提供資料表名稱。DescribeTable 中輸出的格式與 CreateTable 相同。它包含建立資料表的時間戳記、其索引鍵結構描述、其佈建的輸送量設定、其預估大小以及任何現有的次要索引。

> **Important**
>
> 呼叫隨需資料表上的 DescribeTable 時，讀取容量單位與寫入容量單位設為 0。

Example

```
aws dynamodb describe-table --table-name Music
```

TableStatus 從 CREATING 變更為 ACTIVE 之後，就表示資料表準備就緒可供使用。

> **Note**
>
> 如果您在 CreateTable 請求之後立即發出 DescribeTable 請求，則 DynamoDB 可能會傳回錯誤 (ResourceNotFoundException)。原因是 DescribeTable 使用最終一致查詢，而且您資料表的中繼資料目前可能無法使用。請等待幾秒，然後再次重試 DescribeTable 請求。
>
> 為計算費用，您的 DynamoDB 儲存費用包含 100 個位元組的個別項目成本。(如需詳細資訊，請前往 DynamoDB 定價。) 個別項目的這個額外 100 個位元組不會用於容量單位計算，或不供 DescribeTable 操作使用。

更新資料表

UpdateTable 操作可讓您執行下列其中一項：

- 修改資料表的佈建輸送量設定 (針對佈建模式的資料表)。
- 變更資料表的讀取/寫入容量模式。
- 在資料表中修改全域次要索引 (請參閱在 DynamoDB 中使用全域次要索引)。
- 啟用或停用資料表上的 DynamoDB Streams (請參閱 DynamoDB Streams 的變更資料擷取)。

Example

下列 AWS CLI 範例顯示如何修改資料表的佈建輸送量設定。

```
aws dynamodb update-table --table-name Music \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10
```

> **Note**
>
> 當您發出 UpdateTable 請求時，資料表的狀態會從 AVAILABLE 變更為 UPDATING。資料表在處於 UPDATING 時仍然完全可供使用。此程序完成時，資料表狀態會從 UPDATING 變更為 AVAILABLE。

Example

下列 AWS CLI 範例顯示如何將資料表的讀取/寫入容量模式修改為隨需模式。

```
aws dynamodb update-table --table-name Music \
    --billing-mode PAY_PER_REQUEST
```

刪除資料表

您可以使用 DeleteTable 操作來移除未使用的資料表。刪除資料表是無法復原的操作。使用 AWS 管理主控台刪除資料表，請參閱 the section called “步驟 6：（選用）清除”。

Example

下列 AWS CLI 範例顯示如何刪除資料表。

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

當您發出 DeleteTable 請求時，資料表的狀態會從 ACTIVE 變更為 DELETING。根據使用的資源 (例如資料表中所存放的資料，以及資料表上的任何串流或索引)，可能需要一些時間才能刪除資料表。

DeleteTable 操作結束時，資料表就不再存在於 DynamoDB 中。

使用刪除保護

您可以使用刪除保護屬性，來保護資料表免於遭到意外刪除。為資料表啟用此屬性有助於確保管理員在常態資料表管理操作期間，不會意外刪除資料表。這將有助於防止您的正常業務營運中斷。

資料表擁有者或授權管理員可控制每個資料表的刪除保護內容。每個資料表的刪除保護屬性預設是關閉的。這包括全域複本，以及從備份還原的資料表。停用資料表的刪除保護時，任何獲得 Identity and Access Management (IAM) 政策授權的使用者都可以刪除該資料表。資料表啟用刪除保護時，沒有人可將其刪除。

若要變更此設定，請移至資料表的其他設定，瀏覽至刪除保護面板，然後選取啟用刪除保護。

刪除保護屬性受到 DynamoDB 主控台、API、CLI/SDK 和 CloudFormation的支援。`CreateTable` API 支援資料表建立時的刪除保護屬性，而 `UpdateTable` API 支援變更現有資料表的刪除保護屬性。

> **Note**
>
> - 如果刪除 AWS 帳戶，該帳戶的所有資料，包括資料表，仍會在 90 天內刪除。
> - 如果 DynamoDB 喪失用來加密資料表的客戶自管金鑰存取權，它仍會封存資料表。存檔涉及製作資料表的備份和刪除原始資料表。

列出資料表名稱

`ListTables` 操作會傳回目前 AWS 帳戶和區域的 DynamoDB 資料表名稱。

Example

下列 AWS CLI 範例示範如何列出 DynamoDB 資料表名稱。

```
aws dynamodb list-tables
```

說明佈建輸送量配額

`DescribeLimits` 操作會傳回目前 AWS 帳戶和區域的目前讀取和寫入容量配額。

Example

下列 AWS CLI 範例示範如何描述目前的佈建輸送量配額。

```
aws dynamodb describe-limits
```

輸出會顯示目前 AWS 帳戶和區域的讀取和寫入容量單位配額上限。

如需這些配額以及如何請求提高配額的詳細資訊，請參閱輸送量預設配額。

選擇 DynamoDB 資料表類別時的注意事項

DynamoDB 提供兩種資料表類別，旨在協助您最佳化成本。預設值為 DynamoDB 標準資料表類別，建議大多數工作負載使用。DynamoDB 標準：不常存取 (DynamoDB 標準-IA) 資料表類別針對以儲存為主要成本的資料表進行最佳化。例如，儲存不常存取資料的資料表 (例如應用程式日誌、舊社交媒體貼文、電子商務訂單歷史記錄以及過去遊戲成就) 都是適合標準-IA 資料表類別的選項。

每個 DynamoDB 資料表都與資料表類別相關聯。與資料表相關聯的所有次要索引都使用相同的資料表類別。您可以在建立資料表時設定資料表類別（預設為 DynamoDB 標準），並使用 、AWS 管理主控台 AWS CLI 或 AWS SDK 更新現有資料表的資料表類別。DynamoDB 也支援 AWS CloudFormation 針對單一區域資料表（非全域資料表的資料表）使用 管理資料表類別。每個資料表類別針對資料儲存以及讀取和寫入要求提供不同的定價。當為您的表選擇一個表類時，請記住下列事項：

- DynamoDB 標準資料表類別提供比 DynamoDB 標準-IA 更低的輸送量成本，對於輸送量是主要成本的資料表來說，是最具成本效益的選項。
- DynamoDB 標準 IA 資料表類別提供比 DynamoDB 標準更低的儲存成本，對於儲存成為主要成本的資料表來說，是最具成本效益的選項。當儲存超過使用 DynamoDB 標準資料表類別的資料表輸送量 (讀取和寫入) 成本的 50% 時，DynamoDB 標準-IA 資料表類別可協助您降低資料表總成本。
- DynamoDB 標準 – IA 資料表提供與 DynamoDB 標準資料表相同的效能、耐用性和可用性。
- 在 DynamoDB 標準資料表與 DynamoDB 標準 – IA 資料表類之間切換不需要變更應用程式的程式碼。不論您的資料表使用何種資料表類型，都可以使用相同的 DynamoDB API 和服務端點。
- DynamoDB 標準 – IA 資料表與所有現有的 DynamoDB 功能相容，例如自動擴展、隨需模式、存留時間 (TTL)、隨需備份、時間點復原 (PITR) 和全域次要索引。

資料表最具成本效益的資料表類別取決於資料表預期的儲存體和輸送量使用模式。您可以使用成本和用量報告和 AWS Cost Explorer，查看資料表的歷史儲存和輸送量 AWS 成本和用量。使用此歷史資料來確定資料表最具成本效益的資料表類別。若要進一步了解如何使用 AWS 成本和用量報告和 AWS Cost Explorer管，請參閱AWS 帳單和成本管理文件。如需資料表類別定價詳細資訊，請參閱 Amazon DynamoDB 定價。

> **Note**
>
> 資料表類別更新是一個背景流程。您仍然可以在資料表類別更新期間正常存取資料表。更新表類的時間取決於您的表流量，儲存大小和其他相關變量。在 30 天的追蹤期間內，資料表上不允許兩個以上的資料表類別更新。

將標籤新增至 DynamoDB 中的資源

您可以使用 tags 來標示 Amazon DynamoDB 資源。標籤可讓您以不同的方式分類您的資源，例如依據目的、所有者、環境或其他條件。標籤可協助您執行以下作業：

- 根據您指派給資源的標籤來快速識別資源。
- 請參閱依標籤細分的 AWS 帳單。

> **Note**
>
> 任何與標記資料表相關的本機次要索引 (LSI) 和全域次要索引 (GSI) 都會自動標上相同的標籤。目前無法標記 DynamoDB Streams 使用量。

Amazon EC2、Amazon S3、DynamoDB 等 AWS 服務支援標記。有效標記可讓您跨帶有特定標籤的服務來建立報告，以提供成本深入資訊。

若要開始使用標記，請執行以下操作：

1. 了解 DynamoDB 中的標記限制。
2. 使用 在 DynamoDB 中標記資源 建立標籤。
3. 使用 使用 DynamoDB 標籤建立成本分配報告 追蹤每個作用中標籤 AWS 的成本。

最後，最好遵循最佳標記策略。如需相關資訊，請參閱 AWS 標記策略。

DynamoDB 中的標記限制

每個標記皆包含由您定義的索引鍵和值。適用以下限制：

- 每個 DynamoDB 資料表都只能有一個具有相同索引鍵的標籤。若您嘗試新增現有的標籤 (相同索引鍵)，現有標籤的值會更新為新的值。
- 標籤鍵與值皆區分大小寫。
- 鍵長度上限為 128 個 Unicode 字元。
- 值長度上限為 256 個 Unicode 字元。
- 允許的字元為字母、空格和數字，以及下列特殊字元：`+ - = . _ : /`
- 每一資源標籤數最多為 50。

- 資料表中所有標籤支援的大小上限為 10 KB。
- AWS指派的標籤名稱和值會自動指派您無法指派的aws:字首。 AWS指派的標籤名稱不會計入標籤限制 50 或 10 KB 的大小上限。使用者指派的標籤名稱在成本分配報告中具有字首 user:。
- 標籤的套用不可回溯。

在 DynamoDB 中標記資源

您可以使用 Amazon DynamoDB 主控台或 AWS Command Line Interface (AWS CLI) 來新增、列出、編輯或刪除標籤。然後，您可以啟動這些使用者定義的標籤，讓它們出現在 AWS 帳單與成本管理 主控台中，以便追蹤成本配置。如需詳細資訊，請參閱使用 DynamoDB 標籤建立成本分配報告。

您也可以使用 AWS 管理主控台上的標籤編輯器進行大量編輯。如需詳細資訊，請參閱使用標籤編輯器。

若要改用 DynamoDB API，請參閱《Amazon DynamoDB API 參考》中的下列操作：

- TagResource
- UntagResource
- ListTagsOfResource

主題

- 設定依標籤篩選的許可
- 將標籤新增至新的或現有資料表 (AWS 管理主控台)
- 將標籤新增至新的或現有資料表 (AWS CLI)

設定依標籤篩選的許可

若要使用標籤篩選 DynamoDB 主控台中的資料表清單，請確定使用者的政策包含對下列操作的存取權：

- `tag:GetTagKeys`
- `tag:GetTagValues`

您可以依照下列步驟，將新的 IAM 政策連接至使用者，以便存取這些操作。

1. 使用管理員使用者前往 IAM 主控台。

2. 在左導覽選單中，選取 Policies (政策)。
3. 選取 Create Policy (建立政策)。
4. 將下列政策貼入至 JSON 編輯器。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        }
    ]
}
```

5. 完成協助程式並指派政策的名稱 (例如 `TagKeysAndValuesReadAccess`)。
6. 在左側導覽選單中，選擇 Users (使用者)。
7. 從清單中選取您通常用於存取 DynamoDB 主控台的使用者。
8. 選取 Add permissions (新增許可)。
9. 選取 Attach existing policies directly (直接連接現有政策)。
10. 從清單中選取您先前建立的政策。
11. 完成協助程式。

將標籤新增至新的或現有資料表 (AWS 管理主控台)

您可以使用 DynamoDB 主控台，在建立新資料表時在其中新增標籤，或為現有的資料表新增、編輯或刪除標籤。

在建立時為資源加上標籤 (主控台)

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。

2. 在導覽窗格中，選擇 Tables (資料表)，然後選擇 Create table (建立資料表)。
3. 在 Create DynamoDB table (建立 DynamoDB 資料表) 頁面上，提供名稱和主索引鍵。在 Tags (標籤) 區段中，選擇 Add new tag (新增標籤)，然後輸入您要使用的標籤。

   如需標籤結構的相關資訊，請參閱 DynamoDB 中的標記限制。

   如需建立資料表的詳細資訊，請參閱 DynamoDB 資料表上的基本操作。

為現有的資源加上標籤 (主控台)

請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。

1. 在導覽窗格中，選擇 Tables (資料表)。
2. 在清單中選擇資料表，然後選擇 Additional settings (其他設定) 索引標籤。您可以新增、編輯或刪除 Tags (標籤) 頁面底部的區段。

將標籤新增至新的或現有資料表 (AWS CLI)

下列範例示範如何在建立資料表和索引時，使用 AWS CLI 指定標籤，以及標記現有資源。

在建立時為資源加上標籤 (AWS CLI)

- 以下範例會建立新 `Movies` 資料表並使用 `blueTeam` 值來新增 `Owner` 標籤：

```
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
```

為現有的資源加上標籤 (AWS CLI)

- 以下範例會為 `Movies` 資料表新增 `blueTeam` 值的 `Owner` 標籤：

```
aws dynamodb tag-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies \
    --tags Key=Owner,Value=blueTeam
```

列出資料表的所有標籤 (AWS CLI)

- 以下範例列出與 Movies 資料表相關聯的所有標籤：

```
aws dynamodb list-tags-of-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies
```

使用 DynamoDB 標籤建立成本分配報告

AWS 使用標籤來整理成本分配報告中的資源成本。 AWS 提供兩種類型的成本分配標籤：

- AWS產生的 tag. AWS defines、建立和套用此標籤。
- 使用者定義的標籤。您可以定義、建立和套用這些標籤。

您必須分別啟用這兩種標籤，它們才會顯示在 Cost Explorer 或成本分配報告中。

若要啟用 AWS產生的標籤：

1. 登入 AWS 管理主控台 並開啟 Billing and Cost Management 主控台，網址為 https：//https://console.aws.amazon.com/billing/home#/。
2. 在導覽窗格中，選擇 Cost Allocation Tags (成本分配標籤)。
3. 在 AWS-Generated Cost Allocation Tags (AWS產生的成本分配標籤) 下，選擇 Activate (啟用)。

若要啟用使用者定義的標籤：

1. 登入 AWS 管理主控台 並開啟 Billing and Cost Management 主控台，網址為 https：//https://console.aws.amazon.com/billing/home#/。
2. 在導覽窗格中，選擇 Cost Allocation Tags (成本分配標籤)。
3. 在 User-Defined Cost Allocation Tags (使用者定義的成本分配標籤) 下，選擇 Activate (啟用)。

在您建立和啟用標籤後， AWS 會產生成本分配報告，其中包含依作用中標籤分組的用量和成本。成本分配報告包含每個帳單期間的所有 AWS 成本。該報告同時包含有標籤和沒標籤的資源，以便您可清楚地整理資源的費用。

> **Note**
>
> 目前，從 DynamoDB 傳出的任何資料都不會在成本分配報告上依標籤細分。

如需詳細資訊，請參閱使用成本分配標籤。

全域資料表：多重作用中的多區域複寫

Amazon DynamoDB 全域資料表是全受管、多區域及多重作用中的資料庫功能，為全域規模應用程式提供易於使用的資料複寫，以及快速的本地化讀寫效能。

全域資料表會自動跨 AWS 區域 帳戶和選擇性跨 AWS 帳戶複寫 DynamoDB 資料表資料，而不需要您建置和維護自己的複寫解決方案。全域資料表非常適合需要透過多區域部署達到業務連續性和高可用性的應用程式。任何全域資料表複本都可以提供讀取和寫入。如果應用程式處理在區域內中斷，應用程式可以將流量轉移到不同區域，實現低或零復原點目標 (RPO) 的高度恢復能力。全域資料表可在可用 DynamoDB 的所有區域中使用。

一致性模式

您可以在建立全域資料表時設定其一致性模式。全域資料表支援兩種一致性模式：多區域最終一致性 (MREC) 和多區域強式一致性 (MRSC)。

如果您在建立全域資料表時未指定一致性模式，全域資料表會預設為多區域最終一致性 (MREC)。全域資料表不能包含一致性模式設定不同的複本。您無法在建立全域資料表後變更其一致性模式。

帳戶組態

DynamoDB 現在支援兩種全域資料表模型，每個模型都針對不同的架構模式而設計：

- 相同帳戶全域資料表 – 所有複本都會在單一 AWS 帳戶中建立和管理。
- 多帳戶全域資料表 – 在參與共用複寫群組時，複寫會部署到多個 AWS 帳戶。

相同帳戶和多帳戶模型都支援多區域寫入、非同步複寫、last-writer-wins衝突解析，以及相同的計費模型。不過，它們在帳戶、許可、加密和資料表控管的管理方式上有所不同。

針對 MRSC 設定的全域資料表僅支援相同帳戶組態。

您可以使用 AWS 管理主控台設定全域資料表。全域資料表使用現有的 DynamoDB API 來讀取和寫入資料至資料表，因此不需要變更應用程式。您只需為佈建或使用的資源付費，無需前期成本或承諾。

相同帳戶和多帳戶全域資料表的比較

屬性	相同帳戶全域資料表	多帳戶全域資料表
主要使用案例	單一 AWS 帳戶中應用程式的多區域彈性	多區域、多帳戶複寫，適用於不同團隊擁有的應用程式、不同的業務單位或跨帳戶的強大安全界限
帳戶模型	在一個 AWS 帳戶中建立和管理的所有複本	在同一部署中跨多個 AWS 帳戶建立的複本
資源擁有權	單一帳戶擁有資料表和所有複本	每個帳戶擁有其本機複本；複寫群組橫跨 帳戶
支援的版本	全域資料表 2019.11.21 版（目前）和 2017.11.29 版（舊版）	全域資料表版本 2019.11.21（目前）
控制平面操作	透過資料表擁有者帳戶建立、修改和刪除複本	分散式控制平面操作：帳戶加入或離開複寫群組
資料平面操作	每個區域的標準 DynamoDB 端點	每個帳戶/區域的資料平面存取；透過複寫群組路由
安全界限	單一 IAM 和 KMS 界限	不同的 IAM、KMS、帳單、CloudTrail 和每個帳戶的控管
最適合	具有資料表集中擁有權的組織	具有聯合團隊、控管界限或多帳戶設定的組織

主題

- 全域資料表核心概念
- DynamoDB 相同帳戶全域資料表
- DynamoDB 多帳戶全域資料表
- 了解全域資料表的 Amazon DynamoDB 計費

- DynamoDB 全域資料表版本
- 全域資料表的最佳實務

全域資料表核心概念

下列各節說明 Amazon DynamoDB 中全域資料表的概念和行為

概念

全域資料表是一種 DynamoDB 功能，可跨 AWS 區域複寫資料表資料。

複本資料表 (或複本) 是一種 DynamoDB 資料表，也是全域資料表的一部分。全域資料表包含兩個或多個不同 AWS 區域的複本資料表。單一全域資料表的每個 AWS 區域只能有一個複本。全域資料表中的所有複本共用相同的資料表名稱、主索引鍵結構描述和項目資料。

當應用程式將資料寫入某個區域的複本時，DynamoDB 會自動將該寫入作業複寫到另一個全域資料表的所有其他複本。如需如何開始使用全域資料表的詳細資訊，請參閱 教學課程：建立全域資料表或 教學課程：建立多帳戶全域資料表。

版本

DynamoDB 全域資料表有兩種版本可用：2019.11.21 版全域資料表 (目前版本) 和 全域資料表 2017.11.29 版 (舊版)。您應該盡可能使用 Global Tables 2019.11.21 版 （目前）。本文件章節資訊適用於 2019.11.21 版 (目前版本)。如需詳細資訊，請參閱判斷全域資料表的版本判斷全域資料表版本。

可用性

全域資料表可讓您更輕鬆實作多區域的高可用性架構，進而提升業務持續性。如果單一區域中的工作負載 AWS 受損，您可以將應用程式流量轉移到不同的區域，並對相同全域資料表中的不同複本資料表執行讀取和寫入。

全域資料表中的每個複本資料表，其提供的耐用性和可用性與單一區域 DynamoDB 資料表相同。全域資料表提供 99.999% 的可用性服務水準協議 (SLA)；單一區域資料表則為 99.99%。

故障注入測試

MREC 和 MRSC 全域資料表都與 AWS Fault Injection Service (AWS FIS) 整合，這是一種全受管服務，用於執行受控故障注入實驗，以改善應用程式的彈性。使用 AWS FIS，您可以：

- 建立可定義特定失敗案例的實驗範本。

- 透過模擬區域隔離（即暫停複寫往返所選複本）驗證應用程式彈性失敗，以在某個 AWS 區域發生中斷時測試錯誤處理、復原機制和多區域流量轉移行為。

例如，在美國東部（維吉尼亞北部）、美國東部（俄亥俄）和美國西部（奧勒岡）具有複本的全域資料表中，您可以在美國東部（俄亥俄）執行實驗來測試區域隔離，而美國東部（維吉尼亞北部）和美國西部（奧勒岡）繼續正常操作。此項受控測試可協助您事先識別和解決潛在問題，以免影響正式作業工作負載。

如需 AWS FIS 支援動作的完整清單和跨區域連線，以暫停區域之間的 DynamoDB 複寫，請參閱 FIS 使用者指南中的動作目標。 AWS

如需 AWS FIS 中可用 Amazon DynamoDB 全域資料表動作的相關資訊，請參閱 AWS FIS 使用者指南中的 DynamoDB 全域資料表動作參考。

若要開始執行故障注入實驗，請參閱 AWS FIS 使用者指南中的規劃您的 FIS 實驗。 AWS

> **Note**
>
> 在 MRSC 中的 AWS FIS 實驗期間，允許最終一致讀取，但不允許資料表設定更新，例如變更帳單模式或設定資料表輸送量，類似於 MREC。請檢查 CloudWatch 指標`FaultInjectionServiceInducedErrors`，以取得有關錯誤碼的其他詳細資訊。

存留時間 (TTL)

針對已設定 MREC 的全域資料表，您可以設定存留時間 (TTL) 刪除。系統會自動同步全域資料表中所有複本的 TTL 設定。當 TTL 從區域中的複本刪除項目時，此刪除作業會複寫至全域資料表中的所有其他複本。TTL 不會消耗寫入容量，因此如果某個區域發生刪除作業，您不需要為該區域的 TTL 刪除付費。不過，若是在全域資料表中具有複本的區域間彼此複寫刪除，則需要支付費用。

針對要複寫刪除的複本，TTL 刪除複寫會耗用該複本的寫入容量。如果寫入輸送量和 TTL 刪除輸送量的組合高於佈建寫入容量，則已設定佈建容量的複本可能會對這些請求限流。

針對已設定多區域強一致性 (MRSC) 的全域資料表，您無法設定存留時間 (TTL) 刪除。

串流

如果全域資料表已設定多區域最終一致性 (MREC)，其會從複本資料表的 DynamoDB Stream 讀取變更，並將變更套用至所有其他複本資料表，藉此複寫這些變更。因此，MREC 全域資料表中的所有複本預設會啟用 Streams，您無法在這些複本上將其停用。MREC 複寫程序可能會將短時間內多個變更

合併為單一複寫寫入，導致每個複本的 Stream 包含略微不同的記錄。MREC 複本上的串流記錄會維持相同項目所有變更的順序，但不同項目變更的相對順序可能因複本而異。

如果您要撰寫應用程式來處理 Streams 記錄，藉此掌握全域資料表中特定區域而非其他區域發生的變更，您可以為每個項目新增一個屬性，以定義該項目發生變更所在的區域。您可以使用此屬性來篩選 Streams 記錄，以找出其他區域發生的變更，包括使用 Lambda 事件篩選條件，以僅調用 Lambda 函式找出特定區域的變更。

如果全域資料表已設定多區域強一致性 (MRSC)，其不會使用 DynamoDB Streams 進行複寫，因此 MRSC 複本預設不會啟用 Streams。您可以在 MRSC 複本上啟用 Streams。每個複本的 MRSC 複本 Stream 記錄都是相同的，包括 Stream 記錄排序。

交易

若是已設定 MREC 的全域資料表，DynamoDB 交易操作 (`TransactWriteItems` 和 `TransactGetItems`) 僅在調用操作的區域中為不可分割。交易寫入不會以一個單位形式跨區域複寫，亦即，在指定時間點，其他複本中的讀取操作可能只會傳回交易中的部分寫入。

例如，如果您有一個全域資料表，其在美國東部 (俄亥俄) 和美國西部 (奧勒岡) 區域有複本，並且在美國東部 (俄亥俄) 區域執行 `TransactWriteItems` 操作，在這些變更進行複寫之後，您可能會在美國西部 (奧勒岡) 區域看到部分完成的交易。只有當變更已在來源區域遞交的情況下，這些變更才會複寫至其他區域。

如果全域資料表已設定多區域強一致性 (MRSC)，則不支援交易操作，且在 MRSC 複本上調用這些操作時會傳回錯誤。

讀取和寫入輸送量

佈建模式

複寫會耗用寫入容量。如果應用程式寫入輸送量和複寫寫入輸送量的組合超過佈建寫入容量，針對佈建容量設定的複本可能會調節請求。對於使用佈建模式的全域資料表，複本之間會同步讀取和寫入容量的自動擴展設定。

您可以在複本層級使用 `ProvisionedThroughputOverride` 參數，獨立設定全域資料表中每個複本的讀取容量設定。根據預設，佈建讀取容量的變更會套用至全域資料表中的所有複本。將新複本新增至全域資料表時，除非明確指定複本層級覆寫，否則來源資料表或複本的讀取容量會用作初始值。

隨需模式

對於針對隨需模式設定的全域資料表，寫入容量會自動同步至所有複本。DynamoDB 會根據流量自動調整容量，而且沒有要管理的複本特定讀取或寫入容量設定。

監控全域資料表

如果全域資料表已設定多區域最終一致性 (MREC)，其會將 ReplicationLatency 指標發布至 CloudWatch。假設某個項目寫入複本資料表，此指標可追蹤該項目出現在全域資料表中其他複本時經過的時間。ReplicationLatency 會以毫秒表示，並會針對全域資料表中每對來源及目的地區域對發送。

典型ReplicationLatency值取決於所選 AWS 區域之間的距離，以及其他變數，例如工作負載類型和輸送量。例如，相較於非洲 (開普敦) (af-south-1) 區域，美國西部 (加利佛尼亞北部) (us-west-1) 區域的來源複本到美國西部 (奧勒岡) (us-west-2) 區域的 ReplicationLatency 較低。

ReplicationLatency 值上升可能表示，某個複本的更新未及時傳播到其他複本資料表。在這種情況下，您可以暫時將應用程式的讀取和寫入活動重新導向至不同的 AWS 區域。

如果全域資料表已設定多區域強一致性 (MRSC)，其不會發布 ReplicationLatency 指標。

管理全域資料表的考量

如果新複本建立未超過 24 小時，您就無法刪除用來新增全域資料表複本的資料表。

如果您停用包含全域資料表複本 AWS 的區域，這些複本會在區域停用後 20 小時永久轉換為單一區域資料表。

DynamoDB 相同帳戶全域資料表

相同帳戶全域資料表會自動在單一 AWS 帳戶中跨 AWS 區域複寫 DynamoDB 資料表資料。相同帳戶全域資料表提供執行多區域應用程式最簡單的模型，因為所有複本共用相同的帳戶界限、擁有權和許可模型。當您選擇複本資料表 AWS 的區域時，全域資料表會自動處理所有複寫。全域資料表可在可用 DynamoDB 的所有區域中使用。

相同帳戶全域資料表提供下列優點：

- 跨您選擇的 AWS 區域自動複寫 DynamoDB 資料表資料，以尋找更接近您使用者的資料
- 在區域隔離或降級期間實現更高的應用程式可用性
- 使用內建的衝突解決方案，讓您可以專注於應用程式的商業邏輯
- 建立相同帳戶全域資料表時，您可以選擇 多區域最終一致性 (MREC)或 多區域強一致性 (MRSC)

主題

- DynamoDB 全域資料表的運作方式

- 教學課程：建立全域資料表
- DynamoDB 全域資料表安全性

DynamoDB 全域資料表的運作方式

以下各節說明 Amazon DynamoDB 全域資料表的概念和行為。

概念

全域資料表是跨 AWS 區域複寫資料表資料的 DynamoDB 功能。

複本資料表 (或複本) 是一種 DynamoDB 資料表，也是全域資料表的一部分。全域資料表包含兩個或多個不同 AWS 區域的複本資料表。單一全域資料表的每個 AWS 區域只能有一個複本。全域資料表中的所有複本共用相同的資料表名稱、主索引鍵結構描述和項目資料。

當應用程式將資料寫入某個區域的複本時，DynamoDB 會自動將該寫入作業複寫到另一個全域資料表的所有其他複本。如需如何開始使用全域資料表的詳細資訊，請參閱 教學課程：建立全域資料表。

版本

DynamoDB 全域資料表有兩種版本可用：2019.11.21 版 (目前版本) 和 2017.11.29 版 (舊版)。您應該盡可能使用 2019.11.21 版 (目前版本)。本文件章節資訊適用於 2019.11.21 版 (目前版本)。如需詳細資訊，請參閱判斷全域資料表版本。

可用性

全域資料表可讓您更輕鬆實作多區域的高可用性架構，進而提升業務持續性。如果單一區域中的工作負載 AWS 受損，您可以將應用程式流量轉移到不同的區域，並對相同全域資料表中的不同複本資料表執行讀取和寫入。

全域資料表中的每個複本資料表，其提供的耐用性和可用性與單一區域 DynamoDB 資料表相同。全域資料表提供 99.999% 的可用性服務水準協議 (SLA)；單一區域資料表則為 99.99%。

一致性模式

您可以在建立全域資料表時設定其一致性模式。全域資料表支援兩種一致性模式：多區域最終一致性 (MREC) 和多區域強一致性 (MRSC)。

如果您在建立全域資料表時未指定一致性模式，全域資料表會預設為多區域最終一致性 (MREC)。全域資料表不能包含一致性模式設定不同的複本。您無法在建立全域資料表後變更其一致性模式。

多區域最終一致性 (MREC)

多區域最終一致性 (MREC) 是全域資料表的預設一致性模式。MREC 全域資料表複本中的項目變更通常會在一秒或更短時間內，以非同步方式複寫至所有其他複本。在極少的情況下，若 MREC 全域資料表中的複本受損或遭到隔離，任何尚未複寫到其他區域的資料，都會在複本正常運作時進行複寫。

如果同時在多個區域中修改相同項目，DynamoDB 會依據每個項目的最新內部時間戳記，使用修改來解決衝突，此方法稱為「最後一個寫入獲勝」衝突解決方法。所有複本的項目最終會收斂至上次寫入所建立的版本。

如果項目上次在發生讀取的區域中更新，則高度一致性讀取操作會傳回項目的最新版本，但如果項目上次在不同區域中更新，則可能會傳回過時的資料。條件式寫入會根據區域中項目的版本來評估條件運算式。

您可以將複本新增至現有 DynamoDB 資料表，來建立 MREC 全域資料表。新增複本不會影響現有單一區域 DynamoDB 資料表或全域資料表複本的效能。您可以將複本新增至 MREC 全域資料表，以擴展資料複寫的區域數量，或從 MREC 全域資料表中移除不再需要的複本。只要任何區域可以使用 DynamoDB，MREC 全域資料表皆可擁有這些區域的複本，且複本數量與 AWS 分割區中區域擁有的數量一樣多。

多區域強一致性 (MRSC)

您可以在建立全域資料表時設定多區域強一致性 (MRSC) 模式。在寫入操作傳回成功回應之前，MRSC 全域資料表複本中的項目變更會至少同步複寫到一個其他區域。任何 MRSC 複本上的高度一致性讀取操作，一律會傳回項目的最新版本。條件式寫入一律會根據項目的最新版本，來評估條件運算式。

MRSC 全域資料表必須剛好部署在三個區域中。您可將 MRSC 全域資料表設定為三個副本，或兩個副本搭配一個見證。見證是 MRSC 全域資料表的元件，其中包含寫入全域資料表複本的資料，並提供完整複本的選用替代方案，同時支援 MRSC 的可用性架構。您無法對見證執行讀取或寫入操作。見證所在的區域與兩個複本不同。建立 MRSC 全域資料表時，您可以在建立 MRSC 資料表時選擇複本和見證部署的區域。您可以透過 `DescribeTable` API 輸出，判斷 MRSC 全域資料表是否已設定見證，以及在哪個區域。見證由 DynamoDB 擁有和管理，且見證不會出現在您設定見證的區域中 AWS 的帳戶中。

下列區域集提供 MRSC 全域資料表：美國區域集（美國維吉尼亞北部、美國東部俄亥俄、美國西部奧勒岡）、歐洲區域集（歐洲愛爾蘭、歐洲倫敦、歐洲巴黎、歐洲法蘭克福）和 AP 區域集（亞太區域東京、亞太區域首爾和亞太區域大阪）。MRSC 全域資料表不能跨越區域集 (例如，MRSC 全域資料表不能同時包含來自美國和歐洲區域集的複本)。

若要建立 MRSC 全域資料表，您可以將一個複本和一個見證或兩個複本新增至未含資料的現有 DynamoDB 資料表。將現有的單一區域資料表轉換為 MRSC 全域資料表時，您必須確保資料表為空。不支援將單一區域資料表轉換為具現有項目的 MRSC 全域資料表。在轉換過程中，務必不要將任何資料寫入資料表。無法在現有 MRSC 全域資料表中新增額外副本。您無法從 MRSC 全域資料表刪除單一複本或見證。您可以從 MRSC 全域資料表刪除兩個複本，或刪除一個複本和見證，同時將剩餘的複本轉換為單一區域 DynamoDB 資料表。

當寫入操作嘗試修改已在其他區域中修改的項目時，該操作會失敗並顯示 `ReplicatedWriteConflictException`。因 而失敗的寫入`ReplicatedWriteConflictException`可以重試，如果項目不再在其他區域中修改，則寫入將會成功。

MRSC 全域資料表有下列考量：

- MRSC 全域資料表不支援存留時間 (TTL)。
- MRSC 全域資料表不支援區域次要索引 (LSI)。
- CloudWatch Contributor Insights 資訊僅會針對執行操作的區域回報。

選擇一致性模式

選擇多區域一致性模式時，主要關鍵在於應用程式優先考量較低的延遲寫入和高度一致性讀取，還是全域強一致性。

相較於 MRSC 全域資料表，MREC 全域資料表的寫入和高度一致性讀取延遲較低。MREC 全域資料表的復原點目標 (RPO) 等於複本之間的複寫延遲，通常為幾秒鐘，取決於複本區域。

在下列情況下，您應該使用 MREC 模式：

- 如果資料已在另一個區域中更新，應用程式可以容忍高度一致性讀取操作傳回過時資料。
- 比起多區域讀取一致性，您更重視較低的寫入和高度一致性讀取延遲。
- 您的多區域高可用性策略可以容忍大於 0 的 RPO。

相較於 MREC 全域資料表，MRSC 全域資料表的寫入和高度一致性讀取延遲較高。MRSC 全域資料表支援零的復原點目標 (RPO)。

在下列情況下，您應該使用 MRSC 模式：

- 您需要跨多個區域的高度一致性讀取。

- 比起較低的寫入延遲，您更重視全域讀取一致性。
- 您的多區域高可用性策略需要 RPO 為零。

監控全域資料表

如果全域資料表已設定多區域最終一致性 (MREC)，其會將 ReplicationLatency 指標發布至 CloudWatch。假設某個項目寫入複本資料表，此指標可追蹤該項目出現在全域資料表中其他複本時經過的時間。ReplicationLatency 會以毫秒表示，並會針對全域資料表中每對來源及目的地區域對發送。

典型ReplicationLatency值取決於所選 AWS 區域之間的距離，以及其他變數，例如工作負載類型和輸送量。例如，相較於非洲 (開普敦) (af-south-1) 區域，美國西部 (加利佛尼亞北部) (us-west-1) 區域的來源複本到美國西部 (奧勒岡) (us-west-2) 區域的 ReplicationLatency 較低。

ReplicationLatency 值上升可能表示，某個複本的更新未及時傳播到其他複本資料表。在這種情況下，您可以暫時將應用程式的讀取和寫入活動重新導向至不同的 AWS 區域。

如果全域資料表已設定多區域強一致性 (MRSC)，其不會發布 ReplicationLatency 指標。

故障注入測試

MREC 和 MRSC 全域資料表都與 AWS Fault Injection Service (AWS FIS) 整合，這是一種全受管服務，用於執行受控故障注入實驗，以改善應用程式的彈性。使用 AWS FIS，您可以：

- 建立實驗範本來定義特定的失敗案例。
- 透過模擬區域隔離（即暫停複寫往返所選複本）來注入失敗以驗證應用程式彈性，以在某個 AWS 區域發生中斷時測試錯誤處理、復原機制和多區域流量轉移行為。

例如，在美國東部（維吉尼亞北部）、美國東部（俄亥俄）和美國西部（奧勒岡）具有複本的全域資料表中，您可以在美國東部（俄亥俄）執行實驗來測試區域隔離，而美國東部（維吉尼亞北部）和美國西部（奧勒岡）繼續正常操作。此項受控測試可協助您事先識別和解決潛在問題，以免影響正式作業工作負載。

如需 AWS FIS 支援動作的完整清單和跨區域連線，以暫停區域之間的 DynamoDB 複寫，請參閱 FIS 使用者指南中的動作目標。 AWS

如需 AWS FIS 中可用 Amazon DynamoDB 全域資料表動作的相關資訊，請參閱 AWS FIS 使用者指南中的 DynamoDB 全域資料表動作參考。

若要開始執行故障注入實驗，請參閱 AWS FIS 使用者指南中的規劃您的 FIS 實驗。 AWS

> **Note**
>
> 在 MRSC 中的 AWS FIS 實驗期間，允許最終一致讀取，但不允許資料表設定更新，例如變更帳單模式或設定資料表輸送量，類似於 MREC。請檢查 CloudWatch 指標`FaultInjectionServiceInducedErrors`，以取得有關錯誤碼的其他詳細資訊。

存留時間 (TTL)

針對已設定 MREC 的全域資料表，您可以設定存留時間 (TTL) 刪除。系統會自動同步全域資料表中所有複本的 TTL 設定。當 TTL 從區域中的複本刪除項目時，此刪除作業會複寫至全域資料表中的所有其他複本。TTL 不會消耗寫入容量，因此如果某個區域發生刪除作業，您不需要為該區域的 TTL 刪除付費。不過，若是在全域資料表中具有複本的區域間彼此複寫刪除，則需要支付費用。

針對要複寫刪除的複本，TTL 刪除複寫會耗用該複本的寫入容量。如果寫入輸送量和 TTL 刪除輸送量的組合高於佈建寫入容量，則已設定佈建容量的複本可能會對這些請求限流。

針對已設定多區域強一致性 (MRSC) 的全域資料表，您無法設定存留時間 (TTL) 刪除。

串流

如果全域資料表已設定多區域最終一致性 (MREC)，其會從複本資料表的 DynamoDB Stream 讀取變更，並將變更套用至所有其他複本資料表，藉此複寫這些變更。因此，MREC 全域資料表中的所有複本預設會啟用 Streams，您無法在這些複本上將其停用。MREC 複寫程序可能會將短時間內多個變更合併為單一複寫寫入，導致每個複本的 Stream 包含略微不同的記錄。MREC 複本上的 Streams 記錄一律會依項目排序，但複本之間的項目順序可能不同。

如果全域資料表已設定多區域強一致性 (MRSC)，其不會使用 DynamoDB Streams 進行複寫，因此 MRSC 複本預設不會啟用 Streams。您可以在 MRSC 複本上啟用 Streams。每個複本的 MRSC 複本 Stream 記錄都是相同的，包括 Stream 記錄排序。

如果您要撰寫應用程式來處理 Streams 記錄，藉此掌握全域資料表中特定區域而非其他區域發生的變更，您可以為每個項目新增一個屬性，以定義該項目發生變更所在的區域。您可以使用此屬性來篩選 Streams 記錄，以找出其他區域發生的變更，包括使用 Lambda 事件篩選條件，以僅調用 Lambda 函式找出特定區域的變更。

交易

若是已設定 MREC 的全域資料表，DynamoDB 交易操作 (`TransactWriteItems` 和 `TransactGetItems`) 僅在調用操作的區域中為不可分割。交易寫入不會以一個單位形式跨區域複寫，亦即，在指定時間點，其他複本中的讀取操作可能只會傳回交易中的部分寫入。

例如，如果您有一個全域資料表，其在美國東部 (俄亥俄) 和美國西部 (奧勒岡) 區域有複本，並且在美國東部 (俄亥俄) 區域執行 `TransactWriteItems` 操作，在這些變更進行複寫之後，您可能會在美國西部 (奧勒岡) 區域看到部分完成的交易。只有當變更已在來源區域遞交的情況下，這些變更才會複寫至其他區域。

如果全域資料表已設定多區域強一致性 (MRSC)，則不支援交易操作，且在 MRSC 複本上調用這些操作時會傳回錯誤。

讀取和寫入輸送量

佈建模式

複寫會耗用寫入容量。如果應用程式寫入輸送量和複寫寫入輸送量的組合超過佈建寫入容量，針對佈建容量設定的複本可能會調節請求。對於使用佈建模式的全域資料表，複本之間會同步讀取和寫入容量的自動擴展設定。

您可以在複本層級使用 `ProvisionedThroughputOverride` 參數，獨立設定全域資料表中每個複本的讀取容量設定。根據預設，佈建讀取容量的變更會套用至全域資料表中的所有複本。將新複本新增至全域資料表時，除非明確指定複本層級覆寫，否則來源資料表或複本的讀取容量會用作初始值。

隨需模式

對於針對隨需模式設定的全域資料表，寫入容量會自動同步至所有複本。DynamoDB 會根據流量自動調整容量，而且沒有要管理的複本特定讀取或寫入容量設定。

設定同步

DynamoDB 全域資料表中的設定是組態參數，可控制各方面的資料表行為和複寫。您可以透過 DynamoDB 控制平面 API 管理這些設定，並在建立或修改全域資料表時加以設定。全域資料表會自動同步所有複本的特定設定，以維持一致性，同時確保區域特定最佳化的彈性。建議您了解哪些設定會同步及其行為方式，以有效設定全域資料表。依各複本之間的設定同步方式來看，設定分為三個主要類別。

下列設定一律會在全域資料表的各複本之間同步：

- 容量模式 (佈建容量或隨需)
- 資料表的佈建寫入容量
- 資料表的寫入自動擴展
- 金鑰結構描述的屬性定義
- 全域次要索引 (GSI) 定義

- GSI 的佈建寫入容量
- GSI 的寫入自動擴展
- 伺服器端加密 (SSE) 類型
- MREC 模式中的 Streams 定義
- 存留時間 (TTL)
- 暖輸送量
- 隨需最大寫入輸送量

下列設定會在各複本之間同步，但每個複本都可以覆寫：

- 資料表的佈建讀取容量
- 資料表的讀取自動擴展
- GSI 的佈建讀取容量
- GSI 的讀取自動擴展
- 資料表類別
- 隨需最大讀取輸送量

> **Note**
>
> 如果任何其他複本上的設定有所修改，則會變更可覆寫的設定值。例如，您有一個 MREC 全域資料表，其中包含美國東部 (維吉尼亞北部) 和美國西部 (奧勒岡)的複本。美國東部 (維吉尼亞北部) 複本已設定 200 RCUs 的佈建讀取輸送量。美國西部 (奧勒岡) 的複本已設定 100 RCUs 的佈建讀取輸送量覆寫。如果您將美國東部 (維吉尼亞北部) 複本上的佈建讀取輸送量設定從 200 RCUs 更新為 300 RCUs，則美國西部 (奧勒岡) 複本也會套用新的佈建讀取輸送量值。如此一來，美國西部 (奧勒岡) 複本的佈建讀取輸送量設定，會從 100 RCUs 的覆寫值變更為 300 RCUs 的新值。

各複本之間絕不會同步下列設定：

- 刪除保護
- 時間點復原
- Tags (標籤)
- 資料表的 CloudWatch Contributor Insights 啟用

- GSI 的 CloudWatch Contributor Insights 啟用
- Kinesis Data Streams 定義
- 資源政策
- MRSC 模式中的 Streams 定義

各複本之間的所有其他設定均不會同步。

DynamoDB Accelerator (DAX)

全域資料表複本的寫入作業會繞過 DynamoDB Accelerator (DAX)，直接更新 DynamoDB。因此，DAX 快取可能會過時，因為寫入不會更新 DAX 快取。只有在快取 TTL 過期時，系統才會重新整理全域資料表複本設定的 DAX 快取。

管理全域資料表的考量

如果新複本建立未超過 24 小時，您就無法刪除用來新增全域資料表複本的資料表。

如果您停用包含全域資料表複本 AWS 的區域，這些複本會在區域停用後 20 小時永久轉換為單一區域資料表。

教學課程：建立全域資料表

本節提供逐步說明，協助您建立 DynamoDB 全域資料表，並設定為您偏好的一致性模式。根據您的應用程式需求，選擇多區域最終一致性 (MREC) 或多區域強一致性 (MRSC) 模式。

MREC 全域資料表提供較低的寫入延遲，以及跨 AWS 區域的最終一致性。MRSC 全域資料表跨區域提供高度一致性讀取，寫入延遲略高於 MREC。請選擇最符合您應用程式對資料一致性、延遲和可用性需求的一致性模式。

主題

- 建立為 MREC 設定的全域資料表
- 建立為 MRSC 設定的全域資料表

建立為 MREC 設定的全域資料表

本節說明如何建立具有多區域最終一致性 (MREC) 模式的全域資料表。MREC 是全域資料表的預設一致性模式，跨 AWS 區域提供具有非同步複寫的低延遲寫入。對一個區域中的項目所做的變更，通常會在一秒內複寫到所有其他區域。這使得 MREC 非常適合以低寫入延遲為優先，並且可以容忍不同區域在短期內可能傳回略有不同版本資料的應用程式。

您可以在可使用 DynamoDB 的任何 AWS 區域中，使用複本建立 MREC 全域資料表，並隨時新增或移除複本。下列範例示範如何在多個區域中以複本建立 MREC 全域資料表。

使用 DynamoDB 主控台建立 MREC 全域資料表

請遵循下列步驟，使用 AWS 管理主控台建立全域資料表。下列範例使用美國及歐洲的複本資料表建立全域資料表。

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在此範例中，請從導覽列的區域選擇器選擇美國東部 (俄亥俄)。
3. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
4. 選擇 Create Table (建立資料表)。
5. 在建立資料表頁面上：

   a. 對於 Table name (資料表名稱)，請輸入 **`Music`**。
   b. 對於 Partition key (分割區索引鍵)，請輸入 **`Artist`**。
   c. 對於排序索引鍵，請輸入 **`SongTitle`**。
   d. 保留其他預設設定，並選擇建立資料表。

      此新資料表會做為新全域資料表中的第一個複本資料表。這是您稍後新增其他複本資料表的原型。
6. 在資料表變成作用中之後：

   a. 從資料表清單選擇 Music 資料表。
   b. 選擇 Global Tables (全域資料表) 標籤。
   c. 選擇 Create replica (建立複本)。
7. 從可用的複寫區域下拉式清單選擇美國西部 (奧勒岡) us-west-2。

   主控台會確認選取區域中沒有同名的資料表。若有同名的資料表，您必須先刪除現有的資料表，才可在該區域中建立新的複本資料表。
8. 選擇 Create replica (建立複本)。這會啟動在美國西部 (奧勒岡) us-west-2 區域建立資料表的流程。

   Music 資料表 (以及任何其他複寫資料表) 的全域資料表標籤顯示資料表已於多個區域複寫。
9. 重複上述步驟來新增另一個區域，但選擇歐洲 (法蘭克福) eu-central-1 作為區域。
10. 若要測試複寫：

a. 請務必 AWS 管理主控台 在美國東部 （俄亥俄） 區域使用 。
b. 選擇探索資料表項目。
c. 選擇 Create item (建立項目)。
d. 為 Artist 輸入 **`item_1`**，為 SongTitle 輸入 **`Song Value 1`**。
e. 選擇 Create item (建立項目)。

11. 切換到其他區域來驗證複寫：

a. 從右上角的區域選擇器選擇歐洲 (法蘭克福)。
b. 確認 Music 資料表包含您建立的項目。
c. 對美國西部 (奧勒岡) 重複驗證。

使用 AWS CLI 或 Java 建立 MREC 全域資料表

CLI

下列程式碼範例示範如何使用具有最終一致性 (MREC) 的多區域複寫，來管理 DynamoDB 全域資料表。

- 建立具有多區域複寫 (MREC) 的資料表。
- 將項目放置於複本資料表，並從其中取得項目。
- 逐一移除複本。
- 藉由刪除資料表進行清除。

AWS CLI 使用 Bash 指令碼

利用多區域複寫來建立資料表。

```
# Step 1: Create a new table (MusicTable) in US East (Ohio), with DynamoDB
 Streams enabled (NEW_AND_OLD_IMAGES)
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
```

```
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2

# Step 2: Create an identical MusicTable table in US East (N. Virginia)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2

# Step 3: Create a table in Europe (Ireland)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2
```

描述多區域資料表。

```
# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

將項目放置於複本資料表中。

```
# Step 5: To verify that replication is working, add a new item to the Music
 table in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

從複本資料表取得項目。

```
# Step 6: Wait for a few seconds, and then check to see whether the item has
 been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

移除複本。

```
# Step 7: Delete the replica table in Europe (Ireland) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2

# Delete the replica table in US East (N. Virginia) Region
```

```
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2
```

透過刪除資料表來進行清除作業。

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateTable

Java

下列程式碼範例示範如何跨多個區域建立和管理具有複本的 DynamoDB 全域資料表。

- 建立具備全域次要索引和 DynamoDB Streams 的資料表。
- 在不同的區域中新增複本以建立全域資料表。
- 從全域資料表移除複本。
- 新增測試項目以驗證跨區域的複寫。

- 描述全域資料表組態和複本狀態。

適用於 Java 2.x 的 SDK

使用 建立具有全域次要索引和 DynamoDB Streams 的資料表 AWS SDK for Java 2.x。

```
   public static CreateTableResponse createTableWithGSI(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
indexName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating table: " + tableName + " with GSI: " +
indexName);

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
                       .attributeType(ScalarAttributeType.S)
                       .build(),
                   AttributeDefinition.builder()
                       .attributeName("SongTitle")
                       .attributeType(ScalarAttributeType.S)
                       .build())
               .keySchema(
                   KeySchemaElement.builder()
                       .attributeName("Artist")
                       .keyType(KeyType.HASH)
                       .build(),
                   KeySchemaElement.builder()
```

```
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .globalSecondaryIndexes(GlobalSecondaryIndex.builder()
                    .indexName(indexName)
                    .keySchema(KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.HASH)
                        .build())
                    .projection(

Projection.builder().projectionType(ProjectionType.ALL).build())
                    .build())
                .streamSpecification(StreamSpecification.builder()
                    .streamEnabled(true)
                    .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
                    .build())
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

等待資料表使用 變成作用中 AWS SDK for Java 2.x。

```
    public static void waitForTableActive(final DynamoDbClient dynamoDbClient,
final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
```

```
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Waiting for table to become active: " + tableName);

            try (DynamoDbWaiter waiter =
                DynamoDbWaiter.builder().client(dynamoDbClient).build()) {
                DescribeTableRequest request =
                    DescribeTableRequest.builder().tableName(tableName).build();

                waiter.waitUntilTableExists(request);
                LOGGER.info("Table is now active: " + tableName);
            }

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to wait for table to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

新增複本以使用 建立或擴展全域資料表 AWS SDK for Java 2.x。

```
    public static UpdateTableResponse addReplica(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final String indexName,
        final Long readCapacity) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
```

```
        }
        if (indexName == null || indexName.trim().isEmpty()) {
            throw new IllegalArgumentException("Index name cannot be null or
empty");
        }
        if (readCapacity == null || readCapacity <= 0) {
            throw new IllegalArgumentException("Read capacity must be a positive
number");
        }

        try {
            LOGGER.info("Adding replica in region: " + replicaRegion.id() + " for
table: " + tableName);

            // Create a ReplicationGroupUpdate for adding a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .create(builder -> builder.regionName(replicaRegion.id())
                    .globalSecondaryIndexes(ReplicaGlobalSecondaryIndex.builder()
                        .indexName(indexName)

.provisionedThroughputOverride(ProvisionedThroughputOverride.builder()
                            .readCapacityUnits(readCapacity)
                            .build())
                        .build())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica addition initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
```

```
        }
    }
```

使用 從全域資料表移除複本 AWS SDK for Java 2.x。

```
    public static UpdateTableResponse removeReplica(
        final DynamoDbClient dynamoDbClient, final String tableName, final Region
replicaRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }

        try {
            LOGGER.info("Removing replica in region: " + replicaRegion.id() + "
for table: " + tableName);

            // Create a ReplicationGroupUpdate for removing a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .delete(builder ->
builder.regionName(replicaRegion.id()).build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica removal initiated in region: " +
replicaRegion.id());

            return response;
```

```
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to remove replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

新增測試項目以使用 驗證複寫 AWS SDK for Java 2.x。

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to table: " + tableName);

            Map<String,
software.amazon.awssdk.services.dynamodb.model.AttributeValue> item = new
HashMap<>();
            item.put(
                "Artist",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(artist)
                    .build());
            item.put(
```

```
                "SongTitle",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(songTitle)
                    .build());

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw e;
        }
    }
```

使用 描述全域資料表組態和複本 AWS SDK for Java 2.x。

```
    public static DescribeTableResponse describeTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);
```

```
            LOGGER.info("Table status: " + response.table().tableStatus());
            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

使用 完成全域資料表操作的範例 AWS SDK for Java 2.x。

```
    public static void exampleUsage(final Region sourceRegion, final Region
replicaRegion) {

        String tableName = "Music";
        String indexName = "SongTitleIndex";
        Long readCapacity = 15L;

        // Create DynamoDB client for the source region
        try (DynamoDbClient dynamoDbClient =
            DynamoDbClient.builder().region(sourceRegion).build()) {

            try {
                // Step 1: Create the initial table with GSI and streams
```

```
                LOGGER.info("Step 1: Creating table in source region: " +
sourceRegion.id());
                createTableWithGSI(dynamoDbClient, tableName, indexName);

                // Step 2: Wait for table to become active
                LOGGER.info("Step 2: Waiting for table to become active");
                waitForTableActive(dynamoDbClient, tableName);

                // Step 3: Add replica in destination region
                LOGGER.info("Step 3: Adding replica in region: " +
replicaRegion.id());
                addReplica(dynamoDbClient, tableName, replicaRegion, indexName,
readCapacity);

                // Step 4: Wait a moment for replica creation to start
                Thread.sleep(5000);

                // Step 5: Describe table to view replica information
                LOGGER.info("Step 5: Describing table to view replicas");
                describeTable(dynamoDbClient, tableName);

                // Step 6: Add a test item to verify replication
                LOGGER.info("Step 6: Adding test item to verify replication");
                putTestItem(dynamoDbClient, tableName, "TestArtist", "TestSong");

                LOGGER.info("Global table setup completed successfully!");
                LOGGER.info("You can verify replication by checking the item in
region: " + replicaRegion.id());

                // Step 7: Remove replica and clean up table
                LOGGER.info("Step 7: Removing replica from region: " +
replicaRegion.id());
                removeReplica(dynamoDbClient, tableName, replicaRegion);
                DeleteTableResponse deleteTableResponse =
dynamoDbClient.deleteTable(
                    DeleteTableRequest.builder().tableName(tableName).build());
                LOGGER.info("MREC global table demonstration completed
successfully!");

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Thread was interrupted", e);
            } catch (DynamoDbException e) {
                LOGGER.severe("DynamoDB operation failed: " + e.getMessage());
```

```
                throw e;
            }
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - CreateTable
  - DescribeTable
  - PutItem
  - UpdateTable

建立為 MRSC 設定的全域資料表

本節說明如何建立多區域強一致性 (MRSC) 全域資料表。MRSC 全域資料表會跨區域同步複寫項目變更，確保任何複本上的高度一致性讀取操作一律會傳回最新版本的項目。將單一區域資料表轉換為 MRSC 全域資料表時，您必須確保資料表為空白。不支援將單一區域資料表轉換為具現有項目的 MRSC 全域資料表。在轉換過程中，務必不要將任何資料寫入資料表。

您可將 MRSC 全域資料表設定為三個複本，或兩個複本搭配一個見證。建立 MRSC 全域資料表時，您可以選擇部署複本和選用見證的區域。下列範例會在美國東部 (維吉尼亞北部) 和美國東部 (俄亥俄) 區域中建立具有複本的 MRSC 全域資料表，並在美國西部 (奧勒岡) 區域中建立見證。

> **Note**
>
> 建立全域資料表之前，請驗證 Service Quotas 輸送量限制在所有目標區域中都一致，因為這是建立全域資料表的必要條件。如需全域資料表輸送量限制的詳細資訊，請參閱全域資料表配額。

使用 DynamoDB 主控台建立 MRSC 全域資料表

請遵循下列步驟，使用 AWS 管理主控台建立 MRSC 全域資料表。

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 從導覽列的區域選擇器選擇支援 MRSC 全域資料表的區域，例如 **us-east-2**。
3. 在導覽窗格中，選擇 Tables (資料表)。

4. 選擇 Create Table (建立資料表)。
5. 在建立資料表頁面上：

   a. 對於 Table name (資料表名稱)，請輸入 **Music**。
   b. 對於分割區索引鍵，請輸入 **Artist** 並保持預設的字串類型。
   c. 對於排序索引鍵，請輸入 **SongTitle** 並保持預設的字串類型。
   d. 保留其他預設設定，並選擇建立資料表。

      此新資料表會做為新全域資料表中的第一個複本資料表。這是您稍後新增其他複本資料表的原型。
6. 等待資料表變成作用中，然後從資料表清單中選取。
7. 選擇全域資料表標籤，然後選擇建立複本。
8. 在建立複本頁面上：

   a. 在多區域一致性之下選擇強一致性。
   b. 針對複寫區域 1 選擇 **US East (N. Virginia) us-east-1**。
   c. 針對複寫區域 2 選擇 **US West (Oregon) us-west-2**。
   d. 檢查美國西部 (奧勒岡) 區域的設定為見證。
   e. 選擇建立複本。
9. 等待複本和見證建立流程完成。資料表可供使用時，複本狀態會顯示為作用中。

使用 AWS CLI 或 Java 建立 MRSC 全域資料表

請在開始前確定您的 IAM 主體具有所需的許可權限，以建立含見證區域的 MRSC 全域資料表。

下列範例 IAM 政策可讓您在美國東部 (俄亥俄) 建立 DynamoDB 資料表 (`MusicTable`)，其中複本位於美國東部 (維吉尼亞北部)，見證區域則在美國西部 (奧勒岡)：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
```

```
                "dynamodb:CreateTableReplica",
                "dynamodb:CreateGlobalTableWitness",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable",
                "dynamodb:DeleteTable",
                "dynamodb:DeleteTableReplica",
                "dynamodb:DeleteGlobalTableWitness",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/MusicTable",
                "arn:aws:dynamodb:us-east-2:123456789012:table/MusicTable",
                "arn:aws:dynamodb:us-west-2:123456789012:table/MusicTable"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "replication.dynamodb.amazonaws.com"
                }
            }
        }
    ]
}
```

下列程式碼範例示範如何建立和管理具有多區域強一致性 (MRSC) 的 DynamoDB 全域資料表。

- 建立具有多區域強一致性的資料表。
- 驗證 MRSC 組態和複本狀態。
- 透過即時讀取測試跨區域的強一致性。
- 透過 MRSC 保證執行條件式寫入。

- 清除 MRSC 全域資料表資源。

Bash

AWS CLI 使用 Bash 指令碼

建立具有多區域強一致性的資料表。

```
# Step 1: Create a new table in us-east-2 (primary region for MRSC)
# Note: Table must be empty when enabling MRSC
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --region us-east-2

# Wait for table to become active
aws dynamodb wait table-exists --table-name MusicTable --region us-east-2

# Step 2: Add replica and witness with Multi-Region Strong Consistency
# MRSC requires exactly three replicas in supported regions
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Create": {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

驗證 MRSC 組態和複本狀態。

```
# Verify the global table configuration and MRSC setting
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
```

```
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

透過跨區域即時讀取測試強一致性。

```
# Write an item to the primary region
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"},"Album":
 {"S":"The Beatles 1967-1970"},"Year": {"N":"1968"}}' \
    --region us-east-2

# Read the item from replica region to verify strong consistency (cannot read or
 write to witness)
# No wait time needed - MRSC provides immediate consistency
echo "Reading from us-east-1 (immediate consistency):"
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --consistent-read \
    --region us-east-1
```

透過 MRSC 保證執行條件式寫入。

```
# Perform a conditional update from a different region
# This demonstrates that conditions work consistently across all regions
aws dynamodb update-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --update-expression "SET #rating = :rating" \
    --condition-expression "attribute_exists(Artist)" \
    --expression-attribute-names '{"#rating": "Rating"}' \
    --expression-attribute-values '{":rating": {"N":"5"}}' \
    --region us-east-1
```

清除 MRSC 全域資料表資源。

```
# Remove replica tables (must be done before deleting the primary table)
```

```
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Delete": {"RegionName": "us-west-2"}}]' \
    --region us-east-2

# Wait for replicas to be deleted
echo "Waiting for replicas to be deleted..."
sleep 30

# Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Java

適用於 Java 2.x 的 SDK

使用 建立準備好進行 MRSC 轉換的區域資料表 AWS SDK for Java 2.x。

```
   public static CreateTableResponse createRegionalTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
```

```
        }

        try {
            LOGGER.info("Creating regional table: " + tableName + " (must be
empty for MRSC)");

            CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
                .attributeDefinitions(
                    AttributeDefinition.builder()
                        .attributeName("Artist")
                        .attributeType(ScalarAttributeType.S)
                        .build(),
                    AttributeDefinition.builder()
                        .attributeName("SongTitle")
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Regional table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create regional table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to create regional table: " + tableName)
                .cause(e)
                .build();
        }
```

```
    }
```

使用 將區域資料表轉換為具有複本和見證的 MRSC AWS SDK for Java 2.x。

```
    public static UpdateTableResponse convertToMRSCWithWitness(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Converting table to MRSC with replica in " +
replicaRegion.id() + " and witness in "
                + witnessRegion.id());

            // Create replica update using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .create(CreateReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Create witness update
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .create(CreateGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
```

```
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .multiRegionConsistency(MultiRegionConsistency.STRONG)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC conversion initiated. Status: "
                + response.tableDescription().tableStatus());
            LOGGER.info("UpdateTableResponse full object: " + response);
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to convert table to MRSC: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to convert table to MRSC: " + tableName)
                .cause(e)
                .build();
        }
    }
```

使用 描述 MRSC 全域資料表組態 AWS SDK for Java 2.x。

```
    public static DescribeTableResponse describeMRSCTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing MRSC global table: " + tableName);
```

```
            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);

            LOGGER.info("Table status: " + response.table().tableStatus());
            LOGGER.info("Multi-region consistency: " +
response.table().multiRegionConsistency());

            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            if (response.table().globalTableWitnesses() != null
                && !response.table().globalTableWitnesses().isEmpty()) {
                LOGGER.info("Number of witnesses: "
                    + response.table().globalTableWitnesses().size());
                response.table()
                    .globalTableWitnesses()
                    .forEach(witness -> LOGGER.info(
                        "Witness region: " + witness.regionName() + ", Status: "
+ witness.witnessStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Table not found: " + tableName)
                .cause(e)
                .build();
        } catch (DynamoDbException e) {
```

```
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to describe table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

新增測試項目，以使用 AWS SDK for Java 2.x驗證 MRSC 強式一致性。

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String album,
        final String year) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to MRSC global table: " + tableName);

            Map<String, AttributeValue> item = new HashMap<>();
            item.put("Artist", AttributeValue.builder().s(artist).build());
            item.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            if (album != null && !album.trim().isEmpty()) {
```

```
                item.put("Album", AttributeValue.builder().s(album).build());
            }
            if (year != null && !year.trim().isEmpty()) {
                item.put("Year", AttributeValue.builder().n(year).build());
            }

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully with strong consistency");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to add test item to table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

使用 從 MRSC 複本讀取具有一致讀取的項目 AWS SDK for Java 2.x。

```
    public static GetItemResponse getItemWithConsistentRead(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Reading item from MRSC global table with consistent
read: " + tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            GetItemRequest getItemRequest = GetItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .consistentRead(true)
                .build();

            GetItemResponse response = dynamoDbClient.getItem(getItemRequest);

            if (response.hasItem()) {
                LOGGER.info("Item found with strong consistency - no wait time
needed");
            } else {
                LOGGER.info("Item not found");
            }

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to read item from table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to read item from table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

使用 搭配 MRSC 保證執行條件式更新 AWS SDK for Java 2.x。

```
    public static UpdateItemResponse performConditionalUpdate(
```

```
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String rating) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }
        if (rating == null || rating.trim().isEmpty()) {
            throw new IllegalArgumentException("Rating cannot be null or empty");
        }

        try {
            LOGGER.info("Performing conditional update on MRSC global table: " +
tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            Map<String, String> expressionAttributeNames = new HashMap<>();
            expressionAttributeNames.put("#rating", "Rating");

            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
            expressionAttributeValues.put(
                ":rating", AttributeValue.builder().n(rating).build());

            UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #rating = :rating")
```

```
                .conditionExpression("attribute_exists(Artist)")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

            UpdateItemResponse response =
dynamoDbClient.updateItem(updateItemRequest);
            LOGGER.info("Conditional update successful - demonstrates strong
consistency");

            return response;

        } catch (ConditionalCheckFailedException e) {
            LOGGER.warning("Conditional check failed: " + e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to perform conditional update: " + tableName +
" - " + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to perform conditional update: " + tableName)
                .cause(e)
                .build();
        }
    }
```

等待 MRSC 複本和見證使用 變成作用中 AWS SDK for Java 2.x。

```
    public static void waitForMRSCReplicasActive(
        final DynamoDbClient dynamoDbClient, final String tableName, final int
maxWaitTimeSeconds)
        throws InterruptedException {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (maxWaitTimeSeconds <= 0) {
            throw new IllegalArgumentException("Max wait time must be positive");
        }
```

```
        try {
            LOGGER.info("Waiting for MRSC replicas and witnesses to become
active: " + tableName);

            final long startTime = System.currentTimeMillis();
            final long maxWaitTimeMillis = maxWaitTimeSeconds * 1000L;
            int backoffSeconds = 5; // Start with 5 second intervals
            final int maxBackoffSeconds = 30; // Cap at 30 seconds

            while (System.currentTimeMillis() - startTime < maxWaitTimeMillis) {
                DescribeTableResponse response =
describeMRSCTable(dynamoDbClient, tableName);

                boolean allActive = true;
                StringBuilder statusReport = new StringBuilder();

                if (response.table().multiRegionConsistency() == null
                    || !MultiRegionConsistency.STRONG
                        .toString()

.equals(response.table().multiRegionConsistency().toString())) {
                    allActive = false;
                    statusReport
                        .append("MultiRegionConsistency: ")
                        .append(response.table().multiRegionConsistency())
                        .append(" ");
                }
                if (response.table().replicas() == null
                    || response.table().replicas().isEmpty()) {
                    allActive = false;
                    statusReport.append("No replicas found. ");
                }
                if (response.table().globalTableWitnesses() == null
                    || response.table().globalTableWitnesses().isEmpty()) {
                    allActive = false;
                    statusReport.append("No witnesses found. ");
                }

                // Check table status
                if (!"ACTIVE".equals(response.table().tableStatus().toString()))
{
                    allActive = false;
                    statusReport
```

```
                        .append("Table: ")
                        .append(response.table().tableStatus())
                        .append(" ");
                }

                // Check replica status
                if (response.table().replicas() != null) {
                    for (var replica : response.table().replicas()) {
                        if (!"ACTIVE".equals(replica.replicaStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Replica(")
                                .append(replica.regionName())
                                .append("): ")
                                .append(replica.replicaStatus())
                                .append(" ");
                        }
                    }
                }

                // Check witness status
                if (response.table().globalTableWitnesses() != null) {
                    for (var witness : response.table().globalTableWitnesses()) {
                        if (!"ACTIVE".equals(witness.witnessStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Witness(")
                                .append(witness.regionName())
                                .append("): ")
                                .append(witness.witnessStatus())
                                .append(" ");
                        }
                    }
                }

                if (allActive) {
                    LOGGER.info("All MRSC replicas and witnesses are now active:
" + tableName);
                    return;
                }
```

```
                LOGGER.info("Waiting for MRSC components to become active.
Status: " + statusReport.toString());
                LOGGER.info("Next check in " + backoffSeconds + " seconds...");

                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, maxBackoffSeconds);
            }

            throw DynamoDbException.builder()
                .message("Timeout waiting for MRSC replicas to become active
after " + maxWaitTimeSeconds + " seconds")
                .build();

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("Failed to wait for MRSC replicas to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

使用 清除 MRSC 複本和見證 AWS SDK for Java 2.x。

```
    public static UpdateTableResponse cleanupMRSCReplicas(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
```

```
        }

        try {
            LOGGER.info("Cleaning up MRSC replicas and witnesses for table: " +
tableName);

            // Remove replica using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .delete(DeleteReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Remove witness
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .delete(DeleteGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC cleanup initiated - removing replica and witness.
Response: " + response);

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to cleanup MRSC replicas: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to cleanup MRSC replicas: " + tableName)
                .cause(e)
                .build();
        }
    }
```

使用 完成 MRSC 工作流程示範 AWS SDK for Java 2.x。

```
   public static void demonstrateCompleteMRSCWorkflow(
       final DynamoDbClient primaryClient,
       final DynamoDbClient replicaClient,
       final String tableName,
       final Region replicaRegion,
       final Region witnessRegion)
       throws InterruptedException {

       if (primaryClient == null) {
           throw new IllegalArgumentException("Primary DynamoDB client cannot be
null");
       }
       if (replicaClient == null) {
           throw new IllegalArgumentException("Replica DynamoDB client cannot be
null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (replicaRegion == null) {
           throw new IllegalArgumentException("Replica region cannot be null");
       }
       if (witnessRegion == null) {
           throw new IllegalArgumentException("Witness region cannot be null");
       }

       try {
           LOGGER.info("=== Starting Complete MRSC Workflow Demonstration ===");

           // Step 1: Create an empty single-Region table
           LOGGER.info("Step 1: Creating empty single-Region table");
           createRegionalTable(primaryClient, tableName);

           // Use the existing GlobalTableOperations method for basic table
waiting
           LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
           GlobalTableOperations.waitForTableActive(primaryClient, tableName);
```

```
            // Step 2: Convert to MRSC with replica and witness
            LOGGER.info("Step 2: Converting to MRSC with replica and witness");
            convertToMRSCWithWitness(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for MRSC conversion to complete using MRSC-specific waiter
            LOGGER.info("Waiting for MRSC conversion to complete...");
            waitForMRSCReplicasActive(primaryClient, tableName);

            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 3: Verify MRSC configuration
            LOGGER.info("Step 3: Verifying MRSC configuration");
            describeMRSCTable(primaryClient, tableName);

            // Step 4: Test strong consistency with data operations
            LOGGER.info("Step 4: Testing strong consistency with data
operations");

            // Add test item to primary region
            putTestItem(primaryClient, tableName, "The Beatles", "Hey Jude", "The
Beatles 1967-1970", "1968");

            // Immediately read from replica region (no wait needed with MRSC)
            LOGGER.info("Reading from replica region immediately (strong
consistency):");
            GetItemResponse getResponse =
                getItemWithConsistentRead(replicaClient, tableName, "The
Beatles", "Hey Jude");

            if (getResponse.hasItem()) {
                LOGGER.info("# Strong consistency verified - item immediately
available in replica region");
            } else {
                LOGGER.warning("# Item not found in replica region");
            }

            // Test conditional update from replica region
            LOGGER.info("Testing conditional update from replica region:");
            performConditionalUpdate(replicaClient, tableName, "The Beatles",
"Hey Jude", "5");
```

```
            LOGGER.info("# Conditional update successful - demonstrates strong
consistency");

            // Step 5: Cleanup
            LOGGER.info("Step 5: Cleaning up resources");
            cleanupMRSCReplicas(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for cleanup to complete using basic table waiter
            LOGGER.info("Waiting for replica cleanup to complete...");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // "Halt" until replica/witness cleanup is complete
            DescribeTableResponse cleanupVerification =
describeMRSCTable(primaryClient, tableName);
            int backoffSeconds = 5; // Start with 5 second intervals
            while (cleanupVerification.table().multiRegionConsistency() != null)
{
                LOGGER.info("Waiting additional time (" + backoffSeconds + "
seconds) for MRSC cleanup to complete...");
                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, 30);
                cleanupVerification = describeMRSCTable(primaryClient,
tableName);
            }

            // Delete the primary table
            deleteTable(primaryClient, tableName);

            LOGGER.info("=== MRSC Workflow Demonstration Complete ===");
            LOGGER.info("");
            LOGGER.info("Key benefits of Multi-Region Strong Consistency
(MRSC):");
            LOGGER.info("- Immediate consistency across all regions (no eventual
consistency delays)");
            LOGGER.info("- Simplified application logic (no need to handle
eventual consistency)");
            LOGGER.info("- Support for conditional writes and transactions across
regions");
            LOGGER.info("- Consistent read operations from any region without
waiting");
```

```
        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("MRSC workflow failed: " + e.getMessage());
            throw e;
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

DynamoDB 全域資料表安全性

全域資料表複本是 DynamoDB 資料表，因此您可以使用與單一區域資料表相同的方法來控制對複本的存取，包括 AWS Identity and Access Management (IAM) 身分政策和以資源為基礎的政策。

本主題說明如何使用 IAM 許可和 AWS Key Management Service (AWS KMS) 加密保護 DynamoDB 全域資料表。您會了解允許跨區域複寫和自動擴展的服務連結角色 (SLR)、建立、更新和刪除全域資料表 所需的 IAM 許可，以及多區域最終一致性 (MREC) 和多區域強式一致性 (MRSC) 資料表之間的差異。您也會了解 AWS KMS 加密金鑰，以安全地管理跨區域複寫。

全域資料表的服務連結角色

DynamoDB 全域資料表依賴服務連結角色 (SLRs) 來管理跨區域複寫和自動擴展功能。

您只需要為每個 AWS 帳戶設定這些角色一次。建立之後，相同的角色會為您帳戶中的所有全域資料表提供服務。如需服務連結角色的詳細資訊，請參閱《IAM 使用者指南》中的使用服務連結角色。

複寫服務連結角色

當您建立第一個全域資料表時，Amazon DynamoDB 會自動建立`AWSServiceRoleForDynamoDBReplication`服務連結角色 (SLR)。此角色會為您管理跨區域複寫。

將資源型政策套用至複本時，請確定您不會拒絕 中定義的任何許可AWSServiceRoleForDynamoDBReplicationPolicy給 SLR 委託人，因為這會中斷複寫。如果您拒絕必要的 SLR 許可權限，將停止與受影響複本之間的複寫，而複本資料表狀態將變更為 REPLICATION_NOT_AUTHORIZED。

- 對於多區域最終一致性 (MREC) 全域資料表，如果複本保持 REPLICATION_NOT_AUTHORIZED 狀態超過 20 小時，則複本會不可逆地轉換為單一區域 DynamoDB 資料表。
- 對於多區域強式一致性 (MRSC) 全域資料表，拒絕必要的許可會導致 AccessDeniedException進行寫入和強式一致讀取操作。如果複本停留在 REPLICATION_NOT_AUTHORIZED 狀態超過七天，複本會永久無法存取，而且寫入和高度一致的讀取操作會繼續失敗並發生錯誤。複本刪除等部分管理操作將會成功。

自動擴展服務連結角色

設定佈建容量模式的全域資料表時，必須為全域資料表設定自動擴展。DynamoDB Auto Scaling 服務會使用 AWS Application Auto Scaling 服務動態調整全域資料表複本上的佈建輸送量容量。Application Auto Scaling 服務會建立名為 的服務連結角色 (SLR)AWSServiceRoleForApplicationAutoScaling_DynamoDBTable。當您第一次設定 DynamoDB 資料表的自動擴展時，會在 AWS 您的帳戶中自動建立此服務連結角色。它允許 Application Auto Scaling 管理的佈建資料表容量並建立 CloudWatch 警示。

將資源型政策套用至複本時，請確定您不會拒絕 中定義的任何許可AWSApplicationAutoscalingDynamoDBTablePolicy給 Application Auto Scaling SLR 主體，因為這會中斷自動擴展功能。

服務連結角色的 IAM 政策範例

具有下列條件的 IAM 政策不會影響 DynamoDB 複寫 SLR 和 AWS Auto Scaling SLR 的必要許可。此條件可新增至其他廣泛限制的政策，以避免意外中斷複寫或自動擴展。

從拒絕政策中排除必要的 SLR 許可

下列範例顯示如何從拒絕陳述式中排除服務連結角色主體：

```
"Condition": {
    "StringNotEquals": {
        "aws:PrincipalArn": [
            "arn:aws::iam::111122223333:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication",
```

```
            "arn:aws::iam::111122223333:role/aws-service-role/dynamodb.application-
autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable"
        ]
    }
}
```

全域資料表如何使用 AWS IAM

下列各節說明不同全域資料表操作的必要許可，並提供政策範例，協助您為使用者和應用程式設定適當的存取權。

> **Note**
>
> 所描述的所有許可都必須套用至受影響區域中的特定資料表資源 ARN ()。資料表資源 ARN 遵循格式 `arn:aws:dynamodb:region:account-id:table/table-name`，您需要指定實際的區域、帳戶 ID 和資料表名稱值。

主題

- 建立全域資料表並新增複本
- 更新全域資料表
- 刪除全域資料表和移除複本

建立全域資料表並新增複本

DynamoDB 全域資料表支援兩種一致性模式：多區域最終一致性 (MREC) 和多區域強式一致性 (MRSC)。MREC 全域資料表可以在任意數量的區域中擁有多個複本，並提供最終一致性。MRSC 全域資料表只需要三個區域 （三個複本或兩個複本和一個見證），並提供與零復原點目標 (RPO) 的高度一致性。

建立全域資料表所需的許可取決於您是否建立具有或沒有見證的全域資料表。

建立全域資料表的許可

初始全域資料表建立和稍後新增複本都需要下列許可。這些許可同時適用於多區域最終一致性 (MREC) 和多區域強一致性 (MRSC) 全域資料表。

- 全域資料表需要跨區域複寫，DynamoDB 會透過AWSServiceRoleForDynamoDBReplication服務連結角色 (SLR) 進行管理。下列許可允許 DynamoDB 在您第一次建立全域資料表時自動建立此角色：

  - iam:CreateServiceLinkedRole
- 若要使用 UpdateTable API 建立全域資料表或新增複本，您必須具有來源資料表資源的下列許可：
  - dynamodb:UpdateTable
- 您必須對 區域中的資料表資源具有下列許可，才能新增複本：
  - dynamodb:CreateTable
  - dynamodb:CreateTableReplica
  - dynamodb:Query
  - dynamodb:Scan
  - dynamodb:UpdateItem
  - dynamodb:PutItem
  - dynamodb:GetItem
  - dynamodb:DeleteItem
  - dynamodb:BatchWriteItem

使用見證的 MRSC 全域資料表的其他許可

使用見證區域建立多區域強一致性 (MRSC) 全域資料表時，您必須對所有參與區域中的資料表資源（包括複本區域和見證區域）具有下列許可：

- dynamodb:CreateGlobalTableWitness

建立全域資料表的 IAM 政策範例

跨三個區域建立 MREC 或 MRSC 全域資料表

下列身分型政策可讓您跨三個區域建立名為「使用者」的 MREC 或 MRSC 全域資料表，包括建立必要的 DynamoDB 複寫服務連結角色。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
```

```
      "Sid": "AllowCreatingUsersGlobalTable",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users",
        "arn:aws:dynamodb:us-west-2:123456789012:table/users"
      ]
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

將 MREC 或 MRSC 全域資料表建立限制在特定區域

下列身分型政策可讓您使用 aws：RequestedRegion 條件金鑰跨特定區域建立 DynamoDB 全域資料表複本，包括建立所需的 DynamoDB 複寫服務連結角色。

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAddingReplicasToSourceTable",
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateTable"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-1"
          ]
        }
      }
    },
    {
      "Sid": "AllowCreatingReplicas",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-2",
            "us-west-2"
          ]
        }
      }
    },
    {
      "Sid": "AllowCreatingSLR",
```

```
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

使用見證建立 MRSC 全域資料表

下列身分型政策可讓您建立名為「使用者」的 DynamoDB MRSC 全域資料表，其中包含 us-east-1 和 us-east-2 中的複本，以及 us-west-2 中的見證，包括建立所需的 DynamoDB 複寫服務連結角色。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreatingUsersGlobalTableWithWitness",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:CreateGlobalTableWitness",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users"
      ]
```

```
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

限制在特定區域的 MRSC 見證建立

此身分型政策可讓您使用 aws：RequestedRegion 條件金鑰，以及跨所有區域建立不受限制的見證，來建立複本僅限於特定區域的 MRSC 全域資料表，包括建立所需的 DynamoDB 複寫服務連結角色。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreatingReplicas",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "*",
      "Condition": {
```

```
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-1",
            "us-east-2"
          ]
        }
      }
    },
    {
      "Sid": "AllowCreatingWitness",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateGlobalTableWitness"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

更新全域資料表

若要使用 UpdateTable API 修改現有全域資料表的複本設定，您需要對進行 API 呼叫的區域中的資料表資源具有下列許可：

- dynamodb:UpdateTable

您也可以更新其他全域資料表組態，例如自動擴展政策和存留時間設定。這些額外的更新操作需要下列許可：

- 若要使用 UpdateTableReplicaAutoScaling API 更新複本自動擴展政策，您必須在包含複本的所有區域中的資料表資源上具有下列許可：
  - 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
- 若要使用 UpdateTimeToLive API 更新存留時間設定，您必須在包含複本的所有區域中對資料表資源擁有下列許可：
  - dynamodb:UpdateTimeToLive

  請注意，存留時間 (TTL) 僅支援使用多區域最終一致性 (MREC) 設定的全域資料表。如需全域資料表如何使用 TTL 的詳細資訊，請參閱 DynamoDB 全域資料表的運作方式。

刪除全域資料表和移除複本

若要刪除全域資料表，您必須移除所有複本。此操作所需的許可會因您是否刪除具有或沒有見證區域的全域資料表而有所不同。

刪除全域資料表和移除複本的許可

移除個別複本和完全刪除全域資料表需要下列許可。刪除全域資料表組態只會移除不同區域中資料表之間的複寫關係。它不會刪除最後一個剩餘區域中的基礎 DynamoDB 資料表。最後一個區域中的資料表會繼續以具有相同資料和設定的標準 DynamoDB 資料表存在。這些許可同時適用於多區域最終一致性 (MREC) 和多區域強一致性 (MRSC) 全域資料表。

- 若要使用 UpdateTable API 從全域資料表移除複本，您需要對進行 API 呼叫的區域中的資料表資源具有下列許可：
  - dynamodb:UpdateTable
- 您需要在移除複本的每個區域中的資料表資源具有下列許可：

- dynamodb:DeleteTable
- dynamodb:DeleteTableReplica

使用見證的 MRSC 全域資料表的其他許可

若要使用見證刪除多區域強式一致性 (MRSC) 全域資料表，您必須對所有參與區域中的資料表資源（包括複本區域和見證區域）具有下列許可：

- dynamodb:DeleteGlobalTableWitness

刪除全域資料表複本的 IAM 政策範例

刪除全域資料表複本

此身分型政策可讓您跨三個區域刪除名為「使用者」的 DynamoDB 全域資料表及其複本：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateTable",
        "dynamodb:DeleteTable",
        "dynamodb:DeleteTableReplica"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users",
        "arn:aws:dynamodb:us-west-2:123456789012:table/users"
      ]
    }
  ]
}
```

使用見證刪除 MRSC 全域資料表

此身分型政策可讓您刪除名為 "users" 的 MRSC 全域資料表複本和見證：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateTable",
        "dynamodb:DeleteTable",
        "dynamodb:DeleteTableReplica",
        "dynamodb:DeleteGlobalTableWitness"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users"
      ]
    }
  ]
}
```

全域資料表的使用方式 AWS KMS

與所有 DynamoDB 資料表一樣，全域資料表複本一律使用存放在 AWS Key Management Service () 中的加密金鑰來加密靜態資料AWS KMS。

全域資料表的所有複本都必須設定為相同類型的 KMS 金鑰 (AWS 擁有的金鑰、 AWS 受管金鑰或客戶自管金鑰)。

> **Important**
>
> DynamoDB 需要存取複本的加密金鑰才能刪除複本。如果您因為需要刪除複本而想停用或刪除用於加密複本的客戶自管金鑰，您應該先刪除複本，等待其中一個剩餘複本的資料表狀態變更為 ACTIVE，然後再停用或刪除金鑰。

對於設定為多區域最終一致性 (MREC) 的全域資料表，如果您停用或撤銷 DynamoDB 對用於加密複本之客戶自管金鑰的存取權，則將停止與複本之間的複寫，複本狀態將變更為 INACCESSIBLE_ENCRYPTION_CREDENTIALS。如果複本在 MREC 全域資料表中保持

INACCESSIBLE_ENCRYPTION_CREDENTIALS 狀態超過 20 小時，則複本會不可復原地轉換為單一區域 DynamoDB 資料表。

對於設定為多區域強一致性 (MRSC) 的全域資料表，如果您停用或撤銷 DynamoDB 對用於加密複本之客戶自管金鑰的存取權，則將停止與複本之間的複寫，嘗試對複本執行寫入或高度一致性讀取將傳回錯誤，且複本狀態將變更為 INACCESSIBLE_ENCRYPTION_CREDENTIALS。如果 MRSC 全域資料表的複本保持 INACCESSIBLE_ENCRYPTION_CREDENTIALS 狀態超過七天，則視撤銷的特定許可權限而定，複本將封存或永久無法存取。

DynamoDB 多帳戶全域資料表

多帳戶全域資料表會自動跨多個 AWS 區域和多個 AWS 帳戶複寫 DynamoDB 資料表資料，以改善彈性、隔離帳戶層級的工作負載，以及套用不同的安全和控管控制。每個複本資料表都位於不同的 AWS 帳戶中，在區域和帳戶層級啟用故障隔離。您也可以將複本與您的 AWS 組織結構對齊。相較於相同帳戶全域資料表，多帳戶全域資料表提供額外的隔離、控管和安全性優勢。

多帳戶全域資料表提供下列優點：

- 在您選擇的 AWS 帳戶和區域中自動複寫 DynamoDB 資料表資料
- 使用不同的政策、護欄和合規界限跨帳戶複寫資料，以增強安全性和管理
- 將複本放在不同的帳戶中，以改善操作彈性和 AWS 帳戶層級的故障隔離
- 使用多帳戶策略時，依業務單位或擁有權調整工作負載
- 將每個複本計費至其個別 AWS 帳戶，以簡化成本歸因

如需詳細資訊，請參閱使用多個 AWS 帳戶的優點。如果您的工作負載不需要多帳戶複寫，或者您想要使用本機覆寫進行更簡單的複本管理，您可以繼續使用相同的帳戶全域資料表。

您可以使用 設定多帳戶全域資料表多區域最終一致性 (MREC)。為 設定的全域資料表多區域強一致性 (MRSC)不支援多帳戶模型。

主題

- DynamoDB 全域資料表的運作方式
- 教學課程：建立多帳戶全域資料表
- DynamoDB 全域資料表安全性

DynamoDB 全域資料表的運作方式

多帳戶全域資料表擴展 DynamoDB 全域資料表的全受管、無伺服器、多區域和多作用中功能，以跨多個 AWS 帳戶。多帳戶全域資料表跨 AWS 區域和帳戶複寫資料，提供與相同帳戶全域資料表相同的主動-主動功能。當您寫入任何複本時，DynamoDB 會將資料複寫至所有其他複本。

與相同帳戶全域資料表的主要差異包括：

- 多區域最終一致性 (MREC) 全域資料表支援多帳戶複寫。
- 您只能從單一區域資料表開始新增複本。不支援將現有的相同帳戶全域資料表轉換為多帳戶設定。若要遷移，您必須先刪除現有的複本，才能返回單一區域資料表，然後再建立新的多帳戶全域資料表。
- 每個複本必須位於個別 AWS 帳戶中。對於具有 N 個複本的多帳戶全域資料表，您必須擁有 N 個帳戶。
- 根據預設，多帳戶全域資料表會在所有複本中使用統一的資料表設定。所有複本會自動共用相同的組態（例如輸送量模式和 TTL），而且與相同帳戶全域資料表不同，每個複本都無法覆寫這些設定。
- 客戶必須在其資源政策中提供複寫許可給 DynamoDB 全域資料表服務主體。

多帳戶全域資料表使用與相同帳戶全域資料表相同的基礎複寫技術。資料表設定會自動複寫到所有區域複本，客戶無法覆寫或自訂每個複本的設定。這可確保在參與相同全域資料表的多個 AWS 帳戶之間一致的組態和可預測的行為。

DynamoDB 全域資料表中的設定會定義資料表的行為方式，以及跨區域複寫資料的方式。這些設定是在建立資料表期間或新增區域複本時，透過 DynamoDB 控制平面 APIs 進行設定。

建立多帳戶全域資料表時，客戶必須為每個區域複本設定 `GlobalTableSettingsReplicationMode = ENABLED`。這可確保在一個區域中所做的組態變更會自動傳播到參與全域資料表的所有其他區域。

您可以在建立資料表後啟用設定複寫。這支援將資料表最初建立為區域資料表，稍後升級至多帳戶全域資料表的情況。

同步設定

下表設定一律會同步多帳戶全域資料表中所有複本：

> **Note**
>
> 與相同帳戶全域資料表不同，多帳戶全域資料表不允許對這些設定進行每個區域覆寫。唯一的例外是，允許覆寫讀取自動擴展政策（資料表和 GSIs），因為它們是單獨的外部資源。

- 容量模式 (佈建容量或隨需)
- 資料表佈建的讀取和寫入容量
- 資料表讀取和寫入自動擴展
- 本機次要索引 (LSI) 定義
- 全域次要索引 (GSI) 定義
- GSI 佈建的讀取和寫入容量
- GSI 讀取和寫入自動擴展
- MREC 模式中的 Streams 定義
- 存留時間 (TTL)
- 暖輸送量
- 隨需最大讀取和寫入輸送量

非同步設定

下列設定不會在複本之間同步，且必須針對每個區域中的每個複本資料表獨立設定。

- 資料表類別
- 伺服器端加密 (SSE) 類型
- 時間點復原
- 伺服器端加密 (SSE) KMS 金鑰 ID
- 刪除保護
- Kinesis Data Streams (KDSD)
- Tags (標籤)
- 資源政策
- 資料表 Cloudwatch-Contributor Insights (CCI)
- GSI Cloudwatch-Contributor Insights (CCI)

監控

如果全域資料表已設定多區域最終一致性 (MREC)，其會將 `ReplicationLatency` 指標發布至 CloudWatch。假設某個項目寫入複本資料表，此指標可追蹤該項目出現在全域資料表中其他複本時經過的時間。`ReplicationLatency` 會以毫秒表示，並會針對全域資料表中每對來源及目的地區域對發送。

典型ReplicationLatency值取決於所選 AWS 區域之間的距離，以及其他變數，例如工作負載類型和輸送量。例如，相較於非洲 (開普敦) (af-south-1) 區域，美國西部 (加利佛尼亞北部) (us-west-1) 區域的來源複本到美國西部 (奧勒岡) (us-west-2) 區域的 ReplicationLatency 較低。

ReplicationLatency 值上升可能表示，某個複本的更新未及時傳播到其他複本資料表。在此情況下，您可以暫時將應用程式的讀取和寫入活動重新導向至不同的 AWS 區域。

處理多帳戶全域資料表中的複寫延遲問題

如果由於客戶在複本資料表上引發的問題而ReplicationLatency超過 3 小時，DynamoDB 會傳送通知，要求客戶解決基礎問題。可能阻止複寫的常見客戶引發問題包括：

- 從複本資料表的資源政策中移除必要的許可
- 選擇退出託管多帳戶全域資料表複本的 AWS 區域
- 拒絕解密資料所需的資料表 AWS KMS 金鑰許可

DynamoDB 會在複寫延遲升高的 3 小時內傳送初始通知，然後在 20 小時後傳送第二次通知，如果問題仍未解決。如果未在所需的時段內修正問題，DynamoDB 會自動取消複本與全域資料表的關聯。受影響的複本接著會轉換為區域資料表。

教學課程：建立多帳戶全域資料表

本節提供step-by-step說明。 DynamoDB AWS

使用 DynamoDB 主控台建立多帳戶全域資料表

請依照下列步驟，使用 建立多帳戶全域資料表 AWS 管理主控台。下列範例會在美國建立具有複本資料表的全域資料表。

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/dynamodb/：// 開啟第一個帳戶的 DynamoDB 主控台 （ 例如 *111122223333*)。
2. 在此範例中，請從導覽列的區域選擇器選擇美國東部 (俄亥俄)。
3. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
4. 選擇 Create Table (建立資料表)。
5. 在建立資料表頁面上：
    a. 對於 Table name (資料表名稱)，請輸入 **`MusicTable`**。
    b. 對於 Partition key (分割區索引鍵)，請輸入 **`Artist`**。

c. 對於排序索引鍵，請輸入 **SongTitle**。

d. 保留其他預設設定，並選擇建立資料表。

6. 將下列資源政策新增至資料表

JSON

```
{
"Version":"2012-10-17",
"Statement": [
    {
        "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
        "Effect": "Allow",
        "Action": [
            "dynamodb:ReadDataForReplication",
            "dynamodb:WriteDataForReplication",
            "dynamodb:ReplicateSettings"
        ],
        "Resource": "arn:aws:dynamodb:us-east-2:111122223333:table/
MusicTable",
        "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": ["444455556666","111122223333"],
                "aws:SourceArn": [
                    "arn:aws:dynamodb:us-east-1:444455556666:table/
MusicTable",
                    "arn:aws:dynamodb:us-east-2:111122223333:table/
MusicTable"
                ]
            }
        }
    },
    {
        "Sid": "AllowTrustedAccountsToJoinThisGlobalTable",
        "Effect": "Allow",
        "Action": [
            "dynamodb:AssociateTableReplica"
        ],
        "Resource": "arn:aws:dynamodb:us-east-2:111122223333:table/
MusicTable",
        "Principal": {"AWS": ["444455556666"]}
    }
```

```
]
}
```

7. 此新資料表會做為新全域資料表中的第一個複本資料表。這是您稍後新增其他複本資料表的原型。
8. 等待資料表變為作用中。對於新建立的資料表，從全域資料表索引標籤導覽至設定複寫，然後按一下啟用。
9. 此帳戶的登出（此處為 *111122223333*）。
10. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/dynamodb/：// 開啟第二個帳戶的 DynamoDB 主控台（例如 *444455556666*）。
11. 在此範例中，從導覽列的區域選擇器中選擇美國東部（維吉尼亞北部）。
12. 主控台會確認選取區域中沒有同名的資料表。若有同名的資料表，您必須先刪除現有的資料表，才可在該區域中建立新的複本資料表。
13. 在建立資料表附近的下拉式清單中，選擇從另一個帳戶建立
14. 在從另一個帳戶建立資料表頁面上：
    a. 新增 **arn:aws:dynamodb:us-east-2:*111122223333*:table/MusicTable**做為來源資料表的資料表 Arn。
    b. 在複本資料表 ARNs 中，再次新增來源資料表的 ARN**arn:aws:dynamodb:us-east-2:*111122223333*:table/MusicTable**。如果多帳戶全域資料表中已有多個複本，您必須將每個現有複本新增至ReplicaTableARN。
    c. 保留其他預設設定，然後選擇提交。
15. 音樂資料表（以及任何其他複本資料表）的全域資料表索引標籤會顯示資料表已在多個區域中複寫。
16. 若要測試複寫：
    a. 您可以使用此資料表存在複本的任何區域
    b. 選擇探索資料表項目。
    c. 選擇 Create item (建立項目)。
    d. 為 Artist 輸入 **item_1**，為 SongTitle 輸入 **Song Value 1**。
    e. 選擇 Create item (建立項目)。
    f. 切換到其他區域來驗證複寫：
    g. 確認 Music 資料表包含您建立的項目。

使用 建立多帳戶全域資料表 AWS CLI

下列範例示範如何使用 建立多帳戶全域資料表 AWS CLI。這些範例示範設定跨帳戶複寫的完整工作流程。

CLI

使用下列 AWS CLI 命令建立具有跨帳戶複寫的多帳戶全域資料表。

```
# STEP 1: Setting resource policy for the table in account 111122223333

cat > /tmp/source-resource-policy.json << 'EOF'
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:us-east-2:111122223333:table/MusicTable",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": ["444455556666","111122223333"],
                    "aws:SourceArn": [
                        "arn:aws:dynamodb:us-east-1:444455556666:table/MusicTable",
                        "arn:aws:dynamodb:us-east-2:111122223333:table/MusicTable"
                    ]
                }
            }
        },
        {
            "Sid": "AllowTrustedAccountsToJoinThisGlobalTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:AssociateTableReplica"
            ],
            "Resource": "arn:aws:dynamodb:us-east-2:111122223333:table/MusicTable",
            "Principal": {"AWS": ["444455556666"]}
        }
```

```
    ]
}
EOF

# Step 2: Create a new table (MusicTable) in US East (Ohio),
#   with DynamoDB Streams enabled (NEW_AND_OLD_IMAGES),
#   and Settings Replication ENABLED on the account 111122223333

aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --global-table-settings-replication-mode ENABLED \
    --resource-policy file:///tmp/source-resource-policy.json \
    --region us-east-2


# Step 3: Creating replica table in account 444455556666

# Resource policy for account 444455556666
cat > /tmp/dest-resource-policy.json << 'EOF'
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:444455556666:table/MusicTable",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": ["444455556666","111122223333"],
                    "aws:SourceArn": [
```

```
                        "arn:aws:dynamodb:us-east-1:444455556666:table/MusicTable",
                        "arn:aws:dynamodb:us-east-2:111122223333:table/MusicTable"
                    ]
                }
            }
        }
    ]
}
EOF

# Execute the replica table creation
aws dynamodb create-table \
    --table-name MusicTable \
    --global-table-source-arn "arn:aws:dynamodb:us-east-2:111122223333:table/
MusicTable" \
    --resource-policy file:///tmp/dest-resource-policy.json \
    --global-table-settings-replication-mode ENABLED \
    --region us-east-1

# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistency,R
{Region:RegionName,Status:ReplicaStatus}}'

# Step 5: To verify that replication is working, add a new item to the Music table
 in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2

# Step 6: Wait for a few seconds, and then check to see whether the item has been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
```

```
    --region us-east-2

# Step 7: Delete the replica table in US East (N. Virginia) Region
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-1

# Clean up: Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

DynamoDB 全域資料表安全性

全域資料表複本是 DynamoDB 資料表，因此您可以使用與單一區域資料表相同的方法來控制對複本的存取，包括 AWS Identity and Access Management (IAM) 身分政策和以資源為基礎的政策。本主題說明如何使用 IAM 許可和 AWS Key Management Service (AWS KMS) 加密保護 DynamoDB 多帳戶全域資料表。您可以了解允許跨區域跨帳戶複寫和自動擴展的資源型政策和服務連結角色 (SLR)，以及建立、更新和刪除全域資料表所需的 IAM 許可，以取得多區域最終一致性 (MREC) 資料表。您也會了解 AWS KMS 加密金鑰，以安全地管理跨區域複寫。

它提供有關建立跨帳戶和跨區域資料表複寫所需的資源型政策和許可的詳細資訊。對於需要實作安全、跨帳戶資料複寫解決方案的客戶而言，了解此安全模型至關重要。

複寫的服務主體授權

DynamoDB 的多帳戶全域資料表使用不同的授權方法，因為複寫是跨帳戶邊界執行。這是使用 DynamoDB 的複寫服務主體來完成：`replication.dynamodb.amazonaws.com`。每個參與帳戶都必須在複本資料表的資源政策中明確允許該主體，授予其許可，該許可可由 `aws:SourceAccount`、等金鑰上的來源內容條件限制為特定複本 — 如需詳細資訊`aws:SourceArn`，請參閱AWS 全域條件金鑰。許可是雙向的，這表示所有複本都必須明確地互相授予許可，才能跨任何特定複本對建立複寫。

下列服務主體許可對於跨帳戶複寫至關重要：

- `dynamodb:ReadDataForReplication` 授予讀取資料以用於複寫的能力。此許可允許讀取一個複本中的變更，並將其傳播到其他複本。
- `dynamodb:WriteDataForReplication` 允許將複寫的資料寫入目的地資料表。此許可允許全域資料表中所有複本的變更同步。

- dynamodb:ReplicateSettings 啟用跨複本的資料表設定同步，在所有參與的資料表之間提供一致的組態。

每個複本都必須將上述許可授予所有其他複本和 本身，即來源內容條件必須包含包含全域資料表的完整複本集。這些許可會在新增至多帳戶全域資料表時，針對每個新複本進行驗證。這會驗證複寫操作僅由授權的 DynamoDB 服務執行，並且僅在預期的資料表之間執行。

多帳戶全域資料表的服務連結角色

DynamoDB 多帳戶全域資料表會複寫所有複本的設定，讓每個複本都設定一致輸送量，並提供無縫的容錯移轉體驗。設定複寫是透過服務主體的 ReplicateSettings 許可來控制，但我們也依賴服務連結角色 (SLRs) 來管理特定跨帳戶跨區域複寫和自動擴展功能。這些角色每個 AWS 帳戶只會設定一次。建立之後，相同的角色會為您帳戶中的所有全域資料表提供服務。如需服務連結角色的詳細資訊，請參閱《IAM 使用者指南》中的使用服務連結角色。

設定管理服務連結角色

當您在帳戶中建立第一個多帳戶全域資料表複本時，Amazon DynamoDB 會自動建立 AWSServiceRoleForDynamoDBGlobalTableSettingsManagement 服務連結角色 (SLR)。此角色會為您管理設定的跨帳戶跨區域複寫。

將資源型政策套用至複本時，請確認您不會拒絕 中定義的任何許可AWSServiceRoleForDynamoDBGlobalTableSettingsManagement給 SLR 主體，因為這可能會干擾設定管理，如果複本或 GSIs 之間的輸送量不相符，則可能會影響複寫。如果您拒絕必要的 SLR 許可，對受影響複本的複寫和複寫可能會停止，而複本資料表狀態會變更為 REPLICATION_NOT_AUTHORIZED。對於多帳戶全域資料表，如果複本保持 REPLICATION_NOT_AUTHORIZED 狀態超過 20 小時，則複本會不可復原地轉換為單一區域 DynamoDB 資料表。SLR 具有下列許可：

- application-autoscaling:DeleteScalingPolicy
- application-autoscaling:DescribeScalableTargets
- application-autoscaling:DescribeScalingPolicies
- application-autoscaling:DeregisterScalableTarget
- application-autoscaling:PutScalingPolicy
- application-autoscaling:RegisterScalableTarget

自動擴展服務連結角色

設定佈建容量模式的全域資料表時，必須為全域資料表設定自動擴展。DynamoDB Auto Scaling 服務會使用 AWS Application Auto Scaling 服務動態調整全域資料表複本上的佈建輸送量容量。Application Auto Scaling 服務會建立名為 AWSServiceRoleForApplicationAutoScaling_DynamoDBTable 的服務連結角色 (SLR)。當您第一次設定 DynamoDB 資料表的自動擴展時，會在 AWS 您的帳戶中自動建立此服務連結角色。它允許 Application Auto Scaling 管理佈建的資料表容量並建立 CloudWatch 警示。

將資源型政策套用至複本時，請確認您未拒絕 AWSApplicationAutoscalingDynamoDBTablePolicy 中定義的任何許可至 Application Auto Scaling SLR 主體，因為這會中斷自動調整規模功能。

全域資料表如何使用 AWS IAM

下列各節說明不同全域資料表操作的必要許可，並提供政策範例，協助您為使用者和應用程式設定適當的存取權。

> **Note**
>
> 所描述的所有許可都必須套用至受影響區域中的特定資料表資源 ARN ()。資料表資源 ARN 遵循格式 `arn:aws:dynamodb:region:account-id:table/table-name`，您需要指定實際的區域、帳戶 ID 和資料表名稱值。

以下是我們在下列各節涵蓋的step-by-step主題：

- 建立多帳戶全域資料表並新增複本
- 更新多帳戶全域資料表
- 刪除全域資料表和移除複本

建立全域資料表並新增複本

建立全域資料表的許可

當新的複本新增至區域資料表以形成多帳戶全域資料表或現有多帳戶全域資料表時，執行動作的 IAM 主體必須經過所有現有成員的授權。所有現有成員都必須在其資料表政策中授予下列許可，才能成功新增複本：

- `dynamodb:AssociateTableReplica` - 此許可允許資料表加入全域資料表設定。這是啟用初始建立複寫關係的基礎許可。

此精確控制僅允許授權帳戶參與全域資料表設定。

建立全域資料表的 IAM 政策範例

2 個複本設定的範例 IAM 政策

多帳戶全域資料表的設定遵循提供安全複寫的特定授權流程。讓我們演練一個實際案例，其中客戶想要建立具有兩個複本的全球資料表，藉此檢查其實際運作方式。第一個複本 (ReplicaA) 位於 ap-east-1 區域中的帳戶 A，而第二個複本 (ReplicaB) 位於 eu-south-1 區域中的帳戶 B。

- 在來源帳戶 （帳戶 A) 中，程序會從建立主要複本資料表開始。帳戶管理員必須將資源型政策連接至此資料表，以明確授予目的地帳戶 （帳戶 B) 執行關聯的必要許可。此政策也會授權 DynamoDB 複寫服務執行必要的複寫動作。
- 目的地帳戶 （帳戶 B) 遵循類似的程序，方法是在建立複本時連接對應的資源型政策，並參考要用來建立複本的來源資料表 ARN。此政策會鏡像帳戶 A 授予的許可，建立信任的雙向關係。在建立複寫之前，DynamoDB 會驗證這些跨帳戶許可，以確認已具備適當的授權。

若要建立此設定：

- 帳戶 A 的管理員必須先將資源型政策連接至ReplicaA。此政策明確授予帳戶 B 和 DynamoDB 複寫服務必要的許可。
- 同樣地，帳戶 B 的管理員必須將相符政策連接至ReplicaB，並在建立資料表呼叫中將對應的許可反向授予帳戶 A，以建立參考複本 A 做為來源資料表的複本 B。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:ap-east-1:111122223333:table/ReplicaA",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
```

```
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": [ "111122223333", "444455556666" ],
                    "aws:SourceArn": [
                        "arn:aws:dynamodb:ap-east-1:111122223333:table/ReplicaA",
                        "arn:aws:dynamodb:eu-south-1:444455556666:table/ReplicaB"
                    ]
                }
            }
        },
        {
            "Sid": "AllowTrustedAccountsToJoinThisGlobalTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:AssociateTableReplica"
            ],
            "Resource": "arn:aws:dynamodb:ap-east-1:111122223333:table/ReplicaA",
            "Principal": {"AWS": ["444455556666"]}
        }
    ]
}
```

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:eu-south-1:444455556666:table/
ReplicaB",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": [ "111122223333", "444455556666" ],
                    "aws:SourceArn": [
```

```
                    "arn:aws:dynamodb:ap-east-1:111122223333:table/ReplicaA",
                    "arn:aws:dynamodb:eu-south-1:444455556666:table/ReplicaB"
                ]
            }
        }
    }
]
}
```

3 個複本設定的範例 IAM 政策

在此設定中，帳戶 A、帳戶 ReplicaB 和帳戶 C 分別有 3 個複本ReplicaA A、複本 B 和ReplicaC C。複本 A 是第一個複本，其開頭為區域資料表，然後將ReplicaB 和ReplicaC 新增至其中。

- 帳戶 A 的管理員必須先將資源型政策連接至ReplicaA，以允許複寫所有成員，並允許帳戶 B 和帳戶 C 的 IAM 主體新增複本。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:ap-east-1:111122223333:table/ReplicaA",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": [ "111122223333", "444455556666",
 "123456789012" ],
                    "aws:SourceArn": [
                        "arn:aws:dynamodb:ap-east-1:111122223333:table/ReplicaA",
                        "arn:aws:dynamodb:eu-south-1:444455556666:table/
ReplicaB",
                        "arn:aws:dynamodb:us-east-1:123456789012:table/ReplicaC"
```

```
                    ]
                }
            }
        },
        {
            "Sid": "AllowTrustedAccountsToJoinThisGlobalTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:AssociateTableReplica"
            ],
            "Resource": "arn:aws:dynamodb:ap-east-1:111122223333:table/ReplicaA",
            "Principal": { "AWS": [ "444455556666", "123456789012" ] }
        }
    ]
}
```

- 帳戶 B 的管理員必須新增指向ReplicaA做為來源的複本 （複本 B)。複本 B 具有下列政策，允許所有成員之間複寫，並允許帳戶 C 新增複本：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:eu-south-1:444455556666:table/
ReplicaB",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": [ "111122223333", "444455556666",
 "123456789012" ],
                    "aws:SourceArn": [
                        "arn:aws:dynamodb:ap-east-1:111122223333:table/ReplicaA",
```

```
                        "arn:aws:dynamodb:eu-south-1:444455556666:table/
ReplicaB",
                        "arn:aws:dynamodb:us-east-1:123456789012:table/ReplicaC"
                    ]
                }
            }
        },
        {
            "Sid": "AllowTrustedAccountsToJoinThisGlobalTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:AssociateTableReplica"
            ],
            "Resource": "arn:aws:dynamodb:eu-south-1:444455556666:table/
ReplicaB",
            "Principal": { "AWS": [ "123456789012" ] }
        }
    ]
}
```

- 最後，帳戶 C 的管理員會使用允許所有成員之間複寫許可的下列政策來建立複本。此政策不允許新增任何進一步的複本。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/ReplicaC",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": [ "111122223333", "444455556666" ],
```

```
                    "aws:SourceArn": [
                        "arn:aws:dynamodb:ap-east-1:111122223333:table/ReplicaA",
                        "arn:aws:dynamodb:eu-south-1:444455556666:table/ReplicaB"
                    ]
                }
            }
        }
    ]
}
```

更新多帳戶全域資料表

若要使用 UpdateTable API 修改現有全域資料表的複本設定，您需要對進行 API 呼叫的區域中的資料表資源具有下列許可： dynamodb:UpdateTable

您也可以更新其他全域資料表組態，例如自動擴展政策和存留時間設定。這些額外的更新操作需要下列許可：

若要使用 UpdateTimeToLive API 更新存留時間設定，您必須在包含複本的所有區域中對資料表資源擁有下列許可： dynamodb:UpdateTimeToLive

若要使用 UpdateTableReplicaAutoScaling API 更新複本自動擴展政策，您必須在包含複本的所有區域中對資料表資源擁有下列許可：

- application-autoscaling:DeleteScalingPolicy
- application-autoscaling:DeleteScheduledAction
- application-autoscaling:DeregisterScalableTarget
- application-autoscaling:DescribeScalableTargets
- application-autoscaling:DescribeScalingActivities
- application-autoscaling:DescribeScalingPolicies
- application-autoscaling:DescribeScheduledActions
- application-autoscaling:PutScalingPolicy
- application-autoscaling:PutScheduledAction
- application-autoscaling:RegisterScalableTarget

> **Note**
>
> 您需要跨所有複本區域和帳戶提供dynamodb:ReplicateSettings許可，更新資料表才能成功。如果任何複本不提供將設定複寫到多帳戶全域資料表中任何複本的許可，則所有複本的所有更新操作都會失敗，AccessDeniedException直到許可修正為止。

刪除全域資料表和移除複本

若要刪除全域資料表，您必須移除所有複本。與相同帳戶全域資料表不同，您無法使用 UpdateTable刪除遠端區域中的複本資料表，而且每個複本必須透過 DeleteTable API 從控制該複本的帳戶刪除。

刪除全域資料表和移除複本的許可

移除個別複本和完全刪除全域資料表需要下列許可。刪除全域資料表組態只會移除不同區域中資料表之間的複寫關係。它不會刪除最後一個剩餘區域中的基礎 DynamoDB 資料表。最後一個區域中的資料表會繼續以具有相同資料和設定的標準 DynamoDB 資料表存在。

您需要在移除複本的每個區域中的資料表資源具有下列許可：

- dynamodb:DeleteTable
- dynamodb:DeleteTableReplica

全域資料表的使用方式 AWS KMS

與所有 DynamoDB 資料表一樣，全域資料表複本一律使用存放在 AWS Key Management Service () 中的加密金鑰來加密靜態資料AWS KMS。

> **Note**
>
> 與相同帳戶全域資料表不同，多帳戶全域資料表中的不同複本可以使用不同類型的 AWS KMS 金鑰 (AWS 擁有的金鑰或客戶受管金鑰） 進行設定。多帳戶全域資料表不支援 AWS 受管金鑰。

使用 CMKs多帳戶全域資料表需要每個複本的金鑰政策，才能授予 DynamoDB 複寫服務主體 (replication.dynamodb.amazonaws.com) 存取金鑰以進行複寫和設定管理的許可。需要以下許可：

- kms:Decrypt
- kms:ReEncrypt*
- kms:GenerateDataKey*
- kms:DescribeKey

Important (重要)

DynamoDB 需要存取複本的加密金鑰才能刪除複本。如果您想要停用或刪除用於加密複本的客戶受管金鑰，因為您要刪除複本，您應該先刪除複本，等待從複寫群組中移除資料表，方法是呼叫其中一個複本中的描述，然後停用或刪除金鑰。

如果您停用或撤銷 DynamoDB 對用於加密複本之客戶受管金鑰的存取權，複本的複寫將停止，複本狀態將變更為 INACCESSIBLE_ENCRYPTION_CREDENTIALS。如果複本保持 INACCESSIBLE_ENCRYPTION_CREDENTIALS 狀態超過 20 小時，複本會不可復原地轉換為單一區域 DynamoDB 資料表。

範例 AWS KMS 政策

此 AWS KMS 政策允許 DynamoDB 存取複本 A 和 B 之間的兩個複寫 AWS KMS 金鑰。每個帳戶中連接至 DynamoDB 複本的 AWS KMS 金鑰需要更新為下列政策：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": { "Service": "replication.dynamodb.amazonaws.com" },
        "Action": [
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": [ "111122223333", "444455556666" ],
                "aws:SourceArn": [
```

```
                "arn:aws:dynamodb:ap-east-1:111122223333:table/ReplicaA",
                "arn:aws:dynamodb:eu-south-1:444455556666:table/ReplicaB"
            ]
        }
      }
    }
  ]
}
```

了解全域資料表的 Amazon DynamoDB 計費

本指南說明 DynamoDB 計費如何用於全域資料表，提出各種有助於全域資料表成本的元件，並包括實際範例。

Amazon DynamoDB 全域資料表是全受管、無伺服器、多區域和多作用中的資料庫。全域資料表設計提供 99.999% 可用性，可提高應用程式恢復能力並加強業務連續性。全域資料表會在您選擇的 AWS 區域自動複寫 DynamoDB 資料表，因此您可以實現快速的本機讀取和寫入效能。

運作方式

全域資料表的計費模式與單一區域 DynamoDB 資料表不同。單一區域 DynamoDB 資料表的寫入操作會使用下列單位計費：

- 隨需容量模式的寫入請求單位 (WRU)，其中每次寫入都會收取一個 WRU，最多 1KB
- 佈建容量模式的寫入容量單位 (WCU)，其中一個 WCU 每秒提供一次寫入，最多 1 KB

您將複本資料表新增至現有單一區域資料表來建立全域資料表時，該單一區域資料表會成為複本資料表，亦即用於計費寫入資料表的單位也會變更。複本資料表的寫入操作會使用下列單位計費：

- 隨需容量模式的複寫寫入請求單位 (rWRU)，其中每個複本資料表會收取一個 rWRU，每個寫入最多 1KB
- 佈建容量模式的複寫寫入容量單位 (rWCU)，其中每個複本資料表的一個 WCU 每秒提供一次寫入，最多 1 KB

全域次要索引 (GSI) 更新的計費單位與單一區域 DynamoDB 資料表相同，即使 GSI 的基礎資料表是複本資料表。GSI 更新操作會使用下列單位計費：

- 隨需容量模式的寫入請求單位 (WRU)，其中每次寫入都會收取一個 WRU，最多 1KB

- 佈建容量模式的寫入容量單位 (WCU)，其中一個 WCU 每秒提供一次寫入，最多 1 KB

複寫寫入單位 (rWCU 和 rWRU) 定價與單一區域寫入單位 (WCU 和 WRU) 相同。跨區域資料傳輸費用適用於全域資料表，因為資料會跨區域複寫。複寫寫入 (rWCU 或 rWRU) 費用會在包含全域資料表複本資料表的每個區域中產生。

單一區域資料表和複本資料表的讀取操作會使用以下單位：

- 隨需容量模式的讀取請求單位 (RRU)，其中每個高度一致性讀取會收取一個 RRU，最多 4KB
- 佈建資料表的讀取容量單位 (RCU)，其中一個 RCU 每秒提供一次高度一致性讀取，最多 4KB

一致性模式和計費

對於多區域強一致性 (MRSC) 和多區域最終一致性 (MREC) 模式，用於寫入操作計費的複寫寫入單位 (rWCU 和 rWRU) 是相同的。使用以見證設定多區域強一致性 (MRSC) 模式的全域資料表，不會產生複寫寫入單位成本 (rWCU 和 rWRU)、儲存成本或複寫至見證的資料傳輸成本。

DynamoDB 全域資料表計費範例

以下將透過多日範例情境說明全域資料表寫入請求計費的實際運作方式 (請注意，此範例僅考慮寫入請求，不包含範例中可能產生的資料表還原和跨區域資料傳輸費用)：

第 1 天 - 單一區域資料表：您在 us-west-2 區域中有一個名為 Table_A 的單一區域隨需 DynamoDB 資料表。您將 100 個 1KB 項目寫入 Table_A。對這些單一區域寫入操作而言，每寫入 1KB 將收取 1 個寫入請求單位 (WRU)。您的第 1 天費用為：

- 在 us-west-2 區域中用於單一區域寫入的 100 個 WRU

第 1 天收取的總請求單位數：100 個 WRU。

第 2 天 - 建立全域資料表：您可以將複本新增至 us-east-2 區域中的 Table_A 來建立全域資料表。Table_A 現在是一個全域資料表，並有兩個複本資料表；一個在 us-west-2 區域中，另一個在 us-east-2 區域中。您將 150 個 1KB 項目寫入 us-west-2 區域中的複本資料表。您的第 2 天費用為：

- 在 us-west-2 區域中用於複寫寫入的 150 個 rWRU
- 在 us-east-2 區域中用於複寫寫入的 150 個 rWRU

第 2 天收取的總請求單位數：300 個 rWRU。

第 3 天 - 新增全域次要索引：您將全域次要索引 (GSI) 新增至 us-east-2 區域中的複本資料表，該複本資料表會從基底 (複本) 資料表投射所有屬性。全域資料表會自動在 us-west-2 區域中的複本資料表建立 GSI。您將 200 個新的 1KB 記錄寫入 us-west-2 區域中的複本資料表。您的第 3 天費用為：

- • 在 us-west-2 區域中用於複寫寫入的 200 個 rWRU
- • 在 us-west-2 區域中用於 GSI 更新的 200 個 WRU
- • 在 us-east-2 區域中用於複寫寫入的 200 個 rWRU
- • 在 us-east-2 區域中用於 GSI 更新的 200 個 WRU

第 3 天收取的總寫入請求單位：400 個 WRU 和 400 個 rWRU。

三天總計的寫入單位費用為 500 個 WRU (第 1 天 100 個 WRU + 第 3 天 400 個 WRU) 和 700 個 rWRU (第 2 天 300 個 rWRU + 第 3 天 400 rWRU)。

總而言之，複本資料表寫入操作會在包含複本資料表的所有區域中以複寫寫入單位計費。如果您有全域次要索引，則會針對所有包含 GSI 區域中的 GSI 更新 (全域資料表中包含複本資料表的所有區域) 以寫入單位向您收費。

DynamoDB 全域資料表版本

DynamoDB 全域資料表有兩種版本可用：全域資料表 2019.11.21 版 (目前版本) 和全域資料表 2017.11.29 版 (舊版)。建議您使用全域資料表 2019.11.21 版 (目前版本)，因為其比 2017.11.29 版 (舊版) 更易於使用、支援更多區域，且大多數的工作負載成本更低。

判斷全域資料表版本

使用 判斷版本 AWS CLI

識別全域資料表 2019.11.21 版 (目前版本) 複本

若要判斷資料表是否為全域資料表 2019.11.21 版 (目前版本) 複本，請針對資料表調用 `describe-table` 命令。如果輸出的 `GlobalTableVersion` 屬性含 "2019.11.21" 值，則資料表為 2019.11.21 版 (目前版本) 全域資料表複本。

`describe-table` 範例 CLI 命令：

```
aws dynamodb describe-table \
--table-name users \
--region us-east-2
```

(簡略版) 輸出的 GlobalTableVersion 屬性包含 "2019.11.21" 值，因此這個資料表是 2019.11.21 版 (目前版本) 全域資料表複本。

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "id",
                "AttributeType": "S"
            },
            {
                "AttributeName": "name",
                "AttributeType": "S"
            }
        ],
        "TableName": "users",
        ...
        "GlobalTableVersion": "2019.11.21",
        "Replicas": [
            {
                "RegionName": "us-west-2",
                "ReplicaStatus": "ACTIVE",
            }
        ],
        ...
    }
}
```

識別全域資料表 2017.11.29 版 (舊版) 複本

全域資料表 2017.11.29 版 (舊版) 使用一組專用命令進行全域資料表管理。若要判斷資料表是否為全域資料表 2017.11.29 版 (舊版) 複本，請針對資料表調用 describe-global-table 命令。如果您收到成功回應，則資料表是 2017.11.29 版 (舊版) 全域資料表複本。如果 describe-global-table 命令傳回 GlobalTableNotFoundException 錯誤，則資料表不是 2017.11.29 版 (舊版) 複本。

describe-global-table 範例 CLI 命令：

```
aws dynamodb describe-global-table \
--table-name users \
--region us-east-2
```

命令傳回成功回應，因此這個資料表是 2017.11.29 版 (舊版) 全域資料表複本。

```
{
    "GlobalTableDescription": {
        "ReplicationGroup": [
            {
                "RegionName": "us-west-2"
            },
            {
                "RegionName": "us-east-2"
            }
        ],
        "GlobalTableArn": "arn:aws:dynamodb::123456789012:global-table/users",
        "CreationDateTime": "2025-06-10T13:55:53.630000-04:00",
        "GlobalTableStatus": "ACTIVE",
        "GlobalTableName": "users"
    }
}
```

使用 DynamoDB 主控台判斷版本

若要識別全域資料表複本的版本，請執行下列動作：

1. 在 https://console.aws.amazon.com/dynamodb/home 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇您要識別全域資料表版本的資料表。
4. 選擇 Global Tables (全域資料表) 標籤。

   摘要區段會顯示所使用的全域資料表版本。

舊版和目前版本之間的行為差異

下列清單說明全域資料表舊版和目前版本之間的行為差異。

- 相較於 2017.11.29 版 (舊版)，2019.11.21 版 (目前版本) 進行數個 DynamoDB 操作時耗用的寫入容量較少，因此，對大多數客戶來說更具成本效益。這些 DynamoDB 操作的差異如下：
  - 針對區域中的 1KB 項目調用 PutItem 並複寫到其他區域時，2017.11.29 版 (舊版) 每個區域需要 2 個 rWRU，但 2019.11.21 版 (目前版本) 只需要 1 個 rWRU。
  - 針對 1KB 項目調用 UpdateItem 時，2017.11.29 版 (舊版) 來源區域需要 2 個 rWRU，且每個目的地區域需要 1 個 rWRU，但在 2019.11.21 版 (目前版本) 的來源和目的地區域都只需要 1 個 rWRU。

- 針對 1KB 項目調用 DeleteItem 時，2017.11.29 版 (舊版) 來源區域需要 1 個 rWRU，且每個目的地區域需要 2 個 rWRU，但在 2019.11.21 版 (目前版本) 的來源或目的地區域都只需要 1 個 rWRU。

下表列出針對兩個區域中 1KB 項目 2017.11.29 版 (舊版) 和 2019.11.21 版 (目前版本) 資料表耗用的 rWRU。

作業	2017.11.29 (舊版)	2019.11.21 (目前版本)	節省成本
PutItem	4 個 rWRU	2 個 rWRU	50%
UpdateItem	3 個 rWRU	2 個 rWRU	33%
DeleteItem	3 個 rWRU	2 個 rWRU	33%

- 2017.11.29 版 （舊版） 僅適用於 11 AWS 區域。不過，2019.11.21 版 (目前版本) 可在所有 AWS 區域中使用。
- 您可以先建立一組空白區域資料表，然後調用 CreateGlobalTable API 構成全域資料表，藉此建立 2017.11.29 版 (舊版) 全域資料表。您可以調用 UpdateTable API 將複本新增至現有區域資料表，藉此建立 2019.11.21 版 (目前版本) 全域資料表。
- 使用 2017.11.29 版 (舊版) 時，您必須先清空資料表中的所有複本，才能在新區域中新增複本 (包括建立期間)。若使用 2019.11.21 版 (目前版本)，即使資料表已包含資料，仍支援將複本新增至區域，或移除複本。
- 2017.11.29 版 (舊版) 使用下列專用控制平面 API 來管理複本：
  - CreateGlobalTable
  - DescribeGlobalTable
  - DescribeGlobalTableSettings
  - ListGlobalTables
  - UpdateGlobalTable
  - UpdateGlobalTableSettings

  2019.11.21 版 (目前版本) 使用 DescribeTable 和 UpdateTable API 來管理複本。
- 針對每個寫入，2017.11.29 版 (舊版) 會發布兩個 DynamoDB Streams 記錄；2019.11.21 版 (目前版本) 只會發布一個 DynamoDB Streams 記錄。

- 2017.11.29 版 (舊版) 會填入並更新 `aws:rep:deleting`、`aws:rep:updateregion` 和 `aws:rep:updatetime` 屬性。2019.11.21 版 (目前版本) 不會填入或更新這些屬性。
- 2017.11.29 版 (舊版) 不會跨複本同步 使用 DynamoDB 的存留時間 (TTL) 功能 設定。2019.11.21 版 (目前版本) 會跨複本同步 TTL 設定。
- 2017.11.29 版 (舊版) 不會將 TTL 刪除複寫至其他複本。2019.11.21 版 (目前版本) 會將 TTL 刪除複寫至所有複本。
- 2017.11.29 版 (舊版) 不會跨複本同步自動擴展設定。2019.11.21 版 (目前版本) 會跨複本同步自動擴展設定。
- 2017.11.29 版 (舊版) 不會跨複本同步全域次要索引 (GSI) 設定。2019.11.21 版 (目前版本) 會跨複本同步 GSI 設定。
- 2017.11.29 版 (舊版) 不會跨複本同步靜態加密設定。2019.11.21 版 (目前版本)會跨複本同步靜態加密設定。
- 2017.11.29 版 (舊版) 發布 `PendingReplicationCount` 指標。2019.11.21 版 (目前版本) 不會發布此指標。

升級到目前版本

全域資料表升級所需的許可

若要升級至 2019.11.21 版 (目前版本)，您必須擁有所有複本區域的 `dynamodb:UpdateGlobalTableversion` 許可。除了存取 DynamoDB 主控台和檢視資料表的必要許可，您還需要這些許可。

下列 IAM 政策會授予許可，將任何全域資料表更新至 2019.11.21 版 (目前版本)。

```
{
    "version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": "*"
        }
    ]
}
```

下列 IAM 政策會授予許可，只將 `Music` 全域資料表 (在兩個區域中具有複本) 升級為 2019.11.21 版 (目前版本)。

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

升級期間預期會發生的情況

- 升級時，所有全域資料表複本會繼續處理讀取和寫入流量。
- 升級程序需要幾分鐘到幾個小時，取決於資料表大小和複本數量而定。
- 在升級過程中，TableStatus 的值會從 ACTIVE 變更為 UPDATING。您可以透過叫用 DescribeTable API 或使用 DynamoDB 主控台中的資料表檢視來檢視資料表的狀態。
- 在升級資料表時，自動擴展不會調整全域資料表的佈建容量設定。強烈建議您在升級期間將資料表設為隨需容量模式。
- 如果您選擇在升級期間使用佈建容量搭配自動擴展，則必須提高政策的讀取和寫入輸送量下限，以容納升級期間預期增加的流量，以免限流情況發生。
- 在升級過程中，ReplicationLatency 指標可以暫時回報延遲峰值，或停止回報指標資料。如需詳細資訊，請參閱 the section called “ReplicationLatency”。
- 升級程序完成時，資料表狀態會變更為 ACTIVE。

升級前、中、後的 DynamoDB Streams 行為

作業	複本區域	升級前的行為	升級期間的行為	升級後的行為
Put 或 Update	來源	使用 UpdateItem 時會發生時間戳記填入。	使用 PutItem 時會發生時間戳記填入。	不會產生客戶可見的時間戳記。

作業	複本區域	升級前的行為	升級期間的行為	升級後的行為
		會產生兩個 Streams 記錄。第一個記錄包含客戶撰寫的屬性。第二個記錄包含 aws:rep:* 屬性。	會產生兩個 Streams 記錄。第一個記錄包含客戶撰寫的屬性。第二個記錄包含 aws:rep:* 屬性。	會產生單一 Streams 記錄，其中包含客戶撰寫的屬性。
		每個客戶寫入會耗用兩個 rWCU。	每個客戶寫入會耗用兩個 rWCU。	每個客戶寫入會耗用一個 rWCU。
		ReplicationLatency 和 PendingReplicationCount 指標會發布至 CloudWatch。	ReplicationLatency 和 PendingReplicationCount 指標會發布至 CloudWatch。	ReplicationLatency 指標會發布至 CloudWatch。
	目的地	複寫會使用 PutItem 進行。	複寫會使用 PutItem 進行。	複寫會使用 PutItem 進行。
		會產生單一 Streams 記錄，其中包含客戶撰寫的屬性和 aws:rep:* 屬性。	會產生單一 Streams 記錄，其中包含客戶撰寫的屬性和 aws:rep:* 屬性。	會產生單一 Streams 記錄，其中僅包含客戶撰寫的屬性，而沒有複寫屬性。

作業	複本區域	升級前的行為	升級期間的行為	升級後的行為
		如果項目存在於目的地區域中，會耗用一個 rWCU。如果項目不存在於目的地區域中，則會耗用兩個 rWCUs。	如果項目存在於目的地區域中，會耗用一個 rWCU。如果項目不存在於目的地區域中，則會耗用兩個 rWCUs。	每個客戶寫入會耗用一個 rWCU。
		`ReplicationLatency` 和 `PendingReplicationCount` 指標會發布至 CloudWatch。	`ReplicationLatency` 和 `PendingReplicationCount` 指標會發布至 CloudWatch。	`ReplicationLatency` 指標會發布至 CloudWatch。
刪除	來源	使用 DeleteItem 刪除時間戳記較小的任何項目。	使用 DeleteItem 刪除時間戳記較小的任何項目。	使用 DeleteItem 刪除時間戳記較小的任何項目。
		會產生單一 Streams 記錄，其中包含客戶撰寫的屬性和 `aws:rep:*` 屬性。	會產生單一 Streams 記錄，其中包含客戶撰寫的屬性和 `aws:rep:*` 屬性。	會產生單一 Streams 記錄，其中包含客戶撰寫的屬性。
		每個客戶刪除會耗用一個 rWCU。	每個客戶刪除會耗用一個 rWCU。	每個客戶刪除會耗用一個 rWCU。

作業	複本區域	升級前的行為	升級期間的行為	升級後的行為
		ReplicationLatency 和 PendingReplicationCount 指標會發布至 CloudWatch。	ReplicationLatency 和 PendingReplicationCount 指標會發布至 CloudWatch。	ReplicationLatency 指標會發布至 CloudWatch。
	目的地	刪除按下列兩個階段進行： • 在階段 1 中，UpdateItem 會設定刪除旗標。 • 在階段 2 中，DeleteItem 會刪除項目。	使用 DeleteItem 刪除項目。	使用 DeleteItem 刪除項目。
		會產生兩個 Streams 記錄。第一個記錄包含 aws:rep:deleting 欄位的變更。第二個記錄包含客戶撰寫的屬性和 aws:rep:* 屬性。	會產生單一 Stream 記錄，其中包含客戶撰寫的屬性。	會產生單一 Stream 記錄，其中包含客戶撰寫的屬性。
		每個客戶刪除會耗用兩個 rWCUs。	每個客戶刪除會耗用一個 rWCU。	每個客戶刪除會耗用一個 rWCU。

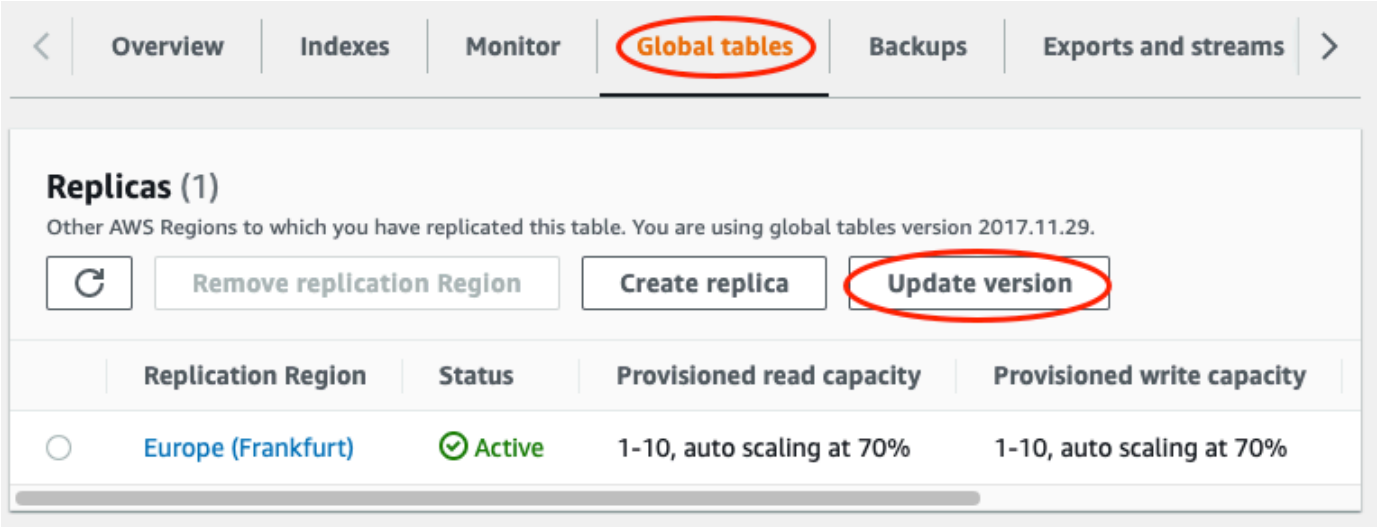
作業	複本區域	升級前的行為	升級期間的行為	升級後的行為
		ReplicationLatency 和 PendingReplicationCount 指標會發布至 CloudWatch。	ReplicationLatency 指標會發布至 CloudWatch。	ReplicationLatency 指標會發布至 CloudWatch。

升級到 2019.11.21 版(目前版本)

請執行下列步驟，使用 AWS 管理主控台升級 DynamoDB 全域資料表版本。

將全域資料表升級至 2019.11.21 版 (目前版本)

1. 在 https://console.aws.amazon.com/dynamodb/home 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇資料表，然後選取您要升級至 2019.11.21 版 (目前版本) 的全域資料表。
3. 選擇 Global Tables (全域資料表) 標籤。
4. 選擇 Update version (更新版本)。



5. 閱讀並同意新的要求，然後選擇 Update version (更新版本)。
6. 升級程序完成之後，主控台上出現的全域資料表版本會變更為 2019.11.21。

全域資料表的最佳實務

下列各節說明部署和使用全域資料表的最佳實務。

版本

DynamoDB 全域資料表有兩種版本可用：2019.11.21 版 (目前版本) 和 2017.11.29 版 (舊版)。您應該盡可能使用 2019.11.21 版 (目前版本)。

刪除保護

若要避免意外刪除全域資料表複本，您應該為其啟用刪除保護。您必須為每個複本啟用刪除保護。

使用 AWS CloudFormation

CloudFormation 目前不支援跨堆疊協調多區域資源，例如全域資料表。如果您在個別區域堆疊中定義全域資料表的每個複本，則在執行複本更新時會因偵測到跨堆疊的偏離而發生錯誤。若要避免此問題，您應該選擇一個區域作為部署全域資料表的參考區域，並在該區域堆疊中定義所有全域資料表複本。

> **Important**
>
> 您無法透過在模板中變更資源類型，將類型為 `AWS::DynamoDB::Table` 的資源轉換為類型為 `AWS::DynamoDB::GlobalTable` 的資源。若您變更單一區域資料表的 CloudFormation 資源類型，試圖將其轉換為全域資料表，可能會導致 DynamoDB 資料表遭到刪除。

您可以使用 `AWS::DynamoDB::GlobalTable` 資源，在單一區域中建立資料表。此資料表部署方式與其他單一區域資料表類似。如果您稍後更新堆疊，將其他區域新增至資源，即可將複本新增至資料表，並安全地將其轉換為全域資料表。

如果您的現有 `AWS::DynamoDB::Table` 資源要轉換為 `AWS::DynamoDB::GlobalTable` 資源，建議的資源類型轉換步驟如下：

1. 設定要保留的 `AWS::DynamoDB::Table` 刪除政策。
2. 從堆疊定義中移除資料表。
3. 將複本新增至 AWS 主控台中的單一區域資料表，並將其轉換為全域資料表。
4. 將新的全域資料表作為新 `AWS::DynamoDB::GlobalTable` 資源匯入堆疊。

備份和時間點復原

為全域資料表中的一個複本，啟用自動備份和時間點復原 (PITR)，可能得以滿足您的災難復原目標。使用 AWS備份建立的複本備份可以跨區域自動複寫，以獲得更高的彈性。在選擇備份和 PITR 啟用策略時，請以多區域高可用性環境來考量災難復原計劃目標。

多區域高可用性的設計

如需部署全域資料表的方案指引，請參閱 DynamoDB 全域資料表設計的最佳實務。

在 DynamoDB 使用項目和屬性

在 Amazon DynamoDB 中，項目是屬性的集合。每個屬性都有名稱和數值。屬性值可以是純量、集合，或文件類型。如需詳細資訊，請參閱Amazon DynamoDB：運作方式。

DynamoDB 提供四種用於基本建立、讀取、更新和刪除 (CRUD) 功能的操作：所有這些操作都是不能中斷的。

- `PutItem`：建立項目。
- `GetItem`：閱讀項目。
- `UpdateItem`：更新項目。
- `DeleteItem`：刪除項目。

這些操作每一項都需要您指定希望處理之項目的主索引鍵。例如，若要使用 `GetItem` 讀取項目，您必須指定該項目的分割區索引鍵和排序索引鍵 (若適用的話)。

此外，除了四種基本 CRUD 操作之外，DynamoDB 也提供了下列項目：

- `BatchGetItem`：最多可從一或多個資料表讀取 100 個項目。
- `BatchWriteItem`：在一或多個資料表中最多可建立或刪除 25 個項目。

這些批次操作可將多個 CRUD 操作合併為單一請求。此外，批次操作會平行讀取和寫入項目，將回應延遲減至最低。

本節說明如何使用這些操作，並包含相關主題，例如條件式更新和原子計數器。本節也包含使用 AWS SDKs的範例程式碼。

主題

- DynamoDB 項目大小和格式
- 讀取項目
- 寫入項目
- 傳回值
- 批次操作
- 原子計數器
- 條件式寫入
- 在 DynamoDB 中使用表達式
- 使用 DynamoDB 的存留時間 (TTL) 功能
- 在 DynamoDB 中查詢資料表
- 掃描 DynamoDB 中的資料表
- PartiQL：一種適用於 Amazon DynamoDB 的 SQL 相容查詢語言
- 使用項目：Java
- 處理項目：.NET

DynamoDB 項目大小和格式

除了主索引鍵之外，DynamoDB 資料表不具結構描述，因此資料表中的項目全部會有不同的屬性、大小和資料類型。

項目的總大小是其屬性名稱和值長度的總和，加上下列任何適用的管理費用。您可以使用下列準則來預估屬性大小：

- 字串是 UTF-8 二進位編碼的 Unicode。字串大小是 (屬性名稱的 UTF-8 編碼位元組數目) + (UTF-8 編碼位元組數目)。
- 數字的長度會不同，最多 38 個有意義位數。前後的零會截去。數字大小大約是 (屬性名稱的 UTF-8 編碼位元組數目) + (每兩個有效位數 1 位元組) + (1 位元組)。
- 必須先以 base64 格式編碼二進位值，才能將它傳送至 DynamoDB，但使用值的原始位元組長度來計算大小。二進制屬性大小是 (屬性名稱的 UTF-8 編碼位元組數目) + (原始位元組數目)。
- Null 屬性或布林值屬性的大小是 (屬性名稱的 UTF-8 編碼位元組數目) + (1 位元組)。
- 不論內容為何，類型為 `List` 或 `Map` 的屬性都需要 3 位元組的額外負荷。`List` 或 `Map` 的大小是 (屬性名稱的 UTF-8 編碼位元組數目) + 總和 (巢狀元素的大小) + (3 位元組)。`List` 或 `Map` 的大小是 (屬性名稱的 UTF-8 編碼位元組數目) + (3 位元組)。

- 每個 List 或 Map 元素也需要 1 位元組的額外負荷。

> **Note**
>
> 建議您選擇較短的屬性名稱，而不是較長的屬性名稱。如此有助您減少所需的儲存量，同時降低您的 RCU/WCU 用量。

基於儲存體計費目的，每個項目會包含每個項目的儲存體額外負荷，這取決於您啟用的功能。

- DynamoDB 中的所有項目都需要 100 個位元組的儲存額外負荷來進行索引。
- 某些 DynamoDB 功能 (使用 DynamoDB 進行 Kinesis Data Streams 的全域資料表、交易、變更資料擷取) 需要額外的儲存額外負荷來考量啟用這些功能所產生的系統建立屬性。例如，全域資料表需要額外 48 個位元組的儲存體額外負荷。

讀取項目

若要從 DynamoDB 資料表讀取項目，請使用 GetItem 操作。您必須提供資料表的名稱，以及您希望取得之項目的主索引鍵。

Example

下列 AWS CLI 範例示範如何從 ProductCatalog 資料表讀取項目。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}'
```

> **Note**
>
> 使用 GetItem，您必須指定整個主索引鍵，而非其中一部分。例如，若資料表有複合主鍵 (分割區索引鍵和排序索引鍵)，您必須提供分割區索引鍵的數值和排序索引鍵的數值。

根據預設，GetItem 請求會執行最終一致讀取。您可以使用 ConsistentRead 參數改為請求高度一致性讀取。(這會使用額外的讀取容量單位，但會傳回項目的最新版本。)

GetItem 會傳回項目所有的屬性。您可以使用投影表達式只傳回一部分的屬性。如需詳細資訊，請參閱 在 DynamoDB 中使用投影表達式。

若要傳回 GetItem 使用的讀取容量單位總數，請將 ReturnConsumedCapacity 參數設為 TOTAL。

Example

下列 AWS Command Line Interface (AWS CLI) 範例顯示一些選用GetItem參數。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --consistent-read \
    --projection-expression "Description, Price, RelatedItems" \
    --return-consumed-capacity TOTAL
```

寫入項目

若要在 DynamoDB 資料表中建立、更新或刪除項目，請使用下列其中一項操作：

- PutItem
- UpdateItem
- DeleteItem

針對每一項操作，您都必須指定整個主索引鍵，非僅指定一部分。例如，若資料表有複合主鍵 (分割區索引鍵和排序索引鍵)，您必須提供分割區索引鍵的值和排序索引鍵的值。

若要傳回這些操作使用的寫入容量單位，請將 ReturnConsumedCapacity 參數設為下列其中一項：

- TOTAL：傳回所耗用的寫入容量單位總數。
- INDEXES：傳回所耗用的寫入容量單位總數，以及資料表的小計和受操作影響的任何次要索引。
- NONE：不傳回寫入容量的詳細資訊。(此為預設值)。

PutItem

PutItem 會建立新的項目。若資料表中已存在具有相同索引鍵的項目，該項目會取代為新的項目。

Example

將新的項目寫入 Thread 表。Thread 的主索引鍵包含 ForumName (分割區索引鍵) 和 Subject (排序索引鍵)。

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

--item 的引數會存放在 item.json 檔案中。

```
{
    "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

若不存在具有指定之索引鍵的項目，UpdateItem 會建立新的項目。否則，它會修改現有項目的屬性。

您可以使用更新表達式指定您希望修改的屬性及其新值。如需詳細資訊，請參閱 在 DynamoDB 中使用更新表達式。

在更新表達式中，您會使用表達式屬性值做為實際數值的預留位置。如需詳細資訊，請參閱 在 DynamoDB 中使用表達式屬性值。

Example

修改 Thread 項目中的各種屬性。選用的 ReturnValues 參數會在項目更新之後顯示更新後的項目。如需詳細資訊，請參閱 傳回值。

```
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
```

--key 的引數會存放在 key.json 檔案中。

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"}
}
```

--expression-attribute-values 的引數會存放在 expression-attribute-values.json 檔案中。

```
{
    ":zero": {"N":"0"},
    ":lastpostedby": {"S":"barney@example.com"}
}
```

DeleteItem

DeleteItem 會刪除具有指定之索引鍵的項目。

Example

下列 AWS CLI 範例顯示如何刪除Thread項目。

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

傳回值

在某些案例中，您可能會希望 DynamoDB 傳回修改前或修改後的特定屬性值。PutItem、UpdateItem 和 DeleteItem 操作具有 ReturnValues 參數，您可以使用此參數傳回修改前或修改後的屬性值。

ReturnValues 的預設值為 NONE，表示 DynamoDB 不會傳回任何已修改的屬性資訊。

以下為 ReturnValues 的其他有效設定，依 DynamoDB API 操作整理。

PutItem

- ReturnValues: ALL_OLD

- 若您覆寫現有項目，ALL_OLD 會傳回覆寫前的整個項目。
- 若您寫入原先不存在的項目，ALL_OLD 將不具任何效果。

UpdateItem

UpdateItem 最常見的用法便是更新現有的項目。但是，UpdateItem 實際上執行的是 upsert，表示若項目尚未存在，它會自動建立項目。

- ReturnValues: ALL_OLD
  - 若您更新現有項目，ALL_OLD 會傳回更新前的整個項目。
  - 若您更新原先不存在的項目 (upsert)，ALL_OLD 將不具任何效果。
- ReturnValues: ALL_NEW
  - 若您更新現有項目，ALL_NEW 會傳回更新後的整個項目。
  - 若您更新原先不存在的項目 (upsert)，ALL_NEW 會傳回整個項目。
- ReturnValues: UPDATED_OLD
  - 若您更新現有項目，UPDATED_OLD 只會傳回更新後的屬性在更新前的樣子。
  - 若您更新原先不存在的項目 (upsert)，UPDATED_OLD 將不具任何效果。
- ReturnValues: UPDATED_NEW
  - 若您更新現有項目，UPDATED_NEW 只會傳回受影響的屬性在更新後的樣子。
  - 若您更新原先不存在的項目 (upsert)，UPDATED_NEW 只會傳回更新後的屬性在更新後的樣子。

DeleteItem

- ReturnValues: ALL_OLD
  - 若您刪除現有項目，ALL_OLD 會傳回您刪除該項目前的整個項目。
  - 若您刪除原先不存在的項目，ALL_OLD 不會傳回任何資料。

批次操作

針對需要讀取或寫入多個項目的應用程式，DynamoDB 提供 BatchGetItem 和 BatchWriteItem 操作。使用這些操作可減少您應用程式與 DynamoDB 之間網路來回行程的次數。此外，DynamoDB 會平行執行個別讀取或寫入操作。您的應用程式可從此平行處理原則中獲益，而無須管理並行或執行緒。

批次操作本質上是多個讀取或寫入請求的包裝函式。例如，若 BatchGetItem 請求包含五個項目，DynamoDB 就會代您執行五個 GetItem 操作。同樣的，若 BatchWriteItem 請求包含兩個 PUT 請求和四個刪除請求，則 DynamoDB 就會執行兩個 PutItem 請求和四個 DeleteItem 請求。

一般而言，除非批次中所有的請求皆失敗，否則批次操作不會失敗。例如，假設您執行 BatchGetItem 操作，但批次中一個個別的 GetItem 請求失敗。在此案例中，BatchGetItem 會傳回失敗 GetItem 請求的索引鍵和資料。其他批次中的 GetItem 請求則不會受到影響。

BatchGetItem

單一 BatchGetItem 操作可包含最多 100 個個別 GetItem 請求，並可擷取最多 16 MB 的資料。此外，BatchGetItem 操作可從多個資料表擷取項目。

Example

從 Thread 表擷取兩個項目，使用投影表達式卻只傳回一部分的屬性。

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json
```

--request-items 的引數會存放在 request-items.json 檔案中。

```
{
    "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

BatchWriteItem 操作可包含最多 25 個個別 PutItem 和 DeleteItem 請求，並可寫入最多 16 MB 的資料。(個別項目的大小上限為 400 KB。) 此外，BatchWriteItem 操作可在多個資料表中寫入或刪除項目。

> **Note**
>
> BatchWriteItem 不支援 UpdateItem 請求。

Example

將兩個項目寫入 ProductCatalog 表。

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json
```

--request-items 的引數會存放在 request-items.json 檔案中。

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "601" },
                    "Description": { "S": "Snowboard" },
                    "QuantityOnHand": { "N": "5" },
                    "Price": { "N": "100" }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "602" },
                    "Description": { "S": "Snow shovel" }
                }
            }
        }
    ]
```

```
}
```

原子計數器

您可以使用 UpdateItem 操作實作原子計數器：原子計數器為在不影響其他寫入請求的情況下，無條件遞增的數字屬性。(所有寫入請求都會按照接收的順序套用)。使用原子計數器，更新便不是等冪的。換言之，每次呼叫 UpdateItem 時，數值就會增加或減少。如果用於更新原子計數器的增量值是正的，那麼它可能會導致多計。如果增量值為負數，則可能會導致少計。

您可以使用原子計數器追蹤網站的訪客數。在此案例中，您的應用程式會增加數值，無論目前的數值為何。若 UpdateItem 操作失敗，應用程式可能只會重試操作。這可能會導致計數器更新兩次，但您或許可以容忍計數器些微多計或少計網站的訪客數。

原子計數器不適用於無法容忍多計或少計的情況 (例如銀行的應用程式)。在此案例中，使用條件式的更新而非原子計數器會更安全。

如需詳細資訊，請參閱增加和減少數值屬性。

Example

下列 AWS CLI 範例會將Price產品的 增加 5。在此範例中，在更新計數器之前已知該項目存在。因為 UpdateItem 並非等冪，Price 會在每次執行此程式碼時增加。

```
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
```

條件式寫入

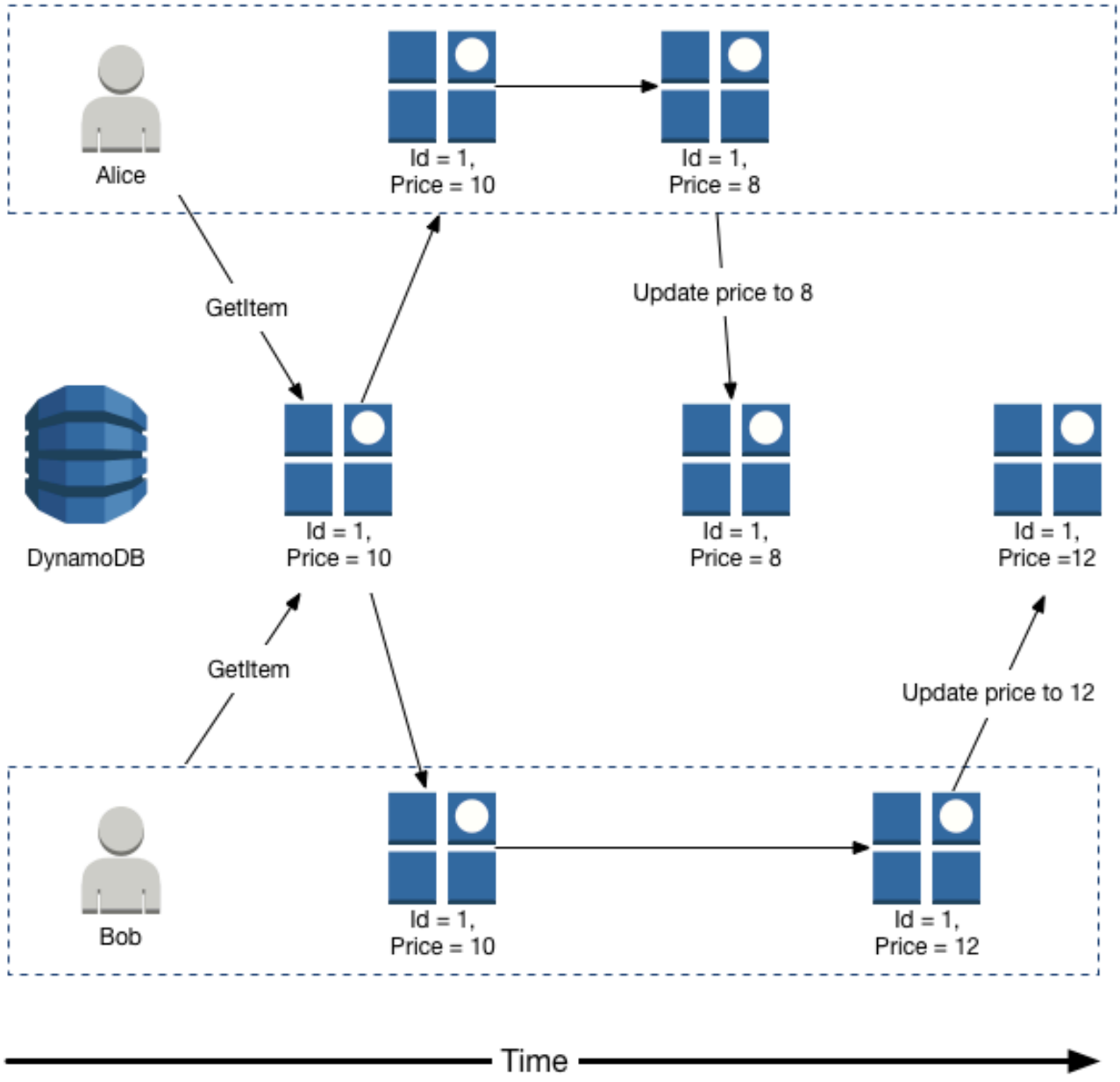
根據預設，DynamoDB 寫入操作 (PutItem、DeleteItem) 是無條件的：每個操作都會覆寫具有指定主索引鍵的現有項目。

DynamoDB 可選擇性的支援這些操作的條件式寫入。條件式寫入只有在項目屬性滿足一或多個預期條件時才會成功。否則會傳回錯誤。

條件式寫入會根據項目的最新更新版本來檢查其條件。請注意，如果項目先前不存在，或者對該項目最近成功的操作是刪除，則條件式寫入將找不到先前的項目。

條件式寫入在許多情況下都很有幫助。例如，您可能會希望 `PutItem` 操作僅在沒有具有相同主索引鍵的項目時才成功。或者，您可以防止 `UpdateItem` 操作修改其中一個屬性為特定數值的項目。

條件式寫入在多個使用者嘗試修改相同項目的案例中會很有幫助。考慮下圖，其中兩名使用者 (Alice 和 Bob) 正在處理 DynamoDB 表中同一個項目。



假設 Alice 使用 AWS CLI 將 `Price` 屬性更新為 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
```

`--expression-attribute-values` 的引數會存放在 `expression-attribute-values.json` 檔案中：

```
{
    ":newval":{"N":"8"}
}
```

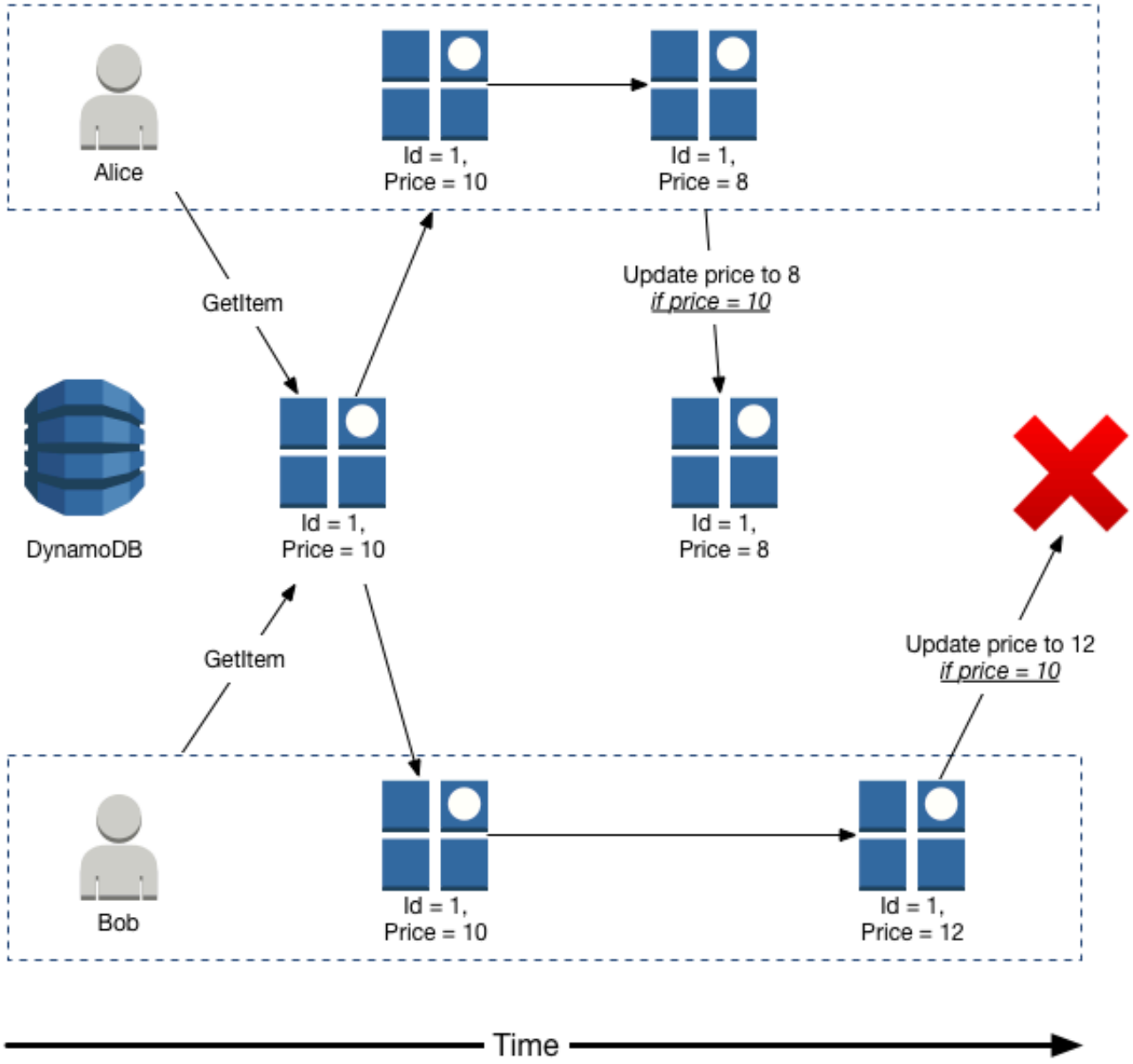
現在假設 Bob 稍後發行一個相似的 `UpdateItem` 請求，但將 `Price` 變更為 12。對 Bob 而言，`--expression-attribute-values` 參數看起來如下。

```
{
    ":newval":{"N":"12"}
}
```

Bob 的請求會成功，但 Alice 先前做出的更新便會遺失。

若要請求條件式 `PutItem`、`DeleteItem` 或 `UpdateItem`，您可以指定條件表達式。條件表達式為包含屬性名稱、條件運算子和內建函數的字串。整個表達式都必須評估為 true。否則，操作會失敗。

現在考慮下圖，示範條件式寫入如何阻擋覆寫 Alice 的更新。



Alice 首先會嘗試將 `Price` 更新為 8，但只有在目前的 `Price` 為 10 時才可以。

```
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
```

`--expression-attribute-values` 的引數會存放在 `expression-attribute-values.json` 檔案中。

```
{
    ":newval":{"N":"8"},
    ":currval":{"N":"10"}
}
```

Alice 的更新會成功，因為條件評估的結果為 true。

接下來，Bob 會嘗試將 `Price` 更新為 12，但目前的 `Price` 必須為 10 才能成功。對 Bob 而言，`--expression-attribute-values` 參數看起來如下。

```
{
    ":newval":{"N":"12"},
    ":currval":{"N":"10"}
}
```

因為 Alice 先前已將 `Price` 更新為 8，條件表達式評估的結果為 false，因此 Bob 的更新會失敗。

如需詳細資訊，請參閱 DynamoDB 條件表達式 CLI 範例。

條件式寫入冪等性

如果針對正受到更新的相同屬性進行條件式檢查，則條件式寫入可能為等冪。此表示僅在項目中的某個屬性值符合您在要求時所預期的值，DynamoDB 才會執行特定的寫入要求。

例如，假設您發出 `UpdateItem` 請求將項目的 `Price` 增加 3，但只有在目前的 `Price` 為 20 時才可以。在傳送請求後、取得結果前，發生網路錯誤，因此您不知道請求是否成功。因為此條件式寫入是等冪的，所以您可以重試相同的 `UpdateItem` 請求，且 DynamoDB 只有在目前的 `Price` 為 20 時才會更新項目。

條件式寫入使用的容量單位

若在條件式寫入期間，`ConditionExpression` 評估的結果為 false，DynamoDB 仍然會使用資料表的寫入容量：使用的量取決於現有項目的大小 (或最少為 1)。例如，如果現有項目為 300kb，而您嘗試建立或更新的新項目為 310kb，那麼條件失敗時，使用的寫入容量單位將是 300，條件成功時則會是

310。如果這是新項目 (沒有現有項目)，那麼條件失敗時，使用的寫入容量單位將是 1，條件成功時則會是 310。

> **Note**
>
> 寫入操作只會使用寫入容量單位。它們永遠不會使用讀取容量單位。

失敗的條件式寫入會傳回 `ConditionalCheckFailedException`。發生此狀況時，您不會在回應中收到任何有關使用的寫入容量資訊。

若要傳回條件式寫入期間使用的寫入容量單位數，請使用 `ReturnConsumedCapacity` 參數：

- `TOTAL`：傳回所耗用的寫入容量單位總數。
- `INDEXES`：傳回所耗用的寫入容量單位總數，以及資料表的小計和受操作影響的任何次要索引。
- `NONE`：不傳回寫入容量的詳細資訊。(此為預設值)。

> **Note**
>
> 與全域次要索引不同，本機次要索引會與其資料表共用佈建的輸送容量。本機次要索引上的讀取和寫入活動會使用其資料表佈建的輸送容量。

在 DynamoDB 中使用表達式

在 Amazon DynamoDB 中，您可以使用表達式指定要從項目讀取的屬性、在滿足條件時寫入資料、指定如何更新項目、定義查詢，以及篩選查詢的結果。

本資料表說明基本表達式語法及可用的表達式類型。

表達式類型	Description
投射表達式	使用 GetItem、Query 或 Scan 等操作時，投影表達式會識別您要從項目擷取的屬性。
條件表達式	使用 PutItem、UpdateItem 和 DeleteItem 等操作時，條件表達式會判定應修改的項目。

表達式類型	Description
更新表達式	更新表達式會指定 UpdateItem 如何修改項目的屬性，例如設定純量值或將元素從清單或映射中刪除。
索引鍵條件表達式	索引鍵條件表達式會判定查詢要讀取哪些資料表或索引的項目。
篩選條件表達式	篩選條件表達式會判定要傳回哪些 Query 結果的項目。所有其他結果皆會捨棄。

如需表達式語法的相關資訊，以及每種表達式類型的詳細資訊，請參閱下列各節。

主題

- 在 DynamoDB 中使用表達式時參考項目屬性
- DynamoDB 中的表達式屬性名稱 (別名)
- 在 DynamoDB 中使用表達式屬性值
- 在 DynamoDB 中使用投影表達式
- 在 DynamoDB 中使用更新表達式
- DynamoDB 中的條件表達式、篩選條件表達式、運算子和函數。
- DynamoDB 條件表達式 CLI 範例

> **Note**
>
> 為了與舊版相容，DynamoDB 還支援不使用表達式的條件式參數。如需詳細資訊，請參閱 舊版 DynamoDB 條件式參數。
> 新的應用程式應該使用表達式，而不是舊版參數。

在 DynamoDB 中使用表達式時參考項目屬性

本節說明如何在 Amazon DynamoDB 中的表達式內參考項目屬性。您可以使用任何屬性，即使該位於多個清單和映射的深層巢狀結構中也一樣。

主題

- 最上層屬性
- 巢狀屬性
- 文件路徑

範例項目：ProductCatalog

本頁面的範例使用 ProductCatalog 資料表的下列範例項目。(此資料表會在 在 DynamoDB 中使用的資料表和資料範例 中說明。)

```
{
    "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"
```

```
}
```

注意下列事項：

- 分割區索引鍵值 (Id) 為 123。沒有排序索引鍵。
- 大多數的屬性都有純量資料類型，例如 String、Number、Boolean 和 Null。
- 一個屬性 (Color) 為 String Set。
- 下列屬性為文件資料類型：
  - RelatedItems 的清單。每個元素是相關產品的 Id。
  - Pictures 的映射。每個元素是圖片的簡短說明，以及對應影像檔的 URL。
  - ProductReviews 的映射。每個元素都代表一個評分及對應至該評分的評論清單。一開始，此映射會填入五星與一星評論。

最上層屬性

如果屬性未內嵌於另一個屬性，則為最上層。針對 ProductCatalog 項目，最上層屬性包括：

- Id
- Title
- Description
- BicycleType
- Brand
- Price
- Color
- ProductCategory
- InStock
- QuantityOnHand
- RelatedItems
- Pictures
- ProductReviews
- Comment
- Safety.Warning

除了 Color (清單)、RelatedItems (清單)、Pictures (映射) 與 ProductReviews (映射) 之外，上述所有最上層屬性都是純量。

巢狀屬性

如果屬性內嵌於另一個屬性，則為巢狀。若要存取巢狀屬性，您可以使用取值運算子：

- [n]：用於清單元素
- . (點)：用於映射元素

存取清單元素

清單元素的取消參考運算子為 [N]，其中 n 是元素編號。清單元素的開頭為零，所以 [0] 代表清單中的第一個元素，[1] 代表第二個，以此類推。以下是一些範例：

- MyList[0]
- AnotherList[12]
- ThisList[5][11]

元素 ThisList[5] 本身是巢狀清單。因此，ThisList[5][11] 是指該清單中的第 12 個元素。

方括號內的數字必須是非負整數。因此，下列表達式無效：

- MyList[-1]
- MyList[0.4]

存取映射元素

映射元素的取消參考運算子為 . (點號)。請在映射中的元素之間使用點號做為分隔符號：

- MyMap.nestedField
- MyMap.nestedField.deeplyNestedField

文件路徑

在表達式中，您可以使用文件路徑指示 DynamoDB 要在何處尋找屬性。針對最上層屬性，文件路徑就是屬性名稱。針對巢狀屬性，您可以使用取消參考運算子來建構文件路徑。

以下是文件路徑的一些範例。(請參閱 在 DynamoDB 中使用表達式時參考項目屬性 中所示的項目。)

- 最上層純量屬性。

  `Description`

- 最上層清單屬性。(這會傳回整個清單，而不只是部分元素。)

  `RelatedItems`

- `RelatedItems` 清單中的第三個元素 (記得清單元素的開頭為零)。

  `RelatedItems[2]`

- 產品的正視圖。

  `Pictures.FrontView`

- 所有五星評論。

  `ProductReviews.FiveStar`

- 第一個五星評論。

  `ProductReviews.FiveStar[0]`

> **Note**
>
> 文件路徑的最大深度為 32。因此，路徑中的取消參考運算子數目不得超過此限制。

您可以在文件路徑中使用任何屬性名稱，只要名稱符合下列要求：

- 第一個字元為 `a-z`、`A-Z` 或 `0-9`
- 第二個字符 (如果有) 為 `a-z` 或 `A-Z`

> **Note**
>
> 若屬性名稱未符合此需求，您必須將表達式屬性名稱定義為預留位置。

如需詳細資訊，請參閱DynamoDB 中的表達式屬性名稱 (別名)。

DynamoDB 中的表達式屬性名稱 (別名)

表達式屬性名稱是您在 Amazon DynamoDB 表達式中所使用的別名 (或預留位置)，其為實際屬性名稱的替代項目。表達式屬性名稱的開頭必須是井字號 (#)，其後接著一或多個英數字元。也允許使用底線 (_) 字元。

本節說明必須使用表達式屬性名稱的數種狀況。

> **Note**
>
> 本節中的範例使用 AWS Command Line Interface (AWS CLI)。

主題

- 保留字
- 包含特殊字元的屬性名稱
- 巢狀屬性
- 重複參考屬性名稱

保留字

有時候您需要撰寫的表達式可能會包含與 DynamoDB 保留字衝突的屬性名稱。(如需完整的保留字清單，請參閱DynamoDB 中的保留字。)

例如，下列 AWS CLI 範例會失敗，因為 COMMENT 是保留字。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Comment"
```

若要解決此情況，您可以將 Comment 取代為表達式屬性名稱 (例如 #c)。# (井字號) 是必要項目，指出這是屬性名稱的預留位置。此 AWS CLI 範例現在看起來如下。

```
aws dynamodb get-item \
     --table-name ProductCatalog \
     --key '{"Id":{"N":"123"}}' \
     --projection-expression "#c" \
```

```
    --expression-attribute-names '{"#c":"Comment"}'
```

> **Note**
>
> 若屬性名稱的開頭是數字、包含空格或包含保留字，您必須使用表達式屬性名稱來取代表達式中的該屬性名稱。

包含特殊字元的屬性名稱

在表達式中，點號 (".") 會被解譯為文件路徑中的分隔符號字元。不過，DynamoDB 允許您使用點字元和其他特殊字元，例如連字號 (「-」) 作為屬性名稱的一部分。這在某些情況下可能會模棱兩可。為了示範，假設您想要從 `ProductCatalog` 項目擷取 `Safety.Warning` 屬性 (請參閱[在 DynamoDB 中使用表達式時參考項目屬性])。

假設您想要使用投射表達式來存取 `Safety.Warning`。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Safety.Warning"
```

DynamoDB 將會傳回空的結果，而不是預期字串 (`Always wear a helmet`)。原因是 DynamoDB 會將表達式中的點解譯為文件路徑分隔符號。在此情況下，您必須定義表達式屬性名稱 (例如 `#sw`) 做為 `Safety.Warning` 的替代項目。您接著可以使用下列投射表達式。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#sw" \
    --expression-attribute-names '{"#sw":"Safety.Warning"}'
```

DynamoDB 接著會傳回正確結果。

> **Note**
>
> 若屬性名稱包含點 (「.」) 或連字號 (「-」)，您必須使用表達式屬性名稱來取代表達式中的該屬性名稱。

巢狀屬性

假設您想要存取巢狀屬性 ProductReviews.OneStar。在表達式屬性名稱中，DynamoDB 會將點 (".") 視為屬性名稱內的字元。若要參考巢狀屬性，請為文件路徑中的每個元素定義表達式屬性名稱：

- #pr – ProductReviews
- #1star – OneStar

您接著可以使用 #pr.#1star 做為投射表達式。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.#1star"  \
    --expression-attribute-names '{"#pr":"ProductReviews", "#1star":"OneStar"}'
```

DynamoDB 接著會傳回正確結果。

重複參考屬性名稱

當您需要反覆參考相同的屬性名稱時，表達式屬性名稱十分有用。例如，請考慮以下表達式，從 ProductCatalog 項目擷取部分檢閱。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "ProductReviews.FiveStar, ProductReviews.ThreeStar,
 ProductReviews.OneStar"
```

更精確地來說，您可以將 ProductReviews 取代為表達式屬性名稱 (例如 #pr)。修訂過的表達式如下所示。

- #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"}'
```

如果您定義表達式屬性名稱，則必須在整個表達式中一致地使用它。您也無法省略 # 符號。

在 DynamoDB 中使用表達式屬性值

Amazon DynamoDB 中，表達式屬性值做為變數。它們是您要比較之實際值的替代項目，在執行時期之前您可能不知道這些值為何。表達式屬性值的開頭必須是冒號 (:)，且後面跟隨一或多個英數字元。

例如，假設您想要傳回提供 Black 顏色且價格不超過 500 (含) 的所有可用 ProductCatalog 項目。您可以搭配篩選條件表達式使用 Scan 操作，如此 AWS Command Line Interface (AWS CLI) 範例所示。

```
aws dynamodb scan \
    --table-name ProductCatalog \
    --filter-expression "contains(Color, :c) and Price <= :p" \
    --expression-attribute-values file://values.json
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{
    ":c": { "S": "Black" },
    ":p": { "N": "500" }
}
```

如果您定義表達式屬性值，則必須在整個表達式中一致地使用它。您也無法省略 : 符號。

表達式屬性值可搭配索引鍵條件表達式、條件表達式、更新表達式與篩選條件表達式使用。

在 DynamoDB 中使用投影表達式

若要從資料表讀取資料，您可以使用 GetItem、Query 或 Scan 等操作。Amazon DynamoDB 會依預設傳回所有項目屬性。若只要取得部分而非全部屬性，請使用投射表達式。

投射表達式是識別所需屬性的字串。若要擷取單一屬性，請指定其名稱。若為多個屬性，則必須以逗號分隔名稱。

以下是根據 在 DynamoDB 中使用表達式時參考項目屬性 中的 ProductCatalog 項目而來的一些投射表達式範例：

- 單一最上層屬性。

  Title

- 三個最上層屬性。DynamoDB 會擷取整個 `Color` 設定。

  ```
  Title, Price, Color
  ```

- 四個最上層屬性。DynamoDB 會傳回 `RelatedItems` 和 `ProductReviews` 的整個內容。

  ```
  Title, Description, RelatedItems, ProductReviews
  ```

> **Note**
>
> 投影表達式不會影響佈建輸送量消耗。DynamoDB 會根據項目大小判定使用的容量單位數，而非根據傳回至應用程式的資料量判定。

預留單字和特殊字元

DynamoDB 有保留字與特殊字元。DynamoDB 可讓您將這些保留字與特殊字元用於名稱，但建議您避免這樣做，因為每次在表達式中使用這些名稱，都必須使用其別名。如需完整清單，請參閱 DynamoDB 中的保留字。

以下情況，您需要使用表達式屬性名稱取代實際名稱：

- 屬性名稱列於 DynamoDB 保留字清單上。
- 屬性名稱不符合第一個字元為 `a-z` 或 `A-Z`，以及第二個字元 (如果有) 為 `a-z`、`A-Z` 或 `0-9` 的要求。
- 屬性名稱包含 `#` (井字號) 或 `:` (冒號)。

下列 AWS CLI 範例示範如何搭配 `GetItem`操作使用投影表達式。此投射表達式會擷取最上層純量屬性 (`Description`)、清單中的第一個元素 (`RelatedItems[0]`)，以及位於映射巢狀結構內的清單 (`ProductReviews.FiveStar`)。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '"Id": { "N": "123" } \
    --projection-expression "Description, RelatedItems[0], ProductReviews.FiveStar"
```

此範例會傳回下列 JSON。

```
{
```

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

在 DynamoDB 中使用更新表達式

UpdateItem 操作會更新現有項目，如尚不存在，則會將新項目新增到資料表中。您必須提供要更新項目的索引鍵。您也必須提供更新表達式，指出您想要修改的屬性以及您想要指派給它們的值。

更新表達式會指定 UpdateItem 如何修改項目的屬性，例如設定純量值或將元素從清單或映射中刪除。

以下是更新表達式的語法摘要。

```
update-expression ::=
    [ SET action [, action] ... ]
    [ REMOVE action [, action] ...]
    [ ADD action [, action] ... ]
    [ DELETE action [, action] ...]
```

更新表達式包含一或多個子句。每個子句的開頭是 SET、REMOVE、ADD 或 DELETE 關鍵字。您可以依任意順序在更新表達式中包含其中任何子句。不過，每個動作關鍵字都只能出現一次。

在每個子句內，會有以逗號分隔的一或多個動作。每個動作都會代表資料修改。

本節中的範例是以在 DynamoDB 中使用投影表達式中所顯示的 ProductCatalog 項目為基礎。

以下主題涵蓋了 SET 動作的一些不同使用案例。

主題

- SET – 修改或新增項目屬性
- REMOVE – 刪除項目中的屬性
- ADD – 更新數字與集合
- DELETE – 移除集合中的元素
- 使用多個更新表達式

SET – 修改或新增項目屬性

在更新表達式中使用 SET 動作，將一或多個屬性新增至項目。這些屬性如已存在，每一個都會為新值所覆寫。如果您想要避免覆寫現有屬性，則可以搭配使用 SET 與 if_not_exists 函數。if_not_exists 函數為 SET 動作專有，而且只能用於更新表達式。

當您使用 SET 更新清單元素時，會將該元素的內容取代為您所指定的新資料。如果元素尚未存在，則 SET 會在清單結尾附加新的元素。

如果您在單一 SET 操作中新增多個元素，則會依元素號碼依序排序元素。

您也可以使用 SET 來加減類型為 Number 的屬性。若要執行多個 SET 動作，請使用逗號分隔它們。

在下列語法摘要中：

- *path* 元素是項目的文件路徑。
- *operand* 元素可以是項目或函數的文件路徑。

```
set-action ::=
    path = value
```

```
value ::=
    operand
    | operand '+' operand
    | operand '-' operand

operand ::=
    path | function

function ::=
    if_not_exists (path, value)
```

如果項目不包含指定路徑中的屬性，則 if_not_exists 會評估為 value。否則，項目會評估為 path。

下列 PutItem 操作會建立範例參考的範例項目。

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

--item 的引數會存放在 item.json 檔案中。(為求簡化，只會使用一些項目屬性)。

```
{
    "Id": {"N": "789"},
    "ProductCategory": {"S": "Home Improvement"},
    "Price": {"N": "52"},
    "InStock": {"BOOL": true},
    "Brand": {"S": "Acme"}
}
```

主題

- 修改屬性
- 新增清單和映射
- 將元素新增至清單
- 新增巢狀映射屬性
- 增加和減少數值屬性
- 將元素附加至清單
- 防止覆寫現有屬性

修改屬性

Example

更新 ProductCategory 和 Price 屬性。

```
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
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{
    ":c": { "S": "Hardware" },
    ":p": { "N": "60" }
}
```

> **Note**
>
> 在 UpdateItem 操作中，--return-values ALL_NEW 會讓 DynamoDB 傳回在更新後所顯示的項目。

新增清單和映射

Example

新增清單和映射。

```
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
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{
```

```
    ":ri": {
        "L": [
            { "S": "Hammer" }
        ]
    },
    ":pr": {
        "M": {
            "FiveStar": {
                "L": [
                    { "S": "Best product ever!" }
                ]
            }
        }
    }
}
```

將元素新增至清單

Example

將新的屬性新增至 RelatedItems 清單。(請記住，清單元素的開頭為零，因此 [0] 代表清單中的第一個元素、[1] 代表第二個元素，以此類推)。

```
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
```

`--expression-attribute-values` 的引數會存放在 `values.json` 檔案中。

```
{
    ":ri": { "S": "Nails" }
}
```

> **Note**
>
> 當您使用 SET 更新清單元素時，會將該元素的內容取代為您所指定的新資料。如果元素尚未存在，則 SET 會在清單結尾附加新的元素。
> 如果您在單一 SET 操作中新增多個元素，則會依元素號碼依序排序元素。

新增巢狀映射屬性

Example

新增一些巢狀映射屬性。

```
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
```

--expression-attribute-names 的引數會存放在 names.json 檔案中。

```
{
    "#pr": "ProductReviews",
    "#5star": "FiveStar",
    "#3star": "ThreeStar"
}
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{
    ":r5": { "S": "Very happy with my purchase" },
    ":r3": {
        "L": [
            { "S": "Just OK - not that great" }
        ]
    }
}
```

Important

如果父映射不存在，則無法更新巢狀映射屬性。如果您嘗試在父映射 (ProductReviews.FiveStar) 不存在時更新巢狀屬性 （例如 ProductReviews)，DynamoDB 會傳回 ValidationException，並顯示「更新表達式中提供的文件路徑無效，無法更新」訊息。
建立稍後更新巢狀對應屬性的項目時，請初始化父屬性的空對應。例如：

```
{
    "Id": {"N": "789"},
    "ProductReviews": {"M": {}},
    "Metadata": {"M": {}}
}
```

這可讓您像 一樣更新巢狀屬性，ProductReviews.FiveStar而不會發生錯誤。

增加和減少數值屬性

您可以新增或扣除現有數值屬性。若要執行此作業，請使用 + (加號) 和 - (減號) 運算子。

Example

減少項目的 Price。

```
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
```

若要增加 Price，您要在更新表達式中使用 + 運算子。

將元素附加至清單

您可以將元素新增至清單結尾。若要執行此作業，請搭配使用 SET 與 list_append 函數。(函數名稱區分大小寫。) list_append 函數為 SET 動作專有，而且只能用於更新表達式。語法如下。

- list_append (*list1*, *list2*)

此函數需要兩個列表作為輸入，並將所有元素從 *list2* 附加到 *list1*。

Example

在將元素新增至清單中，您會建立 RelatedItems 清單，並使用兩個元素填入它：Hammer 和 Nails。現在，您可在 RelatedItems 結尾多附加兩個元素。

```
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
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{
    ":vals": {
        "L": [
            { "S": "Screwdriver" },
            {"S": "Hacksaw" }
        ]
    }
}
```

最後，您將另一個元素附加到 RelatedItems 的開頭。若要執行此作業，請切換 list_append 元素的順序。(請記住，list_append 採用兩份清單做為輸入，並將第二份清單附加至第一份清單。)

```
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
```

產生的 RelatedItems 屬性現在會包含五個元素，順序如下：Chisel、Hammer、Nails、Screwdriver、Hacksaw。

防止覆寫現有屬性

Example

設定項目的 Price，但只有在項目還沒有 Price 屬性時。(如果 Price 已存在，則不會發生任何事。)

```
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 – 刪除項目中的屬性

在更新表達式中使用 REMOVE 動作，從而在 Amazon DynamoDB 中移除項目的一或多個屬性。若要執行多個 REMOVE 動作，請使用逗號分隔它們。

下列是更新表達式中 REMOVE 的語法摘要。唯一的運算元是您想要移除的屬性文件路徑。

```
remove-action ::=
    path
```

Example

移除項目中的一些屬性。(如果屬性不存在，則不會發生任何事。)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE Brand, InStock, QuantityOnHand" \
    --return-values ALL_NEW
```

移除清單中的元素

您可以使用 REMOVE 刪除清單中的個別元素。

Example

在 將元素附加至清單 中，您修改清單屬性 (RelatedItems)，讓它包含五個元素：

- [0]—Chisel
- [1]—Hammer
- [2]—Nails
- [3]—Screwdriver
- [4]—Hacksaw

下列 AWS Command Line Interface (AWS CLI) 範例Nails會從清單中刪除 Hammer和 。

```
aws dynamodb update-item \
```

```
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE RelatedItems[1], RelatedItems[2]" \
    --return-values ALL_NEW
```

移除 Hammer 和 Nails 之後，會轉移其餘的元素。此清單現在包含下列項目：

- [0]—Chisel
- [1]—Screwdriver
- [2]—Hacksaw

ADD – 更新數字與集合

> **Note**
>
> 一般而言，建議使用 SET，而不是ADD確保等冪性操作。

在更新表達式中使用 ADD 動作，將新的屬性及其值新增至項目。

如果屬性已存在，則 ADD 的行為取決於屬性的資料類型：

- 如果屬性是數字，而您要新增的值也是數字，則此值會以數學方式新增到現有屬性。(如果值是負數，則會從現有屬性減去。)
- 如果屬性是集合，而您要新增的值也是集合，則此值會附加至現有集合。

> **Note**
>
> ADD 動作只支援數字和集合資料類型。

若要執行多個 ADD 動作，請使用逗號分隔它們。

在下列語法摘要中：

- *path* 元素是屬性的文件路徑。屬性必須是 Number 或集合資料類型。
- *value* 元素是您要新增至屬性的數字 (針對 Number 資料類型)，或要附加至屬性的集合 (針對集合類型)。

```
add-action ::=
    path value
```

以下主題涵蓋了 ADD 動作的一些不同使用案例。

主題

- 新增數字
- 將元素新增至集合

新增數字

假設 QuantityOnHand 屬性不存在。下列 AWS CLI 範例會將 QuantityOnHand設定為 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
```

現在已有 QuantityOnHand，您可以重新執行範例，QuantityOnHand 每次的增量為 5。

將元素新增至集合

假設 Color 屬性不存在。下列 AWS CLI 範例會將 Color 設定為具有兩個元素的字串集。

```
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
```

現在已有 Color，您可以在其中新增更多元素。

```
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 – 移除集合中的元素

> **Important**
>
> DELETE 動作僅支援 Set 資料類型。

在更新表達式中使用 DELETE 動作，移除集合中的一或多個元素。若要執行多個 DELETE 動作，請使用逗號分隔它們。

在下列語法摘要中：

- *path* 元素是屬性的文件路徑。屬性必須是集合資料類型。
- *subset* 是您要從 *path* 刪除的一或多個元素。您必須指定 *subset* 做為集合類型。

```
delete-action ::=
    path subset
```

Example

在 將元素新增至集合 中，您會建立 Color 字串集。本範例會從該集合中移除一些元素。

```
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
```

使用多個更新表達式

您可以在單一更新表達式中使用多個動作。套用任何動作之前，所有屬性參考都會針對項目的狀態進行解析。

Example

指定項目 時{"id": "1", "a": 1, "b": 2, "c": 3}，下列表達式會移除a並轉移 b和 的值c：

```
aws dynamodb update-item \
```

```
    --table-name test \
    --key '{"id":{"S":"1"}}' \
    --update-expression "REMOVE a SET b = a, c = b" \
    --return-values ALL_NEW
```

結果為 {"id": "1", "b": 1, "c": 2}。即使a移除b並在相同的表達式中重新指派，這兩個參考都會解析為其原始值。

Example

如果您想要修改屬性的值並完全移除其他屬性，您可以在單一陳述式中使用 SET 和 REMOVE 動作。此操作會將 Price 值減少為 15，同時也會從項目中移除 InStock 屬性。

```
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

如果您想要新增至清單，同時變更其他屬性的值，您可以在單一陳述式中使用兩個 SET 動作。此操作會將「釘子」新增到 RelatedItems 清單屬性中，並將 Price 值設定為 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
```

DynamoDB 中的條件表達式、篩選條件表達式、運算子和函數。

若要操控 DynamoDB 資料表中的資料，您可以使用 PutItem、UpdateItem 和 DeleteItem 操作。針對這些資料操控操作，您可以指定條件表達式來判斷應該修改的項目。如果條件表達式評估為 true，則操作會成功。否則，操作會失敗。

本節說明在 Amazon DynamoDB 中撰寫篩選條件表達式和條件表達式所使用的內建函數與關鍵字。如需有關 DynamoDB 的函數和進行程式設計的詳細資訊，請參閱 使用 DynamoDB 和 AWS SDKs程式設計 和 DynamoDB API 參考。

主題

- 篩選條件和條件表達式的語法
- 進行比較
- 函數
- 邏輯評估
- 括號
- 條件的優先順序

篩選條件和條件表達式的語法

在以下語法摘要中，*operand* 可以是以下項目：

- 最上層屬性名稱，例如 Id、Title、Description 或 ProductCategory
- 參考巢狀屬性的文件路徑

```
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)
```

進行比較

使用這些比較運算子來比較對應單一值的運算元：

- *a* = *b*：如果 *a* 等於 *b*，則為 true。
- *a* <> *b*：如果 *a* 不等於 *b*，則為 true。
- *a* < *b*：如果 *a* 小於 *b*，則為 true。
- *a* <= *b*：如果 *a* 小於或等於 *b*，則為 true。
- *a* > *b*：如果 *a* 大於 *b*，則為 true。
- *a* >= *b*：如果 *a* 大於或等於 *b*，則為 true。

使用 BETWEEN 與 IN 關鍵字可比較運算元與某範圍的值或列舉值清單：

- *a* BETWEEN *b* AND *c*：如果 *a* 大於或等於 *b* 且小於或等於 *c*，則為 true。
- *a* IN (*b*, *c*, *d*) ：如果 *a* 等於清單中的任何值 (例如，*b*、*c* 或 *d* 的任何一個)，則為 true。此清單最多可包含 100 個值，並以逗號分隔。

函數

使用下列函數可判斷項目中是否具有某屬性，或評估某屬性的值。這些函數名稱區分大小寫。針對巢狀屬性，您必須提供其完整文件路徑。

函式	Description
`attribute_exists (path)`	如果項目包含 path 指定的屬性，則為 true。 範例：檢查 Product 資料表中的項目是否具有側視圖。 • `attribute_exists (#Pictures.#SideView)`
`attribute_not_exists (path)`	如果項目中沒有 path 指定的屬性，則為 true。

函式	Description
	範例：檢查項目是否具有 Manufacturer 屬性。 • `attribute_not_exists (Manufacturer)`

函式	Description
attribute_type (*path*, *type*)	如果指定路徑的屬性是特定資料類型，則為 true。type 參數必須是下列其中一種： • S：字串 • SS：字串集合 • N：數字 • NS：數字集合 • B：二進位 • BS：二進位集合 • BOOL：布林值 • NULL：Null • L：清單 • M：映射 您必須為 type 參數使用表達式屬性值。 範例：檢查 QuantityOnHand 屬性的類型是否為清單。在此範例中，:v_sub 是字串 L 的預留位置。 • `attribute_type (ProductReviews.FiveStar, :v_sub)` 您必須為 type 參數使用表達式屬性值。

函式	Description
`begins_with ( path, substr)`	如果 path 指定的屬性以特定子字串開頭，則為 true。 範例：檢查正視圖 URL 的前幾個字元是否為 http://。 • `begins_with (Pictures.FrontView, :v_sub)` 表達式屬性值 :v_sub 是 http:// 的預留位置。

函式	Description
`contains (path, operand)`	如果 path 指定的屬性為下列其中一個項目，則為 true： • 內含特定子字串的 String。 • 在組合中內含特定元素的 Set。 • 在清單內含特定元素的 List。 如果 path 指定的屬性是 String，則 operand 必須是 String。如果 path 指定的屬性是 Set，則 operand 必須是集合的元素類型。 路徑和運算元必須不同。也就是說，`contains (a, a)` 會傳回錯誤。 範例：檢查 Brand 屬性是否包含子字串 Company。 • `contains (Brand, :v_sub)` 表達式屬性值 :v_sub 是 Company 的預留位置。 範例：檢查產品是否提供紅色款式。 • `contains (Color, :v_sub)` 表達式屬性值 :v_sub 是 Red 的預留位置。

函式	Description
`size (path)`	傳回代表屬性大小的數字。以下是可搭配 `size` 使用的有效資料類型。 如果屬性的類型為 `String`，`size` 會傳回字串長度。 範例：檢查字串 `Brand` 是否小於或等於 20 個字元。表達式屬性值 `:v_sub` 是 20 的預留位置。 • `size (Brand) <= :v_sub` 如果屬性的類型為 `Binary`，`size` 會傳回屬性值中的位元組數目。 範例：假設 `ProductCatalog` 項目具有名為 `VideoClip` 的二進制屬性，其中包含使用中產品的短片。以下表達式會檢查 `VideoClip` 是否超過 64,000 個位元組。表達式屬性值 `:v_sub` 是 64000 的預留位置。 • `size(VideoClip) > :v_sub` 如果屬性是 `Set` 資料類型，`size` 會傳回集合中的元素數目。 範例：檢查產品是否提供多色款式。表達式屬性值 `:v_sub` 是 1 的預留位置。 • `size (Color) < :v_sub`

函式	Description
	如果屬性的類型是 List 或 Map，size 會傳回子元素數目。 範例：檢查 OneStar 評論數目是否超過特定閾值。表達式屬性值 :v_sub 是 3 的預留位置。 • `size(ProductReviews.OneStar) > :v_sub`

邏輯評估

使用 AND、OR 與 NOT 關鍵字可執行邏輯評估。在下列清單中，*a* 與 *b* 代表要評估的條件。

- *a* AND *b*：如果 *a* 與 *b* 皆為 true，則為 true。
- *a* OR *b*：如果 *a* 或 *b* (或兩者) 為 true，則為 true。
- NOT *a*：如果 *a* 為 false，則為 true。如果 *a* 為 true，則為 false。

以下是運算中 AND 的程式碼範例。

```
dynamodb-local (*)> select * from exprtest where a > 3 and a < 5;
```

括號

使用括號可變更邏輯評估的優先順序。例如，假設條件 *a* 與 *b* 皆為 true，且條件 *c* 為 false。下列表達式會評估為 true：

- *a* OR *b* AND *c*

不過，如果您以括號括住條件，則會先評估該條件。例如，下列表達式會評估為 false：

- (*a* OR *b*) AND *c*

> **Note**
>
> 您可以在表達式中巢狀使用括號。最內部的條件最先評估。

以下是邏輯評估中帶括號的程式碼範例。

```
dynamodb-local (*)> select * from exprtest where attribute_type(b, string)
or ( a = 5 and c = “coffee”);
```

條件的優先順序

DynamoDB 使用下列優先順序規則，從左到右評估條件：

- = <> < <= > >=
- IN
- BETWEEN
- attribute_exists attribute_not_exists begins_with contains
- 括號
- NOT
- AND
- OR

DynamoDB 條件表達式 CLI 範例

以下是使用條件表達式的一些 AWS Command Line Interface (AWS CLI) 範例。這些範例是以 在 DynamoDB 中使用表達式時參考項目屬性 中引進的 ProductCatalog 資料表為基礎。此資料表的分割區索引鍵是 Id；沒有排序索引鍵。以下 PutItem 操作會建立將在範例中參考的範例 ProductCatalog 項目。

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

--item 的引數會存放在 item.json 檔案中。(為求簡化，只會使用一些項目屬性)。

```
{
    "Id": {"N": "456" },
    "ProductCategory": {"S": "Sporting Goods" },
    "Price": {"N": "650" }
}
```

主題

- 條件式放置
- 條件式刪除
- 條件式更新
- 條件式表達式範例

條件式放置

PutItem 操作將會覆寫具有相同主索引鍵 (如有) 的項目。若您想要避免這種情況，請使用條件表達式。只有在相關項目還沒有相同的主索引鍵時，才會允許繼續寫入。

下列範例會在嘗試寫入操作之前，使用 attribute_not_exists() 檢查資料表中是否存在主索引鍵。

> **Note**
>
> 如果您的主索引鍵同時包含分割區索引鍵 (pk) 和排序索引鍵 (sk)，則在嘗試寫入操作之前，參數會檢查 attribute_not_exists(pk) AND attribute_not_exists(sk) 作為完整陳述式評估為 true 或 false。

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json \
    --condition-expression "attribute_not_exists(Id)"
```

如果條件表達式評估為 false，則 DynamoDB 會傳回以下錯誤訊息：The conditional request failed (條件式請求失敗)。

> **Note**
>
> 如需 attribute_not_exists 和其他函數的詳細資訊，請參閱 DynamoDB 中的條件表達式、篩選條件表達式、運算子和函數。。

條件式刪除

若要執行條件式刪除，您可以搭配使用 DeleteItem 操作與條件表達式。條件表達式必須評估為 true，操作才會成功；否則，操作會失敗。

考慮以上定義的項目。

假設您想要刪除項目，但只限在下列條件下：

- `ProductCategory` 是 "Sporting Goods" 或 "Gardening Supplies"。
- `Price` 介於 500 與 600 之間。

以下範例會嘗試刪除項目。

```
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
```

`--expression-attribute-values` 的引數會存放在 `values.json` 檔案中。

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

> **Note**
>
> 在條件表達式中，`:` (冒號字元) 表示表達式屬性值 (即實際值的預留位置)。如需詳細資訊，請參閱 在 DynamoDB 中使用表達式屬性值。
> 如需 IN、AND 和其他關鍵字的詳細資訊，請參閱 DynamoDB 中的條件表達式、篩選條件表達式、運算子和函數。。

在此範例中，`ProductCategory` 比較會評估為 true，但 `Price` 比較會評估為 false。這會導致條件表達式評估為 false，使 `DeleteItem` 操作失敗。

條件式更新

若要執行條件式更新，您可以搭配使用 `UpdateItem` 操作與條件表達式。條件表達式必須評估為 true，操作才會成功；否則，操作會失敗。

> **Note**
>
> UpdateItem 也支援更新表達式；其中，您可以指定想要對項目進行的修改。如需詳細資訊，請參閱在 DynamoDB 中使用更新表達式。

假設您已開始使用以上定義的項目。

以下範例會執行 UpdateItem 操作。其試圖將產品的 Price 減少 75，但如果目前的 Price 小於或等於 500，條件表達式會阻止更新。

```
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
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{
    ":discount": { "N": "75"},
    ":limit": {"N": "500"}
}
```

如果起始 Price 是 650，則 UpdateItem 操作會將 Price 減少為 575。如果您再次執行 UpdateItem 操作，則 Price 會減少為 500。如果您執行它第三次，則條件表達式會評估為 false，而更新會失敗。

> **Note**
>
> 在條件表達式中，: (冒號字元) 表示表達式屬性值 (即實際值的預留位置)。如需詳細資訊，請參閱 在 DynamoDB 中使用表達式屬性值。
> 如需 ">" 和其他運算子的詳細資訊，請參閱 DynamoDB 中的條件表達式、篩選條件表達式、運算子和函數。。

條件式表達式範例

如需下列範例中所用函數的詳細資訊，請參閱 DynamoDB 中的條件表達式、篩選條件表達式、運算子和函數。。若要進一步了解如何指定表達式中的不同屬性類型，請參閱 在 DynamoDB 中使用表達式時參考項目屬性。

檢查項目中的屬性

您可以檢查任何屬性是否存在。如果條件表達式評估為 true，則操作會成功；否則，操作會失敗。

只有在產品沒有 Price 屬性時，以下範例才會使用 attribute_not_exists 來刪除產品。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_not_exists(Price)"
```

DynamoDB 也提供 attribute_exists 函數。只有在產品收到不佳的檢閱時，以下範例才會刪除產品。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_exists(ProductReviews.OneStar)"
```

檢查屬性類型

您可以使用 attribute_type 函數，以檢查屬性值的資料類型。如果條件表達式評估為 true，則操作會成功；否則，操作會失敗。

下列範例只有在具有字串集合類型的 Color 屬性的情況下，才會使用 attribute_type 刪除產品。

```
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
```

--expression-attribute-values 的引數會存放在 expression-attribute-values.json 檔案中。

```
{
```

```
    ":v_sub":{"S":"SS"}
}
```

檢查字串起始值

您可以使用 begins_with 函數，來檢查字串屬性值是否以特定子字串做為開頭。如果條件表達式評估為 true，則操作會成功；否則，操作會失敗。

只有在 begins_with 映射的 FrontView 元素以特定值作為開頭時，下列範例才會使用 Pictures 刪除產品。

```
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
```

--expression-attribute-values 的引數會存放在 expression-attribute-values.json 檔案中。

```
{
    ":v_sub":{"S":"http://"}
}
```

檢查集合中的元素

您可以使用 contains 函數，檢查集合中的元素或尋找字串內的子字串。如果條件表達式評估為 true，則操作會成功；否則，操作會失敗。

只有在 Color 字串集合具有含特定值的元素時，下列範例才會使用 contains 刪除產品。

```
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
```

--expression-attribute-values 的引數會存放在 expression-attribute-values.json 檔案中。

```
{
```

```
    ":v_sub":{"S":"Red"}
}
```

檢查屬性值的大小

您可以使用 `size` 函數，來檢查屬性值的大小。如果條件表達式評估為 true，則操作會成功；否則，操作會失敗。

只有在 `VideoClip` 二進制屬性大於 `64000` 位元組時，下列範例才會使用 `size` 刪除產品。

```
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
```

`--expression-attribute-values` 的引數會存放在 expression-attribute-values.json 檔案中。

```
{
    ":v_sub":{"N":"64000"}
}
```

使用 DynamoDB 的存留時間 (TTL) 功能

DynamoDB 的存留時間 (TTL) 功能是一種具成本效益的方法，可用於刪除不再需要的項目。TTL 可讓您為每個項目定義過期時間戳記，以標示該項目何時不再需要。DynamoDB 會在項目過期後的幾天內自動刪除它們，且不會耗用寫入輸送量。

若要使用 TTL，請先在資料表中啟用該功能，接著定義一個屬性以儲存 TTL 過期時間戳記。時間戳記必須以秒精細程度以 Unix epoch 時間格式儲存為數字資料類型。TTL 程序會忽略 TTL 屬性不是數字類型的項目。每次建立或更新項目時，您可計算其過期時間，並將結果儲存在 TTL 屬性中。

具有有效且已過期 TTL 屬性的項目，可能會在過期後數天內由系統自動刪除。您仍可更新尚未刪除的過期項目，例如變更或移除其 TTL 屬性。在更新過期項目時，建議使用條件運算式，以確保該項目未在此期間被刪除。使用篩選運算式，從掃描與查詢結果中排除過期項目。

被刪除的項目運作方式與一般刪除操作所刪除的項目相同。項目刪除後，會以「服務刪除」的形式送入 DynamoDB Streams，而非「使用者刪除」，並會從本機次要索引與全域次要索引中移除，與其他刪除操作相同。

如果您使用 Global Tables version 2019.11.21 (目前版本)，且同時啟用 TTL 功能，DynamoDB 會將 TTL 刪除複製至所有資料表複本。初始 TTL 刪除在 TTL 到期的區域內不會消耗寫入容量單位 (WCU)。不過，當使用佈建容量時，複製 TTL 刪除至資料表複本會在每個複本區域中消耗一個複製的寫入容量單位；若使用隨需容量模式，則會消耗一個複製的寫入單位，並產生相應費用。

如需 TTL 的詳細資訊，請參閱下列主題：

主題

- 在 DynamoDB 中啟用存留時間 (TTL)
- 在 DynamoDB 中運算存留時間 (TTL)
- 操作過期項目與存留時間 (TTL)

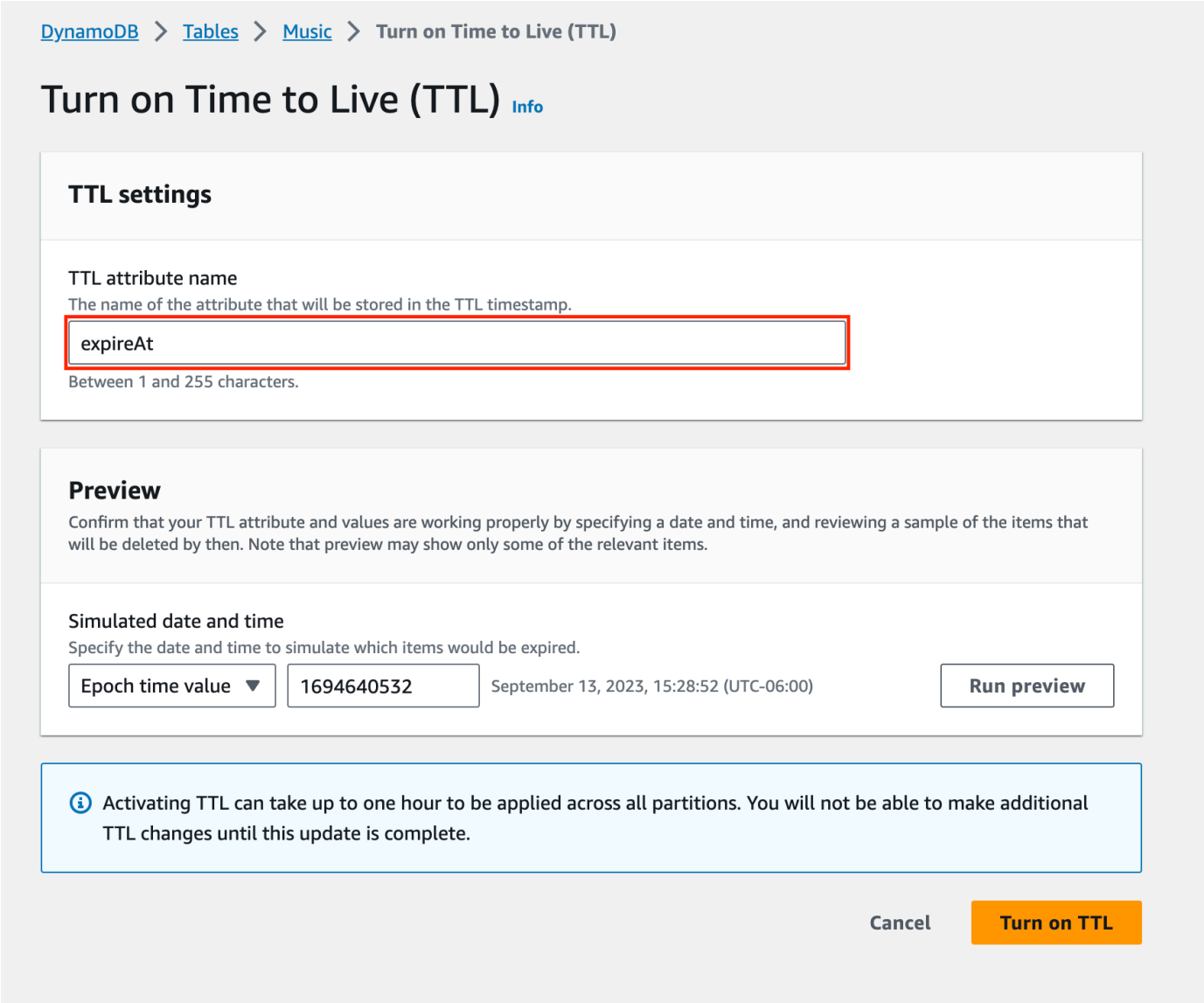
在 DynamoDB 中啟用存留時間 (TTL)

> **Note**
>
> 為協助偵錯與驗證 TTL 功能是否正常運作，項目的 TTL 值會以純文字形式記錄在 DynamoDB 診斷日誌中。

您可以在 Amazon DynamoDB 主控台、 AWS Command Line Interface (AWS CLI) 中啟用 TTL，或使用 Amazon DynamoDB API 參考與任何支援的 AWS SDKs。在所有分割區啟用 TTL 約需一小時。

使用 AWS 主控台啟用 DynamoDB TTL

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 選擇 Tables (資料表)，然後選擇您想要修改的資料表。
3. 在附加設定分頁的存留時間 (TTL)區段中，選取開啟以啟用 TTL。
4. 在資料表上啟用 TTL 時，DynamoDB 會要求您識別服務在判斷項目是否符合過期資格時將尋找的特定屬性名稱。如下所示的 TTL 屬性名稱區分大小寫，且必須與讀寫操作中定義的屬性一致。若不一致，過期項目將無法被刪除。若要重新命名 TTL 屬性，您必須先停用 TTL，然後使用新的屬性名稱重新啟用。停用後，TTL 仍會在約 30 分鐘內持續處理刪除作業。還原資料表後，必須重新設定 TTL。



5. (選用) 您可以模擬到期日期與時間，並比對部分項目以進行測試。這將提供項目範例清單，並確認其中包含設定了 TTL 屬性名稱與到期時間的項目。

啟用 TTL 後，當您在 DynamoDB 主控台檢視項目時，TTL 屬性會顯示為 TTL。您可以將滑鼠游標懸停到屬性上，來檢視項目過期的日期和時間。

使用 API 啟用 DynamoDB TTL

Python

您可以透過 UpdateTimeToLive 操作以程式碼啟用 TTL。

```
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')
```

您可以使用 DescribeTimeToLive 操作確認 TTL 是否已啟用，該操作會回報資料表的 TTL 狀態。TimeToLive 狀態可能為 ENABLED 或 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

您可以透過 UpdateTimeToLiveCommand 操作以程式碼啟用 TTL。

```
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');
```

使用 啟用存留時間 AWS CLI

1. 啟用 TTLExample 表中的 TTL。

```
aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-
specification "Enabled=true, AttributeName=ttl"
```

2. 描述 TTLExample 表中的 TTL。

```
aws dynamodb describe-time-to-live --table-name TTLExample
{
    "TimeToLiveDescription": {
        "AttributeName": "ttl",
        "TimeToLiveStatus": "ENABLED"
    }
}
```

3. 使用 BASH shell 及 AWS CLI，將項目新增至已設定存留時間屬性的 TTLExample 資料表。

```
EXP=`date -d '+5 days' +%s`
aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl":
 {"N": "'$EXP'"}}'
```

此範例建立的過期時間為從目前的日期開始加 5 天。然後，將過期時間轉換成 Epoch 時間格式，最終將項目新增到 "TTLExample" 表。

> **Note**
>
> 設定存留時間過期數值的其中一種方式，是計算要新增到過期時間的秒數。例如，5 天等於 432,000 秒。但是通常建議從日期開始操作。

取得目前時間的 Epoch 時間格式非常容易，如以下範例所示。

- Linux 終端機：`date +%s`
- Python：`import time; int(time.time())`
- Java：`System.currentTimeMillis() / 1000L`
- JavaScript：`Math.floor(Date.now() / 1000)`

使用 啟用 DynamoDB TTL 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
```

您可以在此處找到有關在 CloudFormation 範本中使用 TTL 的其他詳細資訊。

在 DynamoDB 中運算存留時間 (TTL)

實作 TTL 的常見方式是依據項目建立或最後更新的時間設定其過期時間。可透過將時間加至 `createdAt` 與 `updatedAt` 時間戳記來完成。例如，新建立的項目可將 TTL 設定為 `createdAt` + 90 天。當項目更新時，可將 TTL 重新計算為 `updatedAt` + 90 天。

計算出的過期時間必須採用 epoch 格式 (以秒為單位)。要符合過期與刪除條件，TTL 不得早於五年前。如果您使用任何其他格式，TTL 處理程序會忽略該項目。如果您將過期時間設定為您希望項目過期的未來某個時間，則項目會在該時間之後過期。例如，假設您將過期時間設定為 1724241326 （即 2024 年 8 月 21 日星期一 11：55：26 (UTC))。項目會在指定的時間後過期。沒有最短 TTL 持續時間。您可以將過期時間設定為任何未來的時間，例如從目前時間起算 5 分鐘。不過，DynamoDB 通常會在過期時間後 48 小時內刪除過期的項目，而不是在項目過期時立即刪除。

主題

- 建立項目並設定存留時間

- 更新項目並重新設定存留時間

建立項目並設定存留時間

下列範例示範如何在建立新項目時，使用 expireAt 作為 TTL 屬性名稱來計算過期時間。指派語句會將目前時間作為變數取得。在此範例中，過期時間設定為從目前時間起算的 90 天。接著，將時間轉換為 epoch 格式，並以整數型別儲存在 TTL 屬性中。

下列程式碼範例示範如何建立含有 TTL 的項目。

Java

適用於 Java 2.x 的 SDK

```
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;
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 PutItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

```
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');
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 PutItem。

Python

適用於 Python (Boto3) 的 SDK

```
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"
)
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 PutItem。

更新項目並重新設定存留時間

此範例延續自上一節。若項目更新，可重新計算其過期時間。下列範例將 `expireAt` 時間戳記重新計算為自目前時間起的 90 天。

下列程式碼範例示範如何更新項目的 TTL。

Java

適用於 Java 2.x 的 SDK

更新資料表中現有 DynamoDB 項目上的 TTL。

```
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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

```
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');
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateItem。

Python

適用於 Python (Boto3) 的 SDK

```
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"
)
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 UpdateItem。

本介紹中的 TTL 範例示範了一種方法，可確保僅保留最近更新的項目於資料表中。更新的項目其存留期會延長，而建立後未更新的項目將在到期後自動刪除，不產生額外成本，有助於減少儲存並維持資料表整潔。

操作過期項目與存留時間 (TTL)

可從讀取與寫入操作中篩選出待刪除的過期項目。當過期資料已失效且不應再使用時，這項設計特別實用。若未進行篩選，這些項目將持續出現在讀取與寫入操作中，直到由背景程序刪除。

> **Note**
>
> 在刪除前，這些項目仍計入儲存與讀取成本。

TTL 刪除可在 DynamoDB 串流中識別，但僅限於刪除發生的區域。被複寫至全域資料表其他區域的 TTL 刪除，無法在那些區域的 DynamoDB 串流中識別。

從讀取操作中篩選過期項目

針對掃描與查詢等讀取操作，可使用篩選條件表達式排除待刪除的過期項目。如下列程式碼片段所示，篩選條件表達式可排除 TTL 時間小於或等於目前時間的項目。例如，Python SDK 程式碼中包含指派語句，將目前時間作為變數 (now) 取得，並將其轉換為 `int` 以符合 epoch 時間格式。

下列程式碼範例示範如何查詢 TTL 項目。

Java

適用於 Java 2.x 的 SDK

查詢篩選表達式，以使用 在 DynamoDB 資料表中收集 TTL 項目 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;
        }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

查詢篩選表達式，以使用 在 DynamoDB 資料表中收集 TTL 項目 適用於 JavaScript 的 AWS SDK。

```
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');
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

查詢篩選表達式，以使用 在 DynamoDB 資料表中收集 TTL 項目 適用於 Python (Boto3) 的 AWS SDK。

```
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")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

有條件地寫入過期項目

可使用條件表達式以避免對過期項目執行寫入。以下程式碼片段為條件式更新，用以檢查過期時間是否晚於目前時間。若條件為 true，則繼續執行寫入操作。

下列程式碼範例示範如何有條件地更新項目的 TTL。

Java

適用於 Java 2.x 的 SDK

使用條件更新資料表中現有的 DynamoDB 項目之 TTL。

```
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;
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用條件更新資料表中現有的 DynamoDB 項目之 TTL。

```
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');
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateItem。

Python

適用於 Python (Boto3) 的 SDK

使用條件更新資料表中現有的 DynamoDB 項目之 TTL。

```
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",
)
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 UpdateItem。

在 DynamoDB 串流中識別已刪除的項目

串流紀錄包含使用者身分欄位 `Records[<index>].userIdentity`。由 TTL 程序刪除的項目包含下列欄位：

```
Records[<index>].userIdentity.type
"Service"

Records[<index>].userIdentity.principalId
"dynamodb.amazonaws.com"
```

下列 JSON 顯示單一串流紀錄的相關部分：

```
"Records": [
  {
 ...
  "userIdentity": {
  "type": "Service",
       "principalId": "dynamodb.amazonaws.com"
    }
   ...
 }
]
```

在 DynamoDB 中查詢資料表

您可以使用 Amazon DynamoDB 中的 `Query` API 操作，以根據主索引鍵值尋找項目。

您必須提供分割區索引鍵屬性的名稱，以及該屬性的單一值。`Query` 會傳回所有具有該分割區索引鍵值的項目。您可以選擇是否提供排序索引鍵屬性，使用比較運算子縮小搜尋結果。

如需如何使用 `Query` (例如請求語法、回應參數和其他範例)，請參閱《Amazon DynamoDB API 參考》中的查詢。

主題

- DynamoDB 中查詢操作的索引鍵條件表達式

- DynamoDB 中查詢操作的篩選條件表達式
- 在 DynamoDB 中將資料表查詢結果分頁
- 在 DynamoDB 中使用查詢操作的其他方面

DynamoDB 中查詢操作的索引鍵條件表達式

您可以在索引鍵條件表達式中使用任何屬性名稱，只要第一個字元為 a-z 或 A-Z，且其餘字元 (若有的話，從第二個字元開始) 為 a-z、A-Z，或 0-9。此外，屬性名稱不得為 DynamoDB 的保留字。(如需這些保留字的完整清單，請參閱「DynamoDB 中的保留字」)。若屬性名稱不符合這些需求，您必須將表達式屬性名稱定義為預留位置。如需詳細資訊，請參閱DynamoDB 中的表達式屬性名稱 (別名)。

針對具有給定分割區索引鍵值的項目，DynamoDB 會依照排序索引鍵值，將這些項目以排序後的順序存放在緊鄰位置。在 Query 操作中，DynamoDB 會以排序後的順序擷取項目，並使用 KeyConditionExpression 及任何存在的 FilterExpression 處理項目。只有在這個時候，Query 的結果才會傳回用戶端。

Query 操作永遠都會傳回一個結果集。如果找不到相符項目，表示結果集是空的。

Query 結果永遠都會根據排序索引鍵值進行排序。如果排序索引鍵的資料類型是 Number，即依數值順序傳回結果。否則，按 UTF-8 位元組順序傳回結果。根據預設，排序順序為遞增排序。若要反轉順序，請將 ScanIndexForward 參數設為 false。

單一 Query 操作最多可擷取 1 MB 的資料。這項限制會在任何 FilterExpression 或 ProjectionExpression 套用到結果之前套用。若回應中有 LastEvaluatedKey 且為非 Null，您即必須為結果集編製分頁 (請參閱在 DynamoDB 中將資料表查詢結果分頁)。

索引鍵條件表達式範例

若要指定搜尋條件，請使用索引鍵條件表達式 (判斷要從資料表或索引讀取之項目的字串)。

您必須將分割區索引鍵名稱及數值指定為相等條件。您無法在索引鍵條件表達式中使用非索引鍵屬性。

您可以選擇性為排序索引鍵提供第二個條件 (若有的話)。排序索引鍵條件必須使用下列其中一個比較運算子：

- *a* = *b*：如果屬性 *a* 等於數值 *b*，則為 true
- *a* < *b*：如果 *a* 小於 *b*，則為 true
- *a* <= *b*：如果 *a* 小於或等於 *b*，則為 true

- *a* > *b*：如果 *a* 大於 *b*，則為 true
- *a* >= *b*：如果 *a* 大於或等於 *b*，則為 true
- *a* BETWEEN *b* AND *c*：如果 *a* 大於或等於 *b* 且小於或等於 *c*，則為 true。

同樣支援下列函數：

- begins_with (*a*, *substr*)：如果屬性 *a* 的值開頭為特定子字串，則為 true。

下列 AWS Command Line Interface (AWS CLI) 範例示範如何使用金鑰條件表達式。這些表達式會使用預留位置 (例如 :name 和 :sub)，而非實際的值。如需更多詳細資訊，請參閱 DynamoDB 中的表達式屬性名稱 (別名) 及 在 DynamoDB 中使用表達式屬性值。

Example

查詢 Thread 表是否有特定的 ForumName (分割區索引鍵)。查詢會讀取所有具有該 ForumName 值的項目，因為 KeyConditionExpression 中不包含排序索引鍵 (Subject)。

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name" \
    --expression-attribute-values  '{":name":{"S":"Amazon DynamoDB"}}'
```

Example

查詢 Thread 表是否有特定的 ForumName (分割區索引鍵)，但這次僅傳回具有指定 Subject (排序索引鍵) 的項目。

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name and Subject = :sub" \
    --expression-attribute-values  file://values.json
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{
    ":name":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"}
}
```

Example

查詢 Reply 表是否有特定的 Id (分割區索引鍵)，但只傳回以特定字元開頭的 ReplyDateTime (排序索引鍵) 項目。

```
aws dynamodb query \
    --table-name Reply \
    --key-condition-expression "Id = :id and begins_with(ReplyDateTime, :dt)" \
    --expression-attribute-values  file://values.json
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{
    ":id":{"S":"Amazon DynamoDB#DynamoDB Thread 1"},
    ":dt":{"S":"2015-09"}
}
```

DynamoDB 中查詢操作的篩選條件表達式

若您需要更精確的 Query 結果，您可以選擇性的提供篩選條件表達式。篩選條件表達式會判斷要傳回 Query 結果中的哪些項目。所有其他結果皆會捨棄。

篩選條件表達式會在 Query 完成之後，並在傳回結果之前套用。因此，無論是否有篩選條件表達式，Query 都會使用相同數量的讀取容量。

Query 操作最多可擷取 1 MB 的資料。系統會先套用這項限制，再評估篩選條件表達式。

篩選條件表達式無法包含分割區索引鍵或排序索引鍵屬性。您必須在索引鍵條件表達式中指定這些屬性，而非篩選條件表達式。

篩選條件表達式的語法和索引鍵條件表達式的語法類似。篩選條件表達式可以使用與索引鍵條件表達式相同的比較子、函數和邏輯運算子。此外，篩選條件表達式可以使用不等於運算子 (<>)、OR 運算子、CONTAINS 運算子、IN 運算子、BEGINS_WITH 運算子、BETWEEN 運算子、EXISTS 運算子和 SIZE 運算子。如需詳細資訊，請參閱DynamoDB 中查詢操作的索引鍵條件表達式及篩選條件和條件表達式的語法。

Example

下列 AWS CLI 範例會查詢特定 ForumName（分割區索引鍵）和 Subject（排序索引鍵）的Thread資料表。在找到的項目中，只會傳回最受歡迎的討論主題：換句話說，只會傳回具有超過一定 Views 數量的主題。

```
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
```

`--expression-attribute-values` 的引數會存放在 `values.json` 檔案中。

```
{
    ":fn":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"},
    ":num":{"N":"3"}
}
```

請注意，`Views` 為 DynamoDB 中的保留字 (請參閱 DynamoDB 中的保留字)，所以此範例會使用 `#v` 作為預留位置。如需詳細資訊，請參閱DynamoDB 中的表達式屬性名稱 (別名)。

> **Note**
>
> 篩選條件表達式會從 `Query` 結果集中移除項目。若可能的話，請避免在預期會擷取大量項目，但又需要捨棄它們大部分的情況下，使用 `Query`。

在 DynamoDB 中將資料表查詢結果分頁

DynamoDB 會對 `Query` 操作的結果進行分頁。透過編製分頁，`Query` 結果會分成數個大小為 1 MB (或更小) 的資料「頁」。應用程式可以處理結果的第一頁、第二頁，以此類推。

單一 `Query` 只會傳回符合 1 MB 大小限制的結果集。為判斷是否有更多結果，並且一次擷取一頁結果，應用程式應執行下列作業：

1. 檢查低層級 `Query` 結果：
   - 若結果包含 `LastEvaluatedKey` 元素且為非空值，請接著進行步驟 2。
   - 若結果中「沒有」`LastEvaluatedKey`，就表示再也沒有要擷取的項目。
2. 使用相同的 `KeyConditionExpression` 建構 `Query`。但這一次採用步驟 1 的 `LastEvaluatedKey` 值，並用它做為新 `Query` 要求的 `ExclusiveStartKey` 參數。
3. 執行新的 `Query` 請求。

4. 前往步驟 1。

換句話說，LastEvaluatedKey 回應的 Query 應做為下一個 ExclusiveStartKey 請求的 Query 使用。若 LastEvaluatedKey 回應中沒有 Query 元素，表示您已擷取到結果的最終頁。如果 LastEvaluatedKey 不是空的，則不一定意味著結果集中有更多資料。檢查 LastEvaluatedKey 是否為空，是確定您是否已到達結果集末頁的唯一方式。

您可以使用 AWS CLI 來檢視此行為。會重複 AWS CLI 傳送低階Query請求至 DynamoDB，直到結果LastEvaluatedKey中不再出現 為止。請考慮下列從特定年份擷取電影標題 AWS CLI 的範例。

```
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
```

通常， 會自動 AWS CLI 處理分頁。不過，在此範例中， AWS CLI --page-size 參數會限制每頁的項目數量。--debug 參數會列印請求及回應的下層資訊。

如果您執行此範例，DynamoDB 的第一個回應會類似以下內容。

```
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}'
```

回應中的 LastEvaluatedKey 會指出並未擷取所有項目。 AWS CLI 然後， 向 DynamoDB 發出另一個Query請求。此請求和回應模式會持續到最終回應出現為止。

```
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}'
```

若沒有 LastEvaluatedKey，就表示已不再有要擷取的項目。

> **Note**
>
> AWS SDKs 會處理低階 DynamoDB 回應（包括是否存在 `LastEvaluatedKey`），並提供用於分頁`Query`結果的各種抽象概念。例如，適用於 Java 的開發套件文件介面會提供 `java.util.Iterator` 支援，讓您可一次處理一個結果。
> 如需各種程式設計語言的程式碼範例，請參閱《Amazon DynamoDB 入門指南》和所需語言的 AWS 軟體開發套件文件。

您還可以透過限制結果集的項目數 (使用 `Query` 操作的 `Limit` 參數) 來減少頁面大小。

如需使用 DynamoDB 進行查詢的詳細資訊，請參閱 在 DynamoDB 中查詢資料表。

在 DynamoDB 中使用查詢操作的其他方面

本節涵蓋 DynamoDB 查詢操作的其他層面，包括限制結果大小、計算掃描項目與傳回項目、監控讀取容量耗用量，以及控制讀取一致性。

限制結果集的項目數

您可以使用 `Query` 操作來限制其讀取的項目數。若要執行此作業，請將 `Limit` 參數設為您希望的最大項目數。

例如，假設您 `Query` 一份資料表，將 `Limit` 值設為 6 且不使用篩選條件表達式。`Query` 結果會包含資料表中符合請求索引鍵條件表達式的前六個項目。

現在假設您在 `Query` 中新增一個篩選條件表達式。在此情況下，DynamoDB 最多可讀取六個項目，然後只傳回符合篩選條件表達式的項目。最後的 `Query` 結果包含六個或更少的項目，即使有更多項目符合篩選條件表達式 (如果 DynamoDB 持續讀取更多項目)。

計算結果中的項目

除了符合您條件的項目之外，`Query` 回應還包含了下列元素：

- `ScannedCount`：套用篩選條件表達式 (若有) 前符合索引鍵條件表達式的項目數。
- `Count`：套用篩選條件表達式 (若有) 後剩餘的項目數。

> **Note**
>
> 若不使用篩選條件表達式，`ScannedCount` 和 `Count` 就會有相同的值。

若 Query 結果集的大小大於 1 MB，則 ScannedCount 和 Count 僅代表總項目的部分計數。您需要執行多項 Query 操作，才能擷取所有的結果 (請參閱在 DynamoDB 中將資料表查詢結果分頁)。

每個 Query 回應都包含經該特定 Query 請求處理過的項目 ScannedCount 和 Count。若要取得所有 Query 請求的總計，您可以為 ScannedCount 及 Count 記錄流水帳。

查詢使用的容量單位

您可以 Query 任何資料表或次要索引，只要您提供分割索引鍵屬性的名稱以及該屬性的單一值即可。Query 會傳回具有該分割區索引鍵值的所有項目。您可以選擇是否提供排序索引鍵屬性，並使用比較運算子縮小搜尋結果。QueryAPI 操作會使用讀取容量單位，如下所示。

若您對下列進行 **Query**	DynamoDB 使用的讀取容量單位就會來自...
資料表	該資料表的佈建讀取容量。
全域次要索引	該索引的佈建讀取容量。
本機次要索引	該基礎資料表的佈建讀取容量。

根據預設，Query 操作不會傳回任何使用之讀取容量的相關資料。但您可以在 ReturnConsumedCapacity 請求中指定 Query 參數，來取得這項資訊。下列為 ReturnConsumedCapacity 的有效設定：

- NONE：不會傳回耗用的容量資料。(此為預設值)。
- TOTAL：回應包括耗用的讀取容量單位總數。
- INDEXES：回應顯示耗用的讀取容量單位總數，以及每個資料表和存取之索引的耗用容量。

DynamoDB 會根據項目的數量以及大小 (而不是根據傳回給應用程式的資料量) 計算耗用的讀取容量單位數。因此，無論您請求所有屬性 (預設行為) 或只請求部分屬性 (使用投影表達式)，使用的容量單位數都相同。無論您是否使用篩選條件表達式，此數量都相同。Query 會耗用最小讀取容量單位，每秒執行一個高度一致性讀取；或者針對最大 4 KB 的項目，每秒執行兩個最終一致讀取。如果您需要讀取大於 4KB 的項目，DynamoDB 需要額外的讀取請求單位。空白資料表和具有稀疏分割區索引鍵數量的極龐大資料表，可能會看到超出查詢資料量的額外 RCU 收費。這涵蓋了處理 Query 請求的成本，即使沒有資料也一樣。

查詢的讀取一致性

根據預設，Query 操作會執行最終一致讀取。這表示 Query 的結果可能不會反映最近完成之 PutItem 或 UpdateItem 操作所造成的變更。如需詳細資訊，請參閱DynamoDB 讀取一致性。

若您需要高度一致性讀取，請在 ConsistentRead 請求中將 true 參數設為 Query。

掃描 DynamoDB 中的資料表

Amazon DynamoDB 中的 Scan 操作會讀取資料表或次要索引中的每個項目。根據預設，Scan 操作會傳回資料表或索引中每個項目的所有資料屬性。您可以使用 ProjectionExpression 參數，以便 Scan 只傳回部分屬性，而不會全部傳回。

Scan 一律會傳回結果集。如果找不到相符項目，表示結果集是空的。

單一 Scan 請求最多可擷取 1 MB 的資料。或者，DynamoDB 可將篩選條件表達式套用至此資料，縮減結果後再傳回給使用者。

主題

- 掃描的篩選條件表達式
- 限制結果集的項目數
- 為結果編製分頁
- 計算結果中的項目
- 掃描使用的容量單位
- 掃描的讀取一致性
- 平行掃描

掃描的篩選條件表達式

若您需要更精確的 Scan 結果，您可以選擇性的提供篩選條件表達式。篩選條件表達式會判斷要傳回 Scan 結果中的哪些項目。所有其他結果皆會捨棄。

篩選條件表達式會在 Scan 完成後、結果傳回前套用。因此，無論是否有篩選條件表達式，Scan 都會使用相同數量的讀取容量。

Scan 操作最多可擷取 1 MB 的資料。系統會先套用這項限制，再評估篩選條件表達式。

使用 Scan，您可在篩選條件表達式中指定任何屬性，包括分割區索引鍵和排序索引鍵屬性。

篩選條件表達式的語法和條件表達式的語法相同。篩選條件表達式可以使用與條件表達式相同的比較子、函數和邏輯運算子。如需邏輯運算子的詳細資訊，請參閱 DynamoDB 中的條件表達式、篩選條件表達式、運算子和函數。。

Example

下列 AWS Command Line Interface (AWS CLI) 範例會掃描Thread資料表，並僅傳回特定使用者上次發佈到 的項目。

```
aws dynamodb scan \
     --table-name Thread \
     --filter-expression "LastPostedBy = :name" \
     --expression-attribute-values '{":name":{"S":"User A"}}'
```

限制結果集的項目數

Scan 操作可讓您限制在結果中傳回的項目數目。若要執行此作業，請將 Limit 參數設定為您希望 Scan 操作在篩選表達式評估前回傳的最大項目數。

例如，假設您 Scan 一份資料表，將 Limit 值設為 6 且不使用篩選條件表達式。此 Scan 結果會包含資料表中的前六個項目。

現在假設您在 Scan 中新增一個篩選條件表達式。在本案例中，DynamoDB 會將篩選條件表達式套用至傳回的六個項目，捨棄那些不符的項目。最終的 Scan 結果會包含 6 個或以下項目，視篩選的項目數而定。

為結果編製分頁

DynamoDB 會對 Scan 操作的結果進行分頁。透過編製分頁，Scan 結果會分成數個大小為 1 MB (或更小) 的資料「頁」。應用程式可以處理結果的第一頁、第二頁，以此類推。

單一 Scan 只會傳回符合 1 MB 大小限制的結果集。

為判斷是否有更多結果，並且一次擷取一頁結果，應用程式應執行下列作業：

1. 檢查低層級 Scan 結果：
   - 若結果包含 LastEvaluatedKey 元素，請接著進行步驟 2。
   - 若結果中沒有 LastEvaluatedKey，就表示再也沒有要擷取的項目。

2. 建構新的 Scan 請求，和前一個請求使用相同的參數。但這一次採用步驟 1 的 LastEvaluatedKey 值，並用它做為新 Scan 要求的 ExclusiveStartKey 參數。
3. 執行新的 Scan 請求。
4. 前往步驟 1。

換句話說，LastEvaluatedKey 回應的 Scan 應做為下一個 ExclusiveStartKey 請求的 Scan 使用。若 Scan 回應中沒有 LastEvaluatedKey 元素，表示您已擷取到結果的最終頁。(檢查是否沒有 LastEvaluatedKey 是確定您是否已到達結果集末頁的唯一方式)。

您可以使用 AWS CLI 來檢視此行為。會重複 AWS CLI 傳送低階Scan請求至 DynamoDB，直到結果LastEvaluatedKey中不再出現 為止。請考慮下列掃描整個Movies資料表，但僅傳回特定類型電影 AWS CLI 的範例。

```
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
```

一般而言， 會自動 AWS CLI 處理分頁。不過，在此範例中， AWS CLI --page-size 參數會限制每頁的項目數量。--debug 參數會列印請求及回應的下層資訊。

> **Note**
>
> 您的分頁結果也會根據您傳遞的輸入參數而有所不同。
>
> - 使用 aws dynamodb scan --table-name Prices --max-items 1 會傳回 NextToken
> - 使用 aws dynamodb scan --table-name Prices --limit 1 會傳回 LastEvaluatedKey。
>
> 另請注意，使用特定的 --starting-token 需要 NextToken 值。

如果您執行此範例，DynamoDB 的第一個回應會類似以下內容。

```
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}'
```

回應中的 LastEvaluatedKey 會指出並未擷取所有項目。 AWS CLI 然後， 向 DynamoDB 發出另一個Scan請求。此請求和回應模式會持續到最終回應出現為止。

```
2017-07-07 12:19:17,830 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"WarGames"}}],"ScannedCount":6}'
```

若沒有 LastEvaluatedKey，就表示已不再有要擷取的項目。

> **Note**
>
> AWS SDKs 會處理低階 DynamoDB 回應（包括是否存在 LastEvaluatedKey)，並提供用於分頁Scan結果的各種抽象概念。例如，適用於 Java 的開發套件文件介面會提供 java.util.Iterator 支援，讓您可一次處理一個結果。
> 如需各種程式設計語言的程式碼範例，請參閱《Amazon DynamoDB 入門指南》和所需語言的 AWS 軟體開發套件文件。

計算結果中的項目

除了符合您條件的項目之外，Scan 回應還包含了下列元素：

- ScannedCount：在套用任何 ScanFilter 前評估的項目數。ScannedCount 值很高，但 Count 結果很少或沒有，表示 Scan 操作不足。如未在請求中使用篩選條件，則 ScannedCount 與 Count 相同。
- Count：套用篩選條件表達式 (若有) 後剩餘的項目數。

> **Note**
>
> 若不使用篩選條件表達式，則 ScannedCount 和 Count 會有相同的值。

若 Scan 結果集的大小大於 1 MB，則 ScannedCount 和 Count 僅代表總項目的部分計數。您需要執行多項 Scan 操作，才能擷取所有的結果 (請參閱為結果編製分頁)。

每個 Scan 回應都包含經該特定 Scan 請求處理過的項目 ScannedCount 和 Count。若要取得所有 Scan 請求的總計，您可以為 ScannedCount 及 Count 記錄流水帳。

掃描使用的容量單位

您可以 Scan 任何資料表或次要索引。Scan 操作會使用讀取容量單位，如下所示。

若您對下列進行 **Scan**	DynamoDB 使用的讀取容量單位就會來自...
資料表	該資料表的佈建讀取容量。
全域次要索引	該索引的佈建讀取容量。
本機次要索引	該基礎資料表的佈建讀取容量。

> **Note**
>
> 資源型政策目前不支援次要索引掃描操作的跨帳戶存取。

根據預設，Scan 操作不會傳回任何使用之讀取容量的相關資料。但您可以在 ReturnConsumedCapacity 請求中指定 Scan 參數，來取得這項資訊。下列為 ReturnConsumedCapacity 的有效設定：

- NONE：不會傳回耗用的容量資料。(此為預設值)。
- TOTAL：回應包括耗用的讀取容量單位總數。
- INDEXES：回應顯示耗用的讀取容量單位總數，以及每個資料表和存取之索引的耗用容量。

DynamoDB 會根據項目的數量以及大小 (而不是根據傳回給應用程式的資料量) 計算耗用的讀取容量單位數。因此，無論您請求所有屬性 (預設行為) 或只請求部分屬性 (使用投影表達式)，使用的容量單位數都相同。無論您是否使用篩選條件表達式，此數量都相同。Scan 會耗用最小讀取容量單位，每秒執行一個高度一致性讀取；或者針對最大 4 KB 的項目，每秒執行兩個最終一致讀取。如果您需要讀取大於 4KB 的項目，DynamoDB 需要額外的讀取請求單位。空白資料表和擁有稀疏分割區索引鍵的超大型

資料表，可能會出現超出掃描資料量的額外 RCU 收費。這涵蓋了處理 Scan 請求的成本，即使資料不存在亦然。

掃描的讀取一致性

根據預設，Scan 操作會執行最終一致讀取。這表示 Scan 的結果可能不會反映最近完成之 PutItem 或 UpdateItem 操作所造成的變更。如需詳細資訊，請參閱 DynamoDB 讀取一致性。

若您要進行高度一致性讀取，請在 Scan 開始時，將 ConsistentRead 請求中的 true 參數設為 Scan。這可確保 Scan 開始前即已完成的所有寫入操作，都會包含在 Scan 回應中。

將 ConsistentRead 設為 true，再結合 DynamoDB Streams，在資料表備份或複寫案例中很實用。首先使用 ConsistentRead 設為 true 的 Scan，取得和資料表資料一致的副本。在 Scan 期間，DynamoDB Streams 會紀錄資料表發生的所有額外寫入活動。Scan 完成後，您可將串流的寫入活動套用至資料表。

> **Note**
>
> 與將 ConsistentRead 保留預設值 (false) 相較，將 ConsistentRead 設成 true 的 Scan 操作會使用兩倍的讀取容量單位。

平行掃描

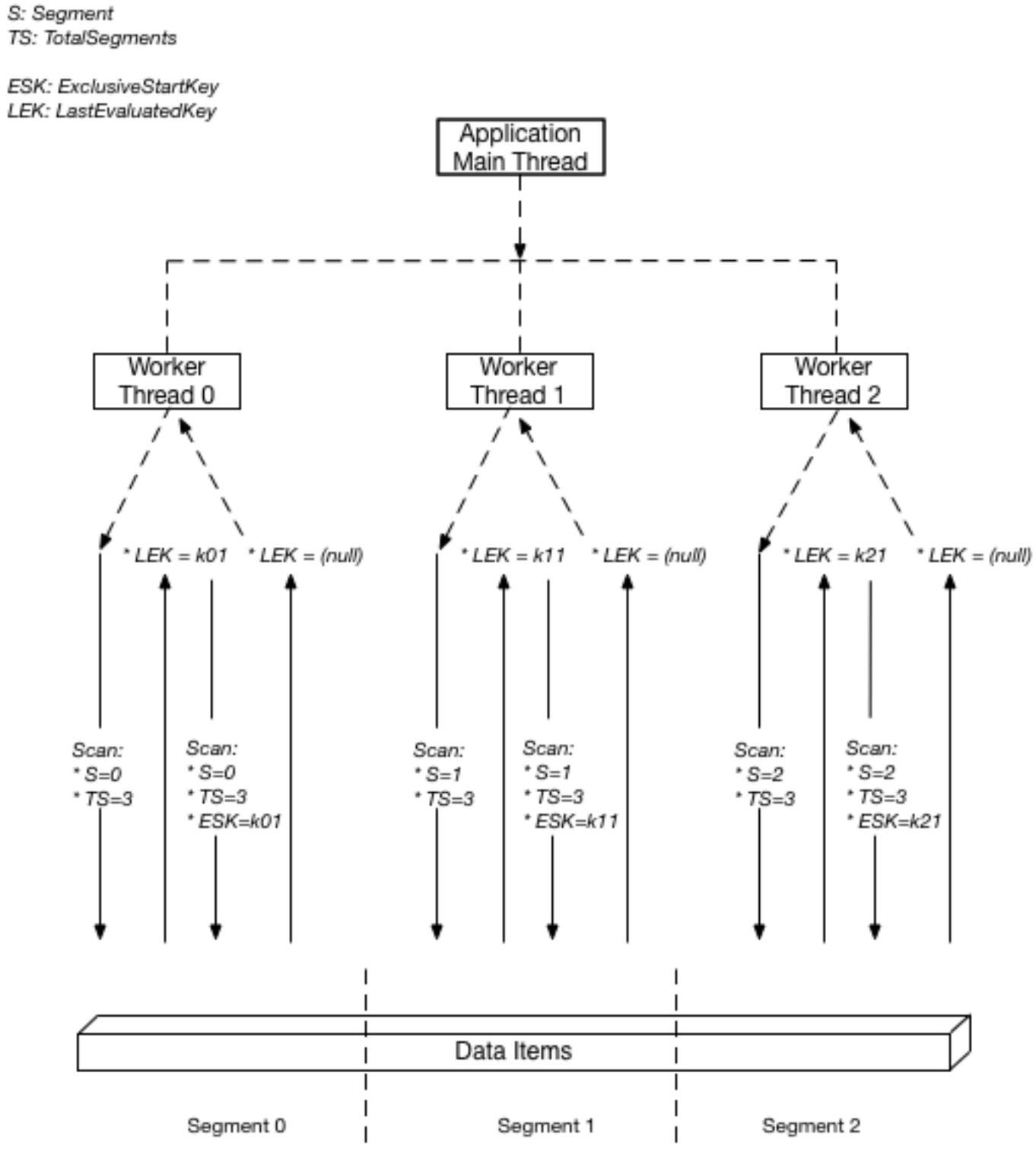
根據預設，Scan 操作會依序處理資料。Amazon DynamoDB 會以 1 MB 的增量將資料傳回應用程式，而應用程式會執行額外的 Scan 操作來擷取下一個 1 MB 的資料。

掃描的資料表或索引越大，Scan 完成所需的時間就更多。此外，循序 Scan 可能並不總是能夠完全使用佈建的讀取輸送容量：即使 DynamoDB 將大型資料表的資料分配到多個實體分割區，Scan 操作一次只能讀取一個分割區。因此，Scan 的輸送量受到單一分割區的最大輸送量限制。

若要解決這些問題，Scan 操作在邏輯上可能會將資料表或次要索引分為多個區段，其中有多個應用程式工作者平行掃描區段。每個工作者都可以是執行緒 (在支援多執行緒的程式設計語言中) 或作業系統程序。若要執行平行掃描，每個工作者都會使用下列參數發出自己的 Scan 請求：

- Segment：特定工作者要掃描的區段。每個工作者應該使用不同的 Segment 值。
- TotalSegments：平行掃描區段的總數。此值必須與應用程式要使用的工作者數目相同。

下圖顯示多執行緒應用程式如何以三度平行處理執行平行 Scan。



在此圖中，應用程式產生三個執行緒，並為每個執行緒指派一個數字。(區段從零開始，所以第一個數字永遠為 0。) 每個執行緒都會發出 Scan 請求、將 Segment 設定為指定號碼，並將

TotalSegments 設定為 3。每個執行緒都會掃描其指定的區段，一次擷取 1 MB 的資料，並將資料傳回至應用程式的主執行緒。

DynamoDB 透過將雜湊函數套用至每個項目的分割區索引鍵，將項目指派給區段。對於指定的TotalSegments值，具有相同分割區索引鍵的所有項目一律會指派給相同的 Segment。這表示在項目 1、項目 2 和項目 3 全部共用 pk="account#123"（但具有不同的排序索引鍵）的表格中，這些項目將由相同的工作者處理，無論排序索引鍵值或項目集合的大小為何。

由於區段指派僅以分割區索引鍵雜湊為基礎，因此區段可能分佈不均勻。某些區段可能不包含任何項目，而其他區段則可能包含具有大型項目集合的許多分割區索引鍵。因此，增加區段總數並不能保證更快的掃描效能，特別是當分割區索引鍵未均勻分佈到金鑰空間時。

Segment 和 TotalSegments 的值會套用至個別 Scan 請求，並且您可以隨時使用不同的值。您可能需要對這些值以及使用的工作者數量進行試驗，直到應用程式達到最佳效能為止。

> **Note**
>
> 具有大量工作者的平行掃描可以輕易耗用所掃描資料表或索引的所有佈建輸送量。如果資料表或索引也會造成來自其他應用程式的大量讀取或寫入活動，就最好避免這類掃描。
> 若要控制每個請求傳回的資料量，請使用 Limit 參數。這有助於預防某個工作者耗用所有佈建輸送量的情況，避免犧牲所有其他工作者。

PartiQL：一種適用於 Amazon DynamoDB 的 SQL 相容查詢語言

Amazon DynamoDB 支援 PartiQL (SQL 相容查詢語言)，可在 Amazon DynamoDB 中選取、插入、更新和刪除資料。使用 PartiQL，您可以輕鬆地與 DynamoDB 資料表互動 AWS Command Line Interface，並使用 PartiQL 的 AWS 管理主控台、NoSQL Workbench 和 DynamoDB APIs 執行臨機操作查詢。

PartiQL 操作提供與其他 DynamoDB 資料平面操作相同的可用性、延遲和效能。

下列各節將介紹 PartiQL 的 DynamoDB 實作。

主題

- 什麼是 PartiQL？
- Amazon DynamoDB 中的 PartiQL
- DynamoDB 專用 PartiQL 入門

- 適用於 DynamoDB 的 PartiQL 資料類型
- 適用於 DynamoDB 的 PartiQL 陳述式
- 搭配 DynamoDB 使用 PartiQL 函數
- 適用於 DynamoDB 的 PartiQL 算術、比較和邏輯運算子
- 使用 DynamoDB 專用 PartiQL 執行交易
- 使用 DynamoDB 專用 PartiQL 執行批次操作
- DynamoDB 專用 PartiQL 的 IAM 安全政策

什麼是 PartiQL？

PartiQL 在包含結構化資料、半結構化資料和巢狀資料的多個資料存放區提供與 SQL 相容的查詢存取。它在 Amazon 中廣泛使用，現在可做為許多 AWS 服務的一部分使用，包括 DynamoDB。

如需 PartiQL 規範和核心查詢語言的教學課程，請參閱 PartiQL 文件。

> **Note**
>
> - Amazon DynamoDB 支援 PartiQL 查詢語言的子集。
> - Amazon DynamoDB 不支援 Amazon Ion 資料格式或 Amazon Ion 文字。

Amazon DynamoDB 中的 PartiQL

若要在 DynamoDB 中執行 PartiQL 查詢，您可以使用：

- DynamoDB 主控台
- NoSQL Workbench
- The AWS Command Line Interface (AWS CLI)
- DynamoDB API

如需使用這些方法存取 DynamoDB 的相關資訊，請參閱存取 DynamoDB。

DynamoDB 專用 PartiQL 入門

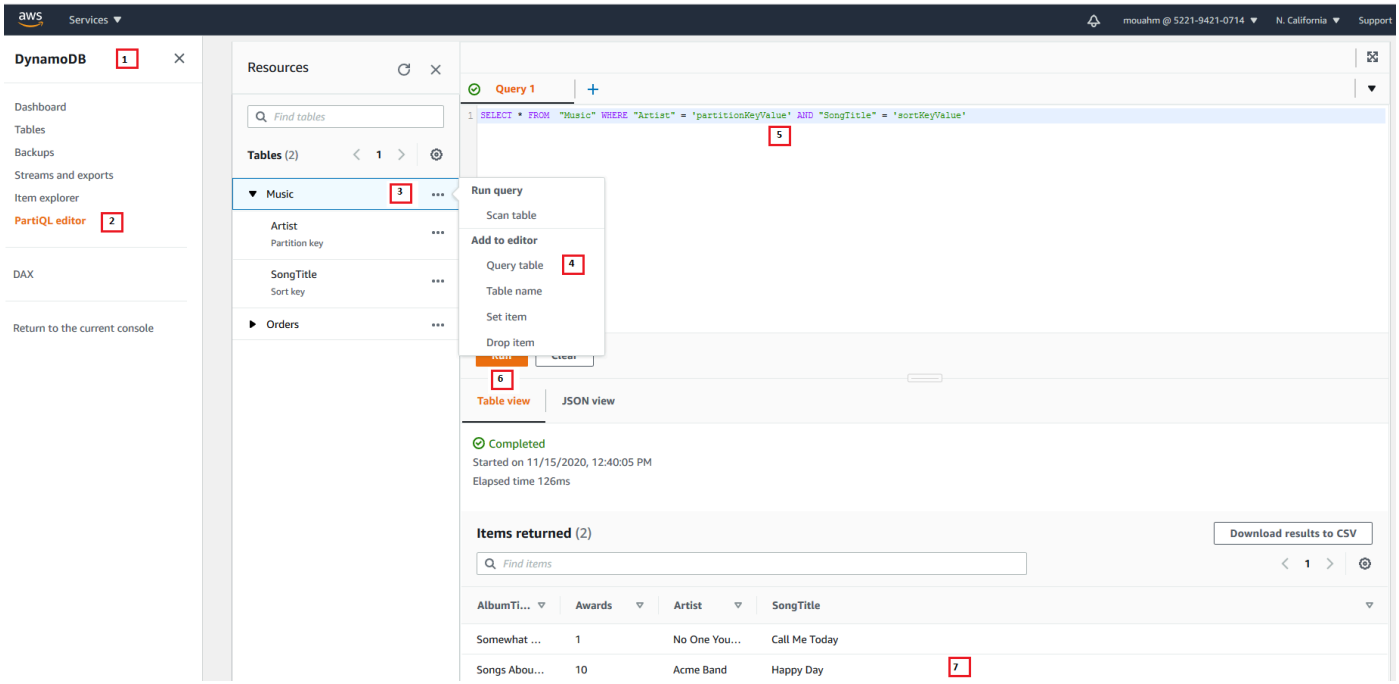
本節說明如何從 Amazon DynamoDB 主控台、 AWS Command Line Interface (AWS CLI) 和 DynamoDB API 使用 DynamoDB 專用 PartiQL。

在下列範例中，DynamoDB 入門教學課程中所定義的 DynamoDB 資料表是先決條件。

如需有關使用 DynamoDB 主控台 AWS Command Line Interface或 DynamoDB APIs 存取 DynamoDB 的資訊，請參閱存取 DynamoDB。

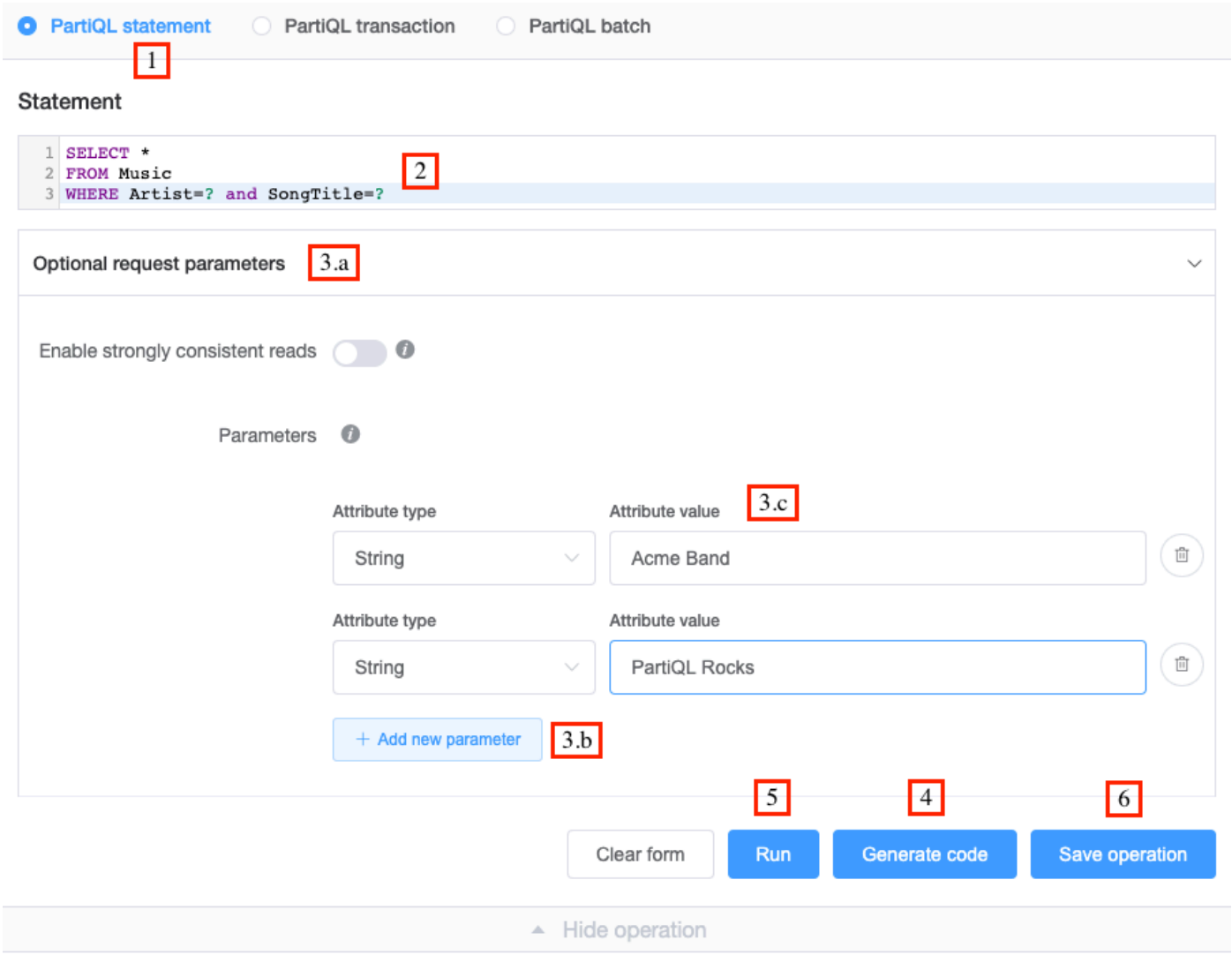
若要下載並使用 NoSQL Workbench 來建置 DynamoDB 專用 PartiQL 陳述式，請選擇位於 NoSQL Workbench for DynamoDB Operation Builder 右上角的 PartiQL operations (PartiQL 操作)。

Console



1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 PartiQL editor (PartiQL 編輯器)。
3. 選擇 Music (音樂) 資料表。
4. 選擇 Query table (查詢資料表)。此動作會產生不會導致完整資料表掃描的查詢。
5. 使用字串值 `Acme Band` 取代 `partitionKeyValue`。使用字串值 `Happy Day` 取代 `sortKeyValue`。
6. 選擇 Run (執行) 按鈕。
7. 您可以選擇 Table view (資料表檢視) 或 JSON view (JSON 檢視) 按鈕來檢視查詢的結果。

NoSQL workbench



1. 選擇 PartiQL statement (PartiQL 陳述式)。
2. 輸入下列 PartiQL SELECT 陳述式

```
SELECT *
FROM Music
WHERE Artist=? and SongTitle=?
```

3. 若要指定 `Artist` 和 `SongTitle` 參數的值：

   a. 選擇 Optional request parameters (選用的請求參數)。

   b. 選擇 Add new parameters (新增新參數)。

   c. 選擇屬性類型 string 和數值 `Acme Band`。

d. 重複步驟 b 和 c，然後選擇類型 string 和數值 `PartiQL Rocks`。

4. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

   從顯示的標籤中選取所需的語言。您現在可以複製此程式碼，並使用在您的應用程式中。

5. 若希望立即執行此操作，請選擇 Run (執行)。

AWS CLI

1. 使用 INSERT PartiQL 陳述式在 `Music` 資料表中建立項目。

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
         VALUE  \
         {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"
```

2. 使用 SELECT PartiQL 陳述式從 Music 資料表中檢索項目。

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

3. 使用 UPDATE PartiQL 陳述式在 `Music` 資料表中更新項目。

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardsWon=1  \
                                            SET AwardDetail={'Grammys':[2020,
 2018]}  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

   在 `Music` 資料表中新增項目的清單值。

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.Grammys
 =list_append(AwardDetail.Grammys,[2016])  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

   在 `Music` 資料表中移除項目的清單值。

```
aws dynamodb execute-statement --statement "UPDATE Music  \
```

```
                                            REMOVE AwardDetail.Grammys[2]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

在 Music 資料表中新增項目的新映射成員。

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.BillBoard=[2020]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

在 Music 資料表中新增項目的新字串集屬性。

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers =<<'member1',
 'member2'>>  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

在 Music 資料表中更新項目的字串集屬性。

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers
 =set_add(BandMembers, <<'newmember'>>)  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

4. 使用刪除 PartiQL 陳述式從 Music 資料表中刪除項目。

```
aws dynamodb execute-statement --statement "DELETE  FROM Music  \
    WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
```

Java

```
import java.util.ArrayList;
import java.util.List;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import software.amazon.dynamodb.AmazonDynamoDB;
```

```
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import software.amazon.dynamodb.model.AttributeValue;
import software.amazon.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.dynamodb.model.ExecuteStatementRequest;
import software.amazon.dynamodb.model.ExecuteStatementResult;
import software.amazon.dynamodb.model.InternalServerErrorException;
import software.amazon.dynamodb.model.ItemCollectionSizeLimitExceededException;
import software.amazon.dynamodb.model.ProvisionedThroughputExceededException;
import software.amazon.dynamodb.model.RequestLimitExceededException;
import software.amazon.dynamodb.model.ResourceNotFoundException;
import software.amazon.dynamodb.model.TransactionConflictException;

public class DynamoDBPartiQGettingStarted {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-1");

        try {
            // Create ExecuteStatementRequest
            ExecuteStatementRequest executeStatementRequest = new
 ExecuteStatementRequest();
            List<AttributeValue> parameters= getPartiQLParameters();

            //Create an item in the Music table using the INSERT PartiQL statement
            processResults(executeStatementRequest(dynamoDB, "INSERT INTO Music
 value {'Artist':?,'SongTitle':?}", parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
 statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
 where Artist=? and SongTitle=?", parameters));

            //Update an item in the Music table using the UPDATE PartiQL statement.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music
 SET AwardsWon=1 SET AwardDetail={'Grammys':[2020, 2018]}  where Artist=? and
 SongTitle=?", parameters));

            //Add a list value for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
 AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])  where Artist=? and
 SongTitle=?", parameters));

            //Remove a list value for an item in the Music table.
```

```
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music REMOVE
AwardDetail.Grammys[2]   where Artist=? and SongTitle=?", parameters));

            //Add a new map member for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist=? and SongTitle=?", parameters));

            //Add a new string set attribute for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music
SET BandMembers =<<'member1', 'member2'>> where Artist=? and SongTitle=?",
parameters));

            //update a string set attribute for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
BandMembers =set_add(BandMembers, <<'newmember'>>) where Artist=? and SongTitle=?",
parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
where Artist=? and SongTitle=?", parameters));

            //delete an item from the Music Table
            processResults(executeStatementRequest(dynamoDB, "DELETE  FROM Music
where Artist=? and SongTitle=?", parameters));
        } catch (Exception e) {
            handleExecuteStatementErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {
        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static List<AttributeValue> getPartiQLParameters() {
        List<AttributeValue> parameters = new ArrayList<AttributeValue>();
        parameters.add(new AttributeValue("Acme Band"));
        parameters.add(new AttributeValue("PartiQL Rocks"));
        return parameters;
    }

    private static ExecuteStatementResult executeStatementRequest(AmazonDynamoDB
client, String statement, List<AttributeValue> parameters ) {
        ExecuteStatementRequest request = new ExecuteStatementRequest();
```

```
        request.setStatement(statement);
        request.setParameters(parameters);
        return client.executeStatement(request);
    }

    private static void processResults(ExecuteStatementResult
executeStatementResult) {
        System.out.println("ExecuteStatement successful: "+
executeStatementResult.toString());

    }

    // Handles errors during ExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
    // your application use-case.
    private static void handleExecuteStatementErrors(Exception exception) {
        try {
            throw exception;
        } catch (ConditionalCheckFailedException ccfe) {
            System.out.println("Condition check specified in the operation failed,
review and update the condition " +
                                       "check before retrying. Error: " +
ccfe.getErrorMessage());
        } catch (TransactionConflictException tce) {
            System.out.println("Operation was rejected because there is an ongoing
transaction for the item, generally " +
                                       "safe to retry with exponential back-off.
Error: " + tce.getErrorMessage());
        } catch (ItemCollectionSizeLimitExceededException icslee) {
            System.out.println("An item collection is too large, you\'re using Local
Secondary Index and exceeded " +
                                       "size limit of items per
partition key. Consider using Global Secondary Index instead. Error: " +
icslee.getErrorMessage());
        } catch (Exception e) {
            handleCommonErrors(e);
        }
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
```

```
            System.out.println("Internal Server Error, generally safe to retry with
 exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
 your account, increase account level throughput before " +
                                       "retrying. Error: " +
 rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
 retry strategy make sure to retry with exponential back-off. " +
                                       "Otherwise consider reducing frequency of
 requests or increasing provisioned capacity for your table or secondary index.
 Error: " +
                                       ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
 before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
 the request was correctly transmitted to the DynamoDB " +
                                       "service, but for some reason, the service
 was not able to process it, and returned an error response instead. Investigate and
 " +
                                       "configure retry strategy. Error type: " +
 ase.getErrorType() + ". Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                                       "service, or the client was unable to parse
 the response from the service. Investigate and configure retry strategy. "+
                                       "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

使用參數化陳述式

您可以使用問號 (?) 預留位置，並在 Parameters 欄位中分別提供值，而不是直接在 PartiQL 陳述式字串中內嵌值。每個 ? 都會依提供的順序，以對應的參數值取代。

使用參數化陳述式是最佳實務，因為它會將陳述式結構與資料值分開，讓陳述式更容易閱讀和重複使用。它也不需要在陳述式字串中手動格式化和逸出屬性值。

ExecuteStatement、 BatchExecuteStatement和 ExecuteTransaction操作支援參數化陳述式。

下列範例使用分割區索引鍵和排序索引鍵的參數化值，從Music資料表中擷取項目。

AWS CLI parameterized

```
aws dynamodb execute-statement \
    --statement "SELECT * FROM \"Music\" WHERE Artist=? AND SongTitle=?" \
    --parameters '[{"S": "Acme Band"}, {"S": "PartiQL Rocks"}]'
```

Java parameterized

```
List<AttributeValue> parameters = new ArrayList<>();
parameters.add(new AttributeValue("Acme Band"));
parameters.add(new AttributeValue("PartiQL Rocks"));

ExecuteStatementRequest request = new ExecuteStatementRequest()
    .withStatement("SELECT * FROM Music WHERE Artist=? AND SongTitle=?")
    .withParameters(parameters);

ExecuteStatementResult result = dynamoDB.executeStatement(request);
```

Python parameterized

```
response = dynamodb_client.execute_statement(
    Statement="SELECT * FROM Music WHERE Artist=? AND SongTitle=?",
    Parameters=[
        {'S': 'Acme Band'},
        {'S': 'PartiQL Rocks'}
    ]
)
```

> **Note**
>
> 上一節中的 Java 範例使用參數化陳述式。`getPartiQLParameters()` 方法會建置參數清單，而每個陳述式會使用?預留位置而非內嵌值。

適用於 DynamoDB 的 PartiQL 資料類型

下表列出您可搭配 DynamoDB 專用 PartiQL 使用的資料類型。

DynamoDB 資料類型	PartiQL 表示法	備註
`Boolean`	TRUE | FALSE	不區分大小寫。
`Binary`	N/A	僅透過程式碼支援。
`List`	[value1, value2,...]	清單類型中可存放的資料類型不限，而且清單中的元素不必屬於相同類型。
`Map`	{ 'name' : value }	映射類型中可存放的資料類型不限，而且映射中的元素不必屬於相同類型。
`Null`	NULL	不區分大小寫。
`Number`	1、1.0、1e0	數字可以是正數、負數或零。數字精確度最高可達 38 位數。
`Number Set`	<<number1, number2>>	數字集中的元素必須是 Number (數字) 類型。
`String Set`	<<'string1', 'string2'>>	字串集中的元素必須是 String (字串) 類型。
`String`	'string value'	必須使用單引號來指定 String (字串) 值。

範例

下述陳述式示範如何插入以下資料類型：String、Number、Map、List、Number Set 和 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'>>
}
```

下述陳述式示範如何將新的元素插入 Map、List、Number Set 和 String Set 類型，並變更 Number 類型的數值。

```
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'
```

下述陳述式示範如何從 Map、List、Number Set 和 String Set 類型移除元素，並變更 Number 類型的數值。

```
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'
```

如需詳細資訊，請參閱 DynamoDB 資料類型。

適用於 DynamoDB 的 PartiQL 陳述式

Amazon DynamoDB 支援下列 PartiQL 陳述式。

> **Note**
>
> DynamoDB 不支援所有 PartiQL 陳述式。
> 此參考提供使用 AWS CLI 或 APIs 手動執行之 PartiQL 陳述式的基本語法和使用範例。

資料操作語言 (DML) 是一組用來管理 DynamoDB 資料表中資料的 PartiQL 陳述式。您可使用 DML 陳述式新增、修改或刪除資料表中的資料。

支援下列 DML 和查詢語言陳述式：

- 適用於 DynamoDB 的 PartiQL Select 陳述式
- 適用於 DynamoDB 的 PartiQL Update 陳述式
- 適用於 DynamoDB 的 PartiQL Insert 陳述式
- 適用於 DynamoDB 的 PartiQL Delete 陳述式

DynamoDB 專用 PartiQL 亦支援 使用 DynamoDB 專用 PartiQL 執行交易 和 使用 DynamoDB 專用 PartiQL 執行批次操作。

適用於 DynamoDB 的 PartiQL Select 陳述式

在 Amazon DynamoDB 中，使用 SELECT 陳述式從資料表檢索資料。

如果 WHERE 子句中未提供與分割區索引鍵相等或 IN 條件，使用 SELECT 陳述式可能會導致完整的資料表掃描。掃描操作會檢查每個項目的要求值，並可能會在單一操作中耗用大型資料表或索引的佈建輸送量。

如果您想避免在 PartiQL 中進行完整的資料表掃描，則可以：

- 確保 WHERE 子句條件會據此進行相應設定，撰寫您的 SELECT 陳述式便不會導致完整的資料表掃描。
- 請使用《DynamoDB 開發人員指南》中 範例：允許在 DynamoDB 專用 PartiQL 中執行選取陳述式並拒絕完整的資料表掃描陳述式 所述的 IAM 政策來停用完整的資料表掃描。

如需詳細資訊，請參閱《DynamoDB 開發人員指南》中的查詢和掃描資料的最佳實務。

主題

- 語法

- Parameters
- 範例

語法

```
SELECT expression  [, ...]
FROM table[.index]
[ WHERE condition ] [ [ORDER BY key [DESC|ASC] , ...]
```

Parameters

###

(必要) 從 * 萬用字元形成的投影，或來自結果集中一或多個屬性名稱或文件路徑的投影清單。一個表達式可以包含對 搭配 DynamoDB 使用 PartiQL 函數 的呼叫或由 適用於 DynamoDB 的 PartiQL 算術、比較和邏輯運算子 修改的欄位。

###

(必要) 要查詢的資料表名稱。

##

(選用) 要查詢的索引名稱。

> **Note**
>
> 查詢索引時，必須在資料表名稱和索引名稱上加上雙引號。
>
> ```
> SELECT *
> FROM "TableName"."IndexName"
> ```

condition

(選用) 查詢的選取條件。

> **Important**
>
> 若要確保 SELECT 陳述式不會導致完整的資料表掃描，WHERE 子句條件必須指定分割區索引鍵。使用等於或 IN 運算子。

例如，如果一個 Orders 資料表包含 OrderID 分割區索引鍵和其他非索引鍵屬性 (包括 Address)，則下列陳述式不會導致完整的資料表掃描：

```
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]
```

不過，下列 SELECT 陳述式會導致完整的資料表掃描：

```
SELECT *
FROM "Orders"
WHERE OrderID > 1

SELECT *
FROM "Orders"
WHERE Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 OR Address='some address'
```

##

(選用) 用來排序傳回結果的雜湊索引鍵或排序索引鍵。預設順序是升序 (ASC)；如果您希望依降序傳回結果，請指定 DESC。

> **Note**
>
> 如果您省略 WHERE 子句，則會檢索資料表中的所有項目。

範例

如果存在一個項目，則下列查詢會指定分割區索引鍵和 OrderID 並使用等於運算子，從 Orders 資料表傳回一個項目。

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1
```

下列查詢會針對具有特定分割區索引鍵、OrderID 和值的 Orders 資料表，使用 OR 運算子傳回其中的所有項目。

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1 OR OrderID = 2
```

下列查詢會針對具有特定分割區索引鍵、OrderID 和值的 Orders 資料表，使用 IN 運算子傳回其中的所有項目。傳回的結果會根據 OrderID 索引鍵屬性值以遞減順序顯示。

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID IN [1, 2, 3] ORDER BY OrderID DESC
```

下列查詢顯示完整的資料表掃描，該掃描會傳回 Orders 資料表中 Total 大於 500 的所有項目，其中 Total 是非索引鍵屬性。

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total > 500
```

下列查詢顯示完整的資料表掃描，該掃描會傳回使用 IN 運算子和非索引鍵屬性 Total 之特定 Total 順序範圍內 Orders 資料表中的所有項目。

```
SELECT OrderID, Total
```

```
FROM "Orders"
WHERE Total IN [500, 600]
```

下列查詢顯示完整的資料表掃描，該掃描會傳回使用 BETWEEN 運算子和非索引鍵屬性 `Total` 之特定 `Total` 順序範圍內 `Orders` 資料表中的所有項目。

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total BETWEEN 500 AND 600
```

下列查詢會在 WHERE 子句條件中指定分割區索引鍵 `CustomerID` 和排序索引鍵 `MovieID`，並在 SELECT 子句中使用文件路徑，以此傳回 FireStick 裝置用於觀看的第一個日期。

```
SELECT Devices.FireStick.DateWatched[0]
FROM WatchList
WHERE CustomerID= 'C1' AND MovieID= 'M1'
```

下列查詢會顯示完整的資料表掃描，該掃描會使用 WHERE 子句條件中的文件路徑傳回在 2019 年 12 月 24 日之後首次使用 FireStick 裝置的項目清單。

```
SELECT Devices
FROM WatchList
WHERE Devices.FireStick.DateWatched[0] >= '12/24/19'
```

適用於 DynamoDB 的 PartiQL Update 陳述式

使用 UPDATE 陳述式來修改 Amazon DynamoDB 資料表中項目內一或多個屬性的值。

> **Note**
>
> 您一次只能更新一個項目，因為您無法發出可刪除多個項目的單個 DynamoDB PartiQL 陳述式。如需更新多個項目的相關資訊，請參閱 使用 DynamoDB 專用 PartiQL 執行交易 或 使用 DynamoDB 專用 PartiQL 執行批次操作。

主題

- 語法
- Parameters

- 傳回值
- 範例

語法

```
UPDATE  table
[SET | REMOVE]  path  [=  data] […]
WHERE condition [RETURNING returnvalues]
<returnvalues>  ::= [ALL OLD | MODIFIED OLD | ALL NEW | MODIFIED NEW] *
```

Parameters

###

(必要) 包含要修改之資料的資料表。

##

(必要) 要建立或修改的屬性名稱或文件路徑。

##

(必要) 屬性值或操作的結果。

與 SET 搭配使用的支援操作：

- LIST_APPEND：將數值新增至清單類型。
- SET_ADD：將數值新增至數字或字串集。
- SET_DELETE：從數字或字串集中刪除數值。

condition

(必要) 要修改之項目的選取條件。此條件必須解析為單一主索引鍵值。

returnvalues

(選用) 若想取得更新之前或之後出現的項目屬性，則請使用 `returnvalues`。有效值為：

- `ALL OLD *`：傳回更新操作之前出現的所有項目屬性。
- `MODIFIED OLD *`：僅傳回新操作之前出現的更新屬性。
- `ALL NEW *`：傳回更新操作之後出現的所有項目屬性。
- `MODIFIED NEW *`：僅傳回 `UpdateItem` 操作之後出現的更新屬性。

傳回值

此陳述式不會傳回值，除非指定 `returnvalues` 參數。

> **Note**
>
> 如果 DynamoDB 資料表中任何項目的 UPDATE 陳述式的 WHERE 子句未評估為 true，則會傳回 `ConditionalCheckFailedException`。

範例

更新現有項目中的屬性值。如果屬性不存在，則會建立此屬性。

下列查詢會透過新增數字類型的屬性 (`AwardsWon`) 和映射類型的屬性 (`AwardDetail`) 來更新 `"Music"` 資料表中的項目。

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

您可以新增 `RETURNING ALL OLD *` 以傳回 `Update` 操作前顯示的屬性。

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL OLD *
```

這會傳回下列內容：

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
```

```
        }
    ]
}
```

您可以新增 RETURNING ALL NEW * 以傳回 Update 操作後顯示的屬性。

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL NEW *
```

這會傳回下列內容：

```
{
    "Items": [
        {
            "AwardDetail": {
                "M": {
                    "Grammys": {
                        "L": [
                            {
                                "N": "2020"
                            },
                            {
                                "N": "2018"
                            }
                        ]
                    }
                }
            },
            "AwardsWon": {
                "N": "1"
            }
        }
    ]
}
```

下列查詢會透過附加至清單 AwardDetail.Grammys 來更新 "Music" 中的項目。

```
UPDATE "Music"
SET AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])
```

```
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

下列查詢會透過從清單 AwardDetail.Grammys 中移除來更新 "Music" 資料表中的項目。

```
UPDATE "Music"
REMOVE AwardDetail.Grammys[2]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

下列查詢會透過將 BillBoard 新增至映射 AwardDetail 來更新 "Music" 資料表中的項目。

```
UPDATE "Music"
SET AwardDetail.BillBoard=[2020]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

下列查詢會透過新增字串集屬性 BandMembers 來更新 "Music" 資料表中的項目。

```
UPDATE "Music"
SET BandMembers =<<'member1', 'member2'>>
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

下列查詢會透過將 newbandmember 新增至字串集 BandMembers 來更新 "Music" 資料表中的項目。

```
UPDATE "Music"
SET BandMembers =set_add(BandMembers, <<'newbandmember'>>)
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

適用於 DynamoDB 的 PartiQL Delete 陳述式

使用 DELETE 陳述式從您的 Amazon DynamoDB 資料表中刪除現有項目。

> **Note**
>
> 您一次只能刪除一個項目。您無法發出可刪除多個項目的單個 DynamoDB PartiQL 陳述式。如需刪除多個項目的相關資訊，請參閱 使用 DynamoDB 專用 PartiQL 執行交易 或 使用 DynamoDB 專用 PartiQL 執行批次操作。

主題

- 語法

- Parameters
- 傳回值
- 範例

語法

```
DELETE FROM table
 WHERE condition [RETURNING returnvalues]
 <returnvalues>  ::= ALL OLD *
```

Parameters

###

(必要) 包含要刪除之項目的 DynamoDB 資料表。

condition

(必要) 要刪除之項目的選取條件；此條件必須解析為單一主索引鍵值。

returnvalues

(選用) 若想取得在刪除之前出現的項目屬性，則請使用 `returnvalues`。有效值為：

- `ALL OLD *`：傳回舊項目的內容。

傳回值

此陳述式不會傳回值，除非指定 `returnvalues` 參數。

> **Note**
>
> 如果 DynamoDB 資料表沒有任何與發出 DELETE 的項目具有相同主索引鍵的項目，則會傳回 SUCCESS，並刪除 0 個項目。如果資料表具有相同主索引鍵的項目，但 DELETE 陳述式的 WHERE 子句中的條件評估結果為 false，則會傳回 `ConditionalCheckFailedException`。

範例

下列查詢會刪除 `"Music"` 資料表中的一個項目。

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
```

您可以新增參數 `RETURNING ALL OLD *` 以傳回已刪除的資料。

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
 RETURNING ALL OLD *
```

`Delete` 陳述式現在傳回下列內容：

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

適用於 DynamoDB 的 PartiQL Insert 陳述式

使用 `INSERT` 陳述式在 Amazon DynamoDB 中將項目新增至資料表。

> **Note**
>
> 您一次只能插入一個項目，因為您無法發出可插入多個項目的單個 DynamoDB PartiQL 陳述式。如需插入多個項目的相關資訊，請參閱 使用 DynamoDB 專用 PartiQL 執行交易 或 使用 DynamoDB 專用 PartiQL 執行批次操作。

主題

- 語法
- Parameters
- 傳回值
- 範例

語法

插入單一項目。

```
INSERT INTO table VALUE item
```

Parameters

###

(必要) 您要插入資料的資料表。索資料表必須已存在。

##

(必要) 有效的 DynamoDB 項目表示為 PartiQL 元組。您必須在 PartiQL 中僅指定一個項目，項目中的每個屬性名稱均區分大小寫，且可以使用單引號 (`'...'`) 表示。

PartiQL 中的字串值也使用單引號 (`'...'`) 表示。

傳回值

此陳述式不會傳回任何值。

> **Note**
>
> 如果 DynamoDB 資料表已經有一個項目的主索引鍵與插入的項目的主索引鍵相同，則會傳回 `DuplicateItemException`。

範例

```
INSERT INTO "Music" value {'Artist' : 'Acme Band','SongTitle' : 'PartiQL Rocks'}
```

搭配 DynamoDB 使用 PartiQL 函數

Amazon DynamoDB 中的 PartiQL 支援下列 SQL 標準函數的內建變體。

> **Note**
>
> DynamoDB 目前不支援任何未包含在此清單中的 SQL 函數。

彙總函數

- 搭配 Amazon DynamoDB 專用 PartiQL 使用 SIZE 函數

條件函數

- 搭配 DynamoDB 專用 PartiQL 使用 EXISTS 函數
- 搭配 DynamoDB 專用 PartiQL 使用 ATTRIBUTE_TYPE 函數
- 搭配 DynamoDB 專用 PartiQL 使用 BEGINS_WITH 函數
- 搭配 DynamoDB 專用 PartiQL 使用 CONTAINS 函數
- 搭配 DynamoDB 專用 PartiQL 使用 MISSING 函數

搭配 DynamoDB 專用 PartiQL 使用 EXISTS 函數

您可以使用 EXISTS 來執行與 `ConditionCheck` 會在 TransactWriteItems API 中執行的相同函數。EXISTS 函數只能在交易中使用。

給定一個值，如果該值是一個非空集合，則傳回 `TRUE`。如果不是，則傳回 `FALSE`。

> **Note**
>
> 此函數只能在交易操作中使用。

語法

```
EXISTS ( statement )
```

引數

###

(必要) 函數評估的 SELECT 陳述式。

> **Note**
>
> SELECT 陳述式必須指定完整的主索引鍵和其他一個條件。

傳回類型

bool

範例

```
EXISTS(
    SELECT * FROM "Music"
    WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks')
```

搭配 DynamoDB 專用 PartiQL 使用 BEGINS_WITH 函數

如果指定的屬性以特定子字串開頭，則傳回 TRUE。

語法

```
begins_with(path, value )
```

引數

##

(必要) 要使用的屬性名稱或文件路徑。

#

(必要) 要搜尋的字串。

傳回類型

bool

範例

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND begins_with("Address", '7834 24th')
```

搭配 DynamoDB 專用 PartiQL 使用 MISSING 函數

如果項目不包含指定的屬性，即傳回 TRUE。只有等於和不等於運算子可以與此函數一起使用。

語法

```
 attributename IS | IS NOT  MISSING
```

引數

attributename

(必要) 要尋找的屬性名稱。

傳回類型

bool

範例

```
SELECT * FROM Music WHERE "Awards" is MISSING
```

搭配 DynamoDB 專用 PartiQL 使用 ATTRIBUTE_TYPE 函數

如果指定路徑的屬性是特定資料類型，則傳回 TRUE。

語法

```
attribute_type( attributename, type )
```

引數

attributename

(必要) 要使用的屬性名稱。

##

(必要) 要檢查的屬性類型。如需有效值的清單，請參閱 DynamoDB attribute_type。

傳回類型

bool

範例

```
SELECT * FROM "Music" WHERE attribute_type("Artist", 'S')
```

搭配 DynamoDB 專用 PartiQL 使用 CONTAINS 函數

如果路徑指定的屬性為下列其中一個項目，則傳回 TRUE：

- 包含特定子字串的字串。
- 在組合中內含特定元素的組合。

如需詳細資訊，請參閱 DynamoDB 的 contains 函數。

語法

```
contains( path, substring )
```

引數

##

(必要) 要使用的屬性名稱或文件路徑。

###

(必要) 要檢查的屬性子字串或集合成員。如需詳細資訊，請參閱 DynamoDB 的 contains 函數。

傳回類型

bool

範例

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND contains("Address", 'Kirkland')
```

搭配 Amazon DynamoDB 專用 PartiQL 使用 SIZE 函數

傳回代表屬性大小的數字 (以位元組為單位)。以下是可搭配 size 使用的有效資料類型。如需詳細資訊，請參閱 DynamoDB 的 size 函數。

語法

```
size( path)
```

引數

##

(必要) 屬性名稱或文件路徑。

如需了解支援的類型，請參閱 DynamoDB size 函數。

傳回類型

int

範例

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND size("Image") >300
```

適用於 DynamoDB 的 PartiQL 算術、比較和邏輯運算子

Amazon DynamoDB 中的 PartiQL 支援以下 SQL 標準運算子。

> **Note**
>
> DynamoDB 目前不支援任何未包含在此清單中的 SQL 運算子。

算術運算子

運算子	Description
+	加
-	減

比較運算子

運算子	Description
=	等於

運算子	Description
<>	不等於
!=	不等於
>	Greater than
<	Less than
>=	大於或等於
<=	小於或等於

邏輯運算子

運算子	Description
AND	如果以 AND 分隔的所有的條件為 TRUE，則為 TRUE
BETWEEN	如果運算元在比較範圍內，則為 TRUE。 此運算子包含您套用運算元的下限和上限。
IN	TRUE 如果運算元等於表達式清單之一 （上限為 50 個雜湊屬性值或上限為 100 個非索引鍵屬性值）。 結果會以最多 10 個項目的頁面傳回。如果IN清單包含更多值，您必須使用回應中NextToken 傳回的 來擷取後續頁面。
IS	如果運算元是給定的 PartiQL 資料類型 (包括 NULL 或 MISSING)，則為 TRUE
NOT	反轉指定布林表達式的值
OR	如果以 OR 分隔任何的條件為 TRUE，則為 TRUE

如需邏輯運算子的詳細資訊，請參閱 進行比較 和 邏輯評估。

使用 DynamoDB 專用 PartiQL 執行交易

本節說明如何搭配 DynamoDB 專用 PartiQL 使用交易。PartiQL 交易限制為全部 100 個陳述式 (動作)。

如需 DynamoDB 交易的詳細資訊，請參閱管理包含 DynamoDB 交易的複雜工作流程。

> **Note**
>
> 整個交易必須由讀取陳述式或寫入陳述式所組成。你不能在一個交易中混合兩者。EXISTS 函數為例外。可用類似於 TransactWriteItems API 操作中的 `ConditionCheck` 方式檢查項目特定屬性的條件。

主題

- 語法
- Parameters
- 傳回值
- 範例

語法

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parameters

###

(必要) DynamoDB 專用 PartiQL 支援的陳述式。

> **Note**
>
> 整個交易必須由讀取陳述式或寫入陳述式所組成。你不能在一個交易中混合兩者。

parametertype

(選用) DynamoDB 類型，如果在指定 PartiQL 陳述式時使用了參數。

parametervalue

(選用) 參數值，如果在指定 PartiQL 陳述式時使用了參數。

傳回值

此陳述式不會傳回寫入作業 (插入、更新或刪除) 的任何值。但是，它根據 WHERE 子句中指定的條件回傳讀取操作 (SELECT) 的不同值。

> **Note**
>
> 如果任何單一 INSERT、UPDATE 或 DELETE 操作傳回錯誤，交易會取消並出現 `TransactionCanceledException` 例外狀況，且取消原因程式碼會包含個別單一操作的錯誤。

範例

以下範例會以交易形式執行多個陳述式。

AWS CLI

1. 將以下 JSON 程式碼儲存至名為 partiql.json 的檔案。

```
[
    {
        "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. 在命令提示中執行下列命令。

```
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());
```

```
        }
    }

}
```

下列範例顯示當 DynamoDB 讀取 WHERE 子句中指定之不同條件的項目時，不同的傳回值。

AWS CLI

1. 將以下 JSON 程式碼儲存至名為 partiql.json 的檔案。

```
[
    // 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. 在命令提示中執行命令。

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

3. 會傳回下列回應：

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

使用 DynamoDB 專用 PartiQL 執行批次操作

本節說明如何搭配 DynamoDB 專用 PartiQL 使用批次陳述式。

> **Note**
>
> - 整個批次必須由讀取陳述式或寫入陳述式所組成；一個批次中不能同時出現這兩種陳述式樣。
> - `BatchExecuteStatement` 和 `BatchWriteItem` 每個批次不可以執行超過 25 條陳述式。
> - `BatchExecuteStatement` 會使用 `BatchGetItem`，其會取得個別陳述式中主索引鍵的清單。

主題

- 語法
- Parameters
- 範例

語法

```
[
  {
```

```
    "Statement": "SELECT pk FROM ProblemSet WHERE pk = 'p#9StkWHYTxm7x2AqSXcrfu7' AND
 sk = 'info'"
  },
  {
    "Statement": "SELECT pk FROM ProblemSet WHERE pk = 'p#isC2ChceGbxHgESc4szoTE' AND
 sk = 'info'"
  }
]
```

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parameters

###

(必要) DynamoDB 專用 PartiQL 支援的陳述式。

> **Note**
>
> - 整個批次必須由讀取陳述式或寫入陳述式所組成；一個批次中不能同時出現這兩種陳述式樣。
> - `BatchExecuteStatement` 和 `BatchWriteItem` 每個批次不可以執行超過 25 條陳述式。

parametertype

(選用) DynamoDB 類型，如果在指定 PartiQL 陳述式時使用了參數。

parametervalue

(選用) 參數值，如果在指定 PartiQL 陳述式時使用了參數。

範例

AWS CLI

1. 將以下 json 儲存至名為 partiql.json 的檔案

```
[
   {
  "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. 在命令提示中執行下列命令。

```
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());
        }
    }

}
```

DynamoDB 專用 PartiQL 的 IAM 安全政策

需要以下許可：

- 若要使用 DynamoDB 專用 PartiQL 讀取項目，您必須擁有資料表或索引的 `dynamodb:PartiQLSelect` 許可。
- 若要使用 DynamoDB 專用 PartiQL 插入項目，您必須擁有資料表或索引的 `dynamodb:PartiQLInsert` 許可。
- 若要使用 DynamoDB 專用 PartiQL 更新項目，您必須擁有資料表或索引的 `dynamodb:PartiQLUpdate` 許可。
- 若要使用 DynamoDB 專用 PartiQL 刪除項目，您必須擁有資料表或索引的 `dynamodb:PartiQLDelete` 許可。

範例：允許在資料表上執行所有 DynamoDB 專用 PartiQL 陳述式 (選取/插入/更新/刪除)

下列 IAM 政策會授予許可，允許在資料表上執行所有 DynamoDB 專用 PartiQL 陳述式。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

範例：允許在資料表上執行 DynamoDB 專用 PartiQL 選取陳述式

下列 IAM 政策會授予許可，允許在特定資料表上執行 `select` 陳述式。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

範例：允許在索引上執行 DynamoDB 專用 PartiQL 插入陳述式

下列 IAM 政策會授予許可，允許在特定索引上執行 `insert` 陳述式。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music/index/index1"
         ]
      }
   ]
```

```
}
```

範例：允許在資料表上僅執行 DynamoDB 專用 PartiQL 交易陳述式

下列 IAM 政策會授予許可，允許僅在特定資料表上執行交易陳述式。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      }
   ]
}
```

範例：允許在資料表上執行 DynamoDB 專用 PartiQL 非交易式讀取和寫入，並封鎖 PartiQL 交易式讀取和寫入交易陳述式。

下列 IAM 政策會授予許可，允許執行 DynamoDB 專用 PartiQL 非交易式讀取和寫入，同時封鎖 DynamoDB 專用 PartiQL 交易式讀取和寫入。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

範例：允許在 DynamoDB 專用 PartiQL 中執行選取陳述式並拒絕完整的資料表掃描陳述式

下列 IAM 政策會授予許可，允許在特定資料表上執行 `select` 陳述式，同時封鎖會導致完整資料表掃描的 `select` 陳述式。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
         ],
         "Condition":{
            "Bool":{
               "dynamodb:FullTableScan":[
                  "true"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
         ]
      }
   ]
}
```

使用項目：Java

您可以使用 適用於 Java 的 AWS SDK 文件 API 在資料表中的 Amazon DynamoDB 項目上執行典型的建立、讀取、更新和刪除 (CRUD) 操作。

> **Note**
>
> 適用於 Java 的開發套件也提供物件持久性模型，讓您將用戶端類別映射至 DynamoDB 資料表。此方法可以減少您必須撰寫的程式碼數量。如需詳細資訊，請參閱 Java 1.x：DynamoDBMapper。

本節包含 Java 範例，執行數個 Java 文件 API 項目動作及數個完整的工作範例。

主題

- 放入項目
- 取得項目
- 批次寫入：放入和刪除多個項目
- 批次取得：取得多個項目
- 更新項目
- 刪除項目
- 範例：使用 適用於 Java 的 AWS SDK 文件 API 的 CRUD 操作
- 範例：使用 適用於 Java 的 AWS SDK 文件 API 的批次操作
- 範例：使用 適用於 Java 的 AWS SDK 文件 API 處理二進位類型屬性

放入項目

`putItem` 方法會將項目存放在資料表中。若已有該項目，其會取代整個項目。若不希望取代整個項目，而是只想要更新特定的屬性，可以使用 `updateItem` 方法。如需詳細資訊，請參閱 更新項目。

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

請遵循下列步驟：

1. 建立 DynamoDB 類別的執行個體。
2. 建立 Table 類別的執行個體，代表您要進行作業的資料表。
3. 建立 Item 類別的執行個體，代表新的項目。您必須指定新項目的主索引鍵及其屬性。
4. 使用您在前述步驟中所建立的 putItem，呼叫 Table 物件的 Item 方法。

下列 Java 程式碼範例示範上述工作。程式碼會將新的項目寫入 ProductCatalog 表。

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);
```

在前述範例中，項目具有的屬性為純量 (String、Number、Boolean、Null)、集合 (String Set) 及文件類型 (List、Map)。

指定選用參數

除了必要的參數之外，您也可為 putItem 方法指定選用的參數。例如，下列 Java 程式碼範例會使用選用參數來指定上傳項目的條件。如果您指定的條件不符合， 會 適用於 Java 的 AWS SDK 擲回 ConditionalCheckFailedException。程式碼範例在 putItem 方法中指定了下列選用參數：

- ConditionExpression 會定義要求的條件。程式碼定義的條件是，具有相同主索引鍵的現有項目，只有在其 ISBN 屬性等於特定值時，才會被取代。
- 用於條件中的 ExpressionAttributeValues 映射。在本案例中，只需要一次替換：條件表達式中的預留位置 :val，會在執行時期更換為實際要查看的 ISBN 值。

下列範例會使用這些選用參數，新增新的書籍項目。

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 與 JSON 文件

您可以將 JSON 文件以屬性形式存放在 DynamoDB 資料表中。若要執行此作業，請使用 Item 的 withJSON 方法。此方法會剖析 JSON 文件，並將每個元素映射到原生的 DynamoDB 資料類型。

假設您希望存放下列 JSON 文件，其包含可完成特定產品訂單的廠商。

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"
        ]
    }
}
```

您可以使用 withJSON 方法，將此存放在 ProductCatalog 表，放在名為 VendorInfo 的 Map 屬性中。下列 Java 程式碼範例會示範如何執行此作業。

```
// 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);
```

取得項目

若要擷取單一項目，請使用 getItem 物件的 Table 方法。請遵循下列步驟：

1. 建立 DynamoDB 類別的執行個體。
2. 建立 Table 類別的執行個體，代表您要進行作業的資料表。
3. 呼叫 getItem 執行個體的 Table 方法。您必須指定希望擷取之項目的主索引鍵。

下列 Java 程式碼範例示範上述步驟。程式碼會取得具有指定分割區索引鍵的項目。

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Item item = table.getItem("Id", 210);
```

指定選用參數

除了必要的參數之外，您也可為 getItem 方法指定選用的參數。例如，下列 Java 程式碼範例使用選用方法，只擷取特定的屬性清單，以及指定高度一致性讀取。(若要進一步了解讀取一致性，請參閱「DynamoDB 讀取一致性」。)

您可以使用 ProjectionExpression，只擷取特定的屬性或元素，而非整個項目。ProjectionExpression 可以使用文件路徑，指定最上層或巢狀屬性。如需詳細資訊，請參閱在 DynamoDB 中使用投影表達式。

getItem 方法的參數不會讓您指定讀取一致性。但您可以建立 GetItemSpec，它會提供低階 GetItem 操作輸入的完整存取。以下程式碼範例會建立 GetItemSpec，然後使用該規格做為 getItem 方法的輸入。

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());
```

若要將 Item 以可供人閱讀的格式印出，請使用 toJSONPretty 方法。上個範例的輸出類似這樣。

```
{
  "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 與 JSON 文件

在 PutItem 與 JSON 文件 一節中，您將 JSON 文件存放在名為 VendorInfo 的 Map 屬性中。您可以使用 getItem 方法擷取 JSON 格式的整份文件。或者可以使用文件路徑標記法擷取文件中的部分元素。下列 Java 程式碼範例會示範這些技術。

```
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());
```

上個範例的輸出類似這樣。

```
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

您可以使用 toJSON 方法，將任一項目 (或其屬性) 轉換為格式化為 JSON 格式的字串。下列程式碼會擷取數個最上層及巢狀屬性，然後將結果以 JSON 印出。

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 210)
    .withProjectionExpression("VendorInfo.V01, Title, Price");

Item item = table.getItem(spec);
System.out.println(item.toJSON());
```

輸出看起來如下。

```
{"VendorInfo":{"V01":{"Name":"Acme Books","Offices":
["Seattle"]}},"Price":30,"Title":"Book 210 Title"}
```

批次寫入：放入和刪除多個項目

批次寫入表示在一個批次中放入和刪除多個項目。您可利用 batchWriteItem 方法，在單一呼叫中對來自一或多個資料表的多個項目執行放入與刪除操作。以下是使用 適用於 Java 的 AWS SDK 文件 API 放置或刪除多個項目的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 TableWriteItems 類別的執行個體，其描述針對資料表所有的寫入與刪除操作。若希望用單一批次寫入操作寫入多份資料表，您必須每份資料表都建立一個 TableWriteItems 執行個體。
3. 提供您在前述步驟中建立的 batchWriteItem 物件，藉以呼叫 TableWriteItems 方法。

4. 處理回應。您應該檢查回應中是否傳回任何未經處理的請求項目。如果您達到佈建輸送量配額或遇到一些其他暫時性錯誤，就可能發生這個狀況。此外，DynamoDB 會限制您在請求中指定的請求大小及操作次數。如果您超出這些限制，DynamoDB 會拒絕此請求。如需詳細資訊，請參閱Amazon DynamoDB 中的配額。

下列 Java 程式碼範例示範上述步驟。範例會在兩份資料表上執行 batchWriteItem 操作：Forum 和 Thread。對應的 TableWriteItems 物件會定義下列動作：

- 在 Forum 表中放入一個項目。
- 在 Thread 表中寫入及刪除一個項目。

該程式碼接著會呼叫 batchWriteItem 來執行操作。

```
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
```

如需運作範例，請參閱 範例：使用 適用於 Java 的 AWS SDK 文件 API 的批次寫入操作。

批次取得：取得多個項目

您可利用 batchGetItem 方法，從一或多個資料表擷取多個項目。若要擷取單一項目，可以使用 getItem 方法。

請遵循下列步驟：

1. 建立 DynamoDB 類別的執行個體。
2. 建立 TableKeysAndAttributes 類別的執行個體，其描述要從資料表擷取的主索引鍵值清單。若希望用單一批次擷取操作讀取多份資料表，您必須每份資料表都建立一個 TableKeysAndAttributes 執行個體。
3. 提供您在前述步驟中建立的 batchGetItem 物件，藉以呼叫 TableKeysAndAttributes 方法。

下列 Java 程式碼範例示範上述步驟。本範例會從 Forum 表擷取兩個項目，從 Thread 表擷取三個項目。

```
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);
    }
}
```

指定選用參數

除了必要的參數之外，您也可在使用 batchGetItem 時指定選用的參數。例如，您可為每個定義的 ProjectionExpression 提供 TableKeysAndAttributes。您如此即可指定要從資料表擷取的屬性。

下列程式碼範例會從 Forum 表擷取兩個項目。withProjectionExpression 參數會指定只擷取 Threads 屬性。

Example

```
TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes("Forum")
    .withProjectionExpression("Threads");

forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(forumTableKeysAndAttributes);
```

更新項目

updateItem 物件的 Table 方法可更新現有的屬性值、新增新的屬性，或是從現有項目中刪除屬性。

updateItem 方法的行為如下：

- 若某項目不存在 (資料表中沒有具備指定主索引鍵的項目)，則 updateItem 會在資料表中新增一個新項目。
- 若項目存在，updateItem 會依 UpdateExpression 參數的指定，執行更新。

> **Note**
>
> 您也可以使用 putItem 來「更新」項目。例如，若呼叫 putItem 將項目新增至資料表，但具備指定主索引鍵的項目已存在，則 putItem 會取代整個項目。如果輸入內並未指定現有項目中的屬性，putItem 就會從項目中移除這些屬性。
> 一般而言，建議您在想要修改任何項目屬性時，使用 updateItem。updateItem 方法只會修改您在輸入內指定的項目屬性，而項目內的其他屬性則保持不變。

請遵循下列步驟：

1. 建立 Table 類別的執行個體，代表您要使用的資料表。
2. 呼叫 updateTable 執行個體的 Table 方法。您必須指定要擷取之項目的主索引鍵，以及描述要修改之屬性及修改方式的 UpdateExpression。

下列 Java 程式碼範例示範上述工作。程式碼會更新 ProductCatalog 表中的書籍項目。其會將新的作者新增到 Authors 集合中，並刪除現有的 ISBN 屬性。它也會將價格減一。

ExpressionAttributeValues 映射內容會用於 UpdateExpression 中。預留位置 :val1 與 :val2，會於執行時期由 Authors 和 Price 的實際值取代。

```
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);
```

指定選用參數

除了必要的參數之外，您也可以為 updateItem 方法指定選用參數，包括必須滿足才會發生更新的條件。如果您指定的條件不符合， 會 適用於 Java 的 AWS SDK 擲回 ConditionalCheckFailedException。例如，下列 Java 程式碼範例會依據條件，將書籍項目的價格更新為 25。其會指定 ConditionExpression，指出只有在現有價格為 20 時才更新價格。

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);
```

原子計數器

您可以使用 updateItem 來實作原子計數器以增加或減少現有屬性的值，卻不干擾其他寫入請求。若要增加原子計數器，請於使用 UpdateExpression 時搭配 set 動作，將數值新增到類型為 Number 的現有屬性。

下列範例示範這項作業，將 Quantity 屬性加 1。其也同時示範在 ExpressionAttributeNames 中使用 UpdateExpression 參數。

```
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);
```

刪除項目

deleteItem 方法會從資料表刪除項目。您必須提供要刪除項目的主索引鍵。

請遵循下列步驟：

1. 建立 DynamoDB 用戶端執行個體。
2. 提供希望刪除之項目的索引鍵，可呼叫 deleteItem 方法。

下列 Java 範例示範這些任務。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

DeleteItemOutcome outcome = table.deleteItem("Id", 101);
```

指定選用參數

您可以為 deleteItem 指定選用參數。例如，下列 Java 程式碼範例指定 ConditionExpression，指出只有當書籍不再出版時 (InPublication 屬性為 false)，才能刪除 ProductCatalog 中的書籍項目。

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);
```

範例：使用 適用於 Java 的 AWS SDK 文件 API 的 CRUD 操作

下列程式碼範例示範對 Amazon DynamoDB 項目執行 CRUD 操作。此範例會建立項目、擷取該項目、執行數次更新，最後刪除該項目。

> **Note**
>
> 適用於 Java 的開發套件也提供物件持久性模型，讓您將用戶端類別映射至 DynamoDB 資料表。此方法可以減少您必須撰寫的程式碼數量。如需詳細資訊，請參閱 Java 1.x：DynamoDBMapper。

> **Note**
>
> 此程式碼範例假設您已根據 在 DynamoDB 中建立資料表，以及載入程式碼範例的資料 一節的說明將資料載入 DynamoDB 的帳戶。
> 如需執行下列範例的逐步說明，請參閱「Java 程式碼範例」。

```
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());
```

```
        }
    }
}
```

範例：使用 適用於 Java 的 AWS SDK 文件 API 的批次操作

本節提供使用 適用於 Java 的 AWS SDK 文件 API 在 Amazon DynamoDB 中批次寫入和批次取得操作的範例。

> **Note**
>
> 適用於 Java 的開發套件也提供物件持久性模型，讓您將用戶端類別映射至 DynamoDB 資料表。此方法可以減少您必須撰寫的程式碼數量。如需詳細資訊，請參閱Java 1.x：DynamoDBMapper。

主題

- 範例：使用 適用於 Java 的 AWS SDK 文件 API 的批次寫入操作
- 範例：使用 適用於 Java 的 AWS SDK 文件 API 的批次取得操作

範例：使用 適用於 Java 的 AWS SDK 文件 API 的批次寫入操作

下列 Java 程式碼範例使用 `batchWriteItem` 方法，執行下列寫入及刪除操作：

- 在 `Forum` 表中放入一個項目。
- 在 `Thread` 表中放入一個項目並刪除一個項目。

建立您的批次寫入請求時，您可以對一或多個資料表指定任何次數的放入和刪除請求。但 `batchWriteItem` 對於批次寫入要求的大小，以及單一批次寫入操作中的寫入及刪除操作次數有所限制。如果您的請求超出這些限制，您的請求會被拒絕。如果您的資料表因佈建輸送量不足而無法處理此請求，回應就會傳回未經處理的請求項目。

以下範例會檢查回應，查看它是否有任何未經處理的請求項目。若的確有所限制，其會重頭迴圈並會重新傳送 `batchWriteItem` 請求，同時附上請求中未經處理的項目。如已按照本指南中的範例操作，應已建立 `Forum` 和 `Thread` 資料表。您也可利用程式設計方式，建立這些資料表並上傳範例資料。如需詳細資訊，請參閱 使用 建立範例資料表和上傳資料 適用於 Java 的 AWS SDK。

如需測試下列範例的逐步說明，請參閱 Java 程式碼範例。

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);
        }

    }

}
```

範例：使用 適用於 Java 的 AWS SDK 文件 API 的批次取得操作

下列 Java 程式碼範例使用 `batchGetItem` 方法，從 `Forum` 和 `Thread` 表擷取多個項目。`BatchGetItemRequest` 指定每個要擷取之項目的資料表名稱與索引鍵清單。此範例處理回應的方式是列印已擷取的項目。

> **Note**
>
> 此程式碼範例假設您已根據 在 DynamoDB 中建立資料表，以及載入程式碼範例的資料 一節的說明將資料載入 DynamoDB 的帳戶。
> 如需執行下列範例的逐步說明，請參閱「Java 程式碼範例」。

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());
        }

    }

}
```

範例：使用 適用於 Java 的 AWS SDK 文件 API 處理二進位類型屬性

下列 Java 程式碼範例示範二進位類型屬性的處理。範例會將項目新增至 Reply 表。該項目包含存放壓縮資料的二進位類型屬性 (ExtendedMessage)。範例接著會擷取項目，並列印所有屬性值。為了進行示範，該範例使用 GZIPOutputStream 類別來壓縮範例串流，並將其指派給 ExtendedMessage 屬性。擷取二進制屬性時，其便會使用 GZIPInputStream 類別進行解壓縮。

> **Note**
>
> 適用於 Java 的開發套件也提供物件持久性模型，讓您將用戶端類別映射至 DynamoDB 資料表。此方法可以減少您必須撰寫的程式碼數量。如需詳細資訊，請參閱 Java 1.x：DynamoDBMapper。

如已完成在 DynamoDB 中建立資料表，以及載入程式碼範例的資料一節，應已建立 Reply 表。您也可以透過編寫程式的方式建立此資料表。如需詳細資訊，請參閱 使用 建立範例資料表和上傳資料 適用於 Java 的 AWS SDK。

如需測試下列範例的逐步說明，請參閱 Java 程式碼範例。

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;
    }
}
```

處理項目：.NET

您可以使用 適用於 .NET 的 AWS SDK 低階 API，對資料表中的項目執行典型的建立、讀取、更新和刪除 (CRUD) 操作。以下是使用 .NET 低階 API 執行資料 CRUD 操作所遵循的一般步驟：

1. 建立 `AmazonDynamoDBClient` 類別的執行個體 (用戶端)。
2. 在對應的請求物件中，提供操作專屬的必要參數。

   例如，上傳項目時使用 `PutItemRequest` 請求物件，擷取現有的項目時使用 `GetItemRequest` 請求物件。

   您可以使用請求物件來提供必要和選用的參數。
3. 透過傳遞您在前一步驟中所建立的請求物件，執行用戶端提供的適當方法。

   `AmazonDynamoDBClient` 用戶端提供 CRUD 操作的 `PutItem`、`GetItem`、`UpdateItem` 和 `DeleteItem` 方法。

主題

- 放入項目
- 取得項目
- 更新項目
- 原子計數器
- 刪除項目

- 批次寫入：放入和刪除多個項目
- 批次取得：取得多個項目
- 範例：使用 適用於 .NET 的 AWS SDK 低階 API 的 CRUD 操作
- 範例：使用 適用於 .NET 的 AWS SDK 低階 API 的批次操作
- 範例：使用 適用於 .NET 的 AWS SDK 低階 API 處理二進位類型屬性

放入項目

PutItem 方法會將項目上傳至資料表。若已有該項目，其會取代整個項目。

> **Note**
>
> 若不希望取代整個項目，而是只想要更新特定的屬性，可以使用 UpdateItem 方法。如需詳細資訊，請參閱 更新項目。

以下是使用低階 .NET 軟體開發套件 API 上傳項目的步驟：

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 您可以透過建立 PutItemRequest 類別的執行個體來提供必要的參數。

   若要放入項目，您必須提供資料表名稱及項目。
3. 提供您在前一個步驟中建立的 PutItem 物件來執行 PutItemRequest 方法。

下列 C# 範例示範前述步驟。範例會將項目上傳至 ProductCatalog 資料表。

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);
```

在上述範例中，您要上傳具有 Id、Title、ISBN 和 Authors 屬性的書籍項目。請注意，Id 是數值類型的屬性，而所有其他屬性則是字串類型。作者是 String 集合。

指定選用參數

您也可以使用 PutItemRequest 物件提供選用參數，如以下 C# 範例所示。此範例會指定下列選用參數：

- ExpressionAttributeNames、ExpressionAttributeValues 和 ConditionExpression 指定，只有當現有項目的 ISBN 屬性為特定值時，才能取代該項目。
- ReturnValues 參數請求回應中為舊項目。

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);
```

如需詳細資訊，請參閱 PutItem。

取得項目

GetItem 方法會擷取項目。

> **Note**
>
> 若要擷取多個項目，您可以使用 BatchGetItem 方法。如需詳細資訊，請參閱 批次取得：取得多個項目。

以下是使用低階 適用於 .NET 的 AWS SDK API 來擷取現有項目的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 您可以透過建立 GetItemRequest 類別的執行個體來提供必要的參數。

   若要取得項目，您必須提供資料表名稱及項目的主索引鍵。
3. 提供您在前一個步驟中建立的 GetItem 物件來執行 GetItemRequest 方法。

下列 C# 範例示範前述步驟。下列範例會從 ProductCatalog 表擷取項目。

```
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.
```

指定選用參數

您也可以使用 GetItemRequest 物件提供選用參數，如以下 C# 範例所示。範例會指定下列選用參數：

- ProjectionExpression 參數會指定要擷取的屬性。
- ConsistentRead 參數，用來執行高度一致性讀取。若要進一步了解讀取一致性，請參閱 DynamoDB 讀取一致性。

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;
```

如需詳細資訊，請參閱 GetItem。

更新項目

UpdateItem 方法會更新現有的項目 (若存在的話)。您可以使用 UpdateItem 操作來更新現有的屬性值、新增新的屬性，或從現有的集合刪除屬性。若找不到具有指定主索引鍵的項目，則其會新增新的項目。

UpdateItem 操作使用以下準則：

- 若還沒有該項目，UpdateItem 會使用輸入中指定的主索引鍵新增項目。
- 若已有該項目，UpdateItem 會套用更新，如下所示：
  - 使用更新中的值取代現有的屬性值。
  - 若您在輸入中提供的屬性不存在，則其會為項目新增新的屬性。
  - 若輸入屬性為 Null，則其會刪除該屬性 (若有的話)。
  - 如果您針對 Action 使用 ADD，就可以將值新增至現有的集合 (字串集或數字集)，或以數學方法加 (使用正數) 減 (使用負數) 現有的數值屬性值。

> **Note**
>
> PutItem 操作也可以執行更新。如需詳細資訊，請參閱 放入項目。例如，如果您呼叫 PutItem 上傳項目，且主索引鍵已存在，則 PutItem 操作會取代整個項目。如果現有的項目中有屬性，但輸入中並未指定這些屬性，則 PutItem 操作會刪除這些屬性。但是，UpdateItem 只會更新指定的輸入屬性。任何其他該項目現有的屬性都不會變更。

以下是使用低階 .NET 軟體開發套件 API 來更新現有項目的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 您可以透過建立 UpdateItemRequest 類別的執行個體來提供必要的參數。

   這是您說明所有更新的請求物件，例如新增屬性、更新現有的屬性或刪除屬性。若要刪除現有的屬性，請以 Null 值指定屬性名稱。
3. 提供您在前一個步驟中建立的 UpdateItem 物件來執行 UpdateItemRequest 方法。

下列 C# 程式碼範例示範前述步驟。範例會更新 ProductCatalog 表中的項目。它會將新的作者新增到 Authors 集合，並刪除現有的 ISBN 屬性。它也會將價格減一。

```
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);
```

指定選用參數

您也可以使用 UpdateItemRequest 物件提供選用參數，如以下 C# 範例所示。它會指定以下選用參數：

- ExpressionAttributeValues 和 ConditionExpression 指定只有當現有價格為 20.00 時才更新價格。
- ReturnValues 參數，其會請求回應中的更新項目。

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);
```

如需詳細資訊，請參閱 UpdateItem。

原子計數器

您可以使用 updateItem 來實作原子計數器以增加或減少現有屬性的值，卻不干擾其他寫入請求。若要更新原子計數器，UpdateExpression 參數請使用屬性類型為 Number 的 updateItem，Action 使用 ADD。

下列範例示範這項作業，將 Quantity 屬性加 1。

```
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);
```

刪除項目

DeleteItem 方法會從資料表刪除項目。

以下是使用低階 .NET 軟體開發套件 API 來刪除項目的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 您可以透過建立 DeleteItemRequest 類別的執行個體來提供必要的參數。

   若要刪除項目，需要有資料表名稱及項目的主索引鍵。
3. 提供您在前一個步驟中建立的 DeleteItem 物件來執行 DeleteItemRequest 方法。

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);
```

指定選用參數

您也可以使用 DeleteItemRequest 物件提供選用參數，如以下 C# 程式碼範例所示。它會指定以下選用參數：

- ExpressionAttributeValues 和 ConditionExpression 指定只有不再發行的書籍項目才可刪除 (InPublication 屬性值為 false)。
- ReturnValues 參數，其會請求回應中的刪除項目。

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);
```

如需詳細資訊，請參閱 DeleteItem。

批次寫入：放入和刪除多個項目

批次寫入表示在一個批次中放入和刪除多個項目。您可利用 BatchWriteItem 方法，在單一呼叫中對來自一或多個資料表的多個項目執行放入與刪除操作。以下是使用低階 .NET 開發套件 API 來擷取多個項目的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 透過建立 BatchWriteItemRequest 類別的執行個體來說明所有的放入與刪除操作。
3. 提供您在前一個步驟中建立的 BatchWriteItem 物件來執行 BatchWriteItemRequest 方法。

4. 處理回應。您應該檢查回應中是否傳回任何未經處理的請求項目。如果您達到佈建輸送量配額或遇到一些其他暫時性錯誤，就可能發生這個狀況。此外，DynamoDB 會限制您在請求中指定的請求大小及操作次數。如果您超出這些限制，DynamoDB 會拒絕此請求。如需詳細資訊，請參閱 BatchWriteItem。

下列 C# 程式碼範例示範前述步驟。範例會建立 BatchWriteItemRequest 來執行下列寫入操作：

- 在 Forum 表中放入一個項目。
- 在 Thread 表中放入及刪除一個項目。

此程式碼會執行 BatchWriteItem 來執行批次操作。

```
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);
```

如需運作範例，請參閱 範例：使用 適用於 .NET 的 AWS SDK 低階 API 的批次操作。

批次取得：取得多個項目

您可利用 `BatchGetItem` 方法，從一或多個資料表擷取多個項目。

> **Note**
>
> 若要擷取單一項目，可以使用 `GetItem` 方法。

以下是使用低階 適用於 .NET 的 AWS SDK API 來擷取多個項目的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 您可以透過建立 BatchGetItemRequest 類別的執行個體來提供必要的參數。

   若要擷取多個項目，需要有資料表名稱及主索引鍵值的清單。
3. 提供您在前一個步驟中建立的 BatchGetItem 物件來執行 BatchGetItemRequest 方法。
4. 處理回應。您應該檢查是否有任何未經處理的索引鍵，如果您達到佈建輸送量配額或遇到一些其他暫時性錯誤，就可能發生這個狀況。

下列 C# 程式碼範例示範前述步驟。範例會從 Forum 和 Thread 這兩份資料表擷取項目。請求會指定兩個 Forum 表的項目及三個 Thread 表的項目。回應會包含這兩份資料表的項目。程式碼會顯示您處理回應的方法。

```
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);
}
```

指定選用參數

您也可以使用 BatchGetItemRequest 物件提供選用參數，如以下 C# 程式碼範例所示。範例會從 Forum 表擷取兩個項目。它指定以下選用參數：

- ProjectionExpression 參數會指定要擷取的屬性。

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);
```

如需詳細資訊，請參閱 BatchGetItem。

範例：使用 適用於 .NET 的 AWS SDK 低階 API 的 CRUD 操作

下列 C# 程式碼範例會示範對 Amazon DynamoDB 項目執行 CRUD 操作。此範例會在 `ProductCatalog` 表中新增項目、擷取它、執行各種更新，最後刪除該項目。如果您尚未建立資料表，也可使用程式設計方式建立資料表。如需詳細資訊，請參閱使用 建立範例資料表和上傳資料 適用於 .NET 的 AWS SDK。

如需測試下列範例的逐步說明，請參閱 .NET 程式碼範例。

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("************************************************");
        }
    }
}
```

範例：使用 適用於 .NET 的 AWS SDK 低階 API 的批次操作

主題

- 範例：使用 適用於 .NET 的 AWS SDK 低階 API 的批次寫入操作
- 範例：使用 適用於 .NET 的 AWS SDK 低階 API 的批次取得操作

本節提供 Amazon DynamoDB 支援的批次操作、批次寫入和批次取得的範例。

範例：使用 適用於 .NET 的 AWS SDK 低階 API 的批次寫入操作

下列 C# 程式碼範例使用 BatchWriteItem 方法來執行下列放入及刪除操作：

- 在 Forum 表中放入一個項目。
- 在 Thread 表中放入一個項目並刪除一個項目。

建立您的批次寫入請求時，您可以對一或多個資料表指定任何次數的放入和刪除請求。但 DynamoDB BatchWriteItem 對於批次寫入要求的大小，以及單一批次寫入操作中的寫入及刪除操作次數有所限制。如需詳細資訊，請參閱 BatchWriteItem。如果您的請求超出這些限制，您的請求會被拒絕。如果您的資料表因佈建輸送量不足而無法處理此請求，回應就會傳回未經處理的請求項目。

以下範例會檢查回應，查看它是否有任何未經處理的請求項目。若的確有所限制，其會重頭迴圈並會重新傳送 BatchWriteItem 請求，同時附上請求中未經處理的項目。您也可利用程式設計方式來建立這些範例資料表並上傳範例資料。如需詳細資訊，請參閱 使用 建立範例資料表和上傳資料 適用於 .NET 的 AWS SDK。

如需測試下列範例的逐步說明，請參閱 .NET 程式碼範例。

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);
        }
    }
}
```

範例：使用 適用於 .NET 的 AWS SDK 低階 API 的批次取得操作

下列 C# 程式碼範例使用 `BatchGetItem` 方法，從 Amazon DynamoDB 的 `Forum` 和 `Thread` 表中擷取多個項目。`BatchGetItemRequest` 指定每份資料表的資料表名稱及主索引鍵清單。此範例處理回應的方式是列印已擷取的項目。

如需測試下列範例的逐步說明，請參閱 .NET 程式碼範例。

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("************************************************");
        }
    }
}
```

範例：使用 適用於 .NET 的 AWS SDK 低階 API 處理二進位類型屬性

下列 C# 程式碼範例示範二進位類型屬性的處理方式。範例會將項目新增至 Reply 表。該項目包含存放壓縮資料的二進位類型屬性 (ExtendedMessage)。範例接著會擷取項目，並列印所有屬性值。為了進行說明，該範例使用 GZipStream 類別來壓縮範例串流，將其指派給 ExtendedMessage 屬性，並在列印屬性值時解壓縮它。

如需測試下列範例的逐步說明，請參閱 .NET 程式碼範例。

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();
            }
        }
    }
}
```

在 DynamoDB 中使用次要索引來改善資料存取

Amazon DynamoDB 透過指定主索引鍵值來提供資料表中項目的快速存取。不過，許多應用程式都會受益於下列情況：有一或多個次要 (或替代) 索引鍵可用，允許使用主索引鍵以外的屬性來有效存取資料。若要解決此問題，您可以在資料表上建立一或多個次要索引，並針對這些索引發出 `Query` 或 `Scan` 請求。

次要索引是一種資料結構，包含資料表中的屬性子集，以及可支援 `Query` 操作的替代索引鍵。您可以使用 `Query` 從索引擷取資料，方式與您搭配使用 `Query` 與資料表相同。一份資料表可以有多個次要索引，讓您的應用程式能夠存取許多不同的查詢模式。

> **Note**
>
> 您也可以對索引進行 `Scan`，方式與您對資料表進行 `Scan` 相同。
> 資源型政策目前不支援次要索引掃描操作的跨帳戶存取。

每個次要索引都只能與一個資料表建立關聯，次要索引可從資料表中取得其資料。這稱為索引的基礎資料表。當您建立索引時，請定義索引的替代索引鍵 (分割區索引鍵和排序索引鍵)。您還可以定義要從基礎資料表投影或複製到索引中的屬性。DynamoDB 會將這些屬性以及基礎資料表中的主索引鍵屬性複製到索引。您接著可以查詢或掃描索引，就像查詢或掃描資料表一樣。

DynamoDB 會自動維護每個次要索引。當您新增、修改或刪除基礎資料表中的項目時，也會更新該資料表上的任何索引，以反映這些變更。

DynamoDB 支援兩種次要索引：

- 全域次要索引：一種含分割區索引鍵或排序索引鍵的索引，這些索引鍵可與基礎資料表上的索引鍵不同。全域次要索引之所以視為全域，是因為索引上的查詢可以跨越所有分割區之間基礎資料表中的所有資料。全域次要索引會儲存在基礎資料表以外的專屬分割區空間，且會和基礎資料表分開調整。
- 本機次要索引：是一種與基礎資料表擁有相同分割區索引鍵但不同排序索引鍵的索引。本機次要索引的「本機」概念是指，本機次要索引的每個分割區都會列入基礎資料表分割區的範圍，其中分割區索引鍵的值皆相同。

如需全域次要索引與本機次要索引的比較，請觀看此影片。

在 GSI 和 LSI 之間做出正確的選擇

主題

- 在 DynamoDB 中使用全域次要索引
- 本機次要索引

當您判斷要使用的索引類型時，應該考量應用程式需求。下列資料表會顯示全域次要索引和本機次要索引之間的主要差異。

特性	全域次要索引	本機次要索引
索引鍵結構描述	全域次要索引的主索引鍵可以是簡單 (分割區索引鍵) 或複合 (分割區索引鍵和排序索引鍵)。	本機次要索引的主索引鍵必須是複合形式 (分割區索引鍵和排序索引鍵)。
索引鍵屬性	索引的分割區索引鍵和排序索引鍵 (若存在) 可以是字串、數字或二進位類型的任何基礎資料表屬性。	索引的分割區索引鍵與基礎資料表的分割區索引鍵具有相同的屬性。排序索引鍵可以是字串、數字或二進位類型的任何基礎資料表屬性。
每個分割區索引鍵值的大小限制	全域次要索引沒有大小限制。	針對每個分割區索引鍵值，所有已索引項目的大小總計必須是 10 GB 或以下。
線上索引操作	您可在建立資料表的同時建立全域次要索引。您也可以將新的全域次要索引新增至現有的	本機次要索引是在建立資料表的同時建立的。您無法對現有資料表新增本機次要索引，也

特性	全域次要索引	本機次要索引
	資料表，或刪除現有的全域次要索引。如需詳細資訊，請參閱 在 DynamoDB 中管理全域次要索引。	無法刪除現有的任何本機次要索引。
查詢和分割區	全域次要索引可讓您跨所有分割區查詢整個資料表。	本機次要索引可讓您查詢單一分割區 (由查詢中的分割區索引鍵值所指定)。
讀取一致性	全域次要索引上的查詢僅支援最終一致性。	當您查詢本機次要索引時，可以選擇最終一致性或強烈一致性。
佈建輸送量耗用	針對讀取和寫入活動，每個全域次要索引都有自己的佈建輸送量設定。對全域次要索引的查詢或掃描會使用來自索引的容量單位，而不是來自基礎資料表。這適用於資料表寫入引發的全域次要索引更新。與全域資料表相關聯的全域次要索引會耗用寫入容量單位。	對本機次要索引的查詢或掃描會使用來自基礎資料表的讀取容量單位。當您寫入至資料表時，也會一併更新其本機次要索引，且這些更新會使用來自基礎資料表的寫入容量單位。與全域資料表相關聯的本機次要索引會耗用複寫的寫入容量單位。
投影屬性	您可以使用全域次要索引查詢或掃描來僅請求投影至索引的屬性。DynamoDB 不會從資料表中擷取任何屬性。	若要查詢或掃描本機次要索引，您可以請求未投影至索引的屬性。DynamoDB 會自動從資料表中擷取這些屬性。

如果您想要建立具有次要索引的多個資料表，則必須循序進行這項操作。例如，您建立第一個資料表並等待它變成 ACTIVE、建立下一個資料表並等待其變成 ACTIVE，以此類推。如果您嘗試並行建立具有次要索引的多份資料表，則 DynamoDB 會傳回 `LimitExceededException`。

每個次要索引使用的資料表類別和容量模式皆與其相關聯的基礎資料表相同。針對每個次要索引，您必須指定下列項目：

- 要建立的索引類型：全域次要索引或本機次要索引。

- 索引的名稱。索引的命名規則與資料表的命名規則相同，如「Amazon DynamoDB 中的配額」中所列。名稱對與其建立關聯的基礎資料表而言必須是唯一的，但您可以將相同的名稱用於與其不同基礎資料表建立關聯的索引。
- 索引的索引鍵結構描述。索引鍵結構描述中的每個屬性都必須是 String、Number 或 Binary 類型的最上層屬性。不允許其他資料類型 (包含文件和集合)。索引鍵結構描述的其他需求取決於索引類型：
  - 針對全域次要索引，分割區索引鍵可以是基礎資料表的任何純量屬性。排序索引鍵是選用項目，它也可以是基礎資料表的任何純量屬性。
  - 針對本機次要索引，分割區索引鍵必須與基礎資料表的分割區索引鍵相同，而排序索引鍵必須是非索引鍵基礎資料表屬性。
- 從基礎資料表投影至索引的其他屬性 (如果有的話)。這些是自動投影至每個索引之資料表索引鍵屬性以外的屬性。您可以投影任何資料類型的屬性 (包含純量、文件和集合)。
- 索引的佈建輸送量設定 (必要時)：
  - 針對全域次要索引，您必須指定讀取和寫入容量單位設定。這些佈建輸送量設定與基礎資料表設定無關。
  - 針對本機次要索引，您不需要指定讀取和寫入容量單位設定。本機次要索引的任何讀取和寫入操作都是取自其基礎資料表的佈建輸送量設定。

為獲得最大的查詢靈活性，您最多可以為每份資料表建立 20 個全域次要索引 (預設配額) 和 5 個本機次要索引。

若要取得資料表次要索引的詳細清單，請使用 DescribeTable 操作。DescribeTable 會傳回資料表中每個次要索引的名稱、儲存大小和項目計數。這些值不會即時更新，但大約每六小時會重新整理一次。

您可以使用 Query 或 Scan 操作來存取次要索引中的資料。您必須指定基礎資料表名稱以及您要使用的索引名稱、要在結果中傳回的屬性，以及您要套用的任何條件表達式或篩選條件。DynamoDB 可以依遞增或遞減順序傳回結果。

當您刪除資料表時，也會刪除與該資料表建立關聯的所有索引。

如需最佳實務做法，請參閱「使用 DynamoDB 中次要索引的最佳實務」。

在 DynamoDB 中使用全域次要索引

某些應用程式可能需要執行多種查詢，並使用各種不同的屬性做為查詢條件。若要支援這些需求，您可以建立一或多個全域次要索引，並在 Amazon DynamoDB 中對這些索引發出 Query 請求。

主題

- 藍本：使用全域次要索引
- 屬性投影
- 多屬性金鑰結構描述
- 從全域次要索引讀取資料
- 資料表與全域次要索引之間的資料同步
- 具有全域次要索引的資料表類別
- 全域次要索引的佈建輸送量考量
- 全域次要索引的儲存考量
- 設計模式
- 在 DynamoDB 中管理全域次要索引
- 偵測和修正 DynamoDB 中的索引鍵違規情況
- 使用全域次要索引：Java
- 使用全域次要索引：.NET
- 在 DynamoDB 中搭配使用 AWS CLI 與全域次要索引

藍本：使用全域次要索引

為了示範，假設有一份名為 GameScores 的資料表，會追蹤手機遊戲應用程式的使用者與分數。GameScores 中的每個項目都是按分割區索引鍵 (UserId) 和排序索引鍵 (GameTitle) 識別。下圖顯示這些項目在資料表中的組織方式。(並未顯示所有屬性。)

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	...
...	...	...	...	...	...	

現在假設您想要撰寫排行榜應用程式來顯示每個遊戲的最高分。已指定索引鍵屬性 (UserId 和 GameTitle) 的查詢會很有效率。但若應用程式僅能根據 GameTitle 從 GameScores 擷取資料，就需要使用 Scan 操作。當愈來愈多項目新增至資料表時，掃描所有資料會變得很慢且效率不彰。這會讓回答問題變得很困難，例如下列問題：

- Meteor Blasters 遊戲曾記錄到的最高分為何？
- 哪位使用者在 Galaxy Invaders 中得到最高分？
- 最高的勝負比為何？

若要加速非索引鍵屬性的查詢，您可以建立全域次要索引。全域次要索引包含基礎資料表中的一系列屬性，但這些屬性會依與該資料表不同的主索引鍵組織。索引鍵不需要有任何來自資料表的索引鍵屬性。甚至不需要有和資料表相同的索引鍵結構描述。

例如，您可以建立名為 GameTitleIndex 的全域次要索引，其中分割區索引鍵為 GameTitle，排序索引鍵為 TopScore。因為基礎資料表的主索引鍵屬性一律會投影到索引，所以也會顯示 UserId 屬性。下圖說明 GameTitleIndex 索引可能的樣子。

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"*
...	...	...

您現在可以查詢 GameTitleIndex 並輕鬆取得 Meteor Blasters 的分數。結果會依排序索引鍵值 TopScore 排序。如果您將 ScanIndexForward 參數設定為 false，結果會依遞減順序傳回，因此最高分優先傳回。

每個全域次要索引都必須有分割區索引鍵，並可以有一個選用的排序索引鍵。索引鍵結構描述可以和基礎資料表結構描述不同。您可以擁有一個具有簡易主索引鍵 (分割區索引鍵) 的資料表，並使用複合主鍵 (分割區索引鍵和排序索引鍵) 建立全域次要索引，反之亦然。索引鍵屬性可包含來自基礎資料表的任何最上層 String、Number 或 Binary 屬性。不允許其他純量類型、文件類型和集合類型。

您可以視需要將其他基礎資料表屬性投影到索引。當您查詢索引時，DynamoDB 可有效率地擷取這些投影屬性。但是，全域次要索引查詢無法自基礎資料表擷取屬性。例如，如果您依上圖所示查詢 GameTitleIndex，則除 TopScore 外，此查詢不能存取任何非索引鍵屬性 (雖然會自動投影索引鍵屬性 GameTitle 和 UserId)。

在 DynamoDB 資料表中，每個索引鍵值必須是唯一的。不過，全域次要索引中的索引鍵值不需要是唯一的。為了示範，假設有一款名為 Comet Quest 的遊戲特別難，許多新的使用者試過後均無法突破零分的成績。以下為可代表此遊戲的一些資料。

UserId	GameTitle	TopScore
123	Comet Quest	0
201	Comet Quest	0
301	Comet Quest	0

當此資料新增至 GameScores 資料表時，DynamoDB 會將其傳播至 GameTitleIndex。如果我們接著使用 GameTitle Comet Quest 與 TopScore 0 來查詢索引，即會傳回下列資料。

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

回應中只會顯示具有指定索引鍵值的項目。在此資料集合中，項目未依特定順序排列。

全域次要索引只會追蹤索引鍵屬性確實存在的資料項目。例如，假設您在 GameScores 表中新增另一個新項目，但只提供必要的主索引鍵屬性。

UserId	GameTitle
400	Comet Quest

因為您未指定 TopScore 屬性，所以 DynamoDB 不會將此項目傳播至 GameTitleIndex。因此，如已針對 GameScores 查詢所有 Comet Quest 項目，可能會取得下列四個項目。

UserId	GameTitle	TopScore
"123"	"Comet Quest"	0
"201"	"Comet Quest"	0
"301"	"Comet Quest"	0
"400"	"Comet Quest"	

針對 GameTitleIndex 的類似查詢仍只會傳回三個項目，而不是四個。這是因為具有不存在 TopScore 的項目不會傳播至索引。

GameTitle	TopScore	UserId
"Comet Quest"	0	"123"
"Comet Quest"	0	"201"
"Comet Quest"	0	"301"

屬性投影

投影是指從資料表複製到次要索引的屬性集合。資料表的分割區索引鍵和排序索引鍵一律會投影到索引中；您可以投影其他屬性來支援應用程式的查詢需求。查詢索引時，Amazon DynamoDB 可以存取投影中的任何屬性，就好像這些屬性在它們自己的資料表中一樣。

在建立次要索引時，您需要指定要投影到索引中的屬性。DynamoDB 為此提供三種不同的選項：

- KEYS_ONLY：索引中的每個項目都只包含資料表分割索引鍵和排序索引鍵值，以及索引鍵值。KEYS_ONLY 選項會產生最小的可行次要索引。
- INCLUDE：除了 KEYS_ONLY 中描述的屬性外，次要索引還包含您指定的其他非索引鍵屬性。
- ALL：次要索引包含來源資料表中的所有屬性。因為索引中的所有資料表資料都會重複，所以 ALL 投影會產生最大的可行次要索引。

在上圖中，GameTitleIndex 只有一個投影的屬性：UserId。因此，雖然應用程式可以在查詢中使用 GameTitle 和 TopScore 有效率地判斷每個遊戲最高分得主的 UserId，卻無法有效率地判斷最高分得主的最高勝負比。若要這樣做，應用程式必須在基礎資料表上執行額外的查詢，以擷取每個最高評分者的優缺點。支援查詢此資料更有效率的方法，便是將這些屬性從基礎資料表投影到全域次要索引，如下圖所示。

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
...	...	...	...	...

由於非索引鍵屬性 Wins 與 Losses 會投影到索引，因此應用程式可以判斷任何遊戲或任何遊戲與使用者 ID 組合的勝負比。

在選擇要投影到全域次要索引的屬性時，您必須權衡佈建輸送量成本和儲存成本：

- 若您只需要存取少量的屬性，並且希望盡可能地降低延遲，建議您考慮只將那些屬性投影到全域次要索引。索引愈小，存放成本就愈低，您的寫入成本也愈低。
- 如果應用程式會頻繁存取某些非索引鍵屬性，您應該考慮將這些屬性投影到全域次要索引。全域次要索引的額外儲存成本會抵銷頻繁執行資料表掃描的成本。
- 如果需要頻繁存取大多數的非索引鍵屬性，您可以將這些屬性 (或整個基礎資料表) 投影到全域次要索引。這能讓您掌握最大的靈活性。但儲存成本可能會增加，甚至翻倍。
- 如果您的應用程式不需頻繁查詢資料表，但必須對資料表中的資料執行許多寫入或更新，請考慮投影 KEYS_ONLY。全域次要索引的大小將會最小，但仍能在需要的時候進行查詢活動。

多屬性金鑰結構描述

全域次要索引支援多屬性索引鍵，可讓您從多個屬性編寫分割區索引鍵和排序索引鍵。使用多屬性索引鍵，您可以從最多四個屬性建立分割區索引鍵，以及從最多四個屬性建立排序索引鍵，每個索引鍵結構描述總計最多八個屬性。

多屬性索引鍵可簡化資料模型，無需手動將屬性串連至合成索引鍵。您可以直接使用網域模型中的自然屬性TOURNAMENT#WINTER2024#REGION#NA-EAST，而不是像 一樣建立複合字串。DynamoDB 會自動處理複合索引鍵邏輯、針對資料分佈將多個分割區索引鍵屬性雜湊在一起，並跨多個排序索引鍵屬性維護階層排序順序。

例如，假設您想依賽事和區域組織配對的遊戲競賽系統。使用多屬性索引鍵，您可以將分割區索引鍵定義為兩個不同的屬性： tournamentId和 region。同樣地，您可以使用 round、 和 等多個屬性來定義排序索引鍵bracket，matchId以建立自然階層。此方法可讓您的資料類型和程式碼保持乾淨，無需處理或剖析字串。

當您使用多屬性索引鍵查詢全域次要索引時，您必須使用等式條件指定所有分割區索引鍵屬性。對於排序索引鍵屬性，您可以依索引鍵結構描述中定義的順序left-to-right查詢它們。這表示您可以單獨查詢第一個排序索引鍵屬性、同時查詢前兩個屬性，或同時查詢所有屬性，但您無法略過中間的屬性。>、BETWEEN、 <或 等不等式條件begins_with( )必須是查詢中的最後一個條件。

在現有資料表上建立全域次要索引時，多屬性索引鍵的效果特別好。您可以使用資料表中已存在的屬性，而無需在資料中回填合成索引鍵。這可讓您直接將新的查詢模式新增至應用程式，方法是建立索引，使用不同的屬性組合來重組您的資料。

多屬性索引鍵中的每個屬性可以有自己的資料類型：String(S)、Number(N) 或 Binary(B)。選擇資料類型時，請考慮Number屬性會按數字排序，而不需要零填補，而String屬性則會按詞典排序。例如，如果您對分數屬性使用 Number類型，則值 5、50、500 和 1000 會依自然數字順序排序。與String類型相同的值會排序為 "1000"、"5"、"50"、"500"，除非您以前導零填補它們。

設計多屬性索引鍵時，請排序從最一般到最具體的屬性。對於分割區索引鍵，將一律一起查詢並提供良好資料分佈的屬性合併在一起。對於排序索引鍵，請先在階層中放置經常查詢的屬性，以最大化查詢彈性。此排序可讓您以任何符合您存取模式的精細程度進行查詢。

如需實作範例多屬性索引鍵，請參閱 。

從全域次要索引讀取資料

您可以使用 Query 和 Scan 操作從全域次要索引擷取項目。GetItem 和 BatchGetItem 操作不能用於全域次要索引。

查詢全域次要索引

您可以使用 Query 操作存取全域次要索引中一或多個項目。詢必須指定基礎資料表名稱以及您要使用的索引名稱、要在查詢結果中傳回的屬性，以及您要套用的任何查詢條件。DynamoDB 可以依遞增或遞減順序傳回結果。

假設下列資料從請求排行榜應用程式遊戲資料的 Query 傳回。

```
{
    "TableName": "GameScores",
    "IndexName": "GameTitleIndex",
    "KeyConditionExpression": "GameTitle = :v_title",
    "ExpressionAttributeValues": {
        ":v_title": {"S": "Meteor Blasters"}
    },
    "ProjectionExpression": "UserId, TopScore",
    "ScanIndexForward": false
}
```

在此查詢中：

- DynamoDB 會存取 GameTitleIndex，並使用 GameTitle 分割區索引鍵尋找 Meteor Blasters 的索引項目。所有具有此索引鍵的索引項目都會相鄰存放，以利快速擷取。
- 在此遊戲中，DynamoDB 使用索引存取此遊戲的所有使用者 ID 與最高分。
- 結果會傳回並遞減排序，因為 ScanIndexForward 參數設定為 false。

掃描全域次要索引

您可以使用 Scan 操作從全域次要索引檢索所有資料。您必須在請求中提供基礎資料表名稱及索引名稱。使用 Scan，DynamoDB 會讀取索引中的所有資料，並將其傳回應用程式。您也可以請求只傳回一部分的資料，並捨棄其他剩餘的資料。若要執行此作業，請使用 FilterExpression 操作的 Scan 參數。如需詳細資訊，請參閱 掃描的篩選條件表達式。

資料表與全域次要索引之間的資料同步

DynamoDB 會自動將每個全域次要索引與其基礎資料表同步。當應用程式寫入或刪除資料表中的項目時，該資料表的任何全域次要索引都會使用最終一致模型非同步更新。應用程式永遠不會直接寫入索引。然而，了解 DynamoDB 維持這些索引之方式的含義很重要。

全域次要索引會從基礎資料表繼承讀取/寫入容量模式。如需詳細資訊，請參閱切換 DynamoDB 容量模式時的注意事項。

在建立全域次要索引時，您要指定一或多個索引鍵屬性及其資料類型。這表示每次您將項目寫入基礎資料表時，這些屬性的資料類型都必須符合索引鍵結構描述的資料類型。在 GameTitleIndex 案例中，索引的 GameTitle 分割區索引鍵是定義為 String 資料類型。索引的 TopScore 排序索引鍵類型為 Number。如果您嘗試在 GameScores 表中新增項目，並為 GameTitle 或 TopScore 指定不同的資料類型，DynamoDB 會因資料類型不符而傳回 ValidationException。

當您在資料表中放置或刪除項目時，該資料表的全域次要索引會以最終一致的方式更新。在正常的情況下，資料表的資料變更會在瞬間傳播至全域次要索引。不過，在某些不太可能發生的失敗情況下，可能發生較久的傳播延遲。因此，您的應用程式必須能夠預期及處理針對全域次要索引查詢，卻傳回非最新結果的情況。

如果將項目寫入資料表，您不需指定任何全域次要索引排序索引鍵的屬性。以 GameTitleIndex 為例，您不需指定 TopScore 屬性的值，也能將新的項目寫入 GameScores 表。在此案例中，DynamoDB 不會將任何資料寫入此特定項目的索引。

相較於索引較少的資料表，全域次要索引較多的資料表會有較高的寫入活動成本。如需詳細資訊，請參閱全域次要索引的佈建輸送量考量。

具有全域次要索引的資料表類別

全域次要索引永遠會使用相同的資料表類別做為其基礎資料表。每當新增資料表的全域次要索引時，新索引都會使用與其基礎資料表相同的資料表類別。更新資料表的資料表類別時，也會更新所有相關聯的全域次要索引。

全域次要索引的佈建輸送量考量

當您對佈建模式資料表建立全域次要索引時，必須為該索引的預期工作負載指定讀取與寫入容量單位。全域次要索引的佈建輸送量設定與其基礎資料表的佈建輸送量設定無關。對全域次要索引執行 Query 操作會使用索引 (而不是基礎資料表) 的讀取容量單位。當您在資料表中放置、更新或刪除項目時，也會更新該資料表中的全域次要索引。這些索引更新會使用來自索引的寫入容量單位，而不是基礎資料表的單位。

例如，如果對全域次要索引執行 Query 操作，並超出其佈建讀取容量，您的請求就會經過限流。如果您對資料表執行大量寫入活動，但該資料表全域次要索引的寫入容量不足，即會限流該資料表的寫入活動。

> **Important**
>
> 若要避免限流問題產生，全域次要索引的佈建寫入容量必須大於等於基礎資料表的寫入容量，因為新的更新會同時寫入基礎資料表及全域次要索引。

若要檢視全域次要索引的佈建輸送量設定，請使用 `DescribeTable` 操作。會傳回所有資料表全域次要索引的詳細資訊。

讀取容量單位

全域次要索引支援最終一致讀取，每個讀取均使用一半的讀取容量單位。這表示單一全域次要索引查詢每個讀取容量單位最多可擷取 2 × 4 KB = 8 KB。

在全域次要索引查詢方面，DynamoDB 會使用為資料表查詢計算佈建讀取活動相同的方式，計算佈建讀取活動。唯一的差別在於計算方式是以索引項目的大小為基礎，而非基礎資料表中項目的大小。讀取容量單位數為所有傳回項目中，所有投影屬性大小的總和。然後，結果四捨五入至下一個 4 KB 界限。如需 DynamoDB 如何計算佈建輸送用量的詳細資訊，請參閱 DynamoDB 佈建容量模式。

`Query` 操作傳回的結果大小上限是 1 MB。這包含所有傳回項目之所有屬性名稱與值的大小。

例如，假設有一個全域次要索引，其每個項目均包含 2,000 個位元組的資料。現在假設您 `Query` 此索引，而此查詢的 `KeyConditionExpression` 會傳回 8 個項目。相符項目的總大小為 2,000 位元組 × 8 個項目 = 16,000 位元組。然後，此結果四捨五入至最近的 4 KB 界限。因為全域次要索引查詢為最終一致，所以總成本為 0.5 × (16 KB/4 KB) 或 2 個讀取容量單位。

寫入容量單位

當新增、更新或刪除資料表項目，並影響到全域次要索引時，全域次要索引會使用該操作的佈建寫入容量單位。寫入的總佈建輸送量成本包含寫入基礎資料表使用的寫入容量單位加上更新全域次要索引使用的寫入容量單位。如果資料表的寫入作業不需更新全域次要索引，就不會使用該索引的寫入容量。

為成功寫入資料表，資料表及其所有全域次要索引的佈建輸送量設定皆必須具有足以容納寫入的寫入容量。否則資料表的寫入會受到限流。

> **Important**
>
> 建立全域次要索引 (GSI) 時，如果因寫入基礎資料表所產生的 GSI 活動超過 GSI 的佈建寫入容量，則會對基礎資料表的寫入操作限流。此限流會影響所有寫入操作，包括編製索引程序，甚

至可能中斷生產工作負載。如需詳細資訊，請參閱對 Amazon DynamoDB 中的限流問題進行疑難排解。

將項目寫入全域次要索引的成本取決於幾個因素：

- 若您將項目寫入定義索引屬性的資料表，或是您將現有的項目更新為先前未定義的索引屬性，則將該項目寫入索引需要進行一次寫入操作。
- 若資料表的更新會變更索引鍵屬性的值 (從 A 到 B)，則需要兩次寫入：一次是從索引刪除先前的項目，第二次則是將新的項目寫入索引。
- 若項目存在於索引中，但寫入資料表致使索引屬性遭到刪除，則需要進行一次寫入，從索引刪除舊項目的投影。
- 若項目在更新之前或之後並不存在於索引中，則該索引將不會有任何額外的寫入成本。
- 如果資料表的更新只變更索引鍵結構描述中投影屬性的值，但並不變更任何索引鍵屬性的值，則需執行一個寫入將投影屬性的值更新至索引中。

所有這些因素都假設索引中每個項目的大小都小於或等於 1 KB 項目大小 (用於計算寫入容量單位)。較大的索引項目需要額外的寫入容量單位。您可以考慮您的查詢所需要傳回的屬性，並只將那些屬性投影到索引中，來將寫入成本降至最小。

全域次要索引的儲存考量

當應用程式將項目寫入資料表時，DynamoDB 會自動將正確的部分屬性複製到應顯示這些屬性的任何全域次要索引中。 AWS 您的帳戶會支付基本資料表中項目的儲存費用，以及該資料表上任何全域次要索引中屬性的儲存費用。

索引項目使用的空間數為下列項目的總和：

- 基礎資料表主索引鍵 (分割區索引鍵和排序索引鍵) 的大小 (位元組)
- 索引鍵屬性的大小 (位元組)
- 投影屬性 (若有的話) 的大小 (位元組)
- 每個索引項目 100 位元組的額外負荷

若要估算全域次要索引的儲存需求，您可以估算索引中項目的平均大小，再乘以基礎資料表中具有全域次要索引鍵屬性的項目數。

如果資料表包含未定義特定屬性的項目，但該屬性定義為索引分割區索引鍵或排序索引鍵，則 DynamoDB 不會將該項目的任何資料寫入索引。

設計模式

設計模式為使用全域次要索引時的常見挑戰提供經過驗證的解決方案。這些模式會示範如何針對特定使用案例建構索引，協助您建置有效率、可擴展的應用程式。

每個模式都包含完整的實作指南，其中包含程式碼範例、最佳實務和實際使用案例，協助您將模式套用至自己的應用程式。

主題

- 多屬性索引鍵模式

多屬性索引鍵模式

概觀

多屬性索引鍵可讓您建立全域次要索引 (GSI) 分割區，並排序每個分割區最多包含四個屬性的索引鍵。這可減少用戶端程式碼，並可讓您更輕鬆地對資料進行初始建模，並在稍後新增新的存取模式。

考慮常見的案例：若要建立依多個階層屬性查詢項目的 GSI，傳統上您需要透過串連值來建立合成索引鍵。例如，在遊戲應用程式中，若要依競賽、區域和回合查詢競賽配對，您可以建立合成 GSI 分割區索引鍵，例如 TOURNAMENT#WINTER2024#REGION#NA-EAST，以及合成排序索引鍵，例如 ROUND#SEMIFINALS#BRACKET#UPPER。此方法有效，但如果您要將 GSI 新增至現有資料表，則在寫入資料、讀取時剖析和在所有現有項目中回填合成索引鍵時，需要字串串連。這使得程式碼更雜亂且具有挑戰性，以在個別關鍵元件上維持類型安全。

多屬性索引鍵可解決 GSIs此問題。您可以使用多個現有屬性來定義 GSI 分割區索引鍵，例如 tournamentId 和 region。DynamoDB 會自動處理複合索引鍵邏輯，將它們雜湊在一起以進行資料分佈。您可以使用網域模型中的自然屬性撰寫項目，GSI 會自動為其編製索引。不串連、不剖析、不回填。您的程式碼會保持乾淨，您的資料會保持輸入狀態，而您的查詢會保持簡單。當您擁有具有自然屬性分組的階層式資料（例如競賽 → 區域 → 回合或組織 → 部門 → 團隊）時，此方法特別有用。

應用程式範例

本指南會逐步解說如何為 電競平台建置競賽配對追蹤系統。平台需要有效率地查詢多個維度的配對：依賽事和區域進行括號管理、依玩家進行配對歷史記錄，以及依排程日期進行查詢。

資料模型

在此演練中，競賽配對追蹤系統支援三種主要存取模式，每個模式都需要不同的金鑰結構：

存取模式 1：依其唯一 ID 查詢特定相符項目

- 解決方案：使用 matchId做為分割區索引鍵的基礎資料表

存取模式 2：查詢特定賽事和區域的所有配對，選擇性地依四捨五入、括號或配對進行篩選

- 解決方案：具有多屬性分割區索引鍵 (tournamentId + region) 和多屬性排序索引鍵 (round + bracket + matchId) 的全域次要索引
- 範例查詢：「NA-EAST 區域中的所有 WINTER2024 相符項目」或「WINTER2024/NA-EAST 上括號中的所有 SEMIFINALS 相符項目」

存取模式 3：查詢玩家的配對歷史記錄，選擇性地依日期範圍或競賽四捨五入進行篩選

- 解決方案：具有單一分割區索引鍵 (player1Id) 和多屬性排序索引鍵 (matchDate + round) 的全域次要索引
- 範例查詢：「玩家 101 的所有配對」或「Player 101 在 2024 年 1 月的配對」

檢查項目結構時，傳統和多屬性方法之間的主要差異會變得明確：

傳統全域次要索引方法（串連索引鍵）：

```
// Manual concatenation required for GSI keys
const item = {
    matchId: 'match-001',                                          // Base table PK
    tournamentId: 'WINTER2024',
    region: 'NA-EAST',
    round: 'SEMIFINALS',
    bracket: 'UPPER',
    player1Id: '101',
    // Synthetic keys needed for GSI
    GSI_PK: `TOURNAMENT#${tournamentId}#REGION#${region}`,       // Must concatenate
    GSI_SK: `${round}#${bracket}#${matchId}`,                    // Must concatenate
    // ... other attributes
};
```

多屬性全域次要索引方法（原生索引鍵）：

```
// Use existing attributes directly - no concatenation needed
const item = {
    matchId: 'match-001',                                    // Base table PK
    tournamentId: 'WINTER2024',
    region: 'NA-EAST',
    round: 'SEMIFINALS',
    bracket: 'UPPER',
    player1Id: '101',
    matchDate: '2024-01-18',
    // No synthetic keys needed - GSI uses existing attributes directly
    // ... other attributes
};
```

使用多屬性索引鍵，您可以使用自然網域屬性寫入項目一次。DynamoDB 會自動跨多個 GSIs 編製索引，而不需要合成串連金鑰。

基礎資料表結構描述：

- 分割區索引鍵：`matchId`(1 屬性)

全域次要索引結構描述 （具有多屬性索引鍵的 TournamentRegionIndex)：

- 分割區索引鍵：`tournamentId`、 `region` (2 個屬性)
- 排序索引鍵：`round`、`bracket`、 `matchId` (3 個屬性)

全域次要索引結構描述 (PlayerMatchHistoryIndex 與多屬性索引鍵 ）：

- 分割區索引鍵：`player1Id`(1 屬性)
- 排序索引鍵：`matchDate`、 `round` (2 個屬性)

基礎資料表：TournamentMatches

matchId (PK)	tourname tId	region	round	括號	player1Ic	player2Ic	matchDa	優勝者	分數
match-0(	WINTER 4	NA-EAST	最終	CHAMPI HIP	101	103	2024-01-0	101	3-1

matchId (PK)	tourname tId	region	round	括號	player1Id	player2Id	matchDa	優勝者	分數
match-0(	WINTER 4	NA-EAST	SEMIFIN S	UPPER	101	105	2024-01-8	101	3-2
match-0(	WINTER 4	NA-EAST	SEMIFIN S	UPPER	103	107	2024-01-8	103	3-0
match-0(	WINTER 4	NA-EAST	QUARTE NALS	UPPER	101	109	2024-01-5	101	3-1
match-0(	WINTER 4	NA-WEST	最終	CHAMPI HIP	102	104	2024-01-0	102	3-2
match-0(	WINTER 4	NA-WEST	SEMIFIN S	UPPER	102	106	2024-01-8	102	3-1
match-0(	SPRING 4	NA-EAST	QUARTE NALS	UPPER	101	108	2024-03-5	101	3-0
match-0(	SPRING 4	NA-EAST	QUARTE NALS	LOWER	103	110	2024-03-5	103	3-2

GSI：TournamentRegionIndex（多屬性索引鍵）

tourname tId (PK)	區域 (PK)	round (SK)	括號 (SK)	matchId (SK)	player1Id	player2Id	matchDa	優勝者	分數
WINTER 4	NA-EAST	最終	CHAMPI HIP	match-0(	101	103	2024-01-0	101	3-1
WINTER 4	NA-EAST	QUARTE NALS	UPPER	match-0(	101	109	2024-01-5	101	3-1
WINTER 4	NA-EAST	SEMIFIN S	UPPER	match-0(	101	105	2024-01-8	101	3-2

tourname tId (PK)	區域 (PK)	round (SK)	括號 (SK)	matchId (SK)	player1Id	player2Id	matchDa	優勝者	分數
WINTER 4	NA-EAST	SEMIFIN S	UPPER	match-0(	103	107	2024-01-8	103	3-0
WINTER 4	NA-WEST	最終	CHAMPI HIP	match-0(	102	104	2024-01-0	102	3-2
WINTER 4	NA-WEST	SEMIFIN S	UPPER	match-0(	102	106	2024-01-8	102	3-1
SPRING 4	NA-EAST	QUARTE NALS	LOWER	match-0(	103	110	2024-03-5	103	3-2
SPRING 4	NA-EAST	QUARTE NALS	UPPER	match-0(	101	108	2024-03-5	101	3-0

GSI： PlayerMatchHistoryIndex（多屬性索引鍵）

player1Id (PK)	matchDa (SK)	round (SK)	tourname tId	region	括號	matchId	player2Id	優勝者	分數
101	2024-01-5	QUARTE NALS	WINTER 4	NA-EAST	UPPER	match-0(	109	101	3-1
101	2024-01-8	SEMIFIN S	WINTER 4	NA-EAST	UPPER	match-0(	105	101	3-2
101	2024-01-0	最終	WINTER 4	NA-EAST	CHAMPI HIP	match-0(	103	101	3-1
101	2024-03-5	QUARTE NALS	SPRING 4	NA-EAST	UPPER	match-0(	108	101	3-0
102	2024-01-8	SEMIFIN S	WINTER 4	NA-WEST	UPPER	match-0(	106	102	3-1

player1Id (PK)	matchDa (SK)	round (SK)	tourname tId	region	括號	matchId	player2Id	優勝者	分數
102	2024-01-0	最終	WINTER 4	NA-WEST	CHAMPI HIP	match-0(	104	102	3-2
103	2024-01-8	SEMIFIN S	WINTER 4	NA-EAST	UPPER	match-0(	107	103	3-0
103	2024-03-5	QUARTE NALS	SPRING 4	NA-EAST	LOWER	match-0(	110	103	3-2

先決條件

開始前，請確保您具備以下條件：

帳戶和許可

- 作用中 AWS 帳戶 （視需要在此處建立一個)
- DynamoDB 操作的 IAM 許可：
  - `dynamodb:CreateTable`
  - `dynamodb:DeleteTable`
  - `dynamodb:DescribeTable`
  - `dynamodb:PutItem`
  - `dynamodb:Query`
  - `dynamodb:BatchWriteItem`

> **Note**
>
> 安全備註：針對生產用途，建立僅具有所需許可的自訂 IAM 政策。在本教學課程中，您可以使用 AWS 受管政策 `AmazonDynamoDBFullAccessV2`。

開發環境

- 安裝在機器上的 Node.js

- AWS 使用下列其中一種方法設定的登入資料：

選項 1： AWS CLI

```
aws configure
```

選項 2：環境變數

```
export AWS_ACCESS_KEY_ID=your_access_key_here
export AWS_SECRET_ACCESS_KEY=your_secret_key_here
export AWS_DEFAULT_REGION=us-east-1
```

安裝必要的套件

```
npm install @aws-sdk/client-dynamodb @aws-sdk/lib-dynamodb
```

實作

步驟 1：使用多屬性索引鍵建立具有 GSIs資料表

使用簡單的基本金鑰結構和使用多屬性金鑰GSIs 建立資料表。

程式碼範例

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

const client = new DynamoDBClient({ region: 'us-west-2' });

const response = await client.send(new CreateTableCommand({
    TableName: 'TournamentMatches',

    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'matchId', KeyType: 'HASH' }              // Simple PK
    ],

    AttributeDefinitions: [
        { AttributeName: 'matchId', AttributeType: 'S' },
        { AttributeName: 'tournamentId', AttributeType: 'S' },
        { AttributeName: 'region', AttributeType: 'S' },
```

```
        { AttributeName: 'round', AttributeType: 'S' },
        { AttributeName: 'bracket', AttributeType: 'S' },
        { AttributeName: 'player1Id', AttributeType: 'S' },
        { AttributeName: 'matchDate', AttributeType: 'S' }
    ],

    // GSIs with multi-attribute keys
    GlobalSecondaryIndexes: [
        {
            IndexName: 'TournamentRegionIndex',
            KeySchema: [
                { AttributeName: 'tournamentId', KeyType: 'HASH' },    // GSI PK
 attribute 1
                { AttributeName: 'region', KeyType: 'HASH' },          // GSI PK
 attribute 2
                { AttributeName: 'round', KeyType: 'RANGE' },          // GSI SK
 attribute 1
                { AttributeName: 'bracket', KeyType: 'RANGE' },        // GSI SK
 attribute 2
                { AttributeName: 'matchId', KeyType: 'RANGE' }         // GSI SK
 attribute 3
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'PlayerMatchHistoryIndex',
            KeySchema: [
                { AttributeName: 'player1Id', KeyType: 'HASH' },       // GSI PK
                { AttributeName: 'matchDate', KeyType: 'RANGE' },      // GSI SK
 attribute 1
                { AttributeName: 'round', KeyType: 'RANGE' }           // GSI SK
 attribute 2
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],

    BillingMode: 'PAY_PER_REQUEST'
}));

console.log("Table with multi-attribute GSI keys created successfully");
```

關鍵設計決策：

基礎資料表：基礎資料表使用簡單的matchId分割區索引鍵進行直接比對查詢，讓基礎資料表結構保持直接，同時 GSIs 提供複雜的查詢模式。

TournamentRegionIndex 全域次要索引：TournamentRegionIndex全域次要索引使用tournamentId + region作為多屬性分割區索引鍵，建立競爭區域隔離，其中資料會依兩個屬性的雜湊合併分佈，在特定競爭區域內容中啟用有效的查詢。多屬性排序索引鍵 (round + bracket + matchId) 提供階層排序，支援階層任何層級的查詢，自然排序從一般（圓形）到特定（相符 ID)。

PlayerMatchHistoryIndex 全域次要索引：PlayerMatchHistoryIndex全域次要索引使用player1Id做為分割區索引鍵，依玩家重新組織資料，為特定玩家啟用跨旅程查詢。多屬性排序索引鍵 (matchDate + round) 提供依時間順序排序的功能，可依日期範圍或特定競賽回合進行篩選。

步驟 2：插入具有原生屬性的資料

使用自然屬性新增競賽配對資料。GSI 會自動為這些屬性編製索引，而不需要合成索引鍵。

程式碼範例

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

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Tournament match data - no synthetic keys needed for GSIs
const matches = [
    // Winter 2024 Tournament, NA-EAST region
    {
        matchId: 'match-001',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'FINALS',
        bracket: 'CHAMPIONSHIP',
        player1Id: '101',
        player2Id: '103',
        matchDate: '2024-01-20',
        winner: '101',
        score: '3-1'
    },
    {
        matchId: 'match-002',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
```

```
        round: 'SEMIFINALS',
        bracket: 'UPPER',
        player1Id: '101',
        player2Id: '105',
        matchDate: '2024-01-18',
        winner: '101',
        score: '3-2'
    },
    {
        matchId: 'match-003',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'SEMIFINALS',
        bracket: 'UPPER',
        player1Id: '103',
        player2Id: '107',
        matchDate: '2024-01-18',
        winner: '103',
        score: '3-0'
    },
    {
        matchId: 'match-004',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'QUARTERFINALS',
        bracket: 'UPPER',
        player1Id: '101',
        player2Id: '109',
        matchDate: '2024-01-15',
        winner: '101',
        score: '3-1'
    },

    // Winter 2024 Tournament, NA-WEST region
    {
        matchId: 'match-005',
        tournamentId: 'WINTER2024',
        region: 'NA-WEST',
        round: 'FINALS',
        bracket: 'CHAMPIONSHIP',
        player1Id: '102',
        player2Id: '104',
        matchDate: '2024-01-20',
        winner: '102',
```

```
        score: '3-2'
    },
    {
        matchId: 'match-006',
        tournamentId: 'WINTER2024',
        region: 'NA-WEST',
        round: 'SEMIFINALS',
        bracket: 'UPPER',
        player1Id: '102',
        player2Id: '106',
        matchDate: '2024-01-18',
        winner: '102',
        score: '3-1'
    },

    // Spring 2024 Tournament, NA-EAST region
    {
        matchId: 'match-007',
        tournamentId: 'SPRING2024',
        region: 'NA-EAST',
        round: 'QUARTERFINALS',
        bracket: 'UPPER',
        player1Id: '101',
        player2Id: '108',
        matchDate: '2024-03-15',
        winner: '101',
        score: '3-0'
    },
    {
        matchId: 'match-008',
        tournamentId: 'SPRING2024',
        region: 'NA-EAST',
        round: 'QUARTERFINALS',
        bracket: 'LOWER',
        player1Id: '103',
        player2Id: '110',
        matchDate: '2024-03-15',
        winner: '103',
        score: '3-2'
    }
];

// Insert all matches
for (const match of matches) {
```

```
    await docClient.send(new PutCommand({
        TableName: 'TournamentMatches',
        Item: match
    }));

    console.log(`Added: ${match.matchId} - ${match.tournamentId}/${match.region} -
 ${match.round} ${match.bracket}`);
}

console.log(`\nInserted ${matches.length} tournament matches`);
console.log("No synthetic keys created - GSIs use native attributes automatically");
```

資料結構說明：

自然屬性用量：每個屬性代表一個真正的競賽概念，不需要字串串連或剖析，提供直接映射到網域模型。

自動全域次要索引索引：GSIs 會使用現有屬性 (tournamentId、regionroundbracket、、matchId代表 TournamentRegionIndex，以及 player1Id、 round代表 PlayerMatchHistoryIndex) 自動索引項目matchDate，而不需要合成串連索引鍵。

不需要回填：當您將具有多屬性索引鍵的新全域次要索引新增至現有資料表時，DynamoDB 會使用其自然屬性自動為所有現有項目編製索引，而不需要使用合成索引鍵更新項目。

步驟 3：查詢具有所有分割區索引鍵屬性的 TournamentRegionIndex 全域次要索引

此範例會查詢具有多屬性分割區索引鍵 (tournamentId +) 的 TournamentRegionIndex 全域次要索引region。所有分割區索引鍵屬性都必須在查詢中指定等式條件 - 您無法tournamentId單獨使用 查詢，或在分割區索引鍵屬性上使用不等式運算子。

程式碼範例

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

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query GSI: All matches for WINTER2024 tournament in NA-EAST region
const response = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region',
```

```
    ExpressionAttributeNames: {
        '#region': 'region',  // 'region' is a reserved keyword
        '#tournament': 'tournament'
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST'
    }
}));

console.log(`Found ${response.Items.length} matches for WINTER2024/NA-EAST:\n`);
response.Items.forEach(match => {
    console.log(`  ${match.round} | ${match.bracket} | ${match.matchId}`);
    console.log(`    Players: ${match.player1Id} vs ${match.player2Id}`);
    console.log(`    Winner: ${match.winner}, Score: ${match.score}\n`);
});
```

預期的輸出：

```
Found 4 matches for WINTER2024/NA-EAST:

  FINALS | CHAMPIONSHIP | match-001
    Players: 101 vs 103
    Winner: 101, Score: 3-1

  QUARTERFINALS | UPPER | match-004
    Players: 101 vs 109
    Winner: 101, Score: 3-1

  SEMIFINALS | UPPER | match-002
    Players: 101 vs 105
    Winner: 101, Score: 3-2

  SEMIFINALS | UPPER | match-003
    Players: 103 vs 107
    Winner: 103, Score: 3-0
```

無效查詢：

```
// Missing region attribute
KeyConditionExpression: 'tournamentId = :tournament'

// Using inequality on partition key attribute
```

```
KeyConditionExpression: 'tournamentId = :tournament AND #region > :region'
```

效能：多屬性分割區索引鍵會雜湊在一起，提供與單一屬性索引鍵相同的 O(1) 查詢效能。

步驟 4：從left-to-right查詢全域次要索引排序索引鍵

排序索引鍵屬性必須依全域次要索引中定義的順序left-to-right查詢。此範例示範在不同階層層級查詢 TournamentRegionIndex：僅依 `round`篩選、依 `round` + 篩選`bracket`，或依所有三個排序索引鍵屬性篩選。您無法略過中間的屬性，例如，您無法在略過 `matchId`時透過 `round`和 查詢`bracket`。

程式碼範例

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

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query 1: Filter by first sort key attribute (round)
console.log("Query 1: All SEMIFINALS matches");
const query1 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS'
    }
}));
console.log(`  Found ${query1.Items.length} matches\n`);

// Query 2: Filter by first two sort key attributes (round + bracket)
console.log("Query 2: SEMIFINALS UPPER bracket matches");
const query2 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round AND bracket = :bracket',
```

```
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS',
        ':bracket': 'UPPER'
    }
}));
console.log(`  Found ${query2.Items.length} matches\n`);

// Query 3: Filter by all three sort key attributes (round + bracket + matchId)
console.log("Query 3: Specific match in SEMIFINALS UPPER bracket");
const query3 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round AND bracket = :bracket AND matchId = :matchId',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS',
        ':bracket': 'UPPER',
        ':matchId': 'match-002'
    }
}));
console.log(`  Found ${query3.Items.length} matches\n`);

// Query 4: INVALID - skipping round
console.log("Query 4: Attempting to skip first sort key attribute (WILL FAIL)");
try {
    const query4 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'TournamentRegionIndex',
        KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND
 bracket = :bracket',
        ExpressionAttributeNames: {
            '#region': 'region'  // 'region' is a reserved keyword
        },
        ExpressionAttributeValues: {
```

```
            ':tournament': 'WINTER2024',
            ':region': 'NA-EAST',
            ':bracket': 'UPPER'
        }
    }));
} catch (error) {
    console.log(`  Error: ${error.message}`);
    console.log(`  Cannot skip sort key attributes - must query left-to-right\n`);
}
```

預期的輸出：

```
Query 1: All SEMIFINALS matches
  Found 2 matches

Query 2: SEMIFINALS UPPER bracket matches
  Found 2 matches

Query 3: Specific match in SEMIFINALS UPPER bracket
  Found 1 matches

Query 4: Attempting to skip first sort key attribute (WILL FAIL)
  Error: Query key condition not supported
  Cannot skip sort key attributes - must query left-to-right
```

Left-to-right查詢規則：您必須從左至右依序查詢屬性，而不略過任何屬性。

有效模式：

- 僅限第一個屬性：`round = 'SEMIFINALS'`
- 前兩個屬性：`round = 'SEMIFINALS' AND bracket = 'UPPER'`
- 全部三個屬性：`round = 'SEMIFINALS' AND bracket = 'UPPER' AND matchId = 'match-002'`

無效的模式：

- 略過第一個屬性：`bracket = 'UPPER'`（略過四捨五入）
- 查詢順序不正確：`matchId = 'match-002' AND round = 'SEMIFINALS'`
- 離開間隙：`round = 'SEMIFINALS' AND matchId = 'match-002'`（略過括號）

> **Note**
>
> 設計秘訣：從最一般到最具體的排序索引鍵屬性，以最大化查詢彈性。

步驟 5：在全域次要索引排序索引鍵上使用不等式條件

不等式條件必須是查詢中的最後一個條件。此範例示範在排序索引鍵屬性上使用比較運算子 (>=、BETWEEN) 和字首比對 (begins_with( ))。使用不等式運算子之後，就無法再新增任何其他排序索引鍵條件，不等式必須是索引鍵條件表達式中的最終條件。

程式碼範例

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

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query 1: Round comparison (inequality on first sort key attribute)
console.log("Query 1: Matches from QUARTERFINALS onwards");
const query1 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 >= :round',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'QUARTERFINALS'
    }
}));
console.log(`  Found ${query1.Items.length} matches\n`);

// Query 2: Round range with BETWEEN
console.log("Query 2: Matches between QUARTERFINALS and SEMIFINALS");
const query2 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
```

```
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 BETWEEN :start AND :end',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':start': 'QUARTERFINALS',
        ':end': 'SEMIFINALS'
    }
}));
console.log(`  Found ${query2.Items.length} matches\n`);

// Query 3: Prefix matching with begins_with (treated as inequality)
console.log("Query 3: Matches in brackets starting with 'U'");
const query3 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round AND begins_with(bracket, :prefix)',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS',
        ':prefix': 'U'
    }
}));
console.log(`  Found ${query3.Items.length} matches\n`);

// Query 4: INVALID - condition after inequality
console.log("Query 4: Attempting condition after inequality (WILL FAIL)");
try {
    const query4 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'TournamentRegionIndex',
        KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND
 round > :round AND bracket = :bracket',
        ExpressionAttributeNames: {
            '#region': 'region'  // 'region' is a reserved keyword
        },
```

```
        ExpressionAttributeValues: {
            ':tournament': 'WINTER2024',
            ':region': 'NA-EAST',
            ':round': 'QUARTERFINALS',
            ':bracket': 'UPPER'
        }
    }));
} catch (error) {
    console.log(`  Error: ${error.message}`);
    console.log(`  Cannot add conditions after inequality - it must be last\n`);
}
```

不等式運算子規則：您可以將比較運算子 (>、>=<、、<=)、 BETWEEN用於範圍查詢，以及 begins_with()用於字首比對。不等式必須是查詢中的最後一個條件。

有效模式：

- 等式條件後接不等式： `round = 'SEMIFINALS' AND bracket = 'UPPER' AND matchId > 'match-001'`
- 第一個屬性上的不等式： `round BETWEEN 'QUARTERFINALS' AND 'SEMIFINALS'`
- 字首比對為最終條件： `round = 'SEMIFINALS' AND begins_with(bracket, 'U')`

無效的模式：

- 在不等式之後新增條件： `round > 'QUARTERFINALS' AND bracket = 'UPPER'`
- 使用多個不等式： `round > 'QUARTERFINALS' AND bracket > 'L'`

> **Important**
>
> begins_with() 會被視為不等式條件，因此沒有任何額外的排序索引鍵條件可以遵循它。

步驟 6：使用多屬性排序索引鍵查詢 PlayerMatchHistoryIndex 全域次要索引

此範例會查詢具有單一分割區索引鍵 (player1Id) 和多屬性排序索引鍵 (matchDate +) 的 PlayerMatchHistoryIndexround。這可透過查詢特定玩家的所有配對而不了解賽事 IDs，而基礎資料表則需要每個賽事區域組合個別的查詢。

程式碼範例

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

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query 1: All matches for Player 101 across all tournaments
console.log("Query 1: All matches for Player 101");
const query1 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player',
    ExpressionAttributeValues: {
        ':player': '101'
    }
}));

console.log(`  Found ${query1.Items.length} matches for Player 101:`);
query1.Items.forEach(match => {
    console.log(`    ${match.tournamentId}/${match.region} - ${match.matchDate} -
 ${match.round}`);
});
console.log();

// Query 2: Player 101 matches on specific date
console.log("Query 2: Player 101 matches on 2024-01-18");
const query2 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player AND matchDate = :date',
    ExpressionAttributeValues: {
        ':player': '101',
        ':date': '2024-01-18'
    }
}));

console.log(`  Found ${query2.Items.length} matches\n`);

// Query 3: Player 101 SEMIFINALS matches on specific date
console.log("Query 3: Player 101 SEMIFINALS matches on 2024-01-18");
const query3 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
```

```
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player AND matchDate = :date AND round
 = :round',
    ExpressionAttributeValues: {
        ':player': '101',
        ':date': '2024-01-18',
        ':round': 'SEMIFINALS'
    }
}));

console.log(`  Found ${query3.Items.length} matches\n`);

// Query 4: Player 101 matches in date range
console.log("Query 4: Player 101 matches in January 2024");
const query4 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player AND matchDate BETWEEN :start
 AND :end',
    ExpressionAttributeValues: {
        ':player': '101',
        ':start': '2024-01-01',
        ':end': '2024-01-31'
    }
}));

console.log(`  Found ${query4.Items.length} matches\n`);
```

模式變化

具有多屬性索引鍵的時間序列資料

使用階層式時間屬性最佳化時間序列查詢

程式碼範例

```
{
    TableName: 'IoTReadings',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'readingId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'readingId', AttributeType: 'S' },
```

```
        { AttributeName: 'deviceId', AttributeType: 'S' },
        { AttributeName: 'locationId', AttributeType: 'S' },
        { AttributeName: 'year', AttributeType: 'S' },
        { AttributeName: 'month', AttributeType: 'S' },
        { AttributeName: 'day', AttributeType: 'S' },
        { AttributeName: 'timestamp', AttributeType: 'S' }
    ],
    // GSI with multi-attribute keys for time-series queries
    GlobalSecondaryIndexes: [{
        IndexName: 'DeviceLocationTimeIndex',
        KeySchema: [
            { AttributeName: 'deviceId', KeyType: 'HASH' },
            { AttributeName: 'locationId', KeyType: 'HASH' },
            { AttributeName: 'year', KeyType: 'RANGE' },
            { AttributeName: 'month', KeyType: 'RANGE' },
            { AttributeName: 'day', KeyType: 'RANGE' },
            { AttributeName: 'timestamp', KeyType: 'RANGE' }
        ],
        Projection: { ProjectionType: 'ALL' }
    }],
    BillingMode: 'PAY_PER_REQUEST'
}

// Query patterns enabled via GSI:
// - All readings for device in location
// - Readings for specific year
// - Readings for specific month in year
// - Readings for specific day
// - Readings in time range
```

優點：自然時間階層（年 → 月 → 日 → 時間戳記）可隨時啟用有效率的查詢，而無需日期剖析或操作。全域次要索引會使用其自然時間屬性自動為所有讀數編製索引。

具有多屬性索引鍵的電子商務訂單

追蹤具有多個維度的訂單

程式碼範例

```
{
    TableName: 'Orders',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'orderId', KeyType: 'HASH' }
```

```
    ],
    AttributeDefinitions: [
        { AttributeName: 'orderId', AttributeType: 'S' },
        { AttributeName: 'sellerId', AttributeType: 'S' },
        { AttributeName: 'region', AttributeType: 'S' },
        { AttributeName: 'orderDate', AttributeType: 'S' },
        { AttributeName: 'category', AttributeType: 'S' },
        { AttributeName: 'customerId', AttributeType: 'S' },
        { AttributeName: 'orderStatus', AttributeType: 'S' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'SellerRegionIndex',
            KeySchema: [
                { AttributeName: 'sellerId', KeyType: 'HASH' },
                { AttributeName: 'region', KeyType: 'HASH' },
                { AttributeName: 'orderDate', KeyType: 'RANGE' },
                { AttributeName: 'category', KeyType: 'RANGE' },
                { AttributeName: 'orderId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'CustomerOrdersIndex',
            KeySchema: [
                { AttributeName: 'customerId', KeyType: 'HASH' },
                { AttributeName: 'orderDate', KeyType: 'RANGE' },
                { AttributeName: 'orderStatus', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// SellerRegionIndex GSI queries:
// - Orders by seller and region
// - Orders by seller, region, and date
// - Orders by seller, region, date, and category

// CustomerOrdersIndex GSI queries:
// - Customer's orders
// - Customer's orders by date
```

```
// - Customer's orders by date and status
```

階層組織資料

模型組織階層

程式碼範例

```
{
    TableName: 'Employees',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'employeeId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'employeeId', AttributeType: 'S' },
        { AttributeName: 'companyId', AttributeType: 'S' },
        { AttributeName: 'divisionId', AttributeType: 'S' },
        { AttributeName: 'departmentId', AttributeType: 'S' },
        { AttributeName: 'teamId', AttributeType: 'S' },
        { AttributeName: 'skillCategory', AttributeType: 'S' },
        { AttributeName: 'skillLevel', AttributeType: 'S' },
        { AttributeName: 'yearsExperience', AttributeType: 'N' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'OrganizationIndex',
            KeySchema: [
                { AttributeName: 'companyId', KeyType: 'HASH' },
                { AttributeName: 'divisionId', KeyType: 'HASH' },
                { AttributeName: 'departmentId', KeyType: 'RANGE' },
                { AttributeName: 'teamId', KeyType: 'RANGE' },
                { AttributeName: 'employeeId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'SkillsIndex',
            KeySchema: [
                { AttributeName: 'skillCategory', KeyType: 'HASH' },
                { AttributeName: 'skillLevel', KeyType: 'RANGE' },
                { AttributeName: 'yearsExperience', KeyType: 'RANGE' }
            ],
```

```
            Projection: { ProjectionType: 'INCLUDE', NonKeyAttributes: ['employeeId',
 'name'] }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// OrganizationIndex GSI query patterns:
// - All employees in company/division
// - Employees in specific department
// - Employees in specific team

// SkillsIndex GSI query patterns:
// - Employees by skill and experience level
```

稀疏多屬性索引鍵

結合多屬性索引鍵來製作稀疏的 GSI

程式碼範例

```
{
    TableName: 'Products',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'productId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'productId', AttributeType: 'S' },
        { AttributeName: 'categoryId', AttributeType: 'S' },
        { AttributeName: 'subcategoryId', AttributeType: 'S' },
        { AttributeName: 'averageRating', AttributeType: 'N' },
        { AttributeName: 'reviewCount', AttributeType: 'N' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'CategoryIndex',
            KeySchema: [
                { AttributeName: 'categoryId', KeyType: 'HASH' },
                { AttributeName: 'subcategoryId', KeyType: 'HASH' },
                { AttributeName: 'productId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
```

```
        {
            IndexName: 'ReviewedProductsIndex',
            KeySchema: [
                { AttributeName: 'categoryId', KeyType: 'HASH' },
                { AttributeName: 'averageRating', KeyType: 'RANGE' },  // Optional
 attribute
                { AttributeName: 'reviewCount', KeyType: 'RANGE' }     // Optional
 attribute
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// Only products with reviews appear in ReviewedProductsIndex GSI
// Automatic filtering without application logic
// Multi-attribute sort key enables rating and count queries
```

SaaS 多租戶

具有客戶隔離的多租戶 SaaS 平台

程式碼範例

```
// Table design
{
    TableName: 'SaasData',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'resourceId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'resourceId', AttributeType: 'S' },
        { AttributeName: 'tenantId', AttributeType: 'S' },
        { AttributeName: 'customerId', AttributeType: 'S' },
        { AttributeName: 'resourceType', AttributeType: 'S' }
    ],
    // GSI with multi-attribute keys for tenant-customer isolation
    GlobalSecondaryIndexes: [{
        IndexName: 'TenantCustomerIndex',
        KeySchema: [
            { AttributeName: 'tenantId', KeyType: 'HASH' },
            { AttributeName: 'customerId', KeyType: 'HASH' },
```

```
            { AttributeName: 'resourceType', KeyType: 'RANGE' },
            { AttributeName: 'resourceId', KeyType: 'RANGE' }
        ],
        Projection: { ProjectionType: 'ALL' }
    }],
    BillingMode: 'PAY_PER_REQUEST'
}

// Query GSI: All resources for tenant T001, customer C001
const resources = await docClient.send(new QueryCommand({
    TableName: 'SaasData',
    IndexName: 'TenantCustomerIndex',
    KeyConditionExpression: 'tenantId = :tenant AND customerId = :customer',
    ExpressionAttributeValues: {
        ':tenant': 'T001',
        ':customer': 'C001'
    }
}));

// Query GSI: Specific resource type for tenant/customer
const documents = await docClient.send(new QueryCommand({
    TableName: 'SaasData',
    IndexName: 'TenantCustomerIndex',
    KeyConditionExpression: 'tenantId = :tenant AND customerId = :customer AND
 resourceType = :type',
    ExpressionAttributeValues: {
        ':tenant': 'T001',
        ':customer': 'C001',
        ':type': 'document'
    }
}));
```

優點：租戶-客戶內容和自然資料組織內的高效率查詢。

金融交易

銀行系統使用 GSIs追蹤帳戶交易

程式碼範例

```
// Table design
{
    TableName: 'BankTransactions',
    // Base table: Simple partition key
```

```
    KeySchema: [
        { AttributeName: 'transactionId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'transactionId', AttributeType: 'S' },
        { AttributeName: 'accountId', AttributeType: 'S' },
        { AttributeName: 'year', AttributeType: 'S' },
        { AttributeName: 'month', AttributeType: 'S' },
        { AttributeName: 'day', AttributeType: 'S' },
        { AttributeName: 'transactionType', AttributeType: 'S' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'AccountTimeIndex',
            KeySchema: [
                { AttributeName: 'accountId', KeyType: 'HASH' },
                { AttributeName: 'year', KeyType: 'RANGE' },
                { AttributeName: 'month', KeyType: 'RANGE' },
                { AttributeName: 'day', KeyType: 'RANGE' },
                { AttributeName: 'transactionId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'TransactionTypeIndex',
            KeySchema: [
                { AttributeName: 'accountId', KeyType: 'HASH' },
                { AttributeName: 'transactionType', KeyType: 'RANGE' },
                { AttributeName: 'year', KeyType: 'RANGE' },
                { AttributeName: 'month', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// Query AccountTimeIndex GSI: All transactions for account in 2023
const yearTransactions = await docClient.send(new QueryCommand({
    TableName: 'BankTransactions',
    IndexName: 'AccountTimeIndex',
    KeyConditionExpression: 'accountId = :account AND #year = :year',
    ExpressionAttributeNames: { '#year': 'year' },
    ExpressionAttributeValues: {
```

```
        ':account': 'ACC-12345',
        ':year': '2023'
    }
}));

// Query AccountTimeIndex GSI: Transactions in specific month
const monthTransactions = await docClient.send(new QueryCommand({
    TableName: 'BankTransactions',
    IndexName: 'AccountTimeIndex',
    KeyConditionExpression: 'accountId = :account AND #year = :year AND #month
 = :month',
    ExpressionAttributeNames: { '#year': 'year', '#month': 'month' },
    ExpressionAttributeValues: {
        ':account': 'ACC-12345',
        ':year': '2023',
        ':month': '11'
    }
}));

// Query TransactionTypeIndex GSI: Deposits in 2023
const deposits = await docClient.send(new QueryCommand({
    TableName: 'BankTransactions',
    IndexName: 'TransactionTypeIndex',
    KeyConditionExpression: 'accountId = :account AND transactionType = :type AND #year
 = :year',
    ExpressionAttributeNames: { '#year': 'year' },
    ExpressionAttributeValues: {
        ':account': 'ACC-12345',
        ':type': 'deposit',
        ':year': '2023'
    }
}));
```

完成範例

下列範例示範從設定到清除的多屬性金鑰：

程式碼範例

```
import {
    DynamoDBClient,
    CreateTableCommand,
    DeleteTableCommand,
    waitUntilTableExists
```

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

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

async function multiAttributeKeysDemo() {
    console.log("Starting Multi-Attribute GSI Keys Demo\n");

    // Step 1: Create table with GSIs using multi-attribute keys
    console.log("1. Creating table with multi-attribute GSI keys...");
    await client.send(new CreateTableCommand({
        TableName: 'TournamentMatches',
        KeySchema: [
            { AttributeName: 'matchId', KeyType: 'HASH' }
        ],
        AttributeDefinitions: [
            { AttributeName: 'matchId', AttributeType: 'S' },
            { AttributeName: 'tournamentId', AttributeType: 'S' },
            { AttributeName: 'region', AttributeType: 'S' },
            { AttributeName: 'round', AttributeType: 'S' },
            { AttributeName: 'bracket', AttributeType: 'S' },
            { AttributeName: 'player1Id', AttributeType: 'S' },
            { AttributeName: 'matchDate', AttributeType: 'S' }
        ],
        GlobalSecondaryIndexes: [
            {
                IndexName: 'TournamentRegionIndex',
                KeySchema: [
                    { AttributeName: 'tournamentId', KeyType: 'HASH' },
                    { AttributeName: 'region', KeyType: 'HASH' },
                    { AttributeName: 'round', KeyType: 'RANGE' },
                    { AttributeName: 'bracket', KeyType: 'RANGE' },
                    { AttributeName: 'matchId', KeyType: 'RANGE' }
                ],
                Projection: { ProjectionType: 'ALL' }
            },
            {
                IndexName: 'PlayerMatchHistoryIndex',
                KeySchema: [
```

```
                    { AttributeName: 'player1Id', KeyType: 'HASH' },
                    { AttributeName: 'matchDate', KeyType: 'RANGE' },
                    { AttributeName: 'round', KeyType: 'RANGE' }
                ],
                Projection: { ProjectionType: 'ALL' }
            }
        ],
        BillingMode: 'PAY_PER_REQUEST'
    }));

    await waitUntilTableExists({ client, maxWaitTime: 120 }, { TableName:
'TournamentMatches' });
    console.log("Table created\n");

    // Step 2: Insert tournament matches
    console.log("2. Inserting tournament matches...");
    const matches = [
        { matchId: 'match-001', tournamentId: 'WINTER2024', region: 'NA-EAST', round:
'FINALS', bracket: 'CHAMPIONSHIP', player1Id: '101', player2Id: '103', matchDate:
'2024-01-20', winner: '101', score: '3-1' },
        { matchId: 'match-002', tournamentId: 'WINTER2024', region: 'NA-EAST', round:
'SEMIFINALS', bracket: 'UPPER', player1Id: '101', player2Id: '105', matchDate:
'2024-01-18', winner: '101', score: '3-2' },
        { matchId: 'match-003', tournamentId: 'WINTER2024', region: 'NA-WEST', round:
'FINALS', bracket: 'CHAMPIONSHIP', player1Id: '102', player2Id: '104', matchDate:
'2024-01-20', winner: '102', score: '3-2' },
        { matchId: 'match-004', tournamentId: 'SPRING2024', region: 'NA-EAST', round:
'QUARTERFINALS', bracket: 'UPPER', player1Id: '101', player2Id: '108', matchDate:
'2024-03-15', winner: '101', score: '3-0' }
    ];

    for (const match of matches) {
        await docClient.send(new PutCommand({ TableName: 'TournamentMatches', Item:
match }));
    }
    console.log(`Inserted ${matches.length} tournament matches\n`);

    // Step 3: Query GSI with multi-attribute partition key
    console.log("3. Query TournamentRegionIndex GSI: WINTER2024/NA-EAST matches");
    const gsiQuery1 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'TournamentRegionIndex',
        KeyConditionExpression: 'tournamentId = :tournament AND #region = :region',
        ExpressionAttributeNames: { '#region': 'region' },
```

```
        ExpressionAttributeValues: { ':tournament': 'WINTER2024', ':region': 'NA-
EAST' }
    }));

    console.log(`  Found ${gsiQuery1.Items.length} matches:`);
    gsiQuery1.Items.forEach(match => {
        console.log(`    ${match.round} - ${match.bracket} - ${match.winner} won`);
    });

    // Step 4: Query GSI with multi-attribute sort key
    console.log("\n4. Query PlayerMatchHistoryIndex GSI: All matches for Player 101");
    const gsiQuery2 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'PlayerMatchHistoryIndex',
        KeyConditionExpression: 'player1Id = :player',
        ExpressionAttributeValues: { ':player': '101' }
    }));

    console.log(`  Found ${gsiQuery2.Items.length} matches for Player 101:`);
    gsiQuery2.Items.forEach(match => {
        console.log(`    ${match.tournamentId}/${match.region} - ${match.matchDate} -
 ${match.round}`);
    });

    console.log("\nDemo complete");
    console.log("No synthetic keys needed - GSIs use native attributes automatically");
}

async function cleanup() {
    console.log("Deleting table...");
    await client.send(new DeleteTableCommand({ TableName: 'TournamentMatches' }));
    console.log("Table deleted");
}

// Run demo
multiAttributeKeysDemo().catch(console.error);

// Uncomment to cleanup:
// cleanup().catch(console.error);
```

最小程式碼 scaffold

程式碼範例

```
// 1. Create table with GSI using multi-attribute keys
await client.send(new CreateTableCommand({
    TableName: 'MyTable',
    KeySchema: [
        { AttributeName: 'id', KeyType: 'HASH' }        // Simple base table PK
    ],
    AttributeDefinitions: [
        { AttributeName: 'id', AttributeType: 'S' },
        { AttributeName: 'attr1', AttributeType: 'S' },
        { AttributeName: 'attr2', AttributeType: 'S' },
        { AttributeName: 'attr3', AttributeType: 'S' },
        { AttributeName: 'attr4', AttributeType: 'S' }
    ],
    GlobalSecondaryIndexes: [{
        IndexName: 'MyGSI',
        KeySchema: [
            { AttributeName: 'attr1', KeyType: 'HASH' },    // GSI PK attribute 1
            { AttributeName: 'attr2', KeyType: 'HASH' },    // GSI PK attribute 2
            { AttributeName: 'attr3', KeyType: 'RANGE' },   // GSI SK attribute 1
            { AttributeName: 'attr4', KeyType: 'RANGE' }    // GSI SK attribute 2
        ],
        Projection: { ProjectionType: 'ALL' }
    }],
    BillingMode: 'PAY_PER_REQUEST'
}));

// 2. Insert items with native attributes (no concatenation needed for GSI)
await docClient.send(new PutCommand({
    TableName: 'MyTable',
    Item: {
        id: 'item-001',
        attr1: 'value1',
        attr2: 'value2',
        attr3: 'value3',
        attr4: 'value4',
        // ... other attributes
    }
}));

// 3. Query GSI with all partition key attributes
await docClient.send(new QueryCommand({
    TableName: 'MyTable',
```

```
    IndexName: 'MyGSI',
    KeyConditionExpression: 'attr1 = :v1 AND attr2 = :v2',
    ExpressionAttributeValues: {
        ':v1': 'value1',
        ':v2': 'value2'
    }
}));

// 4. Query GSI with sort key attributes (left-to-right)
await docClient.send(new QueryCommand({
    TableName: 'MyTable',
    IndexName: 'MyGSI',
    KeyConditionExpression: 'attr1 = :v1 AND attr2 = :v2 AND attr3 = :v3',
    ExpressionAttributeValues: {
        ':v1': 'value1',
        ':v2': 'value2',
        ':v3': 'value3'
    }
}));

// Note: If any attribute name is a DynamoDB reserved keyword, use
 ExpressionAttributeNames:
// KeyConditionExpression: 'attr1 = :v1 AND #attr2 = :v2'
// ExpressionAttributeNames: { '#attr2': 'attr2' }
```

其他資源

- DynamoDB 最佳實務
- 使用資料表和資料
- 全域次要索引
- 查詢和掃描操作

在 DynamoDB 中管理全域次要索引

本節說明如何在 Amazon DynamoDB 中建立、修改和刪除全域次要索引。

主題

- 建立具有全域次要索引的資料表
- 描述資料表上的全域次要索引
- 將全域次要索引新增至現有資料表

- 刪除全域次要索引
- 建立期間修改全域次要索引

建立具有全域次要索引的資料表

若要建立具有一或多個全域次要索引的資料表，請搭配 CreateTable 參數使用 GlobalSecondaryIndexes 操作。為達最大的查詢靈活性，您最多可以為每個資料表建立 20 個全域次要索引 (預設配額)。

您必須指定一個屬性做為分割區索引鍵。您可以選擇為排序索引鍵指定另一個屬性。上述任一索引鍵屬性皆不需與資料表的索引鍵屬性相同。例如，在 GameScores 表中 (請參閱 在 DynamoDB 中使用全域次要索引)，TopScore 和 TopScoreDateTime 都不是索引鍵屬性。您可以建立分割區索引鍵為 TopScore、排序索引鍵為 TopScoreDateTime 的全域次要索引。您可能會使用這類索引來判斷高分與一天中玩遊戲的時間之間是否有相互關聯性。

每個索引鍵屬性都必須是類型為 String、Number 或 Binary 的純量。(而不可以是文件或集合)。您可以將任何資料類型的屬性投影到全域次要索引。這包括純量、文件和集合。如需資料類型的完整清單，請參閱 資料類型。

如果使用的是佈建模式，您必須為索引提供 ProvisionedThroughput 設定，包含 ReadCapacityUnits 和 WriteCapacityUnits。這些佈建輸送量設定與資料表的佈建輸送量設定無關，但運作方式類似。如需詳細資訊，請參閱 全域次要索引的佈建輸送量考量。

全域次要索引會從基礎資料表繼承讀取/寫入容量模式。如需詳細資訊，請參閱切換 DynamoDB 容量模式時的注意事項。

> **Note**
>
> 建立新 GSI 時，檢查您選擇的分割區索引鍵是否會在新索引的分割區索引鍵值之間產生資料或流量的不均勻分佈或小範圍分佈，這一點非常重要。如果發生這種情況，您可能會看到回填和寫入操作同時發生，並限流對基礎資料表的寫入。該服務會採取措施以將此情況的可能性降至最低，但沒有與索引分割區索引鍵、所選擇的投射或主索引鍵稀疏程度有關的客戶資料形狀的洞察。
>
> 如果您懷疑新的全域次要索引在分割區索引鍵值上可能具有小範圍或偏斜的資料或流量分佈，請在將新索引新增到對操作而言重要的資料表之前，考慮以下項目。
>
> - 在應用程式驅動的流量為最少時新增索引可能是最安全的做法。
> - 考慮在基礎資料表和索引上啟用 CloudWatch Contributor Insights。這將為您的流量分佈提供有價值的洞察。

- 在整個過程中觀看 ThrottledRequests、 WriteThrottleEvents和 OnlineIndexPercentageProgress CloudWatch 指標。視需要調整佈建的寫入容量，以便在合理的時間內完成回填，而不會對進行中的操作造成任何重大限流影響。OnlineIndexConsumedWriteCapacity和 OnlineThrottleEvents 預期在索引回填期間會顯示 0。
- 如果您因為寫入限流而發生操作影響，請準備好取消索引建立。

描述資料表上的全域次要索引

若要檢視資料表上所有全域次要索引的狀態，請使用 DescribeTable 操作。回應的 GlobalSecondaryIndexes 部分會顯示資料表上的所有索引，以及每個索引的目前狀態 (IndexStatus)。

全域次要索引的 IndexStatus 會是下列其中一項：

- CREATING：正在建立索引，尚無法使用。
- ACTIVE：索引已可供使用，應用程式可以在索引上執行 Query 操作。
- UPDATING：正在變更索引的佈建輸送量設定。
- DELETING：正在刪除索引，無法再使用。

當 DynamoDB 建置完全域次要索引時，索引狀態會從 CREATING 變更為 ACTIVE。

將全域次要索引新增至現有資料表

若要將全域次要索引新增至現有資料表，請搭配 GlobalSecondaryIndexUpdates 參數使用 UpdateTable 操作。您必須提供下列項目：

- 索引名稱。該名稱在資料表的所有索引中必須是唯一的。
- 索引的索引鍵結構描述。您必須指定一個屬性做為索引的分割區索引鍵，並可以選擇性地指定另一個屬性做為索引的排序索引鍵。上述任一索引鍵屬性皆不需與資料表的索引鍵屬性相同。每個結構描述屬性的資料類型都必須是純量：String、Number 或 Binary。
- 要從資料表投影到索引的屬性：
  - KEYS_ONLY：索引中的每個項目都只包含資料表分割索引鍵和排序索引鍵值，以及索引鍵值。
  - INCLUDE：除了 KEYS_ONLY 中描述的屬性外，次要索引會包含您指定的其他非索引鍵屬性。
  - ALL：索引包含來源資料表中的所有屬性。

- 索引的佈建輸送量設定，包含 ReadCapacityUnits 和 WriteCapacityUnits。這些佈建輸送量設定與資料表的佈建輸送量設定無關。

每個 UpdateTable 操作只能建立一個全域次要索引。

索引建立階段

當您將新的全域次要索引新增至現有資料表時，資料表在索引建置時仍可繼續使用。但新的索引在其狀態從 CREATING 變更為 ACTIVE 前，都不可進行 Query 操作。

> **Note**
>
> 全域次要索引建立不使用 Application Auto Scaling。提高 MIN Application Auto Scaling 容量不會減少全域次要索引的建立時間。

DynamoDB 幕後的索引建置分為兩個階段：

資源配置

DynamoDB 會配置建立索引所需的運算和儲存資源。

在資源配置階段，IndexStatus 屬性是 CREATING，而 Backfilling 屬性是 false。您可使用 DescribeTable 操作擷取資料表的狀態和其所有次要索引。

當索引處於資源配置階段時，您無法刪除索引或刪除其父資料表。您也無法修改索引或資料表的佈建輸送量。您無法新增或刪除資料表上的其他索引。不過，您可以修改這些其他索引的佈建輸送量。

回填

對於資料表中的每個項目，DynamoDB 會根據屬性的投影，判斷將哪一組屬性寫入索引 (KEYS_ONLY、INCLUDE 或 ALL)。然後它會將這些屬性寫入索引。在回填階段期間，DynamoDB 會追蹤正在資料表中新增、刪除或更新的項目。也會適當地在索引中新增、刪除或更新這些項目的屬性。

在回填階段，IndexStatus 屬性會設為 CREATING，而 Backfilling 屬性是 true。您可使用 DescribeTable 操作擷取資料表的狀態和其所有次要索引。

當索引正在回填時，您無法刪除其父資料表。但您仍然可以刪除索引，或修改資料表及其任何全域次要索引的佈建輸送量。

> **Note**
>
> 在回填階段，違規項目的寫入可能一部分成功，而一部分則遭到拒絕。回填後，所有違反新索引鍵結構描述的項目寫入都會遭到拒絕。建議您在回填階段完成後執行 Violation Detector 工具，以偵測和解決任何可能已發生的索引鍵違規情況。如需詳細資訊，請參閱偵測和修正 DynamoDB 中的索引鍵違規情況。

進行資源配置和回填階段時，索引的狀態為 CREATING。在這期間，DynamoDB 會對資料表執行讀取操作。從基礎資料表填入全域次要索引的讀取操作不需支付費用。

當索引建置完成時，其狀態會變更為 ACTIVE。您無法 Query 或 Scan 索引，直到其 ACTIVE 為止。

> **Note**
>
> 在部分情況下，DynamoDB 會因索引鍵違規情況而無法將資料表中的資料寫入索引。如果發生下列情況，就會發生此情形：
>
> - 屬性值的資料類型與索引鍵結構描述資料類型的資料類型不相符。
> - 屬性的大小超過索引鍵屬性的最大長度。
> - 索引鍵屬性具有空字串或空的二進制屬性值。
>
> 索引鍵違規情況不會影響全域次要索引建立。不過，當索引變成 ACTIVE 時，索引中不會出現違規索引鍵。
> DynamoDB 提供獨立工具來尋找和解決這些問題。如需詳細資訊，請參閱 偵測和修正 DynamoDB 中的索引鍵違規情況。

將全域次要索引新增至大型資料表

建置全域次要索引所需的時間取決於幾個因素，例如：

- 資料表的大小
- 資料表中符合納入索引資格的項目數目
- 投影到索引的屬性數目
- 索引建置期間主要資料表上的寫入活動

如果您要將全域次要索引新增至極大的資料表，建立程序可能需要很久才能完成。若要監控進度並判斷索引是否有足夠的寫入容量，請參閱下列 Amazon CloudWatch 指標：

- `OnlineIndexPercentageProgress`

如需與 DynamoDB 相關的 CloudWatch 指標的詳細資訊，請參閱 DynamoDB 指標。

> **Important**
>
> 在建立或更新全域次要索引之前，您可能需要允許提列非常大的資料表。請聯絡 AWS Support 以允許列出您的資料表。

回填索引時，DynamoDB 會使用內部系統容量從資料表讀取。這是要將索引建立的影響降到最小，並確保資料表的讀取容量不會用盡。

刪除全域次要索引

如果您不再需要全域次要索引，可以使用 `UpdateTable` 操作將其刪除。

每個 `UpdateTable` 操作只能刪除一個全域次要索引。

刪除全域次要索引時，父資料表中所有讀取和寫入活動皆不受影響。當刪除正在進行時，仍可修改其他索引上的佈建輸送量。

> **Note**
>
> - 當您使用 `DeleteTable` 動作刪除資料表時，也會刪除該資料表中所有全域次要索引。
> - 您的帳戶將不會因全域次要索引的刪除操作而被收取費用。

建立期間修改全域次要索引

當索引建置時，您可以使用 `DescribeTable` 操作來判斷它所處的階段。索引的描述包含布林值屬性 `Backfilling`，指出 DynamoDB 目前是否正在將資料表中的項目載入索引。如果 `Backfilling` 為 true，則資源配置階段已完成，且正在回填索引。

在回填階段，您可以刪除正在建立的索引。在這段階段，您無法新增或刪除資料表上的其他索引。

> **Note**
>
> 針對已在 `CreateTable` 操作中建立的索引，`Backfilling` 屬性不會出現在 `DescribeTable` 輸出中。如需詳細資訊，請參閱索引建立階段。

偵測和修正 DynamoDB 中的索引鍵違規情況

在建立全域次要索引的回填階段，Amazon DynamoDB 會檢查資料表中的每個項目，判斷是否符合納入索引中的資格。某些項目可能不符合資格，因為這些項目會導致索引鍵違規情況。在這些情況下，項目會保留在資料表中，但索引沒有該項目的對應項目。

出現以下狀況時會發生索引鍵違規情況：

- 屬性值與索引鍵結構描述資料類型之間存在資料類型不相符的問題。例如，假設其中一個項目在 `GameScores` 資料表中有一個類型為 `String` 的值 `TopScore`。如果新增分割區索引鍵為 `TopScore`、類型為 `Number` 的全域次要索引，資料表中的項目會違反索引鍵。
- 來自資料表的屬性數值超過索引鍵屬性的最大長度。分割區索引鍵的最大長度是 2048 個位元組，而排序索引鍵的最大長度是 1024 個位元組。如果資料表中的任何對應屬性值超過這些限制，資料表中的項目將違反索引鍵。

> **Note**
>
> 如果為用作索引鍵的屬性設定了「字串」或「二進制」屬性值，則屬性值的長度必須大於零；否則，資料表中的項目將違反索引鍵。
> 這個工具此時不會標記此索引鍵違規情況。

如果發生索引鍵違規情況，回填階段會繼續，而不會中斷。但是，索引中不會包含任何違規項目。回填階段完成後，所有違反新索引鍵結構描述的項目寫入都會遭到拒絕。

若要識別並修正資料表中違反索引鍵的屬性值，請使用 Violation Detector 工具。若要執行 Violation Detector，請建立組態檔案，指定要掃描的資料表名稱、全域次要索引分割區索引鍵及排序索引鍵的名稱和資料類型，以及發現任何索引鍵違規情況時要採取的動作。Violation Detector 可以在以下兩種不同模式之一執行：

- 偵測模式：偵測索引鍵違規情況。使用偵測模式報告資料表中會導致全域次要索引中發生索引鍵違規情況的項目。(您可以選擇請求在找到這些違規的資料表項目時立即將其刪除。) 偵測模式的輸出會寫入檔案，可供您用其進行進一步分析。
- 校正模式：校正索引鍵違規情況。在校正模式下，Violation Detector 會從偵測模式讀取與輸出檔案格式相同的輸入檔案。校正模式會從輸入檔案讀取紀錄，並且會針對每個紀錄刪除或更新資料表中的對應項目。(請注意，如果您選擇更新項目，則必須編輯輸入檔案並為這些更新設定適當的值。)

下載並執行 Violation Detector

Violation Detector 可以作為可執行的 Java 封存檔 (`.jar` 檔案)，並在 Windows、macOS 或 Linux 電腦上執行。Violation Detector 需要 Java 1.7 (或更新版本) 和 Apache Maven。

- 從 GitHub 下載 Violation Detector

請遵循 `README.md` 檔案中的說明使用 Maven 下載並安裝 Violation Detector。

若要啟動 Violation Detector，請移至您已建立 `ViolationDetector.java` 的目錄，然後輸入下列命令。

```
java -jar ViolationDetector.jar [options]
```

Violation Detector 命令列接受下列選項：

- `-h | --help`：列印 Violation Detector 的使用摘要和選項。
- `-p | --configFilePath value`：Violation Detector 組態檔案的完全合格名稱。如需詳細資訊，請參閱 Violation Detector 組態檔案。
- `-t | --detect value`：偵測資料表中的索引鍵違規情況，並將其寫入 Violation Detector 輸出檔案。如果此參數的值設定為 `keep`，則不會修改具有索引鍵違規情況的項目。如果值設為 `delete`，則會從資料表中刪除具有索引鍵違規情況的項目。
- `-c | --correct value`：從輸入檔案讀取索引鍵違規情況，並對資料表中的項目採取更正動作。如果此參數的值設定為 `update`，則會使用新的非違規值來更新具有索引鍵違規情況的項目。如果值設為 `delete`，則會從資料表中刪除具有索引鍵違規情況的項目。

Violation Detector 組態檔案

在執行時期，Violation Detector 工具需要組態檔案。此檔案中的參數決定 Violation Detector 可以存取哪些 DynamoDB 資源，以及其可以消耗多少佈建的輸送量。下表描述了這些參數。

參數名稱	Description	是否為必要？

awsCredentialsFile	包含您 AWS 憑證的檔案的完全合格名稱。憑證檔案必須採用下列格式：
	
	`accessKey = access_key_id_goes_here`
	`secretKey = secret_key_goes_here`
		是

dynamoDBRegion	資料表所在的 AWS 區域。例如：`us-west-2` 。	是
tableName	所要掃描 DynamoDB 資料表的名稱。	是
gsiHashKeyName	索引分割區索引鍵的名稱。	是
gsiHashKeyType	索引分割區索引鍵的資料類型：`String`、`Number` 或 `Binary`： `S | N | B`	是
gsiRangeKeyName	索引排序索引鍵的名稱。如果索引只有簡易的主索引鍵 (分割區索引鍵)，請勿指定此參數。	否
gsiRangeKeyType	索引排序索引鍵的資料類型：`String`、`Number` 或 `Binary`： `S | N | B` 如果索引只有簡易的主索引鍵 (分割區索引鍵)，請勿指定此參數。	否

參數名稱	Description	是否為必要？
recordDetails	是否將索引鍵違規情況的完整詳細資訊寫入輸出檔案。如果設定為 true (預設值)，則會報告違規項目的完整資訊。如果設定為 false，則只會報告違規情況的數量。	否
recordGsiValueInViolationRecord	是否將違反索引鍵的數值寫入輸出檔案。如果設定為 true (預設值)，則會報告索引鍵值。如果設定為 false，則不會報告索引鍵值。	否
detectionOutputPath	Violation Detector 輸出檔案的完整路徑。此參數支援寫入本機目錄或 Amazon Simple Storage Service (Amazon S3)。範例如下： detectionOutputPath = //*local/path/filename.csv* detectionOutputPath = s3://*bucket*/*filename.csv* 輸出檔案中的資訊會以逗號分隔值 (CSV) 格式顯示。如果您未設定 detectionOutputPath，則輸出檔案會命名為 violation_detection.csv 並寫入目前的工作目錄。	否

參數名稱	Description	是否為必要？
numOfSegments	當 Violation Detector 掃描資料表時，要使用的平行掃描區段數目。預設值為 1，表示會以連續方式掃描資料表。如果數值為 2 或更高，則 Violation Detector 會將資料表分割為許多邏輯區段和相同數量的掃描執行緒。 numOfSegments 的最高設定為 4096。 對於較大的資料表，平行掃描通常比循序掃描更快。此外，如果資料表大小足以跨越多個分割區，則平行掃描會將讀取活動平均分配到多個分割區。如需在 DynamoDB 中執行平行掃描的詳細資訊，請參閱 平行掃描。	否
numOfViolations	索引鍵違規情況寫入輸出檔案的上限。如果設定為 -1 (預設值)，則會掃描整個資料表。如果設定為正整數，則 Violation Detector 會在偵測到該數量的違規情況後停止。	否
numOfRecords	要掃描的資料表中的項目數。如果設定為 -1 (預設值)，則會掃描整個資料表。如果設定為正整數，則 Violation Detector 會在掃描到資料表中該數量的項目後停止。	否

參數名稱	Description	是否為必要？
`readWriteIOPSPercent`	規範資料表掃描期間耗用的已佈建讀取容量單位百分比。有效值範圍從 1 到 100。預設值 (25) 表示 Violation Detector 將消耗不超過資料表佈建的讀取輸送量的 25%。	否
`correctionInputPath`	Violation Detector 修正輸入檔案的完整路徑。如果您在校正模式下執行 Violation Detector，則會使用此檔案的內容來修改或刪除資料表中違反全域次要索引的資料項目。 `correctionInputPath` 檔案的格式與 `detectionOutputPath` 檔案的格式相同。這可讓您在校正模式下將偵測模式的輸出作為輸入處理。	否

參數名稱	Description	是否為必要？
correctionOutputPath	Violation Detector 修正輸出檔案的完整路徑。只有在發生更新錯誤時，才會建立此檔案。 此參數支援寫入本機目錄或 Amazon S3。範例如下： correctionOutputPath = //*local/path/filename.csv* correctionOutputPath = s3://*bucket*/*filename.csv* 輸出檔案中的資訊會以 CSV 格式顯示。如果您未設定 correctionOutputPath ，則輸出檔案會命名為 violation_update_errors.csv 並寫入目前的工作目錄。	否

偵測

若要偵測索引鍵違規情況，請將 Violation Detector 與 --detect 命令列選項搭配使用。若要顯示此選項的運作方式，請考慮 ProductCatalog 資料表。以下是資料表中的項目清單。僅顯示主索引鍵 (Id) 以及 Price 屬性。

ID (主索引鍵)	Price
101	5
102	20

ID (主索引鍵)	Price
103	200
201	100
202	200
203	300
204	400
205	500

所有 Price 的數值均為類型 Number。不過，因為 DynamoDB 不具結構描述，所以可以新增具有非數字 Price 的項目。例如，假設您將另一個項目新增至 ProductCatalog 資料表。

ID (主索引鍵)	Price
999	"Hello"

資料表現在共有九個項目。

現在，您將新的全域次要索引新增至資料表 PriceIndex。此索引的主索引鍵是分割區索引鍵 Price，其類型是 Number。建置索引後，將會包含八個項目，但 ProductCatalog 資料表含有九個項目。造成此差異的原因是數值 "Hello" 的類型為 String，但 PriceIndex 的主索引鍵為類型 Number。此 String 值違反全域次要索引鍵，因此不會出現在索引中。

若要在此情況中使用 Violation Detector，請先建立如下的組態檔案。

```
# 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
```

接著，您可以執行 Violation Detector，如下列範例所示。

```
$  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
```

如果 `recordDetails` 組態參數設為 `true`，Violation Detector 會將每個違規情況的詳細資訊寫入輸出檔案，如下列範例所示。

```
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,,
```

輸出檔案採用 CSV 格式。檔案中的第一行是標頭，後接每個違反索引鍵的項目的一筆紀錄。這些違規情況紀錄的欄位如下所示：

- 資料表雜湊索引鍵 - 資料表中項目的分割區索引鍵值。
- 資料表範圍索引鍵 - 資料表中項目的排序索引鍵值。
- GSI 雜湊索引鍵值 - 全域次要索引的分割區索引鍵值。
- GSI 雜湊索引鍵違規情況類型 - `Type Violation` 或 `Size Violation`。
- GSI 雜湊索引鍵違規情況說明 - 違反的原因。
- GSI 雜湊索引鍵更新值 (針對使用者) - 在校正模式下，屬性的新使用者提供的數值。
- GSI 範圍索引鍵值 - 全域次要索引的排序索引鍵值。
- GSI 範圍索引鍵違規情況類型 - `Type Violation` 或 `Size Violation`。

- GSI 範圍索引鍵違規情況說明 - 違反的原因。
- GSI 範圍索引鍵更新值 (針對使用者) - 在校正模式下，屬性的新使用者提供的數值。
- 更新時刪除空白屬性 (是/否) - 在校正模式下，決定要刪除 (是) 或保留 (否) 資料表中的違規情況，但只有在下列其中一個欄位為空白時：
  - `GSI Hash Key Update Value(FOR USER)`
  - `GSI Range Key Update Value(FOR USER)`

  如果這些欄位中的任何一個並非空白，則 `Delete Blank Attribute When Updating(Y/N)` 不起作用。

> **Note**
>
> 輸出格式可能會因組態檔案和命令列選項而有所不同。例如，如果資料表有簡單的主索引鍵 (沒有排序索引鍵)，則輸出中不會出現任何排序索引鍵欄位。
> 檔案中的違規情況紀錄可能不會以排序順序顯示。

更正

若要校正索引鍵違規情況，請將 Violation Detector 與 `--correct` 命令列選項搭配使用。在校正模式下，Violation Detector 會讀取 `correctionInputPath` 參數指定的輸入檔案。此檔案具有與 `detectionOutputPath` 檔案相同的格式，以便您可以使用來自偵測的輸出作為校正的輸入。

Violation Detector 提供兩種不同的方法來校正索引鍵違規情況：

- 刪除違規情況：刪除具有違反屬性值的資料表項目。
- 更新違規情況：更新資料表項目，以非違規值取代違規屬性。

在任何一種情況下，您都可以使用偵測模式的輸出檔案作為校正模式的輸入。

繼續進行 `ProductCatalog` 範例，假設您想要從資料表刪除違規情況。若要執行此操作，請使用以下命令列。

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --correct delete
```

此時，系統會要求您確認是否要刪除違規項目。

```
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
```

現在 ProductCatalog 和 PriceIndex 都具有相同數量的項目。

使用全域次要索引：Java

您可以使用 適用於 Java 的 AWS SDK 文件 API 建立具有一或多個全域次要索引的 Amazon DynamoDB 資料表、描述資料表上的索引，並使用索引執行查詢。

下列是資料表操作的常用步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 透過建立對應的請求物件，為操作提供必要及選用的參數。
3. 呼叫您在前一步驟中建立之用戶端所提供的適當方法。

主題

- 建立具有全域次要索引的資料表
- 使用全域次要索引描述資料表
- 查詢全域次要索引
- 範例：使用 適用於 Java 的 AWS SDK 文件 API 的全域次要索引

建立具有全域次要索引的資料表

您可在建立資料表的同時建立全域次要索引。若要執行這項操作，請使用 CreateTable，並提供一或多個全域次要索引的規格。以下 Java 程式碼範例會建立資料表來保存天氣資料的相關資訊。分割區索引鍵為 Location，而排序索引鍵為 Date。名為 PrecipIndex 的全域次要索引允許快速存取各個地點的降水資料。

以下是使用 DynamoDB Document API 建立具有全域次要索引的資料表的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 CreateTableRequest 類別的執行個體，以提供請求資訊。

您必須提供資料表名稱、其主索引鍵，以及佈建的輸送量數值。對於全域次要索引，您必須提供索引名稱、其佈建的輸送量設定值、索引排序索引鍵的屬性定義、索引的索引鍵結構描述以及屬性投影。

3. 以參數形式提供請求物件，以便呼叫 createTable 方法。

下列 Java 程式碼範例示範上述步驟。程式碼會建立具有全域次要索引 (PrecipIndex) 的資料表 (WeatherData)。索引分割區索引鍵是 Date，而其排序索引鍵是 Precipitation。所有的資料表屬性都會投影到索引。使用者可以查詢此索引以取得特定日期的天氣資料，可選擇依降水量排序資料。

因為 Precipitation 不是資料表的索引鍵屬性，所以其並非必要項目。不過，沒有 Precipitation 的 WeatherData 項目不會在 PrecipIndex 中顯示。

```
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());
```

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 ACTIVE。之後，您可以開始將資料項目放入資料表中。

使用全域次要索引描述資料表

若要取得資料表上全域次要索引的資訊，請使用 DescribeTable。對於每個索引，您可以存取其名稱、索引鍵結構描述和投影屬性。

以下是存取資料表的全域次要索引資訊的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 Table 類別的執行個體，代表您要進行作業的索引。
3. 在 Table 物件上呼叫 describe 方法。

下列 Java 程式碼範例示範上述步驟。

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());
    }
}
```

查詢全域次要索引

您可以在全域次要索引上使用 Query，與 Query 資料表的方式相同。您需要指定索引名稱、索引分割區索引鍵和排序索引鍵的查詢條件 (如存在)，以及要傳回的屬性。在本例中，索引為 PrecipIndex，其分割區索引鍵為 Date，排序索引鍵為 Precipitation。索引查詢會傳回特定日期的所有天氣資料，其中降水量大於零。

以下是使用 適用於 Java 的 AWS SDK 文件 API 查詢全域次要索引的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 Table 類別的執行個體，代表您要進行作業的索引。

3. 針對您想要查詢的索引建立 Index 類別的執行個體。
4. 在 Index 物件上呼叫 query 方法。

屬性名稱 Date 是 DynamoDB 保留字。因此，您必須使用表達式屬性名稱作為 KeyConditionExpression 中的預留位置。

下列 Java 程式碼範例示範上述步驟。

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());
}
```

範例：使用 適用於 Java 的 AWS SDK 文件 API 的全域次要索引

下列 Java 程式碼範例示範如何使用全域次要索引。此範例會建立名為 Issues 的資料表，可用於軟體開發的簡單錯誤追蹤系統。分割區索引鍵為 IssueId，而排序索引鍵為 Title。此資料表上有三個全域次要索引：

- CreateDateIndex：分割區索引鍵為 CreateDate，而排序索引鍵為 IssueId。除了資料表索引鍵之外，屬性 Description 和 Status 都會投影到索引中。
- TitleIndex：分割區索引鍵為 Title，而排序索引鍵為 IssueId。除了資料表索引鍵之外，其他屬性均不會投影到索引中。

- DueDateIndex：分割區索引鍵為 DueDate，沒有排序索引鍵。所有的資料表屬性都會投影到索引。

建立 Issues 資料表後，程式會載入所含資料表示軟體錯誤報告的資料表。然後，便會使用全域次要索引查詢資料。最後，程式會刪除 Issues 資料表。

如需測試下列範例的逐步說明，請參閱 Java 程式碼範例。

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);
        }

    }
```

使用全域次要索引：.NET

您可以使用 適用於 .NET 的 AWS SDK 低階 API 來建立具有一或多個全域次要索引的 Amazon DynamoDB 資料表、描述資料表上的索引，以及使用索引執行查詢。這些操作會映射至對應的 DynamoDB 操作。如需詳細資訊，請參閱《Amazon DynamoDB API 參考》。

下列是使用 .NET 低階 API 執行資料表操作的一般步驟。

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。
2. 透過建立對應的請求物件，為操作提供必要及選用的參數。

   例如，建立 `CreateTableRequest` 物件來建立資料表，以及建立 `QueryRequest` 物件來查詢資料表或索引。
3. 執行您在前一步驟中建立之用戶端所提供的適當方法。

主題

- 建立具有全域次要索引的資料表
- 使用全域次要索引描述資料表
- 查詢全域次要索引
- 範例：使用 適用於 .NET 的 AWS SDK 低階 API 的全域次要索引

建立具有全域次要索引的資料表

您可在建立資料表的同時建立全域次要索引。若要執行這項操作，請使用 `CreateTable`，並提供一或多個全域次要索引的規格。以下 C# 程式碼範例會建立資料表來保存天氣資料的相關資訊。分割區索引鍵為 `Location`，而排序索引鍵為 `Date`。名為 `PrecipIndex` 的全域次要索引允許快速存取各個地點的降水資料。

以下是使用 .NET 低階 API 建立具有全域次要索引之資料表的步驟。

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。

2. 建立 CreateTableRequest 類別的執行個體，以提供請求資訊。

   您必須提供資料表名稱、其主索引鍵，以及佈建的輸送量數值。對於全域次要索引，您必須提供索引名稱、其佈建的輸送量設定值、索引排序索引鍵的屬性定義、索引的索引鍵結構描述以及屬性投影。

3. 以參數形式提供請求物件，以便執行 CreateTable 方法。

下列 C# 程式碼範例示範前述步驟。程式碼會建立具有全域次要索引 (PrecipIndex) 的資料表 (WeatherData)。索引分割區索引鍵是 Date，而其排序索引鍵是 Precipitation。所有的資料表屬性都會投影到索引。使用者可以查詢此索引以取得特定日期的天氣資料，可選擇依降水量排序資料。

因為 Precipitation 不是資料表的索引鍵屬性，所以其並非必要項目。不過，沒有 Precipitation 的 WeatherData 項目不會在 PrecipIndex 中顯示。

```
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);
```

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 ACTIVE。之後，您可以開始將資料項目放入資料表中。

使用全域次要索引描述資料表

若要取得資料表上全域次要索引的資訊，請使用 DescribeTable。對於每個索引，您可以存取其名稱、索引鍵結構描述和投影屬性。

以下是使用 .NET 低階 API 存取資料表的全域次要索引資訊的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 以參數形式提供請求物件，以便執行 describeTable 方法。

   建立 DescribeTableRequest 類別的執行個體，以提供請求資訊。您必須提供資料表名稱。
3.

下列 C# 程式碼範例示範前述步驟。

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);
```

```
      }
}
```

查詢全域次要索引

您可以在全域次要索引上使用 Query，與 Query 資料表的方式相同。您需要指定索引名稱、索引分割區索引鍵和排序索引鍵的查詢條件 (如存在)，以及要傳回的屬性。在本例中，索引為 PrecipIndex，其分割區索引鍵為 Date，排序索引鍵為 Precipitation。索引查詢會傳回特定日期的所有天氣資料，其中降水量大於零。

以下是使用 .NET 低階 API 查詢全域次要索引的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 建立 QueryRequest 類別的執行個體，以提供請求資訊。
3. 以參數形式提供請求物件，以便執行 query 方法。

屬性名稱 Date 是 DynamoDB 保留字。因此，您必須使用表達式屬性名稱作為 KeyConditionExpression 中的預留位置。

下列 C# 程式碼範例示範前述步驟。

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();
}
```

範例：使用 適用於 .NET 的 AWS SDK 低階 API 的全域次要索引

下列 C# 程式碼範例示範如何使用全域次要索引。此範例會建立名為 Issues 的資料表，可用於軟體開發的簡單錯誤追蹤系統。分割區索引鍵為 IssueId，而排序索引鍵為 Title。此資料表上有三個全域次要索引：

- CreateDateIndex：分割區索引鍵為 CreateDate，而排序索引鍵為 IssueId。除了資料表索引鍵之外，屬性 Description 和 Status 都會投影到索引中。
- TitleIndex：分割區索引鍵為 Title，而排序索引鍵為 IssueId。除了資料表索引鍵之外，其他屬性均不會投影到索引中。
- DueDateIndex：分割區索引鍵為 DueDate，沒有排序索引鍵。所有的資料表屬性都會投影到索引。

建立 Issues 資料表後，程式會載入所含資料表示軟體錯誤報告的資料表。然後，便會使用全域次要索引查詢資料。最後，程式會刪除 Issues 資料表。

如需測試下列範例的逐步說明，請參閱 .NET 程式碼範例。

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;
                }
```

```
            }
        }
    }
}
```

在 DynamoDB 中搭配使用 AWS CLI 與全域次要索引

您可以使用 AWS CLI 建立含有一或多個全域次要索引的 Amazon DynamoDB 資料表、描述資料表上的索引，以及使用索引執行查詢。

主題

- 建立具有全域次要索引的資料表
- 將全域次要索引新增至現有資料表
- 使用全域次要索引描述資料表
- 查詢全域次要索引

建立具有全域次要索引的資料表

您可在建立資料表的同時建立全域次要索引。若要執行這項操作，請使用 `create-table` 參數，並提供一或多個全域次要索引的規格。以下範例會建立名為 `GameScores` 的資料表，以及名為 `GameTitleIndex` 的全域次要索引。基礎資料表具有分割區索引鍵 `UserId` 和排序索引鍵 `GameTitle`，可讓您有效率地找到個別使用者在特定遊戲中的最佳分數，而 GSI 具有分割區索引鍵 `GameTitle` 和排序索引鍵 `TopScore`，可讓您快速找到特定遊戲的整體最高分數。

```
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
                }
            }
        ]"
```

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 ACTIVE。之後，您可以開始將資料項目放入資料表中。您可以使用 describe-table 來判斷資料表建立的狀態。

將全域次要索引新增至現有資料表

全域次要索引也可以在建立資料表之後新增或修改。若要執行這項操作，請使用 update-table 參數，並提供一或多個全域次要索引的規格。下列範例使用與上一個範例相同的結構描述，但假設資料表已經建立，我們稍後會新增 GSI。

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

使用全域次要索引描述資料表

若要取得資料表上全域次要索引的資訊，請使用 describe-table 參數。對於每個索引，您可以存取其名稱、索引鍵結構描述和投影屬性。

```
aws dynamodb describe-table --table-name GameScores
```

查詢全域次要索引

您可以依照與 query 資料表大致相同的方式在全域次要索引上使用 query 操作。您必須指定索引名稱、索引排序索引鍵的查詢準則，以及您要傳回的屬性。在本例中，索引是 GameTitleIndex，而索引排序索引鍵為 GameTitle。

傳回的唯一屬性是已投影到索引的屬性。您也可以修改此查詢來選擇非索引鍵屬性，但這需要相對昂貴的資料表擷取活動。如需資料表擷取的詳細資訊，請參閱 屬性投影。

```
aws dynamodb query --table-name GameScores\
    --index-name GameTitleIndex \
    --key-condition-expression "GameTitle = :v_game" \
    --expression-attribute-values '{":v_game":{"S":"Alien Adventure"} }'
```

本機次要索引

某些應用程式只需要使用基礎資料表的主索引鍵查詢資料。但是，在某些情況下，替代排序索引鍵會有所幫助。為了給應用程式提供排序索引鍵選擇，您可以在 Amazon DynamoDB 資料表上建立一或多個本機次要索引，並針對這些索引發出 Query 或 Scan 請求。

主題

- 案例：使用本機次要索引
- 屬性投影
- 建立本機次要索引
- 從本機次要索引讀取資料
- 項目寫入和本機次要索引
- 本機次要索引的佈建輸送量考量
- 本機次要索引的儲存考量
- 本機次要索引中的項目集合
- 使用本機次要索引：Java

- 使用本機次要索引：.NET
- 在 DynamoDB AWS CLI 中使用本機次要索引

案例：使用本機次要索引

例如，請考慮 Thread 資料表。此資料表適用於應用程式，例如 AWS 開發論壇。下圖顯示這些項目在資料表中的組織方式。(並未顯示所有屬性。)

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 會連續儲存具有相同分割區索引鍵值的所有項目。在本例中，給定一個特定的 ForumName，Query 操作就可以立即找到該論壇的所有主題。在具有相同分割區索引鍵值的項目群組中，項目會依排序索引鍵值排序。如果查詢中還提供了排序索引鍵 (Subject)，DynamoDB 可以縮小傳回的結果範圍，例如傳回「S3」論壇中以字母「a」開頭的 Subject 的所有主題。

某些請求可能需要更複雜的資料存取模式。例如：

- 哪些論壇主題獲得最多的觀看次數和回覆？
- 特定論壇中哪個主題的訊息數量最多？
- 在特定時段內，有多少個主題發布在特定論壇？

若要回答這些問題，Query 動作並不足夠。您還必須 Scan 整個資料表。對於包含數百萬個項目的資料表，這會消耗大量佈建的讀取輸送量，而且需要很長的時間才能完成。

不過，您可以在非索引鍵屬性上指定一或多個本機次要索引，例如 Replies 或 LastPostDateTime。

本機次要索引會針對指定分割區索引鍵值維護替代排序索引鍵。本機次要索引也包含基礎資料表中部分或全部屬性的複本。您可以指定在建立資料表時要投影到本機次要索引中的屬性。本機次要索引中的資料由與基礎資料表相同的分割區索引鍵組織，但具有不同的排序索引鍵。這可讓您在此不同維度之間有效地存取資料項目。為獲得更大的查詢或掃描靈活性，您最多可以為每個資料表建立五個本機次要索引。

假設應用程式需要尋找在最近三個月內張貼於特定論壇中的所有主題。如果沒有本機次要索引，應用程式必須 Scan 整個 Thread 資料表，並捨棄任何不在指定時間範圍內的貼文。使用本機次要索引，Query 操作可以使用 LastPostDateTime 作為排序索引鍵，並快速找到資料。

下圖顯示名為 LastPostIndex 的本機次要索引。請注意，分割區索引鍵與 Thread 資料表的分割區索引鍵相同，但排序索引鍵是 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"*
...	...	...

每個本機次要索引均須符合下列條件：

- 分割區索引鍵與其基礎資料表的分割區索引鍵相同。
- 排序索引鍵只包含一個純量屬性。
- 基礎資料表的排序索引鍵會投影到索引中，在索引中充當非索引鍵屬性。

在本例中，分割區索引鍵是 ForumName，而本機次要索引的排序索引鍵是 LastPostDateTime。此外，基礎資料表中的排序索引鍵值 (在本例中為 Subject) 會投影到索引中，但不是索引鍵的一部分。如果應用程式需要以 ForumName 和 LastPostDateTime 為基礎的清單，則可針對 LastPostIndex 發出 Query 請求。查詢結果會依 LastPostDateTime 排序，並且可以依升序或降序排列傳回。查詢也可以套用索引鍵條件，例如僅傳回在特定的時間範圍內具有 LastPostDateTime 的項目。

每個本機次要索引會自動包含來自其基礎資料表的分割區索引鍵和排序索引鍵；您可以選擇將非索引鍵屬性投影到索引中。當您查詢索引時，DynamoDB 可有效率地擷取這些投影屬性。在查詢本機次要索引時，查詢還可以檢索未投影到索引中的屬性。DynamoDB 會自動從基礎資料表擷取這些屬性，但延遲較大，且佈建的輸送量成本也較高。

對於任何本機次要索引，每個不同的分割區索引鍵值最多可存放 10 GB 的資料。此圖包含基礎資料表中的所有項目，以及索引中具有相同分割區索引鍵值的所有項目。如需詳細資訊，請參閱 本機次要索引中的項目集合。

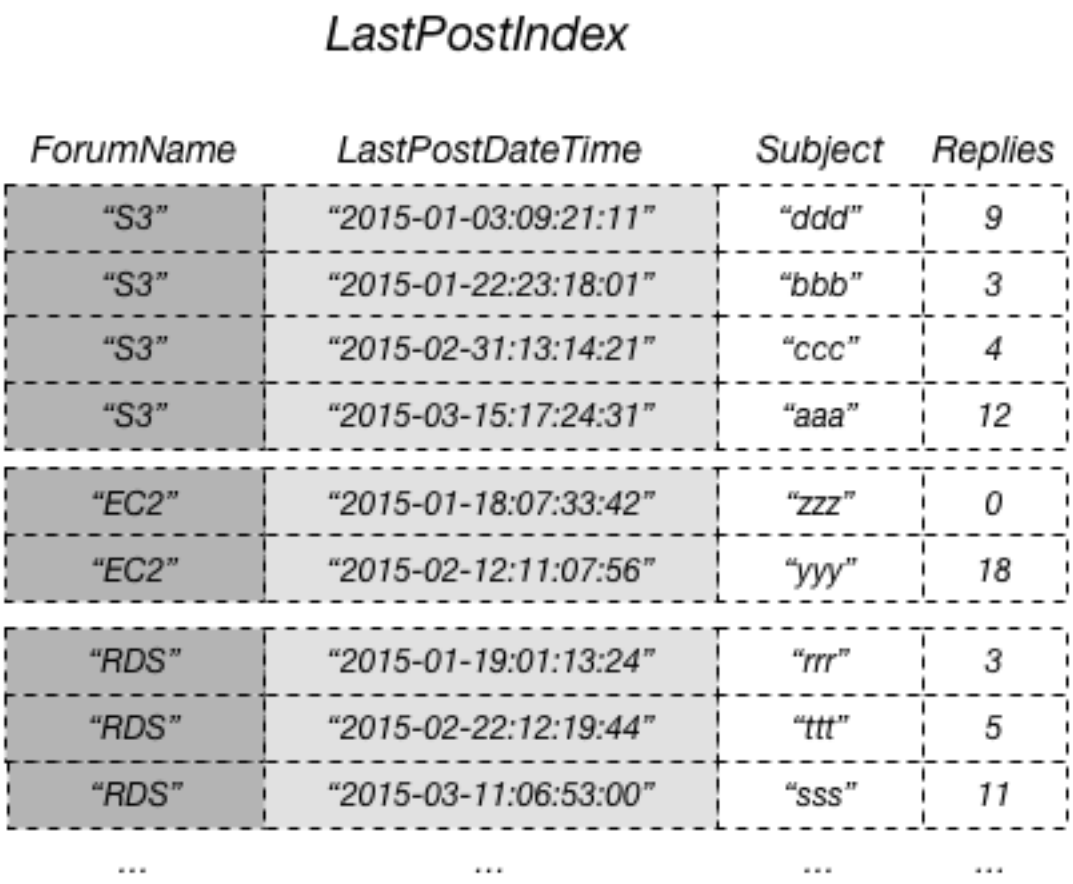
屬性投影

應用程式可以借助 LastPostIndex 將 ForumName 和 LastPostDateTime 用作查詢條件。不過，若要擷取任何其他屬性，DynamoDB 必須對 Thread 資料表執行額外的讀取操作。這些額外讀取稱為擷取，可以增加查詢所需的佈建輸送量總量。

假設您想要填入一個網頁，其中包含「S3」中所有主題的清單，以及每個主題的回覆次數 (依上次回覆日期/時間，從最近回覆開始排序)。若要填入此清單，您需要下列屬性：

- Subject
- Replies
- LastPostDateTime

查詢這些資料並避免擷取操作的最有效方法是將 Replies 屬性從資料表投影至本機次要索引，如此圖所示。



投影是指從資料表複製到次要索引的屬性集合。資料表的分割區索引鍵和排序索引鍵一律會投影到索引中；您可以投影其他屬性來支援應用程式的查詢需求。查詢索引時，Amazon DynamoDB 可以存取投影中的任何屬性，就好像這些屬性在它們自己的資料表中一樣。

在建立次要索引時，您需要指定要投影到索引中的屬性。DynamoDB 為此提供三種不同的選項：

- KEYS_ONLY：索引中的每個項目都只包含資料表分割索引鍵和排序索引鍵值，以及索引鍵值。KEYS_ONLY 選項會產生最小的可行次要索引。
- INCLUDE：除了 KEYS_ONLY 中描述的屬性外，次要索引還包含您指定的其他非索引鍵屬性。
- ALL：次要索引包含來源資料表中的所有屬性。因為索引中的所有資料表資料都會重複，所以 ALL 投影會產生最大的可行次要索引。

在上圖中，非索引鍵屬性 Replies 投影到 LastPostIndex。應用程式可以查詢 LastPostIndex，而不是完整的 Thread 資料表，以便使用 Subject、Replies 和 LastPostDateTime 來填充網頁。如果請求任何其他非索引鍵屬性，DynamoDB 需要從 Thread 資料表擷取這些屬性。

從應用程式的角度來看，從基礎資料表擷取其他屬性是自動且透明的程序，因此不需要重寫任何應用程式邏輯。不過，此類擷取會大幅降低使用本機次要索引的效能優勢。

在選擇要投影到本機次要索引的屬性時，您必須權衡佈建輸送量成本和儲存成本：

- 若您只需要存取少量的屬性，並且希望盡可能的降低延遲，建議您考慮只將那些屬性投影到本機次要索引。索引愈小，存放成本就愈低，您的寫入成本也愈低。如果您偶爾需要擷取屬性，佈建輸送量的成本可能會超過儲存這些屬性的長期成本。
- 如果應用程式會頻繁存取某些非索引鍵屬性，您應該考慮將這些屬性投影到本機次要索引。本機次要索引的額外儲存成本會抵銷頻繁執行資料表掃描的成本。
- 如果需要頻繁存取大多數的非索引鍵屬性，您可以將這些屬性 (或整個基礎資料表) 投影到本機次要索引。這可為您提供最大的靈活性和最低的佈建輸送量耗用量，因為不需要擷取。但是，如果投影所有屬性，您的儲存成本將會增加，甚至翻倍。
- 如果您的應用程式不需頻繁查詢資料表，但必須對資料表中的資料執行許多寫入或更新，請考慮投影 KEYS_ONLY。本機次要索引的大小將會最小，但仍能在需要的時候進行查詢活動。

建立本機次要索引

若要在資料表上建立具有一或多個本機次要索引，請使用 `CreateTable` 操作的 `LocalSecondaryIndexes` 參數。建立資料表時，會建立資料表上的本機次要索引。當您使用刪除資料表時，也會刪除該資料表上的任何本機次要索引。

您必須指定一個非索引鍵屬性作為本機次要索引的排序索引鍵。您選擇的屬性必須是純量 `String`、`Number` 或 `Binary`。不允許其他純量類型、文件類型和集合類型。如需資料類型的完整清單，請參閱 資料類型。

> **Important**
>
> 對於具有本機次要索引的資料表，每個分割區索引鍵值都有 10 GB 的大小限制。具有本機次要索引的資料表可以存放任何數量的項目，只要任何一個分割區索引鍵值的總大小不超過 10 GB 即可。如需詳細資訊，請參閱 項目集合大小限制。

您可以將任何資料類型的屬性投影到本機次要索引。這包括純量、文件和集合。如需資料類型的完整清單，請參閱 資料類型。

從本機次要索引讀取資料

您可以使用 Query 和 Scan 操作從本機次要索引擷取項目。GetItem 和 BatchGetItem 操作不能用於本機次要索引。

查詢本機次要索引

在 DynamoDB 資料表中，每個項目的組合分割區索引鍵值和排序索引鍵值必須是唯一的值。不過，在本機次要索引中，排序索引鍵值對於指定的分割區索引鍵值不一定是唯一的值。如果本機次要索引中有多個項目具有相同的排序索引鍵值，Query 操作會傳回具有相同分割區索引鍵值的所有項目。在回應中，符合的項目不會以任何特定的順序傳回。

您可以使用最終一致或高度一致性讀取來查詢本機次要索引。若要指定您想要的一致性類型，請使用 Query 操作的 ConsistentRead 參數。來自本機次要索引的高度一致性讀取一律會傳回最新的更新值。如果查詢需要從基礎資料表中取得其他屬性，這些屬性將與索引保持一致。

Example

請考慮從 Query (請求來自特定論壇主題的資料) 傳回的下列資料。

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

在此查詢中：

- DynamoDB 存取 LastPostIndex，使用 ForumName 分割區索引鍵尋找「EC2」的索引項目。所有具有此索引鍵的索引項目都會相鄰存放，以利快速擷取。
- 在此論壇中，DynamoDB 使用索引來尋找符合指定 LastPostDateTime 條件的索引鍵。
- 由於 Replies 屬性投影到索引中時，DynamoDB 可以檢索此屬性，而不會耗用任何額外的佈建輸送量。

- 此 Tags 屬性不會投影到索引中，因此 DynamoDB 必須存取 Thread 資料表並擷取此屬性。
- 結果會傳回，並依 LastPostDateTime 排序。索引項目依分割區索引鍵值排序，再依排序索引鍵值排序，然後由 Query 依儲存的順序將其傳回。(您可以使用 ScanIndexForward 參數，以遞減順序傳回結果。)

由於 Tags 屬性未投影到本機次要索引中，DynamoDB 必須消耗額外的讀取容量單位，才能從基礎資料表擷取此屬性。如果需要經常執行此查詢，您應將 Tags 投影至 LastPostIndex，避免從基礎資料表擷取。不過，如果您僅偶爾需要存取 Tags，則將 Tags 投影至索引中的額外儲存成本便可能不值得。

掃描本機次要索引

您可以使用 Scan 從本機次要索引檢索所有資料。您必須在請求中提供基礎資料表名稱及索引名稱。使用 Scan，DynamoDB 會讀取索引中的所有資料，並將其傳回應用程式。您也可以請求只傳回一部分的資料，並捨棄其他剩餘的資料。若要執行此作業，請使用 Scan API 的 FilterExpression 參數。如需詳細資訊，請參閱 掃描的篩選條件表達式。

項目寫入和本機次要索引

DynamoDB 會自動讓所有本機次要索引與各自的基礎資料表同步。應用程式永遠不會直接寫入索引。然而，了解 DynamoDB 維持這些索引之方式的含義很重要。

在建立本機次要索引時，您可以指定一個屬性作為索引的排序索引鍵。您也可以為該屬性指定資料類型。這表示每次您將項目寫入基礎資料表時，如果項目定義了索引鍵屬性，則其類型必須與索引鍵結構描述的資料類型相符。在 LastPostIndex 案例中，索引的 LastPostDateTime 排序索引鍵是定義為 String 資料類型。如果您嘗試在 Thread 資料表中新增項目，並為 LastPostDateTime (如 Number) 指定不同的資料類型，DynamoDB 會因資料類型不符而傳回 ValidationException。

基礎資料表中的項目與本機次要索引中的項目之間沒有一對一的關係要求。事實上，這種行為對許多應用程式來說可能是有利的。

相較於索引較少的資料表，本機次要索引較多的資料表會有較高的寫入活動成本。如需詳細資訊，請參閱 本機次要索引的佈建輸送量考量。

> **Important**
>
> 對於具有本機次要索引的資料表，每個分割區索引鍵值都有 10 GB 的大小限制。具有本機次要索引的資料表可以存放任何數量的項目，只要任何一個分割區索引鍵值的總大小不超過 10 GB 即可。如需詳細資訊，請參閱 項目集合大小限制。

本機次要索引的佈建輸送量考量

在 DynamoDB 中建立資料表時，您可以為資料表的預期工作負載佈建讀取和寫入容量單位。該工作負載包括在資料表本機次要索引上的讀取和寫入活動。

若要檢視佈建的輸送容量的目前費率，請參閱 Amazon DynamoDB 定價。

讀取容量單位

在查詢本機次要索引時，消耗的讀取容量單位數目取決於資料的存取方式。

與資料表查詢一樣，索引查詢可以使用最終一致性或高度一致性讀取，取決於 `ConsistentRead` 的值。一個高度一致性讀取會消耗一個讀取容量單位；最終一致讀取則只會消耗一半的讀取容量單位。因此，透過選擇最終一致讀取，您可以減少讀取容量單位的費用。

對於僅請求索引鍵和投影屬性的索引查詢，DynamoDB 會使用為資料表查詢計算佈建讀取活動相同的方式，計算佈建讀取活動。唯一的差別在於計算方式是以索引項目的大小為基礎，而非基礎資料表中項目的大小。讀取容量單位數為所有傳回項目中，所有投影屬性大小的總和；結果會四捨五入至下一個 4 KB 界限。如需 DynamoDB 如何計算佈建輸送用量的詳細資訊，請參閱 DynamoDB 佈建容量模式。

對於讀取未投影到本機次要索引的屬性的索引查詢，除了從索引讀取投影的屬性之外，DynamoDB 還需要從基礎資料表擷取這些屬性。當您在 `Query` 操作的 `Select` 或 `ProjectionExpression` 參數中包含任何非投影屬性，即會發生這些擷取。擷取會在查詢回應中造成額外的延遲，也會產生較高的佈建輸送量成本：除了從先前描述的本機次要索引讀取之外，您還需要針對每個擷取的基礎資料表項目支付讀取容量單位費用。此費用是用於讀取資料表中的每個項目，而不僅僅是請求的屬性。

`Query` 操作傳回的結果大小上限是 1 MB。這包含所有傳回項目之所有屬性名稱與值的大小。不過，如果針對本機次要索引進行查詢導致 DynamoDB 從基礎資料表擷取項目屬性，則結果中資料的大小上限可能會降低。在這種情況下，結果大小是以下各項的總和：

- 索引中相符項目的大小，且項目大小會四捨五入至下一個 4 KB。
- 基礎資料表中每個相符項目的大小，且每個項目的大小會四捨五入至下一個 4 KB。

若使用此公式，查詢操作傳回的結果的大小上限仍為 1 MB。

例如，假設有一個資料表，其中每個項目的大小均為 300 個位元組。該資料表上有本機次要索引，但每個項目只有 200 個位元組投影到索引中。現在假設您 `Query` 此索引，查詢將需為每個項目擷取資料表，且查詢會傳回 4 個項目。DynamoDB 會計算以下總數：

- 索引中相符項目的大小：200 個位元組 × 4 個項目 = 800 個位元組；接著會四捨五入至 4 KB。

- 基礎資料表中每個相符項目的大小：(300 個位元組，四捨五入至 4 KB) × 4 個項目 = 16 KB。

因此，結果中的資料總大小為 20 KB。

寫入容量單位

當新增、更新或刪除資料表中的項目時，更新本機次要索引會耗用資料表的佈建寫入容量單位。寫入的總佈建輸送量成本為寫入資料表使用的寫入容量單位，加上更新本機次要索引使用的寫入容量單位。

將項目寫入本機次要索引的成本取決於幾個因素：

- 若您將項目寫入定義索引屬性的資料表，或是您將現有的項目更新為先前未定義的索引屬性，則將該項目寫入索引需要進行一次寫入操作。
- 若資料表的更新會變更索引鍵屬性的值 (從 A 到 B)，則需要兩次寫入：一次是從索引刪除先前的項目，第二次則是將新的項目寫入索引。
- 若項目存在於索引中，但寫入資料表致使索引屬性遭到刪除，則需要進行一次寫入，從索引刪除舊項目的投影。
- 若項目在更新之前或之後並不存在於索引中，則該索引將不會有任何額外的寫入成本。

所有這些因素都假設索引中每個項目的大小都小於或等於 1 KB 項目大小 (用於計算寫入容量單位)。較大的索引項目需要額外的寫入容量單位。您可以考慮查詢需要傳回的屬性，並只將那些屬性投影到索引中，從而將寫入成本降至最小。

本機次要索引的儲存考量

當應用程式將項目寫入資料表時，DynamoDB 會自動將正確的部分屬性複製到應顯示這些屬性的任何本機次要索引中。 AWS 您的帳戶需要支付基本資料表中項目的儲存費用，以及該資料表上任何本機次要索引中屬性的儲存費用。

索引項目使用的空間數為下列項目的總和：

- 基礎資料表主索引鍵 (分割區索引鍵和排序索引鍵) 的大小 (位元組)
- 索引鍵屬性的大小 (位元組)
- 投影屬性 (若有的話) 的大小 (位元組)
- 每個索引項目 100 位元組的額外負荷

若要估算本機次要索引的儲存需求，您可以估算索引中項目的平均大小，再乘以索引中的項目數量。

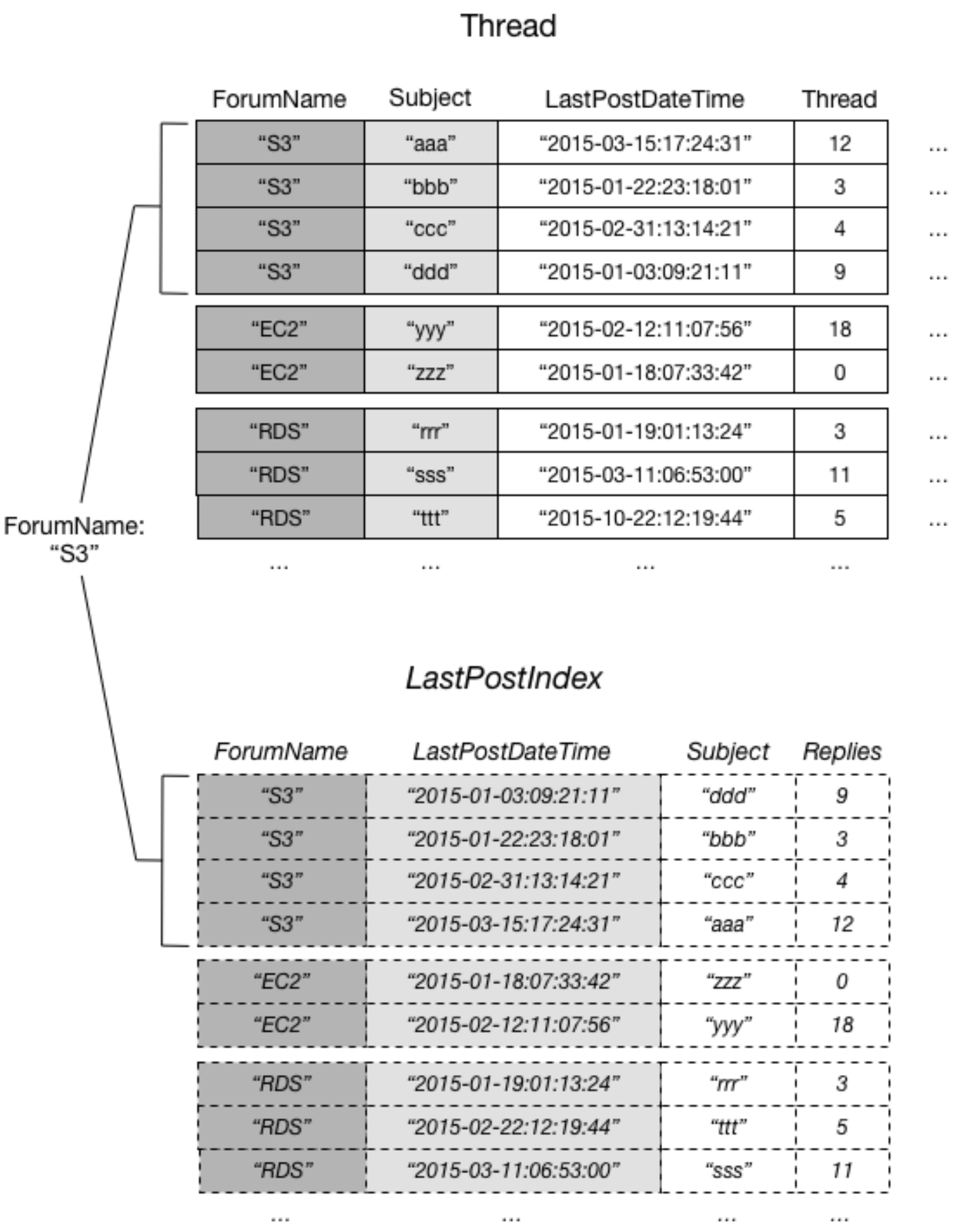
如果資料表包含特定屬性未經定義的項目，但該屬性卻已定義為索引的排序索引鍵時，DynamoDB 不會將該項目的任何資料寫入索引。

本機次要索引中的項目集合

> **Note**
>
> 此節僅適用於具有本機次要索引的資料表。

在 DynamoDB 中，項目集合是資料表及其所有本機次要索引中具有相同分割區索引鍵值的任何項目群組。在本節所使用的範例中，Thread 資料表的分割區索引鍵為 ForumName，LastPostIndex 的分割區索引鍵也為 ForumName。所有具有相同 ForumName 的資料表和索引項目均屬於相同的項目集合。例如，在 Thread 資料表與 LastPostIndex 本機次要索引中，論壇 EC2 有一個項目集合，論壇 RDS 則有一個不同的項目集合。

下圖顯示論壇 S3 的項目集合。


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

在此圖表中，項目集合由 Thread 和 LastPostIndex 中的所有項目組成，其中 ForumName 分割區索引鍵值為「S3」。如果資料表上有其他本機次要索引，則這些索引中 ForumName 等於「S3」的任何項目也將成為項目集合的一部分。

您可以在 DynamoDB 中使用下列任何操作來傳回項目集合的相關資訊：

- BatchWriteItem
- DeleteItem
- PutItem
- UpdateItem
- TransactWriteItems

這些操作中的每一個都支援 ReturnItemCollectionMetrics 參數。在將此參數設定為 SIZE，您可以檢視索引中每個項目集合大小的相關資訊。

Example

以下是 Thread 資料表上 UpdateItem 操作的輸出範例，其中 ReturnItemCollectionMetrics 設定為 SIZE。更新的項目的 ForumName 值為「EC2」，因此輸出包含有關該項目集合的資訊。

```
{
    ItemCollectionMetrics: {
        ItemCollectionKey: {
            ForumName: "EC2"
        },
        SizeEstimateRangeGB: [0.0, 1.0]
    }
}
```

SizeEstimateRangeGB 物件會顯示此項目集合的大小介於 0 到 1 GB 之間。DynamoDB 會定期更新此大小估計值，因此下次修改項目時，數字可能會有所不同。

項目集合大小限制

包含一或多個本機次要索引之資料表，其任何項目集合的大小上限為 10GB。這不適用於沒有本機次要索引之資料表的項目集合，也不適用於全域次要索引中的項目集合。僅具有一或多個本機次要索引的資料表才會受到影響。

如果項目集合超過 10 GB 的限制，DynamoDB 可能會傳回 ItemCollectionSizeLimitExceededException，而且您可能無法將更多項目新增至項目集合，

或增加項目集合中項目的大小。(仍允許會縮小項目集合大小的讀取和寫入操作。) 您仍可將項目新增至其他項目集合。

若要減少項目集合的大小，您可以執行下列操作之一：

- 刪除有問題的分割區索引鍵值的任何不必要項目。在從基礎資料表中刪除這些項目時，DynamoDB 也會移除具有相同分割區索引鍵值的任何索引項目。
- 透過移除屬性或減少屬性大小來更新項目。如果這些屬性投影到任何本機次要索引中，DynamoDB 也會減少對應索引項目的大小。
- 使用相同的分割區索引鍵和排序索引鍵來建立新資料表，然後將項目從舊資料表移至新資料表。如果資料表具有不常存取的歷史資料，這可能是一個很好的方法。您也可以考慮將此歷史資料封存至 Amazon Simple Storage Service (Amazon S3)。

當項目集合的總大小降到 10 GB 以下時，您可以再次新增具有相同分割區索引鍵值的項目。

作為最佳實務，我們建議您檢測應用程式以監控項目集合的大小。方法之一是在使用 `BatchWriteItem`、`DeleteItem`、`PutItem` 或 `UpdateItem` 時將 `ReturnItemCollectionMetrics` 參數設定為 `SIZE`。您的應用程式應該檢查輸出中的 `ReturnItemCollectionMetrics` 物件，並在項目集合超過使用者定義的限制 (例如 8 GB) 時記錄錯誤訊息。設定小於 10 GB 的限制會提供一個預警系統，讓您及時掌握項目集合正接近限制，以便採取相關措施。

項目集合和分割區

在包含一或多個本機次要索引的資料表中，每個項目集合會儲存在一個分割區中。此項目集合的總大小僅限於該分割區的容量：10GB。應用程式中，若資料模型包含大小無上限的項目集合，或您可能合理預期某些項目集合日後會超過 10GB，您應考慮改用全域次要索引。

您應將應用程式設計為讓資料表資料均勻分佈在不同的分割區索引鍵值之間。對於具有本機次要索引的資料表，您的應用程式不應在單一分割區上的單一項目集合內建立讀取和寫入活動的「熱點」。

使用本機次要索引：Java

您可以使用 適用於 Java 的 AWS SDK 文件 API 建立具有一或多個本機次要索引的 Amazon DynamoDB 資料表、描述資料表上的索引，並使用索引執行查詢。

以下是使用 適用於 Java 的 AWS SDK 文件 API 進行資料表操作的常見步驟。

1. 建立 `DynamoDB` 類別的執行個體。
2. 透過建立對應的請求物件，為操作提供必要及選用的參數。

3. 呼叫您在前一步驟中建立之用戶端所提供的適當方法。

主題

- 使用本機次要索引建立資料表
- 使用本機次要索引描述資料表
- 查詢本機次要索引
- 範例：使用 Java Document API 的本機次要索引

使用本機次要索引建立資料表

您必須同時建立資料表和本機次要索引。若要執行這項操作，請使用 `createTable` 方法，並提供一或多個本機次要索引的規格。以下 Java 程式碼範例會建立資料表來保存音樂收藏中歌曲的相關資訊。分割區索引鍵為 `Artist`，而排序索引鍵為 `SongTitle`。次要索引 `AlbumTitleIndex` 有助於依照專輯標題查詢。

以下是使用 DynamoDB Document API 建立具有本機次要索引的資料表的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 `CreateTableRequest` 類別的執行個體，以提供請求資訊。

   您必須提供資料表名稱、其主索引鍵，以及佈建的輸送量數值。對於本機次要索引，您必須提供索引名稱、索引排序索引鍵的名稱和資料類型、索引的索引鍵結構描述以及屬性投影。
3. 以參數形式提供請求物件，以便呼叫 `createTable` 方法。

下列 Java 程式碼範例示範上述步驟。程式碼會建立一個資料表 (`Music`) 與次要索引 `AlbumTitle` 屬性。資料表分割區索引鍵和排序索引鍵以及索引排序索引鍵，是唯一投影到索引的屬性。

```
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());
```

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 ACTIVE。之後，您可以開始將資料項目放入資料表中。

使用本機次要索引描述資料表

若要取得資料表上本機次要索引的資訊，請使用 describeTable 方法。對於每個索引，您可以存取其名稱、索引鍵結構描述和投影屬性。

以下是使用 適用於 Java 的 AWS SDK Document API 存取資料表的本機次要索引資訊的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 Table 類別的執行個體。您必須提供資料表名稱。
3. 在 Table 物件上呼叫 describeTable 方法。

下列 Java 程式碼範例示範上述步驟。

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());
    }
}
```

查詢本機次要索引

您可以依照與 Query 資料表大致相同的方式在本機次要索引上使用 Query 操作。您必須指定索引名稱、索引排序索引鍵的查詢準則，以及您要傳回的屬性。在本例中，索引是 AlbumTitleIndex，而索引排序索引鍵為 AlbumTitle。

傳回的唯一屬性是已投影到索引的屬性。您也可以修改此查詢來選擇非索引鍵屬性，但這需要相對昂貴的資料表擷取活動。如需資料表擷取的詳細資訊，請參閱 屬性投影。

以下是使用 適用於 Java 的 AWS SDK 文件 API 查詢本機次要索引的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 Table 類別的執行個體。您必須提供資料表名稱。
3. 建立 Index 類別的執行個體。您必須提供索引名稱。
4. 呼叫 Index 類別的 query 方法。

下列 Java 程式碼範例示範上述步驟。

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());
}
```

本機次要索引上的一致讀取

與僅支援最終一致讀取的全域次要索引不同，本機次要索引支援最終一致和強烈一致讀取。來自本機次要索引的高度一致性讀取一律會傳回最新的更新值。如果查詢需要從基礎資料表擷取其他屬性，則這些擷取的屬性也與索引一致。

根據預設，`Query`會使用最終一致讀取。若要請求強式一致讀取，請在 `true`中將 `ConsistentRead`設定為 `QuerySpec`。下列範例查詢`AlbumTitleIndex`使用強式一致讀取：

Example

```
QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("Artist = :v_artist and AlbumTitle = :v_title")
    .withValueMap(new ValueMap()
        .withString(":v_artist", "Acme Band")
        .withString(":v_title", "Songs About Life"))
    .withConsistentRead(true);
```

> **Note**
>
> 強式一致讀取每傳回 4 KB 的資料會耗用一個讀取容量單位（四捨五入），而最終一致讀取則會耗用一半。例如，傳回 9 KB 資料的強式一致讀取耗用 3 個讀取容量單位 (9 KB/4 KB = 2.25，四捨五入至 3)，而使用最終一致讀取的相同查詢耗用 1.5 個讀取容量單位。如果您的應

用程式可以容忍讀取可能稍微過時的資料，請使用最終一致讀取來減少讀取容量用量。如需詳細資訊，請參閱讀取容量單位。

範例：使用 Java Document API 的本機次要索引

下列 Java 程式碼範例示範如何使用 Amazon DynamoDB 中的本機次要索引。例如，您可以建立名為 CustomerOrders 的資料表，其中分割區索引鍵為 CustomerId，排序索引鍵為 OrderId。此資料表上有兩個本機次要索引：

- OrderCreationDateIndex：排序索引鍵為 OrderCreationDate，並且以下屬性會投影到索引：
  - ProductCategory
  - ProductName
  - OrderStatus
  - ShipmentTrackingId
- IsOpenIndex：排序索引鍵為 IsOpen，並且所有的資料表屬性都會投影到索引。

建立 CustomerOrders 資料表後，程式會載入所含資料表示客戶訂單的資料表。然後，其便會使用本機次要索引查詢資料。最後，程式會刪除 CustomerOrders 資料表。

如需測試下列範例的逐步說明，請參閱 Java 程式碼範例。

Example

```
package com.example.dynamodb;

import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

import java.util.HashMap;
import java.util.Map;

public class DocumentAPILocalSecondaryIndexExample {

    static DynamoDbClient client = DynamoDbClient.create();
```

```
    public static String tableName = "CustomerOrders";

    public static void main(String[] args) {
        createTable();
        loadData();
        query(null);
        query("IsOpenIndex");
        query("OrderCreationDateIndex");
        deleteTable(tableName);
    }

    public static void createTable() {
        CreateTableRequest request = CreateTableRequest.builder()
            .tableName(tableName)
            .provisionedThroughput(ProvisionedThroughput.builder()
                .readCapacityUnits(1L)
                .writeCapacityUnits(1L)
                .build())
            .attributeDefinitions(

AttributeDefinition.builder().attributeName("CustomerId").attributeType(ScalarAttributeType.S)

AttributeDefinition.builder().attributeName("OrderId").attributeType(ScalarAttributeType.N).bu

AttributeDefinition.builder().attributeName("OrderCreationDate").attributeType(ScalarAttribute

AttributeDefinition.builder().attributeName("IsOpen").attributeType(ScalarAttributeType.N).bui
            .keySchema(

KeySchemaElement.builder().attributeName("CustomerId").keyType(KeyType.HASH).build(),

KeySchemaElement.builder().attributeName("OrderId").keyType(KeyType.RANGE).build())
            .localSecondaryIndexes(
                LocalSecondaryIndex.builder()
                    .indexName("OrderCreationDateIndex")
                    .keySchema(

KeySchemaElement.builder().attributeName("CustomerId").keyType(KeyType.HASH).build(),

KeySchemaElement.builder().attributeName("OrderCreationDate").keyType(KeyType.RANGE).build())
                    .projection(Projection.builder()
                        .projectionType(ProjectionType.INCLUDE)
                        .nonKeyAttributes("ProductCategory", "ProductName")
                        .build())
```

```
                    .build(),
                LocalSecondaryIndex.builder()
                    .indexName("IsOpenIndex")
                    .keySchema(

 KeySchemaElement.builder().attributeName("CustomerId").keyType(KeyType.HASH).build(),

 KeySchemaElement.builder().attributeName("IsOpen").keyType(KeyType.RANGE).build())
                    .projection(Projection.builder()
                        .projectionType(ProjectionType.ALL)
                        .build())
                    .build())
            .build();

        System.out.println("Creating table " + tableName + "...");
        client.createTable(request);

        try (DynamoDbWaiter waiter = client.waiter()) {
            WaiterResponse<DescribeTableResponse> response =
 waiter.waitUntilTableExists(r -> r.tableName(tableName));
            response.matched().response().ifPresent(System.out::println);
        }
    }

    public static void query(String indexName) {

 System.out.println("\n***********************************************************\n");
        System.out.println("Querying table " + tableName + "...");

        if ("IsOpenIndex".equals(indexName)) {
            System.out.println("\nUsing index: '" + indexName + "': Bob's orders that
 are open.");
            System.out.println("Only a user-specified list of attributes are returned
\n");

            Map<String, AttributeValue> values = new HashMap<>();
            values.put(":v_custid",
 AttributeValue.builder().s("bob@example.com").build());
            values.put(":v_isopen", AttributeValue.builder().n("1").build());

            QueryRequest request = QueryRequest.builder()
                .tableName(tableName)
                .indexName(indexName)
```

```
                .keyConditionExpression("CustomerId = :v_custid and IsOpen
= :v_isopen")
                .expressionAttributeValues(values)
                .projectionExpression("OrderCreationDate, ProductCategory, ProductName,
OrderStatus")
                .build();

            System.out.println("Query: printing results...");
            client.query(request).items().forEach(System.out::println);

        } else if ("OrderCreationDateIndex".equals(indexName)) {
            System.out.println("\nUsing index: '" + indexName + "': Bob's orders that
were placed after 01/31/2015.");
            System.out.println("Only the projected attributes are returned\n");

            Map<String, AttributeValue> values = new HashMap<>();
            values.put(":v_custid",
AttributeValue.builder().s("bob@example.com").build());
            values.put(":v_orddate", AttributeValue.builder().n("20150131").build());

            QueryRequest request = QueryRequest.builder()
                .tableName(tableName)
                .indexName(indexName)
                .keyConditionExpression("CustomerId = :v_custid and OrderCreationDate
>= :v_orddate")
                .expressionAttributeValues(values)
                .select(Select.ALL_PROJECTED_ATTRIBUTES)
                .build();

            System.out.println("Query: printing results...");
            client.query(request).items().forEach(System.out::println);

        } else {
            System.out.println("\nNo index: All of Bob's orders, by OrderId:\n");

            Map<String, AttributeValue> values = new HashMap<>();
            values.put(":v_custid",
AttributeValue.builder().s("bob@example.com").build());

            QueryRequest request = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression("CustomerId = :v_custid")
                .expressionAttributeValues(values)
                .build();
```

```
            System.out.println("Query: printing results...");
            client.query(request).items().forEach(System.out::println);
        }
    }

    public static void deleteTable(String tableName) {
        System.out.println("Deleting table " + tableName + "...");
        client.deleteTable(DeleteTableRequest.builder().tableName(tableName).build());

        try (DynamoDbWaiter waiter = client.waiter()) {
            waiter.waitUntilTableNotExists(r -> r.tableName(tableName));
        }
    }

    public static void loadData() {
        System.out.println("Loading data into table " + tableName + "...");

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("alice@example.com").build(),
            "OrderId", AttributeValue.builder().n("1").build(),
            "IsOpen", AttributeValue.builder().n("1").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150101").build(),
            "ProductCategory", AttributeValue.builder().s("Book").build(),
            "ProductName", AttributeValue.builder().s("The Great Outdoors").build(),
            "OrderStatus", AttributeValue.builder().s("PACKING ITEMS").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("alice@example.com").build(),
            "OrderId", AttributeValue.builder().n("2").build(),
            "IsOpen", AttributeValue.builder().n("1").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150221").build(),
            "ProductCategory", AttributeValue.builder().s("Bike").build(),
            "ProductName", AttributeValue.builder().s("Super Mountain").build(),
            "OrderStatus", AttributeValue.builder().s("ORDER RECEIVED").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("alice@example.com").build(),
            "OrderId", AttributeValue.builder().n("3").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150304").build(),
            "ProductCategory", AttributeValue.builder().s("Music").build(),
            "ProductName", AttributeValue.builder().s("A Quiet Interlude").build(),
            "OrderStatus", AttributeValue.builder().s("IN TRANSIT").build(),
            "ShipmentTrackingId", AttributeValue.builder().s("176493").build()));
```

```
        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("1").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150111").build(),
            "ProductCategory", AttributeValue.builder().s("Movie").build(),
            "ProductName", AttributeValue.builder().s("Calm Before The Storm").build(),
            "OrderStatus", AttributeValue.builder().s("SHIPPING DELAY").build(),
            "ShipmentTrackingId", AttributeValue.builder().s("859323").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("2").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150124").build(),
            "ProductCategory", AttributeValue.builder().s("Music").build(),
            "ProductName", AttributeValue.builder().s("E-Z Listening").build(),
            "OrderStatus", AttributeValue.builder().s("DELIVERED").build(),
            "ShipmentTrackingId", AttributeValue.builder().s("756943").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("3").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150221").build(),
            "ProductCategory", AttributeValue.builder().s("Music").build(),
            "ProductName", AttributeValue.builder().s("Symphony 9").build(),
            "OrderStatus", AttributeValue.builder().s("DELIVERED").build(),
            "ShipmentTrackingId", AttributeValue.builder().s("645193").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("4").build(),
            "IsOpen", AttributeValue.builder().n("1").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150222").build(),
            "ProductCategory", AttributeValue.builder().s("Hardware").build(),
            "ProductName", AttributeValue.builder().s("Extra Heavy Hammer").build(),
            "OrderStatus", AttributeValue.builder().s("PACKING ITEMS").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("5").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150309").build(),
            "ProductCategory", AttributeValue.builder().s("Book").build(),
            "ProductName", AttributeValue.builder().s("How To Cook").build(),
            "OrderStatus", AttributeValue.builder().s("IN TRANSIT").build(),
```

```
            "ShipmentTrackingId", AttributeValue.builder().s("440185").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("6").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150318").build(),
            "ProductCategory", AttributeValue.builder().s("Luggage").build(),
            "ProductName", AttributeValue.builder().s("Really Big Suitcase").build(),
            "OrderStatus", AttributeValue.builder().s("DELIVERED").build(),
            "ShipmentTrackingId", AttributeValue.builder().s("893927").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("7").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150324").build(),
            "ProductCategory", AttributeValue.builder().s("Golf").build(),
            "ProductName", AttributeValue.builder().s("PGA Pro II").build(),
            "OrderStatus", AttributeValue.builder().s("OUT FOR DELIVERY").build(),
            "ShipmentTrackingId", AttributeValue.builder().s("383283").build()));
    }

    private static void putItem(Map<String, AttributeValue> item) {

 client.putItem(PutItemRequest.builder().tableName(tableName).item(item).build());
    }
}
```

使用本機次要索引：.NET

主題

- 使用本機次要索引建立資料表
- 使用本機次要索引描述資料表
- 查詢本機次要索引
- 範例：使用 適用於 .NET 的 AWS SDK 低階 API 的本機次要索引

您可以使用 適用於 .NET 的 AWS SDK 低階 API 來建立具有一或多個本機次要索引的 Amazon DynamoDB 資料表、描述資料表上的索引，並使用索引執行查詢。這些操作會映射至對應的低階 DynamoDB API 動作。如需詳細資訊，請參閱 .NET 程式碼範例。

下列是使用 .NET 低階 API 執行資料表操作的一般步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 透過建立對應的請求物件，為操作提供必要及選用的參數。

   例如，建立 CreateTableRequest 物件來建立資料表，以及建立 QueryRequest 物件來查詢資料表或索引。
3. 執行您在前一步驟中建立之用戶端所提供的適當方法。

使用本機次要索引建立資料表

您必須同時建立資料表和本機次要索引。若要執行這項操作，請使用 CreateTable，並提供一或多個本機次要索引的規格。以下 C# 程式碼範例會建立資料表來保存音樂收藏中歌曲的相關資訊。分割區索引鍵為 Artist，而排序索引鍵為 SongTitle。次要索引 AlbumTitleIndex 有助於依照專輯標題查詢。

以下是使用 .NET 低階 API 建立具有本機次要索引之資料表的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 建立 CreateTableRequest 類別的執行個體，以提供請求資訊。

   您必須提供資料表名稱、其主索引鍵，以及佈建的輸送量數值。對於本機次要索引，您必須提供索引名稱、索引排序索引鍵的名稱和資料類型、索引的索引鍵結構描述以及屬性投影。
3. 以參數形式提供請求物件，以便執行 CreateTable 方法。

下列 C# 程式碼範例示範前述步驟。程式碼會建立一個資料表 (Music) 與次要索引 AlbumTitle 屬性。資料表分割區索引鍵和排序索引鍵以及索引排序索引鍵，是唯一投影到索引的屬性。

```
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);
```

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 ACTIVE。之後，您可以開始將資料項目放入資料表中。

使用本機次要索引描述資料表

若要取得資料表上本機次要索引的資訊，請使用 DescribeTable API。對於每個索引，您可以存取其名稱、索引鍵結構描述和投影屬性。

以下是使用 .NET 低階 API 存取資料表的本機次要索引資訊的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 建立 DescribeTableRequest 類別的執行個體，以提供請求資訊。您必須提供資料表名稱。
3. 以參數形式提供請求物件，以便執行 describeTable 方法。
4.

下列 C# 程式碼範例示範前述步驟。

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);
        }

    }
}
```

查詢本機次要索引

您可以依照與 Query 資料表大致相同的方式在本機次要索引上使用 Query。您必須指定索引名稱、索引排序索引鍵的查詢準則，以及您要傳回的屬性。在本例中，索引是 AlbumTitleIndex，而索引排序索引鍵為 AlbumTitle。

傳回的唯一屬性是已投影到索引的屬性。您也可以修改此查詢來選擇非索引鍵屬性，但這需要相對昂貴的資料表擷取活動。如需資料表擷取的詳細資訊，請參閱 [屬性投影]

以下是使用 .NET 低階 API 查詢本機次要索引的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 建立 QueryRequest 類別的執行個體，以提供請求資訊。
3. 以參數形式提供請求物件，以便執行 query 方法。

下列 C# 程式碼範例示範前述步驟。

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();
}
```

範例：使用 適用於 .NET 的 AWS SDK 低階 API 的本機次要索引

下列 C# 程式碼範例示範如何在 Amazon DynamoDB 中使用本機次要索引。例如，您可以建立名為 CustomerOrders 的資料表，其中分割區索引鍵為 CustomerId，排序索引鍵為 OrderId。此資料表上有兩個本機次要索引：

- OrderCreationDateIndex：排序索引鍵為 OrderCreationDate，並且以下屬性會投影到索引：
  - ProductCategory
  - ProductName
  - OrderStatus
  - ShipmentTrackingId
- IsOpenIndex：排序索引鍵為 IsOpen，並且所有的資料表屬性都會投影到索引。

建立 CustomerOrders 資料表後，程式會載入所含資料表示客戶訂單的資料表。然後，其便會使用本機次要索引查詢資料。最後，程式會刪除 CustomerOrders 資料表。

如需測試下列範例的逐步說明，請參閱 .NET 程式碼範例。

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;
                }
            }
        }
    }
}
```

在 DynamoDB AWS CLI 中使用本機次要索引

您可以使用 AWS CLI 建立含有一或多個本機次要索引的 Amazon DynamoDB 資料表、描述資料表上的索引，以及使用索引執行查詢。

主題

- 使用本機次要索引建立資料表
- 使用本機次要索引描述資料表
- 查詢本機次要索引

使用本機次要索引建立資料表

您必須同時建立資料表和本機次要索引。若要執行這項操作，請使用 `create-table` 參數，並提供一或多個本機次要索引的規格。以下範例會建立資料表 (`Music`) 來保存音樂收藏中歌曲的相關資訊。分割區索引鍵為 `Artist`，而排序索引鍵為 `SongTitle`。`AlbumTitle` 屬性上的次要索引 `AlbumTitleIndex` 有助於依照專輯標題查詢。

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

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 ACTIVE。之後，您可以開始將資料項目放入資料表中。您可以使用 describe-table 來判斷資料表建立的狀態。

使用本機次要索引描述資料表

若要取得資料表上本機次要索引的資訊，請使用 describe-table 參數。對於每個索引，您可以存取其名稱、索引鍵結構描述和投影屬性。

```
aws dynamodb describe-table --table-name Music
```

查詢本機次要索引

您可以依照與 query 資料表大致相同的方式在本機次要索引上使用 query 操作。您必須指定索引名稱、索引排序索引鍵的查詢準則，以及您要傳回的屬性。在本例中，索引是 AlbumTitleIndex，而索引排序索引鍵為 AlbumTitle。

傳回的唯一屬性是已投影到索引的屬性。您也可以修改此查詢來選擇非索引鍵屬性，但這需要相對昂貴的資料表擷取活動。如需資料表擷取的詳細資訊，請參閱 屬性投影。

```
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"} }'
```

管理 DynamoDB 交易的複雜工作流程

Amazon DynamoDB Transactions 可簡化開發人員的使用體驗，讓這些人員同時對資料表內和跨資料表的多個項目，進行統籌協調式、全有或全無的變更。交易在 DynamoDB 中提供了不可分割性、一致性、隔離性和耐久性 (ACID)，讓您能夠輕鬆地維持應用程式的資料正確性。

您可以利用 DynamoDB 的交易讀取和寫入 API，來管理複雜的業務工作流程，這些流程需要將新增、更新或刪除多個項目的動作，當做全有或全無的單一操作來執行。例如，當玩家在電玩遊戲中交換物品或在遊戲中購買物品時，遊戲開發人員可以確保玩家的資料正確更新。

使用交易寫入 API，您可分組多個 `Put`、`Update`、`Delete` 和 `ConditionCheck` 動作。然後，將動作提交為單一 `TransactWriteItems` 操作，以單位形式成功或失敗。多項 `Get` 動作也可以相同的方式處理，在將這些動作分組後，當做單一 `TransactGetItems` 操作提交。

在您的 DynamoDB 資料表中啟用交易功能，不需額外付費。您只需針對交易中所進行的讀取和寫入付費即可。DynamoDB 會對交易中的每個項目進行兩項基本的讀取和寫入動作：一項是用來準備交易，一項是用來遞交交易。這兩項基本的讀取/寫入操作，會顯示在您的 Amazon CloudWatch 指標中。

若要開始使用 DynamoDB 交易，請下載最新的 AWS SDK 或 AWS Command Line Interface (AWS CLI)。然後遵循 DynamoDB 交易範例。

下列各節提供交易 API 的詳細概觀，並說明如何在 DynamoDB 中使用這些 API。

主題

- Amazon DynamoDB Transactions：運作方式
- 將 IAM 與 DynamoDB 交易搭配使用
- DynamoDB 交易範例

Amazon DynamoDB Transactions：運作方式

使用 Amazon DynamoDB Transactions 時，您可以將多項動作歸為一組，然後將這些動作以全有或全無的單一 `TransactWriteItems` 或 `TransactGetItems` 操作提交。下列各節說明 API 操作、容量管理、最佳實務，以及關於在 DynamoDB 中使用交易操作的其他詳細資訊。

主題

- TransactWriteItems API
- TransactGetItems API
- DynamoDB 交易的隔離層級
- DynamoDB 中的交易衝突處理
- 使用 DynamoDB Accelerator (DAX) 中的交易 API
- 交易的容量管理
- 交易的最佳實務

- 將交易 API 與全域資料表搭配使用
- DynamoDB 交易與 AWSLabs 用戶端程式庫的比較

TransactWriteItems API

`TransactWriteItems` 是一項冪等性的同步寫入操作，可將最多 100 個寫入動作分組成為全有或全無的單一操作。這些動作最多可將相同 AWS 帳戶和相同區域中一或多個 DynamoDB 資料表中的 100 個不同項目設為目標。交易中項目的彙總大小不能超過 4 MB。這些動作的完成具有不可分割性，也就是全部成功或全部失敗。

> **Note**
>
> - `TransactWriteItems` 操作與 `BatchWriteItem` 操作的不同之處，在於前者所包含的全部動作，都必須順利地完成，否則就完全不會進行變更。如果是使用 `BatchWriteItem` 操作，即可允許批次中只有某些動作順利完成 (其他的動作失敗)。
> - 無法使用索引執行交易。

在同一筆交易中，您無法針對同一個項目進行多項操作。例如，您不能對同一筆交易中的同一個項目既執行 `ConditionCheck` 又執行 `Update` 動作。

您可以將下列類型的動作新增到交易中：

- `Put`：發起 `PutItem` 操作，來建立新的項目，或是用新的項目取代舊的項目 (具有條件或不指定任何條件)。
- `Update`：發起 `UpdateItem` 操作，來編輯現有項目的屬性，或是在新項目尚未存在時，將新項目加入到資料表。利用此項動作，來新增、刪除或更新現有項目的屬性 (具有條件或無條件)。
- `Delete`：發起 `DeleteItem` 操作，透過指定資料表中某項目的主索引鍵，來刪除該單一項目。
- `ConditionCheck`：查看某項目是否存在，或是查看某項目特定屬性的條件。

交易在 DynamoDB 中完成時，其變更會開始傳播至全域次要索引 (GSI)、串流和備份。此傳播會逐步發生：來自相同交易的串流記錄，可能會在不同時間出現，且可能與其他交易的記錄交錯。串流取用者不應假設交易具有原子性或排序保證。

為了確保交易中修改項目的原子快照，請使用 TransactGetItems 操作，以同時讀取所有相關項目。此操作提供一致的資料檢視，確保您所見內容為已完成交易的所有變更，或完全不顯示任何變更。

因為傳播不是立即或瞬時完成，所以如果資料表是從備份還原 (RestoreTableFromBackup) 或匯出到中途傳播的某個時間點 (ExportTableToPointInTime)，可能只包含最近交易期間完成的部分變更。

冪等性

在進行 TransactWriteItems 呼叫時，您可以選擇性地加入用戶端符記，以確保請求具有冪等性。如果讓交易具有冪等性，則在因為連線逾時或其他連線問題，而多次提交同一項操作時，有助於防止應用程式發生錯誤。

如果原始的 TransactWriteItems 呼叫順利完成，便會成功傳回後續具有相同用戶端字符的 TransactWriteItems 呼叫，不建立任何變更。如果設定了 ReturnConsumedCapacity 參數，則初始的 TransactWriteItems 呼叫會傳回在建立變更時所使用的寫入容量單位數量。後續具有相同用戶端符記的 TransactWriteItems 呼叫，會傳回在讀取項目時所用掉的讀取容量單位數量。

關於冪等性的重點

- 用戶端符記在使用該符記的請求完成後 10 分鐘內為有效。10 分鐘過後，使用同一個用戶端符記的任何請求，會被視為新的請求。在經過 10 分鐘之後，您就不應該針對相同的請求，重複使用同一個用戶端符記。
- 如果您在 10 分鐘的冪等性有效期間內，使用相同的用戶端符記來重複請求，但變更了其他的某些請求參數，則 DynamoDB 會傳回 IdempotentParameterMismatch 例外。

寫入的錯誤處理

在下列的情況中，寫入交易不會成功：

- 當其中一個條件表達式中的條件不符時。
- 因為在同一個 TransactWriteItems 操作中，有超過一個以上的動作鎖定相同的項目，而造成交易驗證錯誤時。
- 如果 TransactWriteItems 請求與 TransactWriteItems 請求中一或多個項目持續執行的 TransactWriteItems 操作相互衝突時。在這種情況中，請求會失敗，並且丟出 TransactionCanceledException 例外。
- 佈建的容量不足，而使交易無法完成時。
- 當項目的大小過大 (超過 400 KB)、本機次要索引 (LSI) 變得過大，或是因為交易所進行的變更造成類似的驗證錯誤時。
- 發生使用者錯誤時，例如無效的資料格式。

如需如何處理 TransactWriteItems 操作衝突的詳細資訊，請參閱 DynamoDB 中的交易衝突處理。

TransactGetItems API

TransactGetItems 是一項同步讀取操作，可將最多 100 個 Get 動作歸成一組。這些動作最多可將相同 AWS 帳戶和區域中一或多個 DynamoDB 資料表中的 100 個不同項目設為目標。交易中項目的彙總大小不能超過 4 MB。

Get 動作的執行具有不可分割性，也就是全部成功或全部失敗：

- Get：發起 GetItem 操作，針對具有指定主索引鍵的項目，擷取該項目的一組屬性。如果找不到符合的項目，則 Get 不會傳回任何資料。

讀取的錯誤處理

在下列的情況中，讀取交易不會成功：

- 如果 TransactGetItems 請求與 TransactGetItems 請求中一或多個項目持續執行的 TransactWriteItems 操作相互衝突時。在這種情況中，請求會失敗，並且丟出 TransactionCanceledException 例外。
- 佈建的容量不足，而使交易無法完成時。
- 發生使用者錯誤時，例如無效的資料格式。

如需如何處理 TransactGetItems 操作衝突的詳細資訊，請參閱 DynamoDB 中的交易衝突處理。

DynamoDB 交易的隔離層級

交易操作 (TransactWriteItems 或 TransactGetItems) 和其他操作的隔離層級如下。

可序列化

如果在前一項操作完成之前，沒有任何操作開始執行，則可序列化隔離可確保多個同時並行操作的結果會相同。

在下列的操作類型之間，具有可序列化的隔離層級：

- 在任何交易操作和任何標準寫入操作 (PutItem、UpdateItem 或 DeleteItem) 之間。
- 在任何交易操作和任何標準讀取操作 (GetItem) 之間。

- 在 TransactWriteItems 操作和 TransactGetItems 操作之間。

雖然在交易操作之間具有可序列化隔離層級，而且每個個別的標準會在 BatchWriteItem 操作中寫入，但是在交易和做為單位的 BatchWriteItem 操作之間，不具有可序列化隔離層級。

同樣地，交易操作與 BatchGetItem 操作中個別 GetItems 之間的隔離層是可以序列化。但是交易與當作一個單位的 BatchGetItem 操作之間的隔離層是專供讀取。

單一 GetItem 請求是兩種可序列化的 TransactWriteItems 請求方式之一，可以發生在 TransactWriteItems 請求之前或之後。與 TransactWriteItems 請求同時發出的多個 GetItem 請求，能夠以任何順序執行，因此結果會是專供讀取。

例如，如果項目 A 和項目 B 的 GetItem 請求與修改項目 A 和項目 B 的 TransactWriteItems 請求同時執行，則會有四種可能性：

- 兩個 GetItem 請求皆會在 TransactWriteItems 請求之前執行。
- 兩個 GetItem 請求皆會在 TransactWriteItems 請求之後執行。
- 項目 A 的 GetItem 請求會在 TransactWriteItems 請求之前執行。針對項目 B，GetItem 會在 TransactWriteItems 之後執行。
- 項目 B 的 GetItem 請求會在 TransactWriteItems 請求之前執行。針對項目 A，GetItem 會在 TransactWriteItems 之後執行。

如果您偏好適用於執行多個 GetItem 請求的可序列化隔離等級，應使用 TransactGetItems。

如果針對多個項目執行非交易式讀取，而這些項目原本屬於同一筆傳輸中的交易寫入請求，可能會出現部分項目可讀取到新狀態，而其他項目仍顯示舊狀態的情況。唯有在交易成功寫入並收到回應，表示該交易已完成時，您才能讀取原屬於該交易寫入請求之所有項目的新狀態。

交易成功完成並收到回應後，後續的最終一致讀取操作在短時間內仍可能傳回舊狀態，這是由於 DynamoDB 的最終一致性模式所致。為了確保在交易後立即讀取最新狀態資料，您應該將 ConsistentRead 設定為 true，使用高度一致性讀取。

專供讀取

專供讀取隔離可確保讀取操作對特定項目一律傳回已確認的值 - 如果交易寫入最終未成功，則讀取操作一律不向該項目呈現此寫入狀態的檢視。專供讀取隔離不會防止在讀取操作後立即修改項目。

在任何交易操作和牽涉到多次標準讀取 (BatchGetItem、Query 或 Scan) 的任何讀取操作之間，其隔離層級為專供讀取。如果交易寫入在 BatchGetItem、Query 或 Scan 操作期間更新項目，則讀

取操作的後續部分會傳回新確認的值 (含有 ConsistentRead) 或可能是先前已確認的值(最終一致讀取)。

操作摘要

做為總結，下表顯示了交易操作 (TransactWriteItems 或 TransactGetItems) 和其他操作之間的隔離層級。

作業	隔離層級
DeleteItem	可序列化
PutItem	可序列化
UpdateItem	可序列化
GetItem	可序列化
BatchGetItem	專供讀取*
BatchWriteItem	不可序列化*
Query	專供讀取
Scan	專供讀取
其他交易操作	可序列化

標有星號的層級 (*) 適用於單位式操作。不過，這些操作內的個別動作具有可序列化的隔離層級。

DynamoDB 中的交易衝突處理

在交易內的項目上進行並行項目層級請求期間，可能會發生交易衝突。下列情境可能發生交易衝突：

- 項目的 PutItem、UpdateItem 或 DeleteItem 請求與包括相同項目的持續 TransactWriteItems 請求發生衝突。
- TransactWriteItems 請求內的項目是另一個持續 TransactWriteItems 請求的一部分。
- TransactGetItems 請求的項目是持續 TransactWriteItems、BatchWriteItem、PutItem、UpdateItem 或 DeleteItem 請求的一部分。

> **Note**
>
> - 當 PutItem、UpdateItem 或 DeleteItem 請求遭到拒絕時，請求失敗並顯示 TransactionConflictException。
> - 如果 TransactWriteItems 或 TransactGetItems 內的項目層級請求遭到拒絕，則請求失敗並顯示 TransactionCanceledException。如果請求失敗， AWS SDKs不會重試請求。
>
> 如果您使用的是 適用於 Java 的 AWS SDK，則例外狀況會包含 CancellationReasons 的清單，並根據TransactItems請求參數中的項目清單排序。對於其他語言，清單的字串表示法會併入異常的錯誤訊息中。
> - 不過，如果有持續執行的 TransactWriteItems 和 TransactGetItems 操作，與同時並行的 GetItem 請求相互衝突時，這兩項操作都可以順利完成。

對於每個失敗的項目層級請求，TransactionConflict CloudWatch 指標會遞增。

使用 DynamoDB Accelerator (DAX) 中的交易 API

DynamoDB Accelerator (DAX) 中支援 TransactWriteItems 和 TransactGetItems，且隔離層級與在 DynamoDB 中相同。

TransactWriteItems 會透過 DAX 寫入。DAX 會將 TransactWriteItems 呼叫傳送給 DynamoDB，然後傳回回應。為在寫入後填入快取，對於 TransactWriteItems 操作中的每個項目，DAX 會在背景呼叫 TransactGetItems，並耗用更多讀取容量單位。(如需詳細資訊，請參閱 交易的容量管理。) 此功能可讓您保持簡單的應用程式邏輯，並使用 DAX 執行交易及非交易操作。

TransactGetItems 呼叫透過 DAX 傳遞，項目不在本機進行快取。這個行為與 DAX 中的高度一致性讀取 API 相同。

交易的容量管理

在您的 DynamoDB 資料表中啟用交易功能，不需額外付費。您只需針對交易中所進行的讀取和寫入付費即可。DynamoDB 會對交易中的每個項目進行兩項基本的讀取和寫入動作：一項是用來準備交易，一項是用來遞交交易。這兩項基本的讀取/寫入操作，會顯示在您的 Amazon CloudWatch 指標中。

在為資料表佈建容量時，請規劃交易 API 所需的額外讀取與寫入容量。例如，假設您的應用程式每秒執行一項交易，而每項交易會在您的資料表中寫入三個 500 位元組的項目。每個項目需要兩個寫入容

量單位 (WCU)：一個單位用來準備交易，另一個單位用來遞交交易。因此，您會需要佈建六個 WCU 給該資料表。

如果您在前一個範例中使用 DynamoDB Accelerator (DAX)，則也會針對 `TransactWriteItems` 呼叫中的每個項目使用兩個讀取容量單位 (RCU)。因此，您會需要佈建六個額外 RCU 給資料表。

同樣地，如果您的應用程式每秒執行一次讀取交易，而每項交易會在您的資料表中讀取三個 500 位元組的項目，則您會需要佈建六個讀取容量單位 (RCU) 給該資料表。讀取每個項目需要兩個 RCU：一個用來準備交易，另一個用來遞交交易。

此外，在出現 `TransactionInProgressException` 例外時，預設的 SDK 動作是重試交易。請規劃這些重試動作會使用的額外讀取容量單位 (RCU)。如果用您自己的程式碼，使用 `ClientRequestToken` 來重試交易時，也請進行同樣的規劃。

交易的最佳實務

在使用 DynamoDB 交易時，請考慮採用下列建議的做法。

- 啟用資料表的自動擴展功能，或是確定您已佈建足夠的輸送容量，來針對您交易中的每個項目，進行兩項讀取或寫入操作。
- 如果您不是使用 AWS 提供的 SDK，請在`TransactWriteItems`呼叫時包含 `ClientRequestToken` 屬性，以確保請求是等冪的。
- 如非必要，請勿在交易中將操作歸為一組。例如，如果包含 10 項操作的單一交易，可以分成多項交易執行，而不會影響到應用程式的正確度，則我們建議拆分該項交易。簡化的交易可提升輸送量，而且成功的機率更高。
- 更新同一個項目的多項交易如果同時執行，可能會造成衝突而導致交易取消。我們建議採用下列的 DynamoDB 資料建模最佳實務，來將此等衝突減到最少。
- 如果在單一交易中，經常更新跨多個項目的一組屬性，請考慮將這些屬性分組成為單一項目，來縮小交易的範圍。
- 避免使用大量擷取資料的交易。如果是大量寫入作業，較理想的做法是使用 `BatchWriteItem`。

將交易 API 與全域資料表搭配使用

交易操作僅在呼叫寫入 API 的 AWS 區域內提供原子性、一致性、隔離性和耐久性 (ACID) 保證。全域資料表不支援跨區域交易。舉例來說，假設您在美國東部 (俄亥俄) 與美國西部 (奧勒岡) 區域中有具有複本的全域資料表，並且在美國東部 (維吉尼亞北部) 區域中執行 `TransactWriteItems` 操作。由於變更受到複寫，所以您可能會在美國西部 (奧勒岡) 區域中看到部分已完成交易。當變更已在來源區域遞交後，這些變更才會複寫至其他區域。

DynamoDB 交易與 AWSLabs 用戶端程式庫的比較

DynamoDB 交易針對 AWSLabs 交易用戶端程式庫，提供了更具成本效益、更強大和更高效能的替代方法。我們建議您更新應用程式，以使用原生的伺服器端交易 API。

將 IAM 與 DynamoDB 交易搭配使用

您可以使用 AWS Identity and Access Management (IAM) 來限制交易操作可在 Amazon DynamoDB 中執行的動作。如需在 DynamoDB 中使用 IAM 政策的詳細資訊，請參閱 適用於 DynamoDB 的以身分為基礎的政策。

`Put`、`Update`、`Delete` 和 `Get` 動作的許可，受到底層 `PutItem`、`UpdateItem`、`DeleteItem` 和 `GetItem` 操作使用的許可所管理。如果是 `ConditionCheck` 動作，您可以使用 IAM 政策中的 `dynamodb:ConditionCheckItem` 許可。

您可以使用下列的 IAM 政策範例，來設定 DynamoDB 交易。

範例 1：允許交易操作

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ]
        }
    ]
}
```

範例 2：只允許交易操作

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        }
    ]
}
```

範例 3：允許非交易讀取與寫入，並且封鎖交易讀取與寫入

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
```

```
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
             "Action": [
                 "dynamodb:PutItem",
                 "dynamodb:DeleteItem",
                 "dynamodb:GetItem",
                 "dynamodb:UpdateItem"
             ],
             "Resource": [
                 "arn:aws:dynamodb:*:*:table/table04"
             ]
         }
    ]
}
```

範例 4：防止在 ConditionCheck 失敗時傳回資訊

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/table01",
            "Condition": {
                "StringEqualsIfExists": {
                    "dynamodb:ReturnValues": "NONE"
                }
            }
        }
    ]
}
```

DynamoDB 交易範例

以 Amazon DynamoDB Transactions 十分實用的情況作為範例，請考慮此線上市場的 Java 應用程式範例。

應用程式在其後端有三個 DynamoDB 資料表：

- `Customers`：此資料表儲存有關市場客戶的詳細資訊。其主索引鍵為 `CustomerId` 唯一識別符。
- `ProductCatalog`：此資料表儲存的詳細資訊包括關於市場上銷售之產品的價格和供貨情況。其主索引鍵為 `ProductId` 唯一識別符。
- `Orders`：此資料表儲存有關市場訂單的詳細資訊。其主索引鍵為 `OrderId` 唯一識別符。

下訂單

下列程式碼片段說明如何使用 DynamoDB 交易來協調建立和處理訂單所需的多個步驟。使用單一「全有或全無」操作，可確保如果交易的任何部分失敗，便不會執行交易中的任何動作，也不會進行任何變更。

在此範例中，您可以為 `customerId` 為 `09e8e9c8-ec48` 的客戶設定訂單。然後，您可以使用下列簡單的訂單處理工作流程，將其當作單一交易執行：

1. 判斷客戶 ID 有效。
2. 確定產品為 IN_STOCK，並將產品狀態更新為 SOLD。
3. 請確定訂單不存在，並建立訂單。

驗證客戶

首先，請定義一個動作，以便驗證客戶資料表中是否存在 customerId 等於 09e8e9c8-ec48 的客戶。

```
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 + ")");
```

更新產品狀態

接下來，如果產品狀態目前是將 IN_STOCK 設定為 true，請定義要將產品狀態更新至 SOLD 的動作。如果項目的產品狀態屬性不等於 IN_STOCK，則設定 ReturnValuesOnConditionCheckFailure 參數會傳回項目。

```
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);
```

建立訂單

最後，只要具有 OrderId 的訂單尚不存在，便可建立訂單。

```
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 + ")");
```

執行交易

下列範例說明如何執行先前定義為「全有或全無」的單一操作動作。

```
    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());
    }
```

讀取訂單詳細資訊

下列範例示範如何以交易方式讀取 Orders 和 ProductCatalog 資料表中的已完成訂單。

```
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());
}
```

使用 Amazon DynamoDB 變更資料擷取

當存放在 DynamoDB 資料表中的項目發生變更時，擷取這類變更的功能可為許多應用程式提供好處。下列是一些範例使用案例：

- 熱門行動應用程式以每秒數千筆更新的速率修改 DynamoDB 資料表中的資料。另一個應用程式擷取及存放這些更新的相關資料，為行動應用程式提供近乎即時的用量指標。
- 財務應用程式會修改 DynamoDB 資料表中的股票市場資料。平行執行的不同應用程式會即時追蹤這些變更，計算風險價值，並根據股票價格變動自動重新平衡投資組合。
- 運輸車輛和工業設備中的感應器會將資料傳送至 DynamoDB 資料表。不同的應用程式會監控效能並在偵測到問題時傳送簡訊提醒，透過應用機器學習演算法預測任何潛在缺陷，以及將資料壓縮和封存到 Amazon Simple Storage Service (Amazon S3)。
- 應用程式在某位朋友上傳新圖片時，立即自動傳送通知給群組中所有朋友的行動裝置。
- 新客戶將資料新增至 DynamoDB 資料表。此事件調用另一個應用程式，該應用程式會將歡迎電子郵件傳送給新客戶。

DynamoDB 支援近乎即時的項目層級變更資料擷取紀錄串流。您可以建立使用這些串流並根據內容採取動作的應用程式。

> **Note**
>
> 新增標籤至 DynamoDB Streams 與透過 DynamoDB Streams 使用屬性型存取控制 (ABAC)，目前未提供支援。

以下影片將為您介紹變更資料擷取概念。

資料表容量模式

主題

- 變更資料擷取的串流選項
- 使用 Kinesis Data Streams 來擷取 DynamoDB 的變更
- DynamoDB Streams 的變更資料擷取

變更資料擷取的串流選項

DynamoDB 提供兩種用於變更資料擷取的串流模型：DynamoDB 專用 Kinesis Data Streams 和 DynamoDB Streams。

為了協助您為應用程式選擇合適的解決方案，下表摘要說明每種串流模型的特色。

Properties	DynamoDB 專用 Kinesis Data Streams	DynamoDB Streams
資料保留	最多 1 年。	24 小時。
Kinesis Client Library (KCL) 支援	支援 KCL 1.X、2.X 版和 3.X 版。	支援 KCL 1.X 版和 2.X 版。
取用者數量	每個碎片最多 5 個同時取用者，每個具有強化廣發功能的碎片最多 20 個同時取用者。	每個碎片最多 2 個同時取用者。
輸送量配額	無限制。	受限於 DynamoDB 資料表和 AWS 區域的輸送量配額。
記錄交付模式	使用 GetRecords 和強化廣發功能透過 HTTP 提取模型，Kinesis Data Streams 使用 SubscribeToShard 透過 HTTP/2 推送紀錄。	使用 GetRecords 透過 HTTP 提取模型。
記錄排序	每個串流紀錄上的時間戳記屬性可用來識別 DynamoDB 資料表中發生變更的實際順序。	對於 DynamoDB 資料表中的每個修改項目，串流紀錄的出現順序與項目的實際修改順序相同。
複製記錄	串流中偶爾會出現重複紀錄。	串流中沒有出現重複紀錄。
串流處理選項	使用 AWS Lambda、Amazon Managed Service for Apache Flink、Kinesis Data Firehose	使用 AWS Lambda 或 DynamoDB Streams Kinesis 轉接器處理串流記錄。

Properties	DynamoDB 專用 Kinesis Data Streams	DynamoDB Streams
	或 AWS Glue Streaming ETL 來處理串流記錄。	
耐久性等級	可用區域會提供自動不中斷的容錯移轉。	可用區域會提供自動不中斷的容錯移轉。

您可以在同一個 DynamoDB 資料表上啟用這兩種串流模型。

下面這段影片將進一步說明這兩個選項之間的差異。

DynamoDB Streams 與 Kinesis Data Streams

使用 Kinesis Data Streams 來擷取 DynamoDB 的變更

您可以使用 Amazon Kinesis Data Streams 來擷取 Amazon DynamoDB 的變更。

Kinesis Data Streams 會擷取任何 DynamoDB 資料表中的項目層級修改，並將其複製到您選擇的 Kinesis 資料串流。您的應用程式可以存取此串流，並以近乎即時的速度檢視項目層級的變更。您每小時可以持續擷取和儲存 TB 級的資料。您可以利用較長的資料保留時間，並透過增強的廣發功能，您可以同時觸及兩個以上的下游應用程式。其他優點包括額外的稽核和安全透明度。

Kinesis Data Streams 也可讓您存取 Amazon Data Firehose 和 Amazon Managed Service for Apache Flink。這可讓您建置應用程式來啟動即時儀表板、產生提醒、實作動態定價和廣告，以及執行複雜的資料分析及機器學習演算法。

> **Note**
>
> 為 DynamoDB 使用 Kinesis 資料串流同時受到資料串流的 Kinesis Data Streams 定價和來源資料表的 DynamoDB 定價的約束。

若要使用主控台 AWS CLI或 Java SDK 在 DynamoDB 資料表上啟用 Kinesis 串流，請參閱 Amazon DynamoDB 專用 Kinesis Data Streams 入門。

主題

- Kinesis Data Streams 如何與 DynamoDB 搭配運作

- Amazon DynamoDB 專用 Kinesis Data Streams 入門
- 以 DynamoDB Streams 和 Kinesis Data Streams 使用碎片和指標
- 使用 Amazon Kinesis Data Streams 和 Amazon DynamoDB 專用的 IAM 政策

Kinesis Data Streams 如何與 DynamoDB 搭配運作

當 DynamoDB 資料表啟用 Kinesis 資料串流時，資料表會傳送資料記錄，以擷取該資料表資料的任何變更。此資料記錄包括：

- 最近建立、更新或刪除之任何項目的具體時間
- 該項目的主索引鍵
- 修改前記錄的快照
- 修改後記錄的快照

這些資料記錄會以接近即時的速度擷取和發布，並予以發布。將其寫入 Kinesis 資料串流後，就可以像其他記錄一樣讀取。您可以使用 Kinesis Client Library、使用 AWS Lambda、呼叫 Kinesis Data Streams API，以及使用其他連線的服務。如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的從 Amazon Kinesis Data Streams 讀取資料。

這些對資料的變更也會以非同步方式擷取。Kinesis 對其串流來源的資料表沒有效能影響。存放在 Kinesis 資料串流中的串流紀錄會採用靜態加密。如需詳細資訊，請參閱 Amazon Kinesis Data Streams 中的資料保護。

Kinesis 資料串流記錄的顯示順序，可能與項目變更發生的順序不同。相同的項目通知也可能會在串流中多次出現。您可以檢查 `ApproximateCreationDateTime` 屬性來識別項目修改發生的順序，並識別重複的記錄。

當您啟用 Kinesis 資料串流做為 DynamoDB 資料表串流目的地時，您可以毫秒或微秒為單位，設定 `ApproximateCreationDateTime` 值的精確度。根據預設，`ApproximateCreationDateTime` 會以毫秒為單位指出變更的時間。此外，您可以在作用中的串流目的地上變更此值。更新後，寫入 Kinesis 的串流記錄會以所需的精確度記錄 `ApproximateCreationDateTime` 值。

寫入 DynamoDB 的二進位值必須使用 base64 編碼格式進行編碼。然而，當資料記錄寫入 Kinesis 資料串流時，這些編碼的二進位值會使用 base64 編碼再次編碼。從 Kinesis 資料串流讀取這些記錄時，為了擷取原始二進位值，應用程式必須將這些值解碼兩次。

使用 Kinesis Data Streams 時，DynamoDB 會根據變更資料擷取單位收取費用。每個單一項目的 1 KB 變更會計為一個變更資料擷取單位。使用與寫入作業的容量單位耗用量相同的邏輯，以寫入串流之項目

的「之前」和「之後」映像中較大者，計算每個項目的 KB 變化量。運作方式與 DynamoDB 隨需模式類似，您不需要為變更資料擷取單位佈建容量輸送量。

為 DynamoDB 資料表開啟 Kinesis 資料串流

您可以使用 、 AWS 管理主控台 AWS SDK 或 AWS Command Line Interface ()，從現有的 DynamoDB 資料表啟用或停用串流至 Kinesis AWS CLI。

- 您只能在與資料表相同的 AWS 帳戶和 AWS 區域中，將資料從 DynamoDB 串流到 Kinesis Data Streams。
- 您只能將 DynamoDB 資料表中的資料串流至一個 Kinesis 資料串流。

在 DynamoDB 資料表上變更 Kinesis Data Streams 目的地

根據預設，所有 Kinesis 資料串流記錄都包含 `ApproximateCreationDateTime` 屬性。此屬性代表每個記錄建立當下約略的時間戳記，以毫秒為單位。您可以使用 https://console.aws.amazon.com/kinesis、 SDK 或 來變更這些值的精確度 AWS CLI

Amazon DynamoDB 專用 Kinesis Data Streams 入門

本節說明如何搭配 Amazon DynamoDB 主控台、 AWS Command Line Interface (AWS CLI) 和 API 使用 Amazon DynamoDB 資料表的 Kinesis Data Streams。

建立作用中的 Amazon Kinesis 資料串流

所有這些範例都使用 `Music` DynamoDB 資料表，該資料表是在 DynamoDB 入門教學課程中建立的。

若要進一步了解如何建置取用者並將 Kinesis 資料串流連線至其他 AWS 服務，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的從 Kinesis Data Streams 讀取資料。

> **Note**
>
> 第一次使用 KDS 碎片時，建議您將碎片設定為隨著使用模式向上擴展和縮減。累積更多使用模式的相關資料後，您可以調整串流中的碎片以進行配對。

Console

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/kinesis/ 的 Kinesis 主控台。

2. 選擇 Create data stream (建立資料串流)，並依照說明來建立名為 `samplestream` 的串流。
3. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
4. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
5. 選擇 Music (音樂) 資料表。
6. 選擇 Exports and streams (匯出與串流) 索引標籤。
7. (選用) 您可以在 Amazon Kinesis 資料串流詳細資訊下，將記錄時間戳記精確度從微秒 (預設) 變更為毫秒。
8. 從下拉式清單選擇 samplestream (範例串流)。
9. 選擇 Turn On (開啟) 按鈕。

AWS CLI

1. 使用 create-stream 命令建立名為 `samplestream` 的 Kinesis 資料串流。

```
aws kinesis create-stream --stream-name samplestream --shard-count 3
```

請先參閱 Kinesis Data Streams 的碎片管理考量事項，再為 Kinesis 資料串流設定碎片數量。

2. 使用 describe-stream 命令確認 Kinesis 串流是否處於作用中狀態且可供使用。

```
aws kinesis describe-stream --stream-name samplestream
```

3. 使用 DynamoDB `enable-kinesis-streaming-destination` 命令在 DynamoDB 資料表上啟用 Kinesis 串流功能。將 `stream-arn` 值替換為上一個步驟中傳回的值 `describe-stream`。可選擇性啟用時間戳記值精確度更精細 (微秒級) 的串流，傳回至每筆記錄上。

啟用微秒級時間戳記精確度串流：

```
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
```

或啟用預設時間戳記精確度串流 (毫秒級)：

```
aws dynamodb enable-kinesis-streaming-destination \
  --table-name Music \
```

```
  --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
```

4. 使用 DynamoDB `describe-kinesis-streaming-destination` 命令確認資料表上的 Kinesis 串流是否處於作用中狀態。

```
aws dynamodb describe-kinesis-streaming-destination --table-name Music
```

5. 使用 `put-item` 命令將資料寫入 DynamoDB 資料表，如《DynamoDB 開發人員指南》中所述。

```
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. 使用 Kinesis get-records CLI 命令來擷取 Kinesis 串流內容。然後使用下面的程式碼片段來將串流內容還原序列化。

```
/**
 * 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. 遵循《Kinesis Data Streams 開發人員指南》中的說明，使用 Java 建立名為 `samplestream` 的 Kinesis 資料串流。

   請先參閱 Kinesis Data Streams 的碎片管理考量事項，再為 Kinesis 資料串流設定碎片數量。

2. 使用以下程式碼片段來啟用 DynamoDB 資料表上的 Kinesis 串流。可選擇性啟用時間戳記值精確度更精細 (微秒級) 的串流，傳回至每筆記錄上。

   啟用微秒級時間戳記精確度串流：

```
EnableKinesisStreamingConfiguration enableKdsConfig =
 EnableKinesisStreamingConfiguration.builder()

 .approximateCreationDateTimePrecision(ApproximateCreationDateTimePrecision.MICROSECOND)
  .build();

EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .enableKinesisStreamingConfiguration(enableKdsConfig)
  .build();
```

```
EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

或啟用預設時間戳記精確度串流 (毫秒級)：

```
EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

3. 遵循《Kinesis Data Streams 開發人員指南》中的說明，從建立的資料串流中進行讀取。
4. 然後使用下面的程式碼片段將串流內容還原序列化

```
/**
 * 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());
}
```

變更作用中的 Amazon Kinesis 資料串流

本節說明如何使用 主控台 AWS CLI 和 API 來變更作用中的 Kinesis Data Streams for DynamoDB 設定。

AWS 管理主控台

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 前往您的資料表。
3. 選擇匯出與串流。

AWS CLI

1. 呼叫 `describe-kinesis-streaming-destination` 以確認串流為 ACTIVE。
2. 呼叫 `UpdateKinesisStreamingDestination`，例如在此範例中：

```
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. 呼叫 `describe-kinesis-streaming-destination` 以確認串流為 UPDATING。
4. 定期呼叫 `describe-kinesis-streaming-destination`，直到串流狀態再次變成 ACTIVE 為止。時間戳記精確度更新通常需要 5 分鐘才會生效。一旦此狀態更新，表示更新已完成，且新的精確度值將套用至未來的記錄。
5. 使用 `putItem` 寫入資料表。

6. 使用 Kinesis `get-records` 命令來取得串流內容。
7. 確認寫入的 `ApproximateCreationDateTime` 具有所需的精確度。

Java API

1. 提供可建構 `UpdateKinesisStreamingDestination` 請求和 `UpdateKinesisStreamingDestination` 回應的程式碼片段。
2. 提供可建構 `DescribeKinesisStreamingDestination` 請求和 `DescribeKinesisStreamingDestination response` 回應的程式碼片段。
3. 定期呼叫 `describe-kinesis-streaming-destination`，直到串流狀態再次變成 `ACTIVE`，表示更新已完成，且新的精確度值將套用至未來的記錄。
4. 執行對資料表的寫入。
5. 從串流讀取並還原序列化串流內容。
6. 確認寫入的 `ApproximateCreationDateTime` 具有所需的精確度。

以 DynamoDB Streams 和 Kinesis Data Streams 使用碎片和指標

Kinesis Data Streams 的碎片管理考量事項

Kinesis 資料串流會計算其在碎片中的輸送量。在 Amazon Kinesis Data Streams 中，您可為資料串流選擇隨需模式或佈建模式。

如果您的 DynamoDB 寫入工作負載高度可變且無法預測，我們建議您使用 Kinesis Data Streams 的隨需模式。使用隨需模式時，無需容量規劃，因為 Kinesis Data Streams 會自動管理碎片，提供必要的輸送量。

對於可預測的工作負載，您可以針對 Kinesis Data Stream 使用佈建模式。採用佈建模式，必須指定資料串流的碎片數目，以容納來自 DynamoDB 的變更資料擷取紀錄。若要判斷 Kinesis 資料串流支援 DynamoDB 資料表所需的碎片數目，您需要以下輸入值：

- DynamoDB 資料表紀錄的平均大小 (以位元組為單位) (`average_record_size_in_bytes`)。
- 您的 DynamoDB 資料表上將執行的每秒寫入操作數目上限。這包括應用程式執行的建立、刪除和更新操作，以及自動產生的操作，例如「存留時間」產生的刪除操作 (`write_throughput`)。
- 與建立或刪除操作相比，您在資料表上執行的更新和覆寫操作的百分比 (`percentage_of_updates`)。請注意，更新和覆寫操作會將已修改項目的舊映像和新映像複製到串流。因此產生兩倍大小的 DynamoDB 項目。

您可使用以下公式帶入輸入值，計算 Kinesis 資料串流所需的碎片數目 (number_of_shards)：

```
number_of_shards = ceiling( max( ((write_throughput * (4+percentage_of_updates) *
 average_record_size_in_bytes) / 1024 / 1024), (write_throughput/1000)), 1)
```

例如，您可能有每秒 1040 個寫入操作 (write_throughput) 的最大輸送量，平均記錄大小為 800 位元組 (average_record_size_in_bytes)。如果其中 25% 的寫入操作為更新操作 (percentage_of_updates)，您需要兩個碎片 (number_of_shards) 以容納 DynamoDB 串流輸送量：

```
ceiling( max( ((1040 * (4+25/100) * 800)/ 1024 / 1024), (1040/1000)), 1).
```

使用公式計算 Kinesis 資料串流佈建模式所需的碎片數量之前，請考慮下列事項：

- 此公式有助於預估容納 DynamoDB 變更資料記錄所需的碎片數量。其不代表 Kinesis 資料串流中所需的碎片總數，例如支援額外 Kinesis 資料串流取用者所需的碎片數量。
- 如果您未設定處理尖峰輸送量的資料串流，則可能仍會在佈建模式中遇到讀取和寫入輸送量例外狀況。在這種情形下，您必須手動擴展資料串流以適應資料流量。
- 此公式會考量 DynamoDB 將變更日誌資料記錄串流至 Kinesis Data Stream 之前，所產生的額外膨脹。

如需進一步了解 Kinesis Data Stream 上的容量模式，請參閱選擇資料串流容量模式。如需進一步了解不同容量模式之間的定價差異，請參閱 Amazon Kinesis Data Streams 定價。

使用 Kinesis Data Streams 監控變更資料擷取

DynamoDB 提供數個 Amazon CloudWatch 指標，可協助您監控將變更資料擷取到 Kinesis 的複寫。如需 CloudWatch 指標的完整清單，請參閱 DynamoDB 指標和維度。

為判斷串流是否有足夠的容量，建議您在啟用串流期間和生產環境期間監控下列項目：

- ThrottledPutRecordCount：因為 Kinesis 資料串流容量不足而受到 Kinesis Data Streams 限流的記錄數量。儘管在特殊使用峰值期間，您可能會遇到限流的情況，仍應盡可能降低 ThrottledPutRecordCount。DynamoDB 會重試將限流記錄傳送到 Kinesis 資料串流，但這可能會導致較高的複寫延遲。

  如果遇到過多且規律的限流，您可能需要按照觀察到的資料表寫入輸送量按比例增加 Kinesis 串流碎片的數量。若要進一步了解如何判斷 Kinesis 資料串流的大小，請參閱判斷 Kinesis Data Stream 的初始大小。

- `AgeOfOldestUnreplicatedRecord`：自 DynamoDB 資料表中出現尚未複寫到 Kinesis 資料串流的最舊項目層級變更以來經過的時間。在正常的操作下，`AgeOfOldestUnreplicatedRecord` 應該以毫秒為單位。當不成功的複寫嘗試是因客戶控制的組態選擇所引起時，此數字會隨著不成功複寫嘗試的增加而增加。

  如果 `AgeOfOldestUnreplicatedRecord` 指標超過 168 小時，從 DynamoDB 資料表到 Kinesis 資料串流的項目層級變更複寫將自動停用。

  可能導致複寫嘗試失敗的客戶控制組態示例包括，佈建的 Kinesis 資料串流容量不足導致過度限流，或是手動更新 Kinesis 資料串流的存取政策因而拒絕 DynamoDB 新增資料至您的資料串流。為盡可能降低此指標，您可能需要確保妥善佈建您的 Kinesis 資料串流容量，並確保 DynamoDB 的許可保持不變。
- `FailedToReplicateRecordCount`：DynamoDB 無法複寫到您的 Kinesis 資料串流的記錄數目。大於 34KB 的某些項目可能會擴充大小，以變更大於 Kinesis Data Streams 1MB 項目大小限制的資料記錄。當這些大於 34KB 的項目包含大量的布林值或空白屬性值時，就會發生此大小擴充。布林值和空白屬性值會以 1 位元組形式儲存在 DynamoDB 中，但是在使用用於 Kinesis Data Streams 複寫的標準 JSON 將其序列化時，最多可擴充至 5 個位元組。DynamoDB 無法將這類變更記錄複寫到您的 Kinesis 資料串流。DynamoDB 會略過這些變更資料記錄，並自動繼續複寫後續記錄。

您可以建立 Amazon CloudWatch 警示，以便在上述任何指標超過特定閾值時，傳送 Amazon Simple Notification Service (Amazon SNS) 通知訊息。

使用 Amazon Kinesis Data Streams 和 Amazon DynamoDB 專用的 IAM 政策

第一次為 Amazon DynamoDB 啟用 Amazon Kinesis Data Streams 時，DynamoDB 會自動為您建立 AWS Identity and Access Management (IAM) 服務連結角色。 DynamoDB 這個角色 (`AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`) 可讓 DynamoDB 代表您管理對 Kinesis Data Streams 的項目層級變更的複寫。請勿刪除此服務連結角色。

如需服務連結角色的詳細資訊，請參閱《IAM 使用者指南》中的使用服務連結角色。

> **Note**
>
> DynamoDB 不支援 IAM 政策的標籤型條件。

若要啟用 Amazon DynamoDB 專用 Amazon Kinesis Data Streams，您必須擁有下列資料表許可：

- dynamodb:EnableKinesisStreamingDestination
- kinesis:ListStreams
- kinesis:PutRecords
- kinesis:DescribeStream

若要為指定的 DynamoDB 資料表描述 Amazon DynamoDB 專用 Amazon Kinesis Data Streams，您必須擁有下列資料表許可。

- dynamodb:DescribeKinesisStreamingDestination
- kinesis:DescribeStreamSummary
- kinesis:DescribeStream

若要停用 Amazon DynamoDB 專用 Amazon Kinesis Data Streams，您必須擁有下列資料表許可。

- dynamodb:DisableKinesisStreamingDestination

若要更新 Amazon DynamoDB 專用 Amazon Kinesis Data Streams，您必須擁有下列資料表許可。

- dynamodb:UpdateKinesisStreamingDestination

下列範例顯示如何使用 IAM 政策授予 Amazon DynamoDB 專用 Amazon Kinesis Data Streams 的許可。

範例：啟用 Amazon DynamoDB 專用 Amazon Kinesis Data Streams

以下 IAM 政策會授予 Music 資料表啟用 Amazon DynamoDB 專用 Amazon Kinesis Data Streams 的許可。其不會授予 Music 資料表停用、更新或描述 DynamoDB 專用 Kinesis Data Streams 的許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
```

```
            "Resource": "arn:aws:iam::*:role/aws-
service-role/kinesisreplication.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBKinesisDataStreamsReplication",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":
 "kinesisreplication.dynamodb.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

範例：更新 Amazon DynamoDB 專用 Amazon Kinesis Data Streams

以下 IAM 政策會授予 `Music` 資料表更新 Amazon DynamoDB 專用 Amazon Kinesis Data Streams 的許可。其不會授予 `Music` 資料表啟用、停用或描述 Amazon DynamoDB 專用 Amazon Kinesis Data Streams 的許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

範例：停用 Amazon DynamoDB 專用 Amazon Kinesis Data Streams

以下 IAM 政策會授予 `Music` 資料表停用 Amazon DynamoDB 專用 Amazon Kinesis Data Streams 的許可。其不會授予 `Music` 資料表啟用、更新或描述 Amazon DynamoDB 專用 Amazon Kinesis Data Streams 的許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

範例：根據資源選擇性地為 Amazon DynamoDB 專用 Amazon Kinesis Data Streams 套用許可

下列 IAM 政策會授予 `Music` 資料表啟用及描述 Amazon DynamoDB 專用 Amazon Kinesis Data Streams 的許可，但不會授予 `Orders` 資料表停用 Amazon DynamoDB 專用 Amazon Kinesis Data Streams 的許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination",
                "dynamodb:DescribeKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        },
```

```
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Orders"
        }
    ]
}
```

為 DynamoDB 專用 Kinesis Data Streams 使用服務連結角色

Amazon DynamoDB 的 Amazon Kinesis Data Streams 使用 AWS Identity and Access Management (IAM) 服務連結角色。 DynamoDB 服務連結角色是特殊類型的 IAM 角色，此角色可直接連結到 DynamoDB 專用 Kinesis Data Streams。服務連結角色由適用於 DynamoDB 的 Kinesis Data Streams 預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓 DynamoDB 專用 Kinesis Data Streams 的設定更為簡單，因為您不必手動新增必要的許可。DynamoDB 專用 Kinesis Data Streams 會定義其服務連結角色的許可，除非另有定義，否則僅有 DynamoDB 專用 Kinesis Data Streams 可以擔任其角色。定義的許可包括信任政策和許可政策，並且該許可政策不能連接到任何其他 IAM 實體。

如需關於支援服務連結角色的其他服務的資訊，請參閱可搭配 IAM 運作的AWS 服務，並尋找 Service-Linked Role (服務連結角色) 欄顯示為 Yes (是) 的服務。選擇具有連結的 Yes (是)，以檢視該服務的服務連結角色文件。

DynamoDB 專用 Kinesis Data Streams 的服務連結角色許可

DynamoDB 專用 Kinesis Data Streams 會使用名為 AWSServiceRoleForDynamoDBKinesisDataStreamsReplication 的服務連結角色。服務連結角色的目的是讓 Amazon DynamoDB 代表您管理對 Kinesis Data Streams 項目層級變更的複寫。

`AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` 服務連結角色信任下列服務以擔任角色：

- `kinesisreplication.dynamodb.amazonaws.com`

此角色許可政策允許 DynamoDB 專用 Kinesis Data Streams 對指定資源完成下列動作：

- 動作：`Kinesis stream` 上的 `Put records and describe`

- 動作：`Generate data keys`開啟 AWS KMS 以將資料放置在使用使用者產生 AWS KMS 金鑰加密的 Kinesis 串流上。

如需政策文件的確切內容，請參閱 DynamoDBKinesisReplicationServiceRolePolicy。

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的服務連結角色許可。

為 DynamoDB 專用 Kinesis Data Streams 建立服務連結角色

您不需要手動建立服務連結角色，當您在 AWS 管理主控台、 AWS CLI或 AWS API 中啟用 DynamoDB 的 Kinesis Data Streams 時，DynamoDB 的 Kinesis Data Streams 會為您建立服務連結角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。在啟用 DynamoDB 專用 Kinesis Data Streams 時，DynamoDB 專用 Kinesis Data Streams 會再次為您建立服務連結角色。

為 DynamoDB 專用 Kinesis Data Streams 編輯服務連結角色

DynamoDB 專用 Kinesis Data Streams 不允許您編輯 `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

為 DynamoDB 專用 Kinesis Data Streams 刪除服務連結角色

您也可以使用 IAM 主控台、 AWS CLI 或 AWS API 手動刪除服務連結角色。若要執行此操作，您必須先手動清除服務連結角色的資源，然後才能手動刪除它。

> **Note**
>
> 若 DynamoDB 專用 Kinesis Data Streams 服務在您試圖刪除資源時正在使用該角色，刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

使用 IAM 手動刪除服務連結角色

使用 IAM 主控台 AWS CLI、 或 AWS API 來刪除`AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

DynamoDB Streams 的變更資料擷取

DynamoDB Streams 可擷取任何 DynamoDB 資料表中依時間順序排序的項目層級修改，並將此資訊存放於日誌中長達 24 小時。應用程式可以存取此日誌，並近乎即時地檢視資料項目修改前後的顯示內容。

靜態加密功能會加密 DynamoDB Streams 中的資料。如需詳細資訊，請參閱 DynamoDB 靜態加密。

DynamoDB 串流是 DynamoDB 資料表中項目變更資訊的排序流程。當您在資料表啟用串流時，DynamoDB 會擷取資料表中資料項目的每項修改資訊。

應用程式每次建立、更新或刪除資料表中的項目時，DynamoDB Streams 都會使用已修改項目的主索引鍵屬性寫入串流紀錄。串流紀錄包含 DynamoDB 資料表中單一項目資料修改的相關資訊。您可以設定串流，讓串流紀錄擷取其他資訊，例如修改項目的「之前」與「之後」影像。

DynamoDB Streams 可協助確保下列事項：

- 每個串流紀錄只在串流中出現一次。
- 對於 DynamoDB 資料表中的每個修改項目，串流紀錄的出現順序與項目的實際修改順序相同。

DynamoDB Streams 會近乎即時地寫入串流紀錄，讓您可以建立使用這些串流並根據內容採取動作的應用程式。

主題

- DynamoDB Streams 的端點
- 啟用串流
- 讀取及處理串流
- DynamoDB Streams 和存留時間
- 使用 DynamoDB Streams Kinesis 轉接器處理串流記錄
- DynamoDB Streams 低階 API：Java 範例
- DynamoDB 串流和 AWS Lambda 觸發
- DynamoDB Streams 和 Apache Flink

DynamoDB Streams 的端點

AWS 會維護 DynamoDB 和 DynamoDB Streams 的個別端點。為了使用資料庫資料表與索引，應用程式必須能存取 DynamoDB 端點。為了讀取及處理 DynamoDB Streams 紀錄，應用程式必須能存取同一區域中的 DynamoDB Streams 端點。

DynamoDB Streams 提供兩組端點。這些類別為：

- IPv4-only 端點：具有命名慣例的端點。 `streams.dynamodb.<region>.amazonaws.com`
- 雙堆疊端點：與 IPv4 和 IPv6 相容並遵循`streams-dynamodb.<region>.api.aws`命名慣例的新端點。

> **Note**
>
> 如需 DynamoDB 和 DynamoDB Streams 區域與端點的完整清單，請參閱 AWS 一般參考 中的區域與端點。

AWS SDKs 為 DynamoDB 和 DynamoDB Streams 提供單獨的用戶端。視您需求的不同，應用程式可以存取 DynamoDB 端點、DynamoDB Streams 端點，或同時存取這兩種端點。若要連線到這兩種端點，您的應用程式必須具體化兩個用戶端：一個用於 DynamoDB，另一個用於 DynamoDB Streams。

啟用串流

您可以使用 AWS CLI 或其中一個 AWS SDKs 在新資料表上啟用串流。您也可以在現有的資料表上啟用或停用串流，或變更串流的設定。DynamoDB Streams 會以非同步方式運作，因此若您啟用串流，也不會影響資料表的效能。

管理 DynamoDB Streams 最簡單的方式就是使用 AWS 管理主控台。

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在 DynamoDB 主控台儀表板上，選擇 Tables (資料表)，然後選取現有的資料表。
3. 選擇 Exports and streams (匯出與串流) 索引標籤。
4. 在 DynamoDB 串流詳細資訊區段中，選擇開啟。
5. 在開啟 DynamoDB 串流頁面中，選擇每次修改資料表資料時會寫入串流的資訊：
   - 僅限金鑰屬性：僅已修改項目的金鑰屬性。

- 新映像：在修改後出現的整個項目。
- 舊映像：在修改前出現的整個項目。
- 新舊映像：項目的新舊映像。

當您滿意設定後，請選擇開啟串流。

6. (選用) 若要停用現有的串流，請選擇 DynamoDB 串流詳細資訊下的關閉。

您也可以使用 CreateTable 或 UpdateTable API 操作來啟用或修改串流。StreamSpecification 參數可決定串流的設定方式：

- StreamEnabled：指定是否啟用 (true) 或停用 (false) 資料表的串流。
- StreamViewType：指定每次修改資料表資料時，將會寫入串流的資訊：
  - KEYS_ONLY：僅已修改項目的索引鍵屬性。
  - NEW_IMAGE：在修改後出現的整個項目。
  - OLD_IMAGE：在修改前出現的整個項目。
  - NEW_AND_OLD_IMAGES：項目的新舊映像。

您可以隨時啟用或停用串流。但若嘗試在已有串流的資料表中啟用串流，就會收到 ValidationException。若嘗試在沒有串流的資料表中停用串流，也會收到 ValidationException。

當您將 StreamEnabled 設定為 true 時，DynamoDB 會建立新的串流，並對其指派唯一的串流描述項。如果您停用再重新啟用資料表串流，則會使用不同的串流描述項建立新的串流。

每個串流都可依 Amazon Resource Name (ARN) 唯一識別。以下是名為 TestTable 的 DynamoDB 資料表上串流的 ARN 範例。

```
arn:aws:dynamodb:us-west-2:111122223333:table/TestTable/stream/2015-05-11T21:21:33.291
```

若要判斷資料表的最新串流描述項，請發出 DynamoDB DescribeTable 請求，並尋找回應中的 LatestStreamArn 元素。

> **Note**
>
> 一旦串流設定完畢，就無法再編輯 `StreamViewType`。若您需要變更已設定完畢的串流，就必須停用目前的串流並建立一個新的串流。

讀取及處理串流

若要讀取及處理串流，您的應用程式必須連線到 DynamoDB Streams 端點，並發出 API 請求。

串流由串流紀錄所組成。每個串流紀錄均代表串流所屬之 DynamoDB 資料表中的一項資料修改。每個串流紀錄會獲派一個序號，其反映出紀錄發布至串流的順序。

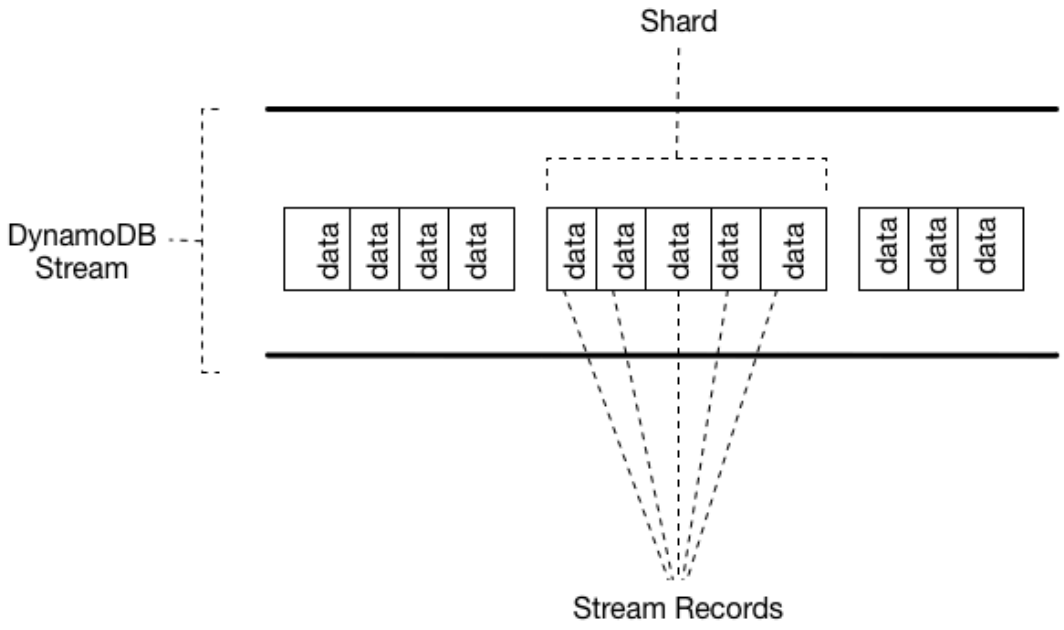
串流紀錄會整理成群組，即碎片。每個碎片皆代表多個串流紀錄的容器，並含有存取及重複處理這些紀錄所需的資訊。碎片內的串流紀錄會在 24 小時後自動移除。

碎片為暫時性：系統會視需要自動建立及刪除這些碎片。任何碎片還可分割成多個新的碎片，這也會自動發生。(父碎片也可能只有一個子碎片。) 碎片可能會為了回應其父資料表上的上層寫入活動而分割，讓應用程式可同時處理多個碎片中的紀錄。

如果您停用串流，所有開啟的碎片都會關閉。串流中的資料在 24 小時內仍可供讀取。

由於碎片具有系屬關係 (父系與子系)，因此應用程式一律必須先處理父碎片，再處理子碎片。這有利於確保串流紀錄也按照正確順序處理。(如果您使用的是 DynamoDB Streams Kinesis 轉接器，則系統會為您處理這個問題。應用程式會依正確順序處理碎片和串流紀錄。除了當應用程式正在執行時分割的碎片，其還會自動處理新的或過期碎片。如需詳細資訊，請參閱 使用 DynamoDB Streams Kinesis 轉接器處理串流記錄。)

下圖說明串流、串流中的碎片及碎片中的串流紀錄之間的關係。



> **Note**
>
> 如果您執行的 `PutItem` 或 `UpdateItem` 操作不會變更項目中的任何資料，則 DynamoDB Streams 不會寫入該操作的串流紀錄。

若要存取串流及處理其中的串流紀錄，您必須執行下列操作：

- 判斷您要存取之串流的唯一 ARN。
- 判斷串流中包含您感興趣之串流紀錄的碎片。
- 存取碎片及擷取您要的串流紀錄。

> **Note**
>
> 最多不得同時從同一個串流碎片讀取 2 個以上的處理程序。每個碎片具有 2 個以上的讀取器會導致限流。

DynamoDB Streams API 提供下列可讓應用程式使用的動作：

- `ListStreams`：傳回目前帳戶及端點的串流描述項清單。您可以只選擇性地請求特定資料表名稱的串流描述項。

- DescribeStream – 傳回串流相關資訊，包含目前串流狀態，其 Amazon Resource Name (ARN)、構成的碎片數量，以及其對應的 DynamoDB 資料表。您可以選擇性使用 ShardFilter 欄位擷取與父碎片相關的現有子碎片。
- GetShardIterator：傳回碎片疊代運算，其描述了碎片內的位置。您可以請求疊代運算提供串流中最舊點、最新點或特定點的存取權。
- GetRecords：傳回指定碎片內的串流紀錄。您必須提供從 GetShardIterator 請求傳回的碎片疊代運算。

如需這些 API 操作的完整說明 (包括範例請求與回應)，請參閱《Amazon DynamoDB Streams API 參考》。

碎片探索

使用兩項強大的方法來探索 DynamoDB 串流中的新碎片。Amazon DynamoDB Streams 使用者可以透過兩種有效的方式追蹤並識別新的碎片：

輪詢整個串流拓撲

使用 DescribeStream API 定期輪詢串流。此步驟會傳回串流中的所有碎片，包括任何已建立的新碎片。您可以透過比較結果，偵測新增的碎片。

探索子碎片

使用 DescribeStream API 搭配 ShardFilter 參數以尋找碎片的子集。透過在請求中指定父碎片，DynamoDB Streams 會傳回其直屬子碎片。此方法適用於僅追蹤碎片譜系，而不掃描整個串流的情況。

從 DynamoDB Streams 取用資料的應用程式，可使用 ShardFilter 參數，有效地從讀取封閉碎片，轉換為讀取其子碎片，避免重複呼叫 DescribeStream API，以擷取和周遊所有封閉及開啟碎片的碎片對應。這有助於在父碎片關閉後快速探索子碎片，提高串流處理應用程式的回應性並更符合成本效益。

這兩項方法都有助於您掌握 DynamoDB Streams 不斷變化的結構，確保您不會錯過重要的資料更新或碎片修改。

DynamoDB Streams 的資料保留限制

DynamoDB Streams 中所有資料的生命週期皆為 24 小時。您可以擷取並分析任何指定資料表過去 24 小時的活動。但超過 24 小時的資料容易隨時受到裁剪 (移除) 的影響。

如果您停用資料表的串流，則串流中的資料在 24 小時內仍可供讀取。過了這段時間後，資料就會過期，且串流紀錄也會自動刪除。沒有手動刪除現有串流的機制。您必須等到保留限制過期 (24 小時)，所有串流紀錄才會被刪除。

DynamoDB Streams 和存留時間

您可以備份或處理存留時間 (TTL) 刪除的項目，方法是在資料表上啟用 Amazon DynamoDB Streams 並處理過期項目的串流紀錄。如需詳細資訊，請參閱讀取及處理串流。

串流紀錄包含使用者身分欄位 `Records[<index>].userIdentity`。

存留時間程序在過期後刪除的項目具有下列欄位：

- `Records[<index>].userIdentity.type`

  `"Service"`
- `Records[<index>].userIdentity.principalId`

  `"dynamodb.amazonaws.com"`

> **Note**
>
> 在全域資料表中使用 TTL 時，執行 TTL 的區域會設定 `userIdentity` 欄位。複寫刪除時，在其他區域中不會設定此欄位。

下列 JSON 顯示單一串流紀錄的相關部分。

```
"Records": [
    {
        ...

        "userIdentity": {
            "type": "Service",
            "principalId": "dynamodb.amazonaws.com"
        }

        ...

    }
]
```

使用 DynamoDB Streams 和 Lambda 封存 TTL 已刪除的項目

合併 DynamoDB 存留時間 (TTL)、DynamoDB Streams 和 AWS Lambda 可協助簡化封存資料、降低 DynamoDB 儲存成本並降低程式碼複雜性。使用 Lambda 做為串流使用者具有許多優勢，最顯著的是與其他使用者 (例如 Kinesis Client Library (KCL)) 相比，成本降低。透過 Lambda 使用事件時，您不會因為 DynamoDB 串流上的 `GetRecords` API 呼叫而需要付費，而且 Lambda 可藉由識別串流事件中的 JSON 模式提供事件篩選功能。透過事件模式內容篩選，您最多可以定義五個不同的篩選條件來控制哪些事件要傳送到 Lambda 進行處理。這有助於減少對 Lambda 函式的調用、簡化程式碼並降低總體成本。

雖然 DynamoDB Streams 包含所有資料修改，例如 `Create`、`Modify` 和 `Remove` 動作，這可能會導致不必要的調用封存 Lambda 函式。例如，假設有一個資料表每小時有 200 萬個資料修改流入串流中，但其中不到 5% 的項目刪除將在 TTL 程序中過期且需要進行封存。搭配 Lambda 事件來源篩選條件，Lambda 函式每小時只會調用 100,000 次。使用事件篩選，您只需為所需的調用付費，而不是在沒有事件篩選的情況下為 200 萬次的調用付費。

事件篩選套用至 Lambda 事件來源映射，它是一種資源，可從選定的事件 (DynamoDB 串流) 中讀取並調用 Lambda 函式。在下圖中，您可以看到 Lambda 函式如何透過串流和事件篩選條件使用存留時間已刪除的項目。

DynamoDB 存留時間事件篩選條件模式

將以下 JSON 新增至事件來源映射篩選條件，可僅對 TTL 刪除的項目調用 Lambda 函式：

```
{
    "Filters": [
        {
            "Pattern": { "userIdentity": { "type": ["Service"], "principalId":
 ["dynamodb.amazonaws.com"] } }
        }
    ]
}
```

建立 AWS Lambda 事件來源映射

使用以下程式碼片段建立已篩選的事件來源映射，您可以將它連接到資料表的 DynamoDB 串流。每個程式碼區塊都包含事件篩選條件模式。

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"]
   }
}
```

使用 DynamoDB Streams Kinesis 轉接器處理串流記錄

建議透過 Amazon Kinesis 轉接器耗用來自 Amazon DynamoDB 的串流。DynamoDB Streams API 類似於 Kinesis Data Streams 的 API 是刻意為之。在這兩種服務中，資料串流由碎片組成，碎片是用於串流紀錄的容器。這兩種服務的 API 都包含 `ListStreams`、`DescribeStream`、`GetShards` 以及 `GetShardIterator` 操作。(雖然這些 DynamoDB Streams 動作與其在 Kinesis Data Streams 中的對應動作類似，但它們並非完全相同。)

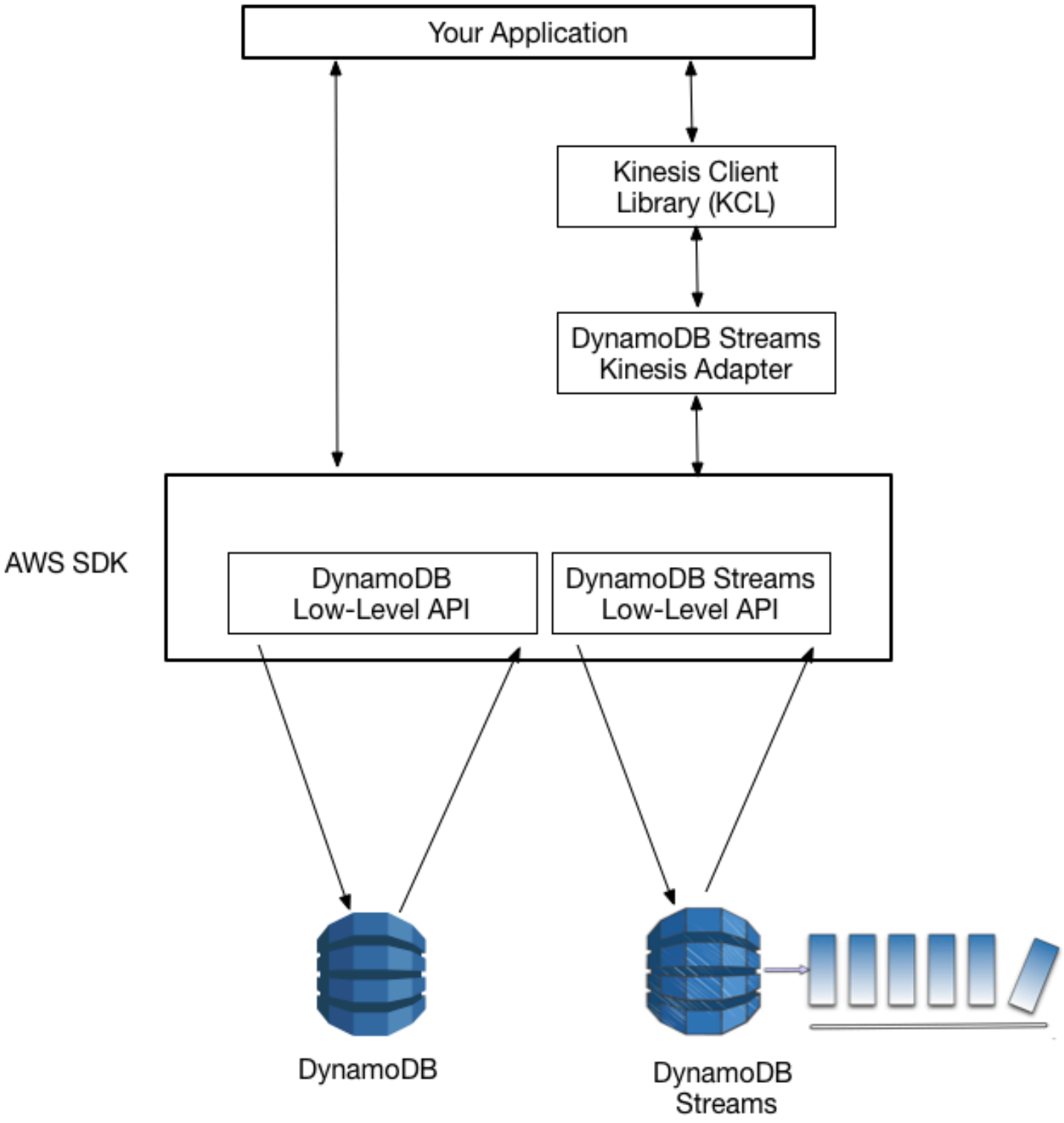
身為 DynamoDB Streams 使用者，您可以使用在 KCL 內找到的設計模式來處理 DynamoDB Streams 碎片和串流紀錄。為此，您可以使用 DynamoDB Streams Kinesis 轉接器。Kinesis 轉接器會實作 Kinesis Data Streams 介面，以便您將 KCL 用於耗用和處理來自 DynamoDB Streams 的紀錄。如需如何設定和安裝 DynamoDB Streams Kinesis Adapter 的指示，請參閱 GitHub 儲存庫。

您可以使用 Kinesis Client Library (KCL) 為 Kinesis Data Streams 撰寫應用程式。KCL 會在低階 Kinesis Data Streams API 上提供有用的抽象，可以簡化程式碼。如需 KCL 的詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的使用 Kinesis Client Library 開發取用者。

DynamoDB 建議使用 KCL 3.x 版搭配適用於 Java 的 AWS SDK v2.x。目前 DynamoDB Streams Kinesis Adapter 1.x 版搭配 適用於 Java 的 AWS SDK 適用於 v1.x 的 AWS SDK，在過渡期間會繼續如預期完全支援整個生命週期，以符合 AWS SDKs和工具維護政策。

> **Note**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您，在 2026 年 1 月 30 日之前，將使用 1.x 版的 KCL 應用程式移轉至最新的 KCL 版本。如需尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis Client Library 頁面。如需最新 Kinesis Client Library 版本的相關詳細資訊，請參閱使用 Kinesis Client Library。如需有關從 KCL 1.x 移轉至 3.x 的詳細資訊，請參閱《從 KCL 1.x 移轉到 KCL 3.x》。

下圖顯示這些程式庫彼此如何互動。



有了 DynamoDB Streams Kinesis 轉接器，您就可以開始針對 KCL 介面進行開發，並將 API 呼叫順暢地導向 DynamoDB Streams 端點。

當應用程式啟動後，其會呼叫 KCL 以將工作者執行個體化。您必須向工作者提供應用程式的組態資訊，例如串流描述項和 AWS 登入資料，以及您提供的記錄處理器類別名稱。當其在紀錄處理器中執行程式碼時，工作者會執行下列任務：

- 連線到串流
- 列舉串流內的碎片
- 檢查並列舉串流中已關閉父碎片的子碎片
- 與其他工作者 (若有) 協調碎片關聯性
- 為其所管理的每個碎片執行個體化記錄處理器
- 從串流提取紀錄
- 在高輸送量期間擴展 GetRecords API 呼叫速率 （ 如果已設定追趕模式 ）
- 將記錄推送至對應的記錄處理器
- 對已處理的記錄執行檢查點作業
- 當工作者執行個體數目變更時，平衡碎片與工作者的關聯
- 當碎片進行分割時，平衡碎片與工作者的關聯

KCL 轉接器支援追趕模式，這是一種自動呼叫速率調整功能，用於處理暫時輸送量增加。當串流處理延遲超過可設定的閾值 （ 預設一分鐘 ） 時，追趕模式會將 GetRecords API 呼叫頻率擴展為可設定的值 （ 預設 3 倍 ），以更快地擷取記錄，然後在延遲下降時恢復正常。在高傳輸量期間，DynamoDB 寫入活動可能會使用預設輪詢率讓消費者負擔過重，這非常寶貴。可以透過`catchupEnabled`組態參數啟用追趕模式 （ 預設 false)。

> **Note**
>
> 如需此處所列 KCL 概念的描述，請參閱 Amazon Kinesis Data Streams 開發人員指南中的使用 Kinesis Client Library 開發取用者。
> 如需搭配 使用串流的詳細資訊， AWS Lambda 請參閱 DynamoDB 串流和 AWS Lambda 觸發

從 KCL 1.x 移轉到 KCL 3.x

概觀

本指南提供如何將取用者應用程式從 KCL 1.x 移轉到 KCL 3.x 的說明。由於 KCL 1.x 和 KCL 3.x 之間的架構差異，移轉需要更新多個元件以確保相容性。

KCL 1.x 使用的類別和介面與 KCL 3.x 不同。您必須先將記錄處理器、記錄處理器處理站及工作者類別移轉至 KCL 3.x 相容格式，並遵循 KCL 1.x 移轉至 KCL 3.x 的移轉步驟。

移轉步驟

主題

- 步驟 1：移轉記錄處理器
- 步驟 2：移轉記錄處理器處理站
- 步驟 3：移轉工作者
- 步驟 4：KCL 3.x 組態概觀和建議
- 步驟 5：從 KCL 2.x 移轉至 KCL 3.x

步驟 1：移轉記錄處理器

以下範例顯示基於 KCL 1.x DynamoDB Streams Kinesis 轉接器所實作的記錄處理器：

```
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();
        }
    }
}
```

移轉 RecordProcessor 類別

1. 將介面從 `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` 和 `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat` 更改為

com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsSha
如下所示：

```
// 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. 更新 initialize 和 processRecords 方法的匯入陳述式：

```
// 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. 將 shutdownRequested 方法取代為以下的新方法：leaseLost、shardEnded 和 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();
        }
    }
```

記錄處理器類別經更新後的版本如下：

```
package com.amazonaws.codesamples;

import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import
 software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor;
import
 software.amazon.dynamodb.streamsadapter.adapter.DynamoDBStreamsKinesisClientRecord;
```

```
import
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsClientRecord;
import software.amazon.awssdk.services.dynamodb.model.Record;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {

    @Override
    public void initialize(InitializationInput initializationInput) {

    }

    @Override
    public void processRecords(DynamoDBStreamsProcessRecordsInput processRecordsInput)
 {
        for (DynamoDBStreamsKinesisClientRecord record: processRecordsInput.records())
            Record ddbRecord = record.getRecord();
            // processing and checkpointing logic for the ddbRecord
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
```

```
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
}
```

> **Note**
>
> DynamoDB Streams Kinesis 轉接器現在使用 SDKv2 記錄模型。在 SDKv2 中，複雜 `AttributeValue` 物件 (BS、NS、M、L、SS) 永遠不會傳回空值。使用 `hasBs()`、`hasNs()`、`hasM()`、`hasL()`、`hasSs()` 方法驗證這些值是否存在。

步驟 2：移轉記錄處理器處理站

記錄處理器處理站負責在取得租用時建立記錄處理器。以下是 KCL 1.x 處理站的範例：

```
package com.amazonaws.codesamples;

import software.amazon.dynamodb.AmazonDynamoDB;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements IRecordProcessorFactory {

    @Override
    public IRecordProcessor createProcessor() {
        return new StreamsRecordProcessor(dynamoDBClient, tableName);
    }
}
```

移轉 `RecordProcessorFactory`

- 將實作的介面從 `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` 更改為 `software.amazon.kinesis.processor.ShardRecordProcessorFactory`，如下所示：

```
// 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() {
```

以下是 3.0 中記錄處理器工廠的範例：

```
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();
    }
}
```

步驟 3：移轉工作者

在 KCL 的版本 3.0，名為排程器的新類別會取代工作者類別。以下是 KCL 1.x 工作者的範例：

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = StreamsWorkerFactory.createDynamoDbStreamsWorker(
        recordProcessorFactory,
        workerConfig,
        adapterClient,
```

```
        amazonDynamoDB,
        amazonCloudWatchClient);
```

移轉至工作者

1. 將 Worker 類別的 import 陳述式變更為 Scheduler 和 ConfigsBuilder 類別的匯入陳述式。

```
// import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.common.ConfigsBuilder;
```

2. 匯入 StreamTracker 並將匯入 StreamsWorkerFactory 變更至 StreamsSchedulerFactory。

```
import software.amazon.kinesis.processor.StreamTracker;
// import software.amazon.dynamodb.streamsadapter.StreamsWorkerFactory;
import software.amazon.dynamodb.streamsadapter.StreamsSchedulerFactory;
```

3. 選擇要啟動應用程式的位置。其可能是 TRIM_HORIZON 或 LATEST。

```
import software.amazon.kinesis.common.InitialPositionInStream;
import software.amazon.kinesis.common.InitialPositionInStreamExtended;
```

4. 建立 StreamTracker 執行個體。

```
StreamTracker streamTracker = StreamsSchedulerFactory.createSingleStreamTracker(
        streamArn,

 InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON)
);
```

5. 建立 AmazonDynamoDBStreamsAdapterClient 物件。

```
import software.amazon.dynamodb.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;

...

AwsCredentialsProvider credentialsProvider = DefaultCredentialsProvider.create();
```

```
AmazonDynamoDBStreamsAdapterClient adapterClient = new
 AmazonDynamoDBStreamsAdapterClient(
        credentialsProvider, awsRegion);
```

6. 建立 ConfigsBuilder 物件。

```
import software.amazon.kinesis.common.ConfigsBuilder;

...
ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamTracker,
                applicationName,
                adapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                UUID.randomUUID().toString(),
                new StreamsRecordProcessorFactory());
```

7. 使用 ConfigsBuilder 建立 Scheduler，如下列範例所示：

```
import java.util.UUID;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;

import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;

...


DynamoDbAsyncClient dynamoClient =
 DynamoDbAsyncClient.builder().region(region).build();
CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();


DynamoDBStreamsPollingConfig pollingConfig = new
 DynamoDBStreamsPollingConfig(adapterClient);
pollingConfig.idleTimeBetweenReadsInMillis(idleTimeBetweenReadsInMillis);
```

```
// Use ConfigsBuilder to configure settings
RetrievalConfig retrievalConfig = configsBuilder.retrievalConfig();
retrievalConfig.retrievalSpecificConfig(pollingConfig);

CoordinatorConfig coordinatorConfig = configsBuilder.coordinatorConfig();
coordinatorConfig.clientVersionConfig(CoordinatorConfig.ClientVersionConfig.CLIENT_VERSION_

Scheduler scheduler = StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
                coordinatorConfig,
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                retrievalConfig,
                adapterClient
        );
```

Important

`CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` 設定可維持 KCL v3 與 KCL v1 之間的 DynamoDB Streams Kinesis Adapter 相容性，而非 KCL v2 與 v3 之間的相容性。

步驟 4：KCL 3.x 組態概觀和建議

如需詳細了解 KCL 3.x 中，在 KCL 1.x 之後推出的相關組態，請參閱 KCL 組態和 KCL 移轉用戶端組態。

Important

我們建議使用 `ConfigsBuilder` 在 KCL 3.x 及更新版本中設定組態，而非直接建立 `checkpointConfig`、`coordinatorConfig`、`leaseManagementConfig`、`metricsConfig`、`pr` 及 `retrievalConfig` 的物件，以避免排程器初始化問題。`ConfigsBuilder` 則提供更靈活且跟更容易維護的 KCL 應用程式設定方式。

KCL 3.x 的更新預設值組態

`billingMode`

KCL 1.x 版中，`billingMode` 的預設值設定為 `PROVISIONED`。不過，KCL 3.x 版 `billingMode` 的預設值為 `PAY_PER_REQUEST` (隨需模式)。我們建議租用資料表使用隨需容量模式，根據用量自動調整容量。如需租用資料表使用佈建容量的指南，請參閱具有佈建容量模式的租用資料表最佳實務。

`idleTimeBetweenReadsInMillis`

在 KCL 1.x 版中，`idleTimeBetweenReadsInMillis` 的預設值設定為 1,000 (或 1 秒)。KCL 3.x 版會將 `idleTimeBetweenReadsInMillis` 的預設值設定為 1,500 (或 1.5 秒)，但 Amazon DynamoDB Streams Kinesis 轉接器會將預設值覆寫為 1,000 (或 1 秒)。

KCL 3.x 中的新組態

`leaseAssignmentIntervalMillis`

此組態會定義新發現碎片開始處理之前的時間間隔，計算方式為 1.5 x `leaseAssignmentIntervalMillis`。如果未明確配置此設定，則時間間隔預設為 1.5 x `failoverTimeMillis`。處理新碎片包含掃描租用資料表，並在租用資料表上查詢全域次要索引 (GSI)。降低 `leaseAssignmentIntervalMillis` 會增加掃描和查詢操作的頻率，進而產生更高的 DynamoDB 成本。我們建議將此值設定為 2000 (或 2 秒)，將處理新碎片的延遲降至最低。

`shardConsumerDispatchPollIntervalMillis`

此組態定義了碎片取用者進行連續輪詢以觸發狀態轉換的間隔時間。在 KCL 1.x 版中，此行為由 `idleTimeInMillis` 參數控制，該參數並未公開為可配置設定。使用 KCL 3.x 版時，我們建議設定此組態，符合您在 KCL 1.x 版設定用於 `idleTimeInMillis` 的值。

步驟 5：從 KCL 2.x 移轉至 KCL 3.x

為了確保順利轉換至最新的 Kinesis Client Library (KCL) 版本及其相容性，請遵循移轉指南從 KCL 2.x 升級到 KCL 3.x 中，步驟 5-8 的指示。

如需常見的 KCL 3.x 疑難排解問題，請參閱 KCL 取用者應用程式疑難排解。

轉返為先前的 KCL 版本

本主題說明如何將取用者應用程式轉返為先前的 KCL 版本。轉返程序包含兩個步驟：

1. 執行 KCL 移轉工具。
2. 重新部署先前 KCL 版本的程式碼。

步驟 1：執行 KCL 移轉工具

當您需要轉返至先前的 KCL 版本時，您必須執行 KCL 移轉工具。此工具會執行兩項重要任務：

- 移除 DynamoDB 租用資料表上，稱為工作者指標資料表和全域次要索引的中繼資料資料表。這些成品由 KCL 3.x 建立，但當您轉返至先前版本時，不需要這些成品。
- 此工具可讓所有工作者在 KCL 1.x 相容模式下執行，並開始使用先前 KCL 版本的負載平衡演算法。如果您使用 KCL 3.x 新負載平衡演算法時遇到問題，此步驟會立即緩解該問題。

> **Important**
>
> DynamoDB 中的協調器狀態資料表必須存在，且不得在移轉、轉返及向前復原過程中刪除。

> **Note**
>
> 請務必讓取用者應用程式中的所有工作者，在指定時間使用相同的負載平衡演算法。KCL 移轉工具可確保 KCL 3.x 取用者應用程式中的所有工作者全數切換到 KCL 1.x 相容模式，以便在應用程式轉返至先前的 KCL 版本期間，所有工作者都執行相同的負載平衡演算法。

您可以在 KCL GitHub 儲存庫的指令碼目錄中，下載 KCL 移轉工具。從具備適當許可的工作者或主機執行指令碼，以寫入協調器狀態資料表、工作者指標資料表和租用資料表。確保已針對 KCL 取用者應用程式設定適當的 IAM 許可。使用指定的命令，每個 KCL 應用程式僅執行一次指令碼：

```
python3 ./KclMigrationTool.py --region region --mode rollback [--
application_name applicationName] [--lease_table_name leaseTableName]
 [--coordinator_state_table_name coordinatorStateTableName] [--
worker_metrics_table_name workerMetricsTableName]
```

Parameters

`--region`

將##取代為 AWS 區域。

`--application_name`

如果您使用 DynamoDB 中繼資料資料表 (租用資料表、協調器狀態資料表和工作者指標資料表) 的預設名稱，則需要此參數。如果您已為這些資料表指定自訂名稱，可以省略此參數。使用您現有應用程式的名稱取代 *applicationName*。如果未提供自訂名稱，工具會使用此名稱衍生預設資料表名稱。

`--lease_table_name`

當您已在 KCL 組態中自訂租用資料表的名稱時，需要此參數。如果您使用的是預設資料表名稱，可以省略此參數。使用您為租用資料表指定的自訂資料表名稱，取代 *leaseTableName*。

`--coordinator_state_table_name`

當您在 KCL 組態中為自訂協調器狀態資料表名稱時，需要此參數。如果您使用的是預設資料表名稱，可以省略此參數。使用您為協調器狀態資料表指定的自訂資料表名稱，取代 *coordinatorStateTableName*。

`--worker_metrics_table_name`

當您在 KCL 組態中自訂工作者指標資料表的名稱時，需要此參數。如果您使用的是預設資料表名稱，可以省略此參數。使用您為工作者指標資料表指定的自訂資料表名稱，取代 *workerMetricsTableName*。

步驟 2：使用先前的 KCL 版本重新部署程式碼

> **Important**
>
> KCL 移轉工具產生的輸出中提及 2.x 版時，應解釋為提及 KCL 1.x 版。執行指令碼不會執行完整的轉返，只會切換到 KCL 1.x 版使用的負載平衡演算法。

執行轉返 KCL 移轉工具後，您會看到以下其中一則訊息：

訊息 1

「轉返已完成。您的應用程式先前執行 2x 相容功能。請使用先前的 KCL 版本部署程式碼，轉返至先前的應用程式二進位檔。」

必要動作：這表示您的工作者在 KCL 1.x 相容模式下執行。將具有先前 KCL 版本的程式碼，重新部署到您的工作者。

訊息 2

「轉返已完成。您的 KCL 應用程式先前執行 3x 功能，並會轉返至 2x 相容功能。如果您在短時間內沒看到緩解措施，請使用先前的 KCL 版本部署程式碼，轉返到先前的應用程式二進位檔。」

必要動作：這表示您的工作者是在 KCL 3.x 模式下執行，而 KCL 移轉工具會將所有工作者切換到 KCL 1.x 相容模式。將具有先前 KCL 版本的程式碼，重新部署到您的工作者。

訊息 3

「應用程式已轉返。所有可刪除的 KCLv3 資源都已清除，以避免產生費用，直到應用程式可以透過移轉向前復原為止。」

必要動作：這表示您的工作者已轉返，以在 KCL 1.x 相容模式下執行。將具有先前 KCL 版本的程式碼，重新部署到您的工作者。

轉返後向前復原至 KCL 3.x

本主題說明如何在轉返後將取用者應用程式向前復原至 KCL 3.x。當您需要向前復原時，您必須完成兩步驟的程序：

1. 執行 KCL 移轉工具。
2. 使用 KCL 3.x 部署程式碼。

步驟 1：執行 KCL 移轉工具

使用以下命令執行 KCL 移轉工具，以向前復原至 KCL 3.x：

```
python3 ./KclMigrationTool.py --region region --mode rollforward [--
application_name applicationName] [--
coordinator_state_table_name coordinatorStateTableName]
```

Parameters

`--region`

將##取代為您的 AWS 區域。

`--application_name`

如果您使用協調器狀態資料表的預設名稱，則需要此參數。如果您已指定協調器狀態資料表的自訂名稱，可以省略此參數。使用您現有應用程式的名稱取代 *`applicationName`*。如果未提供自訂名稱，工具會使用此名稱衍生預設資料表名稱。

`--coordinator_state_table_name`

當您在 KCL 組態中為自訂協調器狀態資料表名稱時，需要此參數。如果您使用的是預設資料表名稱，可以省略此參數。使用您為協調器狀態資料表指定的自訂資料表名稱，取代 *`coordinatorStateTableName`*。

以向前復原模式執行移轉工具後，KCL 會建立 KCL 3.x 所需的下列 DynamoDB 資源：

- 租用資料表上的全域次要索引
- 工作者指標資料表

步驟 2：使用 KCL 3.x 部署程式碼

執行向前復原的 KCL 移轉工具之後，請使用 KCL 3.x 將程式碼部署至工作者。若要完成移轉，請參閱步驟 8：完成移轉。

逐步解說：DynamoDB Streams Kinesis 轉接器

本節會逐步解說使用 Amazon Kinesis Client Library 與 Amazon DynamoDB Streams Kinesis 轉接器的 Java 應用程式。此應用程式示範資料複寫的範例，將一份資料表的寫入活動套用至第二份資料表，讓兩份資料表的內容保持同步。如需來源碼，請參閱「完整程式：DynamoDB Streams Kinesis 轉接器」。

此程式執行下列操作：

1. 建立兩個 DynamoDB 資料表，並命名為 `KCL-Demo-src` 和 `KCL-Demo-dst`。這些資料表各會啟用串流。
2. 在來源資料表中新增、更新與刪除項目，以產生更新活動。這會導致將資料寫入資料表的串流。
3. 從串流讀取紀錄、將其重新建構為 DynamoDB 請求，再將請求套用至目標資料表。
4. 掃描來源與目標資料表，以確定其內容相同。
5. 刪除資料表以清除。

下列章節將說明這些步驟，並在演練結尾顯示完整的應用程式。

主題

- 步驟 1：建立 DynamoDB 資料表
- 步驟 2：在來源資料表中產生更新活動
- 步驟 3：處理串流
- 步驟 4：確定這兩個資料表的內容相同
- 步驟 5：清除
- 完整程式：DynamoDB Streams Kinesis 轉接器

步驟 1：建立 DynamoDB 資料表

第一步是建立兩個 DynamoDB 資料表：一個來源資料表與一個目標資料表。來源資料表串流中的 StreamViewType 是 NEW_IMAGE。這表示每當在此資料表中修改項目時，項目的「修改後」影像就會寫入串流。串流可透過此方法追蹤資料表上的所有寫入活動。

下列範例顯示用來建立這兩份資料表的程式碼。

```
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)
```

步驟 2：在來源資料表中產生更新活動

下一步是在來源資料表上產生一些寫入活動。在此活動進行期間，來源資料表的串流也會近乎即時地更新。

此應用程式會使用呼叫 PutItem、UpdateItem 與 DeleteItem API 操作的方法，來定義小幫手類別，以寫入資料。下列程式碼範例示範如何使用這些方法。

```
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");
```

步驟 3：處理串流

程式現在會開始處理串流。DynamoDB Streams Kinesis 轉接器會作為 KCL 與 DynamoDB Streams 端點之間的透明層，讓程式碼可以充分利用 KCL，而不需要發出低階 DynamoDB Streams 呼叫。此程式會執行下列任務：

- 它會以遵守 KCL 介面定義的方法 (StreamsRecordProcessor、initialize 與 processRecords) 來定義紀錄處理器類別 shutdown。processRecords 方法包含從來源資料表串流讀取與寫入目標資料表所需的邏輯。
- 它會定義紀錄處理器類別的類別處理站 (StreamsRecordProcessorFactory)。這是使用 KCL 之 Java 程式的必要任務。
- 它會執行個體化與類別處理站相關聯的新 KCL Worker。
- 它會在紀錄處理完成時關閉 Worker。

或者，在 Streams KCL Adapter 組態中啟用追趕模式，以便在串流處理延遲超過一分鐘 （預設） 時自動將 GetRecords API 呼叫速率擴展 3 倍 （預設），協助您的串流取用者處理資料表中的高輸送量峰值。

若要進一步了解 KCL 介面定義，請參閱 Amazon Kinesis Data Streams 開發人員指南中的使用 Kinesis Client Library 開發取用者。

下列程式碼範例顯示 StreamsRecordProcessor 中的主迴圈。case 陳述式會根據串流紀錄中顯示的 OperationType 來決定要執行的動作。

```
for (Record record : records) {
    String data = new String(record.getData().array(), Charset.forName("UTF-8"));
    System.out.println(data);
    if (record instanceof RecordAdapter) {
                software.amazon.dynamodb.model.Record streamRecord = ((RecordAdapter)
 record)
                    .getInternalObject();

                switch (streamRecord.getEventName()) {
                    case "INSERT":
                    case "MODIFY":
                        StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getNewImage());
                        break;
                    case "REMOVE":
                        StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getKeys().get("Id").getN());
                }
    }
    checkpointCounter += 1;
    if (checkpointCounter % 10 == 0) {
        try {
            checkpointer.checkpoint();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

步驟 4：確定這兩個資料表的內容相同

此時，來源與目標資料表的內容會同步。此應用程式會對這兩個資料表發出 Scan 請求，以確認其內容事實上相同。

DemoHelper 類別包含呼叫低階 Scan API 的 ScanTable 方法。下列範例示範其使用方法。

```
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!");
}
```

步驟 5：清除

示範已完成，因此應用程式會刪除來源與目標資料表。請參閱以下程式碼範例。即使在刪除資料表後，其串流也會保持可用長達 24 小時，然後就自動刪除。

```
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
```

完整程式：DynamoDB Streams Kinesis 轉接器

以下是執行此逐步解說：DynamoDB Streams Kinesis 轉接器所述之任務的完整 Java 程式。當您執行它時，應該會看到類似如下的輸出。

```
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**
>
> 若要執行此程式，請確定用戶端應用程式可使用政策存取 DynamoDB 和 Amazon CloudWatch。如需詳細資訊，請參閱適用於 DynamoDB 的以身分為基礎的政策。

原始程式碼包含四個`.java`檔案。若要建置此程式，請新增下列相依性，其中包括 Amazon Kinesis Client Library (KCL) 3.x 和適用於 Java v2 的 AWS SDK 做為暫時性相依性：

Maven

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>dynamodb-streams-kinesis-adapter</artifactId>
    <version>2.1.0</version>
</dependency>
```

Gradle

```
implementation 'com.amazonaws:dynamodb-streams-kinesis-adapter:2.1.0'
```

來源檔案為：

- StreamsAdapterDemo.java
- StreamsRecordProcessor.java
- StreamsRecordProcessorFactory.java
- StreamsAdapterDemoHelper.java

StreamsAdapterDemo.java

```
package com.amazonaws.codesamples;

import
 com.amazonaws.services.dynamodbv2.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import com.amazonaws.services.dynamodbv2.streamsadapter.StreamsSchedulerFactory;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.polling.DynamoDBStreamsPollingConfig;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.InitialPositionInStream;
import software.amazon.kinesis.common.InitialPositionInStreamExtended;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;
```

```
import software.amazon.kinesis.processor.StreamTracker;
import software.amazon.kinesis.retrieval.RetrievalConfig;

public class StreamsAdapterDemo {

    private static DynamoDbAsyncClient dynamoDbAsyncClient;
    private static CloudWatchAsyncClient cloudWatchAsyncClient;
    private static AmazonDynamoDBStreamsAdapterClient
 amazonDynamoDbStreamsAdapterClient;

    private static String tablePrefix = "KCL-Demo";
    private static String streamArn;

    private static Region region = Region.US_EAST_1;
    private static AwsCredentialsProvider credentialsProvider =
 DefaultCredentialsProvider.create();

    public static void main( String[] args ) throws Exception {
        System.out.println("Starting demo...");
        dynamoDbAsyncClient = DynamoDbAsyncClient.builder()
                .credentialsProvider(credentialsProvider)
                .region(region)
                .build();
        cloudWatchAsyncClient = CloudWatchAsyncClient.builder()
                .credentialsProvider(credentialsProvider)
                .region(region)
                .build();
        amazonDynamoDbStreamsAdapterClient = new
 AmazonDynamoDBStreamsAdapterClient(credentialsProvider, region);

        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";

        setUpTables();

        StreamTracker streamTracker =
 StreamsSchedulerFactory.createSingleStreamTracker(streamArn,

 InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON));

        ShardRecordProcessorFactory shardRecordProcessorFactory =
                new StreamsAdapterDemoProcessorFactory(dynamoDbAsyncClient, destTable);

        ConfigsBuilder configsBuilder = new ConfigsBuilder(
```

```
                streamTracker,
                "streams-adapter-demo",
                amazonDynamoDbStreamsAdapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                "streams-demo-worker",
                shardRecordProcessorFactory
        );

        DynamoDBStreamsPollingConfig pollingConfig = new
DynamoDBStreamsPollingConfig(amazonDynamoDbStreamsAdapterClient);
        RetrievalConfig retrievalConfig = configsBuilder.retrievalConfig();
        retrievalConfig.retrievalSpecificConfig(pollingConfig);

        System.out.println("Creating scheduler for stream " + streamArn);
        Scheduler scheduler = StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
                configsBuilder.coordinatorConfig(),
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                retrievalConfig,
                amazonDynamoDbStreamsAdapterClient
        );

        System.out.println("Starting scheduler...");
        Thread t = new Thread(scheduler);
        t.start();

        Thread.sleep(250000);

        System.out.println("Stopping scheduler...");
        scheduler.shutdown();
        t.join();

        if (StreamsAdapterDemoHelper.scanTable(dynamoDbAsyncClient, srcTable).items()
                .equals(StreamsAdapterDemoHelper.scanTable(dynamoDbAsyncClient,
destTable).items())) {
            System.out.println("Scan result is equal.");
        } else {
            System.out.println("Tables are different!");
        }
```

```
        System.out.println("Done.");
        cleanupAndExit(0);
    }

    private static void setUpTables() {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        streamArn = StreamsAdapterDemoHelper.createTable(dynamoDbAsyncClient,
srcTable);
        StreamsAdapterDemoHelper.createTable(dynamoDbAsyncClient, destTable);

        awaitTableCreation(srcTable);

        performOps(srcTable);
    }

    private static void awaitTableCreation(String tableName) {
        Integer retries = 0;
        Boolean created = false;
        while (!created && retries < 100) {
            DescribeTableResponse result =
StreamsAdapterDemoHelper.describeTable(dynamoDbAsyncClient, tableName);
            created = result.table().tableStatusAsString().equals("ACTIVE");
            if (created) {
                System.out.println("Table is active.");
                return;
            } else {
                retries++;
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    // do nothing
                }
            }
        }
        System.out.println("Timeout after table creation. Exiting...");
        cleanupAndExit(1);
    }

    private static void performOps(String tableName) {
        StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName, "101",
"test1");
        StreamsAdapterDemoHelper.updateItem(dynamoDbAsyncClient, tableName, "101",
"test2");
```

```
        StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName, "101");
        StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName, "102",
 "demo3");
        StreamsAdapterDemoHelper.updateItem(dynamoDbAsyncClient, tableName, "102",
 "demo4");
        StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName, "102");
    }

    private static void cleanupAndExit(Integer returnValue) {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";

 dynamoDbAsyncClient.deleteTable(DeleteTableRequest.builder().tableName(srcTable).build());

 dynamoDbAsyncClient.deleteTable(DeleteTableRequest.builder().tableName(destTable).build());
        System.exit(returnValue);
    }
}
```

StreamsRecordProcessor.java

```
package com.amazonaws.codesamples;

import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsClientRecord;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.Record;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;

import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {
```

```
    private Integer checkpointCounter;

    private final DynamoDbAsyncClient dynamoDbAsyncClient;
    private final String tableName;

    public StreamsRecordProcessor(DynamoDbAsyncClient dynamoDbAsyncClient, String
tableName) {
        this.dynamoDbAsyncClient = dynamoDbAsyncClient;
        this.tableName = tableName;
    }

    @Override
    public void initialize(InitializationInput initializationInput) {
        this.checkpointCounter = 0;
    }

    @Override
    public void processRecords(DynamoDBStreamsProcessRecordsInput
dynamoDBStreamsProcessRecordsInput) {
        for (DynamoDBStreamsClientRecord record:
dynamoDBStreamsProcessRecordsInput.records()) {
            String data = new String(record.data().array(), StandardCharsets.UTF_8);
            System.out.println(data);
            Record streamRecord = record.getRecord();

            switch (streamRecord.eventName()) {
                case INSERT:
                case MODIFY:
                    StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName,
                            streamRecord.dynamodb().newImage());
                case REMOVE:
                    StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName,
                            streamRecord.dynamodb().keys().get("Id").n());
            }
            checkpointCounter += 1;
            if (checkpointCounter % 10 == 0) {
                try {
                    dynamoDBStreamsProcessRecordsInput.checkpointer().checkpoint();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

```
    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
        System.out.println("Lease Lost");
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            e.printStackTrace();
        }
    }
}
```

StreamsRecordProcessorFactory.java

```
package com.amazonaws.codesamples;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class StreamsAdapterDemoProcessorFactory implements ShardRecordProcessorFactory
 {
    private final String tableName;
    private final DynamoDbAsyncClient dynamoDbAsyncClient;

    public StreamsAdapterDemoProcessorFactory(DynamoDbAsyncClient asyncClient, String
 tableName) {
        this.tableName = tableName;
        this.dynamoDbAsyncClient = asyncClient;
    }
```

```
    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new StreamsRecordProcessor(dynamoDbAsyncClient, tableName);
    }
}
```

StreamsAdapterDemoHelper.java

```
package com.amazonaws.codesamples;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.ResourceInUseException;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import software.amazon.awssdk.services.dynamodb.model.StreamSpecification;
import software.amazon.awssdk.services.dynamodb.model.StreamViewType;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class StreamsAdapterDemoHelper {

    /**
     * @return StreamArn
     */
    public static String createTable(DynamoDbAsyncClient client, String tableName) {
        List<AttributeDefinition> attributeDefinitions = new ArrayList<>();
```

```
        attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName("Id")
                .attributeType("N")
                .build());

        List<KeySchemaElement> keySchema = new ArrayList<>();
        keySchema.add(KeySchemaElement.builder()
                .attributeName("Id")
                .keyType(KeyType.HASH) // Partition key
                .build());

        StreamSpecification streamSpecification = StreamSpecification.builder()
                .streamEnabled(true)
                .streamViewType(StreamViewType.NEW_IMAGE)
                .build();

        CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
                .attributeDefinitions(attributeDefinitions)
                .keySchema(keySchema)
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .streamSpecification(streamSpecification)
                .build();

        try {
            System.out.println("Creating table " + tableName);
            CreateTableResponse result = client.createTable(createTableRequest).join();
            return result.tableDescription().latestStreamArn();
        } catch (Exception e) {
            if (e.getCause() instanceof ResourceInUseException) {
                System.out.println("Table already exists.");
                return describeTable(client, tableName).table().latestStreamArn();
            }
            throw e;
        }
    }

    public static DescribeTableResponse describeTable(DynamoDbAsyncClient client,
String tableName) {
        return client.describeTable(DescribeTableRequest.builder()
                        .tableName(tableName)
                        .build())
                .join();
    }
```

```
   public static ScanResponse scanTable(DynamoDbAsyncClient dynamoDbClient, String
tableName) {
       return dynamoDbClient.scan(ScanRequest.builder()
                       .tableName(tableName)
                       .build())
               .join();
   }

   public static void putItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
String id, String val) {
       Map<String, AttributeValue> item = new HashMap<>();
       item.put("Id", AttributeValue.builder().n(id).build());
       item.put("attribute-1", AttributeValue.builder().s(val).build());

       putItem(dynamoDbClient, tableName, item);
   }

   public static void putItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
                              Map<String, AttributeValue> items) {
       PutItemRequest putItemRequest = PutItemRequest.builder()
               .tableName(tableName)
               .item(items)
               .build();
       dynamoDbClient.putItem(putItemRequest).join();
   }

   public static void updateItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
String id, String val) {
       Map<String, AttributeValue> key = new HashMap<>();
       key.put("Id", AttributeValue.builder().n(id).build());

       Map<String, String> expressionAttributeNames = new HashMap<>();
       expressionAttributeNames.put("#attr2", "attribute-2");

       Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
       expressionAttributeValues.put(":val", AttributeValue.builder().s(val).build());

       UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
               .tableName(tableName)
               .key(key)
               .updateExpression("SET #attr2 = :val")
               .expressionAttributeNames(expressionAttributeNames)
               .expressionAttributeValues(expressionAttributeValues)
```

```
                .build();

        dynamoDbClient.updateItem(updateItemRequest).join();
    }

    public static void deleteItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
 String id) {
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("Id", AttributeValue.builder().n(id).build());

        DeleteItemRequest deleteItemRequest = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .build();
        dynamoDbClient.deleteItem(deleteItemRequest).join();
    }
}
```

DynamoDB Streams 低階 API：Java 範例

> **Note**
>
> 此頁面上的程式碼並不詳盡，並且無法應對使用 Amazon DynamoDB Streams 的所有案例。從 DynamoDB 使用串流紀錄的建議方式，是透過使用 Kinesis Client Library (KCL) 的 Amazon Kinesis 轉接器，如 使用 DynamoDB Streams Kinesis 轉接器處理串流記錄 中所述。

本節包含顯示作用中 DynamoDB Streams 的 Java 程式。此程式執行下列操作：

1. 建立啟用串流的 DynamoDB 資料表。
2. 說明此資料表的串流設定。
3. 修改資料表中的資料。
4. 描述串流中的碎片。
5. 讀取碎片中的串流紀錄。
6. 擷取子碎片並持續讀取記錄。
7. 清除。

當您執行此程式時，會看到與下面類似的輸出。

```
Testing Streams Demo
Creating an Amazon DynamoDB table TestTableForStreams with a simple primary key: Id
Waiting for TestTableForStreams to be created...
Current stream ARN for TestTableForStreams: arn:aws:dynamodb:us-
east-2:123456789012:table/TestTableForStreams/stream/2018-03-20T16:49:55.208
Stream enabled: true
Update view type: NEW_AND_OLD_IMAGES

Performing write activities on TestTableForStreams
Processing item 1 of 100
Processing item 2 of 100
Processing item 3 of 100
...
Processing item 100 of 100
Shard: {ShardId: shardId-1234567890-...,SequenceNumberRange: {StartingSequenceNumber:
 10000257248679750 8907,},}
    Shard iterator: EjYFEkX2a26eVTWe...
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, NewImage={Message=AttributeValue(S=New Item!),
 Id=AttributeValue(S=4)}, SequenceNumber=200000158404754583 3909, SizeBytes=22,
 StreamViewType=NEW_AND_OLD_IMAGES)
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, NewImage={Message=AttributeValue(S=This is an updated
 item), Id=AttributeValue(S=4)}, OldImage={Message=AttributeValue(S=New Item!),
 Id=AttributeValue(S=4)}, SequenceNumber=210000360486976789 2701, SizeBytes=55,
 StreamViewType=NEW_AND_OLD_IMAGES)
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, OldImage={Message=AttributeValue(S=This is an updated
 item), Id=AttributeValue(S=4)}, SequenceNumber=220000109977111289 8434, SizeBytes=36,
 StreamViewType=NEW_AND_OLD_IMAGES)
...
Deleting the table...
Table StreamsDemoTable deleted.
Demo complete
```

Example範例

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeStreamRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeStreamResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetRecordsRequest;
import software.amazon.awssdk.services.dynamodb.model.GetRecordsResponse;
import software.amazon.awssdk.services.dynamodb.model.GetShardIteratorRequest;
import software.amazon.awssdk.services.dynamodb.model.GetShardIteratorResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.Record;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.Shard;
import software.amazon.awssdk.services.dynamodb.model.ShardFilter;
import software.amazon.awssdk.services.dynamodb.model.ShardFilterType;
import software.amazon.awssdk.services.dynamodb.model.ShardIteratorType;
import software.amazon.awssdk.services.dynamodb.model.StreamSpecification;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.streams.DynamoDbStreamsClient;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

public class StreamsLowLevelDemo {


    public static void main(String[] args) {
        final String usage = "Testing Streams Demo";
        try {
            System.out.println(usage);

            String tableName = "StreamsDemoTable";
            String key = "Id";
```

```
            System.out.println("Creating an Amazon DynamoDB table " + tableName + "
with a simple primary key: " + key);
            Region region = Region.US_WEST_2;
            DynamoDbClient ddb = DynamoDbClient.builder()
                    .region(region)
                    .build();

            DynamoDbStreamsClient ddbStreams = DynamoDbStreamsClient.builder()
                    .region(region)
                    .build();
            DescribeTableRequest describeTableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();
            TableDescription tableDescription = null;
            try{
                tableDescription = ddb.describeTable(describeTableRequest).table();
            }catch (Exception e){
                System.out.println("Table " + tableName + " does not exist.");
                tableDescription = createTable(ddb, tableName, key);
            }

            // Print the stream settings for the table
            String streamArn = tableDescription.latestStreamArn();

            StreamSpecification streamSpec = tableDescription.streamSpecification();
            System.out.println("Current stream ARN for " + tableDescription.tableName()
+ ": " +
                   streamArn);
            System.out.println("Stream enabled: " + streamSpec.streamEnabled());
            System.out.println("Update view type: " + streamSpec.streamViewType());
            System.out.println();
            // Generate write activity in the table
            System.out.println("Performing write activities on " + tableName);
            int maxItemCount = 100;
            for (Integer i = 1; i <= maxItemCount; i++) {
                System.out.println("Processing item " + i + " of " + maxItemCount);
                // Write a new item
                putItemInTable(key, i, tableName, ddb);
                // Update the item
                updateItemInTable(key, i, tableName, ddb);
                // Delete the item
                deleteDynamoDBItem(key, i, tableName, ddb);
            }
```

```
            // Process Stream
            processStream(streamArn, maxItemCount, ddb, ddbStreams, tableName);

            // Delete the table
            System.out.println("Deleting the table...");
            DeleteTableRequest deleteTableRequest = DeleteTableRequest.builder()
                    .tableName(tableName)
                    .build();
            ddb.deleteTable(deleteTableRequest);
            System.out.println("Table " + tableName + " deleted.");
            System.out.println("Demo complete");
            ddb.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    private static void processStream(String streamArn, int maxItemCount,
DynamoDbClient ddb, DynamoDbStreamsClient ddbStreams, String tableName) {
        // Get all the shard IDs from the stream. Note that DescribeStream returns
        // the shard IDs one page at a time.
        String lastEvaluatedShardId = null;
        do {
            DescribeStreamRequest describeStreamRequest =
DescribeStreamRequest.builder()
                    .streamArn(streamArn)
                    .exclusiveStartShardId(lastEvaluatedShardId).build();
            DescribeStreamResponse describeStreamResponse =
ddbStreams.describeStream(describeStreamRequest);

            List<Shard> shards = describeStreamResponse.streamDescription().shards();

            // Process each shard on this page

            fetchShardsAndReadRecords(streamArn, maxItemCount, ddbStreams, shards);

            // If LastEvaluatedShardId is set, then there is
            // at least one more page of shard IDs to retrieve
            lastEvaluatedShardId =
describeStreamResponse.streamDescription().lastEvaluatedShardId();

        } while (lastEvaluatedShardId != null);

    }
```

```
   private static void fetchShardsAndReadRecords(String streamArn, int maxItemCount,
DynamoDbStreamsClient ddbStreams, List<Shard> shards) {
       for (Shard shard : shards) {
           String shardId = shard.shardId();
           System.out.println("Shard: " + shard);

           // Get an iterator for the current shard
           GetShardIteratorRequest shardIteratorRequest =
GetShardIteratorRequest.builder()
                   .streamArn(streamArn).shardId(shardId)
                   .shardIteratorType(ShardIteratorType.TRIM_HORIZON).build();

           GetShardIteratorResponse getShardIteratorResult =
ddbStreams.getShardIterator(shardIteratorRequest);

           String currentShardIter = getShardIteratorResult.shardIterator();

           // Shard iterator is not null until the Shard is sealed (marked as
READ_ONLY).
           // To prevent running the loop until the Shard is sealed, we process only
the
           // items that were written into DynamoDB and then exit.
           int processedRecordCount = 0;
           while (currentShardIter != null && processedRecordCount < maxItemCount) {
               // Use the shard iterator to read the stream records
               GetRecordsRequest getRecordsRequest = GetRecordsRequest.builder()
                       .shardIterator(currentShardIter).build();
               GetRecordsResponse getRecordsResult =
ddbStreams.getRecords(getRecordsRequest);
               List<Record> records = getRecordsResult.records();
               for (Record record : records) {
                   System.out.println("        " + record.dynamodb());
               }
               processedRecordCount += records.size();
               currentShardIter = getRecordsResult.nextShardIterator();
           }
           if (currentShardIter == null){
               System.out.println("Shard has been fully processed. Shard iterator is
null.");
               System.out.println("Fetch the child shard to continue processing
instead of bulk fetching all shards");
               DescribeStreamRequest describeStreamRequestForChildShards =
DescribeStreamRequest.builder()
```

```
                        .streamArn(streamArn)
                        .shardFilter(ShardFilter.builder()
                                .type(ShardFilterType.CHILD_SHARDS)
                                .shardId(shardId).build())
                        .build();
                DescribeStreamResponse describeStreamResponseChildShards =
ddbStreams.describeStream(describeStreamRequestForChildShards);
                fetchShardsAndReadRecords(streamArn, maxItemCount, ddbStreams,
describeStreamResponseChildShards.streamDescription().shards());
            }
        }
    }

    private static void putItemInTable(String key, Integer i, String tableName,
DynamoDbClient ddb) {
        Map<String, AttributeValue> item = new HashMap<>();
        item.put(key, AttributeValue.builder()
                .s(i.toString())
                .build());
        item.put("Message", AttributeValue.builder()
                .s("New Item!")
                .build());
        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(item)
                .build();
        ddb.putItem(request);
    }

    private static void updateItemInTable(String key, Integer i, String tableName,
DynamoDbClient ddb) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(i.toString())
                .build());


        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put("Message", AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s("This is an updated item").build())
                .action(AttributeAction.PUT)
                .build());
```

```
        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();
        ddb.updateItem(request);
    }

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

        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(keyToGet)
                .build();
        ddb.deleteItem(deleteReq);
    }

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

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

            System.out.println("Waiting for " + tableName + " to be created...");

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

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



}
```

DynamoDB 串流和 AWS Lambda 觸發

Amazon DynamoDB 與 整合， AWS Lambda 因此您可以建立觸發，也就是自動回應 DynamoDB Streams 中事件的程式碼片段。您可以利用觸發條件建立對 DynamoDB 資料表資料修改做出反應的應用程式。

主題

- 教學課程 #1：使用篩選條件來處理 Amazon DynamoDB 的所有事件， AWS Lambda 並使用 AWS CLI
- 教學課程 #2：使用篩選條件來處理 DynamoDB 和 Lambda 的所有事件
- 搭配 Lambda 使用 DynamoDB Streams 的最佳實務

如果您在資料表上啟用 DynamoDB Streams，您可以將串流 Amazon Resource Name (ARN) 與您寫入的 AWS Lambda 函數建立關聯。然後，即可將該 DynamoDB 資料表的所有變動動作擷取為串流上

的項目。例如，您可以設定觸發條件，以便在修改資料表中的項目時，新記錄會立即顯示在該資料表的串流中。

> **Note**
>
> 如果您為單一 DynamoDB 串流訂閱兩個以上的 Lambda 函式，可能會發生讀取限流。

此 AWS Lambda 服務會以每秒四次的速度輪詢串流以偵測新記錄。當有新的串流記錄可用時，系統會同步調用 Lambda 函式。對於相同的 DynamoDB 串流，最多可以訂閱兩個 Lambda 函式。如果您為相同 DynamoDB 串流訂閱兩個以上的 Lambda 函式，可能會發生讀取限流。

Lambda 函式可以傳送通知、啟動工作流程，或執行您指定的其他許多動作。您可以撰寫 Lambda 函式僅將每筆串流紀錄複製到耐久性儲存，例如 Amazon S3 File Gateway (Amazon S3)，並在您的資料表中建立永久的寫入活動稽核軌跡。或者，假設您有一個會寫入 `GameScores` 表的手機遊戲應用程式。每當 `TopScore` 資料表的 `GameScores` 屬性更新時，對應的串流紀錄就會寫入資料表串流。然後，這個事件就會觸發在社交媒體網路張貼賀電的 Lambda 函式。(此函數也可以編寫為忽略所有非更新至 `GameScores` 或不修改 `TopScore` 屬性的串流紀錄)。

如果函數傳回錯誤，Lambda 會不斷重試批次直到處理成功或資料過期。您還可以設定 Lambda，使用較小批次重試、限制重試次數、在記錄太舊時捨棄，以及其他選項。

作為效能最佳實務，Lambda 函式必須為短期函數。為了避免產生不必要的處理延遲，此函數也不應執行複雜的邏輯。尤其對於高速串流而言，相較於同步長時間執行的 Lambda，觸發非同步後續處理 Step Function 工作流程是較佳的做法。

您可以在 DynamoDB 串流上設定資源型政策，以授予 Lambda 函數的跨帳戶讀取存取權， AWS 藉此跨帳戶使用 Lambda 觸發。若要進一步了解如何設定串流以允許跨帳戶存取，請參閱《DynamoDB 開發人員指南》中的使用跨帳戶 AWS Lambda 函數共用存取權。

如需 的詳細資訊 AWS Lambda，請參閱 AWS Lambda 開發人員指南。

教學課程 #1：使用篩選條件來處理 Amazon DynamoDB 的所有事件， AWS Lambda 並使用 AWS CLI

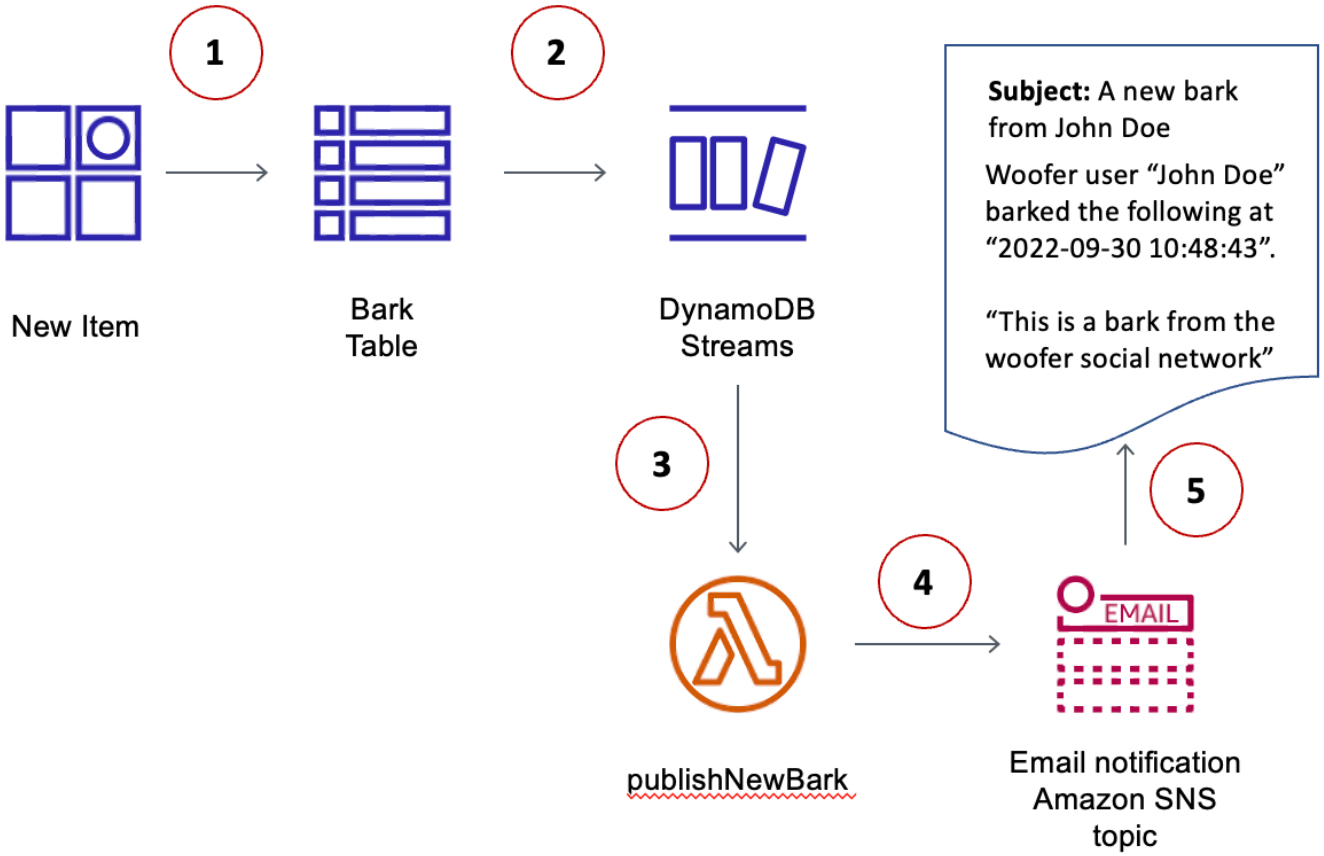
在本教學課程中，您將建立 AWS Lambda 觸發程序來處理來自 DynamoDB 資料表的串流。

主題

- 步驟 1：建立啟用串流的 DynamoDB 資料表

- 步驟 2：建立 Lambda 執行角色
- 步驟 3：建立 Amazon SNS 主題
- 步驟 4：建立並測試 Lambda 函式
- 步驟 5：建立並測試觸發器

本教學案例為 Woofer，簡易的社交網路。Woofer 使用者使用互相傳送的 barks (簡短的文字訊息) 進行通訊。下圖顯示此應用程式的元件和工作流程。



1. 使用者將某個項目寫入 DynamoDB 資料表 (BarkTable)。資料表中的每個項目代表一個 bark。
2. 寫入新串流紀錄即反映新的項目已新增至 BarkTable。
3. 新的串流記錄會觸發 AWS Lambda 函數 (publishNewBark)。
4. 如果該串流紀錄指出新的項目已新增至 BarkTable，則 Lambda 函式會從串流紀錄讀取資料，將訊息發布至 Amazon Simple Notification Service (Amazon SNS) 中的主題。
5. Amazon SNS 主題的訂閱用戶會收到此訊息。(在此教學中，唯一的訂閱用戶是電子郵件地址)。

開始之前

本教學課程使用 AWS Command Line Interface AWS CLI。若您尚未執行此作業，請遵循《AWS Command Line Interface 使用者指南》中的說明安裝及設定 AWS CLI。

步驟 1：建立啟用串流的 DynamoDB 資料表

在此步驟中，您會建立 DynamoDB 資料表 (BarkTable) 存放 Woofer 使用者的所有 bark。主索引鍵包含 Username (分割區索引鍵) 和 Timestamp (排序索引鍵)。這些屬性的類型皆為字串。

BarkTable 已啟用串流。在本教學課程稍後，您可以透過將 AWS Lambda 函數與串流建立關聯來建立觸發。

1. 輸入下列命令建立資料表。

```
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. 在輸出中，尋找 LatestStreamArn。

```
...
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

記下 *region* 和 *accountID*，因為您在本教學的其他步驟中會需要它們。

步驟 2：建立 Lambda 執行角色

在此步驟中，您會建立 AWS Identity and Access Management (IAM) 角色 (WooferLambdaRole) 並為其指派許可。這個角色會由您在 步驟 4：建立並測試 Lambda 函式 中建立的 Lambda 函式所使用。

您也要為該角色建立政策。此政策包含 Lambda 函式在執行時期所需要的全部許可。

1. 使用下列內容建立名為 trust-relationship.json 的檔案。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Principal": {
         "Service": "lambda.amazonaws.com"
       },
       "Action": "sts:AssumeRole"
     }
   ]
 }
```

2. 輸入下列命令建立 WooferLambdaRole。

```
aws iam create-role --role-name WooferLambdaRole \
    --path "/service-role/" \
    --assume-role-policy-document file://trust-relationship.json
```

3. 使用下列內容建立名為 role-policy.json 的檔案。（將 *region*和 取代*accountID*為您的 AWS 區域和帳戶 ID。）

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/
BarkTable/stream/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

政策有四個陳述式可讓 WooferLambdaRole 執行下列操作：

- 執行 Lambda 函式 (publishNewBark)。您稍後要在本教學中建立此函數。
- 存取 Amazon CloudWatch Logs。Lambda 函式會在執行時期將診斷寫入 CloudWatch Logs。
- 從 BarkTable 的 DynamoDB 串流讀取資料。
- 將訊息發布到 Amazon SNS。

4. 輸入下列命令將此政策連接至 WooferLambdaRole。

```
aws iam put-role-policy --role-name WooferLambdaRole \
    --policy-name WooferLambdaRolePolicy \
    --policy-document file://role-policy.json
```

步驟 3：建立 Amazon SNS 主題

在此步驟中，您要建立 Amazon SNS 主題 (wooferTopic)，並用電子郵件地址訂閱此主題。您的 Lambda 函式會使用此主題發布 Woofer 使用者的新 bark。

1. 輸入以下命令來建立新的 Amazon SNS 主題。

```
aws sns create-topic --name wooferTopic
```

2. 輸入下列命令使用電子郵件地址訂閱 wooferTopic。(將 *region* 和 *accountID* 更換為 AWS 區域和帳戶 ID，將 *example@example.com* 更換為有效的電子郵件地址。)

```
aws sns subscribe \
    --topic-arn arn:aws:sns:region:accountID:wooferTopic \
    --protocol email \
    --notification-endpoint example@example.com
```

3. Amazon SNS 會將確認訊息傳送至電子郵件地址。選擇該郵件中的 Confirm subscription (確認訂閱) 連結，完成訂閱程序。

步驟 4：建立並測試 Lambda 函式

在此步驟中，您會建立 AWS Lambda 函數 (publishNewBark) 來處理來自 的串流記錄BarkTable。

publishNewBark 函數只處理對應至 BarkTable 新項目的串流事件。此函數會讀取此種事件的資料，然後調用 Amazon SNS 來進行發布。

1. 使用下列內容建立名為 publishNewBark.js 的檔案。將 *region*和 取代*accountID*為您的 AWS 區域和帳戶 ID。

```
'use strict';
var AWS = require("aws-sdk");
var sns = new AWS.SNS();

exports.handler = (event, context, callback) => {

    event.Records.forEach((record) => {
        console.log('Stream record: ', JSON.stringify(record, null, 2));

        if (record.eventName == 'INSERT') {
            var who = JSON.stringify(record.dynamodb.NewImage.Username.S);
            var when = JSON.stringify(record.dynamodb.NewImage.Timestamp.S);
            var what = JSON.stringify(record.dynamodb.NewImage.Message.S);
            var params = {
                Subject: 'A new bark from ' + who,
                Message: 'Woofer user ' + who + ' barked the following at ' + when
 + ':\n\n ' + what,
```

```
                TopicArn: 'arn:aws:sns:region:accountID:wooferTopic'
            };
            sns.publish(params, function(err, data) {
                if (err) {
                    console.error("Unable to send message. Error JSON:",
 JSON.stringify(err, null, 2));
                } else {
                    console.log("Results from sending message: ",
 JSON.stringify(data, null, 2));
                }
            });
        }
    });
    callback(null, `Successfully processed ${event.Records.length} records.`);
};
```

2. 建立 zip 檔以包含 publishNewBark.js。如果您有 zip 命令列公用程式，即可輸入下列命令執行此作業。

```
zip publishNewBark.zip publishNewBark.js
```

3. 當您建立 Lambda 函式時，您要為自己在 步驟 2：建立 Lambda 執行角色 中建立的 WooferLambdaRole 指定 Amazon Resource Name (ARN)。輸入下列命令以擷取此 ARN。

```
aws iam get-role --role-name WooferLambdaRole
```

在輸出中，尋找 WooferLambdaRole 的 ARN。

```
...
"Arn": "arn:aws:iam::region:role/service-role/WooferLambdaRole"
...
```

輸入下列命令建立 Lambda 函式。將 *roleARN* 取代為 WooferLambdaRole 的 ARN。

```
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. 現在，測試 publishNewBark 確認能否作用。若要執行此作業，您要提供類似 DynamoDB Streams 中真實紀錄的輸入。

   使用下列內容建立名為 payload.json 的檔案。將 *region* 和 *accountID* 更換為 AWS 區域 和帳戶 ID。

```
{
    "Records": [
        {
            "eventID": "7de3041dd709b024af6f29e4fa13d34c",
            "eventName": "INSERT",
            "eventVersion": "1.1",
            "eventSource": "aws:dynamodb",
            "awsRegion": "region",
            "dynamodb": {
                "ApproximateCreationDateTime": 1479499740,
                "Keys": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "NewImage": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Message": {
                        "S": "This is a bark from the Woofer social network"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "SequenceNumber": "13021600000000001596893679",
                "SizeBytes": 112,
                "StreamViewType": "NEW_IMAGE"
            },
```

```
            "eventSourceARN": "arn:aws:dynamodb:region:account ID:table/BarkTable/
stream/2016-11-16T20:42:48.104"
        }
    ]
}
```

輸入下列命令測試 publishNewBark 函數。

```
aws lambda invoke --function-name publishNewBark --payload file://payload.json --
cli-binary-format raw-in-base64-out output.txt
```

如果測試成功，您就會看到以下輸出。

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

此外，output.txt 檔案還會包含以下文字。

```
"Successfully processed 1 records."
```

您也會在幾分鐘內收到新的電子郵件訊息。

> **Note**
>
> AWS Lambda 會將診斷資訊寫入 Amazon CloudWatch Logs。如果 Lambda 函式發生問題，您可以使用這些診斷進行疑難排解：
>
> 1. 在 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
> 2. 在導覽窗格中，選擇 Logs (日誌)。
> 3. 選擇下列的日誌群組：/aws/lambda/publishNewBark
> 4. 選擇最新的日誌串流，檢視函數的輸出 (和錯誤)。

步驟 5：建立並測試觸發器

在 步驟 4：建立並測試 Lambda 函式 中，您已測試過 Lambda 函式，確保其能正確執行。在此步驟中，透過將 Lambda 函式 (publishNewBark) 與事件來源 (BarkTable 串流) 建立關聯來建立觸發條件。

1. 當您建立觸發時，您需要指定 BarkTable 串流的 ARN。輸入下列命令以擷取此 ARN。

   ```
   aws dynamodb describe-table --table-name BarkTable
   ```

   在輸出中，尋找 LatestStreamArn。

   ```
   ...
    "LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
   stream/timestamp
   ...
   ```

2. 輸入以下命令來建立觸發。將 *streamARN* 更換為實際的串流 ARN。

   ```
   aws lambda create-event-source-mapping \
       --region region \
       --function-name publishNewBark \
       --event-source streamARN  \
       --batch-size 1 \
       --starting-position TRIM_HORIZON
   ```

3. 測試觸發。輸入下列命令將項目新增至 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"}
   ```

   您應該會在幾分鐘內收到新的電子郵件訊息。

4. 開啟 DynamoDB 主控台，在 BarkTable 中多新增幾個項目。您必須指定 Username 和 Timestamp 屬性的值。(您也應該指定 Message 的值，雖然它不是必要的)。您應該會因為每個新增至 BarkTable 的項目而收到新的電子郵件訊息。

   Lambda 函式只處理您新增至 BarkTable 的新項目。如果您要更新或刪除資料表中的項目，此函數不會執行任何作業。

> **Note**
>
> AWS Lambda 會將診斷資訊寫入 Amazon CloudWatch Logs。如果 Lambda 函式發生問題，您可以使用這些診斷進行疑難排解。
>
> 1. 在 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
> 2. 在導覽窗格中，選擇 Logs (日誌)。
> 3. 選擇下列的日誌群組：`/aws/lambda/publishNewBark`
> 4. 選擇最新的日誌串流，檢視函數的輸出 (和錯誤)。

教學課程 #2：使用篩選條件來處理 DynamoDB 和 Lambda 的所有事件

在本教學課程中，您將建立 AWS Lambda 觸發程序，以僅處理來自 DynamoDB 資料表之串流中的某些事件。

主題

- 全部整合 - CloudFormation
- 整合練習 - CDK

透過 Lambda 事件篩選，您可以使用篩選條件表達式來控制 Lambda 要傳送哪些事件給函數進行處理。對於每個 DynamoDB 串流，最多可以設定 5 個不同的篩選條件。如果您使用批次處理間隔，Lambda 會將篩選條件標準套用至每個新事件，以查看是否應將其納入目前的批次中。

篩選條件會透過稱為 `FilterCriteria` 的結構進行套用。`FilterCriteria` 的 3 項主要屬性是 `metadata properties`、`data properties` 和 `filter patterns`。

以下是 DynamoDB Streams 事件的範例結構：

```
{
  "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` 是事件物件的欄位。若是 DynamoDB Streams，`metadata properties` 是類似 dynamodb 或 `eventName` 的欄位。

`data properties` 是事件主體的欄位。若要篩選 `data properties`，請務必將資料屬性包含在適當金鑰內的 `FilterCriteria` 之中。對於 DynamoDB 事件來源，資料金鑰為 `NewImage` 或 `OldImage`。

最後，篩選條件規則將會定義您要套用至特定屬性的篩選條件表達式。以下是一些範例：

比較運算子	範例	規則語法 (部分)
Null	產品類型為 Null	`{ "product_type": { "S": null } }`
空白	產品名稱為空	`{ "product_name": { "S": [ ""] } }`
Equals	州等於佛羅里達州	`{ "state": { "S": ["FL"] } }`

比較運算子	範例	規則語法 (部分)
及	產品原產州等於佛羅里達州，且產品類別為巧克力	`{ "state": { "S": ["FL"] } , "category": { "S": [ "CHOCOLATE"] } }`
或	產品原產州是佛羅里達州或加州	`{ "state": { "S": ["FL","CA"] } }`
Not	產品原產州不是佛羅里達州	`{"state": {"S": [{"anything-but": ["FL"]}]}}`
存在	自製產品存在	`{"homemade": {"S": [{"exists": true}]}}`
不存在	自製產品不存在	`{"homemade": {"S": [{"exists": false}]}}`
Begins with	PK 以 COMPANY 開頭	`{"PK": {"S": [{"prefix": "COMPANY"}]}}`

您可以為一個 Lambda 函式最多指定 5 個事件篩選模式。請注意，這 5 個事件中的每一個都將評估為邏輯 OR。因此，如果您設定兩個分別名為 `Filter_One` 和 `Filter_Two` 的篩選條件，Lambda 函式將會執行 `Filter_One` OR `Filter_Two`。

> **Note**
>
> Lambda 事件篩選頁面中有部分選項可用於篩選和比較數值，但不適用於 DynamoDB 篩選條件事件，因為 DynamoDB 中的數字會儲存為字串。例如 `"quantity": { "N": "50" }`，從 `"N"` 屬性可以得知這是一個數字。

全部整合 - CloudFormation

若要在實務中顯示事件篩選功能，請參考以下 CloudFormation 範本的範例。此範本將產生一個簡單的 DynamoDB 資料表，其中含有分割區索引鍵 PK 和排序索引鍵 SK，且已啟用 Amazon DynamoDB Streams。它會建立一個 Lambda 函式和一個簡單的 Lambda 執行角色，以允許將日誌寫入 Amazon CloudWatch，並從 Amazon DynamoDB Streams 讀取事件。它也會在 DynamoDB Streams 和 Lambda 函式之間新增事件來源映射，以便每次在 Amazon DynamoDB Streams 中發生事件時，都可以執行該函數。

```
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
```

部署此雲端編組範本之後，即可插入下列 Amazon DynamoDB 項目：

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

由於此雲端編組範本內嵌簡單的 Lambda 函式，因此可以在 Amazon CloudWatch 日誌群組中查看 Lambda 函式事件，如下所示：

```
{
  "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"
}
```

篩選條件範例

- 僅限符合指定州的產品

此範例修改了 CloudFormation 範本，使其納入符合所有來自佛羅里達州 (縮寫為 "FL") 之產品的篩選條件。

```
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
```

重新部署堆疊後，即可將下列 DynamoDB 項目新增至資料表。請注意，它不會出現在 Lambda 函式日誌中，因為此範例中的產品來自加州。

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

- 僅限以 PK 和 SK 中部分值開頭的項目

此範例修改了 CloudFormation 範本，使其納入下列條件：

```
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
```

請注意，AND 條件要求條件位於模式內部，且其中的索引鍵 PK 和 SK 位於相同表達式內並以逗號分隔。

以 PK 和 SK 中的部分值開頭或來自特定州。

此範例修改了 CloudFormation 範本，使其納入下列條件：

```
  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
```

請注意，新增 OR 條件的方法是在篩選條件區段中引入新的模式。

整合練習 - CDK

以下範例 CDK 專案編組範本逐步解說事件篩選功能。在使用此 CDK 專案之前，必須先安裝必要條件，包括執行準備指令碼。

建立 CDK 專案

首先，`cdk init`在空目錄中叫用 來建立新的 AWS CDK 專案。

```
mkdir ddb_filters
cd ddb_filters
cdk init app --language python
```

`cdk init` 命令使用專案資料夾的名稱來命名專案的各種元素，包括類別、子資料夾和檔案。資料夾名稱中的任何連字號都會轉換為底線。否則，該名稱應遵循 Python 識別符的形式。例如，不應以數字開頭或包含空格。

若要使用新專案，請啟用其虛擬環境。如此可將專案的相依性安裝在本機專案資料夾中，而不是全域安裝。

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

> **Note**
>
> 您可以將其識別為用來啟用虛擬環境的 Mac/Linux 命令。Python 範本包含一個批次檔案 `source.bat`，允許在 Windows 上使用相同的命令。傳統的 Windows 命令 `.venv\Scripts\activate.bat` 也同樣適用。如果您使用 AWS CDK Toolkit v1.70.0 或更早版本初始化 AWS CDK 專案，您的虛擬環境位於 `.env`目錄中，而不是 `.venv`。

基本基礎架構

在您偏好的文字編輯器中開啟檔案 ./ddb_filters/ddb_filters_stack.py。此檔案會在您建立 AWS CDK 專案時自動產生。

接下來，新增函數 _create_ddb_table 和 _set_ddb_trigger_function。這些函數將會建立佈建模式或隨需模式的 DynamoDB 資料表，其中含有分割區索引鍵 PK 和排序索引鍵 SK，且依預設會啟用 Amazon DynamoDB Streams 以顯示新舊映像。

Lambda 函式會儲存在資料夾 lambda 內的檔案 app.py 中。此檔案會在稍後建立。它會包含環境變數 APP_TABLE_NAME (將成為此堆疊所建立 Amazon DynamoDB 資料表的名稱)。在相同的函數中，我們將對 Lambda 函式授予串流讀取權限。最後，它會訂閱 DynamoDB Streams 做為 Lambda 函式的事件來源。

在 __init__ 方法中的檔案尾端，您將會呼叫相應的建構以在堆疊中對其初始化。對於需要其他元件和服務的較大專案，最好在基本堆疊之外定義這些建構。

```
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)
```

現在，我們將建立一個非常簡單的 Lambda 函式，以將日誌列印至 Amazon CloudWatch。若要執行此操作，請建立名為 `lambda` 的新資料夾。

```
mkdir lambda
touch app.py
```

使用您偏好的文字編輯器，將下列內容新增至 `app.py` 檔案：

```
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)
```

確定您位於 /ddb_filters/ 資料夾中，並輸入下列命令以建立範例應用程式：

```
cdk deploy
```

在特定時間點，系統會要求您確認是否要部署解決方案。輸入 Y 以接受變更。

```
##########################################################################################
# + # ${LambdaHandler/ServiceRole} # arn:${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
```

部署變更後，請開啟 AWS 主控台，並將一個項目新增至資料表。

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
```

```
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

此時，CloudWatch 日誌應已包含此項目中的所有資訊。

篩選條件範例

- 僅限符合指定州的產品

開啟檔案 ddb_filters/ddb_filters/ddb_filters_stack.py 加以修改，使其納入符合所有等於 "FL" 之產品的篩選條件。這可以在第 45 行中 event_subscription 的正下方進行修改。

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
)
```

- 僅限以 PK 和 SK 中部分值開頭的項目

修改 Python 指令碼以納入下列條件：

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
```

```
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
        ]
    },
```

- 以位於 PK 和 SK 中的部分值開頭或來自特定州。

修改 python 指令碼以納入下列條件：

```
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"]}}}}
                )
            },
        ]
    },
)
```

請注意，新增 OR 條件的方法是在篩選條件陣列中新增更多元素。

清除

請在工作目錄的底部找到篩選條件堆疊，然後執行 `cdk destroy`。系統會要求您確認刪除此資源：

```
cdk destroy
Are you sure you want to delete: DdbFiltersStack (y/n)? y
```

搭配 Lambda 使用 DynamoDB Streams 的最佳實務

AWS Lambda 函數會在容器中執行，這是與其他函數隔離的執行環境。當您第一次執行函數時， 會 AWS Lambda 建立新的容器，並開始執行函數的程式碼。

Lambda 函式有一個處理常式，每次調用只執行一次。此處理常式包含函數的主要商業邏輯。例如，步驟 4：建立並測試 Lambda 函式 中顯示的 Lambda 函式有一個可處理 DynamoDB Streams 紀錄的處理常式。

您也可以在建立容器之後，但在第一次 AWS Lambda 執行處理常式之前，提供僅執行一次的初始化程式碼。步驟 4：建立並測試 Lambda 函式 中顯示的 Lambda 函式有初始程式碼，可匯入適用於 Node.js 中 JavaScript 的開發套件，並建立 Amazon SNS 的用戶端。這些物件應該只在處理常式外部定義一次。

函數執行後， AWS Lambda 可能會選擇重複使用容器來後續叫用函數。在這種情況下，您的函數處理常式或許能夠重複使用您在初始化程式碼中定義的資源。(您完全無法控制 AWS Lambda 保留容器多長時間，或容器是否得以重複使用。)

對於使用 的 DynamoDB 觸發程序 AWS Lambda，我們建議下列事項：

- AWS 服務用戶端應該在初始化程式碼中執行個體化，而不是在處理常式中。這可讓 AWS Lambda 在容器生命週期內重複使用現有的連線。
- 一般而言，您不需要明確管理連線或實作連線集區， 會為您 AWS Lambda 管理。

DynamoDB 串流的 Lambda 取用者無法保證僅交付一次，並可能偶爾導致出現重複項目。請確定您的 Lambda 函式程式碼具有冪等性，以防止因重複處理而產生意外問題。

如需詳細資訊，請參閱《 AWS Lambda 開發人員指南》中的使用 AWS Lambda 函數的最佳實務。

DynamoDB Streams 和 Apache Flink

您可以使用 Apache Flink 取用 Amazon DynamoDB Streams 記錄。透過 Amazon Managed Service for Apache Flink，您可以使用 Apache Flink 即時轉換並分析串流資料。Apache Flink 是開放原始碼串流處理架構，用於處理即時資料。Apache Flink 專用的 Amazon DynamoDB Streams 連接器可簡化 Apache Flink 工作負載的建置和管理，方便您將應用程式與其他 AWS 服務整合。

Amazon Managed Service for Apache Flink 可協助您快速建置端對端串流處理應用程式，以進行日誌分析、點擊流分析、物聯網 (IoT)、廣告技術、遊戲等。四個最常見的使用案例是串流擷取、轉換和載入 (ETL)、事件驅動型應用程式、回應式即時分析，以及資料串流的互動式查詢。如需從 Amazon DynamoDB Streams 寫入 Apache Flink 的詳細資訊，請參閱 Amazon DynamoDB Streams 連接器。

使用 DynamoDB Accelerator (DAX) 的記憶體內加速

Amazon DynamoDB 是專為擴展與效能所設計。在大多數情況下，DynamoDB 回應時間可以測量到個位數毫秒。但是，有些使用案例需要以微秒為單位的回應時間。針對這些使用案例，DynamoDB Accelerator (DAX) 在存取最終一致資料時可提供快速的回應時間。

DAX 是與 DynamoDB 相容的快取服務，可讓您利用快速的記憶體內效能，滿足高需求的應用程式。DAX 可解決三種核心案例：

1. 作為記憶體快取，DAX 會以十倍為單位 (從個位數毫秒到微秒) 來縮短最終一致讀取工作負載的回應時間。
2. DAX 可提供與 DynamoDB API 相容的受管服務，減少操作和應用程式的複雜度。因此，它只需要極少的功能變更，便能搭配現有應用程式使用。
3. 針對需要大量讀取或爆量的工作負載，DAX 會降低過度佈建讀取容量單位的需求，以此增加輸送量以及節省可能的操作成本。這對需要重複讀取個別索引鍵的應用程式特別有利。

DAX 支援伺服器端加密。使用靜態加密功能時，DAX 在磁碟上保留的資料會受到加密。DAX 將資料寫入磁碟，做為自主節點到僅供讀取複本的變更環節之一。如需更多詳細資訊，請參閱 DAX 靜態加密。

DAX 也支援傳輸中加密功能，確保應用程式與叢集之間的所有請求和回應都經由 Transport Layer Security (TLS) 加密，並透過驗證叢集 x509 憑證對叢集的連線進行身分驗證。如需更多詳細資訊，請參閱 DAX 傳輸中加密。

主題

- DAX 使用案例
- DAX 使用須知
- DAX 的運作方式
- DAX 叢集元件
- 建立 DAX 叢集
- DAX 與 DynamoDB 一致性模式
- 使用 DynamoDB Accelerator (DAX) 用戶端開發
- 管理 DAX 叢集
- 監控 DynamoDB Accelerator
- DAX T3/T2 爆量執行個體

- DAX 存取控制
- DAX 靜態加密
- DAX 傳輸中加密
- 使用 DAX 的服務連結 IAM 角色
- 跨 AWS 帳戶存取 DAX
- DAX 叢集調整大小指南

DAX 使用案例

DAX 可讓您以微秒延遲，存取 DynamoDB 資料表的最終一致資料。多可用區域 DAX 叢集每秒可處理數百萬個請求。

DAX 適合下列應用程式類型：

- 需要盡可能最快讀取回應時間的應用程式。部分範例包括即時競標、社群遊戲與交易應用程式。DAX 為這些使用案例提供快速的記憶體內讀取效能。
- 比其他應用程式更常讀取少量項目的應用程式。例如，假設有一部電子商務系統，正對某項熱門產品展開一日促銷。在促銷期間，與其他所有產品相比，針對該產品 (及其在 DynamoDB 中的資料) 的需求會驟增。若要減輕「熱」鍵與不一致流量分佈的影響，您可以將讀取活動卸載至 DAX 快取，直到一日促銷結束為止。
- 需要大量讀取但對成本也很敏感的應用程式。透過 DynamoDB，您可以佈建應用程式所需的每秒讀取數目。如果讀取活動增加，您可以增加資料表的佈建讀取輸送量 (需額外付費)。或者，您可以將活動從應用程式卸載至 DAX 叢集，並減少需要額外購買的讀取容量單位數。
- 需要對大型資料集重複讀取的應用程式。這類應用程式可能會從其他應用程式重新分配資料庫資源。例如，長時間執行的區域天氣資料分析可能會暫時使用 DynamoDB 資料表中的所有讀取容量。這種情況會對其他需要存取相同資料的應用程式造成負面影響。透過 DAX，可改為對快取資料執行天氣分析。

DAX 不適合下列應用程式類型：

- 需要高度一致性讀取 (或無法容忍最終一致讀取) 的應用程式。
- 讀取回應時間不需要到微秒，或不需從基礎資料表卸載重複讀取活動的應用程式。
- 寫入密集型的應用程式。大量寫入會導致叢集中 DAX 節點的複寫增加。這會造成增加消耗資源，並產生可用性問題風險。

- 沒有許多重複讀取的應用程式。快取命中率超過 90% 時，DAX 可達到最佳效能。較低的快取命中率會增加快取遺漏，導致整個 DAX 叢集耗用更多資源。

DAX 使用須知

- 如需可使用 DAX AWS 的區域清單，請參閱 Amazon DynamoDB 定價。
- DAX 支援以 Go、Java、Node.js、Python 和 .NET 撰寫的應用程式，並使用 AWS這些程式設計語言提供的用戶端。
- DAX 僅適用於 EC2-VPC 平台。
- DAX 叢集服務角色政策必須允許 `dynamodb:DescribeTable` 動作，才能維護有關 DynamoDB 資料表的中繼資料。
- DAX 叢集會維護其存放項目屬性名稱的中繼資料。它會無限期地維護該中繼資料 (即使項目過期或從快取中移出也一樣)。長期下來，使用屬性名稱未限制數量的應用程式可能會在 DAX 叢集中造成記憶體用盡。此限制僅適用於頂層屬性名稱，而非巢狀屬性名稱。有問題的頂層屬性名稱包括時間戳記、UUID 和工作階段 ID。

  此限制僅適用於屬性名稱，而非其值。與以下相似的項目則不是問題。

```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "CreationDate": "2017-10-24T01:02:03+00:00"
}
```

  但與以下相似的項目，若數量夠多且每個都具有不同的時間戳記，則可能會造成問題。

```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "2017-10-24T01:02:03+00:00": "created"
}
```

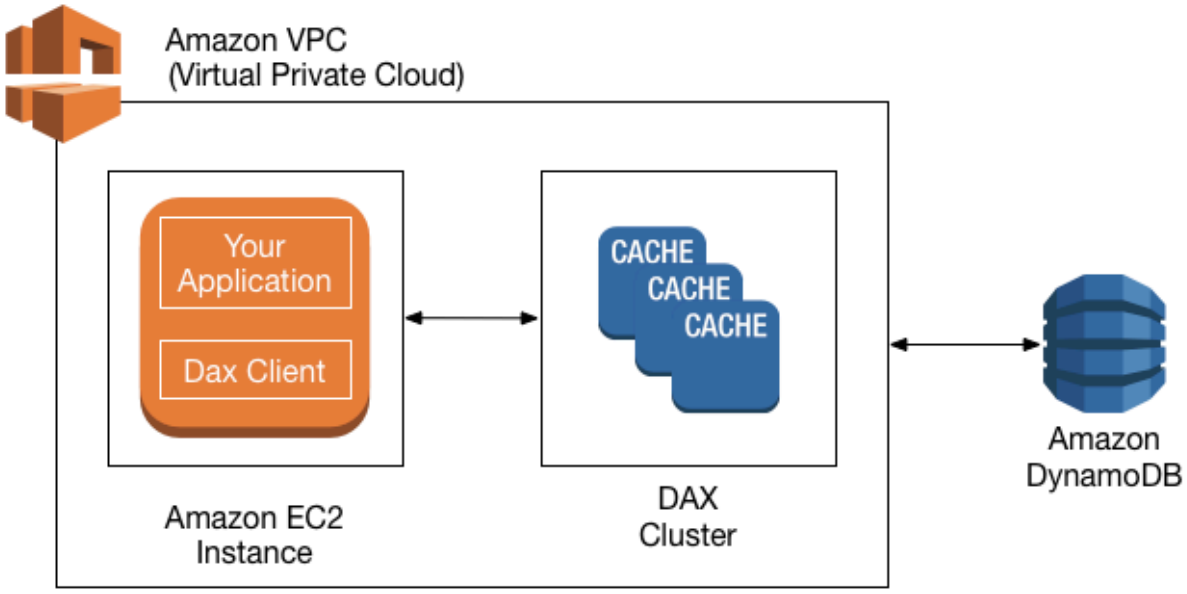
DAX 的運作方式

Amazon DynamoDB Accelerator (DAX) 旨在於 Amazon Virtual Private Cloud (Amazon VPC) 環境內執行。Amazon VPC 服務會定義虛擬網路，與傳統資料中心幾乎一模一樣。使用 VPC，您可以控制其

IP 位址範圍、子網路、路由表、網路閘道及安全設定。您可以在虛擬網路中啟動 DAX 叢集，然後使用 Amazon VPC 安全群組來控制對該叢集的存取。

> **Note**
>
> 如果您在 2013 年 12 月 4 日之後建立 AWS 帳戶，則每個 AWS 區域都已有一個預設 VPC。VPC 已立即可供您使用，無須進行任何其他設定步驟。
> 如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的預設 VPC 和預設子網路。

下圖顯示 DAX 的高層級概觀。



若要建立 DAX 叢集，您可以使用 AWS 管理主控台。除非另有指定，否則您的 DAX 叢集會在您的預設 VPC 中執行。若要執行應用程式，請在 Amazon VPC 內啟動 Amazon EC2 執行個體。接著在 EC2 執行個體上部署應用程式 (使用 DAX 用戶端)。

DAX 用戶端會在執行時期，將應用程式的所有 DynamoDB API 請求導向該 DAX 叢集。若 DAX 可直接處理其中一個 API 請求，它便會執行該作業。否則，它會將請求傳遞給 DynamoDB。

DAX 叢集最後會將結果傳回您的應用程式。

主題

- DAX 如何處理請求
- 項目快取
- 查詢快取

DAX 如何處理請求

DAX 叢集會包含一或多個節點。而每個節點都會執行自己的 DAX 快取軟體執行個體。其中一個節點會作為叢集的主節點。其他節點 (如其存在) 則為讀取複本。如需詳細資訊，請參閱節點。

您的應用程式可以透過指定 DAX 叢集的端點，存取 DAX。DAX 用戶端軟體可使用叢集端點，執行智慧型負載平衡和路由。

讀取操作

DAX 可以回應下列 API 呼叫：

- `GetItem`
- `BatchGetItem`
- `Query`
- `Scan`

若請求指定最終一致讀取 (預設行為)，它會嘗試從 DAX 讀取該項目：

- DAX 如有該項目 (快取命中)，DAX 不需存取 DynamoDB 就能將該項目傳回應用程式。
- DAX 若沒有該項目 (快取未中)，DAX 會將請求傳遞給 DynamoDB。當它從 DynamoDB 收到回應後，DAX 便會將結果傳回應用程式。但它也會將結果寫入主節點上的快取。

> **Note**
>
> 叢集中如有任何僅供讀取複本，DAX 會自動將該複本與主節點保持同步。如需詳細資訊，請參閱叢集。

若請求指定高度一致性讀取，DAX 會將請求傳遞給 DynamoDB。DynamoDB 的結果將不會在 DAX 中進行快取。他們只會傳回應用程式。

寫入操作

下列 DAX API 操作會視為「全部寫入」：

- `BatchWriteItem`
- `UpdateItem`
- `DeleteItem`
- `PutItem`

這些操作會先將資料寫入 DynamoDB 資料表，然後再寫入 DAX 叢集。資料必須同時成功寫入資料表與 DAX，操作才算成功。

其他操作

DAX 無法識別任何管理資料表的 DynamoDB 操作 (例如 `CreateTable`、`UpdateTable`)。您的應用程式如需執行這些操作，它必須直接存取 DynamoDB 而非使用 DAX。

如需有關 DAX 和 DynamoDB 一致性的詳細資訊，請參閱 DAX 與 DynamoDB 一致性模式。

如需有關 DAX 中交易的運作方式資訊，請參閱 使用 DynamoDB Accelerator (DAX) 中的交易 API。

請求率限制

如果傳送至 DAX 的請求數超過節點的容量，DAX 會傳回 ThrottlingException，將速率限制為其接受其他請求的程度。DAX 會持續評估您的 CPU 使用率，以判斷其可以處理的請求量，同時維持良好的叢集狀態。

您可以監控 DAX 發布至 Amazon CloudWatch 的 ThrottledRequestCount 指標。如果您每隔一段時間就會看到這些例外狀況，請考慮擴展您的叢集。

項目快取

DAX 會維護項目快取，存放來自 `GetItem` 及 `BatchGetItem` 操作的結果。快取中的項目代表來自 DynamoDB 的最終一致資料，並依其主索引鍵值存放。

應用程式傳送 `GetItem` 或 `BatchGetItem` 請求時，DAX 會嘗試使用指定的索引鍵值，直接從項目快取讀取項目。若找到項目 (快取命中)，DAX 會立即將其傳回應用程式。如果找不到項目 (快取未中)，DAX 會將請求傳送至 DynamoDB。DynamoDB 使用最終一致讀取來處理請求，並將項目傳回 DAX。DAX 會將它們存放在項目快取中，然後傳回至應用程式。

項目快取有存留時間 (TTL) 設定，預設為 5 分鐘。DAX 會指派時間戳記給其寫入項目快取的每個快取。若項目存留在快取中的時間長於 TTL 設定，項目便會過期。若是對過期的項目發出 `GetItem` 請求，會將其視為快取未中。DAX 會因此而傳送 `GetItem` 請求給 DynamoDB。

> **Note**
>
> 您可以在建立新的 DAX 叢集時，為項目快取指定 TTL 設定。如需詳細資訊，請參閱管理 DAX 叢集。

DAX 也會為項目快取維護一份最近最少使用 (LRU) 的清單。LRU 清單會追蹤項目第一次寫入快取的時間，以及上一次從快取中讀取項目的時間。當項目快取變滿時，DAX 會移出較舊的項目 (無論項目過期與否)，以清出空間存放新的項目。LRU 演算法一律會為項目快取啟用，使用者無法加以設定。

如果您將項目快取 TTL 設定指定為零，項目快取中的項目僅會因 LRU 移出或全部寫入操作而重新整理。

如需 DAX 中項目快取的一致性詳細資訊，請參閱 DAX 項目快取行為。

查詢快取

DAX 也會維護查詢快取，存放來自 `Query` 及 `Scan` 操作的結果。此快取中的項目代表來自對 DynamoDB 資料表之查詢與掃描的結果集。這些結果集會依其參數值存放。

當應用程式傳送 `Query` 或 `Scan` 請求時，DAX 會嘗試使用指定的參數值，從查詢快取讀取相符的結果集。若有找到結果集 (快取命中)，DAX 會立即將其傳回應用程式。如果找不到結果集 (快取未中)，DAX 會將請求傳送至 DynamoDB。DynamoDB 使用最終一致讀取來處理請求，並將結果集傳回 DAX。DAX 會將它存放在查詢快取中，然後傳回至應用程式。

> **Note**
>
> 您可以在建立新的 DAX 叢集時，為查詢快取指定 TTL 設定。如需詳細資訊，請參閱管理 DAX 叢集。

DAX 也會為查詢快取維護一份 LRU 清單。清單會追蹤結果集第一次寫入快取的時間，以及上一次從快取讀取結果的時間。當查詢快取變滿時，DAX 會移出較舊的結果集 (無論結果集過期與否)，以清出空間存放新的結果集。LRU 演算法一律會為查詢快取啟用，使用者無法加以設定。

如果您將查詢快取 TTL 設定指定為零，查詢回應將不會快取。

如需 DAX 中查詢快取的一致性詳細資訊，請參閱 DAX 查詢快取行為。

DAX 叢集元件

Amazon DynamoDB Accelerator (DAX) 叢集由 AWS 基礎設施元件組成。本節說明這些元件及其運作方式。

主題

- 節點
- 叢集
- 區域與可用區域
- 參數群組
- 安全群組
- 叢集 ARN
- 叢集端點
- 節點端點
- 子網路群組
- 事件
- Maintenance window (維護時段)

節點

節點是 DAX 叢集最小的建置區塊。每個節點都會執行 DAX 軟體的執行個體，並且維持快取資料的單一複本。

您可以使用下列兩種方式的其中之一，擴展您的 DAX 叢集：

- 為叢集新增更多節點。這會增加叢集整體的讀取輸送量。
- 使用更大的節點類型。更大的節點類型可提供更多容量，並可以增加輸送量。(您必須使用新的節點類型建立新叢集。)

叢集中每個節點的節點類型都相同，並且會執行相同的 DAX 快取軟體。如需可用的節點類型清單，請參閱 Amazon DynamoDB 定價。

叢集

叢集是 DAX 作為單一單位管理的一或多個節點的邏輯群組。叢集中的其中一個節點會指定為主節點，其他節點 (若有的話) 則為僅供讀取複本。

主節點負責下列項目：

- 滿足應用程式對快取資料的請求。
- 處理對 DynamoDB 的寫入操作。
- 根據叢集的移出政策，從快取中移出資料。

對主節點上的快取資料進行變更時，DAX 會使用複寫日誌將變更散佈到所有僅供讀取複本節點。當從所有僅供讀取複本收到確認後，DynamoDB 會從主節點刪除複寫日誌。

DAX 叢集最多可支援每個叢集 11 個節點 (主節點，加上最多 10 個僅供讀取複本)。

僅供讀取複本負責下列項目：

- 滿足應用程式對快取資料的請求。
- 根據叢集的移出政策，從快取中移出資料。

但是，與主節點不同的是，僅供讀取複本不會對 DynamoDB 進行寫入。

僅供讀取複本還有另外兩種用途：

- 可擴展性。若您有大量的用戶端需要並行存取 DAX，您可以新增更多複本進行讀取擴展。DAX 會將負載平均分佈到叢集中的所有節點。(另一種增加輸送量的方式是使用更大的快取節點類型。)
- 高可用性。在主節點故障時，DAX 會自動容錯移轉至僅供讀取複本，並將其指派為新的主節點。若複本節點失敗，DAX 叢集中的其他節點仍然可以處理請求，直到可復原失敗的節點為止。為取得最大的容錯能力，建議您將僅供讀取複本部署在分離的可用區域中。這項組態可確保即使整個可用區域都不可使用時，DAX 叢集仍然能夠繼續運作。

> **Important**
>
> 針對生產用途，我們強烈建議搭配至少三個節點使用 DAX，並將每個節點置放在不同的可用區域中。要容錯的 DAX 叢集需要三個節點。

可使用一或兩個節點來部署 DAX 叢集，以供開發或測試工作負載使用。單節點和雙節點叢集無法容錯，因此針對正式作業用途，我們不建議使用少於三個節點。若單節點或雙節點叢集發生軟體或硬體錯誤，叢集可能會無法使用或遺失快取資料。

Important

DAX 叢集最多支援 500 個 DynamoDB 資料表。如果超過 500 個資料表，您的叢集可用性和效能就可能降低。

區域與可用區域

AWS 區域中的 DAX 叢集只能與位於相同區域的 DynamoDB 資料表互動。因此，請確認您在正確的區域中啟動您的 DAX 叢集。若您在其他區域中具有 DynamoDB 資料表，您必須在那些區域中啟動 DAX 叢集。

每個區域皆設計為與其他區域完全隔離。在每個區域中皆有多個可用區域。藉由在不同的可用區域中啟動您的節點，您可以實現最大的容錯能力。

Important

請不要將您所有的叢集節點置放在單一可用區域中。在這項組態中，若發生可用區域故障的情況，您的 DAX 叢集將會無法使用。

針對生產用途，我們強烈建議搭配至少三個節點使用 DAX，並將每個節點置放在不同的可用區域中。要容錯的 DAX 叢集需要三個節點。

可使用一或兩個節點來部署 DAX 叢集，以供開發或測試工作負載使用。一和兩個節點叢集無法容錯，因此針對生產用途，我們不建議使用少於三個節點。若一或兩個節點叢集發生軟體或硬體錯誤，叢集可能會無法使用或遺失快取資料。

參數群組

「參數群組」可用來管理 DAX 叢集的執行時期設定。DAX 有數個參數，可讓您用來最佳化效能 (例如定義快取資料的 TTL 政策)。參數群組為可以套用到叢集的具名參數組。藉由執行此作業，您可以確保該叢集中的所有節點都以完全相同的方式設定。

安全群組

DAX 叢集會在 Amazon Virtual Private Cloud (Amazon VPC) 環境中執行。此環境是一個您 AWS 帳戶的專用虛擬網路，與其他 VPC 隔離。「安全群組」做為 VPC 的虛擬防火牆，可讓您控制傳入及傳出網路流量。

當您在 VPC 中啟動叢集時，您會為您的安全群組新增「輸入」規則，允許傳入網路流量。輸入規則會指定您叢集的通訊協定 (TCP) 和連接埠號碼 (8111)。在為您的安全群組新增此規則後，在您 VPC 中執行的應用程式便可存取 DAX 叢集。

叢集 ARN

每個 DAX 叢集都會獲得指派一個 Amazon Resource Name (ARN)。ARN 格式如下。

```
arn:aws:dax:region:accountID:cache/clusterName
```

您可以在 IAM 政策中使用叢集 ARN，定義 DAX API 操作的許可。如需更多詳細資訊，請參閱 DAX 存取控制。

叢集端點

每個 DAX 叢集都會提供「叢集端點」供您的應用程式使用。藉由使用此端點存取叢集，您的應用程式便不需要知道叢集中個別節點的主機名稱和連接埠號碼。您的應用程式會自動「得知」叢集中的所有節點，即使您新增或移除僅供讀取複本。

以下是 us-east-1 區域中未設為使用傳輸中加密的叢集端點範例。

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

以下是相同區域中未設為使用傳輸中加密的叢集端點範例。

```
daxs://my-encrypted-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

節點端點

DAX 叢集上的每個個別節點都有其自身的主機名稱和連接埠號碼。以下是「節點端點」的範例。

```
myDAXcluster-a.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111
```

您的應用程式可使用其端點直接存取節點。但是，我們建議您將 DAX 叢集作為單一單位處理，並改為使用叢集端點進行存取。叢集端點可讓您的應用程式無須維持節點的清單，並在您為叢集新增節點或從叢集移除節點時，將該清單保持在最新狀態。

子網路群組

對 DAX 叢集節點的存取會限制於在 Amazon VPC 環境中 Amazon EC2 執行個體上執行的應用程式。您可以使用「子網路群組」授予叢集從特定子網路上所執行 Amazon EC2 執行個體進行存取的權限。子網路群組是子網路的集合 (一般是私有)，您可以為在 Amazon VPC 環境中執行的叢集指定這些子網路。

建立 DAX 叢集時，您必須指定子網路群組。DAX 會使用此子網路群組選取子網路及該子網路中的 IP 位址，以與您的節點建立關聯。

事件

DAX 會記錄您叢集內重要的事件，例如新增節點失敗、新增節點成功，或是變更安全群組。藉由監控重要的事件，您可以了解叢集目前的狀態，並根據事件採取正確的動作。您可以使用 DAX 管理 API 中的 AWS 管理主控台 或 `DescribeEvents` 動作存取這些事件。

您也可以請求將通知傳送至特定的 Amazon Simple Notification Service (Amazon SNS) 主題。您接著便會在您的 DAX 叢集中發生事件時立即得知。

Maintenance window (維護時段)

每個叢集都有每週一次的維護時段，用於套用系統變更。變更依序套用時，會取代現有節點，並將套用變更的新節點新增至叢集。在此期間，您的應用程式可能會發現暫時錯誤或限流。因此我們建議您在最低用量時間內排定維護時段，並視需要定期調整此排程。您可以指定最高達 24 小時的時間範圍，並讓任何您請求的維護活動在此期間進行。

若您在建立或修改快取叢集時並未指定偏好的維護時段，則 DAX 會在工作日隨機選取一天並指派 60 分鐘的維護時段。這個 60 分鐘時段是從每個 AWS 區域 的 8 小時時段內隨機選取。下表列出每個區域的時段，預設維護時段會從此時段中指派。

區域代碼	區域名稱	Maintenance window (維護時段)
ap-northeast-1	亞太地區 (東京) 區域	下午 1 時至下午 9 時 (UTC)
ap-southeast-1	亞太地區 (新加坡) 區域	下午 2 時至下午 10 時 (UTC)
ap-southeast-2	亞太地區 (雪梨) 區域	中午 12 時至下午 8 時 (UTC)

區域代碼	區域名稱	Maintenance window (維護時段)
ap-south-1	亞太地區 (孟買) 區域	下午 5 時 30 分至次日上午 1 時 30 分 (UTC)
cn-northwest-1	中國 (寧夏) 區域	下午 11 時至次日上午 7 時 (UTC)
cn-north-1	中國 (北京) 區域	下午 2 時至下午 10 時 (UTC)
eu-central-1	歐洲 (法蘭克福) 區域	下午 11 時至次日上午 7 時 (UTC)
eu-north-1	歐洲 (斯德哥爾摩) 區域	上午 1 時至上午 9 時 (UTC)
eu-south-2	歐洲 (西班牙) 區域	下午 9 時至次日上午 5 時 (UTC)
eu-west-1	歐洲 (愛爾蘭) 區域	下午 10 時至次日上午 6 時 (UTC)
eu-west-2	歐洲 (倫敦) 區域	下午 11 時至次日上午 7 時 (UTC)
eu-west-3	歐洲 (巴黎) 區域	下午 11 時至次日上午 7 時 (UTC)
sa-east-1	南美洲 (聖保羅) 區域	上午 1 時至上午 9 時 (UTC)
us-east-1	美國東部 (維吉尼亞北部) 區域	上午 3 時至上午 11 時 (UTC)
us-east-2	美國東部 (俄亥俄) 區域	下午 11 時至次日上午 7 時 (UTC)
us-west-1	美國西部 (加利佛尼亞北部) 區域	上午 6 時至下午 2 時 (UTC)
us-west-2	美國西部 (奧勒岡) 區域	上午 6 時至下午 2 時 (UTC)

建立 DAX 叢集

本節會帶您逐步在預設 Amazon Virtual Private Cloud (Amazon VPC) 環境中首次設定和使用 Amazon DynamoDB Accelerator (DAX)。您可以使用 AWS Command Line Interface (AWS CLI) 或 建立第一個 DAX 叢集 AWS 管理主控台。

在您建立 DAX 叢集之後，您可以從相同 VPC 中執行的 Amazon EC2 執行個體存取它。您接著可以搭配應用程式使用您的 DAX 叢集。如需詳細資訊，請參閱 使用 DynamoDB Accelerator (DAX) 用戶端開發。

主題

- 為 DAX 建立可存取 DynamoDB 的 IAM 服務角色
- DAX 和 IPv6
- 使用 建立 DAX 叢集 AWS CLI
- 使用 建立 DAX 叢集 AWS 管理主控台

為 DAX 建立可存取 DynamoDB 的 IAM 服務角色

若要讓 DAX 叢集代您存取 DynamoDB 資料表，您必須建立服務角色。服務角色是 AWS Identity and Access Management (IAM) 角色，授權 AWS 服務代表您。服務角色可允許 DAX 存取您的 DynamoDB 資料表，就像是您自己存取資料表一般。您需要建立服務角色，才能建立 DAX 叢集。

如果您是使用主控台，則建立叢集的工作流程會檢查是否有既有 DAX 服務角色。如果找不到，則主控台會為您建立新的服務角色。如需詳細資訊，請參閱the section called “步驟 2：建立 DAX 叢集”。

如果您使用的是 AWS CLI，則必須指定先前建立的 DAX 服務角色。否則，您需要事先建立新的服務角色。如需詳細資訊，請參閱 步驟 1：為 DAX 建立 IAM 服務角色，以使用 存取 DynamoDB AWS CLI。

建立服務角色所需的許可

AWS 受管 `AdministratorAccess` 政策提供了建立 DAX 叢集和服務角色所有需要的許可。若您的使用者已連接 `AdministratorAccess`，您便無需採取進一步的動作。

否則，您必須將下列許可新增至您的 IAM 政策，讓您的使用者可以建立服務角色：

- `iam:CreateRole`

- iam:CreatePolicy
- iam:AttachRolePolicy
- iam:PassRole

將這些許可連接到嘗試執行動作的使用者。

> **Note**
>
> iam:CreateRole、iam:AttachRolePolicy、 iam:CreatePolicy和 iam:PassRole許可不包含在 DynamoDB 的 AWS 受管政策中。這是根據方案設計，因為這些許可可能會提供權限提升：也就是使用者可以使用這些許可來建立新的管理員政策，然後將該政策連接至現有角色。因此，您 (DAX 叢集管理員) 必須將這些許可明確地新增至您的政策。

疑難排解

若您的使用者政策遺漏 iam:CreateRole、iam:CreatePolicy 和 iam:AttachPolicy 許可，您將會收到錯誤訊息。下表列出這些訊息，並說明如何修正問題。

如果您看到此錯誤訊息...	請執行下列操作：
User: arn:aws:iam:: *accountID* :user/*userName* is not authorized to perform: iam:CreateRole on resource: arn:aws:iam:: *accountID* :role/service-role/ *roleName*	將 iam:CreateRole 新增至使用者政策。
User: arn:aws:iam:: *accountID* :user/*userName* is not authorized to perform: iam:CreatePolicy on resource: policy *policyName*	將 iam:CreatePolicy 新增至使用者政策。
User: arn:aws:iam:: *accountID* :user/*userName* is not authorized to perform: iam:AttachRolePolicy on resource: role *daxServiceRole*	將 iam:AttachRolePolicy 新增至使用者政策。

如需 DAX 叢集管理所需 IAM 政策的詳細資訊，請參閱 DAX 存取控制。

DAX 和 IPv6

DynamoDB DAX 現在支援 IPv6 定址，可讓您建立以IPv4-only, IPv6-only 或雙堆疊聯網模式運作的叢集。這有助於增強聯網功能，以滿足不斷發展的基礎設施需求。

網路類型：

您可以使用下列網路類型建立叢集：

- IPv4-only
- IPv6-only
- 雙堆疊 （ 同時支援 IPv4 和 IPv6)

主要功能：

透過 IPv6 支援，您可以執行下列動作：

- 網路組態選項：
  - 上的IPv4-only 和雙堆疊叢集`dual_stack subnets`。
  - IPv6-only子網路上的IPv6-only 叢集。
- 子網路群組管理：
  - 建立IPv4-only, IPv6-only 或雙堆疊支援的子網路群組
  - 使用其他 VPC 子網路修改現有的子網路群組
  - 將IPv6-only 的子網路新增至 IPv6-configured子網路群組
  - 將 IPv4 或雙堆疊子網路新增至 IPv4 和雙堆疊設定的群組
- 用戶端組態：
  - 進行資料平面呼叫時，您可以使用下列方式為 dual_stack 叢集設定偏好的 IP 通訊協定：
    - `ip_discovery` Python SDK 中的 參數
    - `ipDiscovery` 其他 SDKs中的 參數
  - 預設：未指定通訊協定偏好設定時的 IPv4

在 DAX 叢集中實作 IPv6 之前，您必須考慮下列事項：

- 叢集建立後無法變更網路類型

- 對於雙堆疊叢集，用戶端組態中的 `ip_discovery/ipDiscovery` 參數會決定要使用的 IP 通訊協定 (IPv4 或 IPv6)
- 不同的應用程式可以根據其組態，使用不同的 IP 通訊協定連線到相同的雙堆疊叢集

Example用戶端設定範例

```
DynamoDbAsyncClient client = ClusterDaxAsyncClient.builder()
        .overrideConfiguration(Configuration.builder()
            .url(endpoint)             // DAX cluster endpoint
            .ipDiscovery(ipDiscovery)       // IP discovery type (IPv4 or IPv6)
            .build())
        .build();
```

Important

當您使用以資源為基礎的 IAM 政策來限制IPv6-only 環境中 DynamoDB 資料表的 IP 地址時，如果您封鎖 IPv4 地址空間 ()，則必須為 DAX 叢集的 IAM 角色建立例外狀況`0.0.0.0/0`。將`ArnNotEquals`條件新增至您的政策，特別允許存取 DAX 叢集的 IAM 角色，同時維護其他存取路徑的 IP 型限制。如果沒有此例外狀況，DAX 無法存取您的 DynamoDB 資料表。
例如：

Example

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection",
      "Condition": {
        "ArnNotEquals": {
          "aws:PrincipalArn": "arn:aws:iam::123456789012:role/
DAXServiceRoleForDynamoDBAccess"
        },
        "IpAddress": {
```

```
            "aws:SourceIp": "0.0.0.0/0"
          }
        }
      }
    ]
}
```

使用 建立 DAX 叢集 AWS CLI

本節說明如何使用 AWS Command Line Interface (AWS CLI) 建立 Amazon DynamoDB Accelerator (DAX) 叢集。若您尚未執行此作業，您必須安裝及設定 AWS CLI。若要執行此作業，請參閱《AWS Command Line Interface 使用者指南》中的以下說明：

- 安裝 AWS CLI
- 設定 AWS CLI

> **Important**
>
> 若要使用 管理 DAX 叢集 AWS CLI，請安裝或升級至 1.11.110 版或更新版本。

所有 AWS CLI 範例都使用`us-west-2`區域和虛構的帳戶 IDs。

主題

- 步驟 1：為 DAX 建立 IAM 服務角色，以使用 存取 DynamoDB AWS CLI
- 步驟 2：建立子網路群組
- 步驟 3：使用 建立 DAX 叢集 AWS CLI
- 步驟 4：使用 設定安全群組傳入規則 AWS CLI

步驟 1：為 DAX 建立 IAM 服務角色，以使用 存取 DynamoDB AWS CLI

您必須先為 Amazon DynamoDB Accelerator (DAX) 叢集建立服務角色，才可建立該叢集。服務角色是 AWS Identity and Access Management (IAM) 角色，授權 AWS 服務代表您。服務角色可允許 DAX 存取您的 DynamoDB 資料表，就像是您自己存取資料表一般。

在此步驟中，您會建立 IAM 政策，然後將該政策連接到 IAM 角色。這可讓您將角色指派給 DAX 叢集，使其可代您執行 DynamoDB 操作。

建立 DAX 的 IAM 服務角色

1. 使用下列內容建立名為 `service-trust-relationship.json` 的檔案。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. 建立服務角色。

```
aws iam create-role \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --assume-role-policy-document file://service-trust-relationship.json
```

3. 使用下列內容建立名為 `service-role-policy.json` 的檔案。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
```

```
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:111122223333:*"
            ]
        }
    ]
}
```

將 *accountID* 取代為 AWS 您的帳戶 ID。若要尋找 AWS 您的帳戶 ID，請在主控台的右上角選擇您的登入 ID。 AWS 您的帳戶 ID 會顯示在下拉式選單中。

在範例中的 Amazon Resource Name (ARN) 里，*accountID* 必須是 12 位數字。請勿使用連字號或其他任何標點符號。

4. 建立服務角色的 IAM 政策。

```
aws iam create-policy \
    --policy-name DAXServicePolicyForDynamoDBAccess \
    --policy-document file://service-role-policy.json
```

在輸出中，記下您所建立政策的 ARN，如以下範例所示。

```
arn:aws:iam::123456789012:policy/DAXServicePolicyForDynamoDBAccess
```

5. 將該政策連接到服務角色。將以下程式碼中的 *arn* 取代成先前步驟中的實際角色 ARN。

```
aws iam attach-role-policy \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --policy-arn arn
```

接下來，您需要為您的預設 VPC 指定子網路群組。子網路群組是您 VPC 中一或多個子網路的集合。請參閱 步驟 2：建立子網路群組。

步驟 2：建立子網路群組

依照此程序，使用 AWS Command Line Interface () 為您的 Amazon DynamoDB Accelerator (DAX) 叢集建立子網路群組AWS CLI。

> **Note**
>
> 如果您已建立預設 VPC 的子網路群組，則可略過此步驟。

DAX 旨在於 Amazon Virtual Private Cloud (Amazon VPC) 環境內執行。如果您是在 2013 年 12 月 4 日之後建立 AWS 帳戶，則您在每個 AWS 區域中都會有預設的 VPC。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的預設 VPC 和預設子網路。

建立子網路群組

1. 若要判斷您預設 VPC 的識別符，請輸入以下命令。

   ```
   aws ec2 describe-vpcs
   ```

   在輸出中，記下您預設 VPC 的識別符，如以下範例所示。

   ```
   vpc-12345678
   ```

2. 判斷與您預設 VPC 相關聯的子網路 ID。使用您的實際 VPC ID (例如 vpc-12345678) 取代 *vpcID*。

   ```
   aws ec2 describe-subnets \
       --filters "Name=vpc-id,Values=vpcID" \
       --query "Subnets[*].SubnetId"
   ```

   在輸出中，記下子網路識別碼 (例如 subnet-11111111)。

3. 建立子網路群組。確認您在 --subnet-ids 參數中至少指定一個子網路 ID。

   ```
   aws dax create-subnet-group \
       --subnet-group-name my-subnet-group \
       --subnet-ids subnet-11111111 subnet-22222222 subnet-33333333 subnet-44444444
   ```

若要建立叢集，請參閱 步驟 3：使用 建立 DAX 叢集 AWS CLI。

步驟 3：使用 建立 DAX 叢集 AWS CLI

請遵循此程序，使用 AWS Command Line Interface (AWS CLI) 在預設 Amazon VPC 中建立 Amazon DynamoDB Accelerator (DAX) 叢集。

建立 DAX 叢集

1. 取得您服務角色的 Amazon Resource Name (ARN)。

   ```
   aws iam get-role \
       --role-name DAXServiceRoleForDynamoDBAccess \
       --query "Role.Arn" --output text
   ```

   在輸出中，記下服務角色 ARN，如以下範例所示。

   ```
   arn:aws:iam::123456789012:role/DAXServiceRoleForDynamoDBAccess
   ```

2. 建立 DAX 叢集。使用前一步驟中的 ARN，取代 *roleARN*。

   ```
   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
   ```

   叢集中的所有節點類型皆為 `dax.r4.large` (`--node-type`)。共有三個節點 (`--replication-factor`)：一個主節點和兩個複本。

   > **Note**
   >
   > 由於 `sudo` 和 `grep` 是保留的關鍵字，您無法在叢集名稱中使用這些字詞來建立 DAX 叢集。例如，`sudo` 和 `sudocluster` 是無效的叢集名稱。

若要檢視叢集狀態，請輸入以下命令。

```
aws dax describe-clusters
```

狀態會在輸出中顯示，例如 `"Status": "creating"`。

> **Note**
>
> 建立叢集需要幾分鐘。當叢集準備就緒時，其狀態會變更為 `available`。同時，請繼續進行 步驟 4：使用 設定安全群組傳入規則 AWS CLI，並遵循其中的說明。

步驟 4：使用 設定安全群組傳入規則 AWS CLI

您 Amazon DynamoDB Accelerator (DAX) 叢集中的節點會使用您 Amazon VPC 的預設安全群組。對於預設安全群組，您必須為未加密的叢集授權 TCP 連接埠 8111 的輸入流量，或為加密的叢集授權連接埠 9111 的輸入流量。這可允許 Amazon VPC 中的 Amazon EC2 執行個體存取您的 DAX 叢集。

> **Note**
>
> 若使用不同的安全群組 (而非 `default`) 啟動您的 DAX 叢集，您必須改為該群組執行此程序。

設定安全群組傳入規則

1. 若要判斷預設安全群組識別符，請輸入以下命令。使用您實際的 VPC ID (來自 步驟 2：建立子網路群組) 取代 *vpcID*。

   ```
   aws ec2 describe-security-groups \
       --filters Name=vpc-id,Values=vpcID Name=group-name,Values=default \
       --query "SecurityGroups[*].{GroupName:GroupName,GroupId:GroupId}"
   ```

   在輸出中，記下安全群組識別碼 (例如 `sg-01234567`)。
2. 然後輸入以下內容。將 *sgID* 取代成您實際的安全群組識別符。請將連接埠 8111 用於未加密的叢集，再將 9111 用於加密的叢集。

   ```
   aws ec2 authorize-security-group-ingress \
       --group-id sgID --protocol tcp --port 8111
   ```

使用 建立 DAX 叢集 AWS 管理主控台

本節說明如何使用 AWS 管理主控台建立 Amazon DynamoDB Accelerator (DAX) 叢集。

主題

- 步驟 1：使用 建立子網路群組 AWS 管理主控台
- 步驟 2：使用 建立 DAX 叢集 AWS 管理主控台
- 步驟 3：使用 設定安全群組傳入規則 AWS 管理主控台

步驟 1：使用 建立子網路群組 AWS 管理主控台

請按照此程序操作，來使用 AWS 管理主控台為您的 Amazon DynamoDB Accelerator (DAX) 叢集建立子網路群組。

> **Note**
>
> 如果您已建立預設 VPC 的子網路群組，則可略過此步驟。

DAX 旨在於 Amazon Virtual Private Cloud (Amazon VPC) 環境內執行。如果您是在 2013 年 12 月 4 日之後建立 AWS 帳戶，則您在每個 AWS 區域中都會有預設的 VPC。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的預設 VPC 和預設子網路。

您必須在建立 DAX 叢集時指定子網路群組，這是建立程序的一部分。子網路群組是您 VPC 中一或多個子網路的集合。當您建立 DAX 叢集時，節點會部署到子網路群組內的子網路。

> **Note**
>
> 具有此 DAX 叢集的 VPC 可以包含其他資源，甚至包含其他服務的 VPC 端點 (但 ElastiCache 的 VPC 端點除外)，並可能導致 DAX 叢集操作發生錯誤。

建立子網路群組

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在導覽窗格中，選擇 DAX 下方的 Subnet groups (子網路群組)。
3. 選擇 Create subnet group (建立子網路群組)。
4. 在 Create subnet group (建立子網路群組) 視窗中，執行下列作業：
    a. Name (名稱)：輸入子網路群組的短名稱。
    b. Description (描述)：輸入子網路群組的描述。

c. VPC ID：為您的 Amazon VPC 環境選擇識別碼。

d. Subnets (子網路)：從清單中選擇一或多個子網路。

> **Note**
>
> 子網路會分佈在多個可用區域中。如果您計劃建立多節點的 DAX 叢集 (主節點和一或多個僅供讀取複本)，建議您選擇多個子網路 ID。DAX 接著便可以將叢集節點部署到多個可用區域。若可用區域無法使用，DAX 會自動容錯移轉至仍然存活的可用區域。您的 DAX 叢集會繼續運作，而不會發生中斷。

當您滿意設定後，請選擇 Create subnet group (建立子網路群組)。

若要建立叢集，請參閱 步驟 2：使用 建立 DAX 叢集 AWS 管理主控台。

步驟 2：使用 建立 DAX 叢集 AWS 管理主控台

請遵循此程序，在您的預設 Amazon VPC 中建立 Amazon DynamoDB Accelerator (DAX) 叢集。

建立 DAX 叢集

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在導覽窗格中，選擇 DAX 下方的 Clusters (叢集)。
3. 選擇 Create cluster (建立叢集)。
4. 在 Create cluster (建立叢集) 視窗中，執行下列作業：

   a. Cluster name (叢集名稱)：輸入您 DAX 叢集的短名稱。

   > **Note**
   >
   > 由於 `sudo` 和 `grep` 是保留的關鍵字，您無法在叢集名稱中使用這些字詞來建立 DAX 叢集。例如，`sudo` 和 `sudocluster` 是無效的叢集名稱。

   b. Cluster description (叢集描述)：輸入叢集的描述。

   c. Node type (節點類型)：為叢集中的所有節點選擇節點類型。

   d. Cluster size (叢集大小)：選擇叢集中的節點數。叢集組成包含一個主節點和最多九個僅供讀取複本。

> **Note**
>
> 如果想要建立單節點叢集，請選擇 1。您的叢集會包含一個主節點。
> 如果您想要建立多節點叢集，請選擇 3 (一個主節點和兩個讀取複本) 到 10 (一個主節點和九個讀取複本) 之間的數字。
>
> > **Important**
> >
> > 針對生產用途，我們強烈建議搭配至少三個節點使用 DAX，其中每個節點都置放在不同的可用區域中。要容錯的 DAX 叢集需要三個節點。
> > 可使用一或兩個節點來部署 DAX 叢集，以供開發或測試工作負載使用。一和兩個節點叢集無法容錯，因此針對生產用途，我們不建議使用少於三個節點。若一或兩個節點叢集發生軟體或硬體錯誤，叢集可能會無法使用或遺失快取資料。

e. 選擇 Next (下一步)。

f. Subnet group (子網路群組)：選取 Choose existing (選擇現有)，然後選擇您在 步驟 1：使用 建立子網路群組 AWS 管理主控台 建立的子網路群組。

g. 存取控制：選擇 default (預設值) 安全群組。

h. 可用區域 (AZ)：選擇 Automatic (自動)。

i. 選擇 Next (下一步)。

j. IAM service role for DynamoDB access (DynamoDB 存取的 IAM 服務角色)：選擇 Create new (建立新的)，然後輸入以下資訊：

- IAM role name (IAM 角色名稱)：輸入 IAM 角色的名稱 (例如 `DAXServiceRole`)。主控台會建立新的 IAM 角色，且您的 DAX 叢集會在執行時期取得此角色。
- 選取 Create policy (建立政策) 旁的方框。
- IAM role policy (IAM 角色政策)：選擇 Read/Write (讀取/寫入)。這會允許 DAX 叢集在 DynamoDB 中執行讀取及寫入操作。
- New IAM policy name (新 IAM 政策名稱)：此欄位會在您輸入 IAM 角色名稱時填入。您也可以輸入 IAM 政策的名稱，例如 `DAXServicePolicy`。主控台即建立新的 IAM 政策，並將政策連接至 IAM 角色。
- Access to DynamoDB tables (存取 DynamoDB 資料表)：選擇 All tables (所有資料表)。

k. Encryption (加密)：選擇 Turn on encryption at rest (開啟靜態加密) 和 Turn on encryption in transit (開啟傳輸中加密)。如需詳細資訊，請參閱 DAX 靜態加密 和 DAX 傳輸中加密。

DAX 也需要個別的服務角色才能存取 Amazon EC2。DAX 會自動為您建立此服務角色。如需詳細資訊，請參閱使用 DAX 的服務連結角色。

5. 當您滿意設定後，選擇 Next (下一步)。
6. Parameter group (參數群組)：選擇 Choose existing (選擇現有)。
7. Maintenance window (維護時段)：若您對於套用的軟體更新沒有偏好，選擇 No preference (無偏好設定)，或選擇 Specify time window (指定時間範圍)，並針對維護時段的排程選項選擇 Weekday (工作日)、Time (UTC) (國際標準時間) 和 Start within (hours) (幾小時後開始)。
8. Tags (標籤)：選擇 Add new tag (新增標籤) 輸入鍵值對，以進行標記。
9. 選擇 Next (下一步)。

在 Review and create (檢閱和建立) 畫面上，您可以查看所有設定。若您已準備好建立叢集，請選擇 Create cluster (建立叢集)。

在 Clusters (叢集) 畫面中，您的 DAX 叢集會列出並顯示 Creating (正在建立) 狀態。

> **Note**
>
> 建立叢集需要幾分鐘。當叢集準備就緒時，其狀態會變更為 Available (可用)。
> 同時，請繼續進行 步驟 3：使用 設定安全群組傳入規則 AWS 管理主控台，並遵循其中的說明。

步驟 3：使用 設定安全群組傳入規則 AWS 管理主控台

您的 Amazon DynamoDB Accelerator (DAX) 叢集透過 TCP 連接埠 8111 (用於未加密的叢集) 或 9111 (用於加密的叢集) 進行通訊，因此您需要授權該連接埠的輸入流量。這可允許 Amazon VPC 中的 Amazon EC2 執行個體存取您的 DAX 叢集。

> **Note**
>
> 若使用不同的安全群組 (而非 `default`) 啟動您的 DAX 叢集，您必須改為該群組執行此程序。

設定安全群組傳入規則

1. 前往 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 在導覽窗格中，選擇 Security Groups (安全群組)。
3. 選擇 default (預設) 安全群組。在 Actions (動作) 選單上，選擇 Edit inbound rules (編輯傳入規則)。
4. 選擇 Add Rule (新增規則) 並輸入以下資訊：
   - Port Range (連接埠範圍)：輸入 8111 (如果您的叢集未加密) 或 9111 (如果您的叢集已加密)。
   - Source (來源)：將此項保留為 Custom (自訂)，然後選擇右側的搜尋欄位，即會顯示下拉式選單。選擇預設安全群組的識別符。
5. 選擇 Save rules (儲存規則) 儲存變更。
6. 若要更新主控台中的名稱，請前往 Name (名稱) 屬性並選擇顯示的 Edit (編輯) 選項。

DAX 與 DynamoDB 一致性模式

Amazon DynamoDB Accelerator (DAX) 是全部寫入快取服務，設計目的是要簡化將快取新增至 DynamoDB 資料表的程序。因為 DAX 與 DynamoDB 分開操作，所以您務必要了解 DAX 和 DynamoDB 的一致性模式，確定應用程式如預期地運作。

在許多使用案例中，應用程式使用 DAX 的方式會影響 DAX 叢集內資料的一致性，以及 DAX 與 DynamoDB 之間資料的一致性。

主題

- DAX 叢集節點之間的一致性
- DAX 項目快取行為
- DAX 查詢快取行為
- 強烈一致和交易讀取
- 負快取
- 寫入策略

DAX 叢集節點之間的一致性

若要達到應用程式的高可用性，建議您使用至少三個節點來佈建 DAX 叢集。然後將這些節點置放到區域中的多個可用區域內。

DAX 叢集執行時，會複寫叢集中所有節點之間的資料 (假設您已佈建超過一個節點)。請考慮使用 DAX 執行成功 UpdateItem 的應用程式。此動作會使用新的值修改主節點中的項目快取。該值接著會複寫到叢集中的所有其他節點。這項複寫最終會一致，而且通常不需要一秒就能完成。

在此情況下，根據每個用戶端所存取的節點，兩個用戶端可能會讀取相同 DAX 叢集中的相同索引鍵，但收到不同的值。在叢集中的所有節點之間完全複寫更新時，這些節點將會一致。(此行為與 DynamoDB 的最終一致性質相似。)

如果您要建置使用 DAX 的應用程式，則應該設計該應用程式，使其容忍最終一致資料。

DAX 項目快取行為

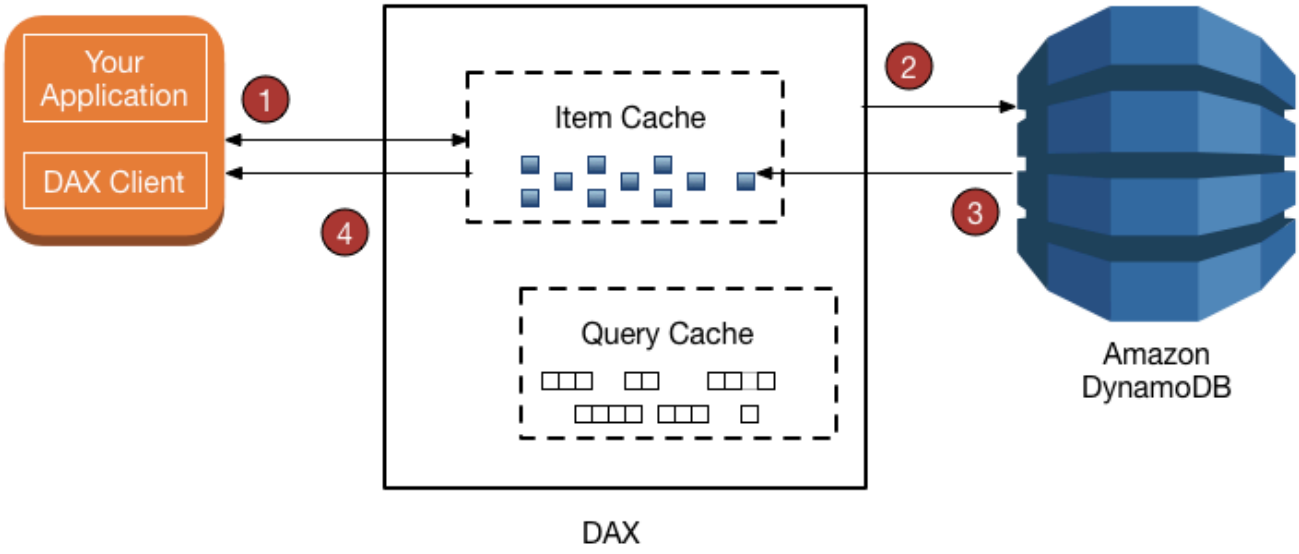
每個 DAX 叢集有兩個相異快取：項目快取與查詢快取。如需詳細資訊，請參閱DAX 的運作方式。

本節會介紹讀取和寫入 DAX 項目快取的一致性隱憂。

讀取一致性

使用 DynamoDB，GetItem 操作預設會執行最終一致讀取。假設您搭配 DynamoDB 用戶端使用 UpdateItem。若您接著立即嘗試讀取相同項目，您可能會看到資料顯示的是更新前的內容。這是因為所有 DynamoDB 儲存體位置之間的傳播延遲。通常可在幾秒內達到一致性。因此若您重試讀取，您可能會看到更新後的項目。

當您搭配 DAX 用戶端使用 GetItem 時，操作 (在此案例中為最終一致讀取) 會以下列方式繼續。



1. DAX 用戶端發出 `GetItem` 請求。DAX 嘗試從項目快取讀取請求的項目。如果項目位於快取中 (「快取命中」)，則 DAX 會將其傳回應用程式。
2. 如果項目無法使用 (「快取未中」)，則 DAX 會對 DynamoDB 執行最終一致的 `GetItem` 操作。
3. DynamoDB 會傳回所請求的項目，而 DAX 會將其存放在項目快取中。
4. DAX 將項目傳回應用程式。
5. (未顯示) 如果 DAX 叢集包含超過一個節點，則會將項目複寫至叢集中的所有其他節點。

根據快取的存留時間 (TTL) 設定及最近最少使用 (LRU) 演算法，項目會保留在 DAX 項目快取中。如需詳細資訊，請參閱DAX 的運作方式。

但是，在此期間，DAX 將不會從 DynamoDB 重新讀取項目。如果有人使用 DynamoDB 用戶端來更新項目，完全略過 DAX，則使用 DynamoDB 用戶端的 `GetItem` 請求所產生的結果會與使用 `GetItem` 用戶端的相同請求不同。在此情況下，除非 DAX 項目的 TTL 過期，否則 DAX 和 DynamoDB 將會針對相同索引鍵保留不一致的值。

如果應用程式修改基礎 DynamoDB 資料表中的資料，略過 DAX，則應用程式需要預期和容忍可能發生的資料不一致。

> **Note**
>
> 除了 GetItem，DAX 用戶端還支援 BatchGetItem 請求。基本上 BatchGetItem 是一或多個 GetItem 請求的包裝函式，因此 DAX 會將每個請求視為個別的 GetItem 操作。

寫入一致性

DAX 是全部寫入快取，可簡化讓 DAX 項目快取與基礎 DynamoDB 資料表保持一致的程序。

DAX 用戶端支援與 DynamoDB 相同的寫入 API 操作 (PutItem、UpdateItem、DeleteItem、BatchWriteItem 和 TransactWriteItems)。當您搭配 DAX 用戶端使用這些操作時，項目會同時在 DAX 和 DynamoDB 中修改。DAX 會更新其項目快取中的項目，無論這些項目的 TTL 值為何。

例如，假設您從 DAX 用戶端發出 GetItem 請求，從 ProductCatalog 資料表讀取項目。(分割區索引鍵是 Id；沒有排序索引鍵。) 您擷取了 Id 為 101 的項目。該項目的 QuantityOnHand 值是 42。DAX 會將項目存放在其項目快取中，並具有特定的 TTL。針對此範例，假設 TTL 為 10 分鐘。然後，在 3 分鐘後，另一個應用程式使用 DAX 用戶端來更新相同的項目；因此，該項目的 QuantityOnHand 值現在是 41。假設未再次更新項目，則在下個 10 分鐘期間，任何後續讀取相同的項目都會傳回已快取的 QuantityOnHand 值 (41)。

DAX 如何處理寫入

DAX 適用於需要高效能讀取的應用程式。身為全部寫入快取，DAX 會同步將您的全部寫入傳遞至 DynamoDB，然後自動地、非同步地將產生的更新複寫到叢集中所有節點的項目快取。您不需要管理快取失效邏輯，因為 DAX 會為您自動進行處理。

DAX 支援下列寫入操作：PutItem、UpdateItem、DeleteItem、BatchWriteItem 和 TransactWriteItems。

當您將 PutItem、UpdateItem、DeleteItem 或 BatchWriteItem 請求傳送至 DAX 時，會執行下列作業：

- DAX 會將請求傳送至 DynamoDB。
- DynamoDB 回覆 DAX，確認寫入成功。
- DAX 將項目寫入至其項目快取。
- DAX 將成功傳回給申請者。

當您將 TransactWriteItems 請求傳送至 DAX 時，會執行下列作業：

- DAX 會將請求傳送至 DynamoDB。
- DynamoDB 回覆 DAX，確認交易完成。
- DAX 將成功傳回給申請者。
- 在背景中，DAX 會對 TransactGetItems 請求中的每個項目進行 TransactWriteItems 請求，以將項目存放在項目快取。TransactGetItems 會用來確保可序列化隔離。

若因為任何原因導致對 DynamoDB 的寫入失敗 (包括限流)，則項目便不會在 DAX 中進行快取。故障異常會傳回給申請者。除非是第一次將資料成功寫入 DAX，否則這會確保不會將資料寫入 DynamoDB 快取。

> **Note**
>
> 每個對 DAX 進行的寫入都會改變項目快取的狀態。但是，寫入項目快取不會影響查詢快取。(DAX 項目快取和查詢快取的用途不同，並且會各自獨立操作。)

DAX 查詢快取行為

DAX 會快取從 Query 獲得的結果，並在其查詢快取中進行 Scan 請求。但是，這些結果完全不會影響項目快取。在應用程式使用 DAX 發出 Query 或 Scan 請求後，結果集會存放於查詢快取而非項目快取中。您無法執行 Scan 操作來「預熱」項目快取，因為項目快取和查詢快取是不同的實體。

查詢-更新-查詢的一致性

項目快取的更新或基礎 DynamoDB 資料表的更新不會讓查詢快取中所存放的結果失效，或對其進行修改。

為了說明，請考慮以下情況。應用程式正在使用 DocumentRevisions 資料表，該資料表具有分割區索引鍵 DocId 和排序索引鍵 RevisionNumber。

1. 用戶端針對 RevisionNumber 大於或等於 5 的所有項目，發出 DocId 101 的 Query。DAX 會將結果集存放在查詢快取中，然後將結果集傳回給使用者。
2. 用戶端針對 RevisionNumber 為 20 的 DocId 101 發出 PutItem 請求。
3. 用戶端發出與步驟 1 所述相同的 Query (DocId 101 以及 RevisionNumber >= 5)。

在此案例中，步驟 3 發出的 Query 快取結果集會和步驟 1 快取的結果集完全相同。原因是，DAX 不會根據對個別項目的更新，使 Query 或 Scan 結果集無效。步驟 2 的 PutItem 操作只有在 Query 的 TTL 過期時，才會反映在 DAX 查詢快取中。

您的應用程式應該考慮查詢快取的 TTL 值，以及應用程式可以容忍查詢快取與項目快取之間不一致結果多長的時間。

強烈一致和交易讀取

若要執行強烈一致 GetItem、BatchGetItem、Query 或 Scan 請求，您必須將 ConsistentRead 參數設為 true。 DAX 會將高度一致性讀取請求傳遞至 DynamoDB。收到來自 DynamoDB 的回應時，DAX 會將結果傳回用戶端，但不會快取結果。DAX 無法提供高度一致性讀取，因為它與 DynamoDB 無法緊密結合。基於此原因，任何從 DAX 進行的後續讀取都必須是最終一致讀取。任何後續的高度一致性讀取都必須傳遞至 DynamoDB。

DAX 處理 TransactGetItems 請求的方式與處理高度一致性讀取相同。DAX 會將所有 TransactGetItems 請求傳遞至 DynamoDB。收到來自 DynamoDB 的回應時，DAX 會將結果傳回用戶端，但不會快取結果。

負快取

在項目快取和查詢快取中，DAX 都支援負快取項目。「負快取項目」會在 DAX 於基礎 DynamoDB 資料表中找不到所請求項目時發生。DAX 會快取空的結果，並將該結果傳回使用者，而不是產生錯誤。

例如，假設應用程式將 GetItem 請求傳送至 DAX 叢集，而 DAX 項目快取中沒有相符項目。這會讓 DAX 從基礎 DynamoDB 資料表中讀取對應的項目。如果項目不存在於 DynamoDB，則 DAX 會將空項目存放至其項目快取，然後將空項目傳回應用程式。現在假設應用程式傳送同一項目的另一個 GetItem 請求。DAX 會找到項目快取中的空項目，並將它立即傳回給應用程式。它完全不會參考 DynamoDB。

負快取項目將會保留在 DAX 項目快取中，直到其項目 TTL 過期、調用 LRU，或是使用 PutItem、UpdateItem 或 DeleteItem 來修改項目。

DAX 查詢快取會以類似的方式處理負快取結果。如果應用程式執行 Query 或 Scan，而且 DAX 查詢快取未包含已快取的結果，則 DAX 會將請求傳送給 DynamoDB。如果結果集中沒有相符項目，則 DAX 會將空結果集存放至查詢快取，並將空結果集傳回應用程式。除非該結果集的 TTL 過期，否則後續的 Query 或 Scan 請求將會產生相同的 (空) 結果集。

寫入策略

DAX 的全部寫入行為適用於許多應用程式模式。但是，有些應用程式模式不適用全部寫入模式。

針對對延遲相當敏感的應用程式，全部寫入 DAX 會造成額外的網路跳轉。因此寫入 DAX 會比直接寫入 DynamoDB 來得稍慢。若您的應用程式對寫入延遲相當敏感，您可以透過改為直接寫入 DynamoDB 來降低延遲。如需詳細資訊，請參閱繞過式寫入。

針對寫入密集應用程式 (例如，執行大量資料載入的應用程式)，您可能會不想要透過 DAX 寫入所有資料，因為應用程式只會讀取該資料的極小部分。當您透過 DAX 寫入大量資料時，必須調用其 LRU 演算法來清出快取中的空間，以讀取新項目。這會降低 DAX 作為讀取快取的有效性。

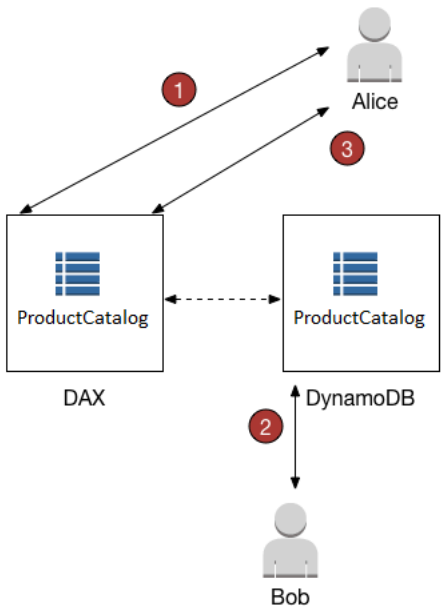
當您將項目寫入 DAX 時，項目快取狀態會變更，以容納新的項目。(例如，DAX 可能需要移出項目快取中較舊的資料，以清出空間來放置新項目。) 新項目將會根據快取的 LRU 演算法以及 TTL 設定保留在項目快取中。只要項目持續存在於項目快取中，DAX 就不會從 DynamoDB 重新讀取項目。

全部寫入

DAX 項目快取會實作全部寫入政策。如需詳細資訊，請參閱DAX 如何處理寫入。

當您寫入項目時，DAX 會確保已快取的項目與存在於 DynamoDB 中的項目同步。針對在寫入項目之後需要立即重新讀取項目的應用程式，這十分有用。不過，如果其他應用程式直接寫入 DynamoDB 資料表，則 DAX 項目快取中的項目將不再與 DynamoDB 同步。

為了示範，請考慮正在使用 `ProductCatalog` 資料表的兩位使用者 (Alice 和 Bob)。Alice 使用 DAX 存取資料表，但 Bob 略過 DAX，並在 DynamoDB 中直接存取資料表。



1. Alice 更新中的 ProductCatalog 資料表中的項目。DAX 將請求轉送至 DynamoDB，且更新成功。DAX 接著將項目寫入至其項目快取，並將成功回應傳回給 Alice。從這個時間點開始，除非最後從快取中移出項目，否則任何從 DAX 讀取項目的使用者都會看到具有 Alice 更新的項目。
2. 稍後，Bob 更新了 Alice 寫入的相同 ProductCatalog 項目。不過，Bob 直接在 DynamoDB 中更新項目。DAX 不會回應透過 DynamoDB 進行的更新，來重新整理其項目快取。因此，DAX 使用者將不會看到 Bob 的更新。
3. Alice 再次從 DAX 中讀取項目。項目位於項目快取中，因此 DAX 會將它傳回 Alice，而不會存取 DynamoDB 資料表。

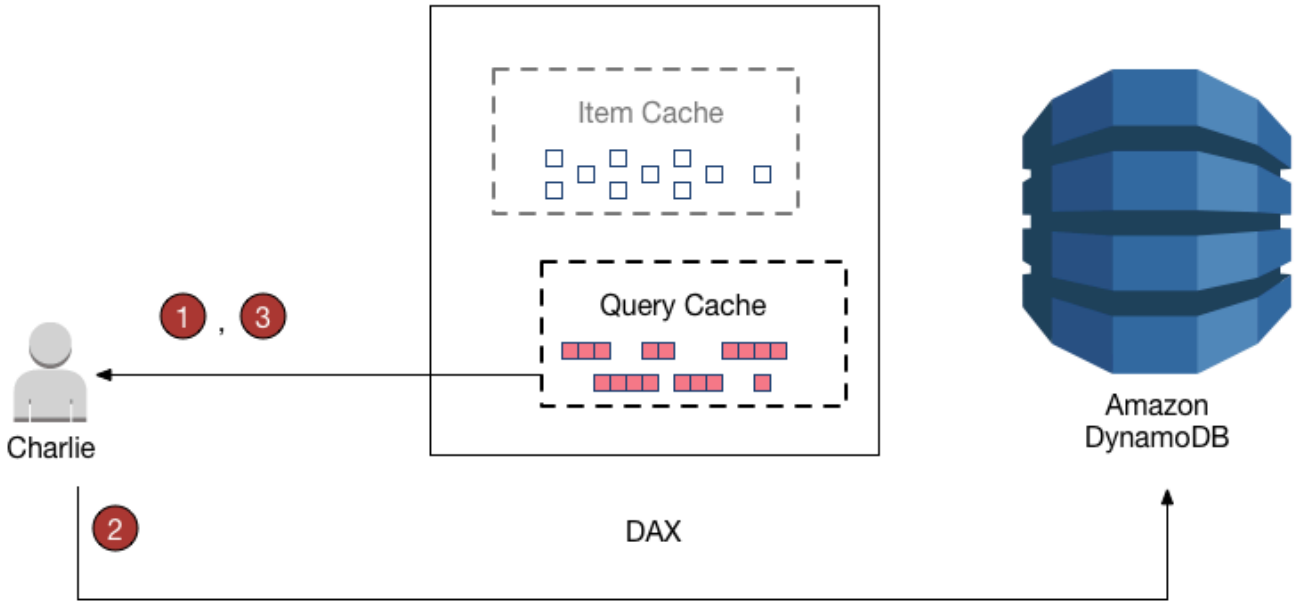
在此情況下，Alice 和 Bob 會看到相同 ProductCatalog 項目的不同呈現。除非 DAX 從項目快取中移出項目，或另一位使用者使用 DAX 再次更新相同的項目，否則就會是這種情況。

繞過式寫入

如果您的應用程式需要寫入大量資料 (例如大量資料載入)，則略過 DAX，將資料直接寫入至 DynamoDB 可能會相當合理。這種「繞過式寫入」策略可減少延遲。但是，項目快取將不會和 DynamoDB 中的資料保持同步。

如果您決定使用繞過式寫入策略，請記住，只要應用程式使用 DAX 用戶端讀取資料，DAX 就會填入其項目快取。這在某些情況下十分有益，因為它確定只會快取最常讀取的資料 (與最常寫入的資料相反)。

例如，請考慮想要使用 DAX 處理不同資料表 (GameScores 資料表) 的使用者 (Charlie)。GameScores 的分割區索引鍵是 UserId；因此，所有 Charlie 的分數都會有相同的 UserId。



1. Charlie 想要擷取其所有的分數，因此他傳送 `Query` 至 DAX。假設先前從未發出過此查詢，DAX 會將查詢轉送至 DynamoDB 進行處理。它會將結果存放在 DAX 查詢快取中，然後將結果傳回給 Charlie。除非移出結果集，否則結果集仍然會位於查詢快取中。
2. 現在假設 Charlie 玩 Meteor Blasters 遊戲，並得到高分。Charlie 將 `UpdateItem` 請求傳送至 DynamoDB，修改 `GameScores` 資料表中的項目。
3. 最後，Charlie 決定重新執行他稍早發出的 `Query` 來從 `GameScores` 擷取他的所有資料。在結果中，Charlie 看不到他在 Meteor Blasters 中的高分。原因是查詢結果來自查詢快取，而不是項目快取。兩個快取彼此獨立，因此其中一個快取中的變更不會影響另一個快取。

DAX 不會使用 DynamoDB 中的最新資料來重新整理查詢快取中的結果集。查詢快取中的每個結果集，其狀態都會是執行 `Query` 或 `Scan` 操作時的狀態。因此，Charlie 的 `Query` 結果不會反映其 `PutItem` 操作。除非 DAX 從查詢快取中移出結果集，否則就會是這種情況。

使用 DynamoDB Accelerator (DAX) 用戶端開發

若要從應用程式使用 DAX，您可以針對程式設計語言使用 DAX 用戶端。DAX 用戶端能對現有 Amazon DynamoDB 應用程式造成最少的干擾，您只需要對程式碼進行一些簡單修改。

> **Note**
>
> 適用於各種程式設計語言的 DAX 用戶端可在以下網站取得：
>
> - http://dax-sdk.s3-website-us-west-2.amazonaws.com

本節示範如何在預設 Amazon VPC 中啟動 Amazon EC2 執行個體、連線至執行個體，以及執行範例應用程式。本節也提供一些資訊，指導如何修改現有應用程式，好讓它可以使用 DAX 叢集。

主題

- 教學課程：使用 DynamoDB Accelerator (DAX) 執行範例應用程式
- 修改現有應用程式以使用 DAX

教學課程：使用 DynamoDB Accelerator (DAX) 執行範例應用程式

本教學課程示範如何在預設的虛擬私有雲端 (VPC) 中啟動 Amazon EC2 執行個體、連線至執行個體，以及使用 Amazon DynamoDB Accelerator (DAX) 執行應用程式範例。

> **Note**
>
> 為完成此教學課程，您預設的 VPC 中必須具備執行中的 DAX 叢集。若您尚未建立 DAX 叢集，請參閱 建立 DAX 叢集 來取得說明。

主題

- 步驟 1：啟動 Amazon EC2 執行個體
- 步驟 2：建立使用者和政策
- 步驟 3：設定 Amazon EC2 執行個體
- 步驟 4：執行範例應用程式

步驟 1：啟動 Amazon EC2 執行個體

您的 Amazon DynamoDB Accelerator (DAX) 叢集可以使用時，就可在預設的 Amazon Virtual Private Cloud (Amazon VPC) 中啟動 Amazon EC2 執行個體。您接著便可以在該執行個體上安裝及執行 DAX 用戶端軟體。

啟動 EC2 執行個體

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/ec2/：// 開啟 Amazon EC2 主控台。
2. 選擇 Launch Instance (啟動執行個體) 並執行下列作業：

   步驟 1：選擇 Amazon Machine Image (AMI)

   1. 在 AMI 清單中，找到 Amazon Linux AMI 並選擇 Select (選取)。

   步驟 2：選擇執行個體類型

   1. 在執行個體類型的清單中，選擇 t2.micro。
   2. 選擇 Next: Configure Instance Details (下一步：設定執行個體詳細資訊)。

   步驟 3：設定執行個體詳細資訊

   1. 前往 Network (網路)，然後選擇您的預設 VPC。
   2. 選擇 Next: Add Storage (下一步：新增儲存體)。

   步驟 4：新增儲存體

   1. 選擇 Next: Add Tags (下一步：新增標籤) 以跳過此步驟。

   步驟 5：新增標籤

   1. 選擇 Next: Configure Security Group (下一步：設定安全群組) 以跳過此步驟。

   步驟 6：設定安全群組

   1. 選擇 Select an existing security group (選取現有的安全群組)。
   2. 在安全群組清單中，選擇 default (預設)。這是您 VPC 的預設安全群組。
   3. 選擇 Next: Review and Launch (下一步：檢閱和啟動)。

   步驟 7：檢閱執行個體啟動

1. 選擇啟動。

3. 在 Select an existing key pair or create a new key pair (選取現有的金鑰對或建立新的金鑰對) 視窗中，執行下列其中一項作業：

   - 如果您沒有 Amazon EC2 金鑰對，請選擇 Create a new key pair (建立新的金鑰對) 並依照指示進行。系統會要求您下載私有金鑰檔案 (`.pem` 檔案)。您在稍後登入 Amazon EC2 執行個體時會需要此檔案。
   - 若您已擁有 Amazon EC2 金鑰對，請前往 Select a key pair (選取金鑰對)，然後從清單中選擇您的金鑰對。您必須已具備可用的私有金鑰檔案 (`.pem` 檔案) 才能登入您的 Amazon EC2 執行個體。

4. 當設定好您的金鑰對後，請選擇 Launch Instances (啟動執行個體)。

5. 在主控台導覽窗格中，選擇 EC2 Dashboard (EC2 儀表板)，然後選擇您啟動的執行個體。在下方窗格的 Description (描述) 標籤上，找到您執行個體的 Public DNS (公有 DNS)，例如：`ec2-11-22-33-44.us-west-2.compute.amazonaws.com`。請記下此公有 DNS 名稱，因為您在 步驟 3：設定 Amazon EC2 執行個體 中將需要它。

> **Note**
>
> Amazon EC2 執行個體需要幾分鐘的時間才會變成可用。同時，請繼續進行 步驟 2：建立使用者和政策，並遵循其中的說明。

步驟 2：建立使用者和政策

在此步驟中，您會使用授予 Amazon DynamoDB Accelerator (DAX) 叢集和 DynamoDB 存取權的政策來建立使用者 AWS Identity and Access Management。您接著便可以執行與您的 DAX 叢集互動的應用程式。

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊 後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者AWS 管理主控台身分登入 。在下一頁中，輸入您的密碼。

   如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。
2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需說明，請參閱《IAM 使用者指南》中的為您的 AWS 帳戶 根使用者 （主控台） 啟用虛擬 MFA 裝置。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的啟用 AWS IAM Identity Center。
2. 在 IAM Identity Center 中，將管理存取權授予使用者。

   如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱AWS IAM Identity Center《 使用者指南》中的使用預設值設定使用者存取 IAM Identity Center 目錄。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

  如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《 使用者指南》中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的建立許可集。
2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的新增群組。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 中的使用者和群組 AWS IAM Identity Center：

  建立權限合集。請按照《AWS IAM Identity Center 使用者指南》中的建立權限合集說明進行操作。
- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。遵循《IAM 使用者指南》的為第三方身分提供者 (聯合) 建立角色中的指示。
- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照《IAM 使用者指南》的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循《IAM 使用者指南》的新增許可到使用者 (主控台) 中的指示。

若要使用 JSON 政策編輯器來建立政策

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在左側的導覽窗格中，選擇 Policies (政策)。

   如果這是您第一次選擇 Policies (政策)，將會顯示 Welcome to Managed Policies (歡迎使用受管政策) 頁面。選擇 Get Started (開始使用)。

3. 在頁面頂端，選擇 Create policy (建立政策)。
4. 在政策編輯器中，選擇 JSON 選項。
5. 輸入或貼上 JSON 政策文件。如需有關 IAM 政策語言的詳細資訊，請參閱 IAM JSON 政策參考。
6. 解決政策驗證期間產生的任何安全性警告、錯誤或一般性警告，然後選擇 Next (下一步)。

> **Note**
>
> 您可以隨時切換視覺化與 JSON 編輯器選項。不過，如果您進行變更或在視覺化編輯器中選擇下一步，IAM 就可能會調整您的政策結構，以便針對視覺化編輯器進行最佳化。如需詳細資訊，請參閱《IAM 使用者指南》中的調整政策結構。

7. （選用）當您在 中建立或編輯政策時 AWS 管理主控台，您可以產生可在 範本中使用的 JSON 或 YAML 政策 CloudFormation 範本。

   若要執行此動作，請在政策編輯器中選擇動作，然後選擇產生 CloudFormation 範本。若要進一步了解 CloudFormation，請參閱AWS CloudFormation 《 使用者指南》中的AWS Identity and Access Management 資源類型參考。
8. 將許可新增至政策後，請選擇下一步。
9. 在檢視與建立頁面上，為您在建立的政策輸入政策名稱與描述 (選用)。檢視此政策中定義的許可，來查看您的政策所授予的許可。
10. (選用) 藉由連接標籤作為鍵值組，將中繼資料新增至政策。如需在 IAM 中使用標籤的詳細資訊，請參閱《IAM 使用者指南》中的AWS Identity and Access Management 資源的標籤。
11. 選擇 Create policy (建立政策) 儲存您的新政策。

政策文件：複製並貼上以下文件以建立 JSON 政策。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
```

```
                "*"
            ]
        },
        {
            "Action": [
                "dynamodb:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

步驟 3：設定 Amazon EC2 執行個體

當您的 Amazon EC2 執行個體可用時，您可以登入執行個體並準備使用它。

> **Note**
>
> 下列步驟假設您從執行 Linux 的電腦連線至 Amazon EC2 執行個體。如需其他連線方式，請參閱《Amazon EC2 使用者指南》中的連線至 Linux 執行個體。

設定 EC2 執行個體

1. 前往 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 使用 ssh 命令登入您的 Amazon EC2 執行個體，如以下範例所示。

   ```
   ssh -i my-keypair.pem ec2-user@public-dns-name
   ```

   您需要指定您的私有金鑰檔案 (.pem 檔案) 和您執行個體的公有 DNS 名稱。(請參閱 步驟 1：啟動 Amazon EC2 執行個體)。

   登入 ID 為 ec2-user。不需要任何密碼。
3. 登入 EC2 執行個體後，請設定您的 AWS 登入資料，如下所示。輸入您的 AWS 存取金鑰 ID 和私密金鑰 (來自 步驟 2：建立使用者和政策)，並將預設區域名稱設定為您目前的區域。(在以下範例中，預設區域名稱為 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]:
```

啟動並設定好 Amazon EC2 執行個體後，您可以使用其中一個提供的範例應用程式測試 DAX 的功能。如需詳細資訊，請參閱步驟 4：執行範例應用程式。

步驟 4：執行範例應用程式

為協助您測試 Amazon DynamoDB Accelerator (DAX) 的功能，您可以在 Amazon EC2 執行個體上執行其中一個提供的應用程式範例。

主題

- Node.js 和 DAX
- 適用於 Go 的 DAX 軟體開發套件
- Java 與 DAX
- .NET 和 DAX
- Python 和 DAX

Node.js 和 DAX

Node.js 的預設用戶端組態

設定 DAX JavaScript SDK 用戶端時，您可以自訂各種參數，以最佳化效能、連線處理和錯誤恢復能力。下表概述控制用戶端與 DAX 叢集互動方式的預設組態設定，包括逾時值、重試機制、憑證管理和運作狀態監控選項。如需詳細資訊，請參閱 DynamoDBClient 操作。

DAX JS SDK 用戶端預設值

參數	類型	說明
`region` 選擇性	`string`	AWS 區域 用於 DAX 用戶端的（範例 - 'us-east-1'）。如果未透過環境變數提供，則此為必要參數。

參數	類型	說明
endpoint 必要	string	軟體開發套件所連線叢集的端點。 範例： 未加密：dax-cluster-name.region.amazonaws.com 加密：daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
requestTimeout 預設 6000 毫秒	number	這用於定義用戶端等待 DAX 回應的時間上限。
writeRetries 預設 1	number	寫入請求失敗時的重試次數。
readRetries 預設 1	number	讀取請求失敗時的重試次數。
maxRetries 預設 1	number	請求失敗時的重試次數上限。 如果已設定 readRetries/writeRetries，則 readRetries 和 writeRetries 中設定的組態會優先於 maxRetries。
connectTimeout 預設 10000 毫秒	number	建立任何叢集節點連線的逾時值 (以毫秒為單位)。
maxConcurrentConnections 預設 100	number	限制用戶端執行個體可在 DAX 叢集中為每個節點建立的並行連線總數。

參數	類型	說明
`maxRetryDelay` 預設 7000 毫秒	`number`	DAX 伺服器將 `waitForRecoveryBeforeRetrying` 旗標設定為 true 表示需要復原時，用戶端會先暫停然後再重試。在這些復原期間，`maxRetryDelay` 參數會決定重試之間的最長等待時間。此復原專屬組態僅適用於處於復原模式的 DAX 伺服器。對於所有其他情境，重試行為會遵循兩種模式之一：根據重試計數的指數延遲 (由 `writeRetries` 、`readRetries` 、或 `maxRetries` 參數控管)，或根據例外狀況類型立即重試。
`credentials` 選擇性	`AwsCredentialIdentity` | `AwsCredentialIdentityProvider`	用於驗證請求的 AWS 登入資料。這能夠以 AwsCredentialIdentity 或 AwsCredentialIdentityProvider 的形式提供。如果未提供，AWS 軟體開發套件會自動使用預設憑證供應者鏈結。範例：`{ accessKeyId: 'AKIA...', secretAccessKey: '...', sessionToken: '...' }` * @default 使用預設 AWS 憑證供應者鏈結。
`healthCheckInterval` 預設 5000 毫秒	`number`	叢集運作狀態檢查之間的間隔 (以毫秒為單位)。間隔時間越短，檢查頻率就更高。

參數	類型	說明
healthCheckTimeout 預設 1000 毫秒	number	完成運作狀態檢查的逾時 (以毫秒為單位)。
skipHostnameVerification 預設 false	boolean	略過 TLS 連線的主機名稱驗證。這不會影響未加密叢集。預設為執行主機名稱驗證，將此設定為 True 會略過驗證。請務必了解將其關閉的意涵，也就是無法驗證您連線的叢集。
unhealthyConsecutiveErrorCount 預設 5	number	設定在運作狀態檢查間隔內向節點發出運作狀態不佳訊號所需的連續錯誤次數。
clusterUpdateInterval 預設 4000 毫秒	number	傳回輪詢叢集成員以進行成員資格變更之間的間隔。
clusterUpdateThreshold 預設 125	number	傳回低於閾值的叢集不會接受輪詢進行成員資格變更。
credentailProvider 選用 | 預設 null	AwsCredentialIdentityProvider	用於驗證 DAX 請求之 AWS 登入資料的使用者定義提供者。

DaxDocument 的分頁組態

名稱	類型	詳細資訊
client	DaxDocument	DaxDocument 類型的執行個體。
pageSize	number	決定每頁的項目數。

名稱	類型	詳細資訊
`startingToken` 選用	any	先前回應的 LastEvaluatedKey 可用於後續請求。

如需了解分頁的使用方式，請參閱 the section called “TryDax.js”。

移轉至 DAX Node.js SDK V3

此移轉指南將協助您轉換現有的 DAX Node.js 應用程式。新的 SDK 需要 Node.js 18 或更高版本，並在建構 DynamoDB Accelerator 程式碼方面引入幾項重要變更。本指南將逐步解說其中的主要差異，包括語法變更、新的匯入方法，以及更新的非同步程式設計模式。

使用 V2 Node.js DAX

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var client = new AWS.DynamoDB.DocumentClient();

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  client = new AWS.DynamoDB.DocumentClient({ service: dax });
}

// Make Get Call using Dax
var params = {
    TableName: 'TryDaxTable',
    pk: 1,
    sk: 1
}
client.get(params, function (err, data) {
```

```
    if (err) {
        console.error(
            "Unable to read item. Error JSON:",
            JSON.stringify(err, null, 2)
          );
    } else {
        console.log(data);
    }
});
```

使用 V3 Node.js DAX

使用 DAX 時建議採用 Node.js V3 Node 版本 18 或更高版本。若要移至 Node 18，請使用下列項目：

```
import { DaxDocument } from '@amazon-dax-sdk/lib-dax';
import { DynamoDBDocument } from '@aws-sdk/lib-dynamodb';
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';

let client: DynamoDBDocument | DaxDocument = DynamoDBDocument.from(
  new DynamoDBClient({ region: 'us-west-2' })
);

if (process.argv.length > 2) {
  client = new DaxDocument({
    endpoints: [process.argv[2]],
    region: 'us-west-2',
  });
}

const params = {
  TableName: 'TryDaxTable',
  Key: { pk: 1, sk: 1 },
};

try {
  const results = await client.get(params);
  console.log(results);
} catch (err) {
  console.error(err);
}
```

DAX SDK for Node.js v3.x 相容於 AWS SDK for Node.js v3.x。DAX SDK for Node.js v3.x 支援使用彙總用戶端。請注意，DAX 不支援建立裸機用戶端。如需不支援功能的詳細資訊，請參閱 the section called “功能與 AWS SDK V3 不相同”。

請按照此程序操作，在 Amazon EC2 執行個體上執行 Node.js 範例應用程式。

執行 DAX 的 Node.js 範例

1. 在您的 Amazon EC2 執行個體上設定 Node.js，如下所示：

   a. 安裝節點版本管理工具 (nvm)。

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
   ```

   b. 使用 nvm 安裝 Node.js。

   ```
   nvm install 18
   ```

   c. 使用 nvm 來使用 Node 18

   ```
   nvm use 18
   ```

   d. 測試 Node.js 已安裝且正常運作。

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   這應該會顯示以下訊息。

   ```
   Running Node.js v18.x.x
   ```

2. 使用節點套件管理工具 (npm) 安裝 DaxDocument Node.js 用戶端。

   ```
   npm install @amazon-dax-sdk/lib-dax
   ```

TryDax 範本程式碼

若要評估 DynamoDB Accelerator (DAX) 的效能優勢，請依照下列步驟執行範例測試，以比較標準 DynamoDB 和 DAX 叢集之間的讀取操作時間。

1. 在設定工作空間並將 `lib-dax` 安裝為依存項目之後，請將 the section called “TryDax.js” 複製到您的專案中。

2. 針對 DAX 叢集執行程式。若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

   - 使用 DynamoDB 主控台：選擇您的 DAX 叢集。叢集端點會在主控台上顯示，如以下範例。

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - 使用 AWS CLI ：輸入下列命令。

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     如下列範例所示，叢集端點會在輸出上顯示。

     ```
     {
         "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. 現在請將叢集端點指定為命令列參數以執行程式。

   ```
   node TryDax.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
   ```

   您應該會看到類似下列的輸出：

   ```
   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.
```

請記下計時資訊。GetItem、Query、Scan 測試所需的微秒數。

4. 在此情況下，您會針對 DAX 叢集執行程式。現在您將再次執行該程式，但此次針對的是 DynamoDB。
5. 現在重新執行程式，但這一次叢集端點 URL 並沒有作為命令列參數。

```
node TryDax.js
```

查看輸出，並記下計時資訊。使用 DAX 的 `GetItem`、`Query` 和 `Scan` 已耗用時間應遠低於使用 DynamoDB 的已耗用時間。

功能與 AWS SDK V3 不相同

- 裸機用戶端：Dax Node.js V3 不支援裸機用戶端。

```
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);
```

- 中介軟體堆疊：Dax Node.js V3 不支援使用中介軟體函式。

```
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();
})();
```

適用於 Go 的 DAX 軟體開發套件

請按照此程序操作，在 Amazon EC2 執行個體上執行 Amazon DynamoDB Accelerator (DAX) 適用於 Go 的開發套件範例應用程式。

為 DAX 執行適用於 Go 的開發套件範例

1. 在您的 Amazon EC2 執行個體上設定適用於 Go 的開發套件：

   a. 安裝 Go 程式設計語言 (Golang)。

   ```
   sudo yum install -y golang
   ```

   b. 測試 Golang 是否已安裝並正常運作。

   ```
   go version
   ```

   這類的訊息應會出現。

   ```
   go version go1.23.4 linux/amd64
   ```

2. 安裝範例 Golang 應用程式。

   ```
   go get github.com/aws-samples/sample-aws-dax-go-v2
   ```

3. 執行下列 Golang 程式。第一個程式會建立名為 TryDaxGoTable 的 DynamoDB 資料表。第二個程式會將資料寫入資料表。

   ```
   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. 執行下列 Golang 程式。

   ```
   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
```

記下計時資訊：GetItem、Query 和 Scan 測試所需要的毫秒數。

5. 在先前的步驟中，您已針對 DynamoDB 端點執行程式。現在，請再次執行程式，但這一次 GetItem、Query 和 Scan 操作會由您的 DAX 叢集處理。

   若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

   - 使用 DynamoDB 主控台：選擇您的 DAX 叢集。叢集端點會在主控台上顯示，如以下範例。

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - 使用 AWS CLI ：輸入下列命令。

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     如下列範例所示，叢集端點會在輸出上顯示。

     ```
     {
         "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"
     }
     ```

   現在重新執行程式，但這一次，請將叢集端點做為命令列參數指定。

   ```
   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
```

查看輸出的剩餘部分，並記下計時資訊。使用 DAX 的 `GetItem`、`Query` 和 `Scan` 已耗用時間應遠低於使用 DynamoDB 的已耗用時間。

6. 執行以下 Golang 程式，刪除 `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
```

功能與 適用於 Go 的 AWS SDK V2 不相同

中介軟體堆疊：DAX Go V2 不支援透過 APIoptions 使用中介軟體堆疊。如需詳細資訊，請參閱使用 Middleware 自訂 適用於 Go 的 AWS SDK v2 用戶端請求。

範例：

```
// 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
}
```

輸出：

```
failed to execute command: custom middleware through APIOptions is not supported in DAX
 client
exit status 1
```

Go 的預設用戶端組態

本指南將逐步解說組態選項，協助您微調 DAX 用戶端效能、連線管理和日誌記錄行為。了解預設設定以及如何自訂這些設定後，您就可以最佳化 Go 應用程式與 DAX 的互動。

本節內容

- DAX Go SDK 用戶端預設值
- 建立用戶端

DAX Go SDK 用戶端預設值

參數	類型	說明
`Region` 必要	`string`	AWS 區域 用於 DAX 用戶端的 (example- 'us-east-1')。如果未透過環境提供，則此為必要參數。
`HostPorts` 必要	`[] string`	SDK 連線的 DAX 叢集端點清單。 例如： 未加密：dax：//my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com 加密：daxs：//my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
`MaxPendingConnectionsPerHost` 預設 10	`number`	並行連線嘗試次數 (可同時建立的連線。)
`ClusterUpdateThreshold` 預設 125 * time.Millisecond	`time.Duration`	叢集重新整理之間必須經過的最短時間。
`ClusterUpdateInterval` 預設 4 * time.Second	`time.Duration`	用戶端自動重新整理 DAX 叢集資訊的間隔。
`IdleConnectionsReapDelay`	`time.Duration`	用戶端在 DAX 用戶端中關閉閒置連線的間隔。

參數	類型	說明
預設 30 * time.Second		
`ClientHealthCheckInterval` 預設 5 * time.Second	`time.Duration`	用戶端在 DAX 叢集端點執行運作狀態檢查的間隔。
`Credentials` 預設	`aws.CredentialsProvider`	DAX 用戶端用來驗證 DAX 服務請求的 AWS 登入資料。請參閱憑證和憑證供應者。
`DialContext` 預設	`func`	DAX 用戶端用來建立 DAX 叢集連線的自訂函式。
`SkipHostnameVerification` 預設 false	bool	略過 TLS 連線的主機名稱驗證。此設定只會影響加密的叢集。設定為 True 時，會停用主機名稱驗證。停用驗證表示您無法驗證所連線叢集的身分，會造成安全風險。預設會啟用主機名稱驗證。
`RouteManagerEnabled` 預設 false	`bool`	此旗標用於移除面向網路的路由錯誤。
`RequestTimeout` 預設 60 * time.Second	`time.Duration`	這用於定義用戶端等待 DAX 回應的時間上限。 優先順序：內容逾時 (如果設定) > `RequestTimmeout` (如果設定) > 預設 60 秒 `RequestTimeout` 。
`WriteRetries` 預設 2	`number`	寫入請求失敗時的重試次數。

參數	類型	說明
`ReadRetries` 預設 2	`number`	讀取請求失敗時的重試次數。
`RetryDelay` 預設 0	`time.Duration`	請求失敗重試時的非限流錯誤延遲 (以秒為單位)。
`Logger` 選擇性	`logging.Logger`	Logger 是以日誌記錄特定分類項目的介面。
`LogLevel` 預設 utils.LogOff	`number`	此日誌層級僅針對 DAX 定義。您可以使用 github.com/aws/aws-dax-go-v2/tree/main/dax/utils 匯入。 `const ( LogOff LogLevelType = 0 LogDebug LogLevelType = 1 LogDebugWithRequestRetries LogLevelType = 2 )`

> **Note**
>
> 對於 `time.Duration`，預設單位為奈秒。如果我們沒有為任何參數指定任何單位，則會將其視為奈秒：`daxCfg.ClusterUpdateInterval = 10` 表示 10 奈秒。(`daxCfg.ClusterUpdateInterval = 10 * time.Millisecond` 表示 10 毫秒)。

建立用戶端

如欲建立 DAX 用戶端：

- 建立 DAX 組態，然後使用 DAX 組態建立 DAX 用戶端。您可使用此功能視需要覆寫 DAX 組態。

```
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)
```

移轉至 DAX Go SDK V2

此移轉指南將協助您轉換現有的 DAX Go 應用程式。

使用 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)
```

```
}
```

使用 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)
}
```

如需 API 使用詳細資訊，請參閱 AWS 範例。

Java 與 DAX

適用於 Java 2.x 的 DAX 軟體開發套件與適用於 Java 2.x 的AWS 軟體開發套件相容。它建立在 Java 8+ 之上，並包含對非封鎖 I/O 的支援。 如需搭配適用於 Java 的 AWS SDK 1.x 使用 DAX 的詳細資訊，請參閱 搭配適用於 Java 1.x 的 AWS 開發套件使用 DAX。

使用用戶端作為 Maven 依存項目

遵循這些步驟，在您的應用程式中將 DAX SDK for Java 用戶端做為依存項目使用。

1. 下載並安裝 Apache Maven。如需詳細資訊，請參閱下載 Apache Maven 和安裝 Apache Maven。
2. 將用戶端 Maven 依存項目新增至您應用程式的專案物件模型 (POM) 檔案。在此範例中，將 *x.x.x* 取代為用戶端的實際版本號碼。

```
<!--Dependency:-->
<dependencies>
    <dependency>
        <groupId>software.amazon.dax</groupId>
        <artifactId>amazon-dax-client</artifactId>
        <version>x.x.x</version>
    </dependency>
</dependencies>
```

TryDax 範本程式碼

在設定工作空間並將 DAX 軟體開發套件新增為依存項目之後，請將 TryDax.java 複製到您的專案中。

使用此命令運行程式碼。

```
java -cp classpath TryDax
```

您應該會看到類似下列的輸出。

```
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.
```

記下計時資訊：GetItem 和 Query 測試所需要的毫秒數。在此例中，您已針對 DynamoDB 端點執行程式。您將再次執行該程式，但此次針對的是您的 DAX 叢集。

若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

- 在 DynamoDB 主控台中，選取您的 DAX 叢集。叢集端點會在主控台中顯示，如以下範例。

  ```
  dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
  ```

- 使用 AWS CLI，輸入下列命令：

  ```
  aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
  ```

  叢集端點地址、連接埠和 URL 會在輸出中顯示，如以下範例所示。

  ```
  {
      "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"
  }
  ```

現在重新執行程式，但這一次，請將叢集端點 URL 作為命令列參數指定。

```
java -cp classpath TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

查看輸出，並記下計時資訊。使用 DAX 的 GetItem 和 Query 已耗用時間應遠低於使用 DynamoDB 的已耗用時間。

軟體開發套件指標

您可以使用適用於 Java 2.x 的 DAX 軟體開發套件，收集您應用程式中的服務用戶端指標並在 Amazon CloudWatch 中分析輸出。如需詳細資訊，請參閱啟用 SDK 指標。

> **Note**
>
> 適用於 Java 的 DAX 軟體開發套件僅會收集 `ApiCallSuccessful` 和 `ApiCallDuration` 指標。

TryDax.java

```
import java.util.Map;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.dax.ClusterDaxAsyncClient;
import software.amazon.dax.Configuration;

public class TryDax {
    public static void main(String[] args) throws Exception {
        DynamoDbAsyncClient ddbClient = DynamoDbAsyncClient.builder()
                .build();

        DynamoDbAsyncClient daxClient = null;
        if (args.length >= 1) {
            daxClient = ClusterDaxAsyncClient.builder()
                    .overrideConfiguration(Configuration.builder()
                            .url(args[0]) // e.g. dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com
                            .build())
                    .build();
        }

        String tableName = "TryDaxTable";
```

```
        System.out.println("Creating table...");
        createTable(tableName, ddbClient);

        System.out.println("Populating table...");
        writeData(tableName, ddbClient, 100, 10);

        DynamoDbAsyncClient testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }

        System.out.println("Running GetItem and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");

        // GetItem
        getItemTest(tableName, testClient, 100, 10, 5);

        // Query
        queryTest(tableName, testClient, 100, 2, 9, 5);

        System.out.println("Deleting table...");
        deleteTable(tableName, ddbClient);
    }

    private static void createTable(String tableName, DynamoDbAsyncClient client) {
        try {
            System.out.println("Attempting to create table; please wait...");

            client.createTable(CreateTableRequest.builder()
                    .tableName(tableName)
                    .keySchema(KeySchemaElement.builder()
                            .keyType(KeyType.HASH)
                            .attributeName("pk")
                            .build(), KeySchemaElement.builder()
                            .keyType(KeyType.RANGE)
                            .attributeName("sk")
                            .build())
                    .attributeDefinitions(AttributeDefinition.builder()
                            .attributeName("pk")
                            .attributeType(ScalarAttributeType.N)
                            .build(), AttributeDefinition.builder()
```

```
                            .attributeName("sk")
                            .attributeType(ScalarAttributeType.N)
                            .build())
                    .billingMode(BillingMode.PAY_PER_REQUEST)
                    .build()).get();
            client.waiter().waitUntilTableExists(DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            System.out.println("Successfully created table.");

        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    private static void deleteTable(String tableName, DynamoDbAsyncClient client) {
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            client.deleteTable(DeleteTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            client.waiter().waitUntilTableNotExists(DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
            e.printStackTrace();
        }
    }

    private static void writeData(String tableName, DynamoDbAsyncClient client, int
 pkmax, int skmax) {
        System.out.println("Writing data to the table...");

        int stringSize = 1000;
        StringBuilder sb = new StringBuilder(stringSize);
        for (int i = 0; i < stringSize; i++) {
            sb.append('X');
        }
        String someData = sb.toString();
```

```
        try {
            for (int ipk = 1; ipk <= pkmax; ipk++) {
                System.out.println(("Writing " + skmax + " items for partition key: " +
ipk));
                for (int isk = 1; isk <= skmax; isk++) {
                    client.putItem(PutItemRequest.builder()
                            .tableName(tableName)
                            .item(Map.of("pk", attr(ipk), "sk", attr(isk), "someData",
attr(someData)))
                            .build()).get();
                }
            }
        } catch (Exception e) {
            System.err.println("Unable to write item:");
            e.printStackTrace();
        }
    }

    private static AttributeValue attr(int n) {
        return AttributeValue.builder().n(String.valueOf(n)).build();
    }

    private static AttributeValue attr(String s) {
        return AttributeValue.builder().s(s).build();
    }

    private static void getItemTest(String tableName, DynamoDbAsyncClient client, int
pk, int sk, int iterations) {
        long startTime, endTime;
        System.out.println("GetItem test - partition key 1-" + pk + " and sort keys 1-"
+ sk);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            try {
                for (int ipk = 1; ipk <= pk; ipk++) {
                    for (int isk = 1; isk <= sk; isk++) {
                        client.getItem(GetItemRequest.builder()
                                .tableName(tableName)
                                .key(Map.of("pk", attr(ipk), "sk", attr(isk)))
                                .build()).get();
                    }
                }
            } catch (Exception e) {
```

```
                System.err.println("Unable to get item:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk * sk);
        }
    }

    private static void queryTest(String tableName, DynamoDbAsyncClient client, int pk,
int sk1, int sk2, int iterations) {
        long startTime, endTime;
        System.out.println("Query test - partition key 1-" + pk + " and sort keys
between " + sk1 + " and " + sk2);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            for (int ipk = 1; ipk <= pk; ipk++) {
                try {
                    // Pagination API for Query.
                    client.queryPaginator(QueryRequest.builder()
                            .tableName(tableName)
                            .keyConditionExpression("pk = :pkval and sk between :skval1
and :skval2")
                            .expressionAttributeValues(Map.of(":pkval", attr(ipk),
":skval1", attr(sk1), ":skval2", attr(sk2)))
                            .build()).items().subscribe((item) -> {
                    }).get();
                } catch (Exception e) {
                    System.err.println("Unable to query table:");
                    e.printStackTrace();
                }
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk);
        }
    }

    private static void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
(1000000.0));
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
1000000.0));
    }
```

```
}
```

.NET 和 DAX

遵循下列步驟，在 Amazon EC2 執行個體上執行 .NET 範例。

> **Note**
>
> 本教學課程使用 .NET 9 開發套件。本教學課程示範如何在預設的 Amazon VPC 中執行程式來存取 Amazon DynamoDB Accelerator (DAX) 叢集。它適用於AWS 適用於 .NET 的 SDK v4。如需 V4 變更的詳細資訊，以及遷移的相關資訊，請參閱遷移至適用於 .NET 的 AWS SDK 第 4 版。如果您願意，您可以使用 AWS Toolkit for Visual Studio 來撰寫 .NET 應用程式，並將其部署到您的 VPC。
> 如需詳細資訊，請參閱《AWS Elastic Beanstalk 開發人員指南》中的使用 AWS Toolkit for Visual Studio 在 .NET 中建立和部署 Elastic Beanstalk 應用程式。

執行 DAX 的 .NET 範例

1. 前往 Microsoft 下載頁面並下載適用於 Linux 的最新 .NET 9 開發套件。下載的檔案為 `dotnet-sdk-N.N.N-linux-x64.tar.gz`。
2. 將 SDK 檔案解壓縮。

   ```
   mkdir dotnet
   tar zxvf dotnet-sdk-N.N.N-linux-x64.tar.gz -C dotnet
   ```

   將 *N.N.N* 替換為 .NET SDK 的實際版本號碼 (例如：`9.0.305`)。
3. 驗證安裝。

   ```
   alias dotnet=$HOME/dotnet/dotnet
   dotnet --version
   ```

   這應該會列印出 .NET SDK 的版本號碼。

   > **Note**
   >
   > 您可能會收到以下錯誤 (而不是版本號碼)：
   > 錯誤：libunwind.so.8：無法開啟共享物件檔案：找不到檔案或目錄

若要解決錯誤，請安裝 libunwind 套件。

```
sudo yum install -y libunwind
```

在執行此作業後，您應該能夠執行 dotnet --version 命令，而不會發生任何錯誤。

4. 建立新的 .NET 專案。

```
dotnet new console -o myApp
```

需要花費幾分鐘來執行一次性設定。完成時，請執行範例專案。

```
dotnet run --project myApp
```

您應該會收到以下訊息：Hello World!

5. myApp/myApp.csproj 檔案包含您專案的中繼資料。若要在您的應用程式中使用 DAX 用戶端，請修改檔案，讓它看起來如下。

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net9.0</TargetFramework>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="AWSSDK.DAX.Client" Version="*" />
    </ItemGroup>
</Project>
```

6. 下載範例程式來源碼 (.zip 檔案)。

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

下載完成後，解壓縮來源檔案。

```
unzip TryDax.zip
```

7. 現在執行 dotNet 的範例程式，一次一個。針對每一個程式，將其內容複製到 myApp/Program.cs，然後執行 MyApp 專案。

執行以下 .NET 程式。第一個程式會建立名為 TryDaxTable 的 DynamoDB 資料表。第二個程式會將資料寫入資料表。

```
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. 接下來，請執行一些程式，以在您的 DAX 叢集上執行 GetItem、Query 和 Scan 操作。若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

   - 使用 DynamoDB 主控台：選擇您的 DAX 叢集。叢集端點會在主控台上顯示，如以下範例。

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - 使用 AWS CLI ：輸入下列命令。

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     如下列範例所示，叢集端點會在輸出上顯示。

     ```
     {
         "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"
     }
     ```

   現在執行下列程式，將您的叢集端點指定為命令列參數。(以您的實際 DAX 叢集端點取代範本端點。)

   ```
   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
```

記下計時資訊：GetItem、Query 和 Scan 測試所需要的毫秒數。

9. 執行以下 .NET 程式，以刪除 TryDaxTable。

```
cp TryDax/dotNet/06-DeleteTable.cs myApp/Program.cs
dotnet run --project myApp
```

如需這些程式的詳細資訊，請參閱下列各節：

- 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

01-CreateTable.cs 程式會建立資料表 (TryDaxTable)。本節剩餘的 .NET 程式呈現於此資料表。

```
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

02-Write-Data.cs 程式會將測試資料寫入 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

`03-GetItem-Test.cs` 程式會在 `GetItem` 上執行 `TryDaxTable` 操作。

```
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

04-Query-Test.cs 程式會在 Query 上執行 TryDaxTable 操作。

```
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

05-Scan-Test.cs 程式會在 Scan 上執行 TryDaxTable 操作。

```
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

06-DeleteTable.cs 程式會刪除 TryDaxTable。請在完成測試之後執行此程式。

```
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 和 DAX

請按照此程序操作，在 Amazon EC2 執行個體上執行 Python 範例應用程式。

執行 DAX 的 Python 範例

1. 使用 pip 公用程式安裝 DAX Python 用戶端。

```
pip install amazon-dax-client
```

2. 下載範例程式來源碼 (.zip 檔案)。

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

下載完成後，解壓縮來源檔案。

```
unzip TryDax.zip
```

3. 執行下列 Python 程式。第一個程式會建立名為 TryDaxTable 的 Amazon DynamoDB 資料表。第二個程式會將資料寫入資料表。

```
python 01-create-table.py
python 02-write-data.py
```

4. 執行下列 Python 程式。

```
python 03-getitem-test.py
python 04-query-test.py
python 05-scan-test.py
```

記下計時資訊：GetItem、Query 和 Scan 測試所需要的毫秒數。

5. 在先前的步驟中，您已針對 DynamoDB 端點執行程式。現在，請再次執行程式，但這一次 GetItem、Query 和 Scan 操作會由您的 DAX 叢集處理。

若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

- 使用 DynamoDB 主控台：選擇您的 DAX 叢集。叢集端點會在主控台上顯示，如以下範例。

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- 使用 AWS CLI ：輸入下列命令。

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

叢集端點會在輸出中顯示，如此範例所示。

```
{
    "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"
}
```

請重新執行程式，但這一次，請將叢集端點做為命令列參數指定。

```
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
```

查看輸出的剩餘部分，並記下計時資訊。使用 DAX 的 `GetItem`、`Query` 和 `Scan` 已耗用時間應遠低於使用 DynamoDB 的已耗用時間。

6. 執行以下 Python 程式，刪除 `TryDaxTable`。

```
python 06-delete-table.py
```

如需這些程式的詳細資訊，請參閱下列各節：

- 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

`01-create-table.py` 程式會建立資料表 (`TryDaxTable`)。本節剩餘的 Python 程式呈現於此資料表。

```
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

02-write-data.py 程式會將測試資料寫入 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

03-getitem-test.py 程式會在 GetItem 上執行 TryDaxTable 操作。此示例是針對 eu-west-1 區域。

```
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

04-query-test.py 程式會在 Query 上執行 TryDaxTable 操作。

```
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

05-scan-test.py 程式會在 Scan 上執行 TryDaxTable 操作。

```
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

06-delete-table.py 程式會刪除 TryDaxTable。請在完成測試 Amazon DynamoDB Accelerator (DAX) 功能之後執行此程式。

```
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!")
```

修改現有應用程式以使用 DAX

如果您已有使用 Amazon DynamoDB 的 Java 應用程式，則可以進行修改，使其可存取您的 DynamoDB Accelerator (DAX) 叢集。您不需要重寫整個應用程式，因為 DAX Java 用戶端類似於適用於 Java 的 AWS SDK 2.x 中包含的 DynamoDB 低階用戶端。如需更多詳細資訊，請參閱使用 DynamoDB 中的項目。

> **Note**
>
> 此範例使用適用於 Java 的 AWS SDK 2.x。如需適用於 Java 1.x 版本的舊式開發套件，請參閱修改適用於 Java 1.x 的開發套件的現有應用程式來使用 DAX。

若要修改您的程式，請將 DynamoDB 用戶端取代為 DAX 用戶端。

```
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();
```

您也可以使用屬於適用於 Java 的 AWS SDK 2.x 的高階程式庫，以 DAX 用戶端取代 DynamoDB 用戶端。

```
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();
```

如需詳細資訊，請參閱映射 DynamoDB 資料表中的項目。

管理 DAX 叢集

本節將介紹一些常見的 Amazon DynamoDB Accelerator (DAX) 叢集管理任務。

主題

- 管理 DAX 叢集的 IAM 許可
- 擴展 DAX 叢集
- 自訂 DAX 叢集設定
- 配置 TTL 設定
- DAX 的標記支援
- AWS CloudTrail 整合
- 刪除 DAX 叢集

管理 DAX 叢集的 IAM 許可

當您使用 AWS 管理主控台 或 AWS Command Line Interface (AWS CLI) 管理 DAX 叢集時，強烈建議您縮小使用者可以執行的動作範圍。這樣做有利於降低風險，同時遵循最低權限。

以下的討論著重於 DAX 管理 API 的存取控制。如需更細資訊，請參閱《Amazon DynamoDB API 參考》中的 Amazon DynamoDB Accelerator。

> **Note**
>
> 如需管理 AWS Identity and Access Management (IAM) 許可的詳細資訊，請參閱下列內容：
>
> - IAM 和建立 DAX 叢集：建立 DAX 叢集。
> - IAM 與 DAX 資料平面操作：DAX 存取控制。

針對 DAX 管理 API，您無法將 API 動作範圍侷限於某項特定資源。`Resource` 元素必須設為 `"*"`。這與 DAX 資料平面 API 操作不同，例如 `GetItem`、`Query` 和 `Scan`。資料平面操作是透過 DAX 用戶端公開，而且這些操作的範圍可以侷限於特定的資源。

為了示範，請考慮以下 IAM 政策文件。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

假設此政策的目的是允許叢集的 DAX 管理 API 呼叫 `DAXCluster01`，且僅限該叢集。

現在假設使用者發出下列 AWS CLI 命令。

```
aws dax describe-clusters
```

此命令會因為未授權異常而失敗，因為基礎 DescribeClusters API 呼叫範圍無法侷限於特定叢集。即使此政策在語法上有效，命令仍會失敗，因為 Resource 元素必須設為 "*"。但若使用者執行某個程式，將 DAX 資料平面呼叫 (例如 GetItem 或 Query) 傳送至 DAXCluster01，則這些呼叫會成功。這是因為 DAX 資料平面 API 的範圍可侷限在特定的資源 (此案例中為 DAXCluster01)。

如果您想要撰寫單一的全方位 IAM 政策，將 DAX 管理 API 和 DAX 資料平面 API 都包含在內，建議您在政策文件中包含兩個不同的陳述式。其中一個陳述式應該處理 DAX 資料平面 API，而另一個陳述式則處理 DAX 管理 API。

以下範例政策示範這種方法。請注意 DAXDataAPIs 陳述式的範圍如何侷限在 DAXCluster01 資源範圍內，但 DAXManagementAPIs 的資源必須是 "*"。每個陳述式中顯示的動作都僅供示範使用。您可以視需要為您的應用程式自訂他們。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXDataAPIs",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]},
            {
            "Sid": "DAXManagementAPIs",
            "Action": [
                "dax:CreateParameterGroup",
                "dax:CreateSubnetGroup",
                "dax:DecreaseReplicationFactor",
                "dax:DeleteCluster",
                "dax:DeleteParameterGroup",
```

```
                "dax:DeleteSubnetGroup",
                "dax:DescribeClusters",
                "dax:DescribeDefaultParameters",
                "dax:DescribeEvents",
                "dax:DescribeParameterGroups",
                "dax:DescribeParameters",
                "dax:DescribeSubnetGroups",
                "dax:IncreaseReplicationFactor",
                "dax:ListTags",
                "dax:RebootNode",
                "dax:TagResource",
                "dax:UntagResource",
                "dax:UpdateCluster",
                "dax:UpdateParameterGroup",
                "dax:UpdateSubnetGroup"
                ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

擴展 DAX 叢集

調整 DAX 叢集有兩種可用選項。第一個選項是 horizontal scaling (水平擴展)，您可在此將僅供讀取複本新增至叢集。第二個選項是 vertical scaling (垂直擴展)，您可在此選取不同的節點類型。如果需要如何為應用程式選擇適當叢集大小和節點類型的建議，請參閱 DAX 叢集調整大小指南。

水平擴展

使用水平擴展，您可以將更多的唯讀複本新增至叢集，改善輸送量。單一 DAX 叢集最多支援 10 個僅供讀取複本，您可在叢集執行時新增或移除複本。

您新增節點時，必須同步來自對等節點的快取資料。因此新增時間會根據快取大小和應用程式工作負載而有所不同。我們建議的最佳實務是預先擴展叢集，以符合預期的流量峰值。如需適當調整規模的指導方針和監控建議的相關資訊，請參閱 DAX 叢集調整大小指南。

下列 AWS CLI 範例示範如何增加或減少節點數量。`--new-replication-factor` 引數會指定叢集中的節點總數。其中一個節點是主節點，其他節點則是僅供讀取複本。

```
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**
>
> 當您修改複寫因素時，叢集狀態會變更為 `modifying`。當修改完成時，狀態會變更為 `available`。

垂直擴展

如果您有大型的資料工作集，您的應用程式可能得益於使用較大的節點類型。較大的節點可讓叢集在記憶體中存放更多資料、減少快取未中，改善應用程式的整體應用程式效能 (DAX 叢集中的所有節點都必須是相同的類型。)

如果 DAX 叢集的寫入操作或快取未中率很高，使用較大的節點類型應可讓您的應用程式受益。寫入操作和快取未中會消耗叢集主節點上的資源，因此，使用較大的節點類型可能會提高主節點的效能，進而為這類的操作提供較高的輸送量。

您無法在執行中的 DAX 叢集上修改節點類型。而是必須使用所需節點類型建立新的叢集。如需支援的節點類型清單，請參閱「節點」。

您可以使用 AWS 管理主控台、 AWS CLI、 或 AWS SDK 建立新的 CloudFormationDAX 叢集。 (對於 AWS CLI，請使用 `--node-type` 參數來指定節點類型。)

自訂 DAX 叢集設定

當您建立 DAX 叢集時，會使用下列預設設定：

- 啟用自動快取移出，存留時間 (TTL) 為 5 分鐘
- 無偏好的可用區域
- 無偏好的維護時段
- 停用通知

針對新的叢集，您可以在建立期間自訂設定。若要在 AWS 管理主控台中執行此作業，請清除 Use default settings (使用預設設定) 以修改下列設定：

- 網路和安全性 - 可讓您在目前 AWS 區域中的不同可用區域中執行個別 DAX 叢集節點。如果您選擇 No Preference (無偏好設定)，節點會自動分散在可用區域中。
- Parameter Group (參數群組)：套用到叢集中每個節點的一組具名參數。您可以使用參數群組指定快取 TTL 行為。您可以隨時變更參數群組中任何指定參數的數值 (預設參數群組 `default.dax.1.0` 除外)。
- Maintenance Window (維護時段)：軟體更新和修補程式套用到叢集中節點的每週時間期間。您可以選擇開始日、開始時間和維護時段的持續時間。如果您選擇 No Preference (無偏好設定)，即會在每個區域的 8 小時時段內隨機選取維護時段。如需詳細資訊，請參閱Maintenance window (維護時段)。

> **Note**
>
> 您也可以隨時在執行中的叢集上變更 Parameter Group (參數群組) 和 Maintenance Window (維護視窗)。

發生維護事件時，DAX 可以使用 Amazon Simple Notification Service (Amazon SNS) 通知您。若要設定通知，請從 Topic for SNS notification (SNS 通知主題) 選擇器中選擇選項。您可以建立新的 Amazon SNS 主題，或使用現有的主題。

如需設定和訂閱 Amazon SNS 主題的詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的 Amazon SNS 入門。

配置 TTL 設定

DAX 會維護兩種從 DynamoDB 讀取的資料快取：

- Item cache (項目快取)：適用於使用 `GetItem` 或 `BatchGetItem` 擷取的項目。
- Query cache (查詢快取)：適用於使用 `Query` 或 `Scan` 擷取的結果集。

如需詳細資訊，請參閱 項目快取 及 查詢快取。

這些快取每個的預設 TTL 都是 5 分鐘。如果您想要使用不同的 TTL 設定，您可以使用自訂參數群組啟動 DAX 叢集。若要在主控台上執行此作業，請在導覽窗格中選擇 DAX | Parameter groups (DAX | 參數群組)。

您也可以使用 AWS CLI執行這些任務。以下範例示範如何使用自訂參數群組啟動新的 DAX 叢集。在此範例中，項目快取的 TTL 會設為 10 分鐘，而查詢快取的 TTL 會設為 3 分鐘。

1. 建立新的參數群組。

```
aws dax create-parameter-group \
    --parameter-group-name custom-ttl
```

2. 將項目快取 TTL 設成 10 分鐘 (600000 毫秒)。

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=record-ttl-millis,ParameterValue=600000"
```

3. 將查詢快取 TTL 設成 3 分鐘 (180000 毫秒)。

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=query-ttl-millis,ParameterValue=180000"
```

4. 驗證參數設定是否正確。

```
aws dax describe-parameters --parameter-group-name custom-ttl \
    --query "Parameters[*].[ParameterName,Description,ParameterValue]"
```

您現在可以使用此參數群組啟動新的 DAX 叢集。

```
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**
>
> 您可以修改連接至執行中 DAX 叢集的參數群組。參數變更僅適用於修改後寫入的項目。現有的快取項目會保留最初寫入時的 TTL 和設定。

DAX 的標記支援

包括 DynamoDB 在內的許多 AWS 服務都支援標記 - 能夠使用使用者定義的名稱標記資源。您可以將標籤指派給 DAX 叢集，讓您可以快速識別具有相同標籤的所有 AWS 資源，或根據您指派的標籤來分類 AWS 帳單。

如需詳細資訊，請參閱將標籤新增至 DynamoDB 中的資源。

使用 AWS 管理主控台

管理 DAX 叢集標籤

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在導覽窗格中，選擇 DAX 下方的 Clusters (叢集)。
3. 選擇您想要使用的叢集。
4. 選擇 Tags (標籤) 索引標籤。您可以在此新增、列出、編輯或刪除您的標籤。

   當您滿意設定後，請選擇 Apply Changes (套用變更)。

使用 AWS CLI

當您使用 AWS CLI 管理 DAX 叢集標籤時，您必須先判斷叢集的 Amazon Resource Name (ARN)。以下範例示範如何決定名為 `MyDAXCluster` 叢集的 ARN。

```
aws dax describe-clusters \
    --cluster-name MyDAXCluster \
    --query "Clusters[*].ClusterArn"
```

在輸出中，ARN 看起來會如下：`arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster`

以下範例示範如何標記叢集。

```
aws dax tag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tags="Key=ClusterUsage,Value=prod"
```

列出叢集的所有標籤。

```
aws dax list-tags \
```

```
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster
```

您可以指定標籤的鍵來移除標籤。

```
aws dax untag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tag-keys ClusterUsage
```

AWS CloudTrail 整合

DAX 已與 整合 AWS CloudTrail，可讓您稽核 DAX 叢集活動。您可以使用 CloudTrail 日誌來檢視在叢集層級進行的所有變更。您也可以查看對叢集元件進行的變更，例如節點、子網路群組和參數群組。如需詳細資訊，請參閱使用 記錄 DynamoDB 操作 AWS CloudTrail。

刪除 DAX 叢集

如果您不再使用 DAX 叢集，建議您刪除它，以避免支付未使用資源的費用。

您可以使用主控台或 AWS CLI來刪除 DAX 叢集。下列是 範例。

```
aws dax delete-cluster --cluster-name mydaxcluster
```

監控 DynamoDB Accelerator

監控是維護 Amazon DynamoDB Accelerator (DAX) 和 AWS 解決方案可靠性、可用性和效能的重要部分。您應該從 AWS 解決方案的所有部分收集監控資料，以便在發生多點失敗時更輕鬆地偵錯。

在開始監控 DAX 之前，應先建立監控計畫，為下列問題提供解答：

- 監控目標是什麼？
- 要監控哪些資源？
- 監控這些資源的頻率為何？
- 要使用哪些監控工具？
- 誰將執行監控任務？
- 發生問題時應該通知誰？

主題

- 適用於 DynamoDB Accelerator 的監控工具

- 使用 Amazon CloudWatch 監控
- 使用 記錄 DAX 操作 AWS CloudTrail

適用於 DynamoDB Accelerator 的監控工具

AWS 提供可用來監控 Amazon DynamoDB Accelerator (DAX) 的工具。您可以設定其中一些工具為您進行監控，而有些工具需要手動介入。建議您盡可能自動化監控任務。

主題

- 自動化監控工具
- 手動監控工具

自動化監控工具

您可以使用下列自動化監控工具來監看 DAX，並在發生錯誤時進行回報：

- Amazon CloudWatch 警示：監看指定時段內的單一指標，並根據與多個時段內給定之閾值相對的指標值來執行一或多個動作。此動作是傳送到 Amazon Simple Notification Service (Amazon SNS) 主題或 Amazon EC2 Auto Scaling 政策的通知。CloudWatch 警示不會只因處於特定狀態就調用動作，狀態必須已變更並已維持一段指定的時間。如需詳細資訊，請參閱使用 Amazon CloudWatch 來監控 DynamoDB 指標。
- Amazon CloudWatch Logs：監控、存放及存取來自 AWS CloudTrail 或其他來源的日誌檔案。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的監控日誌檔案。
- Amazon CloudWatch Events：匹配事件並將它們路由至一或多個目標函式或串流以進行變更、擷取狀態資訊，以及採取修正動作。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的什麼是 Amazon CloudWatch Events？。
- AWS CloudTrail 日誌監控 – 在帳戶之間共用日誌檔案、透過將日誌檔案傳送到 CloudWatch Logs 來即時監控 CloudTrail 日誌檔案、在 Java 中寫入日誌處理應用程式，以及驗證您的日誌檔案在 CloudTrail 交付後並未變更。如需詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的使用 CloudTrail 日誌檔案。

手動監控工具

監控 DAX 的另一個重要部分是手動監控 CloudWatch 警示未涵蓋的項目。DAX Trusted Advisor、CloudWatch 和其他 AWS 管理主控台 儀表板可讓您at-a-glance檢視環境 AWS 的狀態。建議您也查看 DAX 上的日誌檔。

- DAX 儀表板會顯示下列項目：
  - 服務運作狀態
- CloudWatch 首頁會顯示下列項目：
  - 目前警示與狀態
  - 警示與資源的圖表
  - 服務運作狀態

  此外，您可以使用 CloudWatch 執行下列動作：
  - 建立自訂儀表板以監控您注重的服務。
  - 用於疑難排解問題以及探索驅勢的圖形指標資料。
  - 搜尋和瀏覽您的所有 AWS 資源指標。
  - 建立與編輯要通知發生問題的警示。

使用 Amazon CloudWatch 監控

您可以使用 Amazon CloudWatch 來監控 DynamoDB Accelerator (DAX)；該服務會收集並處理來自 DAX 的原始資料，進而將這些資料轉換為便於讀取且幾近即時的指標。會記錄兩週期間的統計數字。接著，您就可以存取歷史資訊，並更清楚 Web 應用程式或服務的執行效能。根據預設，系統會自動將 DAX 指標資料傳送至 CloudWatch。如需更多詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的什麼是 Amazon CloudWatch？。

主題

- 如何使用 DAX 指標？
- 檢視 DAX 指標和維度
- 建立 CloudWatch 警示來監控 DAX
- 生產監控

如何使用 DAX 指標？

DAX 回報的指標可提供資訊，您可透過不同方式加以分析。下列清單顯示一些常見的指標用途。這些是建議，以協助您開始，而不是完整清單。

如何？	相關指標
判斷是否發生任何系統錯誤	監控 `FaultRequestCount` 以判斷是否有任何請求導致 HTTP 500 (伺服器錯誤) 代碼。這表示 DAX 內部服務錯誤或基礎資料表的 SystemErrors 指標中的 HTTP 500。
判斷是否發生任何使用者錯誤	監控 `ErrorRequestCount` 以判斷是否有任何請求導致 HTTP 400 (用戶端器錯誤) 代碼。如果您發現錯誤數量不斷成長，您可能希望進行調查並確保您傳送的是正確的用戶端請求。
判斷是否發生任何快取遺漏	監控 `ItemCacheMisses` 以判斷在快取中未找到某項目的次數，`QueryCacheMisses` 和 `ScanCacheMisses` 來判斷在快取中未找到某查詢或掃描結果的次數。
監控快取命中率	使用 CloudWatch 指標數學來定義使用數學表達式的快取命中率指標。 例如，針對項目快取，您可以使用表達式 m1/SUM([m1, m2])*100，其中 m1 是 `ItemCacheHits` 指標而 m2 是叢集的 `ItemCacheMisses` 指標。 針對查詢和掃描快取，您可以使用對應的查詢和掃描快取指標來遵循相同的模式。

檢視 DAX 指標和維度

當您與 Amazon DynamoDB 互動時，其會將下列指標和維度傳送至 Amazon CloudWatch。您可以使用下列程序來檢視 DynamoDB Accelerator (DAX) 的指標。

檢視指標 (主控台)

指標會先依服務命名空間分組，再依各命名空間內不同的維度組合分類。

1. 在 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在導覽窗格中，選擇指標。
3. 選取 DAX 命名空間。

檢視指標 (AWS CLI)

- 在命令提示中，使用下列命令。

```
aws cloudwatch list-metrics --namespace "AWS/DAX"
```

DAX 指標與維度

以下部分包含 DAX 傳送至 CloudWatch 的指標和維度。

DAX 指標

下列是 DAX 提供的指標。DAX 只會將具有非零值的指標傳送至 CloudWatch。

> **Note**
>
> CloudWatch 每隔一分鐘彙總以下 DAX 指標：
>
> - 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

並非所有統計數字，例如 Average 或 Sum，皆適用於所有指標。不過，所有這些值都可以透過 DAX 主控台，或使用 CloudWatch 主控台 AWS CLI，或為所有指標提供 AWS SDKs。在下表中，每個指標皆有適用於該指標的有效統計數字列表。

指標	Description
CPUUtilization	節點或叢集的 CPU 使用率百分比。 單位：Percent 有效的統計數字： • Minimum • Maximum • Average

指標	Description
CacheMemoryUtilization	節點或叢集上項目快取和查詢快取正在使用的可用快取記憶體百分比。快取的資料會在記憶體使用率達到 100% 之前開始移除 (請參閱 EvictedSize 指標)。如果 CacheMemoryUtilization 在任何節點上達到 100%，寫入請求將進行限流，且您應該考慮切換到節點類型較大的叢集。 單位：Percent 有效的統計數字： • Minimum • Maximum • Average
NetworkBytesIn	節點或叢集在所有網路介面上收到的位元組數目。 單位：Bytes 有效的統計數字： • Minimum • Maximum • Average
NetworkBytesOut	節點或叢集在所有網路介面上送出的位元組數目。此指標識別單一節點或叢集上的傳出流量 (位元組數目)。 單位：Bytes 有效的統計數字： • Minimum • Maximum • Average

指標	Description
BaselineNetworkBytesInUtilization	傳入流量在指定時間消耗的基準網路頻寬百分比。作為參考：50% 表示傳入流量正在使用一半的可用網路頻寬。 單位：Percent 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
BaselineNetworkBytesOutUtilization	傳出流量在指定時間消耗的基準網路頻寬百分比。作為參考：50% 表示傳出流量正在使用一半的可用網路頻寬。 單位：Percent 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
NetworkPacketsIn	節點或叢集在所有網路介面上收到的封包數目。 單位：Count 有效的統計數字： • Minimum • Maximum • Average

指標	Description
NetworkPacketsOut	節點或叢集在所有網路介面上送出的封包數目。此指標識別單一節點或叢集上的傳出流量 (封包數目)。 單位：Count 有效的統計數字： • Minimum • Maximum • Average
GetItemRequestCount	節點或叢集處理的 GetItem 請求數量。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
BatchGetItemRequestCount	節點或叢集處理的 BatchGetItem 請求數量。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum

指標	Description
BatchWriteItemRequestCount	節點或叢集處理的 BatchWriteItem 請求數量。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
DeleteItemRequestCount	節點或叢集處理的 DeleteItem 請求數量。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
PutItemRequestCount	節點或叢集處理的 PutItem 請求數量。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum

指標	Description
UpdateItemRequestCount	節點或叢集處理的 UpdateItem 請求數量。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
TransactWriteItemsCount	節點或叢集處理的 TransactWriteItems 請求數量。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
TransactGetItemsCount	節點或叢集處理的 TransactGetItems 請求數量。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum

指標	Description
ItemCacheHits	從節點或叢集快取傳回項目的次數。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
ItemCacheMisses	項目不在節點或叢集快取中且必須從 DynamoDB 擷取的次數。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
QueryCacheHits	從節點或叢集快取傳回查詢結果的次數。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum

指標	Description
QueryCacheMisses	查詢結果不在節點或叢集快取中且必須從 DynamoDB 擷取的次數。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
ScanCacheHits	從節點或叢集快取傳回掃描結果的次數。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum

指標	Description
ScanCacheMisses	掃描結果不在節點或叢集快取中且必須從 DynamoDB 擷取的次數。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
TotalRequestCount	節點或叢集處理的請求總數。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum

指標	Description
ErrorRequestCount	節點或叢集回報的導致使用者錯誤的請求總數。包含節點或叢集限流的請求。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
ThrottledRequestCount	節點或叢集限流的請求總數。由 DynamoDB 限流的請求不包含在內，但可以使用 DynamoDB 指標監控这样的请求。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum

指標	Description
FaultRequestCount	節點或叢集回報的導致內部錯誤的請求總數。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
FailedRequestCount	節點或叢集回報的導致錯誤的請求總數。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
QueryRequestCount	節點或叢集處理的查詢數目。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum

指標	Description
ScanRequestCount	節點或叢集處理的掃描請求數目。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
ClientConnections	用戶端連線至節點或叢集的同時連線數目。 單位：Count 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
EstimatedDbSize	節點或叢集在項目快取及查詢快取中快取資料量近似值。 單位：Bytes 有效的統計數字： • Minimum • Maximum • Average

指標	Description
EvictedSize	節點或叢集为騰出空間給新請求的資料而移除的資料量。如果遺漏率升高，并且此指標也不斷增加，這可能表示您的工作集已經增加。您應該考慮切換到節點類型較大的叢集。 單位：Bytes 有效的統計數字： • Minimum • Maximum • Average • Sum
CPUCreditUsage	節點為 CPU 使用率花費的 CPU 點數數量。一個 CPU 額度等於一個 vCPU 以 100% 使用率執行 1 分鐘，或同等的 vCPU、使用率與時間的組合 (例如，一個 vCPU 以 50% 使用率執行 2 分鐘，或兩個 vCPU 以 25% 使用率執行 2 分鐘)。 CPU 額度指標僅提供 5 分鐘頻率。如果您要指定大於 5 分鐘的期間，請使用 Sum 統計數字代替 Average。 單位：Credits (vCPU-minutes) 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum

指標	Description
CPUCreditBalance	自節點啟動或開始後，累積獲得的 CPU 點數數量。 獲得額度後，額度會在額度餘額中累積，並在支付額度時，從額度餘額中移出。點數餘額有最大值限制，它取決於 DAX 節點大小。到達限制之後，任何獲得的新額度都會遭到捨棄。 CPUCreditBalance 中的點數可供節點支付以大幅提升並超越基準 CPU 使用率。 單位：Credits (vCPU-minutes) 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum
CPUSurplusCreditBalance	當 CPUCreditBalance 值為 0 時，DAX 節點已支出的剩餘點數數量。 CPUSurplusCreditBalance 值由獲得的 CPU 額度支付。如果剩餘點數超過節點在 24 小時期間可獲得的最大點數數量，超過最大值的支出剩餘點數將必須負擔額外的費用。 單位：Credits (vCPU-minutes) 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum

指標	Description
CPUSurplusCreditsCharged	若支出剩餘額度數量未由獲得的 CPU 額度付清，會產生額外的費用。 發生以下情況時，將收取支出剩餘點數的費用：支出剩餘點數超過節點在 24 小時期間可獲得的最大點數數量。在小時結束或節點終止時，將收取超過最大值的支出剩餘點數的費用。 單位：Credits (vCPU-minutes) 有效的統計數字： • Minimum • Maximum • Average • SampleCount • Sum

> **Note**
>
> CPUCreditUsage、CPUCreditBalance、CPUSurplusCreditBalance 和 CPUSurplusCreditsCharged 指標僅適用於 T3 節點。

DAX 指標的維度

DAX 指標是由帳戶、叢集 ID 或叢集 ID 和節點 ID 結合值量化。您可以使用 CloudWatch 主控台擷取 DAX 資料以及下表中的任何維度。

維度	Description
`Account`	提供帳戶中所有節點的彙總統計數字。
`ClusterId`	將資料限制為叢集。
`ClusterId, NodeId`	將資料限制為叢集內的節點。

建立 CloudWatch 警示來監控 DAX

您可以建立 Amazon CloudWatch 警報，在警示變更狀態時傳送 Amazon Simple Notification Service (Amazon SNS) 訊息。警示會在您指定的期間監看單一指標。警示會根據在數個期間與指定閾值相關的指標值，來執行一個或多個動作。此動作是傳送到 Amazon SNS 主題或 Auto Scaling 政策的通知。警示僅會針對持續狀態變更調用動作。CloudWatch 警示不會只因為處於特定狀態而調用動作。狀態必須已變更，且在指定的期間數內維持此狀態。

我要如何收到這些查詢快取遺漏的通知？

1. 建立 Amazon SNS 主題 (`arn:aws:sns:us-west-2:522194210714:QueryMissAlarm`)。

   如需更多詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的設定 Amazon Simple Notification Service。
2. 建立警示。

```
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. 測試警示。

```
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**
>
> 您可以增加或減少閾值，使其符合您的應用程式需求。您也可以使用 CloudWatch 指標數學來定義快取遺漏率指標並針對該指標設定警示。

如果請求在叢集中造成內部錯誤，我如何收到通知？

1. 建立 Amazon SNS 主題 (`arn:aws:sns:us-west-2:123456789012:notify-on-system-errors`)。

   如需更多詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的設定 Amazon Simple Notification Service。

2. 建立警示。

```
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. 測試警示。

```
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
```

生產監控

您應在各個時間點和不同的負載條件下測量效能，以在您的環境中確立 DAX 正常效能的基準。當您監控 DAX 時，應該考慮存放歷史監控資料。這個存放的資料會提供基準，讓您與目前的效能資料比較，識別出正常的效能模式和效能異常狀況，再規劃方式來處理問題。

若要確立基準，您至少應在負載測試和生產階段監控下列項目。

- CPU 使用率和限流請求，可讓您判斷是否需要在叢集中使用大型節點類型。您可透過 CPUUtilization CloudWatch 指標了解叢集的 CPU 使用率。此指標的平均統計資料可協助您掌握叢集中所有節點的平均 CPU 使用率。對於叢集擴展決策，我們建議您使用最高統計資料，也就是所有節點的最高使用率。

  > **Note**
  >
  > AWS 已改善CPUUtilization指標的精細程度。您可能會觀察到指標從 2024 年 5 月 17 日到 2024 年 6 月 22 日所發生的變化。

- 操作延遲 (在用戶端測量) 應該保持在應用程式的延遲需求之內。
- 應保持低錯誤率，如 ErrorRequestCount、FaultRequestCount 和 FailedRequestCount CloudWatch 指標中所見。

- 網路位元組耗用可協助您判斷是否應在叢集中使用更多節點或更大的節點類型。若要監控耗用量，您可以針對 CloudWatch 中可用的 `BaselineNetworkBytesInUtilization` 和 `BaselineNetworkBytesOutUtilization` 指標設定提醒，分別指出傳入和傳出流量的執行個體類型可用網路頻寬的耗用百分比。
- 快取記憶體使用率和移出的大小，如此就能判斷叢集的節點類型是否有足夠的記憶體可保留您的工作集，若是不足，則切換至較大的節點類型。

> **Note**
>
> 在大量快取遺漏和寫入的情況下，快取記憶體使用率可能增加達 100%，且可能造成可用性停擺。

- 用戶端連接，可讓您監控叢集連接中任何無法解釋的峰值。

使用 記錄 DAX 操作 AWS CloudTrail

Amazon DynamoDB Accelerator (DAX) 已與 服務整合 AWS CloudTrail，此服務可提供 DAX 中使用者、角色或 AWS 服務所採取動作的記錄。

若要深入了解 DAX 和 CloudTrail，請參閱 使用 記錄 DynamoDB 操作 AWS CloudTrail 中的 DynamoDB Accelerator (DAX) 章節。

DAX T3/T2 爆量執行個體

DAX 可讓您在固定效能執行個體 (例如 R4、R5 和 R7) 與爆量效能執行個體 (例如 T2 和 T3) 之間選擇。爆量效能執行個體提供基準水準的 CPU 效能，並可在需要時大幅超越基準水準。

基準效能與大幅提升效能的能力，取決於 CPU 的點數。工作負載低於基準水準閾值時，爆量效能執行個體會以執行個體大小決定的速率持續累積 CPU 點數。當工作負載增加時，則可能會使用這些點數。一個 CPU 點數提供一分鐘、一個 CPU 核心的完整效能。

許多工作負載不需要一致的高 CPU 水準，但可在需要時受益於完全運用非常快速的 CPU。爆量效能執行個體是專為這些使用案例而設計。如果您的資料庫需要一致的高 CPU 效能，我們建議您使用固定效能執行個體。

DAX T2 執行個體系列

DAX T2 執行個體是爆量效能執行個體，除了提供基準水準的 CPU 效能之外，還可在需要時大幅超越基準水準。T2 執行個體是需要可預測價格的測試與開發工作負載的理想選擇。DAX T2 執行個體專為

標準模式設定，換言之，如果執行個體的累積點數不足，CPU 使用率則會逐漸降到基準水準。如需標準模式的詳細資訊，請參閱《Amazon EC2 使用者指南》中的爆量效能執行個體的標準模式。

DAX T3 執行個體系列

DAX T3 執行個體是下一代爆量的一般用途執行個體類型，其可提供 CPU 基準效能，而且只要有所需要，就可大幅提升 CPU 用量。T3 執行個體在運算、記憶體和網路資源之間取得平衡，非常適合 CPU 使用率中等、遇到暫時使用量峰值的工作負載。DAX T3 執行個體設定為無限制模式，這表示它們可以在 24 小時時段內大幅提升並超越基準，但需額外付費。如需無限制模式的詳細資訊，請參閱《Amazon EC2 使用者指南》中的爆量效能執行個體的無限制模式。

只要工作負載需要，DAX T3 執行個體就能維持高 CPU 效能。針對大多數的一般用途工作負載，T3 執行個體可提供足夠效能，不會收取任何額外費用。當 T3 執行個體的 CPU 平均使用率在 24 小時時段內等於或低於基準時，T3 執行個體的每小時價格將自動涵蓋所有暫時使用量峰值。

例如，`dax.t3.small` 執行個體以每小時 24 個 CPU 點數的速率持續接收點數。此功能提供相當於一個 CPU 核心 20% 的基準效能 (20% × 60 分鐘 = 12 分鐘)。如果執行個體未使用其收到的點數，則會將這些點數儲存在其 CPU 點數餘額中，最多可達 576 個 CPU 點數。當 `t3.small` 執行個體需要爆量到一個核心的 20% 以上時，它會提取 CPU 點數餘額，以便自動處理此突增的使用量。

雖然 DAX T2 執行個體會在 CPU 點數提取到零後限制在基準效能，但即使 CPU 點數餘額為零，DAX T3 執行個體仍可大幅提升並超越基準。對於大多數的工作負載 (CPU 平均使用率等於或低於基準效能)，`t3.small` 的每小時基本價格能涵蓋所有 CPU 爆量。如果執行個體在 CPU 點數餘額在提取到零後以平均 25% 的 CPU 使用率 (高於基準 5%) 執行 24 小時，則會額外收取 11.52 美分的費用 (9.6 美分/vCPU-小時 × 1 vCPU × 5% × 24 小時)。如需定價詳細資訊，請參閱 Amazon DynamoDB 定價。

DAX 存取控制

DynamoDB Accelerator (DAX) 可與 DynamoDB 一起使用，為您的應用程式無縫新增快取層。但是，DAX 和 DynamoDB 具有不同的存取控制機制。這兩個服務都使用 AWS Identity and Access Management (IAM) 來實作各自的安全政策，但 DAX 和 DynamoDB 的安全模型不同。

我們強烈建議您了解兩種安全模型，以便為使用 DAX 的應用程式實作合適的安全措施。

本節說明 DAX 提供的存取控制機制，並提供範例 IAM 政策。您可以使用此政策來為您的需求量身打造。

使用 DynamoDB，您可以建立限制使用者在個別 DynamoDB 資源上可執行動作的 IAM 政策。例如，您可以建立使用者角色，只允許使用者對特定 DynamoDB 資料表執行唯讀動作。(如需詳細資訊，請參

閱「Amazon DynamoDB 的 Identity and Access Management」。) 相較之下，DAX 安全模型著重在叢集安全，以及叢集代替您執行 DynamoDB API 動作的能力。

> **Warning**
>
> 若您目前使用 IAM 角色及政策限制對 DynamoDB 資料表資料的存取，使用 DAX 可以推翻這些政策。例如，使用者可透過 DAX 存取 DynamoDB 資料表，但無法藉由直接存取 DynamoDB 來明確存取相同的資料表。如需詳細資訊，請參閱Amazon DynamoDB 的 Identity and Access Management。
>
> DAX 不會強制對 DynamoDB 上的資料執行使用者層級隔離。相反的，使用者會在存取該叢集時，繼承 DAX 叢集 IAM 政策的許可。因此，透過 DAX 存取 DynamoDB 資料表時，唯一有效的存取控制便是 DAX 叢集 IAM 政策的許可。其他任何許可都不會獲得承認。
>
> 若您需要隔離，我們建議您建立其他 DAX 叢集，並為每個叢集劃定 IAM 政策的範圍。例如，您可以建立多個 DAX 叢集，然後只允許每個叢集存取單一資料表。

DAX 的 IAM 服務角色

建立 DAX 叢集時，您必須為叢集與 IAM 角色建立關聯。這稱為叢集的服務角色。

假設您想要建立名為 DAXCluster01 的新 DAX 叢集。您可以建立名為 DAXServiceRole 的服務角色，然後將該角色與 DAXCluster01 建立關聯。DAXServiceRole 的政策可定義 DAXCluster01 代替與 DAXCluster01 互動之使用者執行的 DynamoDB 動作。

建立服務角色時，您必須指定 DAXServiceRole 與 DAX 服務本身的信任關係。信任關聯會判斷可取得角色及使用其許可的實體。以下為 DAXServiceRole 的信任關係文件範例：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
```

```
    ]
}
```

此信任關係允許 DAX 叢集取得 DAXServiceRole，並代替您執行 DynamoDB API 呼叫。

允許的 DynamoDB API 動作會在 IAM 政策文件中說明。您可以將此文件連接到 DAXServiceRole。以下為範例政策文件。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DaxAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
            ]
        }
    ]
}
```

此政策允許 DAX 在 DynamoDB 資料表上執行必要 DynamoDB API 動作。DAX 需要 `dynamodb:DescribeTable` 動作才能維護資料表的相關中繼資料，其他則是在表內項目上執行的讀取與寫入動作。該資料表名為 Books，位於 us-west-2 區域，由 AWS 帳戶 ID 123456789012 擁有。

> **Note**
>
> DAX 支援透過機制在跨服務存取期間防止發生混淆代理人問題。如需詳細資訊，請參閱《IAM 使用者指南》中的混淆代理問題。

允許 DAX 叢集存取的 IAM 政策

建立 DAX 叢集後，您需要授予使用者許可，讓使用者能夠存取 DAX 叢集。

例如，假設您希望將存取 DAXCluster01 的許可授予名為 Alice 的使用者。您可先建立 IAM 政策 (AliceAccessPolicy)，藉此定義收件人可以存取的 DAX 叢集和 DAX API 動作。您接著便會藉由將此政策連接到使用者 Alice，來授予存取。

下列政策文件會給予收件人 DAXCluster01 的完整存取。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

政策文件允許 DAX 叢集的存取，但並未授予任何 DynamoDB 許可。(DynamoDB 許可會由 DAX 服務角色授予。)

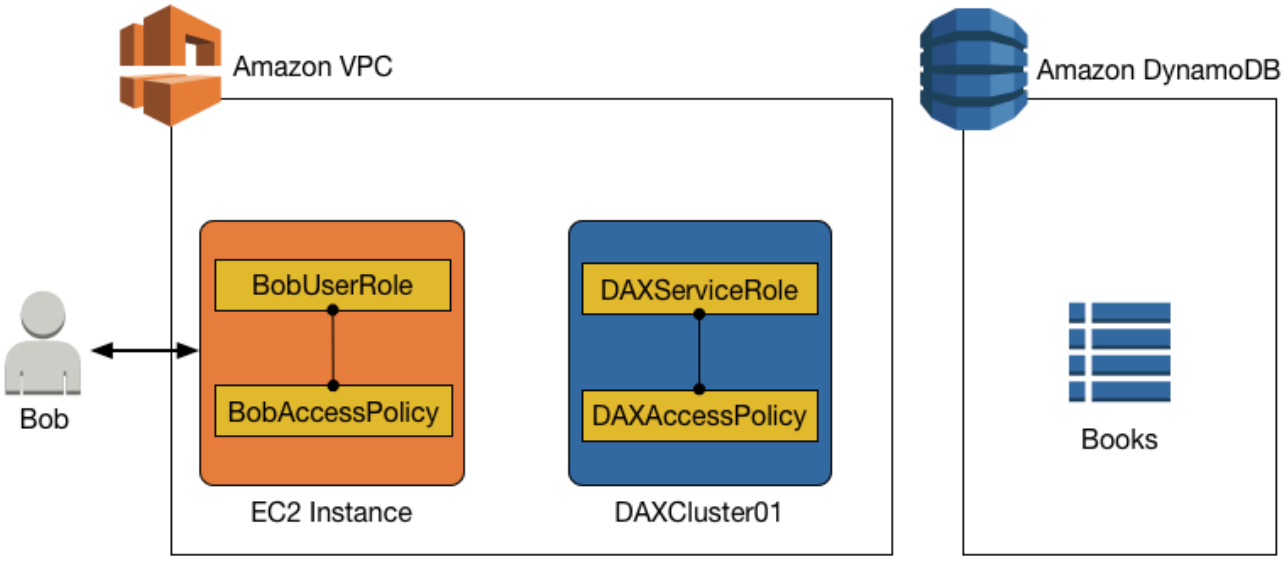
針對使用者 Alice，您首先要使用先前顯示的政策文件建立 `AliceAccessPolicy`。您接著會將政策連接到 Alice。

> **Note**
>
> 除了將政策連接到使用者之外，您也可以將政策連接到 IAM 角色。透過這種方式，所有取得該角色的使用者都會擁有您在政策中定義的許可。

使用者政策結合 DAX 服務角色，會判斷收件人可透過 DAX 存取的 DynamoDB 資源和 API 動作。

案例研究：存取 DynamoDB 和 DAX

下列案例可以幫助您進一步了解用於 DAX 的 IAM 政策。(本節剩餘的部分將參考這個案例)。以下圖表顯示案例的高層級概觀。



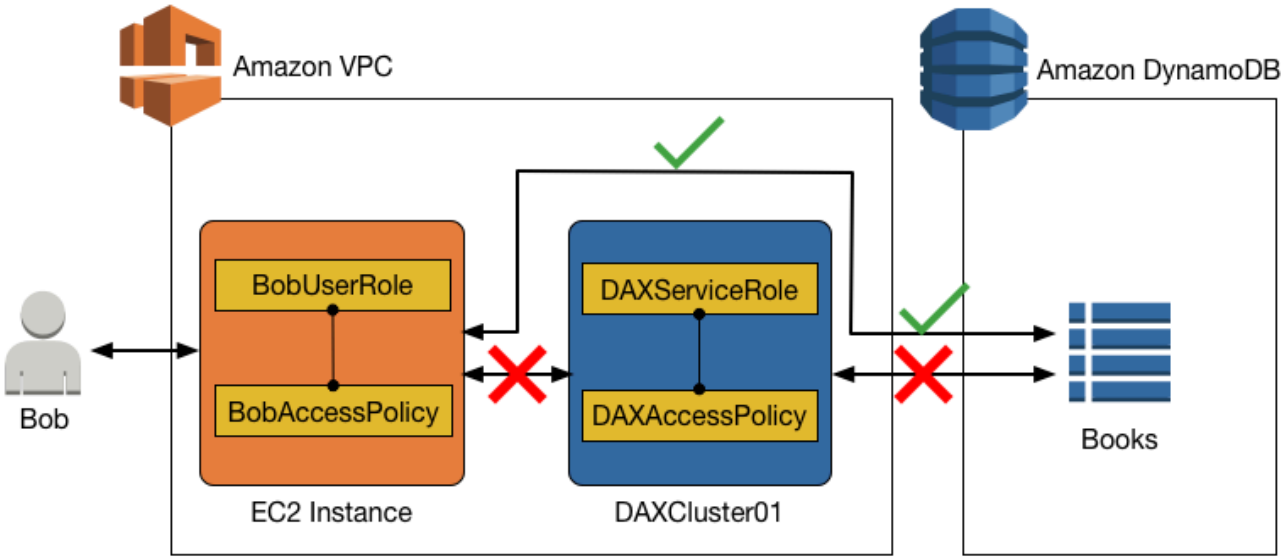
在此案例中，有下列實體：

- 使用者 (Bob)。
- IAM 角色 (`BobUserRole`)。Bob 會在執行時期取得此角色。
- IAM 政策 (`BobAccessPolicy`)。此政策會連接到 `BobUserRole`。`BobAccessPolicy` 會定義 `BobUserRole` 允許存取的 DynamoDB 和 DAX 資源。
- DAX 叢集 (`DAXCluster01`)。
- IAM 服務角色 (`DAXServiceRole`)。此角色允許 `DAXCluster01` 存取 DynamoDB。

- IAM 政策 (DAXAccessPolicy)。此政策會連接到 DAXServiceRole。DAXAccessPolicy 會定義 DAXCluster01 允許存取的 DynamoDB API 和資源。
- DynamoDB 資料表 (Books)。

BobAccessPolicy 和 DAXAccessPolicy 中的政策陳述式組合會決定 Bob 可以如何使用 Books 資料表。例如，Bob 可能可以直接 (使用 DynamoDB 端點)、間接 (使用 DAX 叢集) 或同時使用這兩種方式存取 Books。Bob 也可能可以從 Books 讀取資料、寫入資料到 Books，或同時具有這兩種許可。

存取 DynamoDB，但不可使用 DAX 存取



您可以允許直接存取 DynamoDB 資料表，同時防止透過 DAX 叢集間接進行存取。如需直接存取 DynamoDB，BobUserRole 的許可是由 BobAccessPolicy (連接到角色) 決定。

DynamoDB 的唯讀存取 (僅限)

Bob 可以使用 BobUserRole 存取 DynamoDB。連接到此角色 (BobAccessPolicy) 的 IAM 政策會判斷 BobUserRole 可存取的 DynamoDB 資料表，以及 BobUserRole 可調用的 API。

考慮下列 BobAccessPolicy 的政策文件。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

當此文件連接到 BobAccessPolicy 時，它會允許 BobUserRole 存取 DynamoDB 端點，並對 Books 資料表執行唯讀操作。

DAX 沒有出現在此政策中，因此透過 DAX 進行存取會遭到拒絕。

DynamoDB 的讀取/寫入存取 (僅限)

若 BobUserRole 需要 DynamoDB 的讀寫存取，將適用以下政策。

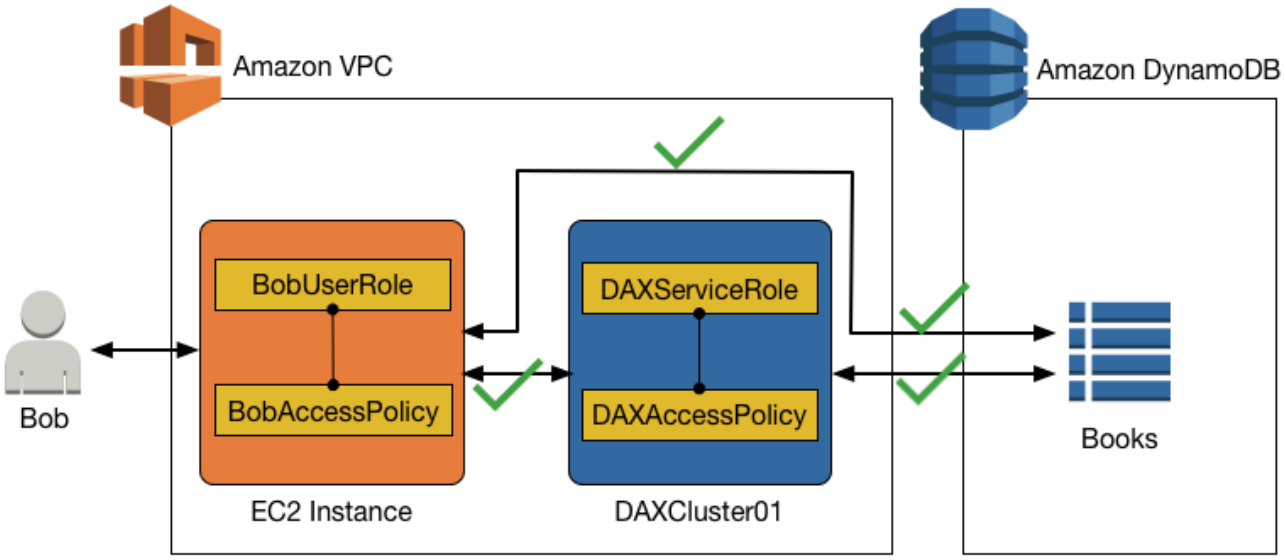
JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
```

```
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

同樣的，DAX 沒有出現在此政策中，因此透過 DAX 進行存取會遭到拒絕。

存取 DynamoDB 和 DAX



若要允許存取 DAX 叢集，您必須在 IAM 政策中包含 DAX 專屬的動作。

下列 DAX 專屬動作會對應到他們在 DynamoDB API 中名稱相似的動作：

- `dax:GetItem`
- `dax:BatchGetItem`
- `dax:Query`

- dax:Scan
- dax:PutItem
- dax:UpdateItem
- dax:DeleteItem
- dax:BatchWriteItem
- dax:ConditionCheckItem

這同樣適用於 dax:EnclosingOperation 條件金鑰。

DynamoDB 的唯讀存取及 DAX 的唯讀存取

假設 Bob 需要透過 DynamoDB 及 DAX 對 Books 資料表進行唯讀存取。以下政策 (連接到 BobUserRole) 會授予這項存取。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
```

```
        }
    ]
}
```

政策具有 DAX 存取的陳述式 (DAXAccessStmt) 及另一個 DynamoDBaccess 的陳述式 (DynamoDBAccessStmt)。這些陳述式會允許 Bob 向 DAXCluster01 傳送 GetItem、BatchGetItem、Query 及 Scan 請求。

但是，DAXCluster01 的服務角色也需要對 DynamoDB 中 Books 資料表的唯讀存取。以下連接到 DAXServiceRole 的 IAM 政策會滿足這項需求。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

讀寫存取 DynamoDB 及使用 DAX 進行唯讀存取

針對給定的使用者角色，您可以提供 DynamoDB 資料表的讀寫存取，同時也允許透過 DAX 進行唯讀存取。

針對 Bob，BobUserRole 的 IAM 政策需要允許對 Books 資料表進行 DynamoDB 讀取及寫入動作，同時也支援透過 DAXCluster01 進行唯讀動作。

以下 BobUserRole 的範例政策文件會授予這項存取。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

此外，DAXServiceRole 將需要允許 DAXCluster01 對 Books 資料表執行唯讀動作的 IAM 政策。

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:DescribeTable"
           ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

讀寫存取 DynamoDB 及讀寫存取 DAX

現在假設 Bob 需要對 Books 資料表進行讀寫存取，無論是直接透過 DynamoDB，或間接透過 DAXCluster01。以下政策文件 (連接到 BobAccessPolicy) 會授予這項存取。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
```

```
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

此外，DAXServiceRole 將需要允許 DAXCluster01 對 Books 資料表執行讀寫動作的 IAM 政策。

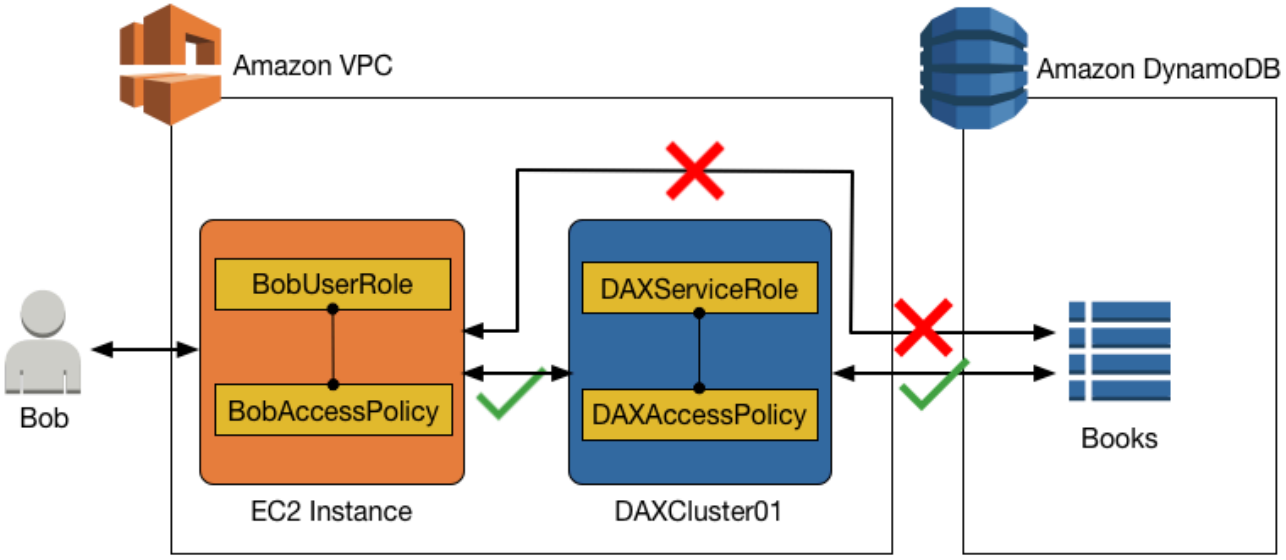
JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable"
```

```
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

透過 DAX 存取 DynamoDB，但不直接存取 DynamoDB

在此案例中，Bob 可透過 DAX 存取 Books 資料表，但無法直接存取 DynamoDB 中的 Books 資料表。因此，當 Bob 取得 DAX 的存取時，他也同時會取得透過其他方式皆無法取得的 DynamoDB 資料表的存取。當您為 DAX 服務角色設定 IAM 政策時，請記得任何透過使用者存取政策取得 DAX 叢集存取的使用者，都會取得在該政策中指定資料表的存取。在此情況下，BobAccessPolicy 可以存取 DAXAccessPolicy 中指定的資料表。



若您目前使用 IAM 角色及政策限制對 DynamoDB 資料表和資料的存取，使用 DAX 可以推翻這些政策。在下列政策中，Bob 可透過 DAX 存取 DynamoDB 資料表，但無法明確的直接存取 DynamoDB 中相同的資料表。

以下連接到 BobUserRole 的政策文件 (BobAccessPolicy) 會授予這項存取。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        }
    ]
}
```

在此存取政策中，沒有任何可直接存取 DynamoDB 的許可。

搭配 BobAccessPolicy，下列 DAXAccessPolicy 可讓 BobUserRole 存取 DynamoDB 資料表 Books，即使 BobUserRole 無法直接存取 Books 資料表。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
```

```
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

如同此範例中所示，當您為使用者存取政策及 DAX 叢集存取政策設定存取控制時，您必須完全了解端點對端點存取，以確保觀察到最低權限原則。您也必須確保給予使用者存取 DAX 叢集不會推翻先前建立的存取控制政策。

DAX 靜態加密

Amazon DynamoDB Accelerator (DAX) 靜態加密可協助保護您的資料免於發生未經授權的基礎儲存存取，為資料提供另一層保護。組織政策、行業或政府法規，以及合規要求可能需要使用靜態加密來保護您的資料。您可以使用加密來提高部署在雲端中的應用程式資料安全性。

使用靜態加密時，磁碟上的 DAX 所維持的資料使用 256 位元進階加密標準 (也稱為 AES-256 加密) 來加密。DAX 將資料寫入磁碟，做為自主節點到僅供讀取複本的變更環節之一。

DAX 靜態加密會自動與 AWS Key Management Service (AWS KMS) 整合，以管理用於加密叢集的單一服務預設金鑰。如果在建立加密的 DAX 叢集時，服務預設金鑰不存在， AWS KMS 會自動為您建立新的 AWS 受管金鑰。此金鑰會與未來建立的加密叢集搭配使用。 AWS KMS 結合安全、高可用性的硬體和軟體，以提供針對雲端擴展的金鑰管理系統。

在資料加密後，DAX 可以透明的方式處理您的資料解密，同時盡量降低對效能的影響。您不需要修改應用程式即可使用加密。

> **Note**
>
> DAX 不會 AWS KMS 針對每個 DAX 操作呼叫 。DAX 只會在叢集啟動時使用金鑰。即使撤銷了存取，DAX 仍會繼續存取資料，直到叢集關閉。不支援客戶指定的 AWS KMS 金鑰。

DAX 靜態加密可用於下列叢集節點類型。

系列	Node type
記憶體最佳化 (R4、R5 及 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
	dax.r7i.4xlarge
	dax.r7i.8xlarge
	dax.r7i.12xlarge

系列	Node type
	dax.r7i.16xlarge dax.r7i.24xlarge
一般用途 (T2)	dax.t2.small dax.t2.medium
一般用途 (T3)	dax.t3.small dax.t3.medium

> **Important**
>
> DAX 靜態加密不支援 `dax.r3.*` 節點類型。

您可以在建立叢集後啟用或停用靜態加密。若在建立時未啟用靜態加密，您必須重新建立叢集以啟用靜態加密。

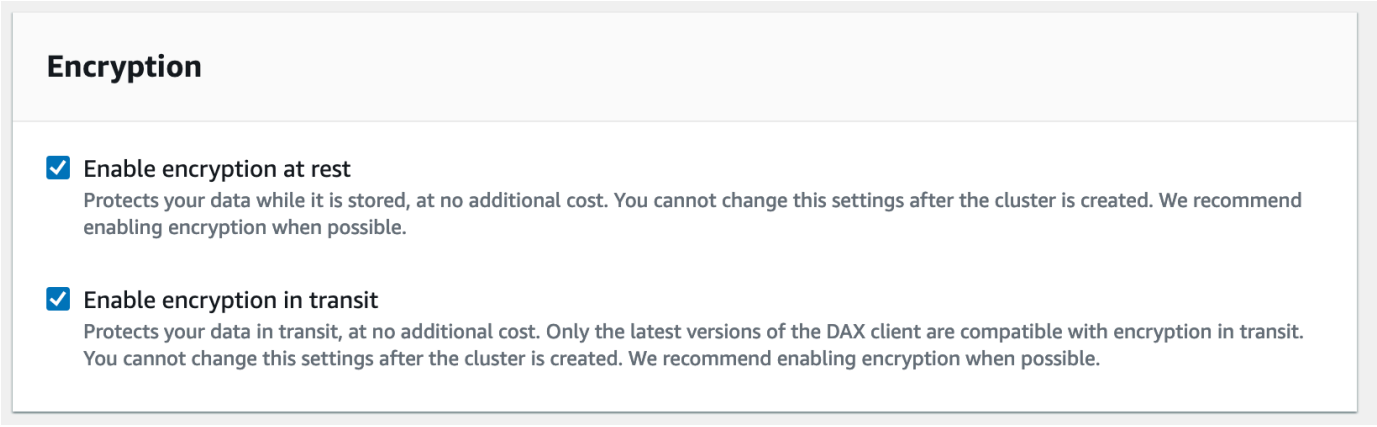
DAX 靜態加密無需額外費用 （需支付AWS KMS 加密金鑰使用費）。如需定價資訊，請參閱 Amazon DynamoDB 定價。

使用 啟用靜態加密 AWS 管理主控台

請遵循下列步驟，使用主控台在資料表上啟用 DAX 靜態加密。

啟用 DAX 靜態加密

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 DAX 下的 Clusters (叢集)。
3. 選擇 Create Cluster (建立叢集)。
4. 針對 Cluster name (叢集名稱) 輸入您叢集的短名稱。為叢集中的所有節點選擇 Node type (節點類型)，而對於叢集大小，請使用 **3** 節點。
5. 在 Encryption (加密) 中，請確認已選擇 Enable Encryption (啟用加密)。



6. 在選擇 IAM 角色、子網路群組、安全群組、以及叢集設定後，請選擇 Launch cluster (啟動叢集)。

若要確認叢集是否加密，檢查 Clusters (叢集) 窗格下的叢集詳細資訊。加密應 ENABLED (啟用)。

DAX 傳輸中加密

Amazon DynamoDB Accelerator (DAX) 支援在應用程式與 DAX 叢集之間傳輸資料時進行加密，讓您能夠在具有嚴格加密要求的應用程式中使用 DAX。

無論您是否選擇傳輸中加密，應用程式與 DAX 叢集之間的流量都會保留在 Amazon VPC 中。此流量會路由至彈性網路介面，其中包含 VPC 中附加至叢集節點的私有 IP。您可以將 VPC 作為信任界限使用，透過標準工具 (例如安全群組、搭配網路 ACL 的子網路分隔和 VPC 流程追蹤) 掌控資料的安全性。DAX 傳輸中加密會新增至此機密性基準，確保應用程式與叢集之間的所有請求和回應都經由 Transport Layer Security (TLS) 加密，並透過驗證叢集 x509 憑證，對叢集的連線進行身分驗證。如果您在建立 DAX 叢集時選擇 encryption at rest (靜態加密)，也可以加密 DAX 寫入磁碟的資料。

您可以輕鬆透過 DAX 使用傳輸中加密功能。僅需在建立新叢集時選取此選項，並在應用程式中使用任何 DAX 用戶端的最新版本。使用傳輸中加密的叢集不支援未加密的流量，因此不會設錯應用程式並略過加密。DAX 用戶端在建立連線時會使用叢集的 x509 憑證來驗證叢集的身分，以此確保 DAX 請求會前往預期的目的地。所有建立 DAX 叢集的方法都支援傳輸中加密功能：AWS 管理主控台、AWS CLI、所有開發套件和 CloudFormation。

無法在現有 DAX 叢集上啟用傳輸中加密功能。若要在現有 DAX 應用程式中啟用傳輸中加密功能，請建立已啟用傳輸中加密功能的新叢集、將應用程式的流量轉移至該叢集，然後刪除舊叢集。

使用 DAX 的服務連結 IAM 角色

Amazon DynamoDB Accelerator (DAX) 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 DAX 的一種特殊 IAM 角色類型。服務連結角色是由 DAX 預先定義，並包含服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓設定 DAX 更為簡單，因為您不必手動新增必要的許可。DAX 定義其服務連結角色的許可，除非另有定義，否則僅有 DAX 可以擔任其角色。已定義的許可包括信任政策和許可政策。該許可政策無法連接至其他任何 IAM 實體。

您必須先刪除角色的相關資源，才能刪除角色。如此可保護您的 DAX 資源，避免您不小心移除資源的存取許可。

如需支援服務連結角色之其他服務的資訊，請參閱《IAM 使用者指南》中的與 IAM 搭配運作的AWS 服務。尋找 Service-linked roles (服務連結角色) 欄中包含 Yes (是) 的服務。選擇 Yes (是) 連結，檢視該服務的服務連結角色文件。

主題

- DAX 的服務連結角色許可
- 建立 DAX 的服務連結角色
- 編輯 DAX 的服務連結角色
- 刪除 DAX 的服務連結角色

DAX 的服務連結角色許可

DAX 使用名為 `AWSServiceRoleForDAX` 的服務連結角色。此角色可讓 DAX 代表您的 DAX 叢集呼叫服務。

> **Important**
>
> `AWSServiceRoleForDAX` 服務連結角色可讓設定和維護 DAX 叢集更為容易。但是，您仍然必須授予每個叢集存取 DynamoDB 的權限，才能使用它。如需詳細資訊，請參閱DAX 存取控制。

`AWSServiceRoleForDAX` 服務連結角色信任下列服務擔任角色：

- `dax.amazonaws.com`

此角色許可政策允許 DAX 對指定資源完成下列動作：

- 在 ec2 上的動作：
  - AuthorizeSecurityGroupIngress
  - CreateNetworkInterface
  - CreateSecurityGroup
  - DeleteNetworkInterface
  - DeleteSecurityGroup
  - DescribeAvailabilityZones
  - DescribeNetworkInterfaces
  - DescribeSecurityGroups
  - DescribeSubnets
  - DescribeVpcs
  - ModifyNetworkInterfaceAttribute
  - RevokeSecurityGroupIngress

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的服務連結角色許可。

允許 IAM 實體建立 AWSServiceRoleForDAX 服務連結角色

新增以下政策陳述式到該 IAM 實體的許可中。

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "dax.amazonaws.com"}}
}
```

建立 DAX 的服務連結角色

您不需要手動建立服務連結角色，當您建立叢集時，DAX 會為您建立服務連結角色。

> **Important**
>
> 若您是在 2018 年 2 月 28 日之前使用 DAX 服務，當其開始支援服務連結角色時，DAX 會在您的帳戶中建立 AWSServiceRoleForDAX 角色。如需詳細資訊，請參閱《IAM 使用者指南》中的我的 AWS 帳戶出現的新角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您建立執行個體或叢集時，DAX 會再次為您建立服務連結角色。

編輯 DAX 的服務連結角色

DAX 不允許您編輯 AWSServiceRoleForDAX 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

刪除 DAX 的服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。不過您必須先刪除您的所有 DAX 叢集，才能刪除服務連結角色。

清除服務連結角色

您必須先確認服務連結角色沒有作用中的工作階段，並移除該角色使用的資源，之後才能使用 IAM 將其刪除。

檢查服務連結角色是否於 IAM 主控台有作用中的工作階段

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在 IAM 主控台的導覽窗格中，選擇角色。然後選擇 AWSServiceRoleForDAX 角色的名稱 (而非核取方塊)。
3. 在所選角色的 Summary (摘要) 頁面中，選擇 Access Advisor (存取 Advisor) 分頁。
4. 在 Access Advisor (存取 Advisor) 分頁中，檢閱服務連結角色的近期活動。

> **Note**
>
> 如果您不確定 DAX 是否正在使用 AWSServiceRoleForDAX 角色，可嘗試刪除該角色。若服務正在使用該角色，則刪除會失敗，而您可以檢視角色使用於哪個區域。若該角色正

受到使用，則您必須刪除您的 DAX 叢集，之後才能刪除該角色。您無法撤銷服務連結角色的工作階段。

如果您想要移除 AWSServiceRoleForDAX 角色，則必須先刪除所有 DAX 叢集。

刪除您的所有 DAX 叢集

使用這些步驟之一刪除您的每個 DAX 叢集。

刪除 DAX 叢集 (主控台)

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在導覽窗格中，選擇 DAX 下方的 Clusters (叢集)。
3. 選擇動作，然後選擇刪除。
4. 在 Delete cluster confirmation (刪除叢集確認) 方塊中，選擇 Delete (刪除)。

刪除 DAX 叢集 (AWS CLI)

請參閱 AWS CLI 命令參考中的 delete-cluster。

刪除 DAX 叢集 (API)

請參閱《Amazon DynamoDB API 參考》中的 DeleteCluster。

刪除服務連結角色

使用 IAM 手動刪除服務連結角色

使用 IAM 主控台、IAM CLI 或 IAM API 刪除 AWSServiceRoleForDAX 服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

跨 AWS 帳戶存取 DAX

假設有一個 DynamoDB Accelerator (DAX) 叢集在一個 AWS 帳戶 （帳戶 A) 中執行，而且 DAX 叢集需要從另一個 AWS 帳戶 （帳戶 B) 中的 Amazon Elastic Compute Cloud (Amazon EC2) 執行個體存取。在本教學課程中，您使用來自帳戶 B 的 IAM 角色在帳戶 B 中啟動 EC2 執行個體，然後使用 EC2 執行個體的臨時安全憑證來擔任帳戶 A 的 IAM 角色。最後，您使用在帳戶 A 中擔任 IAM 角色的臨時

安全憑證，透過 Amazon VPC 對等互連對帳戶 A 中的 DAX 叢集進行應用程式呼叫。為了執行這些工作，您在兩個 AWS 帳戶中都需要具備系統管理存取權。

> **Important**
>
> DAX 叢集無法從其他帳戶存取 DynamoDB 資料表。

主題

- 設定 IAM
- 設定 VPC
- 修改 DAX 用戶端以允許跨帳戶存取權

設定 IAM

1. 使用下列內容建立名為 `AssumeDaxRoleTrust.json` 的文字檔案，這可讓 Amazon EC2 代表您工作。

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "ec2.amazonaws.com"
               },
               "Action": "sts:AssumeRole"
           }
       ]
   }
   ```

2. 在帳戶 B 中，建立 Amazon EC2 可在啟動執行個體時使用的角色。

   ```
   aws iam create-role \
       --role-name AssumeDaxRole \
       --assume-role-policy-document file://AssumeDaxRoleTrust.json
   ```

3. 使用下列內容建立名為 AssumeDaxRolePolicy.json 的文字檔案，這會允許在帳戶 B 中的 EC2 執行個體上執行的程式碼擔任帳戶 A 中的 IAM 角色。將 *accountA* 取代為帳戶 A 的實際 ID。

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": "sts:AssumeRole",
               "Resource": "arn:aws:iam::111122223333:role/DaxCrossAccountRole"
           }
       ]
   }
   ```

4. 將該政策新增至您剛建立的角色。

   ```
   aws iam put-role-policy \
       --role-name AssumeDaxRole \
       --policy-name AssumeDaxRolePolicy \
       --policy-document file://AssumeDaxRolePolicy.json
   ```

5. 建立執行個體設定檔以允許執行個體使用該角色。

   ```
   aws iam create-instance-profile \
       --instance-profile-name AssumeDaxInstanceProfile
   ```

6. 將角色與執行個體設定檔建立關聯。

   ```
   aws iam add-role-to-instance-profile \
       --instance-profile-name AssumeDaxInstanceProfile \
       --role-name AssumeDaxRole
   ```

7. 使用下列內容建立名為 DaxCrossAccountRoleTrust.json 的文字檔案，這可讓帳戶 B 擔任帳戶 A 角色。將 *accountB* 取代為帳戶 B 的實際 ID。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/AssumeDaxRole"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

8. 在帳戶 A 中，建立帳戶 B 可以擔任的角色。

```
aws iam create-role \
    --role-name DaxCrossAccountRole \
    --assume-role-policy-document file://DaxCrossAccountRoleTrust.json
```

9. 建立一個名為 DaxCrossAccountPolicy.json 的文字檔案以允許存取 DAX 叢集。將 *dax-cluster-arn* 取代為 DAX 叢集的正確 Amazon Resource Name (ARN)。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
```

```
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-east-1:111122223333:cache/dax-
cluster-name"
        }
    ]
}
```

10. 在帳戶 A 中，將政策新增至角色。

```
aws iam put-role-policy \
    --role-name DaxCrossAccountRole \
    --policy-name DaxCrossAccountPolicy \
    --policy-document file://DaxCrossAccountPolicy.json
```

設定 VPC

1. 尋找帳戶 A DAX 叢集的子網路群組。將 *cluster-name* 取代為帳戶 B 必須存取的 DAX 叢集名稱。

```
aws dax describe-clusters \
    --cluster-name cluster-name
    --query 'Clusters[0].SubnetGroup'
```

2. 使用該*#####*尋找該叢集的 VPC。

```
aws dax describe-subnet-groups \
    --subnet-group-name subnet-group \
    --query 'SubnetGroups[0].VpcId'
```

3. 使用該 *vpc-id* 尋找該 VPC 的 CIDR。

```
aws ec2 describe-vpcs \
    --vpc vpc-id \
    --query 'Vpcs[0].CidrBlock'
```

4. 從帳戶 B，使用不同於上一個步驟中找到的 CIDR 的非重疊 CIDR 來建立 VPC。然後，建立至少一個子網路。您可以在 或 中使用 VPC 建立精靈AWS CLI。 AWS 管理主控台

5. 從帳戶 B，請求與帳戶 A VPC 的對等連線，如建立和接受 VPC 對等互連中所述。從帳戶 A 接受連線。

6. 從帳戶 B 尋找新 VPC 的路由表。將 *vpc-id* 取代為您在帳戶 B 中建立的 VPC ID。

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id,Values=vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

7. 新增路由，將目的地為帳戶 A CIDR 的流量傳送至 VPC 對等互連。請記得將每個 *user input placeholder* 取代為您帳戶的正確值。

```
aws ec2 create-route \
    --route-table-id accountB-route-table-id \
    --destination-cidr accountA-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

8. 使用您之前找到的 *vpc-id*，從帳戶 A 尋找 DAX 叢集的路由表。

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id, Values=accountA-vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

9. 從帳戶 A，新增路由，將目的地為帳戶 B CIDR 的流量傳送至 VPC 對等互連。使用您帳戶的正確值取代每個 *user input placeholder*。

```
aws ec2 create-route \
    --route-table-id accountA-route-table-id \
    --destination-cidr accountB-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

10. 從帳戶 B，在先前建立的 VPC 中啟動 EC2 執行個體。將其命名為 `AssumeDaxInstanceProfile`。您可以使用 中的啟動精靈 AWS 管理主控台 或 AWS CLI。記下執行個體的安全群組。

11. 從帳戶 A 尋找 DAX 叢集所使用的安全群組。記得將 *cluster-name* 取代為您的 DAX 叢集名稱。

```
aws dax describe-clusters \
    --cluster-name cluster-name \
    --query 'Clusters[0].SecurityGroups[0].SecurityGroupIdentifier'
```

12. 更新 DAX 叢集的安全群組，以允許來自您在帳戶 B 中建立之 EC2 執行個體的安全群組的傳入流量。請記得使用正確的帳戶值取代 *user input placeholders*。

```
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
```

此時，帳戶 B 的 EC2 執行個體上的應用程式可以使用執行個體設定檔來擔任 `arn:aws:iam::accountA-id:role/DaxCrossAccountRole` 角色並使用 DAX 叢集。

修改 DAX 用戶端以允許跨帳戶存取權

> **Note**
>
> AWS Security Token Service (AWS STS) 登入資料是臨時登入資料。某些用戶端會自動處理重新整理，而有些則需要額外的邏輯來重新整理憑證。我們建議您遵循適當說明文件的指引。

Java

本節可協助您修改現有的 DAX 用戶端程式碼，以允許跨帳戶 DAX 存取。如果您還沒有 DAX 用戶端程式碼，可以在 Java 與 DAX 教學課程中找到可行的程式碼範例。

1. 新增以下匯入專案。

```
import com.amazonaws.auth.STSAssumeRoleSessionCredentialsProvider;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import
 com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
```

2. 從 取得登入資料提供者 AWS STS 並建立 DAX 用戶端物件。請記得將每個 *user input placeholder* 取代為您帳戶的正確值。

```
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

本節可協助您修改現有的 DAX 用戶端程式碼，以允許跨帳戶 DAX 存取。如果您還沒有 DAX 用戶端程式碼，可以在 .NET 和 DAX 教學課程中找到可行的程式碼範例。

1. 將 AWSSDK.SecurityToken NuGet 套件新增到解決方案。

```
<PackageReference Include="AWSSDK.SecurityToken" Version="latest version" />
```

2. 使用 SecurityToken 和 SecurityToken.Model 套件。

```
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
```

3. 從 AmazonSimpleTokenService 取得臨時憑證並建立 ClusterDaxClient 物件。請記得將每個 *user input placeholder* 取代為您帳戶的正確值。

```
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

本節可協助您修改現有的 DAX 用戶端程式碼，以允許跨帳戶 DAX 存取。如果您還沒有 DAX 用戶端程式碼，可以在 GitHub 上找到可行的程式碼範例。

1. 匯入 AWS STS 和 工作階段套件。

```
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. 從 AmazonSimpleTokenService 取得臨時憑證並建立 DAX 用戶端物件。請記得將每個 *user input placeholder* 取代為您帳戶的正確值。

```
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

本節可協助您修改現有的 DAX 用戶端程式碼，以允許跨帳戶 DAX 存取。如果您還沒有 DAX 用戶端程式碼，可以在 Python 和 DAX 教學課程中找到可行的程式碼範例。

1. 匯入 boto3。

```
import boto3
```

2. 從 sts 取得臨時憑證並建立 AmazonDaxClient 物件。請記得將每個 *user input placeholder* 取代為您帳戶的正確值。

```
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

本節可協助您修改現有的 DAX 用戶端程式碼，以允許跨帳戶 DAX 存取。如果您還沒有 DAX 用戶端程式碼，可以在 Node.js 和 DAX 教學課程中找到可行的程式碼範例。請記得將每個 *user input placeholder* 取代為您帳戶的正確值。

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');
const region = 'region';
const endpoints = [daxEndpoint1, ...];

const getCredentials = async() => {
  return new Promise((resolve, reject) => {
    const sts = new AWS.STS();
    const roleParams = {
      RoleArn: 'arn:aws:iam::accountA:role/RoleName',
      RoleSessionName: 'tryDax',
```

```
    };
    sts.assumeRole(roleParams, (err, session) => {
      if(err) {
        reject(err);
      } else {
        resolve({
          accessKeyId: session.Credentials.AccessKeyId,
          secretAccessKey: session.Credentials.SecretAccessKey,
          sessionToken: session.Credentials.SessionToken,
        });
      }
    });
  });
};

const createDaxClient = async() => {
  const credentials = await getCredentials();
  const daxClient = new AmazonDaxClient({endpoints: endpoints, region: region,
 accessKeyId: credentials.accessKeyId, secretAccessKey: credentials.secretAccessKey,
 sessionToken: credentials.sessionToken});
  return new AWS.DynamoDB.DocumentClient({service: daxClient});
};

createDaxClient().then((client) => {
  client.get(...);
  ...
}).catch((error) => {
  console.log('Caught an error: ' + error);
});
```

DAX 叢集調整大小指南

本指南會為您提供建議，以便您為應用程式選擇合適的 Amazon DynamoDB Accelerator (DAX) 叢集大小和節點類型。這些指示會逐步引導您預估應用程式的 DAX 流量、選取叢集組態並加以測試。

如果您目前有 DAX 叢集，並且想評估叢集是否具有合適的節點數量與大小，請參考擴展 DAX 叢集。

主題

- 概觀
- 預估流量
- 負載測試

概觀

不論您是要建立新的叢集或維護現有叢集，為工作負載適當擴展 DAX 叢集都很重要。隨著時間經過與應用程式工作負載變更，您也應該定期審視擴展決策來確保合適性。

此程序通常遵循下列步驟：

1. 預估流量。在此步驟中，您將預測應用程式傳送至 DAX 的流量、流量的性質 (讀取比對寫入操作)，以及預計的快取命中率。
2. 負載測試。在此步驟中，您將建立叢集並向叢集傳送流量，來驗證上一個步驟的預估。請重複這個步驟直到找到合適的叢集組態為止。
3. 生產監控。當應用程式在生產階段使用 DAX 時，務必監控叢集以持續確認叢集仍會隨著工作負載變更正確調整規模。

預估流量

典型 DAX 工作負載的特性有三個主要因素：

- 快取命中率
- 每秒讀取容量單位 (RCU)
- 每秒寫入容量單位 (WCU)

預估快取命中率

如果您已有 DAX 叢集，則可以使用 `ItemCacheHits` 和 `ItemCacheMisses` Amazon CloudWatch 指標來判斷快取命中率。快取命中率等於 `ItemCacheHits` /(`ItemCacheHits` + `ItemCacheMisses`)。如果您的工作負載包含 `Query` 或 `Scan` 操作，也請查看 `QueryCacheHits`、`QueryCacheMisses`、`ScanCacheHits` 與 `ScanCacheMisses` 指標。應用程式間的快取命中率會有所差異，而且非常容易受到叢集的存留時間 (TTL) 設定影響。使用 DAX 的應用程式通常有 85% 至 95% 的命中率。

預估讀取和寫入容量單位

如果您的應用程式已經有 DynamoDB 資料表，請查看 `ConsumedReadCapacityUnits` 及 `ConsumedWriteCapacityUnits` CloudWatch 指標。使用 `Sum` 統計數字，除以週期中的秒數。

如果您也已經有 DAX 叢集，請記得此 DynamoDB `ConsumedReadCapacityUnits` 指標只會考量快取未中。因此，如果要了解 DAX 叢集每秒處理的讀取容量單位，請將數字除以快取未中率 (即 1：快取命中率)。

如果您還沒有 DynamoDB 資料表，請參閱讀取和寫入容量單位的相關文件，來根據應用程式的預估請求率、每個請求的存取項目數和項目大小來預估流量。

預估流量時，請將後續增長和預期/非預期的尖峰納入考量，來確保叢集有足夠的流量增加空間。

負載測試

預估流量後的下一步是測試負載叢集組態。

1. 進行第一次負載測試時，建議您先從 `dax.r4.large` 節點類型開始，這是最低成本固定效能、記憶體最佳化的節點類型。
2. 容錯叢集至少需要三個節點，分布於三個可用區域。在此情況下，如果有一個可用區域失效，則可用區域的有效數量會減少三分之一。進行第一次負載測試時，建議您先從兩個節點的叢集開始，這會模擬三個節點叢集中有一個可用區域失效的情形。
3. 負載測試期間，請對您的測試叢集推動持續流量 (如上一步驟中所預估)。
4. 在負載測試期間監控叢集的效能。

理想情況下，您在負載測試期間推動的流量概況，應該盡可能地符合應用程式實際流量。這包括操作的分配 (例如 70% `GetItem`、25% `Query` 和 5% `PutItem`)、各操作的請求率、每個請求的項目存取數和項目大小的分佈。如果要達到與應用程式預期相當的快取命中率，請密切注意測試流量中的金鑰分佈情形。

> **Note**
>
> 對 T2 節點類型 (`dax.t2.small` 與 `dax.t2.medium`) 進行負載測試時請小心。T2 節點類型提供高載 CPU 效能，會根據節點的 CPU 點數餘額隨著時間變化。在 T2 節點上執行的 DAX 叢集可能看似正常運作，但只要有任何節點暴增超過本身執行個體的基準效能，節點就會消耗累積的 CPU 點數餘額。如果點數餘額不足，效能會逐漸降低至基準效能等級。

請在負載測試期間監控您的 DAX 叢集，來判斷為負載測試使用的節點類型是否合適。此外，也請在負載測試期間監控請求率與快取命中率，來確保測試基礎設施真的有在推動您想要的流量。

您應該留意所選叢集執行個體類型的網路位元組使用量。若超過 Amazon EC2 執行個體的可用基準頻寬，即表示您的叢集可能無法承受應用程式的工作負載，且需要進行擴展。

如果負載測試指出選取的叢集組態無法承受應用程式的工作負載，您應切換至更大的節點類型，尤其是在叢集的主節點上發現高 CPU 使用率、高移出率或高快取記憶體時更是如此。如果命中率一直很高，而且讀取與寫入流量的比率也很高的話，則建議您考慮為叢集新增更多節點。如果需要額外指導來了解何時該使用更大的節點類型 (垂直擴展) 或新增更多節點 (水平擴展)，則請參閱 擴展 DAX 叢集。

變更叢集組態後，請重複進行負載測試。

適用於 DynamoDB 資料表的資料建模

在我們深入探討資料建模之前，請務必了解一些 DynamoDB 基礎知識。DynamoDB 是一個使用鍵值的 NoSQL 資料庫，擁有靈活的結構描述。除了每個項目的索引鍵屬性之外，資料屬性集可以統整在一起，也可以各自獨立。DynamoDB 索引鍵結構描述的形式可以是簡單主索引鍵，其中分割區索引鍵可唯一識別項目，也可以採用複合主鍵的形式，其中使用分割區索引鍵和排序索引鍵的組合來唯一定義項目。分割區索引鍵會經過雜湊處理，以判斷資料的實體位置並加以擷取。因此，請務必選擇高基數且可水平擴充的屬性作為分割區索引鍵，以確保資料均勻分佈。排序索引鍵屬性在索引鍵結構描述中是選用項目，而且有排序索引鍵就可讓您在 DynamoDB 中建立一對多關係的模型，以及建立項目集合。排序索引鍵也稱為範圍索引鍵，可用來排序項目集合中的項目，也能進行靈活的範圍型操作。

如需 DynamoDB 索引鍵結構描述的詳細資訊和最佳實務，可參考下列內容：

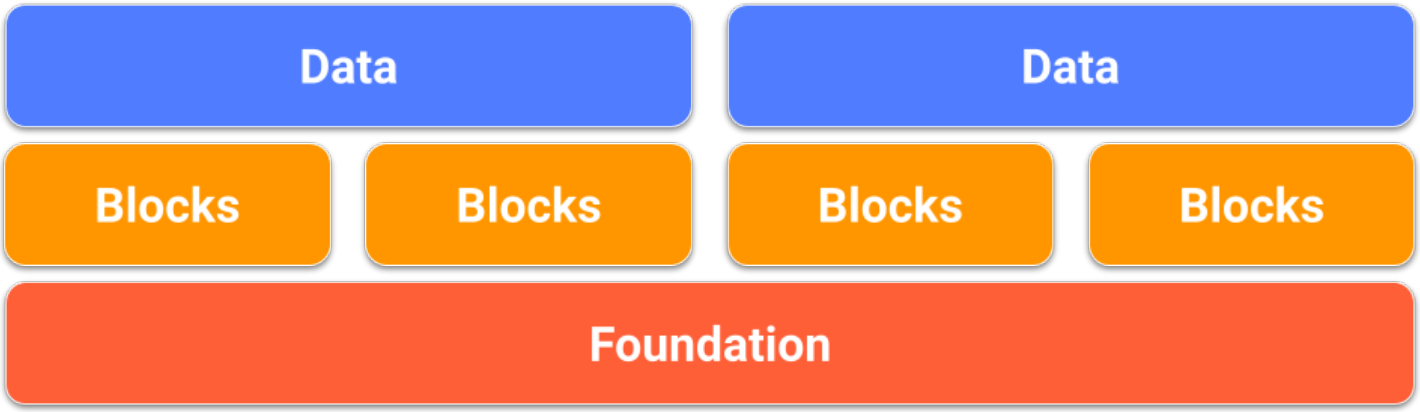
- the section called “DynamoDB 的分割區與資料分配”
- the section called “分割區索引鍵設計”
- the section called “排序索引鍵設計”
- 選擇適合的 DynamoDB 分割區索引鍵

通常需要有次要索引才能在 DynamoDB 中支援其他查詢模式。次要索引是影子資料表，其中相同的資料會透過與基礎資料表不同的索引鍵結構描述來組織。本機次要索引 (LSI) 與基礎資料表共用相同的分割區索引鍵，並允許使用替代的排序索引鍵來共用基礎資料表的容量。全域次要索引 (GSI) 可以有與基礎資料表不同的分割區索引鍵以及不同的排序索引鍵屬性，這表示 GSI 的輸送量管理與基礎資料表無關。

如需次要索引和最佳實務的進一步詳細資訊，可參考下列內容：

- the section called “使用索引”
- the section called “次要索引”

現在讓我們來進一步了解資料建模。在 DynamoDB 或任何類似 NoSQL 資料庫上設計靈活且高度最佳化結構描述的程序，可能是一項要學習的極具挑戰性技能。本單元的目標是協助您發展心智圖，以設計讓您從使用案例進入生產環境的結構描述。我們將首先介紹任何設計 (單一資料表與多資料表設計) 的基礎選擇。接著，我們會檢閱多種設計模式 (建置區塊)，這些模式可供您的應用程式用來實現各種組織性或效能結果。最後，我們會介紹適用於不同使用案例和產業的各種完整結構描述設計套件。



主題

- 物品集合 - 如何在 DynamoDB 中建立一對多關係的模型
- DynamoDB 中的資料建模基礎
- DynamoDB 中的資料建模建置區塊
- DynamoDB 中的資料建模結構描述設計套件
- 在 DynamoDB 中製作關聯式資料模型的最佳實務

物品集合 - 如何在 DynamoDB 中建立一對多關係的模型

在 DynamoDB 中，物品集合是共享相同分割區索引鍵值的物品群組，這表示這些物品是相關聯的。物品集合是在 DynamoDB 中建立一對多關係模型的主要機制。物品集合只能存在於配置為使用複合主鍵的資料表或索引。

> **Note**
>
> 物品集合可以存在於基底資料表中，也可以存在於次要索引中。有關物品集合如何與索引互動的詳細資訊，請參閱 本機次要索引中的項目集合。

請考慮下表，其中顯示三位不同的使用者及其遊戲內物品清單：

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

對於每個集合中的特定物品，排序索引鍵是由用於將資料分組的資訊 (例如 `inventory::armor`、`inventory::weapon` 或 `info`) 組成的串連。每個物品集合可以具有不同組合的屬性作為排序索引鍵。使用者 `account1234` 具有 `inventory::weapons` 物品，而使用者 `account1387` 沒有 (因為他們還沒有找到)。使用 `account1138` 只使用兩個物品作為排序索引鍵 (因為他們還沒有物品清單)，而其他使用者使用三個物品。

DynamoDB 允許您選擇性地從這些物品集合中擷取物品，以執行以下操作：

- 從特定使用者擷取所有物品
- 從特定使用者擷取一個物品
- 擷取屬於特定使用者的特定類型的所有物品

使用物品集合組織資料來加快查詢

在此範例中，這三個物品集合中的每個物品都代表一個玩家，和我們根據遊戲和玩家的存取模式選擇的資料模型。遊戲需要什麼資料？ 什麼時候需要？ 需要的頻率為何？ 這樣做的成本是多少？ 這些資料建模決策是根據這些問題的答案做出。

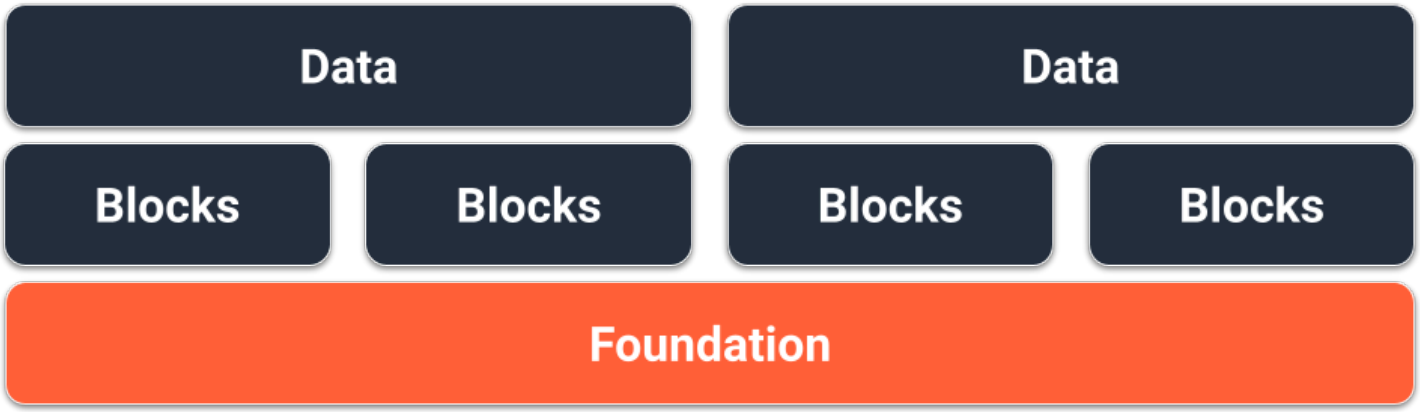
在這個遊戲中，有一個頁面為玩家呈現其武器庫存清單，還有另一個頁面呈現盔甲。玩家開啟他們的庫存清單時，會先顯示武器，因為我們希望該頁面能夠以極快速度載入，之後才會載入後續的庫存清單頁

面。隨著玩家獲得更多的遊戲內物品，這些物品類型可能會相當大，因此我們決定每個庫存清單頁面在資料庫的玩家物品集合中，都獨立自成一個項目。

以下區段詳細介紹如何透過 `Query` 操作與物品集合互動。

DynamoDB 中的資料建模基礎

本節藉由檢驗兩種類型的資料表設計來介紹基礎層：單一資料表和多資料表。



單一資料表設計基礎

在我們 DynamoDB 結構描述的基礎上，其中一個選擇是單一資料表設計。單一資料表設計是一種模式，可讓您將多種資料類型 (實體) 儲存在單一 DynamoDB 資料表中。它旨在透過消除維護多個資料表和它們之間複雜關係的需要，來最佳化資料存取模式，提高效能並降低成本。這是有可能的，因為 DynamoDB 會將具有相同分割區索引鍵 (稱為項目集合) 的項目儲存在彼此相同的分割區上。在此設計中，不同類型的資料會以項目的形式儲存在同一個資料表中，並且每個項目由一個唯一的排序索引鍵來識別。

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

優點

- 資料位置性，可在單一資料庫呼叫中支援多個實體類型的查詢
- 降低讀取的整體財務和延遲成本：
  - 對總計小於 4KB 的兩個項目的單一查詢是 0.5 RCU 最終一致
  - 對總計小於 4KB 的兩個項目的兩個查詢是 1 RCU 最終一致 (每個 0.5 RCU)
  - 傳回兩個單獨資料庫呼叫的平均時間將高於單一呼叫
- 減少要管理的資料表數量：
  - 不需要跨多個 IAM 角色或政策維護許可
  - 資料表的容量管理會在所有實體中進行平均，通常會產生更可預測的取用模式
  - 監控需要更少的警示
  - 客戶管理的加密金鑰只需在一個資料表上進行輪換
- 資料表的平滑流量：
  - 透過將多種使用模式彙總到同一個資料表，整體使用量往往更平滑 (股票指數的績效往往比任何個股表現更平滑)，這對於透過佈建模式資料表達到更高的使用率有更好的效果

缺點

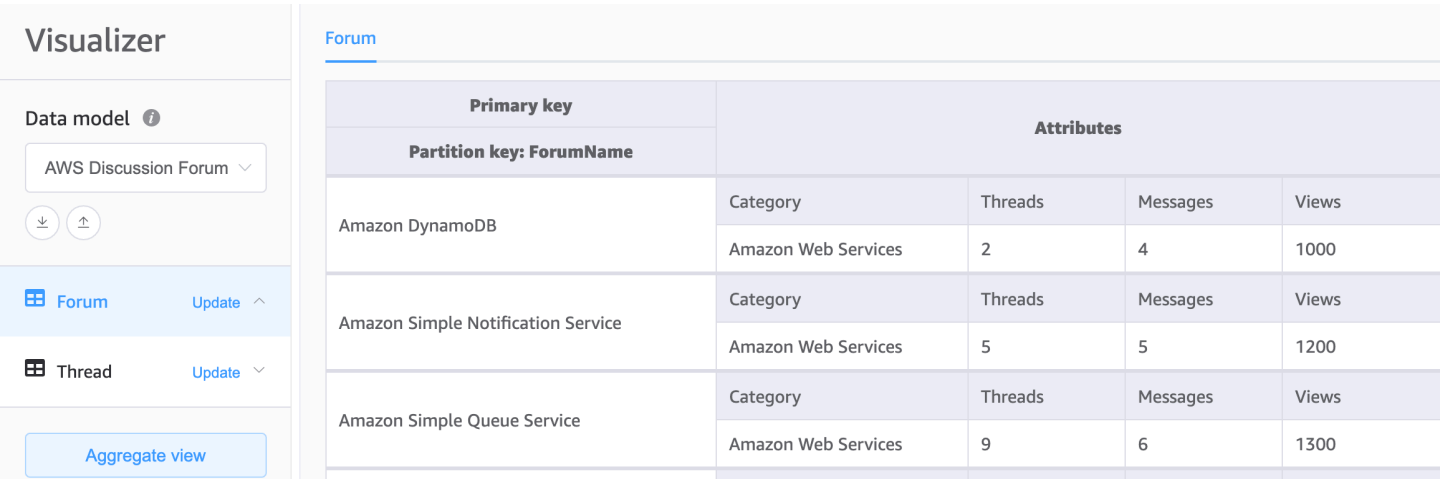
- 與關聯式資料庫相比，由於矛盾的設計，學習曲線可能會很陡峭
- 所有實體類型的資料需求必須一致
  - 備份是採用全有或全無原則，因此如果資料不是關鍵任務資料，請考慮將其保存在單獨的資料表中
  - 資料表加密會在所有項目之間共用。對於具有個別租戶加密需求的多租戶應用程式，將需要用戶端加密
  - 混合了歷史資料和操作資料的資料表，不會因啟用不常存取的儲存類別而獲得更多優勢。如需詳細資訊，請參閱DynamoDB 資料表類別
- 即使只需處理一部分實體，所有變更的資料都會傳送至 DynamoDB Streams。
  - 由於有 Lambda 事件篩選條件，這在使用 Lambda 時不會影響您的帳單，但在使用 Kinesis Consumer Library 時會增加成本
- 使用 GraphQL 時，單一資料表設計將更加難以實現
- 如果您使用更高層級的 SDK 客戶端 (如 Java 的 `DynamoDBMapper` 或增強型用戶端)，那處理結果會比較困難，因為同一回應中的項目可能與不同的類別有關

使用情況

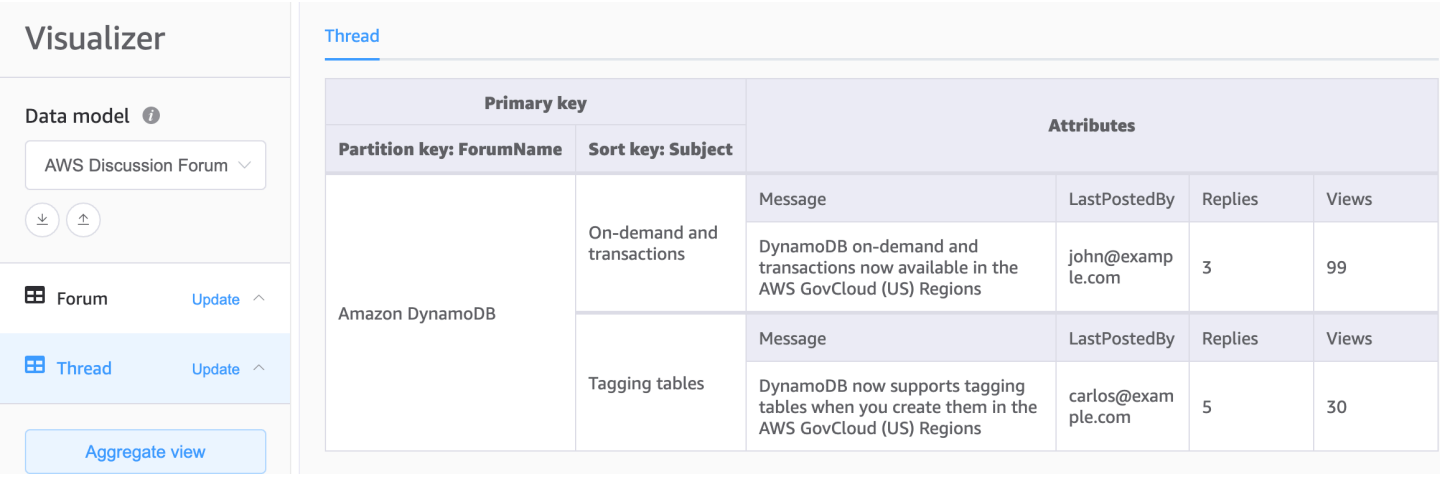
單一資料表設計非常適合經常同時查詢多個實體類型的應用程式，或需要維護不同資料類型之間關係的應用程式。如果您的存取模式受益於資料地區性，且您想要盡可能減少管理多資料表的額外負荷，這種設計就特別有效。

多資料表設計基礎

在我們 DynamoDB 結構描述的基礎上，第二個選擇是多資料表設計。多資料表設計是一種模式，更像是傳統資料庫設計，您可以在每個 DynamoDB 資料表中儲存單一類型 (實體) 資料。每個資料表中的資料仍會依據分割索引鍵進行組織，因此單一實體類型內的效能會針對可擴展性和效能進行最佳化，但是跨多資料表的查詢必須獨立完成。



Primary key	Attributes			
Partition key: ForumName				
Amazon DynamoDB	Category	Threads	Messages	Views
	Amazon Web Services	2	4	1000
Amazon Simple Notification Service	Category	Threads	Messages	Views
	Amazon Web Services	5	5	1200
Amazon Simple Queue Service	Category	Threads	Messages	Views
	Amazon Web Services	9	6	1300



Primary key		Attributes			
Partition key: ForumName	Sort key: Subject				
Amazon DynamoDB	On-demand and transactions	Message	LastPostedBy	Replies	Views
		DynamoDB on-demand and transactions now available in the AWS GovCloud (US) Regions	john@example.com	3	99
	Tagging tables	Message	LastPostedBy	Replies	Views
		DynamoDB now supports tagging tables when you create them in the AWS GovCloud (US) Regions	carlos@example.com	5	30

優點

- 對於那些不習慣使用單一資料表設計的人來說，該設計更簡單
- 由於每個解析程式會對應到單一實體 (資料表)，因此 GraphQL 解析程式的實現更容易
- 允許跨不同實體類型的唯一資料需求：
  - 可為關鍵任務的單一資料表進行備份
  - 可以每個資料表進行管理的資料表加密。對於具有個別租戶加密需求的多租戶應用程式，不同的租戶資料表可讓每個客戶擁有自己的加密金鑰
  - 不常存取的儲存類別只能在具有歷史資料的資料表上啟用，以實現完整的成本節省優勢。如需詳細資訊，請參閱DynamoDB 資料表類別
- 每個資料表都有自己的變更資料串流，允許針對每種類型的項目 (而非單一整合式處理器) 設計專用的 Lambda 函式

缺點

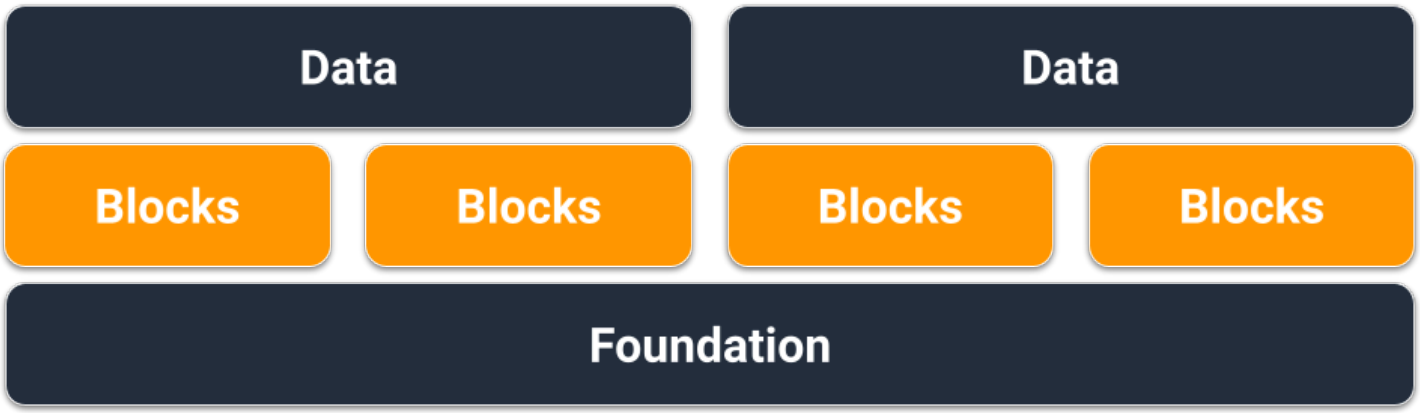
- 對於需要跨多資料表資料的存取模式，則需要從 DynamoDB 進行多次讀取，而且可能需要在用戶端程式碼上處理/加入資料。
- 對多資料表進行操作和監控需要更多 CloudWatch 警示，並且每個資料表必須獨立進行擴展
- 每個資料表的許可都需要單獨管理。未來新增資料表都將需要變更任何必要的 IAM 角色或政策

使用情況

如果您的應用程式的存取模式不需要一併查詢多個實體或資料表，那麼多資料表設計會是良好且足夠的方法。

DynamoDB 中的資料建模建置區塊

本節介紹建置區塊層，為您提供了可在應用程式中使用的設計模式。



主題

- 複合排序索引鍵建置區塊
- 多租用建置區塊
- 稀鬆索引建置區塊
- 建置區塊生存時間
- 封存建置區塊生存時間
- 垂直分割建置區塊
- 寫入碎片建置區塊

複合排序索引鍵建置區塊

當人們思考 NoSQL 時，他們也可能認為它是非關聯式的。最終，DynamoDB 結構描述中無法放入關係是不合理的，它們只是看起來與關聯式資料庫及其外部索引鍵不同而已。其中一項可以用來在 DynamoDB 中開發資料邏輯階層的最關鍵模式，就是複合排序索引鍵。最常見的設計模式，是以井字號將階層中的每一層 (父層 > 子層 > 孫層) 隔開。例如 `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

雖然 DynamoDB 中的分割區索引鍵一律需要確切的值才能查詢資料，但我們可以將部分條件從左至右套用至排序索引鍵，類似於周遊二元樹。

在上面的範例中，我們有一間需要在使用者工作階段中維護購物車的電子商務商店。每當使用者登入時，他們可能希望看到整個購物車，包括儲存供日後使用的項目。但是，在他們進入結帳程序時，只應載入使用中購物車內的項目以供購買。由於這兩個 `KeyConditions` 都明確要求 CART 排序索引鍵，因此 DynamoDB 會在讀取時直接忽略額外的願望清單資料。雖然儲存的項目和使用中的項目都屬於同一個購物車，但我們需要在應用程式的不同部分中以不同的方式處理它們，因此在排序索引鍵的字首上套用 `KeyCondition`，是僅擷取應用程式每個部分所需資料的最佳方式。

此建置區塊的主要特徵

- 相關項目彼此在本機儲存，以便有效存取資料
- 使用 `KeyCondition` 表達式，可以選擇性擷取階層的子集，表示沒有浪費的 RCU
- 應用程式的不同部分可以將其項目儲存在特定字首下，防止被覆寫的項目或衝突的寫入

多租用建置區塊

許多客戶使用 DynamoDB 來託管多租戶應用程式的資料。針對這些案例，我們想要設計結構描述，以便將單一租戶的所有資料保留在其本身的資料表邏輯分割區中。這會利用「項目集合」的概念，該術語是指 DynamoDB 資料表中具有相同分割區索引鍵的所有項目。如需 DynamoDB 如何處理多租戶的詳細資訊，請參閱 Multitenancy on 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]

在此範例中，我們正在執行一個可能有成千上萬使用者的相片託管網站。每個使用者最初只會將相片上傳到自己的個人資料，但預設情況下，我們不允許使用者查看任何其他使用者的相片。理想情況下，應該為每個使用者呼叫 API 的授權新增額外隔離等級，以確保他們只從自己的分割請求資料，但在結構描述等級中，唯一分割區索引鍵就足夠了。

此建置區塊的主要特徵

- 任何一位使用者或租用戶讀取的資料量只能與其分割區中的項目總量相同
- 由於帳戶關閉或合規要求而需刪除租戶資料，可以巧妙而廉價的方式完成。只要執行分割區索引鍵等於其租戶 ID 的查詢，然後為傳回的每個主索引鍵執行 `DeleteItem` 操作

> **Note**
>
> 專為多租用戶而設計，您可以在單一資料表中使用不同的加密金鑰提供者來安全地隔離資料。適用於 Amazon DynamoDB 的AWS 資料庫加密 SDK 可讓您在 DynamoDB 工作負載中納入用

戶端加密。您可以執行屬性層級加密，讓您在將特定屬性值儲存在 DynamoDB 資料表之前先加密，並搜尋加密的屬性，而無需事先解密整個資料庫。

稀鬆索引建置區塊

有時，存取模式需要尋找符合罕見項目的項目，或是接收狀態的項目 (需要提升回應)。我們不會定期查詢這些項目的整個資料集，而是可以利用全域次要索引 (GSI) 稀疏的載入資料的事實。這表示只有基礎資料表中具有在索引中定義之屬性項目，才會複寫至索引。

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

在此範例中，我們可看到 IOT 使用案例，在欄位中的每個裝置都會定期回報狀態。對於大多數報告，我們期望裝置會報告一切正常，但有時可能會出現故障，必須提升給維修技術人員處理。對於具有升級的報告而言，系統會新增該項目的屬性 `EscalatedTo`，但不會顯示該屬性。本範例中的 GSI 由 `EscalatedTo` 分割，由於 GSI 會從基礎資料表中引入金鑰，我們仍然可以看到哪個 DeviceID 報告了故障以及何時報告故障。

雖然在 DynamoDB 中的讀取成本比寫入低，但稀疏索引是非常強大的工具，適用於特定類型項目的執行個體罕見，但讀取以找到它們卻很常見的使用案例。

此建置區塊的主要特徵

- 稀疏 GSI 的寫入和儲存成本僅適用於符合金鑰模式的項目，因此 GSI 的成本可以大幅低於已複製所有項目的其他 GSI
- 複合排序索引鍵仍可用來進一步縮小符合所需查詢的項目範圍，例如，排序索引鍵可以使用時間戳記，以便只檢視最後 X 分鐘內報告的錯誤 (`SK > 5 minutes ago, ScanIndexForward: False`)

建置區塊生存時間

大多數資料都有一定的持續時間，可以視為值得保存在主要資料儲存中。為了協助在 DynamoDB 中的資料老化，它具有稱作存留時間 (TTL) 的功能。TTL 功能可讓您在資料表層級定義特定屬性，這些屬性需要監視具有 epoch 時間戳記 (已為過去的事) 的項目。這使您可以免費從資料表中刪除過期記錄。

> **Note**
>
> 如果您使用全域資料表版本 2019.11.21 (目前版本) 且您也使用生存時間功能，DynamoDB 會複製 TTL 刪除至所有資料表複本。初始 TTL 刪除不會在 TTL 過期發生時消耗區域中的寫入容量。但是，對複本資料表複寫的 TTL 刪除會消耗每個複本區域中的複寫寫入容量，並且將收取適當的費用。

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

在此範例中，我們有一個應用程式專門設計用來讓使用者建立短期訊息。在 DynamoDB 中建立訊息時，應用程式的程式碼會將 TTL 屬性設定為未來七天的日期。在大約七天內，DynamoDB 會看到這些項目的 epoch 時間戳記已經過去，並將其刪除。

由於 TTL 完成的刪除是免費的，因此強烈建議您使用此功能從資料表中移除歷史資料。這將減少每個月的整體儲存體費用，並可能降低使用者讀取成本，因為他們查詢擷取的資料將較少。雖然 TTL 是在

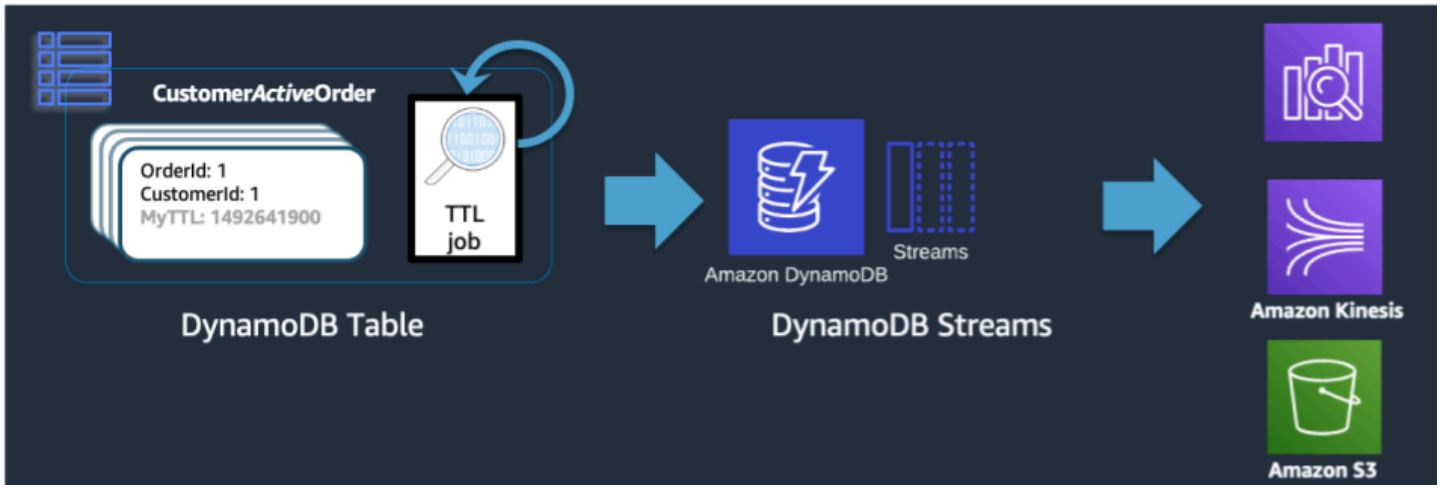
資料表層級啟用的，但要為哪些項目或實體建立 TTL 屬性，以及將 epoch 時間戳記設定為未來多久時間，將由您決定。

此建置區塊的主要特徵

- TTL 刪除會在幕後執行，不會影響您的資料表效能
- TTL 是一種非同步處理程序，大約每六個小時執行一次，但可能需要超過 48 小時才能刪除過期的記錄
  - 如果必須在 48 小時內清除過時的資料，請勿在鎖定記錄或狀態管理等使用案例中依賴 TTL 進行刪除
- 您可以將 TTL 屬性命名為有效的屬性名稱，但值必須是數字類型

封存建置區塊生存時間

雖然 TTL 是從 DynamoDB 刪除舊資料的有效工具，但許多使用案例都需要將資料封存的保存時間比主要資料儲存更長。在此情況下，我們可以利用 TTL 對記錄的定時刪除，將過期的記錄推送到長期資料儲存。



在 DynamoDB 完成 TTL 刪除時，它仍會作為 `Delete` 事件推送到 DynamoDB 串流中。在 DynamoDB TTL 是執行刪除的執行者時，`principal:dynamodb` 的串流記錄上會有一個屬性。使用 Lambda 訂閱者到 DynamoDB Stream，我們只能為 DynamoDB 主體屬性套用事件篩選條件，並知道符合該篩選條件的任何記錄都會推送到封存存放區，例如 Amazon Glacier。

此建置區塊的主要特徵

- 一旦歷史項目不再需要 DynamoDB 的低延遲讀取，將它們遷移到較冷的儲存服務，例如 Amazon Glacier，可以大幅降低儲存成本，同時滿足使用案例的資料合規需求

- 如果資料保留在 Amazon S3 中，則可以使用具成本效益的分析工具 (如 Amazon Athena 或 Redshift Spectrum) 來執行資料的歷史分析

垂直分割建置區塊

熟悉文件模型資料庫的使用者將熟悉在單一 JSON 文件中儲存所有相關資料的想法。雖然 DynamoDB 支援 JSON 資料類型，但不支援在巢狀 JSON 上執行 `KeyConditions`。由於 `KeyConditions` 會決定要從磁碟讀取多少資料，以及查詢有效消耗了多少 RCU，因此可能會導致大規模的不足。為了更最佳化 DynamoDB 的寫入和讀取，我們建議您將文件的個別實體分割為個別 DynamoDB 項目，也稱為垂直分割。

```
{
        ”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

如上所示，垂直分割是實際操作中單一資料表設計的關鍵範例，但如果需要，也可以跨多個資料表進行實作。由於 DynamoDB 帳單會以 1 KB 的增量進行寫入，因此理想情況下，您應該以產生小於 1 KB 項目的方式對文件進行分割。

此建置區塊的主要特徵

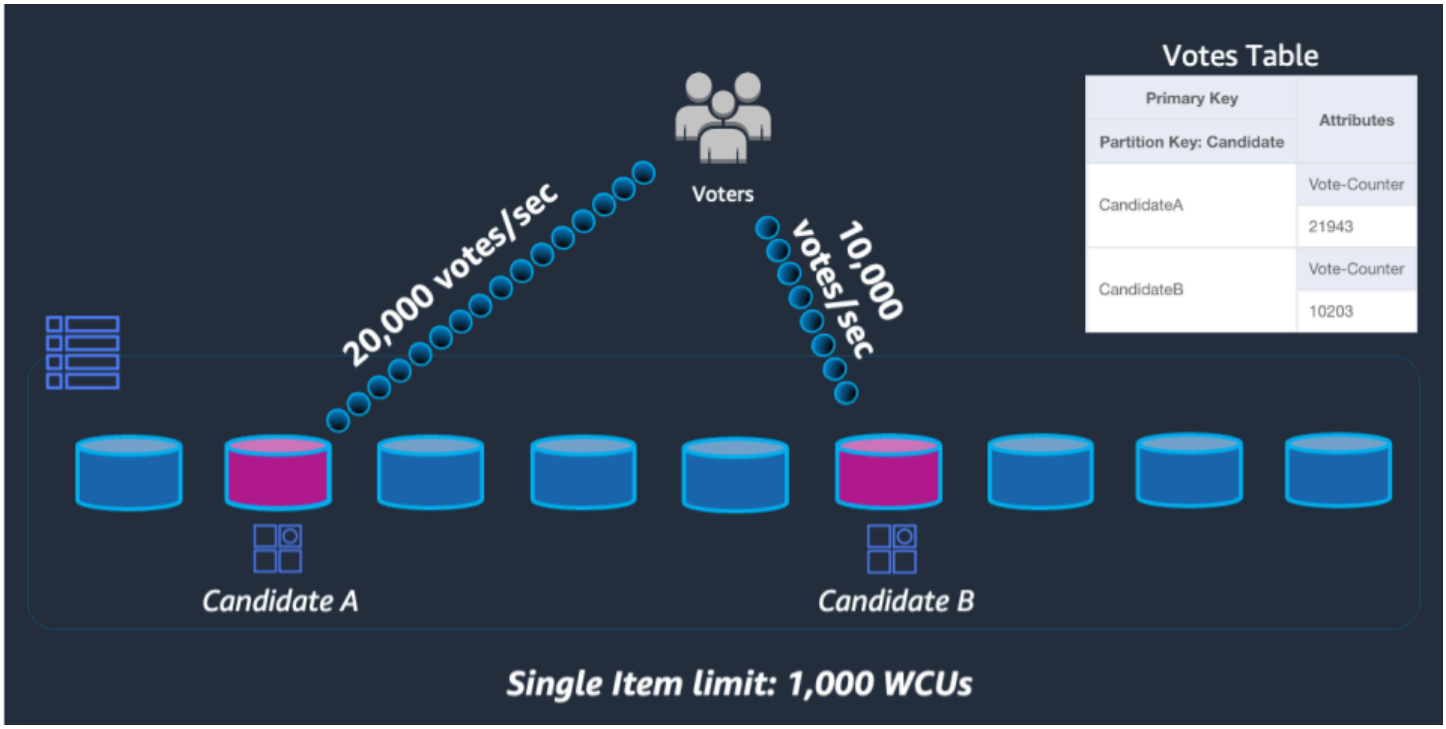
- 資料關係的階層是透過排序索引鍵字首維護的，因此如果需要，可以在用戶端重建單一文件結構
- 資料結構的單數元件可以獨立更新，導致小項目更新僅為 1 WCU
- 透過使用排序索引鍵 `BeginsWith`，應用程式可在單一查詢中擷取類似資料，彙總讀取成本以降低總成本/延遲
- 大型文件可輕易地超過 DynamoDB 中的 400 KB 個別項目大小限制，而垂直分割則有助於解決此限制

寫入碎片建置區塊

DynamoDB 的極少數硬性限制之一，就是單一實體分割區每秒可維持多少輸送量 (不一定是單一分割區索引鍵) 的限制。這些限制目前是：

- 1000 個 WCU (或每秒寫入 1000 <= 1KB 的項目) 和 3000 RCU (或每秒讀取 3000 <= 4KB) 強烈一致或
- 每秒 6000 <= 4 KB 讀取最終一致

如果對資料表的要求超過上述任一限制，則會將錯誤傳回給 `ThroughputExceededException` 的用戶端 SDK，通常稱為限流。需要超過該限制的讀取操作使用案例，通常是將讀取快取放在 DynamoDB 前面，以達到最佳效果，但是寫入操作需要稱為寫入碎片的結構描述層級設計。



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

為了解決此問題，我們將在應用程式的 `UpdateItem` 代碼中的每個競爭者的分割區索引鍵結尾附加一個隨機整數。隨機整數產生器的範圍，將需要具有上限相符或超過指定競爭者每秒的寫入量除以

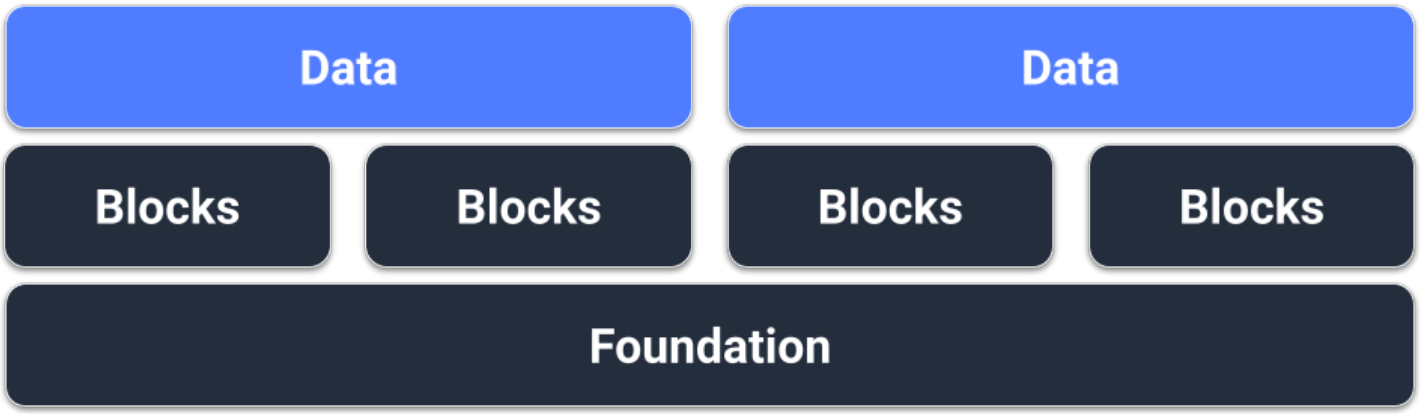
1000。為了支援每秒 20,000 次投票，它看起來會像 rand (0,19)。現在該資料儲存在單獨的邏輯分割區下，它必須在讀取時重新合併在一起。由於投票總數不需要是即時的，因此排定為每 X 分鐘讀取所有投票分割區的 Lambda 函式可能會偶爾為每位競爭者執行彙總，並將其寫回單一投票總記錄以供即時讀取。

此建置區塊的主要特徵

- 對於無法避免的指定分割區索引鍵具有極高寫入輸送量的使用案例，可以人為將寫入操作分散到多個 DynamoDB 分割區
- 具有低基數分割區索引鍵的 GSI 也應該使用此模式，因為 GSI 上的限流會套用背壓，以在基礎資料表上進行寫入操作

DynamoDB 中的資料建模結構描述設計套件

了解 DynamoDB 的資料建模結構描述設計套件，包括社群網路、遊戲設定檔、投訴管理、週期性付款、裝置狀態和線上商店的使用案例、存取模式和最終結構描述設計。



先決條件

在嘗試為 DynamoDB 設計結構描述之前，我們必須先收集有關結構描述需要支援的使用案例的一些先決條件資料。與關聯式資料庫不同，DynamoDB 預設會碎片化，這表示資料會在幕後存放在多部伺服器上，因此針對資料位置進行設計非常重要。我們需要為每個結構描述設計整理以下清單：

- 實體清單 (ER 圖)
- 每個實體的預估數量和輸送量
- 需要被支援的存取模式 (查詢和寫入)
- 資料保留要求

主題

- DynamoDB 中的社交網路結構描述設計
- DynamoDB 中的遊戲設定檔結構描述設計
- DynamoDB 中的投訴管理系統結構描述設計
- DynamoDB 中的週期性付款結構描述設計
- 監控 DynamoDB 中的裝置狀態更新
- 使用 DynamoDB 做為線上商店的資料存放區

DynamoDB 中的社交網路結構描述設計

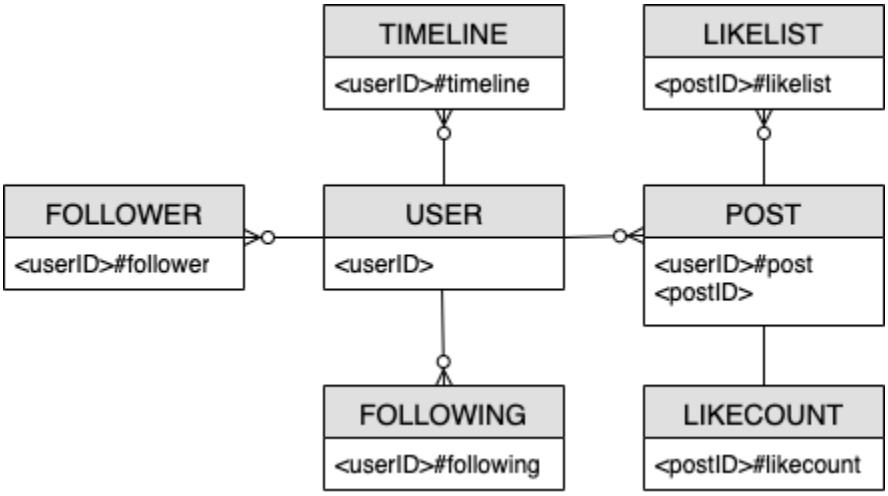
社交網路商業使用案例

此使用案例討論如何使用 DynamoDB 做為社交網路。社交網絡是一種線上服務，可以讓不同的使用者與彼此互動。我們將設計的社交網路會讓使用者看到一個時間軸，其中包括他們的文章、他們的跟隨者、他們跟隨的人，以及他們跟隨的人撰寫的文章。此結構描述設計的存取模式為：

- 取得指定 userID 的使用者資訊
- 取得指定 userID 的跟隨者清單
- 取得指定 userID 的跟隨清單
- 取得指定 userID 的文章清單
- 取得指定 postID 中喜歡該篇文章的使用者清單
- 取得指定 postID 的喜歡計數
- 取得指定 userID 的時間軸

社交網路實體關係圖

這是我們將用於社交網路結構描述設計的實體關係圖 (ERD)。



社交網路存取模式

這些是我們針對社交網路結構描述設計考量的存取模式。

- `getUserInfoByUserID`
- `getFollowerListByUserID`
- `getFollowingListByUserID`
- `getPostListByUserID`
- `getUserLikesByPostID`
- `getLikeCountByPostID`
- `getTimelineByUserID`

社交網路結構描述設計演變

DynamoDB 是一種 NoSQL 資料庫，因此它不允許您執行聯結，這是一種結合多個資料庫資料的操作。不熟悉 DynamoDB 的客戶可能會在不需要時，將關聯式資料庫管理系統 (RDBMS) 設計理念 (例如為每個實體建立資料表) 套用至 DynamoDB。DynamoDB 的單一資料表設計之目的，是根據應用程式的存取模式，以預先聯結的形式寫入資料，然後立即使用資料，無需額外運算。如需詳細資訊，請參閱 DynamoDB 中的單一資料表與多資料表設計。

現在，讓我們逐步介紹如何發展結構描述設計以解決所有存取模式。

步驟 1：位址存取模式 1 (**`getUserInfoByUserID`**)

為了取得特定使用者的資訊，我們需要 Query 基礎資料表，索引鍵條件為 PK=<userID>。查詢操作可讓您將結果分頁，這在使用者有許多跟隨者的情況下很實用。如需查詢的詳細資訊，請參閱 在 DynamoDB 中查詢資料表。

在我們的範例中，我們追蹤使用者的兩種類型資料：他們的「計數」和他們的「資訊」。使用者的「計數」反映他們有多少追隨者，他們追隨的使用者數量以及他們建立了多少文章。使用者的「資訊」反映了他們的個人資訊，例如他們的姓名。

我們看到這兩種資料由以下的兩個項目來表示。排序索引鍵 (SK) 中有「計數」的項目比含有「資訊」的項目更有可能變更。DynamoDB 會將項目的大小視為更新前後所顯示的大小，消耗的佈建輸送量將反映這些項目大小中較大的部分。因此即使您只更新一小部分項目的屬性，UpdateItem 還是會使用完整數量的佈建輸送量 (之前和之後項目大小中較大的一個)。您可以透過單一 Query 操作取得項目，並使用 UpdateItem 從現有的數字屬性中新增或減去。

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://....	...

步驟 2：位址存取模式 2 (**getFollowerListByUserID**)

若要取得跟隨指定使用者的使用者清單，我們需要 Query 基礎資料表，索引鍵條件為 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				

步驟 3：位址存取模式 3 (**getFollowingListByUserID**)

若要取得指定使用者跟隨的使用者清單，我們需要 Query 基礎資料表，索引鍵條件為 PK=<userID>#following。然後您可以使用 TransactWriteItems 操作將多項請求結合為一組，並執行以下操作：

- 將使用者 A 新增到使用者 B 的跟隨者清單中，然後將使用者 B 的跟隨者計數增加一。
- 將使用者 B 新增到使用者 A 的跟隨者清單中，然後將使用者 A 的跟隨者計數增加一。

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				

步驟 4：位址存取模式 4 (**`getPostListByUserID`**)

若要取得指定使用者建立的文章清單，我們需要 `Query` 基礎資料表，索引鍵條件為 PK=<userID>#post。這裡需要注意的一件重要事情是，使用者的 postID 必須是增量的：第二個 postID 值必須大於第一個 postID 值 (因為使用者希望以排序方式查看自己的貼文)。您可以透過根據時間值 (例如通用唯一字典順序排序識別符 (ULID)) 產生 postID 來實現此目的。

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	

步驟 5：位址存取模式 5 (**getUserLikesByPostID**)

若要取得對指定使用者貼文按下喜歡的使用者清單，我們需要 Query 基礎資料表，索引鍵條件為 PK=<postID>#likelist。這種方法是我們用於擷取跟隨者和跟隨清單中存取模式 2 (getFollowerListByUserID) 和存取模式 3 (getFollowingListByUserID) 相同的模式。

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				

步驟 6：位址存取模式 6 (**getLikeCountByPostID**)

要取得指定文章的喜歡計數，我們需要透過 PK=<postID>#likecount 的索引鍵條件在基礎資料表上執行 GetItem 操作。每當有許多跟隨者 (例如名人) 的使用者建立文章時，此存取模式就會造成限流問題，因為當分割區的輸送量超過每秒 1000 WCU 時，就會發生限流問題。這個問題不是 DynamoDB 的結果，它只是出現在 DynamoDB 中，因為它位於軟體堆疊的末端。

您應該評估它是否對所有使用者同時查看類似計數真的很重要，或者是否可以隨著時間的推移逐漸發生。一般來說，文章的類似計數不需要立即 100% 準確。您可以在應用程式和 DynamoDB 之間放置佇列，以便定期進行更新以實作此策略。

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			

步驟 7：位址存取模式 7 (**`getTimelineByUserID`**)

如要取得指定使用者的時間軸，我們需要透過 PK=<userID>#timeline 的索引鍵條件在基礎資料表上執行 Query 操作。讓我們考慮一個場景，即使用者的追隨者需要同步查看他們的文章。每次使用者寫一篇文章時，他們的跟隨者清單都會被讀取，並且他們的 userID 和 postID 會慢慢輸入到其所有跟隨者的時間軸索引鍵中。然後，當您的應用程式啟動時，您可以使用 Query 操作讀取時間軸索引鍵，並使用任何新項目的 BatchGetItem 操作來使用 userID 和 postID 的組合填充時間軸畫面。您無法使用 API 呼叫讀取時間軸，但是如果文章可以經常編輯，則這是一個更具成本效益的解決方案。

時間軸是一個顯示最近文章的地方，因此我們需要一種清理舊文章的方法。您可以使用 DynamoDB 的 TTL 功能免費執行此操作，而不是使用 WCU 刪除它們。

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			

下表摘要整理了所有存取模式，以及結構描述設計處理這些模式的方式：

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值	其他條件/篩選條件
getUserInfoByUserID	基礎資料表	Query	PK=<userID>		
getFollowerListByUserID	基礎資料表	Query	PK=<userID>#follower		

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值	其他條件/篩選條件
getFollowingListByUserID	基礎資料表	Query	PK=<userID>#following		
getPostListByUserID	基礎資料表	Query	PK=<userID>#post		
getUserLikesByPostID	基礎資料表	Query	PK=<postID>#likelist		
getLikeCountByPostID	基礎資料表	GetItem	PK=<postID>#likecount		
getTimelineByUserID	基礎資料表	Query	PK=<userID>#timeline		

社交網路最終結構描述

這是最終的結構描述設計。若要將此結構描述設計下載為 JSON 檔案，請參閱 GitHub 上的 DynamoDB 範例。

基礎資料表：

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			

使用 NoSQL Workbench 與此結構描述設計

您可以將此最終結構描述匯入 NoSQL Workbench，這是為 DynamoDB 提供資料建模、資料視覺化，和查詢開發功能的視覺化工具，以進一步探索和編輯新專案。請依照下列步驟以開始使用：

1. 下載 NoSQL Workbench。如需詳細資訊，請參閱the section called “下載”。
2. 下載上面列出的 JSON 結構描述檔案，該檔案已經是 NoSQL Workbench 模型格式。
3. 將 JSON 結構描述檔案匯入到 NoSQL Workbench。如需詳細資訊，請參閱the section called “匯入現有的模型”。
4. 一旦您匯入到 NOSQL Workbench 後，便可以編輯資料模型。如需詳細資訊，請參閱the section called “編輯現有的模型”。

DynamoDB 中的遊戲設定檔結構描述設計

遊戲設定檔商業使用案例

本使用案例討論如何使用 DynamoDB 儲存遊戲系統的玩家設定檔。使用者 (在本案例中為玩家) 需要先建立設定檔，然後才能與許多現代遊戲 (尤其是線上遊戲) 進行互動。遊戲設定檔通常包括下列項目：

- 基本資訊，例如使用者名稱
- 遊戲資料，例如物品和裝備
- 遊戲記錄，例如任務和活動
- 朋友清單等社交資訊

為了滿足此應用程式的精細資料查詢存取要求，主索引鍵 (分割區索引鍵和排序索引鍵) 將使用通用名稱 (PK 和 SK)，因此它們可以用各種類型的值進行過載，如下所示。

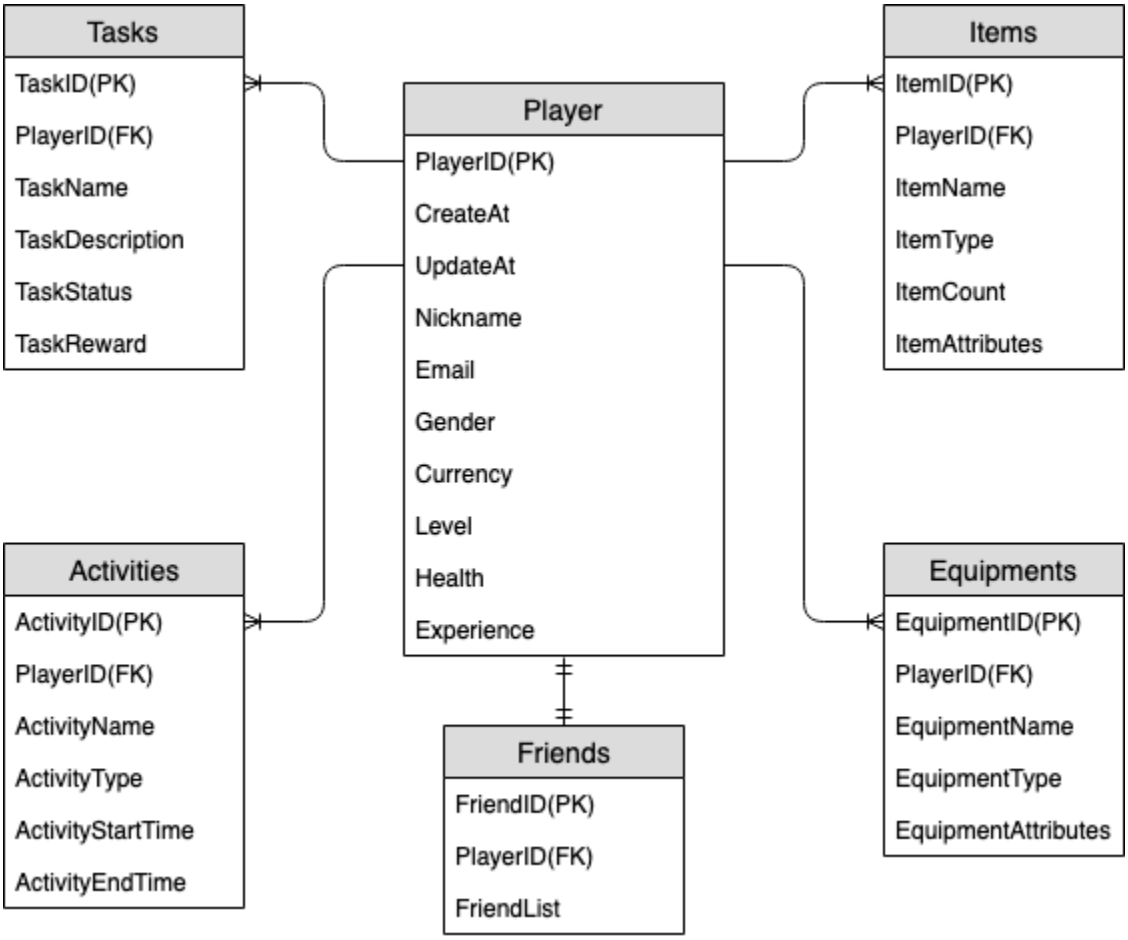
此結構描述設計的存取模式為：

- 取得使用者的好友清單
- 取得玩家的所有資訊
- 取得使用者的物品清單
- 從使用者的物品清單中取得特定項目
- 更新使用者的角色
- 更新使用者的物品計數

遊戲設定檔的大小在不同的遊戲中會有所不同。壓縮大型屬性值可以讓它們符合 DynamoDB 中的項目限制並減少成本。輸送量管理策略取決於各種因素，例如：玩家數量，每秒遊戲遊玩數量，以及工作負載的季節性。通常對於新推出的遊戲而言，玩家數量和受歡迎程度未知，因此我們將從隨需輸送量模式開始。

遊戲設定檔實體關係圖

這是我們將用於遊戲設定檔架構設計的實體關係圖 (ERD)。



遊戲設定檔存取模式

這些是我們針對社交網路結構描述設計考量的存取模式。

- `getPlayerFriends`
- `getPlayerAllProfile`
- `getPlayerAllItems`
- `getPlayerSpecificItem`
- `updateCharacterAttributes`
- `updateItemCount`

遊戲設定檔結構描述設計演進

從上面的 ERD 我們可以看到，這是一個一對多關係類型的資料建模。在 DynamoDB 中，一對多資料模型可以組織成項目集合，這與傳統建立多資料表並透過外部索引鍵建立連結的關聯式資料庫不同。項

目集合是一組項目，共用相同的分割區索引鍵值，但具有不同的排序索引鍵值。在項目集合中，每個項目都有唯一的排序索引鍵值，可將其與其他項目區分開來。考量這一點，讓我們對每種實體類型的 HASH 和 RANGE 值使用以下模式。

首先，我們使用通用名稱 (如 PK 和 SK) 將不同類型的實體儲存在同一個資料表中，以打造前瞻性的模型。為了更好的可讀性，我們可以包括字首來表示資料的類型或包含名為 Entity_type 或 Type 的任意屬性。在目前範例中，我們使用以 player 開頭的字串將 player_ID 儲存為 PK；使用 entity name# 作為 SK 的字首，並新增一個 Type 屬性來指示這筆資料是哪種實體類型。這使我們能夠支援將來儲存更多實體類型，並使用例如 GSI 超載和稀疏 GSI 等先進技術來滿足更多存取模式。

讓我們開始實作存取模式。新增玩家、新增裝備等存取模式，都可以透過 PutItem 操作來實現，所以我們可以忽略它們。在本文件中，我們將著重於上面列出的典型存取模式。

步驟 1：位址存取模式 1 (**getPlayerFriends**)

我們會透過此步驟解決存取模式 1 (getPlayerFriends)。在我們目前的設計中，朋友關係很簡單，遊戲中的朋友數量也很少。為簡單起見，我們使用清單資料類型來儲存朋友清單 (1:1 模型建置)。在這種設計中，我們使用 GetItem 來滿足這種存取模式。在 GetItem 操作中，我們明確提供了分割區索引鍵和排序索引鍵值以取得特定項目。

但是，如果一個遊戲有大量的朋友，並且他們之間的關係很複雜 (例如朋友關係是雙向的，具有邀請和接受元件)，則必須使用多對多關係來單獨儲存每個朋友，以便擴展到無限的朋友清單大小。而且，如果朋友關係變更涉及同時對多個項目進行操作，則 DynamoDB 交易可用於將多個動作分組在一起，並將它們提交為單一「全有或全無」TransactWriteItems 或 TransactGetItems 操作。

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"}}}]

步驟 2：位址存取模式 2 (**getPlayerAllProfile**)、3 (**getPlayerAllItems**) 和 4 (**getPlayerSpecificItem**)

我們使用此步驟解決存取模式 2 (getPlayerAllProfile)、3 (getPlayerAllItems) 和 4 (getPlayerSpecificItem)。這三種存取模式的共同點是使用 Query 操作的範圍查詢。根據查詢的範圍，會使用索引鍵條件和篩選條件表達式，這些表達式通常在實際開發中使用。

在查詢操作中，我們為分割區索引鍵提供單一值，並取得具有該分割區索引鍵值的所有項目。存取模式 2 (getPlayerAllProfile) 是以這種方式實現。或者，我們可以新增一個排序索引鍵條件表達式 - 確定要從資料表中讀取項目的字串。存取模式 3 (getPlayerAllItems) 是透過新增排序索引鍵 begins_with ITEMS# 的索引鍵條件來實現。此外，為了簡化應用程式端的開發，我們可以使用篩選條件表達式來實現存取模式 4 (getPlayerSpecificItem)。

以下是使用篩選條件表達式來篩選 Weapon 類別項目的虛擬程式碼範例：

```
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**
>
> 篩選條件表達式會在查詢完成之後，結果傳回給用戶端之前進行套用。因此，無論是否有篩選條件表達式，查詢都會使用相同數量的讀取容量。

如果存取模式是查詢大型資料集並篩選出大量資料以僅保留一小部分資料，則適當的方法是更有效地設計 DynamoDB 分割區索引鍵和排序索引鍵。例如，在上面的範例中取得特定 ItemType，如果每個玩家都有很多物品並且查詢特定 ItemType 是典型的存取模式，將 ItemType 作為複合索引鍵帶入 SK 會更有效。資料模型看起來會像這樣：ITEMS#ItemType#ItemId。

步驟 3：位址存取模式 5 (**updateCharacterAttributes**)、和 6 (**updateItemCount**)

我們使用此步驟解決存取模式 5 (updateCharacterAttributes) 和 6 (updateItemCount)。當玩家需要修改角色時，例如減少貨幣，或者修改物品中某種武器數量時，可使用 UpdateItem 以實現這

些存取模式。如要更新玩家的貨幣，但確保它永遠不會低於最低金額，僅在餘額大於或等於最小金額時，我們才可以新增 the section called “CLI 範例” 來減少餘額。此為一個虛擬程式碼範例：

```
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

使用 DynamoDB 進行開發並使用原子計數器減少庫存時，我們可以透過使用樂觀鎖定來確保冪等性。以下是原子計數器的虛擬程式碼範例：

```
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"}}}

此外，在玩家用貨幣購買物品的情況下，整個過程需要扣除貨幣並同時新增一個物品。我們可以使用 DynamoDB 交易將多個動作分組在一起，並將它們作為單一「全有或全無」`TransactWriteItems` 或 `TransactGetItems` 操作來提交。`TransactWriteItems` 是一種同步且冪等性的寫入操作，在單一「全有或全無」操作中分成多達 100 個群組的寫入操作。這些動作的完成具有不可分割性，也就是全部成功或全部失敗。交易有助於消除貨幣重複或消失的風險。如需交易的詳細資訊，請參閱 DynamoDB 交易範例。

下表摘要整理了所有存取模式，以及結構描述設計處理這些模式的方式：

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值	其他條件/篩選條件
getPlayerFriends	基礎資料表	GetItem	PK=PlayerID	SK=“FRIENDS#playerID”	

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值	其他條件/篩選條件
getPlayerAllProfile	基礎資料表	Query	PK=PlayerID		
getPlayerAllItems	基礎資料表	Query	PK=PlayerID	SK begins_with “ITEMS#”	
getPlayerSpecificItem	基礎資料表	Query	PK=PlayerID	SK begins_with “ITEMS#”	filterExpression: "ItemType = :itemType" expressionAttributeValues: { ":itemType": "Weapon" }
updateCharacterAttributes	基礎資料表	UpdateItem	PK=PlayerID	SK=“#METADATA#playerID”	UpdateExpression: "SET currency = currency - :amount" ConditionExpression: "currency >= :minAmount"

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值	其他條件/篩選條件
updateItemCount	基礎資料表	UpdateItem	PK=PlayerID	SK =“ITEMS#ItemID”	update-expression: "SET ItemCount = ItemCount - :incr" expression-attribute-values : '{":incr": {"N":"1"}}'

遊戲設定檔最終結構描述

這是最終的結構描述設計。若要將此結構描述設計下載為 JSON 檔案，請參閱 GitHub 上的 DynamoDB 範例。

基礎資料表：

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"}}}						

使用 NoSQL Workbench 與此結構描述設計

您可以將此最終結構描述匯入 NoSQL Workbench，這是為 DynamoDB 提供資料建模、資料視覺化，和查詢開發功能的視覺化工具，以進一步探索和編輯新專案。請依照下列步驟以開始使用：

1. 下載 NoSQL Workbench。如需詳細資訊，請參閱the section called “下載”。
2. 下載上面列出的 JSON 結構描述檔案，該檔案已經是 NoSQL Workbench 模型格式。
3. 將 JSON 結構描述檔案匯入到 NoSQL Workbench。如需詳細資訊，請參閱the section called “匯入現有的模型”。
4. 一旦您匯入到 NOSQL Workbench 後，便可以編輯資料模型。如需詳細資訊，請參閱the section called “編輯現有的模型”。

DynamoDB 中的投訴管理系統結構描述設計

投訴管理系統商業使用案例

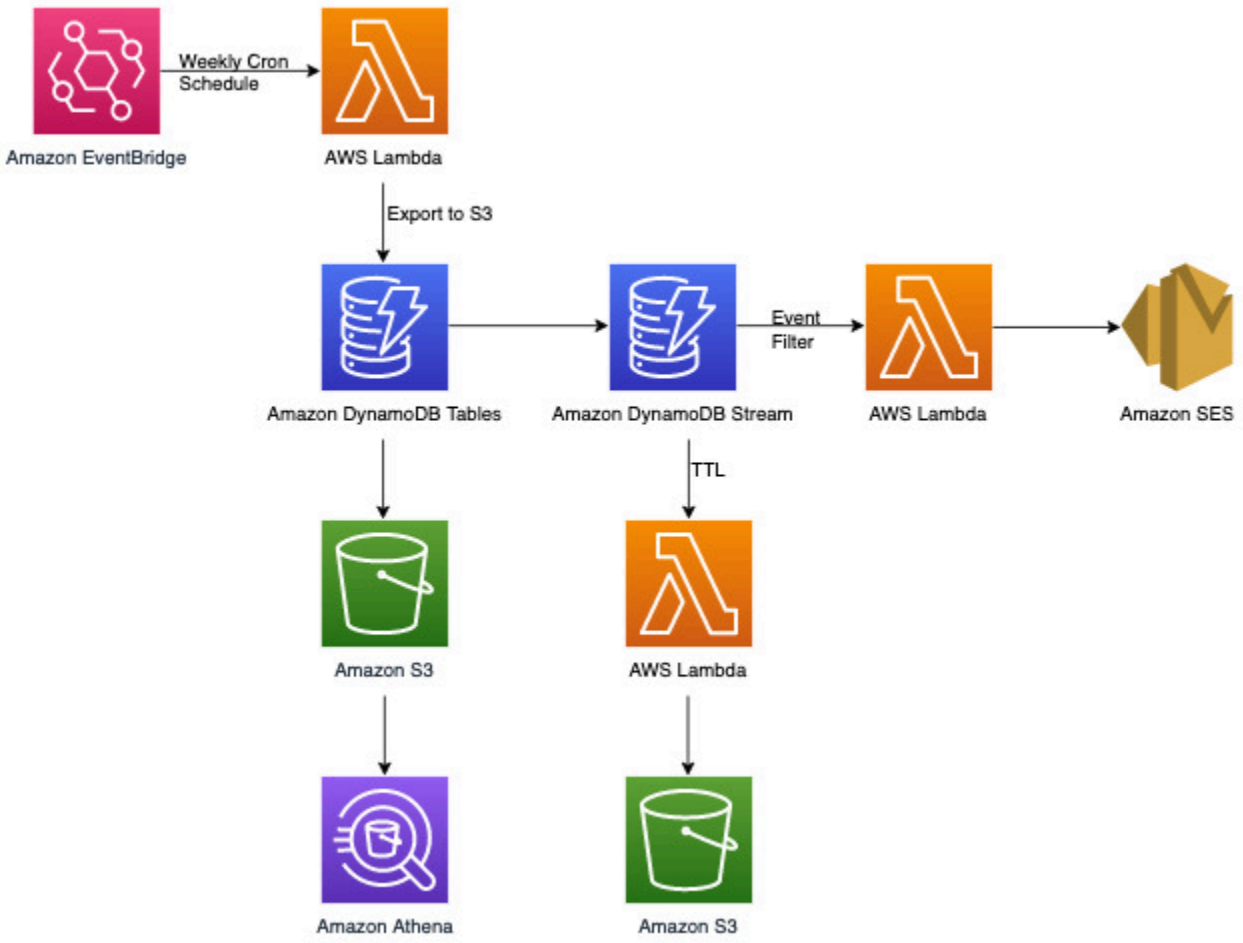
DynamoDB 是非常適合投訴管理系統 (或聯絡中心) 使用案例的資料庫，因為與其相關聯的大多數存取模式都是採用鍵值型交易查詢。這種情況的典型存取模式會是：

- 建立和更新投訴
- 呈報投訴
- 建立和讀取投訴的評論
- 取得某一位客戶的所有投訴
- 取得某一位客服人員的所有評論並取得所有呈報

某些評論可能包含描述投訴或解決方案的附件。雖然這些全都是鍵值存取模式，但可能會有其他需求，例如，有新評論新增至投訴時傳送通知，或是執行分析查詢，以便了解每週在嚴重性 (或客服人員表現) 方面的投訴分佈情形。與生命週期管理或合規有關的其他需求，可能包括從記錄投訴起算經過三年後封存投訴資料。

投訴管理系統架構圖

下圖顯示投訴管理系統的架構圖。此圖表顯示投訴管理系統使用的不同 AWS 服務 整合。



除了稍後將在「DynamoDB 資料建模」一節中討論的鍵值交易存取模式之外，還有三項非交易需求。上面的架構圖可分成以下三個工作流程：

1. 有新評論新增至投訴時傳送通知
2. 對每週資料執行分析查詢
3. 封存超過三年的資料

讓我們進一步深入探討上述每一項。

有新評論新增至投訴時傳送通知

我們可以使用下面的工作流程來達成此需求：

DynamoDB Streams 是一種變更資料擷取機制，用來記錄 DynamoDB 資料表上的所有寫入活動。您可以設定 Lambda 函式來觸發部分或全部變更。您可以在 Lambda 觸發程序上設定事件篩選條件，以篩選掉與使用案例無關的事件。在此範例中，我們只能在有新評論新增時使用篩選條件來觸發 Lambda，並傳送通知至可從 AWS Secrets Manager 或任何其他憑證存放區提取的相關電子郵件 ID。

對每週資料執行分析查詢

DynamoDB 適合用於主要進行線上交易處理 (OLTP) 的工作負載。對於具有分析需求的其他 10-20% 存取模式，可以使用受管的匯出至 Amazon S3 功能將資料匯出到 S3，這樣做不會影響 DynamoDB 資料表上的即時流量。讓我們來了解下面這個工作流程：

Amazon EventBridge 可用於 AWS Lambda 按排程觸發 - 它可讓您設定 cron 表達式，以便 Lambda 調用定期進行。Lambda 可以調用 `ExportToS3` API 呼叫並將 DynamoDB 資料儲存在 S3 中。隨後 SQL 引擎 (如 Amazon Athena) 就能存取此 S3 資料，以對 DynamoDB 資料執行分析查詢，而不會影響資料表上的即時交易工作負載。依嚴重性層級尋找投訴數目的 Athena 範例查詢如下所示：

```
SELECT Item.severity.S as "Severity", COUNT(Item) as "Count"
FROM "complaint_management"."data"
WHERE NOT Item.severity.S = ''
GROUP BY Item.severity.S ;
```

產生的 Athena 查詢結果如下：

Results (3)

Search rows

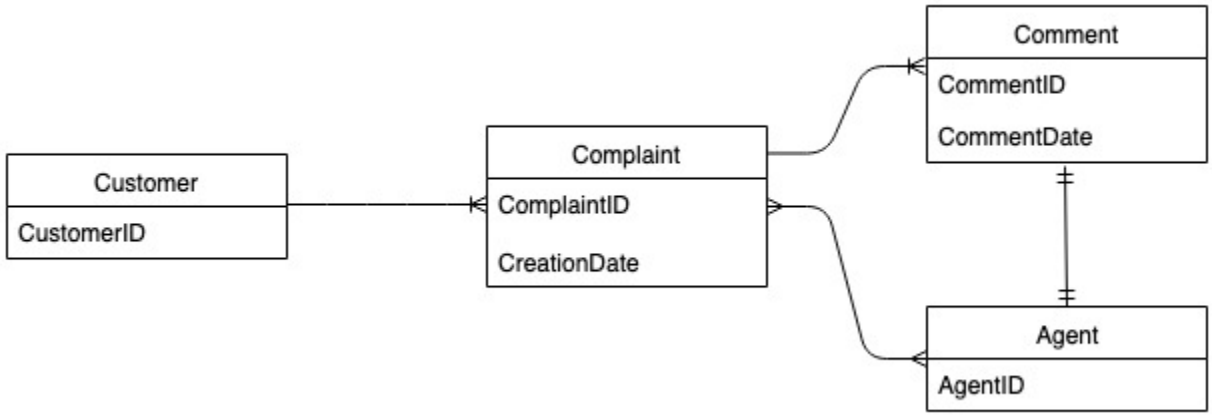
#	Severity	Count
1	P3	1
2	P2	2
3	P1	1

封存超過三年的資料

您可以利用 DynamoDB 存留時間 (TTL) 功能刪除 DynamoDB 資料表中過時的資料，無需額外費用 (但 2019.11.21 (目前) 版的全域資料表複本除外，其中複寫到其他區域的 TTL 刪除項目會耗用寫入容量)。此資料隨即出現，且可從 DynamoDB Streams 取用並封存至 Amazon S3。此需求的工作流程如下：

投訴管理系統實體關係圖

這是我們將用於投訴管理系統結構描述設計的實體關係圖 (ERD)。



投訴管理系統存取模式

這些是我們針對投訴管理結構描述設計考量的存取模式。

1. createComplaint
2. updateComplaint
3. updateSeveritybyComplaintID
4. getComplaintByComplaintID
5. addCommentByComplaintID
6. getAllCommentsByComplaintID
7. getLatestCommentByComplaintID
8. getAComplaintbyCustomerIDAndComplaintID
9. getAllComplaintsByCustomerID
10. escalateComplaintByComplaintID
11. getAllEscalatedComplaints
12. getEscalatedComplaintsByAgentID (從最新到最舊的順序)
13. getCommentsByAgentID (兩個日期之間)

投訴管理系統結構描述設計演變

由於這是投訴管理系統，因此大多數存取模式都以投訴作為主要實體。高基數的 `ComplaintID` 將可確保資料在基礎分割區中均勻分佈，同時也是我們確定的存取模式最常見的搜尋條件。因此，`ComplaintID` 是此資料集中良好的分割區索引鍵選擇。

步驟 1：位址存取模式 1 (**`createComplaint`**)、2 (**`updateComplaint`**)、3 (**`updateSeveritybyComplaintID`**) 和 4 (**`getComplaintByComplaintID`**)

我們可以使用稱為「中繼資料」(或「AA」) 的通用排序索引鍵來儲存投訴專屬資訊，例如 `CustomerID`、`State`、`Severity` 和 `CreationDate`。我們使用單一操作搭配 `PK=ComplaintID` 和 `SK="metadata"` 來執行下列操作：

1. `PutItem` 用來建立新的投訴
2. `UpdateItem` 用來更新投訴中繼資料中的嚴重性或其他欄位
3. `GetItem` 用來提取投訴的中繼資料

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>

步驟 2：位址存取模式 5 (**addCommentByComplaintID**)

此存取模式須在投訴與投訴的評論之間採用一對多關係模型。我們將在這裡採用垂直分割技術來使用排序索引鍵，並建立具有不同類型資料的項目集合。只要查看存取模式 6 (getAllCommentsByComplaintID) 和 7 (getLatestCommentByComplaintID)，就會了解評論需依時間排序。我們也可以同時接收多個評論，如此就能使用複合排序索引鍵技術將時間和 CommentID 附加到排序索引鍵屬性中。

其他處理這類可能發生的評論衝突的選項，包括增加時間戳記的精細度，或是加上一個累加數字作為尾碼，而不要使用 Comment_ID。在這種情況下，我們將在對應評論之項目的排序索引鍵值加上首碼「comm#」，以實現範圍型操作。

另外還需要確保投訴中繼資料中的 currentState 能反映新增新評論的狀態。新增評論可能表示，投訴已指派給客服人員或已解決等情況。為了在投訴中繼資料中綁定新增評論和更新目前狀態，我們將採用全有或全無的方式使用 TransactWriteItems API。產生的資料表狀態現在看起來像這樣：

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>

讓我們在資料表中新增更多資料，另外也新增 ComplaintID 作為有別於 PK 的另一個欄位，以便在 ComplaintID 需要其他索引時，讓模型能夠因應未來需要。另請注意，某些評論可能包含附件，我們會將這些附件儲存在 Amazon Simple Storage Service 中，並且只會在 DynamoDB 中維護其參考或 URL。最佳實務是盡可能保持交易資料庫精簡，以最佳化成本和效能。資料現在看起來像這樣：

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>

步驟 3：位址存取模式 6 (**getAllCommentsByComplaintID**) 和 7 (**getLatestCommentByComplaintID**)

若要取得投訴的所有評論，我們可以在排序索引鍵上使用 query 操作搭配 begins_with 條件。與其耗用額外的讀取容量來讀取中繼資料項目，隨後因篩選相關結果而造成額外負荷，倒不如使用項這樣的排序索引鍵條件幫助我們只讀取所要的內容。例如使用 PK=Complaint123 及 SK begins_with comm# 進行查詢操作，會傳回下列內容並略過中繼資料項目：

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>

既然我們在模式 7 (`getLatestCommentByComplaintID`) 中需要投訴的最新評論，那麼就使用兩個額外的查詢參數：

1. `ScanIndexForward` 應設定為 False，才能以遞減順序排序結果
2. `Limit` 應設定為 1，才能取得最新 (單獨一個) 評論

類似存取模式 6 (`getAllCommentsByComplaintID`)，我們使用 `begins_with comm#` 作為排序索引鍵條件以略過中繼資料項目。現在，您可以使用查詢操作搭配 `PK=Complaint123` 和 `SK=begins_with comm#`、`ScanIndexForward=False`、`Limit 1`，對此設計執行存取模式 7。結果會傳回以下目標項目：

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>

讓我們新增更多虛設資料到資料表中。

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>

步驟 4：位址存取模式 8 (**getAComplaintbyCustomerIDAndComplaintID**) 和 9 (**getAllComplaintsByCustomerID**)

存取模式 8 (getAComplaintbyCustomerIDAndComplaintID) 和 9 (getAllComplaintsByCustomerID) 導入了新的搜尋條件：CustomerID。若要從現有資料表提取它，則須使用昂貴的 Scan 來讀取所有資料，然後篩選相關項目以找出所指的 CustomerID。我們只要建立全域次要索引 (GSI) 並搭配 CustomerID 作為分割區索

引鍵，就能讓這項搜尋更有效率。務必記住客戶與投訴之間的一對多關係以及存取模式 9 (getAllComplaintsByCustomerID)，ComplaintID 會是排序索引鍵的合適選擇。

GSI 中的資料如下所示：

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>

此 GSI 上存取模式 8 (getAComplaintbyCustomerIDAndComplaintID) 的範例查詢會是：customer_id=custXYZ、sort key=Complaint1321。結果會是：

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>

若要針對存取模式 9 (getAllComplaintsByCustomerID) 取得某一個客戶的所有投訴，GSI 上的查詢會是：customer_id=custXYZ 作為分割區索引鍵條件。結果會是：

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>

步驟 5：位址存取模式 10 (**`escalateComplaintByComplaintID`**)

此存取將介紹呈報相關層面。若要呈報投訴，我們可以使用 `UpdateItem` 將 `escalated_to` 和 `escalation_time` 等屬性新增至現有投訴中繼資料項目中。DynamoDB 提供了靈活的結構描述設計，這表示非索引鍵屬性集可以合併在一起，也可以分散到不同的項目中。如需範例，請參閱下列內容：

`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

步驟 6：位址存取模式 11 (**`getAllEscalatedComplaints`**) 和 12 (**`getEscalatedComplaintsByAgentID`**)

預計在整個資料集中，只有少數投訴會呈報。因此，在呈報相關屬性上建立索引將能實現高效率的查詢與符合成本效益的 GSI 儲存。我們可以利用稀疏索引技術來實現這個目標。具有分割區索引鍵 `escalated_to` 及排序索引鍵 `escalation_time` 的 GSI 如下所示：

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>

若要針對存取模式 11 (`getAllEscalatedComplaints`) 取得所有呈報的投訴，只要掃描此 GSI 即可。請注意，由於 GSI 的大小，此掃描將會是高效能且具有成本效益。若要取得特定客服人員的 (存取模式 12 (`getEscalatedComplaintsByAgentID`)) 的已呈報投訴，分割區索引鍵會是 `escalated_to=agentID`，且我們會將 `ScanIndexForward` 設定為 `False`，以便從最新到最舊排序。

步驟 7：位址存取模式 13 (**`getCommentsByAgentID`**)

對於最後一個存取模式，我們需要以新的維度執行查詢：`AgentID`。另外還需要依時間進行排序來讀取兩個日期之間的評論，因此我們會建立一個 GSI，並以 `agent_id` 作為分割區索引鍵及 `comm_date` 作為排序索引鍵。此 GSI 中的資料如下所示：

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"]

此 GSI 的範例查詢會是 `partition key agentID=AgentA` 和 `sort key=comm_date between (2023-04-30T12:30:00, 2023-05-01T09:00:00)`，其結果為：

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"]

下表摘要整理了所有存取模式，以及結構描述設計處理這些模式的方式：

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值	其他條件/篩選條件
createComplaint	基礎資料表	PutItem	PK=complaint_id	SK=metadata	
updateComplaint	基礎資料表	UpdateItem	PK=complaint_id	SK=metadata	
updateSeveritybyComplaintID	基礎資料表	UpdateItem	PK=complaint_id	SK=metadata	
getComplaintByComplaintID	基礎資料表	GetItem	PK=complaint_id	SK=metadata	
addCommentByComplaintID	基礎資料表	TransactWriteItems	PK=complaint_id	SK=metadata, SK=comm#comm_date#comm_id	

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值	其他條件/篩選條件
getAllCommentsByComplaintID	基礎資料表	Query	PK=complaint_id	SK begins_with "comm#"	
getLatestCommentByComplaintID	基礎資料表	Query	PK=complaint_id	SK begins_with "comm#"	scan_index_forward=False, Limit 1
getAComplaintbyCustomerIDAndComplaintID	Customer_complaint_GSI	Query	customer_id=customer_id	complaint_id = complaint_id	
getAllComplaintsByCustomerID	Customer_complaint_GSI	Query	customer_id=customer_id	N/A	
escalateComplaintByComplaintID	基礎資料表	UpdateItem	PK=complaint_id	SK=metadata	
getAllEscalatedComplaints	Escalations_GSI	Scan	N/A	N/A	
getEscalatedComplaintsByAgentID (從最新到最舊的順序)	Escalations_GSI	Query	escalated_to=agent_id	N/A	scan_index_forward=False

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值	其他條件/篩選條件
getCommentsByAgentID(兩個日期之間)	Agents_Comments_GSI	Query	agent_id= agent_id	SK between (date1, date2)	

投訴管理系統最終結構描述

以下是最終結構描述設計。若要將此結構描述設計下載為 JSON 檔案，請參閱 GitHub 上的 DynamoDB 範例。

基礎資料表

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"]

使用 NoSQL Workbench 與此結構描述設計

您可以將此最終結構描述匯入 NoSQL Workbench，這是為 DynamoDB 提供資料建模、資料視覺化，和查詢開發功能的視覺化工具，以進一步探索和編輯新專案。請依照下列步驟以開始使用：

1. 下載 NoSQL Workbench。如需詳細資訊，請參閱the section called “下載”。

2. 下載上面列出的 JSON 結構描述檔案，該檔案已經是 NoSQL Workbench 模型格式。
3. 將 JSON 結構描述檔案匯入到 NoSQL Workbench。如需詳細資訊，請參閱the section called “匯入現有的模型”。
4. 一旦您匯入到 NOSQL Workbench 後，便可以編輯資料模型。如需詳細資訊，請參閱the section called “編輯現有的模型”。

DynamoDB 中的週期性付款結構描述設計

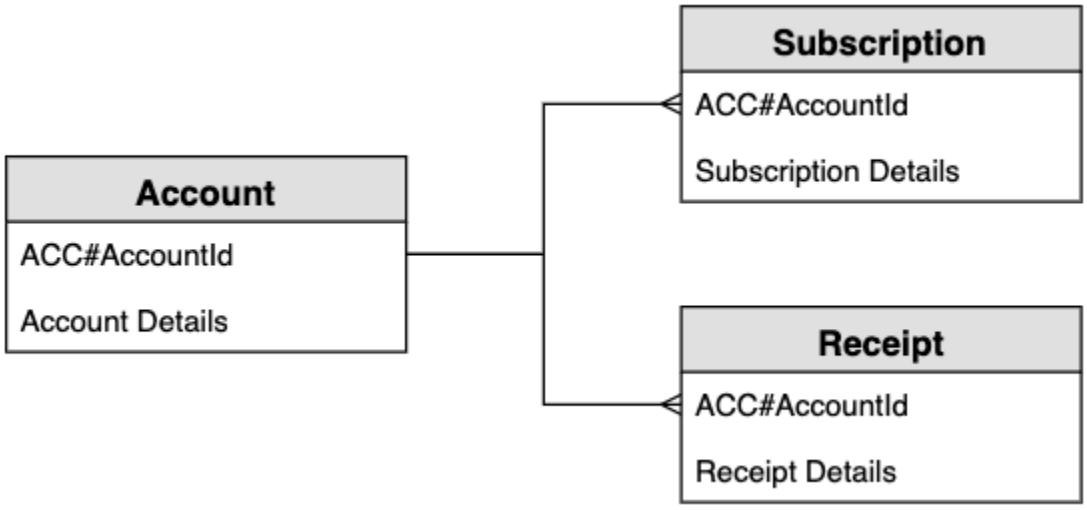
週期性付款商業使用案例

此使用案例將說明如何使用 DynamoDB 實作週期性付款系統。資料模型具有以下實體：帳戶、訂閱及收據。我們的使用案例包括以下細節：

- 每個帳戶可以有多個訂閱
- 需要處理下一次付款時，訂閱會有 `NextPaymentDate`，傳送電子郵件提醒給客戶時，則會有 `NextReminderDate`
- 有一個代表訂閱的項目會儲存並在付款處理完成時更新 (平均項目大小約為 1KB，輸送量取決於帳戶和訂閱的數目)
- 付款處理器還會在過程中建立一張收據並儲存在資料表中，而且會使用 TTL 屬性將它設定為在經過一段時間後過期。

週期性付款實體關係圖

這是我們將用於週期性付款系統結構描述設計的實體關係圖 (ERD)。



週期性付款系統存取模式

這些是我們針對週期性付款系統結構描述設計考量的存取模式。

1. createSubscription
2. createReceipt
3. updateSubscription
4. getDueRemindersByDate
5. getDuePaymentsByDate
6. getSubscriptionsByAccount
7. getReceiptsByAccount

週期性付款結構描述設計

通用名稱 PK 和 SK 用於金鑰屬性，以允許在相同資料表中儲存不同類型的實體，例如帳戶、訂閱和收據實體。使用者首先建立訂閱，這代表使用者同意在每個月的同一天支付一定金額來換取使用產品。使用者可以選擇要在每個月的哪一天處理付款。另外還會在處理付款之前傳送提醒。應用程式的運作方式是每天執行兩個批次任務：一個批次任務傳送當天應送出的提醒，另一個批次任務則處理當天應付的任何款項。

步驟 1：位址存取模式 1 (**createSubscription**)

存取模式 1 (createSubscription) 用於初次建立訂閱，並設定包括 SKU、NextPaymentDate、NextReminderDate 和 PaymentDetails 等詳細資訊。此步驟僅顯示具有一個訂閱的一個帳戶的資料表狀態。項目集合中可以有多個訂閱，因此這是一對多關係。

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

步驟 2：位址存取模式 2 (**createReceipt**) 和 3 (**updateSubscription**)

存取模式 2 (createReceipt) 用於建立收據項目。每個月處理付款後，付款處理器會將收據寫回基礎資料表。項目集合中可以有多個收據，因此這是一對多關係。付款處理器也會更新訂閱項目 (存取模式 3 (updateSubscription)) 以更新下個月的 NextReminderDate 或 NextPaymentDate。

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

步驟 3：位址存取模式 4 (**getDueRemindersByDate**)

應用程式會分批處理當天的付款提醒。因此，應用程式需要存取不同維度的訂閱：日期，而非帳戶。這是一個很好的全域次要索引 (GSI) 使用案例。在此步驟中，我們會新增索引 GSI-1，它會使用 NextReminderDate 作為 GSI 分割區索引鍵。我們不需要複寫所有項目。這個 GSI 是稀疏索引，因此不會複寫收據項目。我們也不需要對應所有的屬性，只需要包括屬性的子集。下方影像顯示了 GSI-1 的結構描述，它提供了應用程式傳送提醒電子郵件所需的資訊。

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

步驟 4：位址存取模式 5 (**getDuePaymentsByDate**)

應用程式會分批處理當天的付款，採取與處理提醒相同的方式。我們會在此步驟中新增 GSI-2，它會使用 NextPaymentDate 作為 GSI 分割區索引鍵。我們不需要複寫所有項目。這個 GSI 是稀疏索引，因為不會複寫收據項目。下方影像顯示了 GSI-2 的結構描述。

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"}}

步驟 5：位址存取模式 6 (**getSubscriptionsByAccount**) 和 7 (**getReceiptsByAccount**)

應用程式會在基礎資料表上使用查詢來鎖定帳戶識別碼 (PK) 為目標，以擷取帳戶的所有訂閱，並使用範圍運算子取得 SK 開頭為「SUB#」的所有項目。應用程式也可以使用相同的查詢結構來擷取所有收據，方法是使用範圍運算子來取得 SK 開頭為「REC#」的所有項目。這樣就能滿足存取模式 6 (getSubscriptionsByAccount) 和 7 (getReceiptsByAccount)。應用程式使用這些存取模式讓使用者看見過去六個月內自己目前的訂閱及過去的收據。在此步驟中，我們不會變更資料表結構描述，而且可以在下面看到我們如何單獨鎖定存取模式 6 (getSubscriptionsByAccount) 中的訂閱項目。

Primary key		Attributes									
Partition key: PK	Sort key: SK										
	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
ACC#123	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

下表摘要整理了所有存取模式，以及結構描述設計處理這些模式的方式：

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值
createSubscription	基礎資料表	PutItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
createReceipt	基礎資料表	PutItem	ACC#account_id	REC#<ReceiptDate>#SKU<SKUID>
updateSubscription	基礎資料表	UpdateItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
getDueRemindersByDate	GSI-1	Query	<NextReminderDate>	
getDuePaymentsByDate	GSI-2	Query	<NextPaymentDate>	
getSubscriptionsByAccount	基礎資料表	Query	ACC#account_id	SK begins_with “SUB#”
getReceiptsByAccount	基礎資料表	Query	ACC#account_id	SK begins_with “REC#”

週期性付款最終結構描述

以下是最終結構描述設計。若要將此結構描述設計下載為 JSON 檔案，請參閱 GitHub 上的 DynamoDB 範例。

基礎資料表

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"}}

使用 NoSQL Workbench 與此結構描述設計

您可以將此最終結構描述匯入 NoSQL Workbench，這是為 DynamoDB 提供資料建模、資料視覺化，和查詢開發功能的視覺化工具，以進一步探索和編輯新專案。請依照下列步驟以開始使用：

1. 下載 NoSQL Workbench。如需詳細資訊，請參閱the section called “下載”。
2. 下載上面列出的 JSON 結構描述檔案，該檔案已經是 NoSQL Workbench 模型格式。
3. 將 JSON 結構描述檔案匯入到 NoSQL Workbench。如需詳細資訊，請參閱the section called “匯入現有的模型”。
4. 一旦您匯入到 NOSQL Workbench 後，便可以編輯資料模型。如需詳細資訊，請參閱the section called “編輯現有的模型”。

監控 DynamoDB 中的裝置狀態更新

本使用案例將說明如何使用 DynamoDB 監控 DynamoDB 中的裝置狀態更新 (或裝置狀態變更)。

使用案例

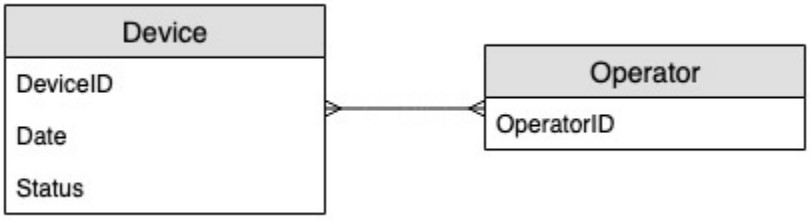
在 IoT 使用案例 (例如智慧工廠) 中，許多裝置需要由作業員監控，並定期將其狀態或日誌傳送至監控系統。當裝置發生問題時，裝置的狀態會從正常轉為警告。根據裝置中異常行為的嚴重性和類型，會有不同的日誌層級或狀態。然後，系統會指派作業員檢查裝置，並在需要時將問題呈報給主管。

此系統的典型存取模式包括：

- 建立裝置的日誌項目
- 取得特定裝置狀態的所有日誌，並顯示最新日誌
- 取得兩個日期之間，給指定作業員的所有日誌
- 取得呈報給指定主管的所有日誌
- 取得呈報給指定主管，並包含特定裝置狀態的所有日誌
- 取得呈報給指定主管，並包含特定日期之特定裝置狀態的所有日誌

實體關係圖

這是用於監控裝置狀態更新實體關係圖 (ERD)。



存取模式

須使用此存取模式監控裝置狀態更新。

1. `createLogEntryForSpecificDevice`
2. `getLogsForSpecificDevice`
3. `getWarningLogsForSpecificDevice`

4. getLogsForOperatorBetweenTwoDates
5. getEscalatedLogsForSupervisor
6. getEscalatedLogsWithSpecificStatusForSupervisor
7. getEscalatedLogsWithSpecificStatusForSupervisorForDate

架構設計演進

步驟 1：位址存取模式 1 (**createLogEntryForSpecificDevice**) 和 2 (**getLogsForSpecificDevice**)

裝置追蹤系統的擴展單位為個別裝置。在這個系統中，deviceID 會唯一識別裝置，進一步讓 deviceID 成為分割區索引鍵的理想候選者。每個設備會定期向追蹤系統發送資訊 (例如每五分鐘左右)，而這類排序會依日期為邏輯排序準則，因此成為排序索引鍵。此使用案例中的範例資料看起來應會與以下內容相似：

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

若要擷取特定裝置的日誌項目，可以使用分割區索引鍵 DeviceID="d#12345" 執行查詢操作。

步驟 2：位址存取模式 3 (**getWarningLogsForSpecificDevice**)

由於 State 是非索引鍵屬性，必須使用篩選條件表達式，才可利用目前的結構描述處理存取模式 3。在 DynamoDB 中，使用索引鍵條件表達式讀取資料後，才會套用篩選條件表達式。例如若想針對 d#12345 擷取警告日誌，利用分割區索引鍵 DeviceID="d#12345" 進行的查詢操作會讀取上表中的四個項目，然後篩選掉非處於警告狀態的項目。這種方法無法有效地大規模進行。如果排除項目的比例較低或不常執行查詢，則篩選條件表達式是排除查詢項目的好方法。但是，我們可以持續改善資料表設計，以便更有效率地從資料表中擷取大量項目，並篩選掉大部分的項目。

若想改變存取模式的處理方法，可以透過複合排序索引鍵。您可以從排序索引鍵已改為 State#Date 的 DeviceStateLog_3.json 匯入範例資料。此排序索引鍵結合了 State、# 以及 Date 屬性。此範例中，# 用來作為分隔符。資料現在看起來會像這樣：

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

若僅想擷取裝置的警告日誌，可利用此結構描述更切合查詢。查詢的索引鍵條件使用分割區索引鍵 DeviceID="d#12345" 與排序索引鍵 State#Date begins_with “WARNING”。此查詢只會讀取三個與警告狀態相關的項目。

步驟 3：位址存取模式 4 (**getLogsForOperatorBetweenTwoDates**)

您可以匯入 DeviceStateLog_4.json，其中 Operator 屬性已新增至具有範例資料的 DeviceStateLog 資料表。

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

由於 Operator 目前不是分割區索引鍵，因此無法根據以 OperatorID 為基礎的資料表執行直接鍵值對查詢。我們必須使用 OperatorID 上的全域次要索引，建立一個新的項目集合。存取模式必須根據日期進行查詢，因此日期是全域次要索引 (GSI) 的排序索引鍵屬性。這就是 GSI 現在的樣子：

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

針對存取模式 4 (getLogsForOperatorBetweenTwoDates)，您可以使用分割區索引鍵 OperatorID=Liz 查詢此 GSI，以及 2020-04-11T05:58:00 和 2020-04-24T14:50:00 之間的排序索引鍵 Date。

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

步驟 4：位址存取模式 5 (**getEscalatedLogsForSupervisor**)、6 (**getEscalatedLogsWithSpecificStatusForSupervisor**) 和 7 (**getEscalatedLogsWithSpecificStatusForSupervisorForDate**)

我們可以利用疏鬆索引處理這些存取模式。

全域次要索引依預設是疏鬆的，因此只有基礎資料表中包含主索引鍵屬性的項目，才會實際出現在索引中。針對要建模的存取模式，這是另一種排除無關項目的方式。

您可以匯入 DeviceStateLog_6.json，其中 `EscalatedTo` 屬性已新增至具有範例資料的 `DeviceStateLog` 資料表。如前所述，並非所有日誌都會呈報給主管。

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

您現在可以建立分割區索引鍵為 `EscalatedTo`、排序索引鍵為 `State#Date` 的新 GSI。請注意，只有當項目同時具有 `EscalatedTo` 和 `State#Date` 屬性時，才會顯示在索引中。

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

其餘的存取模式摘要如下：

下表摘要整理了所有存取模式，以及結構描述設計處理這些模式的方式：

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值	其他條件/篩選條件
createLogEntryForSpecificDevice	基礎資料表	PutItem	DeviceID=deviceId	State#Date=state#date	
getLogsForSpecificDevice	基礎資料表	Query	DeviceID=deviceId	State#Date begins_with "state1#"	ScanIndexForward = False
getWarningLogsForSpecificDevice	基礎資料表	Query	DeviceID=deviceId	State#Date begins_with "WARNING"	
getLogsForOperatorBetweenTwoDates	GSI-1	Query	Operator=operatorName	Date between date1 and date2	
getEscalatedLogsForSupervisor	GSI-2	Query	EscalatedTo=supervisorName		
getEscalatedLogsWithSpecifi	GSI-2	Query	EscalatedTo=supervisorName	State#Date begins_with "state1#"	

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值	其他條件/篩選條件
cStatusFor Supervisor					
getEscalatedLogsWithSpecificStatusForSupervisorForDate	GSI-2	Query	EscalatedTo=supervisorName	State#Date begins_with "state1#date1"	

最終結構描述

以下是最終結構描述設計。若要將此結構描述設計下載為 JSON 檔案，請參閱 GitHub 上的 DynamoDB 範例。

基礎資料表

Primary key		Attributes			
Partition key: DeviceID	**Sort key: State#Date**				
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State	
		Liz	2020-04-24T14:55:00	NORMAL	
	WARNING1#2020-04-24T14:40:00	Operator	Date	State	
		Liz	2020-04-24T14:40:00	WARNING1	
	WARNING1#2020-04-24T14:45:00	Operator	Date	State	
		Liz	2020-04-24T14:45:00	WARNING1	
	WARNING1#2020-04-24T14:50:00	Operator	Date	State	
		Liz	2020-04-24T14:50:00	WARNING1	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State	
		Liz	2020-04-11T06:00:00	NORMAL	
	NORMAL#2020-04-11T09:30:00	Operator	Date	State	
		Sue	2020-04-11T09:30:00	NORMAL	
	WARNING2#2020-04-11T09:25:00	Operator	Date	State	
		Sue	2020-04-11T09:25:00	WARNING2	
	WARNING3#2020-04-11T05:50:00	Operator	Date	State	
		Sue	2020-04-11T05:50:00	WARNING3	
	WARNING3#2020-04-11T05:55:00	Operator	Date	State	
		Liz	2020-04-11T05:55:00	WARNING3	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State	
		Sue	2020-04-27T16:10:00	WARNING4	
	WARNING4#2020-04-27T16:15:00	Operator	Date	State	EscalatedTo
		Sue	2020-04-27T16:15:00	WARNING4	Sara

GSI-1

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
Liz	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

GSI-2

Primary key		Attributes			
Partition key: EscalatedTo	Sort key: State#Date				
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator	Date	State
		d#11223	Sue	2020-04-27T16:15:00	WARNING4

使用 NoSQL Workbench 與此結構描述設計

您可以將此最終結構描述匯入 NoSQL Workbench，這是為 DynamoDB 提供資料建模、資料視覺化，和查詢開發功能的視覺化工具，以進一步探索和編輯新專案。請依照下列步驟以開始使用：

1. 下載 NoSQL Workbench。如需詳細資訊，請參閱the section called “下載”。
2. 下載上面列出的 JSON 結構描述檔案，該檔案已經是 NoSQL Workbench 模型格式。
3. 將 JSON 結構描述檔案匯入到 NoSQL Workbench。如需詳細資訊，請參閱the section called “匯入現有的模型”。
4. 一旦您匯入到 NOSQL Workbench 後，便可以編輯資料模型。如需詳細資訊，請參閱the section called “編輯現有的模型”。

使用 DynamoDB 做為線上商店的資料存放區

此使用案例說明如何將 DynamoDB 做為線上商店 (或電子商店) 的資料存放區。

使用案例

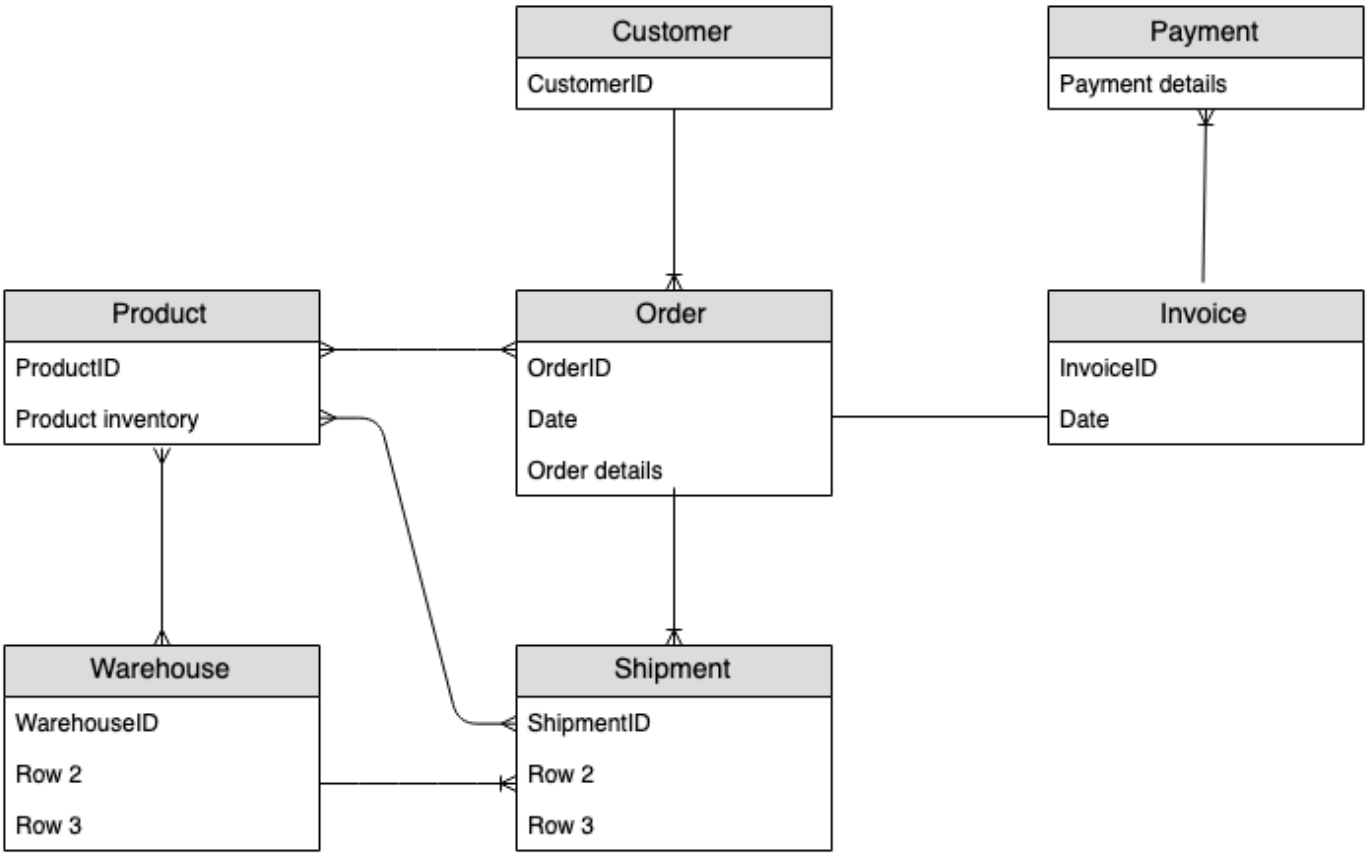
使用者可在線上商店瀏覽各種產品，並進行購買。結帳時客戶可以使用折扣碼或禮品卡付款，並用信用卡支付剩餘金額。從數個倉庫中找出購買產品後，便會運送到客戶提供的地址。線上商店存放區的典型存取模式包括：

- 取得具指定 CustomerID 的客戶
- 取得具指定 ProductID 的產品
- 取得具指定 WarehouseID 的倉庫
- 透過 ProductID 取得所有倉庫的產品庫存
- 取得具指定 OrderID 的訂單
- 取得具指定 OrderID 的所有產品

- 取得具指定 OrderID 的發票
- 取得具指定 OrderID 的所有運送貨物
- 取得指定日期範圍內，具指定 ProductID 的所有訂單
- 取得具指定 InvoiceID 的發票
- 取得具指定 InvoiceID 的所有付款
- 取得具指定 ShipmentID 的所有運送詳細資訊
- 取得具指定 WarehouseID 的所有運送貨物
- 取得具指定 WarehouseID 的所有產品庫存
- 取得指定日期範圍內，具指定 CustomerID 的所有發票
- 取得指定日期範圍內，由指定 CustomerID 訂購的所有產品

實體關係圖

針對如何使用 DynamoDB 做為線上商店的資料存放區，這是用於建模的實體關係圖 (ERD)。



存取模式

針對如何使用 DynamoDB 做為線上商店的資料存放區，這些是可採用的存取模式。

1. getCustomerByCustomerId
2. getProductByProductId
3. getWarehouseByWarehouseId
4. getProductInventoryByProductId
5. getOrderDetailsByOrderId
6. getProductByOrderId
7. getInvoiceByOrderId
8. getShipmentByOrderId
9. getOrderByProductIdForDateRange
10. getInvoiceByInvoiceId
11. getPaymentByInvoiceId
12. getShipmentDetailsByShipmentId
13. getShipmentByWarehouseId
14. getProductInventoryByWarehouseId
15. getInvoiceByCustomerIdForDateRange
16. getProductsByCustomerIdForDateRange

架構設計演進

使用 DynamoDB 專用 NoSQL Workbench，匯入 AnOnlineShop_1.json 以建立命名為 AnOnlineShop 的新資料模型，和一個命名為 OnlineShop 的新資料表。請注意，我們將分割區索引鍵和排序索引鍵命名為通用名稱 PK 和 SK，藉此可在同一個資料表中儲存不同類型的實體。

步驟 1：位址存取模式 1 (**getCustomerByCustomerId**)

匯入 AnOnlineShop_2.json 處理存取模式 1 (getCustomerByCustomerId)。某些實體與其他實體並無關聯，因此我們將針對這些實體使用相同的 PK 和 SK 值。在範例資料中，請注意索引鍵字首為 c#，以和稍後會從其他實體新增的 customerId 進行區別。也請於其他實體重複執行此做法。

若要處理此存取模式，可以利用 PK=customerId 和 SK=customerId 執行 GetItem 操作。

步驟 2：位址存取模式 2 (**getProductByProductId**)

匯入 AnOnlineShop_3.json，針對 product 實體處理存取模式 2 (getProductByProductId)。產品實體字首為 p#，並且已使用相同的排序索引鍵屬性來儲存 customerID 與 productID。透過通用命名和垂直分割建立項目集合，可設計有效的單一資料表。

若要處理此存取模式，可以利用 PK=productId 和 SK=productId 執行 GetItem 操作。

步驟 3：位址存取模式 3 (**getWarehouseByWarehouseId**)

匯入 AnOnlineShop_4.json，針對 getWarehouseByWarehouseId 實體處理存取模式 3 (warehouse)。目前我們已將 customer、product 與 warehouse 實體加入同一個資料表，且這些實體均使用字首和 EntityType 屬性。類型屬性 (或字首命名) 能提升模型的可讀性。如果只單純將不同實體的英數 ID 儲存在同一屬性中，閱讀上會相當困難。若少了識別符，就會難以分辨不同實體。

若要處理此存取模式，可以利用 PK=warehouseId 和 SK=warehouseId 執行 GetItem 操作。

基礎資料表：

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"}}	

步驟 4：位址存取模式 4 (**getProductInventoryByProductId**)

匯入 AnOnlineShop_5.json，處理存取模式 4 (getProductInventoryByProductId)。而 warehouseItem 實體會用於追蹤每個倉庫中的產品數量。當倉庫新增或移除產品時，通常會更新此項目。正如在 ERD 所看到的，product 和 warehouse 之間存在多對多關係。在此處，從 product 至 warehouse 的一對多關係已建立模型 warehouseItem。接著，從 product 至 warehouse 的一對多關係也會進行建模。

可透過查詢 PK=ProductId 和 SK begins_with "w#" 處理存取模式 4。

關於 begins_with() 和其他可套用至排序索引鍵的表達式，詳細資訊請參閱索引鍵條件表達式。

基礎資料表：

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"}}	

步驟 5：位址存取模式 5 (**getOrderDetailsByOrderId**) 和 6 (**getProductByOrderId**)

透過匯入 AnOnlineShop_6.json，將更多 customer、product 和 warehouse 項目新增至資料表。然後，匯入 AnOnlineShop_7.json，針對 order 建立項目集合以處理存取模式 5 (getOrderDetailsByOrderId) 和 6 (getProductByOrderId)。您可以看到 order 和 product 之間的一對多關係已建模為 orderItem 實體。

若想處理存取模式 5 (getOrderDetailsByOrderId)，請透過 PK=orderId 查詢資料表，了解與訂單相關的所有資訊，包括 customerId 和已訂購產品。

基礎資料表：

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

若想處理存取模式 6 (getProductByOrderId)，僅能以 order 讀取產品，此方法可透過利用 PK=orderId 和 SK begins_with "p#" 查詢資料表完成。

基礎資料表：

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

步驟 6：位址存取模式 7 (**getInvoiceByOrderId**)

匯入 AnOnlineShop_8.json，新增 invoice 實體到訂單項目集合，以處理存取模式 7 (getInvoiceByOrderId)。若要處理此存取模式，可以利用 PK=orderId 和 SK begins_with "i#" 執行查詢操作。

基礎資料表：

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

步驟 7：位址存取模式 8 (**`getShipmentByOrderId`**)

匯入 AnOnlineShop_9.json，新增 `shipment` 實體到訂單項目集合，以處理存取模式 8 (`getShipmentByOrderId`)。我們可透過在單一資料表設計中新增更多實體類型，擴展同一個的垂直分割模型。請注意，訂單項目集合中項目間的關係，並不同於 `order`、`shipment`、`orderItem` 及 `invoice` 實體間所擁有的關係。

若想透過 `orderId` 取得運送貨物，您可以利用 `PK=orderId` 和 `SK begins_with “sh#”` 執行查詢操作。

基礎資料表：

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

步驟 8：位址存取模式 9 (**`getOrderByProductIdForDateRange`**)

在上個步驟中，我們建立了訂單項目集合。此存取模式具有新的查詢維度 (`ProductID` 和 `Date`)，會要求您掃描整個資料表並篩選相關記錄，以擷取目標項目。為處理這類存取模式，必須建立全域次要索引 (GSI)。匯入 AnOnlineShop_10.json，利用 GSI 建立新的項目集合，以從數個訂單項目集合擷取 `orderItem` 資料。目前資料內有 `GSI1-PK` 和 `GSI1-SK`，分別為 `GSI1` 的分割區索引鍵和排序索引鍵。

DynamoDB 會自動將資料表中包含 GSI 索引鍵屬性的項目填入 GSI，不須額外手動插入 GSI。

若想處理存取模式 9，請利用 `GSI1-PK=productId` 和 `GSI1SK between (date1, date2)` 查詢 `GSI1`。

基礎資料表：

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

步驟 9：位址存取模式 10 (**`getInvoiceByInvoiceId`**) 和 11 (**`getPaymentByInvoiceId`**)

匯入 AnOnlineShop_11.json，處理與 `invoice` 相關的存取模式 10 (`getInvoiceByInvoiceId`) 和 11 (`getPaymentByInvoiceId`)。即使這兩種存取模式並不相同，仍須透過索引鍵條件以實現。`Payments` 的定義為 `invoice` 實體上具地圖資料類型的屬性。

> **Note**
>
> 為了儲存不同實體的資訊，`GSI1-PK` 和 `GSI1-SK` 會進行多載，以便從同一個 GSI 取得多個存取模式。如需 GSI 多載的詳細資訊，請參閱 在 DynamoDB 中多載全域次要索引。

若想處理存取模式 10 和 11，請利用 `GSI1-PK=invoiceId` 和 `GSI1-SK=invoiceId` 查詢 `GSI1`。

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

步驟 10：位址存取模式 12 (**`getShipmentDetailsByShipmentId`**) 和 13 (**`getShipmentByWarehouseId`**)

匯入 AnOnlineShop_12.json，處理存取模式 12 (`getShipmentDetailsByShipmentId`) 和 13 (`getShipmentByWarehouseId`)。

請注意，`shipmentItem` 實體已新增至基礎資料表上的訂單項目集合，以在單一查詢作業中擷取訂單的所有詳細資訊。

基礎資料表：

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				

GSI1 分割區和排序索引鍵已用於建立 shipment 和 shipmentItem 之間的一對多關係模型。若想處理存取模式 12 (getShipmentDetailsByShipmentId)，請利用 GSI1-PK=shipmentId 和 GSI1-SK=shipmentId 查詢 GSI1。

GSI1：

Primary key		Attributes							
Partition key: GSI1-PK	Sort key: GSI1-SK								
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#54321	shipmentItem	2				
		PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
sh#88899	sh#88899	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				
		PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
sh#98765	sh#98765	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

針對存取模式 13 (`getShipmentByWarehouseId`)，我們必須建立另一個 GSI (`GSI2`)，來為 `warehouse` 和 `shipment` 之間新的一對多關係建立模型。若想處理此存取模式，請利用 `GSI2-PK=warehouseId` 和 `GSI2-SK begins_with “sh#”` 查詢 `GSI2`。

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

步驟 11：位址存取模式 14 (**`getProductInventoryByWarehouseId`**)、15 (**`getInvoiceByCustomerIdForDateRange`**) 和 16 (**`getProductsByCustomerIdForDateRange`**)

匯入 AnOnlineShop_13.json，新增與下一組存取模式相關的資料。若想處理存取模式 14 (`getProductInventoryByWarehouseId`)，請利用 `GSI2-PK=warehouseId` 和 `GSI2-SK begins_with “p#”` 查詢 `GSI2`。

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	

若想處理存取模式 15 (`getInvoiceByCustomerIdForDateRange`)，請利用 `GSI2-PK=customerId` 和 `GSI2-SK between (i#date1, i#date2)` 查詢 GSI2。

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	

若想處理存取模式 16 (`getProductsByCustomerIdForDateRange`)，請利用 `GSI2-PK=customerId` 和 `GSI2-SK between (p#date1, p#date2)` 查詢 GSI2。

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**
>
> 在 NoSQL Workbench 中，面向代表適用於 DynamoDB 的應用程式不同資料存取模式。面向為您提供了一種查看資料表中的資料子集的方法，而無需查看不符合面向限制的記錄。面向是一種視覺資料建模工具，在 DynamoDB 中不作為可用的建構存在，因為其純粹用於輔助存取模式建模。
>
> 匯入 AnOnlineShop_facets.json，以檢視此使用案例的面向。

下表摘要整理了所有存取模式，以及結構描述設計處理這些模式的方式：

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值
getCustomerByCustomerId	基礎資料表	GetItem	PK=customerId	SK=customerId
getProductByProductId	基礎資料表	GetItem	PK=productId	SK=productId
getWarehouseByWarehouseId	基礎資料表	GetItem	PK=warehouseId	SK=warehouseId
getProductInventoryByProductId	基礎資料表	Query	PK=productId	SK begins_with "w#"
getOrderDetailsByOrderId	基礎資料表	Query	PK=orderId	
getProductByOrderId	基礎資料表	Query	PK=orderId	SK begins_with "p#"
getInvoiceByOrderId	基礎資料表	Query	PK=orderId	SK begins_with "i#"
getShipmentByOrderId	基礎資料表	Query	PK=orderId	SK begins_with "sh#"
getOrderByProductIdForDateRange	GSI1	Query	PK=productId	SK between date1 and date2
getInvoiceByInvoiceId	GSI1	Query	PK=invoiceId	SK=invoiceId
getPaymentByInvoiceId	GSI1	Query	PK=invoiceId	SK=invoiceId

存取模式	基礎資料表/GSI/LSI	作業	分割區索引鍵值	排序索引鍵值
getShipmentDetailsByShipmentId	GSI1	Query	PK=shipmentId	SK=shipmentId
getShipmentByWarehouseId	GSI2	Query	PK=warehouseId	SK begins_with "sh#"
getProductInventoryByWarehouseId	GSI2	Query	PK=warehouseId	SK begins_with "p#"
getInvoiceByCustomerIdForDateRange	GSI2	Query	PK=customerId	SK between i#date1 and i#date2
getProductsByCustomerIdForDateRange	GSI2	Query	PK=customerId	SK between p#date1 and p#date2

線上商店最終結構描述

以下是最終結構描述設計。若要將此結構描述設計下載為 JSON 檔案，請參閱 GitHub 上的 DynamoDB 設計模式。

基礎資料表

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#54321	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

使用 NoSQL Workbench 與此結構描述設計

您可以將此最終結構描述匯入 NoSQL Workbench，這是為 DynamoDB 提供資料建模、資料視覺化，和查詢開發功能的視覺化工具，以進一步探索和編輯新專案。請依照下列步驟以開始使用：

1. 下載 NoSQL Workbench。如需詳細資訊，請參閱the section called “下載”。
2. 下載上面列出的 JSON 結構描述檔案，該檔案已經是 NoSQL Workbench 模型格式。
3. 將 JSON 結構描述檔案匯入到 NoSQL Workbench。如需詳細資訊，請參閱the section called “匯入現有的模型”。
4. 一旦您匯入到 NOSQL Workbench 後，便可以編輯資料模型。如需詳細資訊，請參閱the section called “編輯現有的模型”。

在 DynamoDB 中製作關聯式資料模型的最佳實務

本節提供在 Amazon DynamoDB 中建構關聯式資料模型的最佳實務。首先，我們要介紹傳統的資料模型建構概念。接著會說明使用 DynamoDB 相較於傳統關聯式資料庫管理系統的優點，讓您了解它如何消除 JOIN 操作的需求並減少額外負荷。

然後我們將說明如何設計可有效擴展的 DynamoDB 資料表。最後我們會提供一個範例來說明，如何在 DynamoDB 中建模關聯式資料的模型。

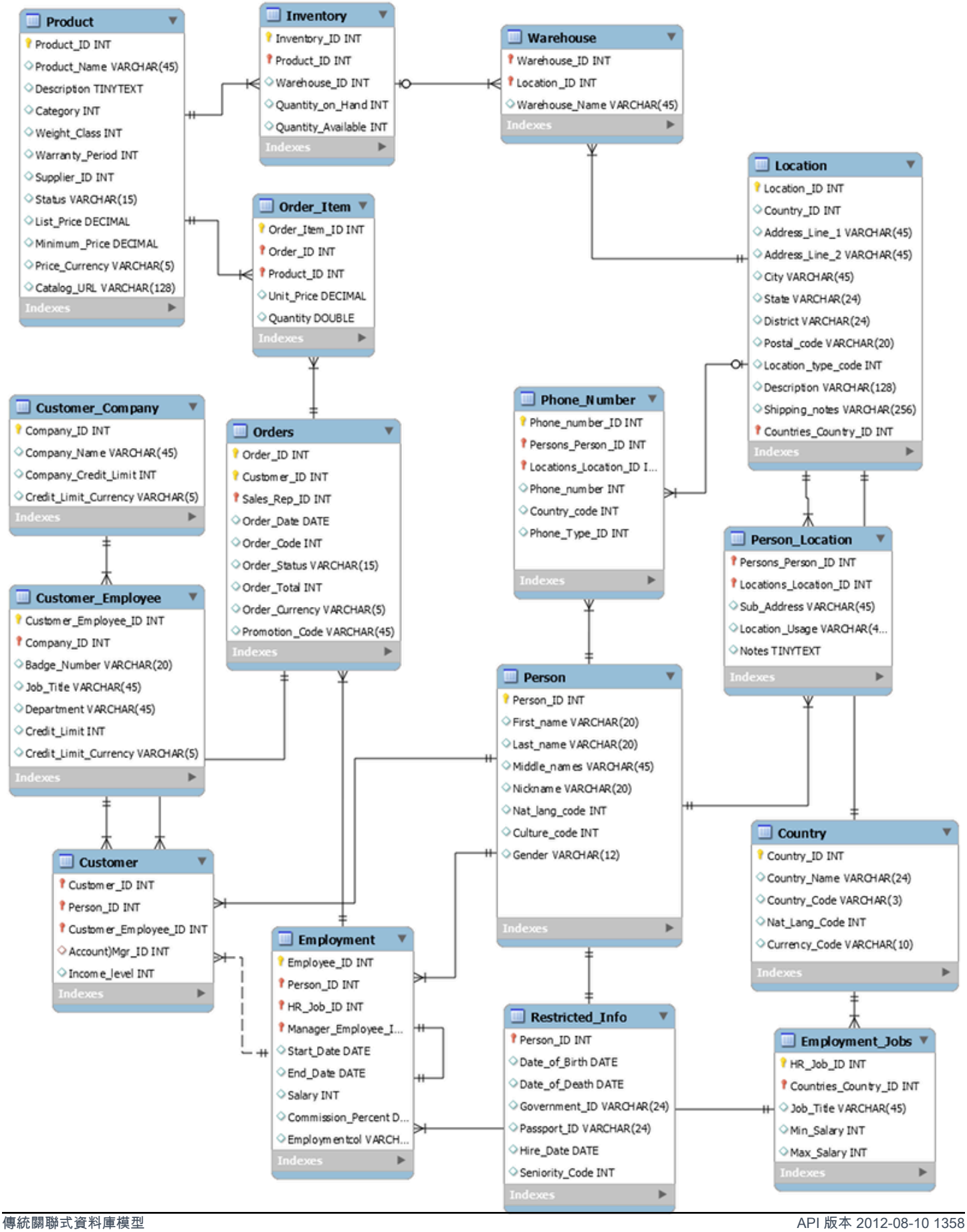
主題

- 傳統關聯式資料庫模型
- DynamoDB 如何免除 JOIN 操作的需求
- DynamoDB 交易如何消除寫入程序的負荷
- 在 DynamoDB 中製作關聯式資料模型的第一步
- 在 DynamoDB 中打造關聯式資料模型的範例

傳統關聯式資料庫模型

傳統關聯式資料庫管理系統 (RDBMS) 會將資料儲存在標準化的關聯式結構中。關聯式資料模型的目標是減少資料的重複 (透過標準化)，以支援參考完整性並減少資料異常。

下列結構描述是一般訂單輸入應用程式的關聯式資料模型範例。此應用程式支援人力資源結構描述，並用它來為虛擬製造商的營運和業務支援系統提供支援。


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

由於 DynamoDB 是非關聯式資料庫服務，相較於傳統關聯式資料庫管理系統，它提供了更多優勢。

DynamoDB 如何免除 JOIN 操作的需求

RDBMS 使用結構式查詢語言 (SQL) 將資料傳回至應用程式。由於資料模型的標準化，此類查詢通常需要使用 JOIN 運算子來合併來自一或多個資料表的資料。

例如，若要在可以運送任何項目的所有倉庫中，依庫存中的數量來排序產生購買順序項目清單，您可以對上述結構描述發出以下 SQL 查詢。

```
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
```

此類 SQL 查詢提供存取資料的彈性 API，但他們需要的處理量很大。查詢中的每個聯結都會增加查詢的執行時期複雜度，因為每個資料表的資料必須暫存，然後再組裝以傳回結果集。

可能影響執行查詢時間長短的其他因素包括資料表的大小，以及要聯結的欄是否有索引。上述查詢會在數個資料表中起始複雜查詢，接著排序結果集。

免除使用 JOINs 的需求，是 NoSQL 資料建模的核心。這就是為什麼我們建置 DynamoDB 來支援 Amazon.com，以及為什麼 DynamoDB 可以在任何規模提供一致的效能。考慮到 SQL 查詢和 的執行時間複雜性JOINs，RDBMS 效能不會大規模保持不變。這會隨著客戶應用程式的增長而導致效能問題。

雖然資料標準化確實可減少儲存到磁碟的資料量，但通常影響效能的最受限資源是 CPU 時間和網路延遲。

DynamoDB 的建置目的是消除 JOINs (並鼓勵取消資料標準化) 和最佳化資料庫架構，以便透過對項目的單一請求來完全回應應用程式查詢，將這兩種限制降到最低。這些品質讓 DynamoDB 能夠在任何規模下提供個位數的毫秒速度效能。這是因為 DynamoDB 操作的執行時期複雜性對於常見存取模式來說是固定的，不會隨著資料大小而改變。

DynamoDB 交易如何消除寫入程序的負荷

可能降低 RDBMS 速度的另一個因素，是使用交易來寫入標準化結構描述。如範例所示，大部分線上交易處理 (OLTP) 應用程式使用的關聯式資料結構儲存在 RDBMS 中時，必須細分並分散在多個邏輯資料表。

因此，符合 ACID 交易架構是必須的，以避免在應用程式嘗試讀取正處於寫入過程之物件時可能發生的競爭條件和資料完整性問題。此類交易框架與關聯式結構描述結合使用時，會大幅增加寫入程序的負荷。

在 DynamoDB 中實作交易會禁止 RDBMS 上常見的擴展問題發生。DynamoDB 以單一 API 呼叫的形式發出交易，並限制該單一交易中可存取的項目數量，藉此達到這個目的。長時間執行的交易可能因為交易永不關閉，所以長時間或永久鎖定資料，進而導致操作問題。

為了防止 DynamoDB 中發生此類問題，會使用兩個不同的 API 操作來實作交易：`TransactWriteItems` 和 `TransactGetItems`。這些 API 操作沒有 RDBMS 中常見的開始和結束語義。此外，DynamoDB 在交易中有 100 個項目的存取限制，同樣是為了防止交易長時間執行。若要進一步了解 DynamoDB 交易，請參閱使用交易。

基於這些原因，當您的企業需要對高流量查詢提供低延遲回應時，在技術與經濟的考量上通常會利用 NoSQL 系統。Amazon DynamoDB 可避免這些問題，協助解決限制關聯式系統可擴展性的問題。

基於以下原因，RDBMS 的效能通常無法適當地擴展：

- 因此會使用昂貴的聯結來重新組合必要的查詢結果檢視。
- 資料庫將資料標準化並存放在多個資料表中，這些資料表需要多個查詢來寫入至磁碟中。
- 這通常會引發 ACID 合規交易系統的效能成本，

DynamoDB 順利擴展的原因為下：

- 結構描述靈活性可讓 DynamoDB 在單一項目中存放複雜階層資料。
- 複合索引鍵設計可讓您將相關的項目存放在相同資料表的鄰近位置。
- 交易以單一操作形式執行。可存取的項目數限制為 100 個，以避免操作長時間執行。

對資料存放的查詢變得簡單多了，通常是使用以下格式：

```
SELECT * FROM Table_X WHERE Attribute_Y = "somevalue"
```

與先前範例中的 RDBMS 相比，DynamoDB 傳回要求資料的程序簡單許多。

在 DynamoDB 中製作關聯式資料模型的第一步

> **Note**
>
> NoSQL 設計思維與 RDBMS 設計不同。針對 RDBMS，您可以建立標準化的資料模型，而不需考量存取模式。您可以在稍後有新問題與查詢要求時擴展此模型。相反地，在 Amazon DynamoDB 中，您不應開始設計結構描述，除非您知道其需要回答的問題。您絕對必須事先了解企業問題和應用程式使用案例。

若要開始設計能有效擴展的 DynamoDB 資料表，您必須先執行數個步驟，以識別操作所需的存取模式與其需要支援的企業支援系統 (OSS/BSS)：

- 針對新應用程式，檢視活動與目標相關的使用者案例。記錄您識別的各種使用案例，並分析案例需要的存取模式。
- 針對現有應用程式，分析查詢記錄以找出人員目前使用系統的方式與索引鍵存取模式為何。

完成此程序後，您將收到一個清單，其看起來可能與以下類似。

訂單項目應用程式的存取模式

模式 #	存取模式描述
1	按員工 ID 查詢員工詳細資訊
2	按員工姓名查詢員工詳細資訊
3	尋找員工的電話號碼 (s)
4	尋找客戶的電話號碼 (s)
5	在日期範圍內為客戶取得訂單
6	顯示日期範圍內的所有開啟的訂單
7	查看所有最近僱用的員工
8	尋找 Warehouse 中的所有員工
9	取得產品訂單上的所有項目

模式 #	存取模式描述
10	在所有倉儲取得產品的庫存
11	依帳戶代表取得客戶
12	依帳戶代表取得訂單
13	取得具有任務標題的員工
14	依產品和倉儲取得庫存
15	取得產品庫存總計

在實際應用中，您的清單可能會更長。但此集合代表您可能會在生產環境中找到的查詢模式複雜度範圍。

DynamoDB 結構描述設計的現代方法使用彙總導向原則，根據存取模式而非剛性實體邊界來分組資料。此方法考慮多種設計模式：

- 單一資料表設計 - 使用複合排序索引鍵、超載的全域次要索引和相鄰清單模式，在一個資料表中存放多個實體類型
- 多資料表設計 - 針對具有獨立操作特性和低存取相關性的實體使用單獨的資料表，以及跨實體查詢的策略 GSIs
- 彙總設計 - 一律同時存取時嵌入相關資料 （訂單 + OrderItems) 或使用項目集合來識別關係 （產品 + 庫存）

這些方法的選擇取決於您的特定存取模式、資料特性和操作需求。您可以使用這些元素來建構資料，如此應用程式就可以使用對資料表或索引的單一查詢來擷取特定存取模式所需的項目。

> **Note**
>
> 單一資料表和多資料表設計之間的選擇取決於您的特定需求。當實體具有高存取相互關聯性和類似的操作特性時，單一資料表設計可正常運作。當實體有獨立的操作需求、不同的存取模式，或是您需要明確的操作界限時，最好採用多資料表設計。本指南中的範例示範了具有策略彙總和非標準化的多資料表方法。

若要使用適用於 DynamoDB 的 NoSQL Workbench 來協助視覺化您的分割區索引鍵設計，請參閱 使用 NoSQL Workbench 建立資料模型。

在 DynamoDB 中打造關聯式資料模型的範例

此範例說明如何在 Amazon DynamoDB 中打造關聯式資料模型。DynamoDB 資料表設計對應至顯示在中的關聯式順序項目結構描述關聯式模型。此設計使用多個專用資料表，而不是單一相鄰清單，提供清晰的操作界限，同時利用策略 GSIs 有效地為所有存取模式提供服務。

設計方法使用彙總導向原則，根據存取模式而非剛性實體邊界來分組資料。關鍵設計決策包括針對具有低存取關聯性的實體使用個別資料表、一律同時存取時內嵌相關資料，以及使用項目集合來識別關係。

下表及其隨附的索引支援關聯式順序項目結構描述：

員工資料表設計

員工資料表會將員工資訊儲存為每個項目的單一實體，針對直接員工查詢進行最佳化，並透過策略 GSIs 支援多個查詢模式。此資料表示範為具有獨立操作特性和低跨實體存取相關性的實體設計個別資料表的原則。

資料表使用不含排序索引鍵的簡單分割區索引鍵 (employee_id)，因為每個員工都是不同的實體。四個 GSIs 可讓不同屬性有效率地查詢：

- EmployeeByName GSI - 將 INCLUDE 投影與所有員工屬性搭配使用，以支援依名稱的完整員工詳細資訊擷取，並以 employee_id 作為排序索引鍵處理潛在的重複名稱
- EmployeeByWarehouse GSI - 僅將 INCLUDE 投影與基本屬性 (name， job_title， hire_date) 搭配使用，將儲存成本降至最低，同時支援以倉儲為基礎的查詢
- EmployeeByJobTitle GSI - 使用 INCLUDE 投影啟用角色型查詢以進行報告和組織分析
- EmployeeByHireDate GSI - 使用靜態分割區索引鍵值 "EMPLOYEE" 搭配 hire_date 作為排序索引鍵，以啟用最近招聘的高效日期範圍查詢。由於員工新增/更新通常低於 1，000 WCU，因此單一分割區可以處理寫入負載，而不會發生熱分割區問題

員工資料表 - 基礎資料表結構

employee_id (PK)	name	phone_numbers	倉儲 ID	job_title	hire_date	entity_type
emp_001	John Smith	【"+1-555-0101"】	wh_sea	管理員	2024-03-15	員工

employee_id (PK)	name	phone_numbers	倉儲 ID	job_title	hire_date	entity_type
emp_002	Jane Doe	【"+1-555-0102"、"+1-555-0103"】	wh_sea	關聯	2025-01-10	員工
emp_003	Bob Wilson	【"+1-555-0104"】	wh_pdx	關聯	2025-06-20	員工
emp_004	Alice 棕色	【"+1-555-0105"】	wh_pdx	主管	2023-11-05	員工
emp_005	Charlie Davis	【"+1-555-0106"】	wh_sea	關聯	2025-12-01	員工

EmployeeByName GSI - 支援員工名稱查詢

名稱 (GSI-PK)	employee_id (GSI-SK)	phone_numbers	倉儲 ID	job_title	hire_date
Alice 棕色	emp_004	【"+1-555-0105"】	wh_pdx	主管	2023-11-05
Bob Wilson	emp_003	【"+1-555-0104"】	wh_pdx	關聯	2025-06-20
Charlie Davis	emp_005	【"+1-555-0106"】	wh_sea	關聯	2025-12-01
Jane Doe	emp_002	【"+1-555-0102"、"+1-555-0103"】	wh_sea	關聯	2025-01-10
John Smith	emp_001	【"+1-555-0101"】	wh_sea	管理員	2024-03-15

EmployeeByWarehouse GSI - 支援倉儲查詢

warehouse_id (GSI-PK)	employee_id (GSI-SK)	name	job_title	hire_date
wh_pdx	emp_003	Bob Wilson	關聯	2025-06-20
wh_pdx	emp_004	Alice 棕色	主管	2023-11-05
wh_sea	emp_001	John Smith	管理員	2024-03-15
wh_sea	emp_002	Jane Doe	關聯	2025-01-10
wh_sea	emp_005	Charlie Davis	關聯	2025-12-01

EmployeeByJobTitle GSI - 支援任務標題查詢

job_title (GSI-PK)	employee_id (GSI-SK)	name	倉儲 ID	hire_date
關聯	emp_002	Jane Doe	wh_sea	2025-01-10
關聯	emp_003	Bob Wilson	wh_pdx	2025-06-20
關聯	emp_005	Charlie Davis	wh_sea	2025-12-01
管理員	emp_001	John Smith	wh_sea	2024-03-15
主管	emp_004	Alice 棕色	wh_pdx	2023-11-05

EmployeeByHireDate GSI - 支援最近僱用查詢

entity_type (GSI-PK)	hire_date (GSI-SK)	employee_id	name	倉儲 ID
員工	2023-11-05	emp_004	Alice 棕色	wh_pdx
員工	2024-03-15	emp_001	John Smith	wh_sea
員工	2025-01-10	emp_002	Jane Doe	wh_sea

entity_type (GSI-PK)	hire_date (GSI-SK)	employee_id	name	倉儲 ID
員工	2025-06-20	emp_003	Bob Wilson	wh_pdx
員工	2025-12-01	emp_005	Charlie Davis	wh_sea

客戶資料表設計

客戶資料表使用 account_rep_id 的策略非正規化來維護客戶資訊，以啟用有效的帳戶代表查詢。此設計選擇會降低查詢效能的儲存體額外負荷，因此不需要在客戶和帳戶代表資料之間加入。

資料表支援每位客戶使用清單屬性的多個電話號碼，示範 DynamoDB 的結構描述彈性。單一 GSI 可啟用帳戶代表性工作流程：

- CustomerByAccountRep GSI - 使用 INCLUDE 投影搭配名稱和電子郵件屬性，以支援帳戶代表客戶管理，而不需要完整的客戶記錄擷取

客戶資料表 - 基礎資料表結構

customer_id (PK)	name	phone_numbers	email	account_rep_id
cust_001	Acme Corp	【"+1-555-1001"】	contact@acme.com	rep_001
cust_002	TechStart Inc	【"+1-555-1002"、"+1-555-1003"】	info@techstart.com	rep_001
cust_003	全球交易員	【"+1-555-1004"】	sales@globaltraders.com	rep_002
cust_004	BuildRight LLC	【"+1-555-1005"】	orders@buildright.com	rep_002
cust_005	FastShip Co	【"+1-555-1006"】	support@fastship.com	rep_003

CustomerByAccountRep GSI - 支援帳戶代表查詢

account_rep_id (GSI-PK)	customer_id (GSI-SK)	name	email
rep_001	cust_001	Acme Corp	contact@acme.com
rep_001	cust_002	TechStart Inc	info@techstart.com
rep_002	cust_003	全球交易員	sales@globaltraders.com
rep_002	cust_004	BuildRight LLC	orders@buildright.com
rep_003	cust_005	FastShip Co	support@fastship.com

訂單資料表設計

訂單資料表針對訂單標頭和訂單項目使用具有個別項目的垂直分割。此設計可實現高效的產品型查詢，同時在同一分割區中維護所有訂單元件，以實現高效存取。每個訂單包含多個項目：

- 訂單標頭 - 包含具有 PK=order_id、SK=order_id 的訂單中繼資料
- 訂單項目 - PK=order_id、SK=product_id 的個別明細項目，啟用直接產品查詢

> **Note**
>
> 這種垂直分割方法會交換內嵌訂單項目的簡單性，以增強查詢彈性。每個訂單項目都會成為單獨的 DynamoDB 項目，可實現高效的產品型查詢，同時將所有訂單資料維持在同一個分割區中，以便在單一請求中高效擷取。

資料表包含 account_rep_id （與客戶資料表重複） 的策略非正規化，以啟用直接帳戶代表查詢，而不需要客戶查詢。對於高輸送量寫入案例，OPEN 訂單包含狀態和碎片屬性，以啟用跨多個分割區的寫入碎片。

四個 GSIs支援具有最佳化投影的不同查詢模式：

- OrderByCustomerDate GSI - 使用 INCLUDE 投影搭配訂單摘要和項目詳細資訊，以支援具有日期範圍篩選的客戶訂單歷史記錄
- OpenOrdersByDate GSI （稀疏、碎片） - 使用具有 5 個碎片的多屬性分割區索引鍵 （狀態 + 碎片），以跨分割區分配 5，000 WPS （每秒寫入數） （每個 WPS，每個分割區都符合 DynamoDB 的 1，000 WCU)。僅索引 OPEN 訂單 （總計 20%)，這有助於降低 GSI 儲存成本。需要跨所有 5 個碎片與用戶端結果合併的平行查詢
- OrderByAccountRep GSI - 使用 INCLUDE 投影搭配訂單摘要屬性，以支援沒有完整訂單詳細資訊的帳戶代表工作流程
- ProductInOrders GSI - 從 OrderItem 記錄 (PK=order_id、SK=product_id) 建立，此 GSI 可讓查詢尋找包含特定產品的所有訂單。使用 INCLUDE 投影搭配訂單內容 (customer_id、order_date、quity) 進行產品需求分析

順序資料表 - 基礎資料表結構 （垂直分割）

PK	SK	customer_id	order_date	status	account_rep_id	數量	價格	碎片
ord_001	ord_001	cust_001	2025-11-15	CLOSED	rep_001			
ord_001	prod_100					5	25.00	
ord_002	ord_002	cust_001	2025-12-20	OPEN	rep_001			0
ord_002	prod_101					10	15.00	
ord_003	ord_003	cust_002	2026-01-05	OPEN	rep_001			2
ord_003	prod_100					3	25.00	

OrderByCustomerDate GSI - 支援客戶訂單查詢

customer_id (GSI-PK)	order_date (GSI-SK)	order_id	status	total_amount	order_items	碎片
cust_001	2025-11-15	ord_001	CLOSED	225.00	【{product_id："prod_100"，數量：5}】	
cust_001	2025-12-20	ord_002	OPEN	150.00	【{product_id："prod_101"，數量：10}】	0
cust_002	2026-01-05	ord_003	OPEN	175.00	【{product_id："prod_100"，數量：3}】	2
cust_003	2025-10-10	ord_004	CLOSED	250.00	【{product_id："prod_101"，數量：5}】	
cust_004	2026-01-03	ord_005	OPEN	200.00	【{product_id："prod_100"，數量：20}】	1

OpenOrdersByDate GSI（稀疏、碎片）- 支援高輸送量的開放訂單查詢

狀態 (GSI-PK-1)	碎片 (GSI-PK-2)	order_date (SK)	order_id	customer_id	account_rep_id	order_items	total_amount
OPEN	0	2025-12-20	ord_002	cust_001	rep_001	【{product_id："prod_101"，數量：10}】	150.00

狀態 (GSI-PK-1)	碎片 (GSI-PK-2)	order_date (SK)	order_id	customer_id	account_rep_id	order_items	total_amount
OPEN	1	2026-01-03	ord_005	cust_004	rep_002	【{product_id："prod_100"，數量：20}】	200.00
OPEN	2	2026-01-05	ord_003	cust_002	rep_001	【{product_id："prod_100"，數量：3}】	175.00

OrderByAccountRep GSI - 支援帳戶代表訂單查詢

account_rep_id (GSI-PK)	order_date (GSI-SK)	order_id	customer_id	status	total_amount
rep_001	2025-11-15	ord_001	cust_001	CLOSED	225.00
rep_001	2025-12-20	ord_002	cust_001	OPEN	150.00
rep_001	2026-01-05	ord_003	cust_002	OPEN	175.00
rep_002	2025-10-10	ord_004	cust_003	CLOSED	250.00
rep_002	2026-01-03	ord_005	cust_004	OPEN	200.00

ProductInOrders GSI - 支援產品訂單查詢

product_id (GSI-PK)	order_id (GSI-SK)	customer_id	order_date	數量
prod_100	ord_001	cust_001	2025-11-15	5
prod_100	ord_003	cust_002	2026-01-05	3
prod_101	ord_002	cust_001	2025-12-20	10

產品資料表設計

產品資料表使用項目集合模式，將產品中繼資料和庫存資料存放在相同的分割區中。此設計利用產品和庫存之間的識別關係 - 庫存不能在沒有父產品的情況下存在。使用 PK=product_id 搭配 SK=product_id 用於產品中繼資料和 SK=warehouse_id 用於清查項目，不需要單獨的清查資料表和 GSI，將成本降低約 50%。

此模式可有效查詢個別倉儲庫存 (GetItem 搭配複合索引鍵） 和產品的所有倉儲庫存 （分割區索引鍵上的查詢）。產品中繼資料項目中的 total_inventory 屬性提供非標準化彙總，以快速查詢總庫存。

產品資料表 - 基礎資料表結構 （項目集合模式）

product_id (PK)	warehouse_id (SK)	product_name	category	unit_price	inventory_quantity	total_inventory
prod_100	prod_100	小工具 A	硬體	25.00		500
prod_100	wh_sea				200	
prod_100	wh_pdx				150	
prod_100	wh_atl				150	
prod_101	prod_101	小工具 B	電子	50.00		300
prod_101	wh_sea				100	
prod_101	wh_pdx				200	

每個資料表都設計有特定的全域次要索引 (GSIs)，可有效支援所需的存取模式。此設計使用彙總導向原則搭配策略去標準化和稀疏索引，以最佳化效能和成本。

主要設計最佳化包括：

- Sparse GSI - OpenOrdersByDate 僅索引 OPEN 訂單 （總計 20%)，這有助於降低 GSI 儲存成本
- 項目收集模式 - 產品資料表使用 PK=product_id、SK=warehouse_id 存放庫存，以消除單獨的庫存資料表
- Order + OrderItems 彙總 - 由於 100% 存取關聯而內嵌為單一項目
- 策略非規範化 - 訂單表格中的 account_rep_id 重複，用於有效率的查詢

最後，您可以再次瀏覽先前定義的存取模式。下表顯示如何使用具有策略 GSIs多資料表設計，有效率地支援每個存取模式。每個模式都使用直接金鑰查詢或單一 GSI 查詢，避免昂貴的掃描，並在任何規模提供一致的效能。

序號	存取模式	查詢條件
1	按員工 ID 查詢員工詳細資訊	員工資料表：GetItem(employee_id="emp_001")
2	按員工姓名查詢員工詳細資訊	EmployeeByName GSI：Query(name="John Smith")
3	尋找員工的電話號碼 (s)	員工資料表：GetItem(employee_id="emp_001")
4	尋找客戶的電話號碼 (s)	客戶資料表：GetItem(customer_id="cust_001")
5	在日期範圍內為客戶取得訂單	OrderByCustomerDate GSI：Query(customer_id="cust_001"， order_date BETWEEN "2025-01-01" AND "2025-12-31")
6	顯示日期範圍內的所有開啟的訂單	OpenOrdersByDate GSI：平行查詢 5 個碎片與多屬性 PK (status="OPEN" + shard=0-4)、SK=order_date BETWEEN "2025-01-01" AND "2025-12-31"、合併結果
7	查看所有最近僱用的員工	EmployeeByHireDate GSI： Query(entity_type="EMPLOYEE"， hire_date >= "2025-01-01")

序號	存取模式	查詢條件
8	尋找 Warehouse 中的所有員工	EmployeeByWarehouse GSI：Query(warehouse_id="wh_sea")
9	取得產品訂單上的所有項目	ProductInOrders GSI：Query(product_id="prod_100")
10	在所有倉儲取得產品的庫存	產品資料表：Query(product_id="prod_100")
11	依帳戶代表取得客戶	CustomerByAccountRep GSI：Query(account_rep_id="rep_001")
12	依帳戶代表取得訂單	OrderByAccountRep GSI：Query(account_rep_id="rep_001")
13	取得具有任務標題的員工	EmployeeByJobTitle GSI：Query(job_title="Manager")
14	依產品和倉儲取得庫存	產品資料表：GetItem(product_id="prod_100"，warehouse_id="wh_sea")
15	取得產品庫存總計	產品資料表：GetItem(product_id="prod_100"，warehouse_id="prod_100")

從關聯式資料庫移轉到 DynamoDB

將關聯式資料庫移轉至 DynamoDB 需要謹慎規劃，才能確保實現成功的結果。本指南有助您了解此程序的運作方式、可用哪些工具，以及如何評估潛在移轉策略，並選取符合需求的策略。

主題

- 移轉至 DynamoDB 的原因
- 將關聯式資料庫移轉至 DynamoDB 時的考量事項
- 了解移轉至 DynamoDB 的運作方式
- 協助移轉至 DynamoDB 的工具
- 選擇移轉至 DynamoDB 的適當策略
- 執行 DynamoDB 離線移轉
- 執行 DynamoDB 混合移轉
- 透過 1:1 方式移轉每個資料表，執行 DynamoDB 線上移轉
- 使用自訂暫存資料表線上移轉至 DynamoDB

移轉至 DynamoDB 的原因

對企業和組織來說，移轉至 Amazon DynamoDB 有各種絕佳的優勢。以下是 DynamoDB 為理想資料庫移轉選項的一些主要優勢：

- 可擴展性：DynamoDB 專為處理大量工作負載及無縫擴展而設計，以容納不斷增長的資料磁碟區和流量。使用 DynamoDB，您可以根據需求輕鬆擴展或縮小資料庫，確保應用程式可以在不影響效能的情況下處理流量突增的情況。
- 效能：DynamoDB 提供低延遲的資料存取，讓應用程式能夠以卓越的速度擷取和處理資料。其分散式架構可確保讀取和寫入操作分散在多個節點，即使在高請求率下也能提供一致的個位數毫秒級回應時間。
- 全受管：DynamoDB 是由 AWS提供的全受管服務。這表示 會 AWS 處理資料庫管理的操作層面，包括佈建、組態、修補、備份和擴展。這可讓您更專注於開發應用程式，而非資料庫管理任務。
- 無伺服器架構：DynamoDB 支援 DynamoDB 隨需無伺服器模型，您只需為應用程式提出的實際讀取和寫入請求付費，無需預先佈建容量。這種按需付費的模式可確保成本效益和最低的營運開銷，因為您只為消耗的資源付費，而不需要佈建和監控容量。

- NoSQL 彈性：不同於傳統關聯式資料庫，DynamoDB 遵循 NoSQL 資料模型，提供結構描述設計的靈活性。DynamoDB 可讓您存放結構化、半結構化和非結構化的資料，因此非常適合處理多樣化和不斷演進的資料類型。這種靈活性可加快開發週期，更輕鬆因應不斷變化的業務需求。
- 高可用性與耐久性：DynamoDB 會跨區域內的多個可用區域複寫資料，以確保高可用性和資料耐久性。其可自動處理複寫、容錯移轉和復原，將資料遺失或服務中斷的風險降至最低。DynamoDB 提供高達 99.999% 的可用性 SLA。
- 安全性與合規：DynamoDB 整合 AWS Identity and Access Management 以進行精細存取控制。其提供靜態和傳輸中加密，可確保資料的安全性。DynamoDB 亦符合多項合規標準，包括 HIPAA、PCI DSS 與 GDPR，協助您符合法規要求。
- 與 AWS 生態系統整合：DynamoDB AWS 可無縫整合其他 AWS 服務 AWS Lambda，例如 CloudFormation和 AWS AppSync。 DynamoDB 此整合可讓您建置無伺服器架構、利用基礎設施即程式碼，以及建立即時資料驅動型應用程式。

將關聯式資料庫移轉至 DynamoDB 時的考量事項

關聯式資料庫系統和 NoSQL 資料庫各有優劣：這些差異導致兩種系統之間的資料庫設計不同。

任務類型	關聯式資料庫	NoSQL database (NoSQL 資料庫)
查詢資料庫	關聯式資料庫可以彈性地查詢資料，但查詢相對昂貴，而且在高流量的狀況下擴展不易 (請參閱 在 DynamoDB 中製作關聯式資料模型的第一步)。關聯式資料庫應用程式可能會在預存程序、SQL 子查詢、大量更新查詢和彙總查詢中實作商業邏輯。	在如 DynamoDB 這類的 NoSQL 資料庫中，可以少數方式有效地查詢資料，但在外部的查詢就非常昂貴而且速度緩慢。寫入 DynamoDB 是單例。先前在預存程序中執行的應用程式商業邏輯必須重構為在 DynamoDB 外部執行，並在 Amazon Amazon EC2 或 等主機上執行自訂程式碼 AWS Lambda。
設計資料庫	您可以專注於設計的靈活性，而不用擔心實作細節或效能。查詢最佳化通常不會影響結	您會特別設計結構描述，盡可能以最快、最便宜的方式進行常用與重要的查詢。系統會打

任務類型	關聯式資料庫	NoSQL database (NoSQL 資料庫)
	構描述設計，但標準化相當重要。	造您的資料結構，以符合企業使用案例的特定要求。

設計 NoSQL 資料庫時，您必須採用與設計關聯式資料庫管理系統 (RDBMS) 不同的思維。針對 RDBMS，您可以建立標準化的資料模型，而不需考量存取模式。您可以在稍後有新問題與查詢要求時擴展此模型。您可以將每種資料整理至其資料表。

使用 NoSQL 設計時，您可以針對 DynamoDB 需要回答的問題來設計結構描述。了解企業問題和應用程式讀取及寫入模式至關重要。此外，您的目標應該是盡可能在 DynamoDB 應用程式中維護越少資料表。擁有較少的資料表可讓事物更具擴展性，需要較少的許可管理，並減少 DynamoDB 應用程式的額外負荷。它還可以幫助降低整體備份成本。

我們會以個別主題來說明建立 DynamoDB 關聯式資料模型和建置新版本前端應用程式的任務。本指南假設您已建置使用 DynamoDB 的新版本應用程式，但仍需判斷如何以最佳方式在切換期間移轉和同步歷史資料。

規模調整考量因素

DynamoDB 資料表中存放的每個項目 (資料列) 大小上限為 400KB。如需詳細資訊，請參閱the section called “配額”。項目大小取決於項目中所有屬性名稱和屬性值的大小總計。如需詳細資訊，請參閱the section called “項目大小和格式”。

如果應用程式需要在項目中存放的資料超過 DynamoDB 大小限制，請將項目分成項目集合、壓縮項目資料，或以物件形式將項目存放在 Amazon Simple Storage Service (Amazon S3)，同時將 Amazon S3 物件識別碼存放在 DynamoDB 項目中。請參閱 the section called “大型項目”。更新項目的成本取決於項目的完整大小。若是需要頻繁更新現有項目的工作負載，更新 1 KB 或 2 KB 小項目的成本會更新大項目低。如需項目集合的詳細資訊，請參閱 the section called “處理項目收集”。

在您選擇分割區和排序索引鍵屬性、其他資料表設定、項目大小和結構，以及是否建立次要索引時，請務必檢閱 DynamoDB 建模文件以及 the section called “成本最佳化” 指南。請務必測試移轉計畫，確保 DynamoDB 解決方案具成本效益，並符合 DynamoDB 的功能和限制。

了解移轉至 DynamoDB 的運作方式

在檢閱可用的移轉工具之前，請思考 DynamoDB 處理寫入的方式。

預設和最常見的寫入操作是單一 PutItem API 操作。您可以在迴圈中執行 PutItem 操作來處理資料集。DynamoDB 支援幾乎無限的並行連線，因此假設您設定和執行大量多執行緒載入常式，例如 MapReduce 或 Spark，寫入速度只會受限於目標資料表的容量 (通常也是無限的)。

將資料載入 DynamoDB 時，請務必了解載入器的寫入速度。如果您要載入的項目 (資料列) 小於 1KB，則此速度單純為每秒的項目數。接著，可使用足夠的 WCU (寫入容量單位) 佈建目標資料表，以處理此速率。如果載入器在任何給定秒內超過佈建容量，則額外的請求可能會受到限流或完全遭拒。您可以在 CloudWatch 圖表中檢查限流情況，其位於 DynamoDB 主控台的監控索引標籤。

第二個可執行的操作是使用名稱為 BatchWriteItem 的相關 API。BatchWriteItem 可讓您將最多 25 個寫入請求合併為一個 API 呼叫。服務會接收這些呼叫，並以對資料表的個別 PutItem 請求形式處理。目前，當您選擇 時BatchWriteItem，在使用 進行單一呼叫時，您不會獲得軟體 AWS 開發套件中包含的自動重試的優點PutItem。因此，如果有任何錯誤 (例如限流例外狀況)，您必須在回應呼叫 BatchWriteItem 上尋找任何寫入失敗的清單。如需在 CloudWatch 限流圖表偵測到限流警告時處理限流警告的詳細資訊，請參閱 the section called “限流”。

第三種類型的資料匯入可透過從 S3 匯入 DynamoDB 功能進行。此功能可讓您在 Amazon S3 中暫存大型資料集，並要求 DynamoDB 自動將資料匯入新資料表。匯入不是即時完成，需要花費與資料集大小比例相應的時間。不過，此方法不需要 ETL 平台或自訂 DynamoDB 程式碼，因此很方便。DynamoDB 會將資料載入該次匯入建立的新資料表。目前，您無法將資料載入現有資料表。DynamoDB 會依原狀匯入資料，無任何轉換。類似於 PutItem，其必須使用上游程序，以您選擇的格式將資料寫入 Amazon S3 儲存貯體。

協助移轉至 DynamoDB 的工具

您可以使用數種常見的移轉和 ETL 工具，將資料移轉至 DynamoDB。

Amazon 提供許多可用於移轉的資料工具，包括 AWS Database Migration Service (DMS)、AWS Glue、Amazon EMR 和 Amazon Managed Streaming for Apache Kafka。這所有工具都可用來執行停機時間移轉，並利用關聯式資料庫的「變更資料擷取 (CDC)」功能來支援線上移轉。選擇工具時，請考慮組織擁有的技能組合和體驗，以及每個工具的功能、效能和成本。

許多客戶選擇撰寫自己的移轉指令碼和作業，以便為移轉程序建立自訂資料轉換。如果您打算執行具大量寫入流量或一般大量載入作業的大量 DynamoDB 資料表，建議您自行撰寫移轉程式碼，以更熟悉 DynamoDB 在大量寫入流量下的行為。執行實務移轉時，可在專案初期體驗限流處理和高效資料表佈建等案例。

選擇移轉至 DynamoDB 的適當策略

大型關聯式資料庫應用程式可能跨越數百個資料表，並支援數個不同的應用程式函數。處理大型移轉時，請考慮將應用程式分成較小的元件或微型服務，並一次移轉一小組資料表。然後，您可以將其他元件波次移轉至 DynamoDB。

選取移轉策略時，各種因素可能會引導您在不同解決方案之間抉擇。我們可以在決策樹中呈現這些選項，根據需求和資源來簡化可用的選項。以下簡要提及相關概念 (但指南稍後會更深入介紹)：

- 離線移轉：如果應用程式在移轉期間可以容忍一些停機時間，此方法可簡化移轉程序。
- 混合移轉：此方法允許移轉期間的部分運行時間，例如允許讀取但不寫入，或允許讀取和插入但不更新和刪除。
- 線上移轉：如果應用程式在移轉期間需要零停機時間，則較不容易移轉，且可能需要大量規劃和自訂開發。其中一個關鍵決策是估計和衡量建置自訂移轉程序的成本，以及在切換期間發生停機時間的業務成本。

If	及	THEN
您可以在維護時段關閉應用程式一段時間，以執行資料移轉。這是離線移轉。		使用 AWS DMS 並使用完全載入任務執行離線遷移。視需要使用 SQL `VIEW` 預先塑造來源資料。
您可以在移轉期間以唯讀模式執行應用程式。這是混合移轉。		停用應用程式或來源資料庫中的寫入。使用 AWS DMS 並使用完全載入任務執行離線遷移。
您可以在移轉期間執行應用程式的讀取和新記錄插入，但不更新或刪除。這是混合移轉。	您具備應用程式開發技能，可以更新現有的關聯式應用程式，以執行所有新記錄的雙重寫入，包括寫入至 DynamoDB	使用 AWS DMS 並使用完全載入任務執行離線遷移。同時，部署現有應用程式的版本，允許讀取和執行雙重寫入。
您需要停機時間最短的移轉。這是線上移轉。	• 您要將來源資料表一對一移轉至 DynamoDB，而不進行重大結構描述變更。	使用 AWS DMS 執行線上資料遷移。執行大量載入任務，接著執行 CDC 同步任務。

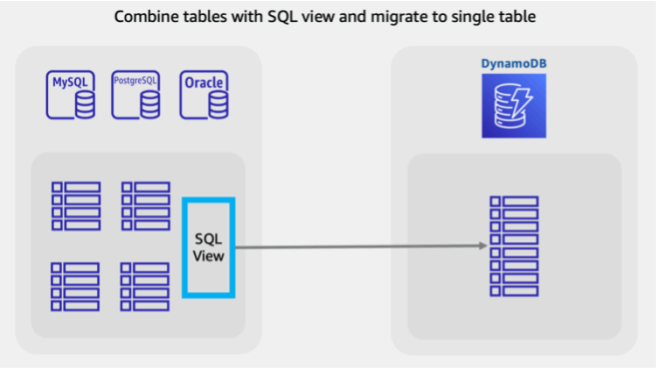
If	及	THEN
您需要停機時間最短的移轉。這是線上移轉。	• 您會依照堆疊結構描述或單一資料表原理，將來源資料表合併為較少的 DynamoDB 資料表。 • 您具備後端資料庫開發技能和 SQL 主機備用容量。	在 SQL 資料庫中建立 NoSQL 就緒資料表。使用 JOIN、UNION、VIEW、觸發條件、預存程序來填入和同步。
您需要停機時間最短的移轉。這是線上移轉。	• 您會依照單一資料表原理，將來源資料表合併為較少的 DynamoDB 資料表。例如： • 您不具備後端資料庫開發技能和 SQL 主機備用容量。	考慮混合或離線移轉方法。
您需要停機時間最短的移轉。這是線上移轉。	您可以略過移轉歷史交易資料，或將其封存在 Amazon S3 中以取代移轉。您只需要移轉幾個小型靜態資料表。	撰寫指令碼或使用任何 ETL 工具來移轉資料表。視需要使用 SQL `VIEW` 預先塑造來源資料。

執行 DynamoDB 離線移轉

如果可以允許停機時間來執行移轉，即適用離線移轉。關聯式資料庫每月至少要花一些停機時間進行維護和修補，也可能需要更長的停機時間來升級硬體或主要版本。

Amazon S3 可在移轉期間作為暫存區。若是以 CSV (逗號分隔值) 或 DynamoDB JSON 格式儲存的資料，可以使用從 S3 匯入 DynamoDB 功能，自動將其匯入新的 DynamoDB 資料表。

建議您結合資料表以運用唯一 NoSQL 存取模式 (例如，將四個舊版資料表轉換為單一 DynamoDB 資料表)。進行單一索引鍵值文件請求或預先分組項目集合的查詢，傳回時的延遲情況通常會比執行多資料表聯結的 SQL 資料庫更好。不過，這會使移轉任務更加困難。SQL 檢視可以在來源資料庫中執行工作，以準備單一資料集，在一個集合中表示所有四個資料表。



此檢視可以非標準化形式顯示 JOIN 資料表，也可以讓實體標準化，並使用 SQL UNION 堆疊資料表。此影片涵蓋了重塑關聯式資料的相關重要決策。若是離線移轉，絕佳方式是使用檢視來結合資料表，塑造適用於 DynamoDB 單一資料表結構描述的資料。

計劃

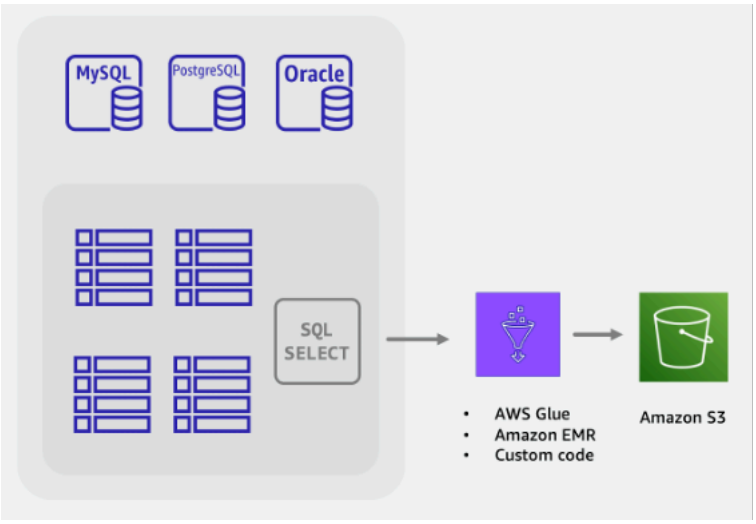
使用 Amazon S3 執行離線移轉

工具

- 擷取和轉換 SQL 資料並將其存放在 S3 儲存貯體的 ETL 作業，例如：
  - AWS Database Migration Service是一項服務，可大量載入歷史資料，亦可處理變更資料擷取 (CDC) 記錄，以同步來源和目標資料表。
  - AWS Glue
  - Amazon EMR
  - 自己的自訂程式碼
- 從 S3 匯入 DynamoDB 功能

離線移轉步驟：

1. 建置可查詢 SQL 資料庫、將資料表資料轉換為 DynamoDB JSON 或 CSV 格式，並將其儲存至 S3 儲存貯體的 ETL 作業。



2. 系統會調用從 S3 匯入 DynamoDB 功能，來建立新的資料表，並自動從 S3 儲存貯體載入資料。

完全離線移轉簡單明瞭，但應用程式擁有者和使用者可能不常用。如果應用程式可以在移轉期間提供較低的服務水準，而不是完全沒有服務，這可能對使用者較有利。

您可以新增功能，在離線移轉期間停用寫入，同時允許持續正常讀取。在移轉關聯式資料時，應用程式使用者仍可以安全地瀏覽及查詢現有資料。如果這是您要尋找的內容，請繼續閱讀了解混合移轉。

執行 DynamoDB 混合移轉

雖然所有資料庫應用程式都會執行讀取和寫入操作，但在規劃混合或線上移轉時應考慮要執行的寫入操作類型。資料庫寫入可以分為插入、更新和刪除三種儲存貯體。有些應用程式可能不需要立即處理刪除。舉例來說，這些應用程式可以將刪除延遲到月底進行大量清除程序。您可以更輕鬆移轉這類應用程式，同時允許部分運行時間。

計劃

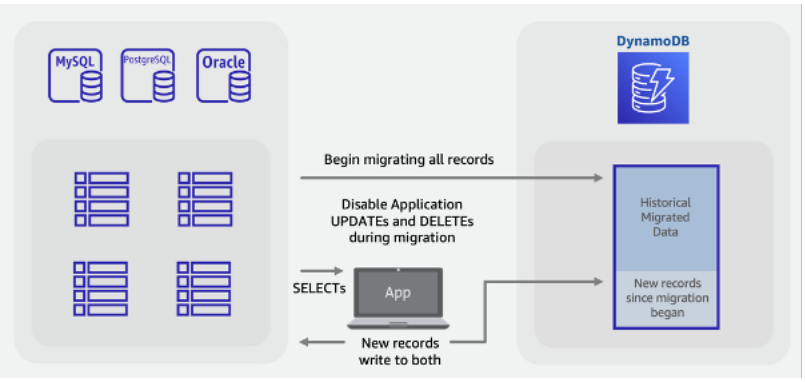
使用應用程式雙重寫入執行混合式線上/離線移轉

工具

- 擷取和轉換 SQL 資料並將其存放在 S3 儲存貯體的 ETL 作業，例如：
  - AWS DMS
  - AWS Glue
  - Amazon EMR
  - 自己的自訂程式碼

混合移轉步驟：

1. 建立目標 DynamoDB 資料表。此資料表會同時收到歷史大量資料和新的即時資料
2. 建立舊版應用程式，停用刪除和更新，同時以雙重寫入 SQL 資料庫和 DynamoDB 形式執行所有插入
3. 開始 ETL 任務或 AWS DMS 任務，以回填現有資料並同時部署新的應用程式版本
4. 當回填作業完成時，DynamoDB 即具備所有現有和新的記錄，並就緒可進行應用程式切換



> **Note**
>
> 回填作業會直接從 SQL 寫入 DynamoDB。我們無法如離線移轉範例一樣使用 S3 匯入功能，因為該功能建立的新資料表要等 DynamoDB 載入資料之後才是即時狀態。

透過 1:1 方式移轉每個資料表，執行 DynamoDB 線上移轉

許多關聯式資料庫具有名稱為「變更資料擷取 (CDC)」的功能，其中資料庫可讓使用者請求在特定時間點之前或之後發生的資料表變更清單。CDC 使用內部日誌來啟用此功能，且資料表不需要任何時間戳記欄即可運作。

將 SQL 資料表的結構描述移轉至 NoSQL 資料庫時，建議您將資料合併並重新塑造至較少的資料表。這樣做可讓您在單一位置收集資料，並避免在多步驟讀取操作中手動聯結相關資料。不過，您不一定要塑造單一資料表的資料，有時可以將資料表一對一移轉至 DynamoDB。這些 1:1 資料表移轉較不複雜，因為您可以使用支援這種移轉類型的常見 ETL 工具，利用來源資料庫 CDC 功能。每一個資料列的資料仍可以轉換為新格式，但每個資料表的範圍保持不變。

請考慮將 SQL 資料表 1:1 移轉至 DynamoDB，請注意 DynamoDB 不支援伺服器端聯結。您需要將邏輯新增至應用程式，才能合併來自多個資料表的資料。

計劃

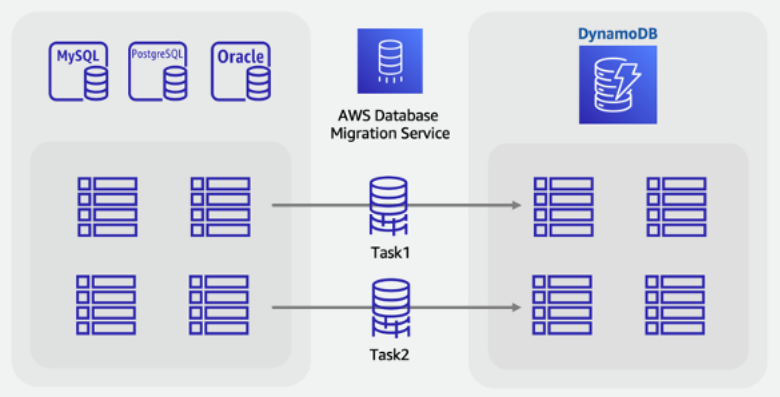
使用 執行將每個資料表線上遷移至 DynamoDB AWS DMS

工具

- AWS DMS (DMS)

線上移轉步驟：

1. 識別來源結構描述中要移轉的資料表
2. 使用與來源相同的索引鍵結構，在 DynamoDB 中建立相同數量的資料表
3. 在 中建立複寫伺服器， AWS DMS 並設定來源和目標端點
4. 定義每個資料列所需的轉換 (例如串連資料欄或將日期轉換為 ISO-8601 字串格式)
5. 為每個資料表建立適用於完全載入和變更資料擷取的移轉任務
6. 監控這些任務，直到進行中複寫階段開始為止
7. 此時，您可以執行任何驗證稽核，然後將使用者切換至讀取和寫入 DynamoDB 的應用程式



使用自訂暫存資料表線上移轉至 DynamoDB

如同上述離線移轉案例，您可以選擇結合資料表以運用唯一 NoSQL 存取模式 (例如，將四個舊版資料表轉換為單一 DynamoDB 資料表)。SQL `VIEW` 可以在來源資料庫中執行工作，以準備單一資料集，在一個集合中表示所有四個資料表。

不過，若是具有即時變更資料的線上移轉，您無法利用 CDC 功能，因為 `VIEW` 不支援這些功能。如果您的資料表包含上次更新的時間戳記欄，且這些資料表已納入 `VIEW` 中，則您可以建置自訂 ETL 作業，以使用這些作業及同步來達成大量載入。

因應此挑戰的創新方法是使用標準 SQL 功能，例如檢視、預存程序和觸發條件，以建立最終所要的 DynamoDB NoSQL 格式新 SQL 資料表。

如果您的資料庫伺服器有備用容量，可以在移轉開始之前建立此單一暫存資料表。您可以撰寫預存程序來讀取現有資料表、視需要轉換資料以及寫入新的暫存資料表。您可以新增一組觸發條件，即時將資料表中的變更複寫到暫存資料表。如果公司政策不允許觸發條件，您也可以變更預存程序以達到相同結果。您可以新增幾行程式碼至任何寫入資料的程序，額外將相同變更寫入暫存資料表。

將此暫存資料表與舊版應用程式資料表完全同步，就是即時移轉的理想起點。現在可以針對此資料表使用使用資料庫 CDC 來完成即時遷移的工具 AWS DMS，例如 。此方法的優點是，其使用關聯式資料庫引擎常見的 SQL 技能和功能。

計劃

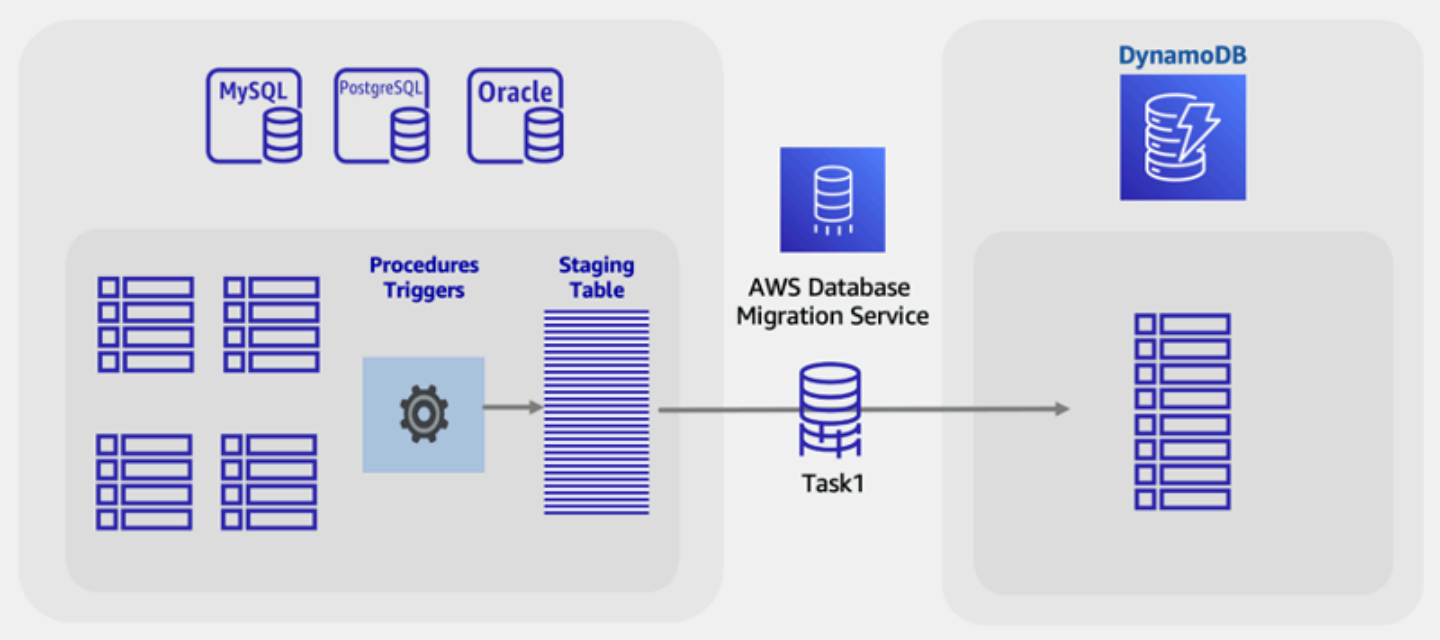
使用 SQL 預備資料表執行線上遷移 AWS DMS

工具

- 自訂 SQL 預存程序或觸發條件
- AWS DMS

線上移轉步驟：

1. 確認來源關聯式資料庫引擎中有額外的磁碟空間和處理容量。
2. 在 SQL 資料庫中建立新的暫存資料表，並啟用時間戳記或 CDC 功能
3. 寫入並執行預存程序，將現有的關聯式資料表資料複製到暫存資料表
4. 部署觸發條件或修改現有程序，在執行現有資料表正常寫入同時，雙重寫入至新的暫存資料表
5. 執行 AWS DMS 以遷移此來源資料表並將其同步至目標 DynamoDB 資料表



本指南提供若干考量事項及方法，可將關聯式資料庫資料移轉至 DynamoDB，並把重點放在使用常見的資料庫工具和技術，將停機時間降至最低。如需詳細資訊，請參閱下列內容：

- AWS DMS 使用者指南
- AWS Glue 使用者指南
- 從 RDBMS 移轉至 DynamoDB 的最佳實務

DynamoDB 專用 NoSQL Workbench

Amazon DynamoDB 專用 NoSQL Workbench 是用於現代資料庫開發和操作的跨平台用戶端 GUI 應用程式。適用於 Windows、macOS 和 Linux。NoSQL Workbench 可讓您設計 DynamoDB 資料模型、將存取模式定義為實際的 DynamoDB 操作，並使用範例資料進行驗證。此外，您可以將資料模型組織到專案中。NoSQL Workbench 包含 DynamoDB 本機，可讓您先測試資料表和索引，再將資料模型遞交至雲端。若要深入了解 DynamoDB 本機版及其要求，請參閱 設定 DynamoDB 本機版 (可下載版本)。

資料模型建立工具

使用適用於 DynamoDB 的 NoSQL Workbench，您可以從頭開始新的專案，或使用符合您使用案例的範例專案。然後，您可以設計資料表和全域次要索引、定義屬性和設定範例資料。您也可以將存取模式視覺化為實際的 DynamoDB 操作 (PutItem、UpdateItem、Query 等），並根據設定的範例資料執行這些操作，以驗證存取模式是否如預期運作，並根據驗證結果調整資料模型。最後，經過驗證後，您會將模型遞交至 DynamoDB 本機或 AWS 您的帳戶，以供進一步測試和生產使用。對於協同合作，您可以匯入和匯出設計的資料模型。如需詳細資訊，請參閱使用 NoSQL Workbench 建立資料模型。

操作建置器

NoSQL Workbench 提供強大的圖形使用者介面供您開發及測試查詢。您可以使用 operation builder (操作建置器) 來檢視、探索及查詢即時資料集。結構化操作建置器支援投影表達式、條件表達式，並以多種語言產生範例程式碼。您可以將資料表從一個 Amazon DynamoDB 帳戶直接複製到不同區域中的另一個 Amazon DynamoDB 帳戶。您也可以在 DynamoDB 本機版和 Amazon DynamoDB 帳戶之間直接複製多個資料表，以便在開發環境之間更快速地複製資料表的索引鍵結構描述 (並可選擇 GSI 結構描述和項目)。如需詳細資訊，請參閱使用 NoSQL Workbench 探索資料集與建立操作。

使用 NoSQL Workbench 進行資料建模的概念，由以下影片詳細說明。

主題

- 下載 DynamoDB 專用 NoSQL Workbench
- 使用 NoSQL Workbench 建立資料模型
- 使用 NoSQL Workbench 探索資料集與建立操作
- NoSQL Workbench 的範例資料模型

- NoSQL Workbench 的版本歷史記錄

下載 DynamoDB 專用 NoSQL Workbench

請依照這些指示下載適用於 Amazon DynamoDB 的 NoSQL Workbench 和 DynamoDB 本機。

若要下載 NoSQL Workbench 和 DynamoDB 本機版

- 下載適用您作業系統的 NoSQL Workbench。

作業系統	下載連結
macOS (Intel)	適用於 macOS 的下載 (Intel)
macOS (Apple 晶片)	適用於 macOS 的下載 (Apple 晶片)
Windows	適用於 Windows 的下載
Linux	下載 Linux 版本

> **Note**
>
> NoSQL Workbench 包含 DynamoDB local 做為安裝程序的一部分。
> 執行 DynamoDB 本機時，需要 Java Runtime Environment (JRE) 17.x 版或更新版本。

> **Note**
>
> NoSQL Workbench 支援 Ubuntu 12.04、Fedora 21 和 Debian 8，或這些 Linux 發行版本的任何更新版本。
> Ubuntu 安裝需要兩個必要軟體： `libfuse2`和 `curl`。
> Ubuntu 22.04 不再預設安裝 libfuse2。若要解決此問題，請針對最新的 Ubuntu 版本執行 `sudo add-apt-repository universe && sudo apt install libfuse2`以安裝 。
> 針對 cURL，執行 `sudo apt update && sudo apt upgrade && sudo apt install curl`。

使用 NoSQL Workbench 建立資料模型

您可以使用適用於 Amazon DynamoDB 的 NoSQL Workbench 中的資料模型工具來建立新的資料模型，或根據滿足應用程式資料存取模式的現有資料模型來設計模型。資料模型建立工具包含幾個協助您開始的範例資料模型。

主題

- 建立新的資料模型
- 匯入現有的資料模型
- 編輯現有的資料模型
- 將範例資料新增至資料模型
- 新增和驗證存取模式
- 從 CSV 檔案匯入範例資料
- 面向
- 使用彙總檢視在資料模型中檢視所有資料表
- 匯出資料模型
- 向 DynamoDB 遞交資料模型

建立新的資料模型

使用 NoSQL Workbench，在 Amazon DynamoDB 中依照這些步驟建立新的資料模型。

建立新的資料模型

1. 開啟 NoSQL Workbench，然後在主畫面上，選取手動建立模型。

   將開啟新頁面，其中包含第一個資料表的空白組態。NoSQL Workbench 會使用預設名稱（即 untitled-2) 建立所有新的資料模型，並將其新增至 Drafts 專案資料夾。
2. 在資料表組態畫面上，指定下列項目：

   - Table Name (資料表名稱)：輸入資料表的唯一名稱。
   - 分割區索引鍵 — 輸入分割區索引鍵名稱，並指定其類型。或者，您也可以選擇更精細的資料類型格式來產生範例資料。
   - 如果您想要新增排序索引鍵，請指定排序索引鍵名稱及其類型。或者，您可以選擇更精細的資料類型格式來產生範例資料。

> **Note**
>
> 若要深入了解主索引鍵設計、設計、有效使用分割區索引鍵以及使用排序索引鍵，請參閱以下內容：
>
> - 主索引鍵
> - 在 DynamoDB 中設計與有效運用分割區索引鍵的最佳實務
> - 使用排序索引鍵組織 DynamoDB 資料的最佳實務

3. 您可以新增其他屬性，以更清楚地驗證模型和存取模式。若要新增其他屬性：

   - 選擇新增屬性。
   - 指定屬性名稱和其類型。
   - 或者，您可以選擇更精細的資料類型格式來產生範例資料。

4. 若要新增全域次要索引，請選擇 Add global secondary index (新增全域次要索引)。指定 Global secondary index name (全域次要索引名稱)、Partition key (分割區索引鍵) 屬性和 Projection type (投射類型)。

   如需在 DynamoDB 中使用全域次要索引的詳細資訊，請參閱全域次要索引。

5. 或者，新增面向。面向是 NoSQL Workbench 中的虛擬建構模組。它不是 DynamoDB 中的功能建構。NoSQL Workbench 中的面向可協助您視覺化應用程式 DynamoDB 的不同資料存取模式，其中只有資料表中的資料子集。

   > **Note**
   >
   > 我們建議您使用 新增和驗證存取模式 視覺化您的應用程式如何存取 DynamoDB 中的資料，而不是 Facets。存取模式會反映您的實際資料庫互動，並協助您為使用案例建置正確的資料模型，而面向是非功能視覺化。

   選擇 Add facet (新增面向)。指定下列內容：

   - Facet name (面向名稱)。
   - 分割區索引鍵別名有助於分辨此面向視圖。
   - 如果您為資料表提供排序索引鍵，則為排序索引鍵別名。

- 選擇屬於此面向的屬性。

若您想新增更多面向，請重複執行此步驟。

6. 最後，按一下儲存按鈕來建立資料表。
7. 如果您需要其他資料表或全域次要索引，請按一下您剛建立之資料表上方的 + 圖示。

匯入現有的資料模型

您可以使用 Amazon DynamoDB 專用 NoSQL Workbench，藉由匯入與修改現有的模型來建立資料模型。您能夠以 NoSQL Workbench 模型格式或 AWS CloudFormation JSON 範本格式匯入資料模型。

匯入資料模型

1. 開啟 NoSQL Workbench，然後在主畫面上選取匯入模型。您可以匯入 NoSQL Workbench 模型格式或 CloudFormation JSON 範本。
2. 您可以將匯入的模型新增至現有專案，或建立新的專案。如果您未選擇不同的專案，您的模型會新增至草稿專案資料夾。
3. 按一下瀏覽桌面，然後選擇要從電腦匯入的模型。
4. Once your model is imported, you can add access patterns, edit tables/indexes, and modify the data locally.

編輯現有的資料模型

編輯現有的使用者

1. 若要開始變更現有模型，請在模型建構器頁面中開啟模型。
2. 在資源選取器面板中，您會看到資料表和全域次要索引的清單。
3. 若要編輯資料表或全域次要索引，請先按一下資源選擇器面板中的名稱，然後使用頂端的動作圖示。可用的動作為刪除、複製和編輯。
4. 如果您想要編輯模型詳細資訊，請按一下模型名稱旁的三點圖示。
5. 從那裡，您可以按一下編輯模型詳細資訊並相應地編輯資訊。
6. 您也可以從相同的選單複製、移動、刪除和匯出模型。
7. 若要變更為另一個模型，您可以再次瀏覽主畫面，或使用模型選取下拉式清單。

將範例資料新增至資料模型

自動產生範例資料

1. 開啟 NoSQL Workbench，然後在主畫面上，按一下您要新增範例資料的模型名稱。
2. 按一下主要內容工具列中的其他動作（三點圖示），然後選取新增範例資料。
3. 輸入您要產生的範例資料項目數量，然後選取確認。
4. 自動產生範例資料可協助您產生 1 到 5000 列的資料，以供立即使用。您可以指定精細的範例資料類型，以根據您的設計和測試需求建立真實的資料。若要利用此功能產生真實的資料，您需要在資料模型建立工具中指定屬性的範例資料類型格式。如需指定範例資料類型格式建立新的資料模型，請參閱 。

若要一次新增一個項目的範例資料

1. 開啟您要編輯的模型，然後選擇您要新增範例資料的資料表。按一下主要內容工具列中的其他動作（三點圖示），然後選取編輯資料。
2. 現在您可以新增、刪除和編輯資料列。進行必要的變更之後，請按一下儲存按鈕。

新增和驗證存取模式

您可以使用適用於 Amazon DynamoDB 的 NoSQL Workbench 來建立、存放和驗證存取模式。

> **Note**
>
> 如需識別正確存取模式的詳細資訊，請參閱識別資料存取模式。

建立存取模式

1. 開啟 NoSQL Workbench，然後在主畫面上，按一下您要新增存取模式的模型名稱。
2. 在左側，選擇存取模式索引標籤，然後按一下 + 圖示。
3. 在下一個畫面上，提供名稱、選用的描述、存取模式的類型，以及要測試存取模式的資料表或全域次要索引。

> **Note**
>
> NoSQL Workbench 目前支援下列存取模式的操作：`Scan`、`Query`、`GetItem`、`PutItem`、`UpdateItem`、`DeleteItem`。Amazon DynamoDB 支援更廣泛的操作清單。

4. 建立存取模式之後，您可以切換到驗證索引標籤，以驗證您的資料模型是否設計為傳回存取模式的預期結果。[將範例資料新增至資料模型] 如需如何自動產生資料表範例資料的詳細資訊，請參閱 。不同類型的存取模式將支援不同的輸入參數。

> **Note**
>
> 為了驗證存取模式，NoSQL Workbench 會在連接埠 `8001`（預設） 上啟動單獨的 DynamoDB 本機資料庫，其中包含儲存在記憶體中的資料表和索引。
>
> - NoSQL Workbench 會自動將模型中的範例資料新增至暫存資料表。
> - 如果您編輯範例資料或資料模型本身，NoSQL Workbench 會更新暫時資料表。
> - 當您關閉應用程式時，會清除此暫時資料庫。

編輯您的存取模式

1. 開啟 NoSQL Workbench，然後在主畫面上，按一下您要編輯存取模式的模型名稱。
2. 在左側，選擇存取模式索引標籤。
3. 若要編輯存取模式，請從左側的清單中選擇。
4. 在頂端列中，按一下編輯動作按鈕。

從 CSV 檔案匯入範例資料

如果您有 CSV 檔案預先存在範例資料，您可以將其匯入 NoSQL Workbench。這使您可以快速填入模型取樣資料，而無需逐行輸入。

CSV 檔案中的欄名稱必須與資料模型中的屬性名稱相符，但不需要以相同的順序排列。例如，如果您的資料模型具有名為 `LoginAlias`、`FirstName`，以及 `LastName`，您的 CSV 列可以是 `LastName`、`FirstName`，以及 `LoginAlias`。

您一次最多可從 CSV 檔案匯入 150 個資料列。

若要從 CSV 檔案匯入資料至 NoSQL Workbench

1. 若要將 CSV 資料匯入資料表，請先按一下資源面板中的資料表名稱，然後按一下主要內容工具列中的其他動作 （三點圖示）。
2. 選取匯入範例資料。
3. 選取 CSV 檔案，然後選擇 Open (開啟)。CSV 資料會附加到您的資料表。

面向

在 NoSQL Workbench 中，構面可讓您檢視資料表中資料的子集，而無需查看不符合構面限制的記錄。面向是一種視覺資料建模工具，在 DynamoDB 中不作為可用的建構存在，因為其純粹用於輔助存取模式建模。

> **Note**
>
> 我們建議您使用 新增和驗證存取模式 視覺化您的應用程式如何存取 DynamoDB 中的資料，而不是 Facets。存取模式會反映您的實際資料庫互動，並協助您為使用案例建置正確的資料模型，而面向是非功能視覺化。

建立面向

1. 在資源選取器面板中，選擇您要編輯的資料表
2. 在頂端列中，按一下編輯動作圖示。
3. 向下捲動至面向篩選條件區段。
4. 選擇 Add facet (新增面向)。指定下列內容：

   - Facet name (面向名稱)。
   - 分割區索引鍵別名有助於分辨此面向視圖。
   - 如果您為資料表提供排序索引鍵，則為排序索引鍵別名。
   - 選擇屬於此面向的屬性。

   若您想新增更多面向，請重複執行此步驟。

使用彙總檢視在資料模型中檢視所有資料表

NoSQL Workbench for Amazon DynamoDB 中的彙總檢視可讓您視覺化資料模型中的所有資料表和索引。每份資料表都會顯示以下資訊：

- 資料表欄位名稱
- 範例資料
- 所有與資料表相關聯的全域次要索引。NoSQL Workbench 會顯示每個索引的下列資訊：
  - 索引欄位名稱
  - 範例資料

檢視所有資料表資訊

1. 開啟 NoSQL Workbench，然後在主畫面上，按一下您要開啟的模型名稱。
2. 在頂端列中，按一下彙總檢視。您會在同一個畫面上看到所有資料表和索引的資料。

將彙總檢視匯出為影像

1. 選取彙總檢視後，按一下三點圖示，然後選擇匯出彙總檢視
2. 所有資料表和索引的 PNG 映像封存將呈現以供下載。

匯出資料模型

在使用 Amazon DynamoDB 專用 NoSQL Workbench 建立資料模型後，即可以 NoSQL Workbench 模型格式或 AWS CloudFormation JSON 範本格式來儲存和匯出該模型。

匯出資料模型

1. 開啟 NoSQL Workbench，然後在主畫面上，按一下您要編輯的模型名稱。
2. 按一下資料模型名稱旁的三點圖示，然後選取匯出模型。
3. 選擇是否以 NoSQL Workbench 模型格式或 CloudFormation JSON 範本格式匯出資料模型。

   - 如果您想要使用 NoSQL Workbench 與其他團隊成員共用模型，或稍後將其匯入 NoSQL Workbench，請選擇 NoSQL Workbench 模型格式。
   - 如果您想要將模型直接部署到 ， AWS 或將其整合到您的infrastructure-as-code工作流程，請選擇 CloudFormation JSON 範本格式。

向 DynamoDB 遞交資料模型

當您滿意資料模型後，即可向 Amazon DynamoDB 遞交模型。

> **Note**
>
> - 此動作會在 中 AWS 為資料模型中呈現的資料表和全域次要索引建立伺服器端資源。
> - NoSQL Workbench 預設會使用隨需容量建立資料表和索引。

向 DynamoDB 遞交資料模型

1. 開啟 NoSQL Workbench，然後在主畫面上，按一下您要遞交的模型名稱。
2. 在頂端列中，按一下遞交。
3. 選擇現有的連線，或按一下新增連線按鈕來建立新的連線。
   - 指定下列資訊，新增新的連線：
     - 帳戶別名
     - AWS 區域
     - 存取金鑰 ID
     - 私密存取金鑰

     如需如何取得存取金鑰的詳細資訊，請參閱取得 AWS 存取金鑰。
   - 您可以選擇指定下列項目：
     - Session token (工作階段字符)
     - IAM 角色 ARN
4. 如果您偏好使用 DynamoDB 本機：
   1. 選擇本機連線索引標籤。
   2. 按一下新增連線按鈕。
   3. 指定 Connection name (連線名稱) 和 Port (連接埠)。

> **Note**
>
> 若要使用 DynamoDB 本機，您需要使用 NoSQL Workbench 畫面左下角的 DynamoDB 本機切換來開啟它。

5. 按一下遞交。

使用 NoSQL Workbench 探索資料集與建立操作

Amazon DynamoDB 專用 NoSQL Workbench 會提供強大的圖形使用者介面來開發及測試查詢。您可以使用 NoSQL Workbench 中的操作建置器來檢視、探索及查詢即時資料集。結構化操作建置器支援投影表達式、條件表達式，並以多種語言產生範例程式碼。您可以將資料表從一個 Amazon DynamoDB 帳戶直接複製到不同區域中的另一個 Amazon DynamoDB 帳戶。您也可以在 DynamoDB 本機版和 Amazon DynamoDB 帳戶之間直接複製資料表，以便在開發環境之間更快速地複製資料表的索引鍵結構描述 (並可選擇 GSI 結構描述和項目)。您可以在操作建置器中儲存多達 50 個 DynamoDB 資料操作。

主題

- 連線至即時資料集
- 建置複雜的操作
- 使用 NoSQL Workbench 複製資料表
- 將資料匯出到 CSV 檔案

連線至即時資料集

若要使用 NoSQL Workbench 連線到 Amazon DynamoDB 資料表，您必須先連線到 AWS 您的帳戶。

新增資料庫連線

1. 在 NoSQL Workbench 的左側導覽窗格中，選擇 Operation builder (操作建置器) 圖示。
2. 選擇 Add Connection (新增連線)。
3. 指定下列資訊：
   - 連線名稱 (Connection name)

- AWS 區域
- 存取金鑰 ID
- 私密存取金鑰

如需如何取得存取金鑰的詳細資訊，請參閱取得 AWS 存取金鑰。

您可以選擇指定下列項目：

- Session token (工作階段字符)
- IAM 角色 ARN

4. 選擇 Connect (連線)。

   如果您不想要註冊免費方案帳戶，且寧可使用 DynamoDB 本機版 (可下載版本)：

   a. 在連線畫面上選擇 Local (本機) 標籤。

   b. 指定下列資訊：

      - 連線名稱 (Connection name)
      - 連接埠

   c. 選擇 connect (連接) 按鈕。

   > **Note**
   >
   > 若要連線至 DynamoDB 本機版，請使用終端機手動啟動 DynamoDB 本機版 (請參閱在電腦上部署 DynamoDB 本機版)，或使用 NoSQL Workbench 導覽選單中的 DDB 本機版切換以直接啟動 DynamoDB 本機版。確保連線連接埠與您的 DynamoDB 本機版連接埠相同。

5. 對已建立的連線選擇 Open (開啟)。

連線至 DynamoDB 資料庫後，左窗格即會顯示可用的資料表清單。選擇資料表之一會傳回資料表所存放資料的範例。

您現在可以針對選取的資料表執行臨機操作查詢。

若要在資料表上執行查詢，請參閱有關建置操作的下一節，請參閱 建置複雜的操作。

建置複雜的操作

Amazon DynamoDB 專用 NoSQL Workbench 中的操作建置器提供視覺化介面，可讓您在其中執行複雜的資料平面操作。它包括對投射表達式和條件表達示的支援。建立操作後，您可以儲存操作以供日後使用 (最多可儲存 50 個操作)。然後，您可以在 Saved Operations (已儲存操作) 選單中瀏覽常用的資料平面操作清單，再使用它們來自動填入和建置新操作。您也可以使用多種語言產生這些操作的範本程式碼。

NoSQL Workbench 支援建置 DynamoDB 專用 PartiQL 陳述式，可讓您使用 SQL 相容查詢語言來與 DynamoDB 互動。NoSQL Workbench 也支援建置 DynamoDB CRUD API 操作。

若要使用 NoSQL Workbench 來建立操作，請在左側導覽窗格中選擇 Operation builder (操作建置器) 圖示。

主題

- 建置 PartiQL 陳述式
- 建置 API 操作

建置 PartiQL 陳述式

若要使用 NoSQL Workbench 來建置 DynamoDB 專用 PartiQL 陳述式，請選擇位於 NoSQL Workbench UI 頂端的 PartiQL 編輯器。

您可以在操作建置器中建立以下 PartiQL 陳述式類型。

主題

- 單一陳述式
- 交易
- 批次

單一陳述式

若要執行或產生 PartiQL 陳述式的程式碼，請執行以下操作。

1. 選擇視窗頂端附近的 PartiQL 編輯器。
2. 輸入有效的 PartiQL statement (PartiQL 陳述式)。
3. 如果您的陳述式使用參數：

a. 選擇 Optional request parameters (選用的請求參數)。
b. 選擇 Add new parameters (新增新參數)。
c. 輸入屬性類型和值。
d. 如果希望新增其他參數，請重複步驟 b 和 c。

4. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

   從顯示的標籤中選取所需的語言。您現在可以複製此程式碼，並使用在您的應用程式中。

5. 若希望立即執行此操作，請選擇 Run (執行)。
6. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)。然後輸入操作的名稱，並選擇 Save (儲存)。

交易

若要執行或產生 PartiQL 交易的程式碼，請執行以下操作。

1. 從更多操作下拉式清單中選擇 PartiQLTransaction。
2. 選擇 Add a new statement (新增新陳述式)。
3. 輸入有效的 PartiQL statement (PartiQL 陳述式)。

> **Note**
>
> 同一個 PartiQL 交易請求中不能同時支援讀取和寫入操作。SELECT 陳述式不能與 INSERT、UPDATE 和 DELETE 陳述式存在於同一個請求中。如需詳細資訊，請參閱使用 DynamoDB 專用 PartiQL 執行交易。

4. 如果您的陳述式使用參數

   a. 選擇 Optional request parameters (選用的請求參數)。
   b. 選擇 Add new parameters (新增新參數)。
   c. 輸入屬性類型和值。
   d. 如果希望新增其他參數，請重複步驟 b 和 c。

5. 如果希望新增更多陳述式，請重複步驟 2 至 4。
6. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

   從顯示的標籤中選取所需的語言。您現在可以複製此程式碼，並使用在您的應用程式中。

7. 若希望立即執行此操作，請選擇 Run (執行)。
8. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)。然後輸入操作的名稱，並選擇 Save (儲存)。

批次

若要執行或產生 PartiQL 批次的程式碼，請執行以下操作。

1. 從更多操作下拉式清單中選擇 PartiQLBatch。
2. 選擇 Add a new statement (新增新陳述式)。
3. 輸入有效的 PartiQL statement (PartiQL 陳述式)。

> **Note**
>
> 同一個 PartiQL 批次請求中不能同時支援讀取和寫入操作，也就是說 SELECT 陳述式不能與 INSERT、UPDATE 和 DELETE 陳述式存在於同一個請求中。不允許對相同項目執行寫入操作。與 BatchGetItem 操作一樣，僅支援單一讀取操作。不支援掃描和查詢操作。如需詳細資訊，請參閱使用 DynamoDB 專用 PartiQL 執行批次操作。

4. 如果您的陳述式使用參數：
   a. 選擇 Optional request parameters (選用的請求參數)。
   b. 選擇 Add new parameters (新增新參數)。
   c. 輸入屬性類型和值。
   d. 如果希望新增其他參數，請重複步驟 b 和 c。
5. 如果希望新增更多陳述式，請重複步驟 2 至 4。
6. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

   從顯示的標籤中選取所需的語言。您現在可以複製此程式碼，並使用在您的應用程式中。
7. 若希望立即執行此操作，請選擇 Run (執行)。
8. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)。然後輸入操作的名稱，並選擇 Save (儲存)。

建置 API 操作

若要使用 NoSQL Workbench 來建立 DynamoDB CRUD API，請選取位於 NoSQL Workbench 左側的操作建置器。

選取開啟，然後選擇連線。

您可以在操作建置器中執行以下操作。

- 刪除資料表
- 建立資料表
- 更新資料表

- 放置項目
- 更新項目
- 刪除項目
- 查詢
- 掃描
- 交易取得項目
- 交易寫入項目

刪除資料表

若要執行 `Delete Table` 操作，請執行下列動作。

1. 從資料表區段尋找要刪除的資料表。
2. 從水平省略符號功能表中，選取刪除資料表。
3. 輸入資料表名稱，確認您想要刪除資料表。
4. 選取刪除。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的刪除資料表。

刪除 GSI

若要執行 `Delete GSI` 操作，請執行下列動作。

1. 從資料表區段尋找要刪除之資料表的 GSI。
2. 從水平省略號功能表中選取刪除 GSI。
3. 輸入 GSI 名稱，確認您想要刪除該 GSI。
4. 選取刪除。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的刪除資料表。

建立資料表

若要執行 `Create Table` 操作，請執行下列動作。

1. 選擇資料表區段旁的 + 圖示。
2. 輸入所需的資料表名稱。
3. 建立分割區索引鍵。
4. 選用：建立排序索引鍵。
5. 自定義容量設定並取消勾選使用預設容量設定核取方塊。
   - 您現在可以選取 Provisioned (已佈建的) 或 On-demand capacity (隨需容量)。

     若選取佈建，您可以設定最小和最大讀寫容量單位。您還可以啟用或停用自動擴展。
   - 如果資料表目前設定為隨需，您將無法指定佈建的輸送量。
   - 如果您從隨需切換到佈建輸送量，則自動擴展將自動套用到所有 GSI，其中包含：最小值：1，最大值：10；目標：70%。
6. 選取略過 GSI 並建立，以建立沒有 GSI 的資料表。或者，您可以選擇下一步，使用此新資料表建立 GSI。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的建立資料表。

建立 GSI

若要執行 `Create GSI` 操作，請執行下列動作。

1. 尋找您要新增 GSI 的資料表。
2. 從水平省略號功能表中，選取建立 GSI。
3. 在索引名稱下，命名您的 GSI。
4. 建立分割區索引鍵。

5. 選用：建立排序索引鍵。
6. 從下拉式清單中選擇投影類型選項。
7. 選擇建立 GSI。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的建立資料表。

更新資料表

若要使用 `Update Table` 操作更新資料表的容量設定，請執行下列動作。

1. 尋找待更新容量設定的資料表。
2. 從水平省略號功能表中，選取更新容量設定。
3. 選取佈建或隨需容量。

   若選取佈建，您可以設定最小和最大讀寫容量單位。您還可以啟用或停用自動擴展。
4. 選取 Update (更新)。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的更新資料表。

更新 GSI

若要使用 `Update Table` 操作更新 GSI 的容量設定，請執行下列動作。

> **Note**
>
> 根據預設，全域次要索引會繼承基礎資料表的容量設定。只有在基礎資料表設定為佈建容量模式時，全域次要索引才具有不同的容量模式。當您對佈建模式資料表建立全域次要索引時，必須為該索引的預期工作負載指定讀取與寫入容量單位。如需詳細資訊，請參閱全域次要索引的佈建輸送量考量。

1. 尋找待更新容量設定的 GSI。
2. 從水平省略號功能表中，選取更新容量設定。
3. 您現在可以選取 Provisioned (已佈建的) 或 On-demand capacity (隨需容量)。

   若選取佈建，您可以設定最小和最大讀寫容量單位。您還可以啟用或停用自動擴展。
4. 選取 Update (更新)。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的更新資料表。

放置項目

您可以使用 `Put Item` 操作以建立項目。若要執行或產生 `Put Item` 操作的程式碼，請執行以下操作。

1. 尋找您要在其中建立項目的資料表。
2. 從動作下拉式清單中，選取建立項目。
3. 輸入分割區索引鍵值。
4. 輸入排序索引鍵值 (如有)。
5. 若想要新增非索引鍵屬性，請執行以下作業：
   a. 選取新增其他屬性。
   b. 指定 Attribute name (屬性名稱)、Type (類型) 和 Value (值)。
6. 如果必須滿足某項條件表達式，`Put Item` 操作才能成功，請執行以下作業：
   a. 選擇 Condition (條件)。
   b. 指定屬性名稱、比較運算子、屬性類型和屬性值。
   c. 如需其他條件，請再次選擇 Condition (條件)。

   如需詳細資訊，請參閱 DynamoDB 條件表達式 CLI 範例。
7. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

   從顯示的標籤中選取所需的語言。您現在可以複製此程式碼，並使用在您的應用程式中。
8. 若希望立即執行此操作，請選擇 Run (執行)。
9. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)，然後輸入操作的名稱，再選擇 Save (儲存)。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的 PutItem。

更新項目

若要執行或產生 `Update Item` 操作的程式碼，請執行以下操作：

1. 尋找您要在其中更新項目的資料表。
2. 選取項目。

3. 輸入所選表達式的屬性名稱和屬性值。
4. 若要新增更多表達式，請在更新表達式下拉式清單中選擇另一項表達式，然後選取 + 圖示。
5. 如果必須滿足某項條件表達式，`Update Item` 操作才能成功，請執行以下作業：

   a. 選擇 Condition (條件)。
   b. 指定屬性名稱、比較運算子、屬性類型和屬性值。
   c. 如需其他條件，請再次選擇 Condition (條件)。

   如需詳細資訊，請參閱 DynamoDB 條件表達式 CLI 範例。
6. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

   選擇您想要的語言標籤。您現在可以複製此程式碼，並使用在您的應用程式中。
7. 若希望立即執行此操作，請選擇 Run (執行)。
8. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)，然後輸入操作的名稱，再選擇 Save (儲存)。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的 UpdateItem。

刪除項目

若要執行 `Delete Item` 操作，請執行下列動作。

1. 尋找您想要在其中刪除項目的資料表。
2. 選取項目。
3. 從動作下拉式清單中，選取刪除項目。
4. 選取刪除，確認您想要刪除項目。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的 DeleteItem。

複製項目

您可以建立具有相同屬性的新項目以複製項目。執行下列步驟以複製項目。

1. 尋找您要在其中複製項目的資料表。
2. 選取項目。
3. 從動作下拉式清單中，選取複製項目。

4. 指定新的分割區索引鍵。
5. 指定新的排序索引鍵 (如有必要)。
6. 選取執行。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的 DeleteItem。

Query

若要執行或產生 `Query` 操作的程式碼，請執行以下操作。

1. 從 NoSQL Workbench UI 頂端選取查詢。
2. 指定分割區索引鍵值。
3. 如果 `Query` 操作需要排序索引鍵：
   a. 選取 Sort key (排序索引鍵)。
   b. 指定比較運算子和屬性值。
4. 選取查詢以執行此查詢操作。如果需要更多選項，請勾選更多選項核取方塊並繼續執行下列步驟。
5. 如果不是所有屬性都應傳回並附帶操作結果，請選取 Projection expression (投射表達式)。
6. 選擇 + 圖示。
7. 輸入附帶查詢結果傳回的屬性。
8. 如需更多屬性，請選擇 +。
9. 如果必須滿足某項條件表達式，`Query` 操作才能成功，請執行以下作業：
   a. 選擇 Condition (條件)。
   b. 指定屬性名稱、比較運算子、屬性類型和屬性值。
   c. 如需其他條件，請再次選擇 Condition (條件)。

   如需詳細資訊，請參閱 DynamoDB 條件表達式 CLI 範例。
10. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

    選擇您想要的語言標籤。您現在可以複製此程式碼，並使用在您的應用程式中。
11. 若希望立即執行此操作，請選擇 Run (執行)。
12. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)，然後輸入操作的名稱，再選擇 Save (儲存)。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的 Query。

Scan

若要執行或產生 `Scan` 操作的程式碼，請執行以下操作。

1. 從 NoSQL Workbench UI 頂端選取掃描。
2. 選取掃描按鈕以執行此基本掃描操作。如果需要更多選項，請勾選更多選項核取方塊並繼續執行下列步驟。
3. 指定屬性名稱以篩選掃描結果。
4. 如果不是所有屬性都應傳回並附帶操作結果，請選取 Projection expression (投射表達式)。
5. 如果必須滿足某項條件表達式，scan 操作才能成功，請執行以下作業：

   a. 選擇 Condition (條件)。
   b. 指定屬性名稱、比較運算子、屬性類型和屬性值。
   c. 如需其他條件，請再次選擇 Condition (條件)。

   如需詳細資訊，請參閱 DynamoDB 條件表達式 CLI 範例。
6. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

   選擇您想要的語言標籤。您現在可以複製此程式碼，並使用在您的應用程式中。
7. 若希望立即執行此操作，請選擇 Run (執行)。
8. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)，然後輸入操作的名稱，再選擇 Save (儲存)。

TransactGetItems

若要執行或產生 `TransactGetItems` 操作的程式碼，請執行以下操作。

1. 從 NoSQL Workbench UI 頂端的更多操作下拉式清單中，選擇 TransactGetItems。
2. 選擇 TransactGetItem 附近的 + 圖示。
3. 指定分割區索引鍵。
4. 指定排序索引鍵 (如有必要)。
5. 選取執行以執行操作、儲存操作以儲存，或選取產生程式碼以產生操作的程式碼。

如需交易的詳細資訊，請參閱 Amazon DynamoDB Transactions。

TransactWriteItems

若要執行或產生 `TransactWriteItems` 操作的程式碼，請執行以下操作。

1. 從 NoSQL Workbench UI 頂端的更多操作下拉式清單中，選擇 TransactWriteItems。
2. 從動作下拉式清單中，選擇操作。
3. 選擇 TransactWriteItem 附近的 + 圖示。
4. 在動作下拉式清單中，選擇您想執行的操作。

   - 如需 `DeleteItem`，請遵循 刪除項目 操作的說明。
   - 如需 `PutItem`，請遵循 放置項目 操作的說明。
   - 如需 `UpdateItem`，請遵循 更新項目 操作的說明。

   若要變更動作順序，請在左側清單中選擇動作，然後選擇上下箭號在清單中上下移動該動作。

   若要刪除動作，請在清單中選擇動作，然後選擇 Delete (刪除) (回收桶) 圖示。
5. 選取執行以執行操作、儲存操作以儲存，或選取產生程式碼以產生操作的程式碼。

如需交易的詳細資訊，請參閱 Amazon DynamoDB Transactions。

使用 NoSQL Workbench 複製資料表

複製資料表功能會在您的開發環境之間複製資料表的索引鍵結構描述 (並可選擇 GSI 結構描述和項目)。您可以將 DynamoDB 本機版之間的資料表複製到 Amazon DynamoDB 帳戶，甚至將資料表從一個帳戶複製到不同區域中的另一個帳戶，以加快實驗速度。

複製資料表

1. 在操作建置器中，選取您的連線和區域 (DynamoDB 本機版不提供區域選取功能)。
2. 連線至 DynamoDB 後，請瀏覽您的資料表，然後選取您要複製的資料表。
3. 從水平省略號功能表中，選取複製選項。
4. 輸入複製目的地詳細資訊：

   a. 選取連線。

   b. 選取區域 (DynamoDB 本機版無法選擇區域)。

   c. 輸入新的資料表名稱。

d. 選擇複製選項：

i. 索引鍵結構描述預設為選取，且無法取消選取。根據預設，如有可用的主索引鍵及排序索引鍵，資料表進行複製時會一併複製。

ii. 如果要複製的資料表具有 GSI，則會預設選取 GSI 結構描述。如有可用的 GSI 主索引鍵及排序索引鍵，資料表進行複製時會一併複製。您可以選擇取消選取 GSI 結構描述，以略過複製 GSI 結構描述。資料表進行複製時，會複製基礎資料表的容量設定，套用為 GSI 的容量設定。您可以使用操作建置器中的 `UpdateTable` 操作，在複製完成後更新資料表的 GSI 容量設定。

5. 輸入要複製的項目數量。如僅需複製索引鍵結構描述和選用的 GSI 結構描述，您可以將待複製項目的值維持在 0。可複製的項目數量上限為 5000。

6. 選擇容量模式：

a. 依預設，會選取隨需模式。DynamoDB 隨需提供讀取和寫入請求pay-per-request定價，因此您只需按使用量付費。如需進一步了解，請參閱 DynamoDB 隨需模式。

b. 佈建模式可讓您指定應用程式需要的每秒讀取和寫入次數。您可以使用自動擴展來自動調整資料表的佈建容量，以回應流量的變動。如需進一步了解，請參閱 DynamoDB 佈建模式。

7. 選取複製以開始複製。

8. 複製程序將在背景執行。當複製資料表的狀態發生變更時，操作建置器索引標籤會顯示通知。您可以選取操作建置器索引標籤，接著選取箭頭按鈕以存取此狀態。箭頭按鈕位於複製資料表狀態小工具上，靠近功能表側邊欄底部。

將資料匯出到 CSV 檔案

您可以從作業建置器將查詢的結果匯出到 CSV 檔案。這可讓您將資料載入試算表或使用偏好的程式設計語言來處理資料。

匯出至 CSV

1. 在作業產生器中，執行您選擇的作業，例如掃描或查詢。

> **Note**
>
> - 您只能將讀取 API 操作和 PartiQL 陳述式的結果匯出至 CSV 檔案。您無法從交易讀取陳述式匯出結果。

- 目前，您每次可以匯出一個頁面的結果到 CSV 檔案。如果有多個結果頁面，您必須個別匯出每個頁面。

2. 從結果中選取您要匯出的項目。
3. 從動作下拉式清單中，選擇匯出 CSV。
4. 為 CSV 檔案選擇檔案名稱及位置，然後選擇 Save (儲存)。

NoSQL Workbench 的範例資料模型

模型器的首頁會顯示 NoSQL Workbench 隨附的許多範例模型。本節說明這些模型及其潛在用途。

主題

- 員工資料模型
- 開發論壇資料模型
- 音樂資料庫資料模型
- 滑雪渡假村資料模型
- 信用卡優惠資料模型
- 書籤資料模型

員工資料模型

這個資料模型是一個簡介模型。它代表了員工的基本詳細資料，例如唯一的別名、名字、姓氏、頭銜、主管和技能。

該資料模型會描繪了一些技術，例如處理複雜工作的屬性，像是擁有不只一個技能。此模型也會透過次要索引 DirectEports 的範例，說明主管與其直屬員工的一對多關係。

此資料模型有助於以下存取模式：

- 使用員工的登入別名擷取員工記錄，您可透過名為 `Employee` 的表格來加快完成此動作。
- 按名稱搜尋員工，您可透過名為 `Name` 的員工表格全域次要索引來加快完成此動作。
- 使用管理員的登入別名擷取管理員的所有直屬報告，您可透過名為 `DirectReports` 的員工表全域次要索引來加快完成此動作。

開發論壇資料模型

這個資料模型代表開發論壇。使用此模型，客戶可以與開發人員社群互動、提出問題，以及回應其他客戶的貼文。每一種 AWS 服務都有專用的論壇。任何人都可以在論壇張貼訊息、開始新的主題，每個主題都會收到許多回覆。

此資料模型有助於以下存取模式：

- 使用論壇名稱擷取論壇記錄，您可透過名為 Forum 的表格來加快完成此動作。
- 擷取某個論壇的特定主題或所有主題，您可透過名為 Thread 的資料表來加快完成此動作。
- 使用張貼使用者的電子郵件地址搜尋回覆，您可透過名為 PostedBy-Message-Index 的 Reply 資料表的全域次要索引來加快完成此動作。

音樂資料庫資料模型

此資料模型代表具有大量歌曲選集的音樂資料庫，能以近乎即時的方式展示下載次數最多的歌曲。

此資料模型有助於以下存取模式：

- 擷取一首歌曲記錄，您可透過名為 Songs 的表格加快完成此動作。
- 擷取特定的下載記錄或歌曲的所有下載記錄，您可透過名為 Songs 的表格加快完成此動作。
- 擷取歌曲特定的每月下載次數記錄，您可透過名為 Song 的表格加快完成此動作。
- 擷取歌曲的所有記錄 (包括歌曲記錄、下載記錄和每月下載次數記錄)，您可透過名為 Songs 的表格加快完成此動作。
- 搜尋最多人下載的歌曲，您可透過名為 DownloadsByMonth 的表格的全域次要索引加快完成此動作。

滑雪渡假村資料模型

此資料模型代表一個滑雪渡假村，其中有每部滑雪纜車每天收集到的大量資料集合。

此資料模型有助於以下存取模式：

- 擷取指定滑雪纜車或整體渡假村的所有資料 (動態和靜態)，您可透過名為 SkiLifts 的表格加快完成此動作。
- 在特定日期擷取滑雪纜車或整體渡假村的所有動態資料 (包括特別的纜車乘客，積雪覆蓋率、雪崩危險性和纜車狀態)，您可透過名為 SkiLifts 的表格加快完成此動作。

- 擷取特定滑雪纜車的所有靜態資料 (包括纜車是否是給有經驗的乘客使用、纜車升高的垂直高度 (英尺)、纜車乘坐時間)，您可透過名為 SkiLifts 的表格加快完成此動作。
- 擷取對特定滑雪纜車或整體渡假村記錄的資料日期 (依特別的乘客總數排序)，您可透過名為 SkiLiftsByRiders 的 SkiLifts 表格的全域次要索引加快完成此動作。

信用卡優惠資料模型

此資料模型由信用卡優惠應用程式使用。

信用卡供應商會不時提供優惠。這些優惠包括不收費的餘額轉移、提高信用額度、降低利率、現金回饋和航空公司里程數。客戶接受或拒絕這些優惠後，相應的優惠狀態會隨之更新。

此資料模型有助於以下存取模式：

- 使用 AccountId 擷取帳戶記錄，您可透過主表格加快完成此動作。
- 擷取有少量預估項目的所有帳戶，您可透過次要索引 AccountIndex 加快完成此動作。
- 使用 AccountId 擷取帳戶和所有與這些帳戶相關的優惠，您可透過主表格加快完成此動作。
- 使用 AccountId 和 OfferId 擷取與這些帳戶相關的帳戶和特定優惠記錄，您可透過主表格加快完成此動作。
- 使用 AccountId、OfferType 和 Status 擷取所有與帳戶相關且為 OfferType 特定的所有 ACCEPTED/DECLINED 優惠記錄，您可透過次要索引 GSI1 加快完成此動作。
- 使用 OfferId 擷取優惠和相關的優惠項目記錄，您可透過主表格加快完成此動作。

書籤資料模型

這個資料模型可供客戶用來存放書籤。

一個客戶可以有很多個書籤，且一個書籤可以屬於許多客戶。這個資料模型代表多對多的關係。

此資料模型有助於以下存取模式：

- customerId 的單一查詢現在可以傳回客戶資料以及書籤。
- 查詢 ByEmail 索引會透過電子郵件地址傳回客戶資料。請注意，此索引不會擷取書籤。
- 查詢 ByUrl 索引會透過 URL 取得書籤資料。請注意，我們有 CustomerID 作為索引的排序金鑰，因為相同的 URL 可以被多個客戶加入書籤。
- 查詢 ByCustomerFolder 索引會依資料夾取得每位客戶的書籤。

NoSQL Workbench 的版本歷史記錄

下表說明在各版 NoSQL Workbench 用戶端工具的重大變更。

版本	變更	描述	Date
3.20.0	適用於 DynamoDB 的新資料模型器	DynamoDB 的資料模型器具有更新的使用者體驗。DynamoDB 的資料模型器現在支援存取模式。	2026 年 2 月 16 日
3.13.5	目前資料表預設的容量模式為隨需模式	您使用預設設定建立資料表時，DynamoDB 建立的資料表會使用隨需容量模式而非佈建容量模式。	2025 年 2 月 24 日
3.13.0	NoSQL Workbench 操作建置器改進	NoSQL Workbench 現在包含對深色模式的原生支援。改善操作建置器中的資料表和項目操作。項目結果和操作建置器請求資訊以 JSON 格式提供。	2024 年 4 月 24 日
3.12.0	使用 NoSQL Workbench 複製資料表並傳回消耗的容量	您現在可以在 DynamoDB 本機版和 DynamoDB Web 服務帳戶之間，或在多個 DynamoDB Web 服務帳戶之間複製資料表，以加快開發迭代的速度。使用 Operations Builder (操作建置器) 執行操作後，檢視已使用的	2024 年 2 月 26 日

版本	變更	描述	Date
		RCU 或 WCU。我們已修正從 CSV 檔案匯入時的覆寫資料問題。	
3.11.0	DynamoDB 本機版改善	您現在可於啟動內建 DynamoDB 本機版執行個體時指定連接埠。NoSQL Workbench 現在可以在沒有管理員權限的情況下安裝在 Windows 上。我們已更新資料模型範本。	2024 年 1 月 17 日
3.10.0	適用於 Apple 晶片的原生支援	NoSQL Workbench 現在可原生支援搭載 Apple 晶片的 Mac。您現在可以為 Number 類型的屬性，設定範例資料產生格式。	2023 年 12 月 5 日
3.9.0	資料模型建立工具改進功能	視覺化工具現在可支援將資料模型提交至 DynamoDB 本機版，並且可選擇覆寫現有資料表。	2023 年 11 月 3 日
3.8.0	產生範例資料	NoSQL Workbench 現在支援為您的 DynamoDB 資料模型產生範例資料。	2023 年 9 月 25 日

版本	變更	描述	Date
3.6.0	操作建置器的改善項目	操作建置器的連線管理改善項目。資料模型建立工具中的屬性名稱現在無需刪除資料即可變更。其他錯誤修復。	2023 年 4 月 11 日
3.5.0	支援新 AWS 區域	NoSQL Workbench 現在支援 ap-south-2、ap-southeast-3、ap-southeast-4、eu-central-2、eu-south-2、me-central-1 和 me-west-1 區域。	2023 年 2 月 23 日
3.4.0	DynamoDB 本機版的支援	NoSQL Workbench 現在支援 DynamoDB 本機版做為安裝程序的一部分。	2022 年 12 月 6 日
3.3.0	控制平面操作支援	操作建置器現在支援控制平面操作。	2022 年 6 月 1 日
3.2.0	CSV 匯入和匯出	您現在可以從視覺化檢視工具中的 CSV 檔案匯入範例資料，也可以將作業產生器查詢的結果匯出至 CSV 檔案。	2021 年 10 月 11 日
3.1.0	儲存操作	NoSQL Workbench 中的操作建置器現在支援儲存操作以供日後使用。	2021 年 7 月 12 日

版本	變更	描述	Date
3.0.0	容量設定和 CloudFormation 匯入/匯出	Amazon DynamoDB 專用 NoSQL Workbench 現在不僅支援為資料表指定讀取/寫入容量模式，還能以 CloudFormation 格式匯入和匯出資料模型。	2021 年 4 月 21 日
2.2.0	PartiQL 支援	Amazon DynamoDB 專用 NoSQL Workbench 新增支援建置 DynamoDB 的 PartiQL 陳述式。	2020 年 12 月 4 日
1.1.0	支援 Linux。	Linux Ubuntu、Fedora 與 Debian 均支援 Amazon DynamoDB 專用 NoSQL Workbench。	2020 年 5 月 4 日
1.0.0	Amazon DynamoDB 專用 NoSQL Workbench：GA 版。	Amazon DynamoDB 專用 NoSQL Workbench 現已正式推出。	2020 年 3 月 2 日
0.4.1	支援 IAM 角色和臨時安全憑證。	Amazon DynamoDB 專用 NoSQL Workbench 新增對 AWS Identity and Access Management (IAM) 角色和臨時安全憑證的支援。	2019 年 12 月 19 日

版本	變更	描述	Date
0.3.1	支援 DynamoDB 本機版 (可下載版本)。	NoSQL Workbench 現在支援連接至 DynamoDB 本機版 (可下載版本)，以設計、建立、查詢和管理 DynamoDB 資料表。	2019 年 11 月 8 日
0.2.1	已發行 NoSQL Workbench 預覽。	這是 DynamoDB 專用 NoSQL Workbench 的初始版本。使用 NoSQL Workbench 設計、建立、查詢及管理 DynamoDB 資料表。	2019 年 9 月 16 日

DynamoDB 的備份與還原

DynamoDB 提供隨需備份和時間點復原 (PITR) 備份，以協助保護您的 DynamoDB 資料免受災難事件影響，並提供資料封存以供長期保留。您可以備份資料量從幾 MB 到數百 TB 不等的資料表，而且不會影響您生產應用程式的效能與可用性。所有備份都會自動加密、編製目錄且易於探索。

透過隨需備份，您可以為 DynamoDB 儲存和管理的資料表建立快照備份。您需要根據備份的大小和持續時間付費。使用隨需備份，您可以將整個 DynamoDB 資料表還原至建立備份時的確切狀態。

建立和管理 DynamoDB 隨需備份有兩種選項：

- DynamoDB
- AWS Backup

您可以使用 DynamoDB 隨需備份功能，為法規合規需求建立資料表的完整備份以供長期保留與封存。您可以隨時從 AWS 管理主控台 或使用單一 API 呼叫來備份及還原資料表資料。

時間點復原 (PITR) 備份完全由 DynamoDB 管理，並以每秒精細程度提供最多 35 天的復原點。若要使用持續備份的時間點復原，請在 DynamoDB 資料表上啟用時間點復原 (PITR)。您需要根據 DynamoDB 資料表的大小以及在資料表上啟用 PITR 的持續時間付費。在 DynamoDB 資料表上啟用時間點復原 (PITR) 可持續備份您的資料。這可協助您建立在該時間點具有原始資料表確切狀態的新 DynamoDB 資料表，在 PITR 復原期間內將資料表還原至特定時間點。

時間點復原有助於保護您的 DynamoDB 資料表免遭意外寫入或刪除操作。有了時間點復原，就無需為建立、維護或排程隨需備份而煩惱。例如，假設測試指令碼意外寫入至生產 DynamoDB 資料表。

透過時間點復原，您可以將該資料表還原到過去 35 天內的任何時間點。您可以將復原期間設為 1 到 35 天。啟用時間點復原後，您可以還原到目前時間前五分鐘至設定的還原期間之間的任何時間點。DynamoDB 維護您資料表的增量備份。

此外，時間點操作並不會影響效能或 API 延遲。

您可使用 AWS 管理主控台、AWS Command Line Interface (AWS CLI) 或 DynamoDB API 將 DynamoDB 資料表還原至某個時間點。時間點復原過程會還原到新資料表。

如需 AWS 區域可用性和定價的詳細資訊，請參閱 Amazon DynamoDB 定價。

> **Note**
>
> - DynamoDB 備份不支援標記和屬性型存取控制 (ABAC)。若要在備份中使用 ABAC，建議您使用 AWS Backup。
> - 標籤不會在還原的資料表中保留。您必須先將標籤新增至還原的資料表，才能在政策中使用標籤式條件。

以下影片將為您介紹備份與還原概念，並詳細介紹有關時間點復原的更多資訊。

備份與還原

主題

- DynamoDB 的時間點備份
- 使用 DynamoDB 的隨需備份與還原
- 了解備份的 Amazon DynamoDB 帳單
- 還原 DynamoDB 資料表
- 搭配 DynamoDB 使用 AWS Backup

DynamoDB 的時間點備份

Amazon DynamoDB 時間點復原 (PITR) 可持續備份 DynamoDB 資料表的資料。時間點復原 (PITR) 備份完全由 DynamoDB 管理，並以每秒精細程度提供最多 35 天的復原點。有了時間點復原，就無需為建立、維護或排程隨需備份而煩惱。本章節將概要說明該過程如何在 DynamoDB 中運作。

開始之前

在您於 Amazon DynamoDB 資料表上啟用時間點復原 (PITR) 之前，建議您考慮下列事項：

- 設定 `RecoveryPeriodinDays` 可讓您縮短進行連續備份的期間。根據預設，您的 `RecoveryPeriodinDays` 為 35。不過，您可以將其設定為介於 1 到 35 之間的任何值。縮短 `RecoveryPeriodinDays` 不會影響 PITR 定價，因為價格是以資料表和本機次要索引的大小為基礎。
- 如果您在資料表上停用時間點復原後又重新啟用，將會重設可以恢復該資料表的開始時間。因此，您只能使用 `LatestRestorableDateTime` 立即還原該資料表。

- 您可以在全域資料表的每個區域複本上啟用時間點復原。在還原資料表時，備份會還原到不屬於全域資料表的獨立資料表。如果您使用全域資料表版本 2019.11.21 (目前) 的全域資料表，您可以從還原的資料表建立新的全域資料表。如需更多詳細資訊，請參閱 DynamoDB 全域資料表的運作方式。
- 您也可以跨 AWS 區域還原 DynamoDB 資料表資料，將還原的資料表建立在不同於來源資料表所在的區域。您可以在 AWS 商業區域、AWS 中國區域和 AWS GovCloud (US) 區域之間進行跨區域還原。您只需為從來源區域中傳輸出來的資料，以及還原為目標區域中的新資料表付費。
- AWS CloudTrail 記錄所有主控台和 API 動作來進行時間點復原，以啟用日誌記錄、持續監控和稽核。如需更多詳細資訊，請參閱 使用 記錄 DynamoDB 操作 AWS CloudTrail。

主題

- DynamoDB 的時間點復原

DynamoDB 的時間點復原

Amazon DynamoDB 時間點復原 (PITR) 可自動備份 DynamoDB 資料表的資料。本章節將概要說明該過程如何在 DynamoDB 中運作。

> **Note**
>
> 根據每個 DynamoDB 資料表的大小計算 PITR 的 DynamoDB 費用，包括資料表資料和本機次要索引。變更復原時段 (例如，從 35 天變更為 1 天) 不會降低價格。無論您選擇的復原期間為何，成本都會保持不變。設定的最長復原期間不會影響您開啟 PITR 時需支付的價格。為了判斷備份費用，DynamoDB 會持續監控已開啟 PITR 的資料表大小。在您關閉每個資料表的 PITR 之前，都會向您收取 PITR 用量的費用。

主題

- 啟用時間點復原
- 啟用 PITR (主控台)
- 啟用 PITR (AWS CLI)
- 啟用 PITR (CloudFormation)
- 啟用 PITR (API)
- 復原期間
- 編輯 PITR

- 刪除已啟用 PITR 的資料表

啟用時間點復原

您可以使用 AWS 管理主控台、 AWS Command Line Interface (AWS CLI) 或 DynamoDB API 啟用 point-in-time復原。啟用後，時間點復原會提供連續備份，直到您明確表示將其關閉。

在啟用時間點復原之後，您可以還原到 `EarliestRestorableDateTime` 和 `LatestRestorableDateTime` 內的任何時間點。`LatestRestorableDateTime` 通常為目前時間的前 5 分鐘。如需詳細資訊，請參閱還原 DynamoDB 資料表至某個時間點。

> **Note**
>
> 時間點復原過程一律會還原到新資料表。

啟用 PITR (主控台)

使用 DynamoDB 主控台啟用 PITR

1. 導覽至 DynamoDB 主控台。
2. 從左側導覽中選擇資料表，然後選取您的 DynamoDB 資料表。
3. 從備份索引標籤中，針對時間點復原選項，選擇編輯。
4. 選擇開啟時間點復原。
5. 為您的備份復原期間選擇介於 1 到 35 之間的值。這表示可復原連續備份的最長期間。

啟用 PITR (AWS CLI)

> **Note**
>
> 如果您在執行 AWS CLI 命令時收到錯誤，請參閱故障診斷 AWS CLI 錯誤。請確定您使用的是最新版本 AWS CLI 。

在開啟 point-in-time-recovery-specification 設定的情況下執行 update-continuous-backups 命令：

```
aws dynamodb update-continuous-backups \
```

```
--table-name Music \
--point-in-time-recovery-specification
 PointInTimeRecoveryEnabled=true,RecoveryPeriodInDays=35
```

啟用 PITR (CloudFormation)

使用 `PointInTimeRecoverySpecification` 屬性已開啟的 AWS::DynamoDB::Table 資源：

```
Resources:
  iotCatalog:
    Type: AWS::DynamoDB::Table
      Properties:
      ...
      PointInTimeRecoverySpecification:
        PointInTimeRecoveryEnabled: true
        RecoveryPeriodInDays: 35
```

請求語法範例：

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays: number
   },
   "TableName": "string"
}
```

啟用 PITR (API)

在 `PointInTimeRecoverySpecification` 參數開啟的情況下執行 UpdateContinuousBackups API 操作。

請求語法範例：

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays" : number
   },
   "TableName": "string"
}
```

回應語法範例：

```
{
   "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
    }
)
```

復原期間

您可以將連續備份的復原期間設定為 1 到 35 天之間的任何數字。此一 `RecoveryPeriodInDays` 決定維護連續備份的期間。例如，如果您將此值設定為 30 天，則只能將資料表還原至過去 30 天的任何時間點。

> **Note**
>
> 根據每個 DynamoDB 資料表的大小計算 PITR 的 DynamoDB 費用，包括資料表資料和本機次要索引。設定的最長復原期間不會影響您開啟 PITR 時需支付的價格。如需定價詳細資訊，請參閱 DynamoDB 定價。

編輯 PITR

您可以針對資料表編輯 PITR 設定，並變更復原期間。如果您變更復原期間，並將其增加到高於先前設定的值，您的 EarliestRestorePoint 將不會立即變更。由於復原期間是滾動時段，DynamoDB 將繼續自動備份，直到達到新的增加期間為止。如果您變更復原期間並將其減少為低於先前設定的值，您的 EarliestRestorePoint會立即減少以符合復原期間，而且任何超出新設定值的連續備份將無法復原。

刪除已啟用 PITR 的資料表

當您刪除一個已啟用時間點復原的資料表時，DynamoDB 會自動建立一個備份快照，稱為系統備份，並保留 35 天 (無需額外費用)。你可以使用系統備份將已刪除的資料表還原到刪除之前的狀態。所有的系統備份都遵循###-##$DeletedTableBackup的標準命名慣例。

> **Note**
>
> 刪除已啟用時間點復原的資料表後，您可以使用系統還原，將該資料表還原到單一時間點。系統備份將在刪除資料表時建立，並且是在刪除資料表之前資料表的快照。

使用 DynamoDB 的隨需備份與還原

Amazon DynamoDB 支援獨立的隨需備份與還原功能。無論您是否使用 AWS Backup，都可以使用這些功能。

您可以使用 DynamoDB 隨需備份功能針對法規合規需求，建立資料表的完整備份以供長期保留與封存。只要按一下 AWS 管理主控台或單一 API 呼叫，您就可以隨時備份和還原資料表資料。備份與還原動作執行時，完全不會影響資料表效能或可用性。

您可以使用 主控台、 AWS 命令列界面 (AWS CLI) 或 DynamoDB API 建立資料表備份。如需詳細資訊，請參閱備份 DynamoDB 資料表。

如需從備份還原資料表的資訊，請參閱「從備份中還原 DynamoDB 資料表」。

使用 DynamoDB 備份和還原 DynamoDB 資料表：運作方式

您可以使用 DynamoDB 隨需備份功能來建立 Amazon DynamoDB 資料表的完整備份。此功能與 AWS 備份無關。本節概述 DynamoDB 備份與還原程序期間所發生的情況。

備份

當您用 DynamoDB 建立隨需備份時，系統會為請求的時間標記建立目錄。備份以非同步方式建立，它會套用所有變更，直到請求最後一個完整資料表快照為止。系統會立即處理 DynamoDB 備份請求，該請求幾分鐘內就可還原。

> **Note**
>
> 每次您建立隨需備份時，系統都會備份整個資料表資料。您可建立的隨需備份數目不限。

DynamoDB 中的所有備份在執行時皆不會使用資料表的任何佈建輸送量。

DynamoDB 備份並不保證項目之間具因果一致性，但是備份中更新之間的扭曲通常遠少於一秒。

您無法在備份進行時，執行下列操作：

- 暫停或取消備份操作。
- 刪除備份的來源資料表。
- 在資料表的備份進行時，停用該資料表的備份。

如果您不想建立排程指令碼和清除任務，您可以使用 AWS Backup 為 DynamoDB 資料表建立具有排程和保留政策的備份計劃。 AWS Backup 會執行備份，並在它們過期時將其刪除。如需詳細資訊，請參閱《AWS Backup 開發人員指南》https://docs.aws.amazon.com/aws-backup/latest/devguide/whatisbackup.html。

此外 AWS Backup，您可以使用 AWS Lambda 函數來排程定期或未來的備份。如需詳細資訊，請參閱無伺服器解決方案以排定 Amazon DynamoDB 隨需備份。

如果您使用 主控台，使用 建立的任何備份 AWS Backup 都會列在 Backups 索引標籤上，並將 Backup 類型設定為 AWS。

> **Note**
>
> 您無法使用 DynamoDB 主控台刪除標示為 Backup type (備份類型) 的 AWS 。若要管理這些備份，請使用 AWS Backup 主控台。

若要了解如何執行備份，請參閱「備份 DynamoDB 資料表」。

還原

還原資料表時，不會使用資料表的任何佈建輸送量。您可以從 DynamoDB 備份執行完整資料表還原，或設定目的地資料表設定。當您執行還原時，可以變更下列資料表設定：

- 全域次要索引 (GSI)
- 本機次要索引 (LSI)
- 帳單模式
- 佈建的讀取與寫入容量
- 加密設定

> **Important**
>
> 當您執行完整資料表還原時，會使用與來源資料表相同的佈建讀取容量單位與寫入容量單位來設定目標資料表，如請求備份時所記錄。還原程序也會還原本機次要索引與全域次要索引。

您也可以跨 AWS 區域還原 DynamoDB 資料表資料，以便在與備份所在的不同區域中建立還原的資料表。您可以在 AWS 商業區域、 AWS 中國區域和 AWS GovCloud (US) 區域之間進行跨區域還原。您只需為從來源區域中傳輸出來的資料，以及還原為目標區域中的新資料表付費。

如果您選擇阻止在新還原的資料表上建立部分或全部次要索引，則還原會更快且更符合經濟效益。

您必須在還原的資料表上手動進行下列設定：

- 自動擴展政策
- AWS Identity and Access Management (IAM) 政策
- Amazon CloudWatch 指標和警示
- Tags (標籤)
- 串流設定
- 存留時間 (TTL) 設定
- 刪除保護設定
- 時間點復原 (PITR) 設定

您只能從備份將整個資料表資料還原至新資料表。只有在還原的資料表變成作用中之後，您才能寫入該資料表。

> **Note**
>
> 您無法在還原操作期間覆寫現有的資料表。

還原時間與資料表的組態（例如資料表的大小和基礎分割區的數量）和其他相關變數直接相關。規劃災難復原的最佳實務是定期記錄平均還原完成時間，並確認這些時間如何影響整體復原時間點目標。

若要了解如何執行還原，請參閱「從備份中還原 DynamoDB 資料表」。

您可以使用 IAM 政策進行存取控制。如需詳細資訊，請參閱 搭配 IAM 使用 DynamoDB 備份與還原。

所有備份與還原主控台以及 API 動作都會擷取並記錄到 AWS CloudTrail 中，以記錄、持續監控與稽核。

備份 DynamoDB 資料表

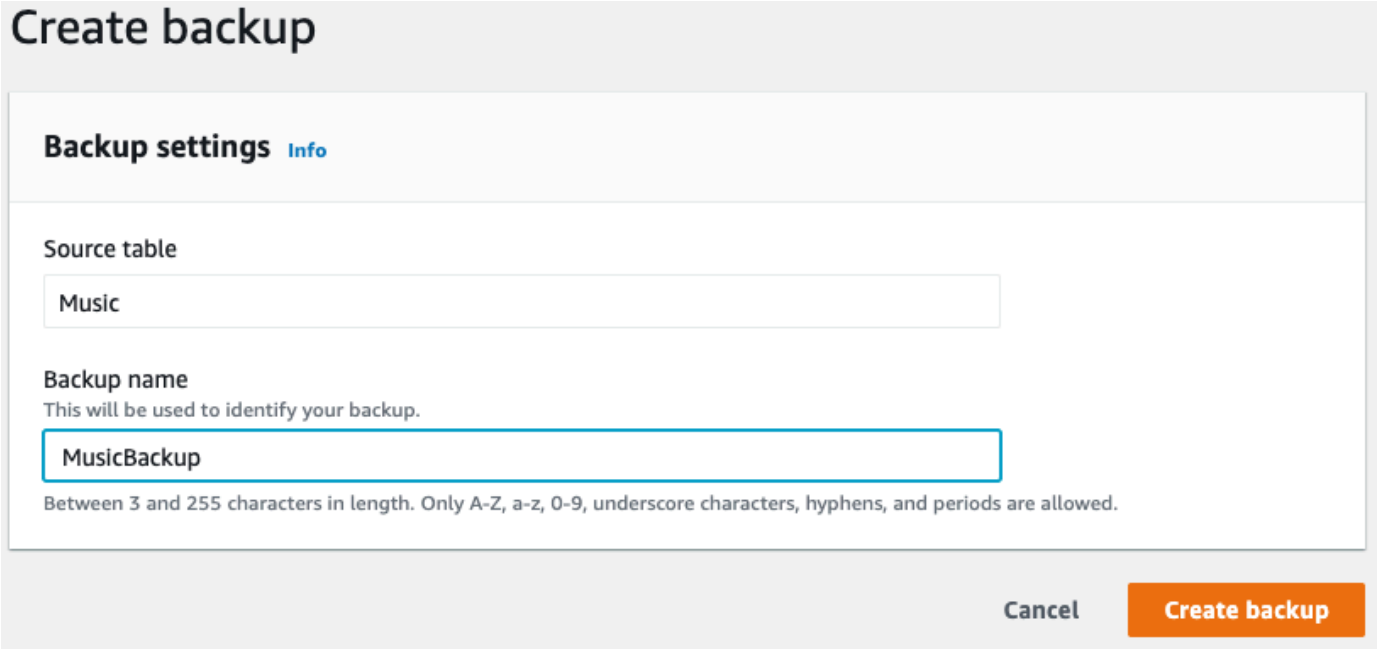
本節說明如何使用 Amazon DynamoDB 主控台或AWS Command Line Interface備份資料表。

建立資料表備份 (主控台)

請遵循下列步驟，使用AWS 管理主控台為現有的 `Music` 資料表建立名為 `MusicBackup` 的備份。

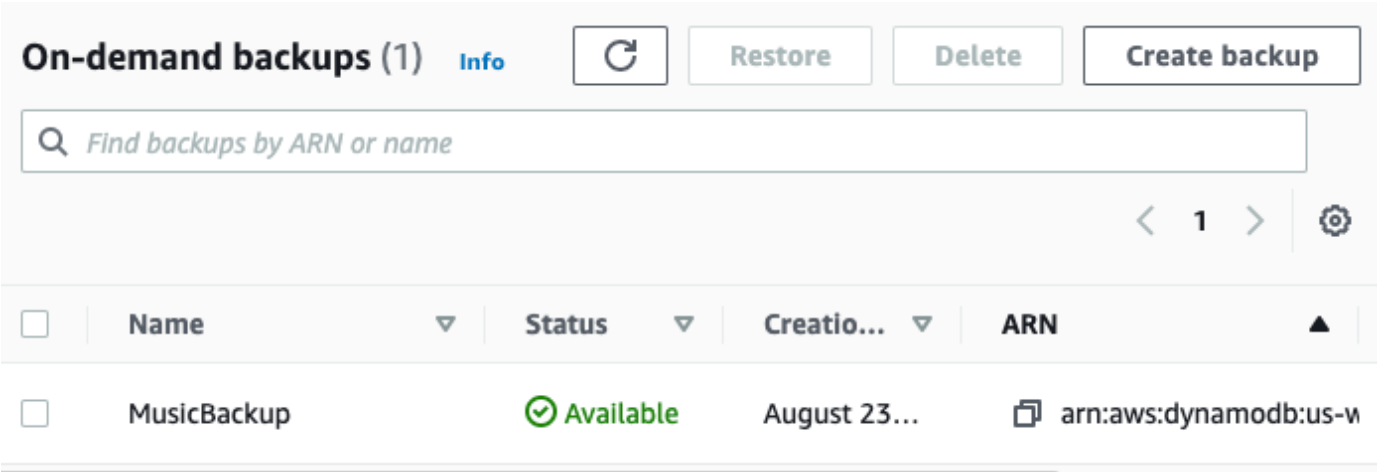
建立資料表備份

1. 登入AWS 管理主控台，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 您可以執行下列其中一項操作來建立備份：
   - 在 `Music` 資料表的 Backups (備份) 索引標籤上，選擇 Create backup (建立備份)。
   - 在主控台左側的導覽窗格中，選擇 Backups (備份)。然後選擇 Create backup (建立備份)。
3. 請確定 `Music` 是資料表名稱，然後輸入 **`MusicBackup`** 做為備份名稱。然後，選擇建立備份以建立備份。



> **Note**
>
> 如果您使用導覽窗格中的 Backups (備份) 區段建立備份，則不會為您預先選取資料表。您必須手動選擇備份的來源資料表名稱。

建立備份期間，備份狀態會設定為 Creating (建立中)。備份完成之後，備份狀態會變更為 Available (可用)。



建立資料表備份 (AWS CLI)

請遵循下列步驟，使用AWS CLI為現有的資料表 `Music` 建立備份。

建立資料表備份

- 為 MusicBackup 資料表建立名為 Music 的備份。

```
aws dynamodb create-backup --table-name Music \
 --backup-name MusicBackup
```

建立備份期間，備份狀態會設定為 CREATING。

```
{
    "BackupDetails": {
        "BackupName": "MusicBackup",
        "BackupArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc",
        "BackupStatus": "CREATING",
        "BackupCreationDateTime": 1489602797.149
    }
}
```

備份完成後，其 BackupStatus 應該變更為 AVAILABLE。若要確認，請使用 describe-backup 命令。您可以從先前步驟的輸出或使用 backup-arn 命令取得 list-backups 的輸入值。

```
aws dynamodb describe-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/01489173575360-
b308cd7d
```

若要追蹤您的備份，您可以使用 list-backups 命令。它會列出處於 CREATING 或 AVAILABLE 狀態的所有備份。

```
aws dynamodb list-backups
```

list-backups 命令與 describe-backup 命令適用於查看備份之來源資料表的相關資訊。

從備份中還原 DynamoDB 資料表

本節說明如何使用 Amazon DynamoDB 主控台或AWS Command Line Interface() 從備份還原資料表 AWS CLI。

> **Note**
>
> 如果您想要使用AWS CLI，您必須先進行設定。如需詳細資訊，請參閱存取 DynamoDB。

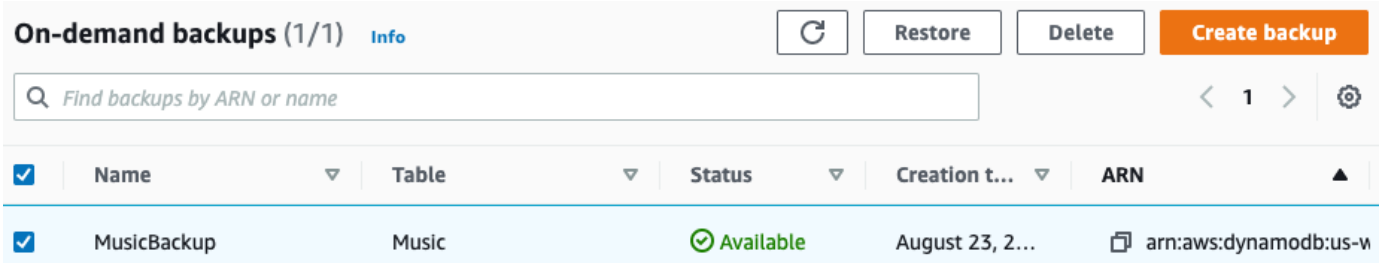
從備份中還原資料表 (主控台)

下列程序示範如何使用在「Music」教學中建立的 MusicBackup 檔案還原 備份 DynamoDB 資料表 資料表。

> **Note**
>
> 此程序假設在使用 Music 檔案還原資料表前 MusicBackup 資料表已不存在。

從備份還原資料表

1. 登入AWS 管理主控台，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 在備份清單中，選擇 MusicBackup。



4. 選擇 Restore (還原)。
5. 輸入 **Music** 做為新資料表的名稱。確認備份名稱和其他備份詳細資訊。然後選擇 Restore table (還原資料表) 啟動還原程序。

> **Note**
>
> 您可以將資料表還原至與備份所在的相同AWS區域或不同區域。您可以阻止在新還原的資料表上建立次要索引。此外，您可以指定不同的加密模式。
> 從備份還原的資料表一律使用 DynamoDB 標準資料表類別建立。

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

正在還原的資料表會顯示為 Creating (正在建立) 狀態。還原程序完成後，Music 資料表的狀態會變更為 Active (作用中)。

從備份中還原資料表 (AWS CLI)

請依照下列步驟，使用備份 DynamoDB 資料表教學課程中建立MusicBackup的AWS CLI來使用 還原Music資料表。

從備份還原資料表

1. 使用 list-backups 命令確認您想要還原的備份。此範例使用 MusicBackup。

```
aws dynamodb list-backups
```

如需備份的其他詳細資訊，請使用 describe-backup 命令。您可以從上一步取得輸入 backup-arn。

```
aws dynamodb describe-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d
```

2. 從備份還原資料表。在此情況下， 會將Music資料表MusicBackup還原至相同的AWS區域。

```
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. 利用自訂的資料表設定，從儲存貯體還原資料表。在本例中，MusicBackup 會還原 Music 資料表並指定還原資料表的加密模式。

> **Note**
>
> sse-specification-override 參數採用的數值與 CreateTable 命令中使用的 sse-specification-override 參數相同。如需詳細資訊，請參閱 在 DynamoDB 中管理加密資料表。

```
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
```

您可以將資料表還原到與備份所在的不同AWS區域。

> **Note**
>
> - 執行跨區域還原必須使用 `sse-specification-override` 參數，而還原至與來源資料表相同的區域時則可選用此參數。
> - 從命令列執行跨區域還原時，您必須將預設AWS區域設定為所需的目的地區域。如需進一步了解，請參閱《AWS Command Line Interface使用者指南》中的命令列選項。

```
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
```

您可以覆寫帳單模式及佈建給還原資料表的輸送量。

```
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
```

您可以阻止在還原的資料表上建立部分或全部次要索引。

> **Note**
>
> 如果您阻止在還原的資料表上建立部分或全部次要索引，可加速還原且更符合經濟效益。

```
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**
>
> 提供的次要索引應該符合現有索引。您無法在還原時建立新索引。

您可以使用不同覆寫的組合。例如，您可以使用單一全域次要索引，同時變更所佈建的輸送量，如下所示。

```
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
```

若要確認還原，請使用 `describe-table` 命令描述 `Music` 資料表。

```
aws dynamodb describe-table --table-name Music
```

正在從備份還原的資料表會顯示為 Creating (正在建立) 狀態。還原程序完成後，`Music` 資料表的狀態會變更為 Active (作用中)。

> **Important**
>
> 當還原正在進行時，請勿修改或刪除 IAM 角色政策，否則可能會造成非預期的行為。例如，假設您在資料表還原時移除資料表的寫入許可。在此案例中，基礎 `RestoreTableFromBackup` 操作會無法將任何還原的資料寫入資料表。
>
> 還原操作完成後，您就可以修改或刪除 IAM 角色政策。

涉及存取目標還原資料表的來源 IP 限制的 IAM 政策，應該將 `aws:ViaAWSService` 金鑰設定為 `false`，以此確保限制僅適用於委託人直接提出的請求。否則，還原將被取消。
如果您的備份使用AWS 受管金鑰或客戶受管金鑰加密，請勿在還原進行時停用或刪除金鑰，否則還原將會失敗。
還原操作完成後，您可以變更還原資料表的加密金鑰，並停用或刪除舊的金鑰。

刪除 DynamoDB 資料表備份

本節說明如何使用AWS 管理主控台或AWS Command Line Interface(AWS CLI) 刪除 Amazon DynamoDB 資料表備份。

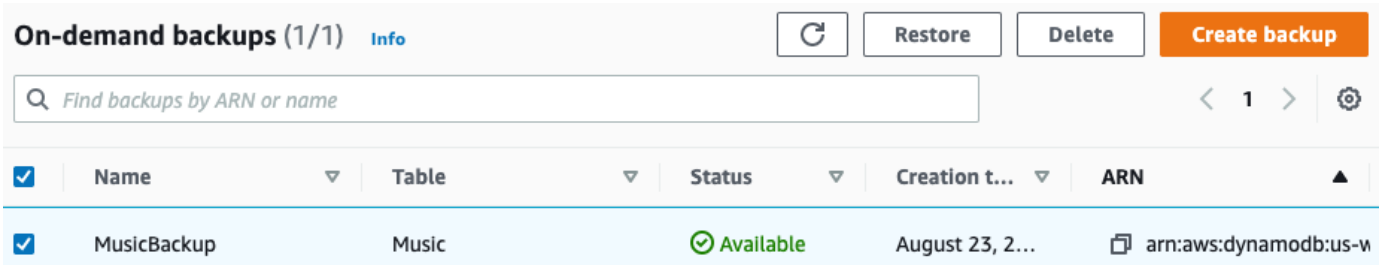
> **Note**
> 如果您想要使用AWS CLI，您必須先進行設定。如需詳細資訊，請參閱使用 AWS CLI。

刪除資料表備份 (主控台)

下列程序顯示如何使用主控台刪除 備份 DynamoDB 資料表 教學中所建立的 `MusicBackup`。

刪除備份

1. 登入AWS 管理主控台，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 在備份清單中，選擇 `MusicBackup`。



4. 選擇 刪除。確認您要刪除備份，方法是輸入**`delete`**，然後按一下 Delete (刪除)。

刪除資料表備份 (AWS CLI)

下列範例會使用AWS CLI來刪除現有資料表 `Music` 資料表的備份。

```
aws dynamodb delete-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc
```

搭配 IAM 使用 DynamoDB 備份與還原

您可以使用 AWS Identity and Access Management (IAM) 來限制某些資源的 Amazon DynamoDB 備份和還原動作。在每份資料表中操作 `CreateBackup` 和 `RestoreTableFromBackup` API。

如需在 DynamoDB 中使用 IAM 政策的詳細資訊，請參閱 適用於 DynamoDB 的以身分為基礎的政策。

您可使用以下 IAM 政策範例，在 DynamoDB 中設定特定的備份與還原功能。

範例 1：允許 CreateBackup 和 RestoreTableFromBackup 動作

下列 IAM 政策會授予許可，允許在所有資料表上執行 `CreateBackup` 與 `RestoreTableFromBackup` DynamoDB 動作：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Important**
>
> DynamoDB RestoreTableFromBackup 權限對來源備份而言為必要，而目標資料表的 DynamoDB 讀取和寫入權限對於還原功能而言則為必要。
> DynamoDB RestoreTableToPointInTime 權限對來源資料表而言為必要，而目標資料表的 DynamoDB 讀取和寫入權限對於還原功能而言則為必要。

範例 2：允許 CreateBackup 並拒絕 RestoreTableFromBackup

下列 IAM 政策授予 `CreateBackup` 動作的許可，並拒絕 `RestoreTableFromBackup` 動作：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:RestoreTableFromBackup"],
            "Resource": "*"
        }

    ]
}
```

範例 3：允許 ListBackups 並拒絕 CreateBackup 及 RestoreTableFromBackup

下列 IAM 政策授予 `ListBackups` 動作的許可，並拒絕 `CreateBackup` 和 `RestoreTableFromBackup` 動作：

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "*"
        }

    ]
}
```

範例 4：允許 ListBackups 並拒絕 DeleteBackup

下列 IAM 政策授予 `ListBackups` 動作的許可，並拒絕 `DeleteBackup` 動作：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:DeleteBackup"],
            "Resource": "*"
        }

    ]
}
```

範例 5：允許所有資源的 RestoreTableFromBackup 和 DescribeBackup，並拒絕特定備份的 DeleteBackup

下列 IAM 政策會授予 `RestoreTableFromBackup` 和 `DescribeBackup` 動作的許可，並拒絕特定備份資源的 `DeleteBackup` 動作：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DeleteBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        }
    ]
```

```
}
```

> **Important**
>
> DynamoDB RestoreTableFromBackup 權限對來源備份而言為必要，而目標資料表的 DynamoDB 讀取和寫入權限對於還原功能而言則為必要。
> DynamoDB RestoreTableToPointInTime 權限對來源資料表而言為必要，而目標資料表的 DynamoDB 讀取和寫入權限對於還原功能而言則為必要。

範例 6：允許特定資料表的 CreateBackup

下列 IAM 政策只授予在 `Movies` 資料表上執行 `CreateBackup` 動作的許可：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/Movies"
            ]
        }
    ]
}
```

範例 7：允許 ListBackups

下列 IAM 政策會授予 `ListBackups` 動作的許可：

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        }
    ]
}
```

> **Important**
>
> 您無法授予對特定資料表執行 ListBackups 動作的許可。

範例 8：允許存取 AWS Backup 功能

您將需要 StartAwsBackupJob 動作的 API 許可，以便成功進行進階功能備份，以及 dynamodb:RestoreTableFromAwsBackup 動作，以成功還原該備份。

下列 IAM 政策授予 AWS Backup 許可，以觸發具有進階功能和還原的備份。另請注意，如果資料表已加密，則政策將需要存取AWS KMS 金鑰。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:StartAwsBackupJob",
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Books"
        },
        {
            "Sid": "AllowRestoreFromAwsBackup",
```

```
            "Effect": "Allow",
            "Action": [
                "dynamodb:RestoreTableFromAwsBackup"
            ],
            "Resource": "*"
        }
    ]
}
```

範例 9：拒絕特定來源資料表的 RestoreTableToPointInTime

以下 IAM 政策拒絕特定來源資料表 `RestoreTableToPointInTime` 動作的權限：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableToPointInTime"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music"
        }
    ]
}
```

範例 10：拒絕為特定來源資料表的所有備份執行 RestoreTableFromBackup

以下 IAM 政策拒絕特定來源資料表中所有備份的 `RestoreTableToPointInTime` 動作權限：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/*"
        }
    ]
}
```

了解備份的 Amazon DynamoDB 帳單

本指南提供關於備份的 DynamoDB 帳單的詳細資訊。我們會分析構成整體成本的各個元件，提供清楚的說明和實際範例。

DynamoDB 提供隨需備份和時間點復原 (PITR) 備份，以協助保護您的 DynamoDB 資料免受災難事件影響，並提供資料封存以供長期保留。

運作方式

DynamoDB 隨需備份為每月計費。如果您在當月的任何特定日期進行備份，您將看到該備份的單一費用，備份將按當月剩餘天數計算 (例如：在 27 日建立備份，您只需要支付該月剩餘幾天的費用，在 27 日收取單一費用)。

如果您將先前進行的備份保留到後續幾個月，您一律會看到該備份在 1 日產生完整一個月的費用。如果在月底之前移除備份，則會根據實際使用量調整費用。

例如，如果您在 7 月 27 日建立備份，並且該備份保留到 8 月，您將看到該備份的費用如下所示：

- 將於 7 月 27 日收取 7 月剩餘天數的費用
- 將於 8 月 1 日收取 8 月整個月的費用
- 將於備份存在的每個後續月份的 1 號收取費用

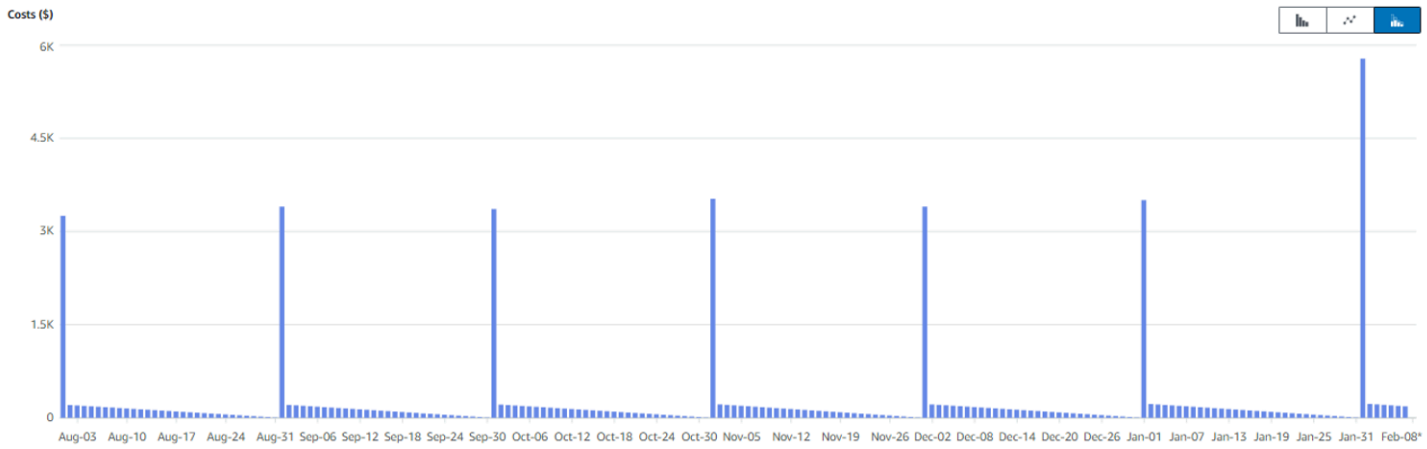
維護 DynamoDB 資料表的備份時，您可能會發現 `DynamoDB (Region)-TimedBackupStorage-ByteHrs` 使用量指標的費用在當月 1 日似乎異常高。此外，如果您在新一個月開始時檢查此指標，並將其與先前的計費週期進行比較，您可能會觀察到使用量似乎大幅遽增的情況。這是設計本身所致。在每月 1 日，任何現有的 DynamoDB 備份都會收取整個月的使用費。在當月移除的任何 DynamoDB

備份都會按比例分配其使用量費用，以反映實際使用量。因此，您可能會看到整個月的費用 (在 1 日收取) 減少。這是因為保留政策會套用過期或手動刪除，以轉移備份。這將在以下案例中探索此狀況。

同樣地，當建立新的備份時，您會發現整個月的峰值較小，其該月剩餘時間的費用會於建立當天收取。

DynamoDB 備份帳單範例

以下是您於當月開始時可能在 Cost Explorer 中看到的內容範例：



請注意 2 月 1 日與前幾個月相比，峰值明顯較大。我們來分析發生此情況的原因。

從 DynamoDB 定價頁面：

「每月計費的總備份儲存體大小，為 DynamoDB 資料表所有備份的總和。DynamoDB 會在一個月內持續監控隨需備份的大小，以判斷您的備份費用。」

這說明了為什麼帳單會在每月 1 日持續顯示使用量大幅遽增的原因。任何即將進入新一個月的現有備份都會收取第一個月的整月費用。換句話說，如果您有 300 個 DynamoDB 備份進入新一個月，您會看到當月第一天針對所有 300 個備份收取完整月份的使用費。

整個月中建立的新備份，從建立當天起到月底，都會產生使用費。

如果在月中刪除備份，該如何計費？

以下有一些案例可供考量：

1. 如果上個月的備份於當月 15 日刪除，仍會在 1 日收取該備份的使用量費用，但該費用將依實際使用量調整，而不是先前收取假設完整一個月的使用量。以下範例提供更詳細的說明。
2. 當您在當月建立備份時，當月剩餘天數的使用費會在建立當日收取。不過，如果您在月底之前刪除此備份，您的使用費將調整為僅包含備份作用中的日期，且仍會在其建立日期收取。

為什麼本月 1 號的使用量似乎比前幾個月高得多？如果我移除備份會發生什麼情況？

為了回答這個重要的兩階段問題，我們使用下列資訊設定案例範例：

- 月長度：30 天
- DynamoDB 備份頻率：10/天、300/月
- DynamoDB 備份保留政策：30 天
- DynamoDB 每次備份成本：2 美元/天、60 美元/月
- 上個月 1 日總計 (TimedBackupStorage-ByteHrs，於當月 1 日檢查)：9,300 美元
- 上個月總計 (TimedBackupStorage-ByteHrs)：18,600 美元
- 本月 1 日總計 (TimedBackupStorage-ByteHrs，於 1 日檢查)：18,000 美元
- DynamoDB 使用量每月變更：無

使用上述資訊，我們可以看到上個月建立了 300 個備份，政策為維持 30 天。在新一個月的第 1 天，所有這些備份仍會保留，因為尚未達到復原期間結束。不過，每過一天就會開始捨棄最舊的備份集，如下所示：

DynamoDB 備份捨棄表

新一個月	第 1 天	第 2 天	第 3 天	第 4 天	第 5 天
轉移的上個月備份總數	300	290	280	270	260

- 在 1 日時，我們可以看到 300 個備份，每個備份為 60 美元/月，TimedBackupStorage-ByteHrs 總計收取 18,000 美元。這與上個月相反，整個月的總計為 18,600 美元。
- 在 2 日時，其中 10 個備份將過期並遭捨棄。發生這種情況時，這些備份適用的費用將根據實際使用量調整，而不是依照假設使用量。這會導致這 10 個備份先前於 1 日收取的 600 美元費用 (10 個備份 x 30 天)，調整為 20 美元 (10 個備份 x 1 天)。
- 到了 2 日，接著有 10 個區塊將過期並遭捨棄，其使用量從 30 天減少為 2 天，費用將減少為 40 美元 (10 個備份 x 2 天)。

每過一天，我們都會看到比上個月更高的峰值開始降低。如果我們延伸此範圍以涵蓋整個月，我們將發現：

DynamoDB 備份費用 (每月 1 日) 進度

10 個區塊中的 300 個備份	第 1 個	第 10 個	第 20 個	第 30 個
區塊 1	600 美元	20 美元	20 美元	20 美元
區塊 2	600 美元	40 美元	40 美元	40 美元
區塊 3	600 美元	60 美元	60 美元	60 美元
區塊 4	600 美元	80 美元	80 美元	80 美元
區塊 5	600 美元	100 美元	100 美元	100 美元
區塊 6	600 美元	120 美元	120 美元	120 美元
區塊 7	600 美元	140 美元	140 美元	140 美元
區塊 8	600 美元	160 美元	160 美元	160 美元
區塊 9	600 美元	180 美元	180 美元	180 美元
區塊 10	600 美元	600 美元	200 美元	200 美元
區塊 11	600 美元	600 美元	220 美元	220 美元
區塊 12	600 美元	600 美元	240 美元	240 美元
區塊 13	600 美元	600 美元	260 美元	260 美元
區塊 14	600 美元	600 美元	280 美元	280 美元
區塊 15	600 美元	600 美元	300 美元	300 美元
區塊 16	600 美元	600 美元	320 美元	320 美元
區塊 17	600 美元	600 美元	340 美元	340 美元
區塊 18	600 美元	600 美元	360 美元	360 美元
區塊 19	600 美元	600 美元	380 美元	380 美元
區塊 20	600 美元	600 美元	600 美元	400 美元

10 個區塊中的 300 個備份	第 1 個	第 10 個	第 20 個	第 30 個
區塊 21	600 美元	600 美元	600 美元	420 美元
區塊 22	600 美元	600 美元	600 美元	440 美元
區塊 23	600 美元	600 美元	600 美元	460 美元
區塊 24	600 美元	600 美元	600 美元	480 美元
區塊 25	600 美元	600 美元	600 美元	500 美元
區塊 26	600 美元	600 美元	600 美元	520 美元
區塊 27	600 美元	600 美元	600 美元	540 美元
區塊 28	600 美元	600 美元	600 美元	560 美元
區塊 29	600 美元	600 美元	600 美元	580 美元
區塊 30	600 美元	600 美元	600 美元	600 美元
每月 1 日總計 ($)	18,000 美元	13,500 美元	10,400 美元	9,300 美元

當每天捨棄新的區塊時，會根據其存在的天數調整其使用量，而不是收取整個月的金額。因此，到了月底，在 1 日看到的費用將從最初的 18,000 美元降至預期的 9,300 美元。這個數字加上整個月新建立的備份 (將計費表與上面類似，但相反)，將產生與上個月 18,600 美元相同的每月費用。

還原 DynamoDB 資料表

您可以使用 AWS 管理主控台、 AWS 命令列界面 (AWS CLI) 或 DynamoDB API，從 PITR 備份或隨需備份還原 DynamoDB 資料表。時間點復原過程會還原到新的 DynamoDB 資料表。

使用時間點復原還原資料表

您可以將資料表還原至任何時間點，直到 `EarliestRestoreableDateTime` 為止。

> **Important**
>
> 如果您在資料表上停用時間點復原後又重新啟用，將會重設可以恢復該資料表的開始時間。因此，您只能使用 `LatestRestorableDateTime` 立即還原該資料表。

使用時間點復原還原時，DynamoDB 會根據所選日期和時間 (日：時：分：秒) 將資料表資料還原到新資料表的狀態。還原資料表時，不會使用資料表的任何佈建輸送量。您可以使用時間點復原執行完整資料表還原，或設定目標資料表設定。您可以變更還原的資料表上的以下資料表設定：

- 全域次要索引 (GSI)
- 本機次要索引 (LSI)
- 帳單模式
- 佈建的讀取與寫入容量
- 加密設定

> **Important**
>
> 當您執行完整資料表還原時，會使用與來源資料表相同的佈建讀取容量單位與寫入容量單位來設定目標資料表，如請求備份時所記錄。例如，假設資料表的佈建輸送量最近降低至 50 個讀取容量單位及 50 個寫入容量單位。然後，您將資料表還原到三週前的狀態，當時的佈建輸送量設為 100 個讀取容量單位和 100 個寫入容量單位。在本例中，DynamoDB 會將資料表的資料還原到該時間點，並使用從該時間開始佈建的輸送量 (100 個讀取容量單位和 100 個寫入容量單位)。

您也可以跨 還原 DynamoDB 資料表資料， AWS 區域 以便在與來源資料表所在的不同區域中建立還原的資料表。您可以在 AWS 商業區域、 AWS 中國區域和 之間進行跨區域還原 AWS GovCloud (US)。您只需為從來源區域中傳輸出來的資料，以及還原為目標區域中的新資料表付費。

> **Note**
>
> 如果來源或目的地區域是亞太地區 (香港) 或中東 (巴林)，則不支援跨區域還原。

如果您阻止在還原的資料表上建立部分或全部索引，可加速還原且更符合經濟效益。您必須在還原的資料表上手動進行下列設定：

- 自動擴展政策
- AWS Identity and Access Management 政策
- Amazon CloudWatch Events 指標和警示
- Tags (標籤)
- 串流設定
- 存留時間 (TTL) 設定
- 時間點復原設定

還原資料表所需的時間會根據多個因素而有所不同，而且不一定與資料表的大小相關。

還原 DynamoDB 資料表至某個時間點

Amazon DynamoDB 時間點復原 (PITR) 可持續備份 DynamoDB 資料表的資料。您可使用 DynamoDB 主控台或 AWS Command Line Interface (AWS CLI)，以將資料表還原至某個時間點。時間點復原過程會還原到新資料表。

如果您想要使用 AWS CLI，您必須先進行設定。如需詳細資訊，請參閱存取 DynamoDB。

主題

- 還原 DynamoDB 資料表至某個時間點 (主控台)
- 還原資料表至某個時間點 (AWS CLI)

還原 DynamoDB 資料表至某個時間點 (主控台)

以下範例示範如何使用 DynamoDB 主控台將名為 `Music` 的現有資料表還原至某個時間點。

> **Note**
>
> 此程序假設您已經啟用時間點復原。若要啟用 `Music` 資料表，在 Backups (備份)索引標籤的 Point-in-time recovery (PITR) (時間點復原) 區段中，選擇Edit (編輯)，然後勾選 Enable point-in-time-recovery (啟用時間點復原)。

還原資料表至某個時間點

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。

2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 在資料表清單中，選擇 `Music` 資料表。
4. 在 `Music` 資料表 Backups (備份) 索引標籤上的 Point-in-time recovery (時間點復原) 區段，選擇 Restore (還原)。
5. 輸入 **`MusicMinutesAgo`** 做為新資料表的名稱。

   > **Note**
   >
   > 您可以將資料表還原至與來源資料表所在的相同 AWS 區域或不同區域。您也可以阻止在還原的資料表上建立次要索引。此外，您可以指定不同的加密模式。

6. 若要確認可還原時間，請將還原日期和時間設為 Earliest (最近)。然後選擇 Restore (還原) 啟動還原程序。

   正在還原的資料表會顯示為 Restoring (正在還原) 狀態。還原程序完成後，`MusicMinutesAgo` 資料表的狀態會變更為 Active (作用中)。

還原資料表至某個時間點 (AWS CLI)

下列程序說明如何使用 AWS CLI 將名為 的現有資料表還原`Music`至某個時間點。

> **Note**
>
> 此程序假設您已經啟用時間點復原。若要針對 `Music` 資料表啟用程序，請執行下列命令。
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name Music \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

還原資料表至某個時間點

1. 使用 `Music` 命令以確認為 `describe-continuous-backups` 資料表啟用了時間點復原。

   ```
   aws dynamodb describe-continuous-backups \
       --table-name Music
   ```

連續備份 (在建立資料表時自動啟用) 和時間點復原已啟用。

```
{
    "ContinuousBackupsDescription": {
        "PointInTimeRecoveryDescription": {
            "PointInTimeRecoveryStatus": "ENABLED",
            "EarliestRestorableDateTime": 1519257118.0,
            "LatestRestorableDateTime": 1520018653.01
        },
        "ContinuousBackupsStatus": "ENABLED"
    }
}
```

2. 還原資料表至某個時間點。在本例中，Music 資料表會還原至相同 AWS 區域的 LatestRestorableDateTime (~5 分鐘前)。

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time
```

> **Note**
>
> 您也可還原至特定時間點。若要執行此作業，請使用 --restore-date-time 引數執行該命令，並指定時間戳記。您可以指定在設定的復原期間內的任何時間點，這可以設定為 1 到 35 天之間的任何值。例如，下列命令可復原資料表至 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
> ```
>
> 當復原到特定時間點時，指定 --no-use-latest-restorable-time 引數是選擇性的。

3. 利用自訂的資料表設定，將資料表還原至某個時間點。在此案例中，Music 資料表會還原至 LatestRestorableDateTime (~5 分鐘以前)。

您可以為還原的資料表指定不同的加密模式，如下所示。

> **Note**
>
> `sse-specification-override` 參數採用的數值與 `CreateTable` 命令中使用的 `sse-specification-override` 參數相同。如需詳細資訊，請參閱 在 DynamoDB 中管理加密資料表。

```
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
```

您可以將資料表還原到與來源資料表所在的不同 AWS 區域。

> **Note**
>
> - 執行跨區域還原必須使用 `sse-specification-override` 參數，而還原至與來源資料表相同的區域時則可選用此參數。
> - 執行跨區域還原必須提供 `source-table-arn` 參數。
> - 從命令列執行跨區域還原時，您必須將預設 AWS 區域設定為所需的目的地區域。如需進一步了解，請參閱《AWS Command Line Interface 使用者指南》中的命令列選項。

```
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
```

您可以覆寫帳單模式及佈建給還原資料表的輸送量。

```
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
```

您可以阻止在還原的資料表上建立部分或全部次要索引。

> **Note**
>
> 如果您阻止在新還原的資料表上建立部分或全部次要索引，可加速還原且更符合經濟效益。

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --global-secondary-index-override '[]'
```

您可以使用不同覆寫的組合。例如，您可以使用單一全域次要索引，同時變更所佈建的輸送量，如下所示。

```
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
```

若要確認還原，請使用 `describe-table` 命令描述 `MusicEarliestRestorableDateTime` 資料表。

```
aws dynamodb describe-table --table-name MusicEarliestRestorableDateTime
```

正在還原的資料表會顯示為 Creating (正在建立) 的狀態，且復原會做為 true 進行。還原程序完成後，`MusicEarliestRestorableDateTime` 資料表的狀態會變更為 Active (作用中)。

> **Important**
>
> 還原進行中時，請勿修改或刪除授予 IAM 實體 （ 例如使用者、群組或角色 ） 執行還原許可的 AWS Identity and Access Management (IAM) 政策。否則，可能會造成意外行為。例如，假設您在資料表還原時移除資料表的寫入許可。在此案例中，基礎 `RestoreTableToPointInTime` 操作無法將任何還原的資料寫入資料表。涉及存取目標還原資料表的來源 IP 限制的 IAM 政策可能也會導致問題。
> 只有在復原操作完成後，才能修改或刪除許可。

搭配 DynamoDB 使用 AWS Backup

Amazon DynamoDB 可協助您透過 AWS Backup 中的加強備份功能符合法規合規和業務持續性需求。AWS Backup 是一項全受管的資料保護服務，可輕鬆跨 AWS 服務、雲端和內部部署環境集中與自動化備份。使用此服務，您可以設定備份政策並監控 AWS 資源集中在一個地方。若要使用 AWS Backup，您必須選擇加入。選擇加入的選項適用於特定帳戶和AWS地區，因此您可能必須使用相同帳戶選擇加入多個地區。如需詳細資訊，請參閱《AWS備份開發人員指南》。

Amazon DynamoDB 與 AWS Backup 自然整合。您可以使用 AWS Backup 自動排程、複製、標記和生命週期您的 DynamoDB 隨需備份。您可以繼續從 DynamoDB 主控台檢視和還原這些備份。您可以使用 DynamoDB 主控台、API 和 AWS 命令列介面 (AWS CLI) 以啟用 DynamoDB 資料表的自動備份。

> **Note**
>
> 透過 DynamoDB 進行的任何備份將保持不變。您仍可以透過目前的 DynamoDB 工作流程建立備份。

透過 AWS Backup 增強的可用備份功能包括：

排程備份-您可以使用備份計劃設定 DynamoDB 資料表的定期排程備份。

跨帳戶和跨區域複製-您可以自動將備份複製到不同AWS地區或帳戶，可讓您支援資料保護需求。

冷儲存分層-您可以設定備份來實作生命週期規則，以刪除備份或將備份轉換至較冷的儲存體。這可協助您最佳化備份成本。

標籤-您可以自動將備份標記為計費和成本分配目的。

加密-透過 AWS Backup 受管的 DynamoDB 隨需備份現在會儲存在 AWS Backup 保存庫。這可讓您使用與 DynamoDB 資料表加密金鑰無關的 AWS KMS key 加密和保護備份。

稽核備份-您可以使用 AWS Backup Audit Manager 來稽核您的 AWS Backup 政策合規性，並尋找尚未與您定義控制項相容的備份活動和資源。您也可以使用它來自動產生每日和隨需報告的稽核軌跡，以滿足您的備份控管目的。

使用 WORM 模式來保護備份— 您可以使用 AWS Backup 文件庫鎖定，以啟用備份的單寫多讀 (WORM) 設定。搭配 AWS Backup Vault Lock，您可以新增額外的防禦層，以保護備份免於意外或惡意刪除作業、備份復原期間的變更，以及生命週期設定的更新。如需進一步了解，請參閱 AWS Backup 保存庫鎖。

這些增強型備份功能於可用於所有 AWS 區域。若要進一步了解這些功能，請參閱《AWS Backup 開發人員指南》。

主題

- 使用 來備份和還原 DynamoDB 資料表 AWS Backup：運作方式
- 使用 AWS Backup 建立 DynamoDB 資料表備份
- 使用 AWS Backup 複製有 DynamoDB 資料表的備份
- 從 AWS Backup 中還原 DynamoDB 資料表備份
- 使用 AWS Backup 刪除 DynamoDB 資料表的備份
- 由 AWS Backup 和 DynamoDB 管理的隨需備份之間的用量備註差異

使用 來備份和還原 DynamoDB 資料表 AWS Backup：運作方式

您可以使用隨需備份功能來建立 Amazon DynamoDB 資料表的完整備份。本節概述備份與還原程序期間所發生的情況。

備份

當您使用 建立隨需備份時 AWS Backup，請求的時間標記會編目。備份以非同步方式建立，它會套用所有變更，直到請求最後一個完整資料表快照為止。

每次您建立隨需備份時，系統都會備份整個資料表資料。您可建立的隨需備份數目不限。

> **Note**
>
> 與 DynamoDB Backups 不同，使用 進行的備份 AWS Backup 並非即時備份。

您無法在備份進行時，執行下列操作：

- 暫停或取消備份操作。
- 刪除備份的來源資料表。
- 在資料表的備份進行時，停用該資料表的備份。

AWS Backup 提供自動備份排程、保留管理和生命週期管理。這消除了對自訂指令碼和手動程序的需求。 會 AWS Backup 執行備份，並在它們過期時將其刪除。如需詳細資訊，請參閱《AWS Backup 開發人員指南》https://docs.aws.amazon.com/aws-backup/latest/devguide/whatisbackup.html。

如果您使用 主控台，使用 建立的任何備份 AWS Backup 都會列在 Backups 索引標籤上，並將 Backup 類型設定為 `AWS_BACKUP`。

> **Note**
>
> 您無法使用 DynamoDB 主控台刪除標示為 Backup type (備份類型) 的 `AWS_BACKUP`。若要管理這些備份，請使用 AWS Backup 主控台。

若要了解如何執行備份，請參閱「備份 DynamoDB 資料表」。

還原

還原資料表時，不會使用資料表的任何佈建輸送量。您可以從 DynamoDB 備份執行完整資料表還原，或設定目的地資料表設定。當您執行還原時，可以變更下列資料表設定：

- 加密設定

> **Important**
>
> 當您執行完整資料表還原時，會使用與來源資料表相同的佈建讀取容量單位與寫入容量單位來設定目標資料表，如請求備份時所記錄。還原程序也會還原本機次要索引與全域次要索引。

您可以將 DynamoDB 資料表資料的備份複製到不同的 AWS 區域，然後在該新區域中還原。您可以在商業區域、 AWS 中國區域和 AWS GovCloud (US) 區域之間 AWS 複製和還原備份。您只需為從來源區域中傳輸出來的資料，以及還原為目標區域中的新資料表付費。

AWS Backup 將還原具有所有原始索引的資料表。

您必須在還原的資料表上手動進行下列設定：

- 自動擴展政策
- AWS Identity and Access Management (IAM) 政策
- Amazon CloudWatch 指標和警示
- Tags (標籤)
- 串流設定
- 存留時間 (TTL) 設定
- 刪除保護設定
- 時間點復原 (PITR) 設定

您只能從備份將整個資料表資料還原至新資料表。只有在還原的資料表變成作用中之後，您才能寫入該資料表。

> **Note**
>
> AWS Backup 還原是非破壞性的。您無法在還原操作期間覆寫現有的資料表。

還原時間與資料表的組態（例如資料表的大小和基礎分割區的數量）和其他相關變數直接相關。規劃災難復原的最佳實務是定期記錄平均還原完成時間，並確認這些時間如何影響整體復原時間點目標。

若要了解如何執行還原，請參閱「從備份中還原 DynamoDB 資料表」。

您可以使用 IAM 政策進行存取控制。如需詳細資訊，請參閱 搭配 IAM 使用 DynamoDB 備份與還原。

所有備份與還原主控台以及 API 動作都會擷取並記錄到 AWS CloudTrail 中，以記錄、持續監控與稽核。

使用 AWS Backup 建立 DynamoDB 資料表備份

本節說明如何開啟 AWS Backup 以從 DynamoDB 資料表建立隨需備份和排程備份。

主題

- 開啟 AWS Backup 功能
- 隨需備份
- 排程備份

開啟 AWS Backup 功能

您必須開啟 AWS Backup 才能與 DynamoDB 搭配使用。

若要開啟 AWS Backup，請進行以下步驟：

1. 登入 AWS 管理主控台，並前往 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 在 [備份設定] 視窗中選擇開啟。
4. 確認畫面隨即出現。選擇開啟功能。

AWS Backup 功能現在可用於您的 DynamoDB 資料表。

如果您在 AWS Backup 功能開啟後選擇關閉它們，請依照下列步驟操作：

1. 登入 AWS 管理主控台，並前往 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 在 [備份設定] 視窗中選擇關閉。
4. 確認畫面隨即出現。選擇關閉功能。

如果您無法開啟或關閉 AWS Backup 功能，可能需由您的 AWS 管理員執行這些操作。

隨需備份

若要建立 DynamoDB 資料表的隨需備份，請依照下列步驟執行：

1. 登入 AWS 管理主控台，並前往 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。

3. 選擇 Create backup (建立備份)。
4. 從顯示的下拉式功能表中，選擇 Create an on-demand backup (建立隨需備份)。
5. 若要使用暖儲存和其他基本功能建立由 AWS Backup 管理的備份，請選擇 Default Settings (預設設定)。若要建立可轉換至冷儲存的備份，或使用 DynamoDB 功能而非 AWS Backup 建立備份，請選擇 Customize settings (自訂設定)。

   若您要改為使用先前的 DynamoDB 功能來建立此備份，請選擇 Customize settings (自訂設定)，然後選擇 Backup with DynamoDB (使用 DynamoDB 進行備份)。
6. 當您完成設定後，選擇 Create backup (建立備份)。

排程備份

若要排程備份，請依照下列步驟執行。

1. 登入 AWS 管理主控台，並前往 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 從顯示的下拉式選單中，選擇使用 AWS Backup 排程備份。
4. 您會被導向至 AWS Backup 以建立備份計畫。

使用 AWS Backup 複製有 DynamoDB 資料表的備份

您可以製作現有備份的複製。您可以視需要將備份複製到多個 AWS 帳戶或 AWS 區域，或自動做為排程備份計劃的一部分。您也可以為 Amazon DynamoDB Encryption Client 自動化一系列跨帳戶和跨區域副本。

如果您有業務持續性或合規性要求，需要將備份儲存在與生產資料最短距離的位置，則跨區域複寫特別有用。

跨帳戶備份對於將備份安全地複製到組織內一個或多個 AWS 帳戶的運作或安全性考量相當有助益。如果不小心刪除您的原始備份，您可以將備份從其目的地帳戶複製到其來源帳戶，然後啟動還原。您必須在組織服務中有兩個屬於相同組織的帳戶，才能執行這項操作。

除非您另有指定，否則複本會繼承來源備份的組態，但有一個例外：如果您指定新副本「永不」過期。使用此設定，新複本仍會繼承其來源到期日。如果您希望新備份副本是永久性的，請將來源備份設定為永不過期，或指定新副本在建立後 100 年過期。

> **Note**
>
> 如果您要複製到其他帳戶，您必須先取得該帳戶的許可。

若要複製備份，請執行以下操作：

1. 登入 AWS 管理主控台，並前往 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 選取要複製的備份旁的核取方塊。
   - 如果要複製的備份顯示為灰色，則必須啟用 AWS Backup 的進階功能。然後建立一個新的備份。現在，您可以將此新備份複製到其他區域和帳戶，以及複製日後的任何其他新備份。
4. 請選擇 Copy (複製)。
5. 若您要複製備份到另一個帳戶或區域，請選取 Copy the recovery point to another destination (將復原點複製到另一個目的地)。然後選擇要複製到您帳戶中的其他區域，還是複製到不同區域的其他帳戶。

   > **Note**
   >
   > 若要將備份還原到另一個區域或帳戶，您必須先將備份複製到該區域或帳戶。

6. 選取要將檔案複製到其中的所需保存庫。如果需要，您也可建立新的備份保存庫。
7. 選擇 Copy backup (複製備份)。

從 AWS Backup 中還原 DynamoDB 資料表備份

本節說明如何從 AWS Backup 還原 DynamoDB 資料表的備份。

主題

- 從 AWS Backup 中還原 DynamoDB 資料表
- 將 DynamoDB 資料表還原至其他區域或帳戶

從 AWS Backup 中還原 DynamoDB 資料表

若要從 AWS Backup 還原您的 DynamoDB 資料表，請依照下列步驟進行：

1. 登入 AWS 管理主控台，並前往 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇 Backups (備份) 索引標籤。
4. 選取您要還原的上一個備份旁的核取方塊。
5. 選擇 Restore (還原)。您會被導向至 Restore table from backup (從備份還原資料表) 畫面。
6. 輸入新還原資料表的名稱、此新資料表將有的加密、要加密還原的金鑰，以及其他選項。
7. 完成時，請選擇 Restore (還原)。

將 DynamoDB 資料表還原至其他區域或帳戶

若要將 DynamoDB 資料表還原到其他區域或帳戶，您必須先將備份複製到新的區域或帳戶。為了複製到另一個帳戶，該帳戶必須先授予您許可。將 DynamoDB 備份複製到新的區域或帳戶後，可以使用上一節中的程序還原該備份。

使用 AWS Backup 刪除 DynamoDB 資料表的備份

本節說明如何使用 AWS Backup 刪除 DynamoDB 資料表的備份。

透過 AWS Backup 功能建立的 DynamoDB 備份儲存在AWSBackup 保存庫。

若要刪除這種備份，請執行以下操作：

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 在接下來的畫面上，選擇繼續前往AWS備份。

   您會導向前往 AWS Backup 主控台。若要進一步了解如何刪除 AWS Backup 主控台 的備份，請參閱刪除備份。

   如需有關 AWS Backup 的詳細資訊，請參閱《AWS 方案指引》中的使用 AWS Backup 進行備份和復原。

由 AWS Backup 和 DynamoDB 管理的隨需備份之間的用量備註差異

本節說明 AWS Backup 和 DynamoDB 管理的隨需備份之間的技術差異。

AWS Backup 有一些與 DynamoDB 不同的工作流程和行為。其中包含：

加密 - 使用 AWS Backup 計劃建立的備份存放在加密的保存庫中，該保存庫具有由 AWS Backup 服務管理的金鑰。保存庫具有存取控制政策，以提供額外的安全性。

備份 ARN - 由 建立的備份檔案現在 AWS Backup 會有 AWS Backup ARN，這可能會影響使用者許可模型。Backup 資源名稱 (ARN) 將從 `arn:aws:dynamodb` 改變為 `arn:aws:backup`。

刪除備份 - 使用 AWS Backup 建立的備份只能從保存 AWS Backup 庫中刪除。您將無法從 DynamoDB 主控台刪除 AWS Backup 檔案。

Backup 程序-與 DynamoDB 備份不同的是，使用 AWS Backup 進行的備份不是即時的。

計費 - 使用 AWS Backup 功能的 DynamoDB 資料表備份會從中計費 AWS Backup。

IAM 角色-如果您是透過 IAM 角色管理存取，則還需要使用下列新許可來設定新的 IAM 角色：

```
"dynamodb:StartAwsBackupJob",
"dynamodb:RestoreTableFromAwsBackup"
```

`dynamodb:StartAwsBackupJob` 使用 AWS Backup 功能成功備份時需要 ，從使用 AWS Backup 功能進行的備份還原`dynamodb:RestoreTableFromAwsBackup`時需要 。

若要在完整的 IAM 政策中檢視這些許可，請參閱使用 IAM 的範例 8。

使用 AWS SDKs DynamoDB 程式碼範例

下列程式碼範例示範如何搭配 AWS 軟體開發套件 (SDK) 使用 DynamoDB。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

AWS 社群貢獻是由多個團隊所建立和維護的範例 AWS。若要提供意見回饋，請使用連結儲存庫中提供的機制。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

程式碼範例

- 使用 AWS SDKs DynamoDB 基本範例
  - Hello DynamoDB
  - 使用 AWS SDK 了解 DynamoDB 的基本概念
  - 使用 AWS SDKs的 DynamoDB 動作
    - BatchExecuteStatement 搭配 AWS SDK 使用
    - BatchGetItem 搭配 AWS SDK 或 CLI 使用
    - BatchWriteItem 搭配 AWS SDK 或 CLI 使用
    - CreateTable 搭配 AWS SDK 或 CLI 使用
    - DeleteItem 搭配 AWS SDK 或 CLI 使用
    - DeleteTable 搭配 AWS SDK 或 CLI 使用
    - DescribeTable 搭配 AWS SDK 或 CLI 使用
    - DescribeTimeToLive 搭配 AWS SDK 或 CLI 使用
    - ExecuteStatement 搭配 AWS SDK 使用
    - GetItem 搭配 AWS SDK 或 CLI 使用
    - ListTables 搭配 AWS SDK 或 CLI 使用
    - PutItem 搭配 AWS SDK 或 CLI 使用
    - Query 搭配 AWS SDK 或 CLI 使用

    - Scan 搭配 AWS SDK 或 CLI 使用
    - UpdateItem 搭配 AWS SDK 或 CLI 使用
    - UpdateTable 搭配 AWS SDK 或 CLI 使用
    - UpdateTimeToLive 搭配 AWS SDK 或 CLI 使用
- DynamoDB AWS SDKs的案例
    - 使用 AWS SDK 透過 DAX 加速 DynamoDB 讀取
    - 使用 AWS Command Line Interface v2 處理進階 DynamoDB 全域次要索引案例
    - 建置應用程式以將資料提交至 DynamoDB 資料表
    - 使用 AWS SDK 比較 DynamoDB 中的多個值與單一屬性
    - 使用 AWS SDK 以 TTL 條件更新 DynamoDB 項目
    - 使用 AWS SDK 連線至本機 DynamoDB 執行個體
    - 使用 AWS SDK 在 DynamoDB 中計數表達式運算子
    - 建立 API Gateway REST API 以追蹤 COVID-19 資料
    - 使用 Step Functions 建立傳訊應用程式
    - 建立相片資產管理應用程式，讓使用者以標籤管理相片
    - 使用 AWS SDK 建立具有全域次要索引的 DynamoDB 資料表
    - 使用 AWS SDK 建立具有暖輸送量設定的 DynamoDB 資料表
    - 建立 Web 應用程式以追蹤 DynamoDB 資料
    - 使用 API Gateway 建立 websocket 聊天應用程式
    - 使用 AWS SDK 建立具有 TTL 的 DynamoDB 項目
    - 使用 AWS SDK 建立和管理具有多區域強一致性的 DynamoDB 全域資料表
    - 使用 AWS SDK 建立和管理示範 MREC 的 DynamoDB 全域資料表
    - 使用 PartiQL DELETE 陳述式搭配 AWS SDK 刪除 DynamoDB 資料
    - 使用 AWS SDK 偵測 Amazon Rekognition 影像中的個人防護裝備
    - 透過 AWS SDK 使用 PartiQL INSERT 陳述式插入 DynamoDB 資料
    - 從瀏覽器調用 Lambda 函式
    - 使用 AWS Command Line Interface v2 管理 DynamoDB 全域次要索引
    - 使用 AWS Command Line Interface v2 管理 DynamoDB 資源型政策
    - 使用 AWS SDK 監控 Amazon DynamoDB 的效能
    - 使用 AWS SDK 執行進階 DynamoDB 查詢操作

- 使用 AWS SDK 在 DynamoDB 中執行清單操作
- 使用 AWS SDK 在 DynamoDB 中執行映射操作
- 使用 AWS SDK 在 DynamoDB 中執行設定操作
- 使用 PartiQL 陳述式批次和 AWS SDK 查詢 DynamoDB 資料表
- 使用 PartiQL 和 AWS SDK 查詢 DynamoDB 資料表
- 透過 AWS SDK 使用全域次要索引查詢 DynamoDB 資料表
- 透過 SDK 使用 start_with 條件查詢 DynamoDB 資料表 AWS
- 使用 SDK 排序索引鍵中的日期範圍查詢 DynamoDB 資料表 AWS
- 使用 AWS SDK 查詢具有複雜篩選條件表達式的 DynamoDB 資料表
- 使用 AWS SDK 查詢具有動態篩選條件表達式的 DynamoDB 資料表
- 使用 SDK 查詢具有篩選條件表達式和限制的 DynamoDB 資料表 AWS
- 使用 AWS SDK 查詢具有巢狀屬性的 DynamoDB 資料表
- 使用 AWS SDK 查詢具有分頁的 DynamoDB 資料表
- 使用 AWS SDK 查詢具有高度一致讀取的 DynamoDB 資料表
- 透過 AWS SDK 使用 PartiQL SELECT 陳述式查詢 DynamoDB 資料
- 使用 AWS SDK 查詢 DynamoDB 資料表的 TTL 項目
- 透過 AWS SDK 使用日期和時間模式查詢 DynamoDB 資料表
- 使用 AWS SDK 儲存 EXIF 和其他影像資訊
- 使用 AWS Command Line Interface v2 設定 DynamoDB 的屬性型存取控制
- 了解使用 AWS SDK 在 DynamoDB 中更新表達式順序
- 使用 AWS SDK 更新具有暖輸送量的 DynamoDB 資料表設定
- 使用 AWS SDK 使用 TTL 更新 DynamoDB 項目
- 透過 AWS SDK 使用 PartiQL UPDATE 陳述式更新 DynamoDB 資料
- 使用 API Gateway 來調用 Lambda 函式
- 使用 Step Functions 調用 Lambda 函式
- 使用 AWS SDK 為 DynamoDB 使用文件模型
- 使用 AWS SDK 為 DynamoDB 使用高階物件持久性模型
- 搭配 AWS SDK 在 DynamoDB 中使用原子計數器操作
- 搭配 AWS SDK 在 DynamoDB 中使用條件式操作
- 搭配 AWS SDK 在 DynamoDB 中使用表達式屬性名稱

- 使用排程事件來調用 Lambda 函式
- 使用 AWS Command Line Interface v2 使用 DynamoDB 本機次要索引
- 使用 AWS Command Line Interface v2 處理 DynamoDB 串流和Time-to-Live
- 使用 v AWS Command Line Interface 2 搭配 DynamoDB 全域資料表和具有最終一致性 (MREC) 的多區域複寫
- 使用 AWS Command Line Interface v2 使用 DynamoDB 資源標記
- 使用 AWS Command Line Interface v2 使用 DynamoDB 資料表加密

- DynamoDB 的無伺服器範例
  - 使用 DynamoDB 觸發條件調用 Lambda 函式
  - 使用 DynamoDB 觸發條件報告 Lambda 函式的批次項目失敗
- AWS DynamoDB 的社群貢獻
  - 建置和測試無伺服器應用程式

使用 AWS SDKs DynamoDB 基本範例

下列程式碼範例示範如何搭配 AWS SDK 使用 Amazon DynamoDB 的基本功能。

範例

- Hello DynamoDB
- 使用 AWS SDK 了解 DynamoDB 的基本概念
- 使用 AWS SDKs的 DynamoDB 動作
  - BatchExecuteStatement 搭配 AWS SDK 使用
  - BatchGetItem 搭配 AWS SDK 或 CLI 使用
  - BatchWriteItem 搭配 AWS SDK 或 CLI 使用
  - CreateTable 搭配 AWS SDK 或 CLI 使用
  - DeleteItem 搭配 AWS SDK 或 CLI 使用
  - DeleteTable 搭配 AWS SDK 或 CLI 使用
  - DescribeTable 搭配 AWS SDK 或 CLI 使用
  - DescribeTimeToLive 搭配 AWS SDK 或 CLI 使用
  - ExecuteStatement 搭配 AWS SDK 使用
  - GetItem 搭配 AWS SDK 或 CLI 使用

- ListTables 搭配 AWS SDK 或 CLI 使用
- PutItem 搭配 AWS SDK 或 CLI 使用
- Query 搭配 AWS SDK 或 CLI 使用
- Scan 搭配 AWS SDK 或 CLI 使用
- UpdateItem 搭配 AWS SDK 或 CLI 使用
- UpdateTable 搭配 AWS SDK 或 CLI 使用
- UpdateTimeToLive 搭配 AWS SDK 或 CLI 使用

Hello DynamoDB

下列程式碼範例示範如何開始使用 DynamoDB。

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
```

```
   static async Task Main(string[] args)
   {
       // Set up dependency injection for Amazon DynamoDB.
       using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
           .ConfigureServices((_, services) =>
               services.AddAWSService<IAmazonDynamoDB>()
           )
           .Build();

       // Now the client is available for injection.
       var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

       try
       {
           var request = new ListTablesRequest();
           var tableNames = new List<string>();

           var paginatorForTables =
dynamoDbClient.Paginators.ListTables(request);

           await foreach (var tableName in paginatorForTables.TableNames)
           {
               tableNames.Add(tableName);
           }

           Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                             "\nLet's list your DynamoDB tables:");
           tableNames.ForEach(table =>
           {
               Console.WriteLine($"Table: {table}");
           });
       }
       catch (AmazonDynamoDBException ex)
       {
           Console.WriteLine($"An Amazon DynamoDB service error occurred while
listing tables. {ex.Message}");
       }
       catch (Exception ex)
       {
           Console.WriteLine($"An error occurred while listing tables.
{ex.Message}");
       }
   }
```

```
}
```

- 如需 API 的詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 ListTables。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

CMakeLists.txt CMake 檔案的程式碼。

```
# 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})
```

hello_dynamodb.cpp 來源檔案的程式碼。

```
#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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 ListTables。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 ListTables。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

如需在 中使用 DynamoDB 的詳細資訊 適用於 JavaScript 的 AWS SDK，請參閱使用 JavaScript 程式設計 DynamoDB。

```
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;
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 ListTables。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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.")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 ListTables。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 詳細資訊，請參閱 適用於 Ruby 的 AWS SDK API 參考中的 ListTables。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 了解 DynamoDB 的基本概念

下列程式碼範例示範如何：

- 建立可存放電影資料的資料表。
- 放入、取得和更新資料表中的單個電影。
- 將影片資料從範例 JSON 檔案寫入資料表。
- 查詢特定年份發表的電影。
- 掃描某個年份範圍內發表的電影。
- 從資料表刪除電影，然後刪除資料表。

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/// <summary>
/// This example application performs the following basic Amazon DynamoDB
/// functions:
///     CreateTableAsync
///     PutItemAsync
///     UpdateItemAsync
///     BatchWriteItemAsync
///     GetItemAsync
///     DeleteItemAsync
///     Query
///     Scan
///     DeleteItemAsync.
/// </summary>
public class DynamoDbBasics
```

```
{
    public static bool IsInteractive = true;

    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    /// <summary>
    /// The main entry point for the DynamoDB Basics example application.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
                    .AddTransient<DynamoDbWrapper>())
            .Build();

        // Now the wrapper is available for injection.
        var dynamoDbWrapper =
 host.Services.GetRequiredService<DynamoDbWrapper>();

        var tableName = "movie_table";

        var movieFileName = @"movies.json";

        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await dynamoDbWrapper.CreateMovieTableAsync(tableName);

        Console.WriteLine(success
            ? $"\nTable: {tableName} successfully created."
            : $"\nCould not create {tableName}.");

        WaitForEnter();

        // Add a single new movie to the table.
        var newMovie = new Movie
```

```
        {
            Year = 2021,
            Title = "Spider-Man: No Way Home",
        };

        success = await dynamoDbWrapper.PutItemAsync(newMovie, tableName);
        if (success)
        {
            Console.WriteLine($"Added {newMovie.Title} to the table.");
        }
        else
        {
            Console.WriteLine("Could not add movie to table.");
        }

        WaitForEnter();

        // Update the new movie by adding a plot and rank.
        var newInfo = new MovieInfo
        {
            Plot = "With Spider-Man's identity now revealed, Peter asks" +
                   "Doctor Strange for help. When a spell goes wrong, dangerous"
+
                   "foes from other worlds start to appear, forcing Peter to" +
                   "discover what it truly means to be Spider-Man.",
            Rank = 9,
        };

        success = await dynamoDbWrapper.UpdateItemAsync(newMovie, newInfo,
tableName);
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie:
{newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }

        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
        // movies in a JSON file.
```

```
        var itemCount = await dynamoDbWrapper.BatchWriteItemsAsync(movieFileName,
tableName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();

        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };

        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await dynamoDbWrapper.GetItemAsync(lookupMovie, tableName);
        if (item?.Count > 0)
        {
            dynamoDbWrapper.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
            Year = 2010,
        };

        success = await dynamoDbWrapper.DeleteItemAsync(tableName,
movieToDelete);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {movieToDelete.Title}.");
        }
```

```
        WaitForEnter();

        // Use Query to find all the movies released in 2010.
        int findYear = 2010;
        Console.WriteLine($"Movies released in {findYear}");
        var queryCount = await dynamoDbWrapper.QueryMoviesAsync(tableName,
findYear);
        Console.WriteLine($"Found {queryCount} movies released in {findYear}");

        WaitForEnter();

        // Use Scan to get a list of movies from 2001 to 2011.
        int startYear = 2001;
        int endYear = 2011;
        var scanCount = await dynamoDbWrapper.ScanTableAsync(tableName,
startYear, endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
        success = await dynamoDbWrapper.DeleteTableAsync(tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is
complete.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
```

```
    {
        if (!IsInteractive)
        {
            return;
        }

        Console.Clear();
        Console.WriteLine();
        Console.Write(new string(' ', 28));
        Console.WriteLine("DynamoDB Basics Example");
        Console.WriteLine(SepBar);
        Console.WriteLine("This demo application shows the basics of using
DynamoDB with the AWS SDK.");
        Console.WriteLine(SepBar);
        Console.WriteLine("The application does the following:");
        Console.WriteLine("\t1. Creates a table with partition: year and
sort:title.");
        Console.WriteLine("\t2. Adds a single movie to the table.");
        Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
        Console.WriteLine("\t4. Updates the rating and plot of the movie that was
just added.");
        Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
        Console.WriteLine("\t6. Deletes a movie.");
        Console.WriteLine("\t7. Uses QueryAsync to return all movies released in
a given year.");
        Console.WriteLine("\t8. Uses ScanAsync to return all movies released
within a range of years.");
        Console.WriteLine("\t9. Finally, it deletes the table that was just
created.");
        WaitForEnter();
    }

    /// <summary>
    /// Simple method to wait for the Enter key to be pressed.
    /// </summary>
    private static void WaitForEnter()
    {
        if (IsInteractive)
        {
            Console.WriteLine("\nPress <Enter> to continue.");
            Console.WriteLine(SepBar);
            _ = Console.ReadLine();
        }
    }
```

```
}
```

將注入的用戶端用於資料表操作。

```
using System.Text.Json;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;

namespace DynamoDBActions;

/// <summary>
/// Methods of this class perform Amazon DynamoDB operations.
/// </summary>
public class DynamoDbWrapper
{
    private readonly IAmazonDynamoDB _amazonDynamoDB;

    /// <summary>
    /// Constructor for the DynamoDbWrapper class.
    /// </summary>
    /// <param name="amazonDynamoDB">The injected DynamoDB client.</param>
    public DynamoDbWrapper(IAmazonDynamoDB amazonDynamoDB)
    {
        _amazonDynamoDB = amazonDynamoDB;
    }
```

建立包含電影資料的資料表。

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
```

```
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

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

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

            TableStatus status;
```

```
            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

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

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

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

新增單一電影到資料表。

```
    /// <summary>
    /// Adds a new item to the table.
```

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

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

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

更新資料表中的單一項目。

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };
```

```
            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

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

從電影資料表擷取單一項目。

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
```

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

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

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

將一批項目寫入電影資料表。

```
/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });

    // Now return the first 250 entries.
    if (allMovies != null && allMovies.Any())
    {
        moviesList = allMovies.GetRange(0, 250);
    }
    return moviesList;
}

/// <summary>
/// Writes 250 items to the movie table.
/// </summary>
/// <param name="movieFileName">A string containing the full path to
/// the JSON file containing movie data.</param>
/// <param name="tableName">The name of the table to write items to.</param>
/// <returns>A long integer value representing the number of movies
/// imported from the JSON file.</returns>
public async Task<long> BatchWriteItemsAsync(
    string movieFileName, string tableName)
```

```
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of
IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch
write operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

從資料表刪除單一項目。

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };

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

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
```

```
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

在資料表中查詢特定年份發表的電影。

```
    /// <summary>
    /// Queries the table for movies released in a particular year and
    /// then displays the information for the movies returned.
    /// </summary>
    /// <param name="tableName">The name of the table to query.</param>
    /// <param name="year">The release year for which we want to
    /// view movies.</param>
    /// <returns>The number of movies that match the query.</returns>
    public async Task<int> QueryMoviesAsync(string tableName, int year)
    {
        try
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
```

```
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
```

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

在資料表中掃描某個年份範圍內發表的電影。

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</
returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
 AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
 info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
 LastEvaluatedKey.
```

```
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

刪除電影資料表。

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
```

```
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
 deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
 table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
 {tableName}. {ex.Message}");
            return false;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的下列主題。
  - BatchWriteItem
  - CreateTable

- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

DynamoDB 入門案例。

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

此案例中使用的 DynamoDB 函數。

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

此案例中使用的公用程式函數。

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

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    {
        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;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的下列主題。
  - BatchWriteItem
  - CreateTable

- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

執行互動式案例以建立資料表並對其執行動作。

```
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))
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

建立呼叫 DynamoDB 動作的結構和方法。

```
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
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的下列主題。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query

- Scan
- UpdateItem

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立 DynamoDB 資料表。

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

建立 Helper 函數以下載並擷取範例 JSON 檔案。

```
    // 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++;
        }
    }
```

從資料表取得項目。

```
    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);
        }
    }
```

完整範例。

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

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { readFileSync } from "node:fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";
```

```
const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
      // The way your data is accessed determines how you structure your keys.
      // The movies table will be queried for movies by year. It makes sense
      // to make year our partition (HASH) key.
      { AttributeName: "year", KeyType: "HASH" },
      { AttributeName: "title", KeyType: "RANGE" },
    ],
  });
```

```
  log("Creating a table.");
  const createTableResponse = await client.send(createTableCommand);
  log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Add a movie to the table.
   */

  log("Adding a single movie to the table.");
  // PutCommand is the first example usage of 'lib-dynamodb'.
  const putCommand = new PutCommand({
    TableName: tableName,
    Item: {
      // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
1981 }`)
      // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
      year: 1981,
      // The preceding KeySchema defines 'title' as our sort (RANGE) key, so
'title'
      // is required.
      title: "The Evil Dead",
      // Every other attribute is optional.
      info: {
        genres: ["Horror"],
      },
    },
  });
  await docClient.send(putCommand);
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
```

```
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/
latest/developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
      ":vals": ["Comedy"],
    },
    ReturnValues: "ALL_NEW",
  });
  const updateResponse = await docClient.send(updateCommand);
  log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

  /**
   * Delete a movie from the table.
   */

  log("Deleting a single movie from the table.");
  const deleteCommand = new DeleteCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
  });
```

```
  await docClient.send(deleteCommand);
  log("Movie deleted.");

  /**
   * Upload a batch of movies.
   */

  log("Adding movies from local JSON file.");
  const file = readFileSync(
    `${dirname}../../../../resources/sample_files/movies.json`,
  );
  const movies = JSON.parse(file.toString());
  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);
  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        [tableName]: putRequests,
      },
    });

    await docClient.send(command);
  }
  log("Movies added.");

  /**
   * Query for movies by year.
   */

  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
```

```
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log("Scan for movies released between 1980 and 1990");
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression.
 Scan will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
```

```
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  log(`Deleting table ${tableName}.`);
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的下列主題。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立 DynamoDB 資料表。

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

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
```

```
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}
```

建立 Helper 函數以下載並擷取範例 JSON 檔案。

```
// Load data into the table.
suspend fun loadData(
    tableName: String,
    fileName: String,
) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}
```

```
suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

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

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}
```

從資料表取得項目。

```
suspend fun getMovie(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

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

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

完整範例。

```
suspend fun main() {
    val tableName = "Movies"
    val fileName = "../../../resources/sample_files/movies.json"
    val partitionAlias = "#a"

    println("Creating an Amazon DynamoDB table named Movies with a key named id
 and a sort key named title.")
    createScenarioTable(tableName, "year")
    loadData(tableName, fileName)
    getMovie(tableName, "year", "1933")
    scanMovies(tableName)
    val count = queryMovieTable(tableName, "year", partitionAlias)
    println("There are $count Movies released in 2013.")
    deletIssuesTable(tableName)
}

suspend fun createScenarioTable(
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
```

```
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}

// Load data into the table.
suspend fun loadData(
    tableName: String,
    fileName: String,
) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode
```

```
    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}

suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

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

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}

suspend fun getMovie(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
```

```
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

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

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

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

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

suspend fun queryMovieTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = "year"

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.N("2013")
```

```
    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

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

suspend fun scanMovies(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Kotlin API 參考》中的下列主題。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem

- PutItem
- Query
- Scan
- UpdateItem

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

由於此範例使用支援檔案，請務必閱讀 PHP 範例 README.md 檔案中的指引。

```
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);
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的下列主題。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立封裝 DynamoDB 資料表的類別。

```
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
```

建立 Helper 函數以下載並擷取範例 JSON 檔案。

```
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]
```

執行互動式案例以建立資料表並對其執行動作。

```
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}")
```

此案例使用以下協助類別在命令提示中提出問題。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的下列主題。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立封裝 DynamoDB 資料表的類別。

```
# 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
```

建立 Helper 函數以下載並擷取範例 JSON 檔案。

```
  # 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
```

執行互動式案例以建立資料表並對其執行動作。

```
  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
```

- 如需 API 詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的下列主題。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    " Create an Amazon Dynamo DB table.

    TRY.
        DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
        DATA(lo_dyn) = /aws1/cl_dyn_factory=>create( lo_session ).
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        DATA(oo_result) = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
```

```
        " It throws exception if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
 { lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.

    " Describe table
    TRY.
        DATA(lo_table) = lo_dyn->describetable( iv_tablename = iv_table_name ).
        DATA(lv_tablename) = lo_table->get_table( )->ask_tablename( ).
        MESSAGE 'The table name is ' && lv_tablename TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table does not exist' TYPE 'E'.
    ENDTRY.

    " Put items into the table.
    TRY.
        DATA(lo_resp_putitem) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1975' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.5' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s = 'Star
 Wars' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1978' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
```

```
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '8.1' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Speed' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1994' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.9' }| ) ) )
          ) ).
        " TYPE REF TO /AWSEX/CL_AWS1_dyn_PUT_ITEM_OUTPUT
        MESSAGE '3 rows inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Get item from table.
    TRY.
        DATA(lo_resp_getitem) = lo_dyn->getitem(
          iv_tablename                = iv_table_name
          it_key                      = VALUE /aws1/cl_dynattributevalue=>tt_key(
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ) ).
        DATA(lt_attr) = lo_resp_getitem->get_item( ).
        DATA(lo_title) = lt_attr[ key = 'title' ]-value.
        DATA(lo_year) = lt_attr[ key = 'year' ]-value.
        DATA(lo_rating) = lt_attr[ key = 'year' ]-value.
```

```
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Query item from table.
    TRY.
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
             ( NEW /aws1/cl_dynattributevalue( iv_n = '1975' ) ) ).
        DATA(lt_keyconditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
            it_attributevaluelist = lt_attributelist
            iv_comparisonoperator = |EQ|
          ) ) ) ).
        DATA(lo_query_result) = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_keyconditions ).
        DATA(lt_items) = lo_query_result->get_items( ).
        READ TABLE lo_query_result->get_items( ) INTO DATA(lt_item) INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Scan items from table.
    TRY.
        DATA(lo_scan_result) = lo_dyn->scan( iv_tablename = iv_table_name ).
        lt_items = lo_scan_result->get_items( ).
        " Read the first item and display the attributes.
        READ TABLE lo_query_result->get_items( ) INTO lt_item INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
```

```
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Update items from table.
    TRY.
        DATA(lt_attributeupdates) = VALUE /aws1/
cl_dynattrvalueupdate=>tt_attributeupdates(
          ( VALUE /aws1/cl_dynattrvalueupdate=>ts_attributeupdates_maprow(
          key = 'rating' value = NEW /aws1/cl_dynattrvalueupdate(
            io_value  = NEW /aws1/cl_dynattributevalue( iv_n = '7.6' )
            iv_action = |PUT| ) ) ) ).
        DATA(lt_key) = VALUE /aws1/cl_dynattributevalue=>tt_key(
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 '1980' ) ) ) ).
        DATA(lo_resp) = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = lt_key
          it_attributeupdates = lt_attributeupdates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Delete table.
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
```

```
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted as it is in use' TYPE 'E'.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的下列主題。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

一個用於處理對適用於 Swift 的 SDK 之 DynamoDB 呼叫的 Swift 類別。

```
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
        }
    }
}
```

MovieTable 類別用來表示電影的結構。

```
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
    }
}
```

使用 MovieTable 類別存取 DynamoDB 資料庫的程式。

```
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)
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的下列主題。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable

- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDKs的 DynamoDB 動作

下列程式碼範例示範如何使用 AWS SDKs 執行個別 DynamoDB 動作。每個範例均包含 GitHub 的連結，您可以在連結中找到設定和執行程式碼的相關說明。

這些摘錄會呼叫 DynamoDB API，是必須在內容中執行之大型程式的程式碼摘錄。您可以在 DynamoDB AWS SDKs的案例 中查看內容中的動作。

下列範例僅包含最常使用的動作。如需完整清單，請參閱《Amazon DynamoDB API 參考》。

範例

- BatchExecuteStatement 搭配 AWS SDK 使用
- BatchGetItem 搭配 AWS SDK 或 CLI 使用
- BatchWriteItem 搭配 AWS SDK 或 CLI 使用
- CreateTable 搭配 AWS SDK 或 CLI 使用
- DeleteItem 搭配 AWS SDK 或 CLI 使用
- DeleteTable 搭配 AWS SDK 或 CLI 使用
- DescribeTable 搭配 AWS SDK 或 CLI 使用
- DescribeTimeToLive 搭配 AWS SDK 或 CLI 使用
- ExecuteStatement 搭配 AWS SDK 使用
- GetItem 搭配 AWS SDK 或 CLI 使用
- ListTables 搭配 AWS SDK 或 CLI 使用
- PutItem 搭配 AWS SDK 或 CLI 使用
- Query 搭配 AWS SDK 或 CLI 使用

- Scan 搭配 AWS SDK 或 CLI 使用
- UpdateItem 搭配 AWS SDK 或 CLI 使用
- UpdateTable 搭配 AWS SDK 或 CLI 使用
- UpdateTimeToLive 搭配 AWS SDK 或 CLI 使用

`BatchExecuteStatement` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `BatchExecuteStatement`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 使用 PartiQL DELETE 刪除資料
- 使用 PartiQL INSERT 插入資料
- 使用多批 PartiQL 陳述式查詢資料表
- 使用 PartiQL SELECT 查詢資料
- 使用 PartiQL UPDATE 更新資料

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用多批 INSERT 陳述式新增項目。

```
        /// <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!;
            }
        }
```

使用多批 SELECT 陳述式取得項目。

```
        /// <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;
            }

        }
```

使用多批 UPDATE 陳述式更新項目。

```
        /// <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;
        }
```

使用多批 DELETE 陳述式刪除項目。

```
        /// <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;
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 BatchExecuteStatement。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用多批 INSERT 陳述式新增項目。

```
    // 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;
        }
    }
```

使用多批 SELECT 陳述式取得項目。

```
    // 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;
```

```
        }
    }
```

使用多批 UPDATE 陳述式更新項目。

```
   // 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;
       }
   }
```

使用多批 DELETE 陳述式刪除項目。

```
   // 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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 BatchExecuteStatement。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

定義範例的函數接收器結構。

```
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
}
```

使用多批 INSERT 陳述式新增項目。

```
// 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
}
```

使用多批 SELECT 陳述式取得項目。

```
// 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
}
```

使用多批 UPDATE 陳述式更新項目。

```
// 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
}
```

使用多批 DELETE 陳述式刪除項目。

```
// 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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 BatchExecuteStatement。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 PartiQL 建立一批項目。

```
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;
};
```

使用 PartiQL 取得一批項目。

```
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;
};
```

使用 PartiQL 更新一批項目。

```
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;
};
```

使用 PartiQL 刪除一批項目。

```
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;
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 BatchExecuteStatement。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 BatchExecuteStatement。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 BatchExecuteStatement。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 PartiQL 讀取一批項目。

```
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
```

使用 PartiQL 刪除一批項目。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 BatchExecuteStatement。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`BatchGetItem` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `BatchGetItem`。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 適用於 .NET 的 AWS SDK API 參考中的 BatchGetItem。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 BatchGetItem。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! 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;
}
```

- 如需 API 詳細資訊，請參閱 適用於 C++ 的 AWS SDK API 參考中的 BatchGetItem。

CLI

AWS CLI

從資料表擷取多個項目

下列 batch-get-items 範例使用三個一組的 GetItem 請求，從 MusicCollection 資料表讀取多個項目，並請求該操作取用的讀取容量單位數量。命令只會傳回 AlbumTitle 屬性。

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json \
    --return-consumed-capacity TOTAL
```

request-items.json 的內容：

```
{
    "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"
    }
}
```

輸出：

```
{
    "Responses": {
        "MusicCollection": [
            {
                "AlbumTitle": {
                    "S": "Somewhat Famous"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Blue Sky Blues"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Louder Than Ever"
                }
            }
        ]
    },
    "UnprocessedKeys": {},
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 1.5
        }
    ]
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的批次操作。

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 BatchGetItem。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

顯示如何使用服務用戶端取得批次項目。

```
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.");
```

```
        }
    }
}
```

顯示如何使用服務用戶端和分頁程式取得批次項目。

```
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());
            });
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 BatchGetItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 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;
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 BatchGetItem。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 BatchGetItem。

PowerShell

Tools for PowerShell V4

範例 1：從 DynamoDB 資料表 'Music' 和 'Songs' 取得具有 SongTitle "Somewhere Down The Road" 的項目。

```
$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
```

輸出：

```
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
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 BatchGetItem。

Tools for PowerShell V5

範例 1：從 DynamoDB 資料表 'Music' 和 'Songs' 取得具有 SongTitle "Somewhere Down The Road" 的項目。

```
$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
```

輸出：

```
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
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 BatchGetItem。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 BatchGetItem。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 BatchGetItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`BatchWriteItem` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `BatchWriteItem`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

將一批項目寫入電影資料表。

```
/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });

    // Now return the first 250 entries.
    if (allMovies != null && allMovies.Any())
    {
        moviesList = allMovies.GetRange(0, 250);
    }
    return moviesList;
}
```

```
    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of
IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
```

```
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch
 write operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
 {ex.Message}");
            throw;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 BatchWriteItem。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 BatchWriteItem。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! 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;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 BatchWriteItem。

CLI

AWS CLI

將多個項目新增至資料表

下列 batch-write-item 範例使用三個一組的 PutItem 請求，將三個新項目新增至 MusicCollection 資料表。並且請求有關操作時取用的寫入容量單位數量，以及作業修改的任何項目收集的資訊。

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json \
    --return-consumed-capacity INDEXES \
    --return-item-collection-metrics SIZE
```

request-items.json 的內容：

```
{
    "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"}
```

```
                }
            }
        }
    ]
}
```

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的批次操作。

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 BatchWriteItem。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 BatchWriteItem。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用服務用戶端將許多項目插入資料表。

```
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);
        }
    }
}
```

使用增強型用戶端將許多項目插入資料表。

```
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);
                }
        }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 BatchWriteItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 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);
  }
};
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 BatchWriteItem。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 BatchWriteItem。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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;
            }
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 BatchWriteItem。

PowerShell

Tools for PowerShell V4

範例 1：建立新的項目，或將現有項目取代為 DynamoDB 資料表 Music 和 Songs 中的新項目。

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
    AlbumTitle = 'Somewhat Famous'
    Price = 1.94
    Genre = 'Country'
    CriticRating = 10.0
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 BatchWriteItem。

Tools for PowerShell V5

範例 1：建立新的項目，或將現有項目取代為 DynamoDB 資料表 Music 和 Songs 中的新項目。

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
    AlbumTitle = 'Somewhat Famous'
    Price = 1.94
    Genre = 'Country'
    CriticRating = 10.0
```

```
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 BatchWriteItem。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 BatchWriteItem。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 BatchWriteItem。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 BatchWriteItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`CreateTable` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `CreateTable`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 使用 DAX 加速讀取
- 進階全域次要索引案例
- 建立具有全域次要索引的資料表
- 建立啟用暖輸送量的資料表
- 建立和管理示範 MREC 的全域資料表
- 建立和管理 MRSC 全域資料表
- 管理全域次要索引
- 管理資源型政策
- 設定屬性型存取控制
- 使用全域資料表和多區域複寫最終一致性 (MREC)
- 使用本機次要索引
- 使用資源標記
- 使用串流和存留時間

- 使用資料表加密

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

```
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

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

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

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

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

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

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
```

```
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
 table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
 {tableName}. {ex.Message}");
            throw;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateTable。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 CreateTable。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

等待資料表變成作用中的程式碼。

```
//! 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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 CreateTable。

CLI

AWS CLI

範例 1：建立具有標籤的資料表

下列 create-table 範例使用指定的屬性和索引鍵結構描述，建立名為 MusicCollection 的資料表。此資料表使用佈建輸送量，並使用預設 AWS 擁有的 CMK 進行靜態加密。該命令也會運用索引鍵 Owner 和值 blueTeam，將標籤套用至資料表。

```
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
```

輸出：

```
{
    "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"
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 2：在隨需模式下建立資料表

以下範例使用隨需模式 (而非佈建的輸送量模式)，建立名為 MusicCollection 的資料表。這適用於工作負載無法預測的資料表。

```
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
```

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 3：建立資料表，並使用客戶管理的 CMK 進行加密

下列範例建立名為 MusicCollection 的資料表，並使用客戶管理的 CMK 將其加密。

```
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
```

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 4：建立具有本機次要索引的資料表

以下範例使用指定的屬性和索引鍵結構描述，建立名為 MusicCollection 的資料表，其中包含名為 AlbumTitleIndex 的本機次要索引。

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 5：建立具有全域次要索引的資料表

以下範例會建立名為 GameScores 的資料表，以及名為 GameTitleIndex 的全域次要索引。基礎資料表具有分割區索引鍵 UserId 和排序索引鍵 GameTitle，可讓您有效率地找到個別使用者在特定遊戲中的最佳分數，而 GSI 具有分割區索引鍵 GameTitle 和排序索引鍵 TopScore，可讓您快速找到特定遊戲的整體最高分數。

```
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
                }
            }
        ]"
```

輸出：

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

```
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 6：建立具有多個全域次要索引的資料表

以下範例會建立名為 GameScores，且具有兩個全域次要索引的資料表。GSI 結構描述是透過檔案傳遞，而不是在命令列上傳遞。

```
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
```

gsi.json 的內容：

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 7：建立已啟用串流的資料表

下列範例建立名為 GameScores，並啟用 DynamoDB Streams 的資料表。每個項目的新舊映像都會寫入串流。

```
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
```

輸出：

```
{
    "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"
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表基本操作。

範例 8：建立已啟用僅限索引鍵串流的資料表

下列範例建立名為 GameScores，並啟用 DynamoDB Streams 的資料表。只有修改過的項目索引鍵屬性才會寫入串流。

```
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
```

輸出：

```
{
    "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
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的 DynamoDB Streams 的變更資料擷取。

範例 9：建立具有 Standard Infrequent Access 類別的資料表

下列範例會建立名為 GameScores 的資料表，並指派 Standard-Infrequent Access (DynamoDB Standard-IA) 資料表類別。此資料表類別針對主要成本的儲存進行最佳化。

```
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
```

輸出：

```
{
    "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
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的資料表類別。

範例 10：建立已啟用刪除保護的資料表

下列範例會建立名為 GameScores，並啟用刪除保護的資料表。

```
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
```

輸出：

```
{
    "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
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用刪除保護。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 CreateTable。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 CreateTable。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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 "";
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 CreateTable。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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;
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 CreateTable。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 CreateTable。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

```
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

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

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 CreateTable。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立資料表。

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 CreateTable。

PowerShell

Tools for PowerShell V4

範例 1：此範例會建立名為 Thread 的資料表，其主索引鍵包含 'ForumName' (索引鍵類型雜湊) 和 'Subject' (索引鍵類型範圍)。用來建構資料表的結構描述，可以使用 -Schema 參數，依照所示或指定方式輸送至每個 cmdlet。

```
$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
```

輸出：

```
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  : {}
```

範例 2：此範例會建立名為 Thread 的資料表，其主索引鍵包含 'ForumName' (索引鍵類型雜湊) 和 'Subject' (索引鍵類型範圍)。也會定義本機次要索引。本機次要索引的索引鍵會從資料表 (ForumName) 上的主要雜湊索引鍵自動設定。用來建構資料表的結構描述，可以使用 -Schema 參數，依照所示或指定方式輸送至每個 cmdlet。

```
$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
```

輸出：

```
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}
```

範例 3：此範例示範如何使用單一管道建立名為 Thread 的資料表，該資料表的主索引鍵包含 'ForumName' (索引鍵類型雜湊) 和 'Subject' (索引鍵類型範圍) 以及本機次要索引。如果管道或 -Schema 參數未提供新的 TableSchema 物件，Add-DDBKeySchema 和 Add-DDBIndexSchema 會為您建立新的 TableSchema 物件。

```
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
```

輸出：

```
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}
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 CreateTable。

Tools for PowerShell V5

範例 1：此範例會建立名為 Thread 的資料表，其主索引鍵包含 'ForumName' (索引鍵類型雜湊) 和 'Subject' (索引鍵類型範圍)。用來建構資料表的結構描述，可以使用 -Schema 參數，依照所示或指定方式輸送至每個 cmdlet。

```
$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
```

輸出：

```
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  : {}
```

範例 2：此範例會建立名為 Thread 的資料表，其主索引鍵包含 'ForumName' (索引鍵類型雜湊) 和 'Subject' (索引鍵類型範圍)。也會定義本機次要索引。本機次要索引的索引鍵會從資料表 (ForumName) 上的主要雜湊索引鍵自動設定。用來建構資料表的結構描述，可以使用 -Schema 參數，依照所示或指定方式輸送至每個 cmdlet。

```
$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
```

輸出：

```
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}
```

範例 3：此範例示範如何使用單一管道建立名為 Thread 的資料表，該資料表的主索引鍵包含 'ForumName' (索引鍵類型雜湊) 和 'Subject' (索引鍵類型範圍) 以及本機次要索引。如果管道或 -Schema 參數未提供新的 TableSchema 物件，Add-DDBKeySchema 和 Add-DDBIndexSchema 會為您建立新的 TableSchema 物件。

```
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
```

輸出：

```
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}
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 CreateTable。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立用於存放電影資料的資料表。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 CreateTable。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
# 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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 CreateTable。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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))
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 CreateTable。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 CreateTable。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 CreateTable。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DeleteItem` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DeleteItem`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

- 使用條件式操作

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };

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

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
```

```
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteItem。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

}
```

此範例中使用的公用程式函數。

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

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DeleteItem。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

等待資料表變成作用中的程式碼。

```
//! 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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DeleteItem。

CLI

AWS CLI

範例 1：刪除項目

下列 delete-item 範例會從 MusicCollection 資料表中刪除項目，並請求已刪除項目的詳細資訊，以及請求所使用的容量。

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

key.json 的內容：

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Scared of My Shadow"}
}
```

輸出：

```
{
    "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
        ]
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的寫入項目。

範例 2：有條件地刪除項目

下列範例只有在項目 ProductCategory 為 Sporting Goods 或 Gardening Supplies，且其價格介於 500 到 600 之間時，才會從 ProductCatalog 資料表中刪除項目。它會傳回已刪除項目的詳細資訊。

```
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
```

names.json 的內容：

```
{
    "#P": "Price"
}
```

values.json 的內容：

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

輸出：

```
{
    "Attributes": {
        "Id": {
            "N": "456"
        },
        "Price": {
            "N": "550"
        },
        "ProductCategory": {
            "S": "Sporting Goods"
        }
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的寫入項目。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DeleteItem。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DeleteItem。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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);
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DeleteItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 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;
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 DeleteItem。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

從資料表刪除項目。

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

使用 DynamoDB 文件用戶端從資料表刪除項目。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 DeleteItem。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

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

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteItem(request)
        println("Item with key matching $keyVal was deleted")
    }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DeleteItem。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 DeleteItem。

PowerShell

Tools for PowerShell V4

範例 1：移除符合所提供索引鍵的 DynamoDB 項目。

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 DeleteItem。

Tools for PowerShell V5

範例 1：移除符合所提供索引鍵的 DynamoDB 項目。

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 DeleteItem。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

您可以指定條件，僅在符合特定條件時刪除項目。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DeleteItem。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 DeleteItem。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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)),
    }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DeleteItem。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 DeleteItem。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 DeleteItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DeleteTable` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DeleteTable`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 使用 DAX 加速讀取
- 建立和管理 MRSC 全域資料表
- 管理全域次要索引
- 使用全域資料表和多區域複寫最終一致性 (MREC)

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
```

```
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
{tableName}. {ex.Message}");
            return false;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteTable。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DeleteTable。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DeleteTable。

CLI

AWS CLI

刪除資料表

下列 delete-table 範例會刪除 MusicCollection 資料表。

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的刪除資料表。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DeleteTable。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DeleteTable。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DeleteTable。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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;
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 DeleteTable。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 DeleteTable。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DeleteTable。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 DeleteTable。

PowerShell

Tools for PowerShell V4

範例 1：刪除指定的資料表。在繼續操作之前，系統會提示您確認。

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

範例 2：刪除指定的資料表。在繼續操作之前，系統不會提示您進行確認。

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 DeleteTable。

Tools for PowerShell V5

範例 1：刪除指定的資料表。在繼續操作之前，系統會提示您確認。

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

範例 2：刪除指定的資料表。在繼續操作之前，系統不會提示您進行確認。

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 DeleteTable。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DeleteTable。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
# 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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 DeleteTable。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DeleteTable。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 DeleteTable。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 DeleteTable。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DescribeTable` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DescribeTable`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 建立和管理示範 MREC 的全域資料表
- 建立和管理 MRSC 全域資料表
- 使用全域資料表和多區域複寫最終一致性 (MREC)
- 使用串流和存留時間
- 使用資料表加密

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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}");

}
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeTable。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 DescribeTable。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DescribeTable。

CLI

AWS CLI

描述資料表

以下 `describe-table` 範例描述 `MusicCollection` 資料表。

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

輸出：

```
{
    "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
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的描述資料表。

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 DescribeTable。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DescribeTable。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DescribeTable。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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;
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 DescribeTable。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 DescribeTable。

PowerShell

Tools for PowerShell V4

範例 1：傳回指定資料表的詳細資訊。

```
Get-DDBTable -TableName "myTable"
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 DescribeTable。

Tools for PowerShell V5

範例 1：傳回指定資料表的詳細資訊。

```
Get-DDBTable -TableName "myTable"
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 DescribeTable。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DescribeTable。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
# 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
```

- 如需 API 詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 DescribeTable。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
  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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的 AWS SDK API 參考》中的 DescribeTable。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DescribeTimeToLive` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DescribeTimeToLive`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 使用串流和存留時間

CLI

AWS CLI

檢視資料表的存留時間設定

下列 `describe-time-to-live` 範例顯示 `MusicCollection` 資料表的存留時間設定。

```
aws dynamodb describe-time-to-live \
    --table-name MusicCollection
```

輸出：

```
{
    "TimeToLiveDescription": {
        "TimeToLiveStatus": "ENABLED",
        "AttributeName": "ttl"
    }
```

```
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的存留時間。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DescribeTimeToLive。

Java

適用於 Java 2.x 的 SDK

使用 AWS SDK for Java 2.x描述現有 DynamoDB 資料表上的 TTL 組態。

```
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;
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DescribeTimeToLive。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 適用於 JavaScript 的 AWS SDK描述現有 DynamoDB 資料表上的 TTL 組態。

```
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');
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 DescribeTimeToLive。

Python

適用於 Python 的 SDK (Boto3)

使用 適用於 Python (Boto3) 的 AWS SDK描述現有 DynamoDB 資料表上的 TTL 組態。

```
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")
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Python (Boto3) API 參考》中的 DescribeTimeToLive。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`ExecuteStatement` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `ExecuteStatement`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 使用 PartiQL DELETE 刪除資料
- 使用 PartiQL INSERT 插入資料
- 使用 PartiQL 查詢資料表
- 使用 PartiQL SELECT 查詢資料
- 使用 PartiQL UPDATE 更新資料

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 INSERT 陳述式新增項目。

```
        /// <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;
        }
```

使用 SELECT 陳述式取得項目。

```
        /// <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;
        }
```

使用 SELECT 陳述式取得項目清單。

```
        /// <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;
        }
```

使用 UPDATE 陳述式更新項目。

```
        /// <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;
        }
```

使用 DELETE 陳述式刪除單個影片。

```
        /// <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;
        }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 ExecuteStatement。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 INSERT 陳述式新增項目。

```
    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;
```

```
        }
    }
```

使用 SELECT 陳述式取得項目。

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

使用 UPDATE 陳述式更新項目。

```
    //  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;
        }
```

```
    }
```

使用 DELETE 陳述式刪除項目。

```
    // 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;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 ExecuteStatement。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

定義範例的函數接收器結構。

```
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
}
```

使用 INSERT 陳述式新增項目。

```
// 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
}
```

使用 SELECT 陳述式取得項目。

```
// 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
}
```

使用 SELECT 陳述式取得項目清單並推斷結果。

```
// 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
}
```

使用 UPDATE 陳述式更新項目。

```
// 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
}
```

使用 DELETE 陳述式刪除項目。

```
// 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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 ExecuteStatement。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 PartiQL 建立項目。

```
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;
};
```

使用 PartiQL 取得項目。

```
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;
};
```

使用 PartiQL 更新項目。

```
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;
};
```

使用 PartiQL 刪除項目。

```
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;
};
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 ExecuteStatement。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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,
```

```
        ]);
    }
```

- 如需 API 詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 ExecuteStatement。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 ExecuteStatement。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 PartiQL 選取單一項目。

```
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
```

使用 PartiQL 更新單一項目。

```
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
```

使用 PartiQL 新增單一項目。

```
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
```

使用 PartiQL 刪除單一項目。

```
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
```

- 如需 API 詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 ExecuteStatement。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`GetItem` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `GetItem`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 使用 DAX 加速讀取
- 建立和管理 MRSC 全域資料表
- 使用全域資料表和多區域複寫最終一致性 (MREC)

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

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

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
```

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

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 GetItem。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 GetItem。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 GetItem。

CLI

AWS CLI

範例 1：讀取資料表中的項目

下列 `get-item` 範例從 `MusicCollection` 資料表擷取項目。資料表具有 hash-and-range 主索引鍵 (`Artist` 和 `SongTitle`)，因此您必須指定這兩個屬性。該命令也會請求操作所取用讀取容量的相關資訊。

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

key.json 的內容：

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的讀取項目。

範例 2：使用一致讀取來讀取項目

以下範例使用高度一致性讀取，從 MusicCollection 資料表擷取項目。

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

key.json 的內容：

```
{
```

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的讀取項目。

範例 3：擷取特定的項目屬性

以下範例使用投影表達式，僅擷取所需項目的三個屬性。

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

names.json 的內容：

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

```
}
```

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的讀取項目。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 GetItem。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 GetItem。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 DynamoDbClient 從資料表取得項目。

```
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);
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 GetItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 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;
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 GetItem。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

從資料表取得項目。

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

使用 DynamoDB 文件用戶端從資料表取得項目。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 GetItem。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

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

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 GetItem。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 GetItem。

PowerShell

Tools for PowerShell V4

範例 1：使用分割區索引鍵 SongTitle 和排序索引鍵 Artist 傳回 DynamoDB 項目。

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

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

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 GetItem。

Tools for PowerShell V5

範例 1：使用分割區索引鍵 SongTitle 和排序索引鍵 Artist 傳回 DynamoDB 項目。

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

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

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 GetItem。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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"]
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 GetItem。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 GetItem。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 GetItem。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 GetItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`ListTables` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `ListTables`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 設定屬性型存取控制

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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);
    }
```

- 如需 API 詳細資訊，請參閱 適用於 .NET 的 AWS SDK API 參考中的 ListTables。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 ListTables。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! 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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 ListTables。

CLI

AWS CLI

範例 1：列出資料表

下列`list-tables`範例列出與目前 AWS 帳戶和區域相關聯的所有資料表。

```
aws dynamodb list-tables
```

輸出：

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的列出資料表名稱。

範例 2：限制頁面大小

下列範例會傳回所有現有資料表的清單，但每次呼叫只會擷取一個項目，並視需要執行多個呼叫以取得整個清單。在大量資源上執行清單命令時，限制頁面大小非常有用，而這可能導致使用預設頁面大小 1000 時發生「逾時」錯誤。

```
aws dynamodb list-tables \
    --page-size 1
```

輸出：

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的列出資料表名稱。

範例 3：限制傳回的項目數

以下範例將傳回的項目數限制為 2。回應包含用來擷取下一頁結果的 NextToken 值。

```
aws dynamodb list-tables \
    --max-items 2
```

輸出：

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog"
    ],
    "NextToken":
 "abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9"
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的列出資料表名稱。

範例 4：擷取下一頁結果

下列命令使用先前呼叫 `list-tables` 命令的 `NextToken` 值，來擷取另一頁結果。由於此案例中的回應不包含 `NextToken` 值，我們了解已經到達結果的末尾。

```
aws dynamodb list-tables \
    --starting-
token abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9
```

輸出：

```
{
    "TableNames": [
        "Reply",
        "Thread"
    ]
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的列出資料表名稱。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 ListTables。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 ListTables。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 ListTables。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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;
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 ListTables。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 ListTables。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun listAllTables() {
    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.listTables(ListTablesRequest {})
        response.tableNames?.forEach { tableName ->
            println("Table name is $tableName")
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 ListTables。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    public function listTables($exclusiveStartTableName = "", $limit = 100)
    {
        $this->dynamoDbClient->listTables([
            'ExclusiveStartTableName' => $exclusiveStartTableName,
            'Limit' => $limit,
        ]);
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 ListTables。

PowerShell

Tools for PowerShell V4

範例 1：傳回所有資料表的詳細資訊，自動迭代，直到服務指出不存在其他資料表為止。

```
Get-DDBTableList
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 ListTables。

Tools for PowerShell V5

範例 1：傳回所有資料表的詳細資訊，自動迭代，直到服務指出不存在其他資料表為止。

```
Get-DDBTableList
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 ListTables。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 ListTables。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

判斷資料表是否存在。

```
# 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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 ListTables。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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)
}
```

判斷資料表是否存在。

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

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 ListTables。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 ListTables。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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)
    }
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 ListTables。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`PutItem` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `PutItem`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 使用 DAX 加速讀取
- 進階全域次要索引案例
- 建立具有 TTL 的項目
- 建立和管理示範 MREC 的全域資料表
- 建立和管理 MRSC 全域資料表
- 使用條件式操作
- 使用全域資料表和多區域複寫最終一致性 (MREC)
- 使用串流和存留時間

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

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

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
```

```
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 PutItem。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

}
```

此範例中使用的公用程式函數。

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

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 PutItem。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

等待資料表變成作用中的程式碼。

```
//! 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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 PutItem。

CLI

AWS CLI

範例 1：將項目新增至資料表

下列put-item 範例會將新項目加入至 MusicCollection 資料表。

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

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

item.json 的內容：

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的寫入項目。

範例 2：有條件地覆寫資料表中的項目

下列 put-item 範例只有在現有項目具有值為 Greatest Hits 的 AlbumTitle 屬性時，才會覆寫 MusicCollection 資料表中的現有項目。命令會傳回項目的上一個值。

```
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
```

item.json 的內容：

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

names.json 的內容：

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

values.json 的內容：

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

輸出：

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

如果索引鍵已存在，您應該會看到下列輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的寫入項目。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 PutItem。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 PutItem。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 DynamoDbClient 將項目放入資料表。

```
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);
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 PutItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 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;
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 PutItem。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

將項目放入資料表。

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

使用 DynamoDB 文件用戶端將項目放入資料表。

```
// 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);
```

```
  }
});
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 PutItem。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

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

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 PutItem。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        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);
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 PutItem。

PowerShell

Tools for PowerShell V4

範例 1：建立新的項目，或將現有項目取代為新項目。

```
$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
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 PutItem。

Tools for PowerShell V5

範例 1：建立新的項目，或將現有項目取代為新項目。

```
$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
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 PutItem。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 PutItem。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 PutItem。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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,
    })
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 PutItem。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 PutItem。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
    }
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 PutItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`Query` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `Query`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 使用 DAX 加速讀取
- 進階全域次要索引案例
- 比較多個值與單一屬性
- 管理全域次要索引
- 執行進階查詢操作
- 使用 begins_with 條件查詢資料表
- 使用日期範圍查詢資料表

- 使用全域次要索引查詢資料表
- 使用複雜篩選條件表達式查詢資料表
- 使用動態篩選條件表達式查詢資料表
- 使用篩選條件表達式和限制查詢資料表
- 查詢具有巢狀屬性的資料表
- 查詢具有分頁的資料表
- 查詢具有高度一致性讀取的資料表
- 查詢 TTL 項目
- 使用日期和時間模式查詢資料表
- 使用表達式屬性名稱
- 使用本機次要索引

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
```

```
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 Query。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 Query。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! 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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 Query。

CLI

AWS CLI

範例 1：查詢資料表

下列 query 範例會查詢 MusicCollection 資料表中的項目。資料表具有 hash-and-range 主索引鍵 (Artist 和 SongTitle)，但此查詢只會指定雜湊索引鍵值。命令傳回名為 "No One You Know" 的歌手演唱曲目標題。

```
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
```

expression-attributes.json 的內容：

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

輸出：

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

```
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用 DynamoDB 中查詢。

範例 2：使用高度一致性讀取查詢資料表，並以遞減順序遍歷索引

下列範例會執行與第一個範例相同的查詢，但會以相反順序傳回結果，並且使用高度一致性讀取。

```
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
```

expression-attributes.json 的內容：

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

輸出：

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

```
        "CapacityUnits": 1.0
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用 DynamoDB 中查詢。

範例 3：篩選出特定結果

下列範例會查詢 MusicCollection，但會在 AlbumTitle 屬性中排除具有特定值的結果。請注意，這不會影響 ScannedCount 或 ConsumedCapacity，因為僅在項目被讀取後才會套用篩選條件。

```
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
```

values.json 的內容：

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

names.json 的內容：

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

輸出：

```
{
    "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
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用 DynamoDB 中查詢。

範例 4：僅擷取項目計數

下列範例會擷取符合查詢的項目計數，但不會自行擷取任何項目。

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

expression-attributes.json 的內容：

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用 DynamoDB 中查詢。

範例 5：查詢索引

下列範例查詢的是本機次要索引 AlbumTitleIndex。查詢會從已投射至本機次要索引的基底資料表，傳回所有屬性。請注意，查詢本機次要索引或全域次要索引時，也需要使用 table-name 參數提供基底資料表的名稱。

```
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
```

expression-attributes.json 的內容：

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

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用 DynamoDB 中查詢。

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 Query。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 Query。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 DynamoDbClient 查詢資源表。

```
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;
    }
}
```

使用 DynamoDbClient 和次要索引查詢資料表。

```
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);
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 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;
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 Query。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 Query。

PowerShell

Tools for PowerShell V4

範例 1：調用查詢，傳回具有指定之 SongTitle 和 Artist 的 DynamoDB 項目。

```
$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
```

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 Query。

Tools for PowerShell V5

範例 1：調用查詢，傳回具有指定之 SongTitle 和 Artist 的 DynamoDB 項目。

```
$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
```

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 Query。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用索引鍵條件表達式查詢項目。

```
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"]
```

查詢項目並投影以傳回資料子集。

```
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"]
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 Query。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

尋找在指定年份製作的電影。

```
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![])
    }
```

```
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的Query。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 Query。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

Scan 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 Scan。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 使用 DAX 加速讀取

- 比較多個值與單一屬性

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</
returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
 AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
```

```
                ProjectionExpression = "#yr, title, info.actors[0],
info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 Scan。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

此範例中使用的公用程式函數。

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

```
# 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
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 Scan。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! 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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 Scan。

CLI

AWS CLI

掃描資料表

下列 scan 範例會掃描整份 MusicCollection 資料表，然後將結果縮小為歌手演出歌曲 "No One You Know"。對於每個項目，僅傳回專輯標題和歌曲標題。

```
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
```

expression-attribute-names.json 的內容：

```
{
    "#ST": "SongTitle",
    "#AT":"AlbumTitle"
}
```

expression-attribute-values.json 的內容：

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

輸出：

```
{
    "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
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的在 DynamoDB 中使用查詢。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 Scan。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 Scan。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 DynamoDbClient 掃描 Amazon DynamoDB 資料表。

```
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);
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Scan。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 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;
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Scan。

適用於 JavaScript (v2) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// 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 + ")"
      );
    });
  }
});
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》。
- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Scan。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun scanItems(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 Scan。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        $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);
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 Scan。

PowerShell

Tools for PowerShell V4

範例 1：傳回 Music 資料表中的所有項目。

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

輸出：

```
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
```

範例 2：傳回 Music 資料表中 CriticRating 大於或等於九的項目。

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 Scan。

Tools for PowerShell V5

範例 1：傳回 Music 資料表中的所有項目。

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

輸出：

```
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
```

範例 2：傳回 Music 資料表中 CriticRating 大於或等於九的項目。

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

輸出：

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
```

```
SongTitle                     Somewhere Down The Road
AlbumTitle                    Somewhat Famous
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 Scan。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Scan。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 Scan。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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(())
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的Scan。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 Scan。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 Scan。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`UpdateItem` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `UpdateItem`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 有條件地更新項目的 TTL
- 計數表達式運算子
- 建立和管理 MRSC 全域資料表
- 執行清單操作
- 執行映射操作
- 執行集合操作
- 了解更新表達式順序
- 更新項目的 TTL
- 使用原子計數器操作
- 使用條件式操作
- 使用表達式屬性名稱

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/// <summary>
/// Updates an existing item in the movies table.
/// </summary>
```

```
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
```

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

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 UpdateItem。

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

}
```

此範例中使用的公用程式函數。

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

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 UpdateItem。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

等待資料表變成作用中的程式碼。

```
//! 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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 UpdateItem。

CLI

AWS CLI

範例 1：更新資料表中的項目

下列 update-item 範例會更新 MusicCollection 資料表中的項目。它會新增屬性 (Year) 並修改 AlbumTitle 屬性。更新後出現的項目的所有屬性，都會在回應中傳回。

```
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
```

key.json 的內容：

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

expression-attribute-names.json 的內容：

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

expression-attribute-values.json 的內容：

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

輸出：

```
{
    "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
        ]
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的寫入項目。

範例 2：有條件地更新項目

下列範例會更新 MusicCollection 資料表中的項目，但前提是現有項目尚未具備 Year 屬性。

```
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)"
```

key.json 的內容：

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

expression-attribute-names.json 的內容：

```
{
```

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

expression-attribute-values.json 的內容：

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

如果項目已具備 Year 屬性，DynamoDB 會傳回下列輸出。

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的寫入項目。

- 如需 API 的詳細資訊，請參閱《AWS CLI 命令參考》中的 UpdateItem。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 UpdateItem。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 DynamoDbClient 更新資料表中的項目。

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

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 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;
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateItem。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
    name: String,
    updateVal: String,
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] =
        AttributeValueUpdate {
            value = AttributeValue.S(updateVal)
            action = AttributeAction.Put
        }

    val request =
        UpdateItemRequest {
            tableName = tableNameVal
            key = itemKey
            attributeUpdates = updatedValues
        }

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 UpdateItem。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 UpdateItem。

PowerShell

Tools for PowerShell V4

範例 1：使用分割區索引鍵 SongTitle 和排序索引鍵 Artist，將 DynamoDB 項目的類型屬性設定為 'Rap'。

```
$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
```

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 UpdateItem。

Tools for PowerShell V5

範例 1：使用分割區索引鍵 SongTitle 和排序索引鍵 Artist，將 DynamoDB 項目的類型屬性設定為 'Rap'。

```
$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
```

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 UpdateItem。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用更新表達式更新項目。

```
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"]
```

使用包含算術運算的更新表達式更新項目。

```
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"]
```

僅在項目符合特定條件時更新項目。

```
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"]
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 UpdateItem。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 UpdateItem。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 UpdateItem。

Swift

適用於 Swift 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 Swift 的AWS SDK API 參考》中的 UpdateItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`UpdateTable` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `UpdateTable`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 建立和管理示範 MREC 的全域資料表
- 建立和管理 MRSC 全域資料表
- 管理全域次要索引
- 更新資料表的暖輸送量設定
- 使用全域資料表和多區域複寫最終一致性 (MREC)
- 使用資料表加密

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

等待資料表變成作用中的程式碼。

```
//! 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;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 UpdateTable。

CLI

AWS CLI

範例 1：修改資料表的計費模式

下列 update-table 範例會增加 MusicCollection 資料表上佈建的讀取和寫入容量。

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --billing-mode PROVISIONED \
    --provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10
```

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的更新資料表。

範例 2：建立全域次要索引

下列範例會在 MusicCollection 資料表上新增全域次要索引。

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
    --global-secondary-index-updates file://gsi-updates.json
```

gsi-updates.json 的內容：

```
[
    {
        "Create": {
```

```
            "IndexName": "AlbumTitle-index",
            "KeySchema": [
                {
                    "AttributeName": "AlbumTitle",
                    "KeyType": "HASH"
                }
            ],
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 10,
                "WriteCapacityUnits": 10
            },
            "Projection": {
                "ProjectionType": "ALL"
            }
        }
    }
]
```

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的更新資料表。

範例 3：在資料表上啟用 DynamoDB Streams

下列命令會在 MusicCollection 資料表上啟用 DynamoDB Streams。

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_IMAGE
```

輸出：

```
{
    "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"
    }
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的更新資料表。

範例 4：啟用伺服器端加密

下列範例會在 `MusicCollection` 資料表上啟用伺服器端加密。

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --sse-specification Enabled=true,SSEType=KMS
```

輸出：

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

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的更新資料表。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 UpdateTable。

PowerShell

Tools for PowerShell V4

範例 1：更新特定資料表的佈建輸送量。

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 UpdateTable。

Tools for PowerShell V5

範例 1：更新特定資料表的佈建輸送量。

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 UpdateTable。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`UpdateTimeToLive` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `UpdateTimeToLive`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 使用串流和存留時間

CLI

AWS CLI

更新資料表上的存留時間設定

下列 `update-time-to-live` 範例會啟用指定資料表上的存留時間。

```
aws dynamodb update-time-to-live \
    --table-name MusicCollection \
    --time-to-live-specification Enabled=true,AttributeName=ttl
```

輸出：

```
{
    "TimeToLiveSpecification": {
        "Enabled": true,
        "AttributeName": "ttl"
    }
```

```
}
```

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的存留時間。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 UpdateTimeToLive。

Java

適用於 Java 2.x 的 SDK

使用 AWS SDK for Java 2.x在現有的 DynamoDB 資料表上啟用 TTL。

```
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;
        }
    }
```

使用 AWS SDK for Java 2.x在現有的 DynamoDB 資料表上停用 TTL。

```
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;
        }
```

```
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateTimeToLive。

JavaScript

適用於 JavaScript (v3) 的 SDK

在現有的 DynamoDB 資料表上啟用 TTL。

```
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');
```

在現有的 DynamoDB 資料表上停用 TTL。

```
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');
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateTimeToLive。

Python

適用於 Python (Boto3) 的 SDK

在現有的 DynamoDB 資料表上啟用 TTL。

```
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")
```

在現有的 DynamoDB 資料表上停用 TTL。

```
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")
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Python (Boto3) API 參考》中的 UpdateTimeToLive。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

DynamoDB AWS SDKs的案例

下列程式碼範例示範如何在 DynamoDB AWS SDKs中實作常見案例。這些案例會向您展示如何呼叫 DynamoDB 中的多個函數或與其他 AWS 服務組合來完成特定任務。每個案例均包含完整原始碼的連結，您可在連結中找到如何設定和執行程式碼的相關指示。

案例的目標是獲得中等水平的經驗，協助您了解內容中的服務動作。

範例

- 使用 AWS SDK 透過 DAX 加速 DynamoDB 讀取
- 使用 AWS Command Line Interface v2 處理進階 DynamoDB 全域次要索引案例
- 建置應用程式以將資料提交至 DynamoDB 資料表
- 使用 AWS SDK 比較 DynamoDB 中的多個值與單一屬性
- 使用 AWS SDK 以 TTL 條件更新 DynamoDB 項目
- 使用 AWS SDK 連線至本機 DynamoDB 執行個體
- 使用 AWS SDK 在 DynamoDB 中計數表達式運算子
- 建立 API Gateway REST API 以追蹤 COVID-19 資料
- 使用 Step Functions 建立傳訊應用程式
- 建立相片資產管理應用程式，讓使用者以標籤管理相片

- 使用 AWS SDK 建立具有全域次要索引的 DynamoDB 資料表
- 使用 AWS SDK 建立具有暖輸送量設定的 DynamoDB 資料表
- 建立 Web 應用程式以追蹤 DynamoDB 資料
- 使用 API Gateway 建立 websocket 聊天應用程式
- 使用 AWS SDK 建立具有 TTL 的 DynamoDB 項目
- 使用 AWS SDK 建立和管理具有多區域強一致性的 DynamoDB 全域資料表
- 使用 AWS SDK 建立和管理示範 MREC 的 DynamoDB 全域資料表
- 使用 PartiQL DELETE 陳述式搭配 AWS SDK 刪除 DynamoDB 資料
- 使用 AWS SDK 偵測 Amazon Rekognition 影像中的個人防護裝備
- 透過 AWS SDK 使用 PartiQL INSERT 陳述式插入 DynamoDB 資料
- 從瀏覽器調用 Lambda 函式
- 使用 AWS Command Line Interface v2 管理 DynamoDB 全域次要索引
- 使用 AWS Command Line Interface v2 管理 DynamoDB 資源型政策
- 使用 AWS SDK 監控 Amazon DynamoDB 的效能
- 使用 AWS SDK 執行進階 DynamoDB 查詢操作
- 使用 AWS SDK 在 DynamoDB 中執行清單操作
- 使用 AWS SDK 在 DynamoDB 中執行映射操作
- 使用 AWS SDK 在 DynamoDB 中執行設定操作
- 使用 PartiQL 陳述式批次和 AWS SDK 查詢 DynamoDB 資料表
- 使用 PartiQL 和 AWS SDK 查詢 DynamoDB 資料表
- 透過 AWS SDK 使用全域次要索引查詢 DynamoDB 資料表
- 透過 SDK 使用 start_with 條件查詢 DynamoDB 資料表 AWS
- 使用 SDK 排序索引鍵中的日期範圍查詢 DynamoDB 資料表 AWS
- 使用 AWS SDK 查詢具有複雜篩選條件表達式的 DynamoDB 資料表
- 使用 AWS SDK 查詢具有動態篩選條件表達式的 DynamoDB 資料表
- 使用 SDK 查詢具有篩選條件表達式和限制的 DynamoDB 資料表 AWS
- 使用 AWS SDK 查詢具有巢狀屬性的 DynamoDB 資料表
- 使用 AWS SDK 查詢具有分頁的 DynamoDB 資料表
- 使用 AWS SDK 查詢具有高度一致讀取的 DynamoDB 資料表
- 透過 AWS SDK 使用 PartiQL SELECT 陳述式查詢 DynamoDB 資料

- 使用 AWS SDK 查詢 DynamoDB 資料表的 TTL 項目
- 透過 AWS SDK 使用日期和時間模式查詢 DynamoDB 資料表
- 使用 AWS SDK 儲存 EXIF 和其他影像資訊
- 使用 AWS Command Line Interface v2 設定 DynamoDB 的屬性型存取控制
- 了解使用 AWS SDK 在 DynamoDB 中更新表達式順序
- 使用 AWS SDK 更新具有暖輸送量的 DynamoDB 資料表設定
- 使用 AWS SDK 使用 TTL 更新 DynamoDB 項目
- 透過 AWS SDK 使用 PartiQL UPDATE 陳述式更新 DynamoDB 資料
- 使用 API Gateway 來調用 Lambda 函式
- 使用 Step Functions 調用 Lambda 函式
- 使用 AWS SDK 為 DynamoDB 使用文件模型
- 使用 AWS SDK 為 DynamoDB 使用高階物件持久性模型
- 搭配 AWS SDK 在 DynamoDB 中使用原子計數器操作
- 搭配 AWS SDK 在 DynamoDB 中使用條件式操作
- 搭配 AWS SDK 在 DynamoDB 中使用表達式屬性名稱
- 使用排程事件來調用 Lambda 函式
- 使用 AWS Command Line Interface v2 使用 DynamoDB 本機次要索引
- 使用 AWS Command Line Interface v2 處理 DynamoDB 串流和Time-to-Live
- 使用 v AWS Command Line Interface 2 搭配 DynamoDB 全域資料表和具有最終一致性 (MREC) 的多區域複寫
- 使用 AWS Command Line Interface v2 使用 DynamoDB 資源標記
- 使用 AWS Command Line Interface v2 使用 DynamoDB 資料表加密

使用 AWS SDK 透過 DAX 加速 DynamoDB 讀取

以下程式碼範例顯示做法：

- 使用 DAX 和 SDK 用戶端建立資料並將其寫入資料表。
- 使用兩個用戶端取得、查詢和掃描資料表，並比較其效能。

如需詳細資訊，請參閱使用 DynamoDB Accelerator 用戶端開發。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 DAX 或 Boto3 用戶端建立資料表。

```
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.")
```

將測試資料寫入資料表。

```
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)
```

為 DAX 用戶端和 Boto3 用戶端多次迭代取得項目，並報告每個迭代所花費的時間。

```
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}."
    )
```

為 DAX 用戶端和 Boto3 用戶端查詢多次迭代的資料表，並報告每個迭代所花費的時間。

```
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}."
    )
```

為 DAX 用戶端和 Boto3 用戶端掃描多次迭代的資料表，並報告每個迭代所花費的時間。

```
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}."
    )
```

刪除資料表。

```
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!")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的下列主題。
  - CreateTable
  - DeleteTable
  - GetItem

- PutItem
- Query
- Scan

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS Command Line Interface v2 處理進階 DynamoDB 全域次要索引案例

下列程式碼範例示範如何使用進階全域次要索引組態。

- 建立具有多個 GSI 的資料表。
- 建立具有隨需容量和 GSI 的資料表。
- 將項目放入具有多個 GSI 的資料表中。
- 查詢具有不同條件的 GSI。

Bash

AWS CLI 使用 Bash 指令碼

建立具有多個 GSI 的資料表。

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

建立具有隨需容量和 GSI 的資料表。

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

將項目放入具有多個 GSI 的資料表中。

```
# 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"}
    }'
```

從具有多個 GSI 的資料表查詢項目。

```
# 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"}}'
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。

- CreateTable
- PutItem
- Query

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

建置應用程式以將資料提交至 DynamoDB 資料表

下列程式碼範例顯示如何建置應用程式，以將資料提交至 Amazon DynamoDB 資料表，並在使用者更新資料表時通知您。

Java

適用於 Java 2.x 的 SDK

示範如何建立動態 Web 應用程式，以使用 Amazon DynamoDB Java API 提交資料，以及使用 Amazon Simple Notification Service Java API 傳送文字訊息。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- DynamoDB
- Amazon SNS

JavaScript

適用於 JavaScript (v3) 的 SDK

此範例說明如何建置應用程式，讓使用者將資料提交至 Amazon DynamoDB 資料表，以及使用 Amazon Simple Notification Service (Amazon SNS) 傳送文字訊息給管理員。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例也可在 適用於 JavaScript 的 AWS SDK v3 開發人員指南中取得。

此範例中使用的服務

- DynamoDB
- Amazon SNS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 比較 DynamoDB 中的多個值與單一屬性

下列程式碼範例示範如何比較 DynamoDB 中的多個值與單一屬性。

- 使用 IN 運算子來比較多個值與單一屬性。
- 比較 IN 運算子與多個 OR 條件。
- 了解使用 IN 的效能和表達式複雜性優勢。

Java

適用於 Java 2.x 的 SDK

使用 在 DynamoDB 中比較多個值與單一屬性 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;
    }
```

比較多個值的範例用量 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();
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - Query
  - Scan

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 比較多個值與單一屬性 適用於 JavaScript 的 AWS SDK。

```
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
    };
  }
}
```

比較多個值的範例用量 適用於 JavaScript 的 AWS SDK。

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

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的下列主題。
  - Query
  - Scan

Python

適用於 Python 的 SDK (Boto3)

使用 比較多個值與單一屬性 適用於 Python (Boto3) 的 AWS SDK。

```
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)}
```

比較多個值的範例用量 適用於 Python (Boto3) 的 AWS SDK。

```
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")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的下列主題。
  - Query
  - Scan

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 以 TTL 條件更新 DynamoDB 項目

下列程式碼範例示範如何有條件地更新項目的 TTL。

Java

適用於 Java 2.x 的 SDK

使用條件更新資料表中現有的 DynamoDB 項目之 TTL。

```
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;
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用條件更新資料表中現有的 DynamoDB 項目之 TTL。

```
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');
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateItem。

Python

適用於 Python (Boto3) 的 SDK

使用條件更新資料表中現有的 DynamoDB 項目之 TTL。

```
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",
)
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 UpdateItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 連線至本機 DynamoDB 執行個體

下列程式碼範例示範如何覆寫端點 URL 以連線至 DynamoDB 和 AWS SDK 的本機開發部署。

如需詳細資訊，請參閱 DynamoDB 本機版。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/// 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:?}"),
    }
}
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 在 DynamoDB 中計數表達式運算子

下列程式碼範例示範如何在 DynamoDB 中計算表達式運算子。

- 了解 DynamoDB 的 300 運算子限制。
- 在複雜表達式中計數運算子。
- 最佳化表達式以保持在限制內。

Java

適用於 Java 2.x 的 SDK

使用 AWS SDK for Java 2.x示範表達式運算子計數。

```
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;
    }
```

運算式運算子計數的範例用量 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();
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 適用於 JavaScript 的 AWS SDK示範表達式運算子計數。

```
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;
}
```

運算式運算子計數的範例用量 適用於 JavaScript 的 AWS SDK。

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

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateItem。

Python

適用於 Python 的 SDK (Boto3)

使用 適用於 Python (Boto3) 的 AWS SDK示範表達式運算子計數。

```
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
```

運算式運算子計數的範例用量 適用於 Python (Boto3) 的 AWS SDK。

```
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")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 UpdateItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

建立 API Gateway REST API 以追蹤 COVID-19 資料

以下程式碼範例示範如何建立 REST API，此 API 使用虛構資料模擬追蹤美國 COVID-19 每日病例的系統。

Python

適用於 Python 的 SDK (Boto3)

示範如何使用 AWS Chalice 搭配 適用於 Python (Boto3) 的 AWS SDK 來建立使用 Amazon API Gateway AWS Lambda和 Amazon DynamoDB 的無伺服器 REST API。REST API 使用虛構資料模擬追蹤美國 COVID-19 每日病例的系統。了解如何：

- 使用 AWS Chalice 來定義 Lambda 函數中的路由，這些函數稱為處理透過 API Gateway 發出的 REST 請求。
- 使用 Lambda 函式在 DynamoDB 資料表中擷取和存放資料，以便為 REST 請求提供服務。
- 在 AWS CloudFormation 範本中定義資料表結構和安全角色資源。
- 使用 AWS Chalice 和 CloudFormation 封裝和部署所有必要的資源。
- 使用 CloudFormation 清理所有已建立的資源。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- API Gateway
- CloudFormation
- DynamoDB
- Lambda

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 Step Functions 建立傳訊應用程式

下列程式碼範例示範如何建立 AWS Step Functions 訊息應用程式，從資料庫資料表擷取訊息記錄。

Python

適用於 Python 的 SDK (Boto3)

示範如何使用 適用於 Python (Boto3) 的 AWS SDK 搭配 AWS Step Functions 來建立訊息應用程式，從 Amazon DynamoDB 資料表擷取訊息記錄，並使用 Amazon Simple Queue Service (Amazon SQS) 傳送它們。狀態機器會與 AWS Lambda 函數整合，以掃描資料庫是否有未傳送的訊息。

- 建立從 Amazon DynamoDB 資料表擷取和更新訊息記錄的狀態機器。
- 更新狀態機器定義，以便也向 Amazon Simple Queue Service (Amazon SQS) 傳送訊息。
- 開始和停用狀態機器執行。
- 使用服務整合從狀態機器連接至 Lambda、DynamoDB 和 Amazon SQS。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- DynamoDB
- Lambda
- Amazon SQS
- 步驟函數

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

建立相片資產管理應用程式，讓使用者以標籤管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

.NET

適用於 .NET 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

適用於 C++ 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

適用於 Java 2.x 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

適用於 JavaScript (v3) 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDK for Kotlin

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

適用於 PHP 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

適用於 Rust 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 建立具有全域次要索引的 DynamoDB 資料表

下列程式碼範例示範如何建立具有全域次要索引的資料表。

Java

適用於 Java 2.x 的 SDK

使用 建立具有全域次要索引的 DynamoDB 資料表 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();
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 CreateTable。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 建立具有暖輸送量設定的 DynamoDB 資料表

下列程式碼範例示範如何建立啟用暖輸送量的資料表。

Java

適用於 Java 2.x 的 SDK

使用 AWS SDK for Java 2.x建立具有暖輸送量設定的 DynamoDB 資料表。

```
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);
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 CreateTable。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 適用於 JavaScript 的 AWS SDK建立具有暖輸送量設定的 DynamoDB 資料表。

```
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
);
*/
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 CreateTable。

Python

適用於 Python 的 SDK (Boto3)

使用 適用於 Python (Boto3) 的 AWS SDK建立具有暖輸送量設定的 DynamoDB 資料表。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 CreateTable。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

建立 Web 應用程式以追蹤 DynamoDB 資料

下列程式碼範例說明如建立 Web 應用程式追蹤 Amazon DynamoDB 資料表中的工作項目，並且使用 Amazon Simple Email Service (Amazon SES) 傳送報告。

.NET

適用於 .NET 的 SDK

說明如何使用 Amazon DynamoDB .NET API 來建立可追蹤 DynamoDB 工作資料的動態 Web 應用程式。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- DynamoDB
- Amazon SES

Java

適用於 Java 2.x 的 SDK

說明如何使用 Amazon DynamoDB API 來建立可追蹤 DynamoDB 工作資料的動態 Web 應用程式。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- DynamoDB
- Amazon SES

Kotlin

SDK for Kotlin

說明如何使用 Amazon DynamoDB API 來建立可追蹤 DynamoDB 工作資料的動態 Web 應用程式。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- DynamoDB
- Amazon SES

Python

適用於 Python 的 SDK (Boto3)

示範如何使用 適用於 Python (Boto3) 的 AWS SDK 建立 REST 服務，以追蹤 Amazon DynamoDB 中的工作項目，並使用 Amazon Simple Email Service (Amazon SES傳送電子郵件報告。這個範例使用 Flask Web 框架來處理 HTTP 路由，並與 React 網頁整合以呈現功能完整的 Web 應用程式。

- 建置與 整合的 Flask REST 服務 AWS 服務。
- 讀取、寫入和更新 DynamoDB 資料表中儲存的工作項目。
- 使用 Amazon SES 傳送工作項目的電子郵件報告。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的 AWS 程式碼範例儲存庫。

此範例中使用的服務

- DynamoDB
- Amazon SES

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 API Gateway 建立 websocket 聊天應用程式

下列程式碼範例示範如何建立由建置於 Amazon API Gateway 上的 websocket API 提供服務的聊天應用程式。

Python

適用於 Python 的 SDK (Boto3)

示範如何使用 適用於 Python (Boto3) 的 AWS SDK 搭配 Amazon API Gateway V2 來建立與 AWS Lambda 和 Amazon DynamoDB 整合的 Websocket API。

- 建立由 API Gateway 提供服務的 websocket API。
- 定義 Lambda 處理常式，該常式將連接存放在 DynamoDB 中，並將訊息傳送給其他聊天參與者。
- 連接至 websocket 聊天應用程式，並使用 Websockets 套件傳送訊息。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 建立具有 TTL 的 DynamoDB 項目

下列程式碼範例示範如何建立含有 TTL 的項目。

Java

適用於 Java 2.x 的 SDK

```
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;
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 PutItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

```
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');
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 PutItem。

Python

適用於 Python (Boto3) 的 SDK

```
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"
)
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 PutItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 建立和管理具有多區域強一致性的 DynamoDB 全域資料表

下列程式碼範例示範如何建立和管理具有多區域強一致性 (MRSC) 的 DynamoDB 全域資料表。

- 建立具有多區域強一致性的資料表。
- 驗證 MRSC 組態和複本狀態。
- 透過即時讀取測試跨區域的強一致性。
- 透過 MRSC 保證執行條件式寫入。
- 清除 MRSC 全域資料表資源。

Bash

AWS CLI 使用 Bash 指令碼

建立具有多區域強一致性的資料表。

```
# Step 1: Create a new table in us-east-2 (primary region for MRSC)
# Note: Table must be empty when enabling MRSC
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
```

```
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --region us-east-2

# Wait for table to become active
aws dynamodb wait table-exists --table-name MusicTable --region us-east-2

# Step 2: Add replica and witness with Multi-Region Strong Consistency
# MRSC requires exactly three replicas in supported regions
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Create": {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

驗證 MRSC 組態和複本狀態。

```
# Verify the global table configuration and MRSC setting
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

透過跨區域即時讀取測試強一致性。

```
# Write an item to the primary region
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"},"Album":
 {"S":"The Beatles 1967-1970"},"Year": {"N":"1968"}}' \
    --region us-east-2

# Read the item from replica region to verify strong consistency (cannot read or
 write to witness)
# No wait time needed - MRSC provides immediate consistency
echo "Reading from us-east-1 (immediate consistency):"
aws dynamodb get-item \
    --table-name MusicTable \
```

```
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --consistent-read \
    --region us-east-1
```

透過 MRSC 保證執行條件式寫入。

```
# Perform a conditional update from a different region
# This demonstrates that conditions work consistently across all regions
aws dynamodb update-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --update-expression "SET #rating = :rating" \
    --condition-expression "attribute_exists(Artist)" \
    --expression-attribute-names '{"#rating": "Rating"}' \
    --expression-attribute-values '{":rating": {"N":"5"}}' \
    --region us-east-1
```

清除 MRSC 全域資料表資源。

```
# Remove replica tables (must be done before deleting the primary table)
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Delete": {"RegionName": "us-west-2"}}]' \
    --region us-east-2

# Wait for replicas to be deleted
echo "Waiting for replicas to be deleted..."
sleep 30

# Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - CreateTable
  - DeleteTable

- DescribeTable
- GetItem
- PutItem
- UpdateItem
- UpdateTable

Java

適用於 Java 2.x 的 SDK

使用 建立準備好進行 MRSC 轉換的區域資料表 AWS SDK for Java 2.x。

```
   public static CreateTableResponse createRegionalTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating regional table: " + tableName + " (must be
empty for MRSC)");

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
                       .attributeType(ScalarAttributeType.S)
                       .build(),
                   AttributeDefinition.builder()
                       .attributeName("SongTitle")
                       .attributeType(ScalarAttributeType.S)
                       .build())
               .keySchema(
                   KeySchemaElement.builder()
                       .attributeName("Artist")
```

```
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Regional table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create regional table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to create regional table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

使用 將區域資料表轉換為具有複本和見證的 MRSC AWS SDK for Java 2.x。

```
    public static UpdateTableResponse convertToMRSCWithWitness(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
```

```
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Converting table to MRSC with replica in " +
replicaRegion.id() + " and witness in "
                + witnessRegion.id());

            // Create replica update using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .create(CreateReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Create witness update
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .create(CreateGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .multiRegionConsistency(MultiRegionConsistency.STRONG)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC conversion initiated. Status: "
                + response.tableDescription().tableStatus());
            LOGGER.info("UpdateTableResponse full object: " + response);
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to convert table to MRSC: " + tableName + " - "
+ e.getMessage());
```

```
            throw DynamoDbException.builder()
                .message("Failed to convert table to MRSC: " + tableName)
                .cause(e)
                .build();
        }
    }
```

使用 描述 MRSC 全域資料表組態 AWS SDK for Java 2.x。

```
   public static DescribeTableResponse describeMRSCTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Describing MRSC global table: " + tableName);

           DescribeTableRequest request =
               DescribeTableRequest.builder().tableName(tableName).build();

           DescribeTableResponse response =
dynamoDbClient.describeTable(request);

           LOGGER.info("Table status: " + response.table().tableStatus());
           LOGGER.info("Multi-region consistency: " +
response.table().multiRegionConsistency());

           if (response.table().replicas() != null
               && !response.table().replicas().isEmpty()) {
               LOGGER.info("Number of replicas: " +
response.table().replicas().size());
               response.table()
                   .replicas()
                   .forEach(replica -> LOGGER.info(
                       "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
```

```
            }

            if (response.table().globalTableWitnesses() != null
                && !response.table().globalTableWitnesses().isEmpty()) {
                LOGGER.info("Number of witnesses: "
                    + response.table().globalTableWitnesses().size());
                response.table()
                    .globalTableWitnesses()
                    .forEach(witness -> LOGGER.info(
                        "Witness region: " + witness.regionName() + ", Status: "
+ witness.witnessStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Table not found: " + tableName)
                .cause(e)
                .build();
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to describe table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

新增測試項目，以使用 AWS SDK for Java 2.x驗證 MRSC 強式一致性。

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String album,
        final String year) {
```

```
        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to MRSC global table: " + tableName);

            Map<String, AttributeValue> item = new HashMap<>();
            item.put("Artist", AttributeValue.builder().s(artist).build());
            item.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            if (album != null && !album.trim().isEmpty()) {
                item.put("Album", AttributeValue.builder().s(album).build());
            }
            if (year != null && !year.trim().isEmpty()) {
                item.put("Year", AttributeValue.builder().n(year).build());
            }

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully with strong consistency");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to add test item to table: " + tableName)
                .cause(e)
                .build();
```

```
        }
    }
```

使用 從 MRSC 複本讀取具有一致讀取的項目 AWS SDK for Java 2.x。

```
    public static GetItemResponse getItemWithConsistentRead(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Reading item from MRSC global table with consistent
read: " + tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            GetItemRequest getItemRequest = GetItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .consistentRead(true)
                .build();

            GetItemResponse response = dynamoDbClient.getItem(getItemRequest);

            if (response.hasItem()) {
```

```
                LOGGER.info("Item found with strong consistency - no wait time
needed");
            } else {
                LOGGER.info("Item not found");
            }

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to read item from table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to read item from table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

使用 搭配 MRSC 保證執行條件式更新 AWS SDK for Java 2.x。

```
    public static UpdateItemResponse performConditionalUpdate(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String rating) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }
        if (rating == null || rating.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Rating cannot be null or empty");
        }

        try {
            LOGGER.info("Performing conditional update on MRSC global table: " +
tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            Map<String, String> expressionAttributeNames = new HashMap<>();
            expressionAttributeNames.put("#rating", "Rating");

            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
            expressionAttributeValues.put(
                ":rating", AttributeValue.builder().n(rating).build());

            UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #rating = :rating")
                .conditionExpression("attribute_exists(Artist)")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

            UpdateItemResponse response =
dynamoDbClient.updateItem(updateItemRequest);
            LOGGER.info("Conditional update successful - demonstrates strong
consistency");

            return response;

        } catch (ConditionalCheckFailedException e) {
            LOGGER.warning("Conditional check failed: " + e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to perform conditional update: " + tableName +
" - " + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to perform conditional update: " + tableName)
                .cause(e)
```

```
                .build();
        }
    }
```

等待 MRSC 複本和見證使用 變成作用中 AWS SDK for Java 2.x。

```
    public static void waitForMRSCReplicasActive(
        final DynamoDbClient dynamoDbClient, final String tableName, final int
maxWaitTimeSeconds)
        throws InterruptedException {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (maxWaitTimeSeconds <= 0) {
            throw new IllegalArgumentException("Max wait time must be positive");
        }

        try {
            LOGGER.info("Waiting for MRSC replicas and witnesses to become
active: " + tableName);

            final long startTime = System.currentTimeMillis();
            final long maxWaitTimeMillis = maxWaitTimeSeconds * 1000L;
            int backoffSeconds = 5; // Start with 5 second intervals
            final int maxBackoffSeconds = 30; // Cap at 30 seconds

            while (System.currentTimeMillis() - startTime < maxWaitTimeMillis) {
                DescribeTableResponse response =
describeMRSCTable(dynamoDbClient, tableName);

                boolean allActive = true;
                StringBuilder statusReport = new StringBuilder();

                if (response.table().multiRegionConsistency() == null
                    || !MultiRegionConsistency.STRONG
                        .toString()
```

```
.equals(response.table().multiRegionConsistency().toString())) {
                    allActive = false;
                    statusReport
                        .append("MultiRegionConsistency: ")
                        .append(response.table().multiRegionConsistency())
                        .append(" ");
                }
                if (response.table().replicas() == null
                    || response.table().replicas().isEmpty()) {
                    allActive = false;
                    statusReport.append("No replicas found. ");
                }
                if (response.table().globalTableWitnesses() == null
                    || response.table().globalTableWitnesses().isEmpty()) {
                    allActive = false;
                    statusReport.append("No witnesses found. ");
                }

                // Check table status
                if (!"ACTIVE".equals(response.table().tableStatus().toString()))
{
                    allActive = false;
                    statusReport
                        .append("Table: ")
                        .append(response.table().tableStatus())
                        .append(" ");
                }

                // Check replica status
                if (response.table().replicas() != null) {
                    for (var replica : response.table().replicas()) {
                        if (!"ACTIVE".equals(replica.replicaStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Replica(")
                                .append(replica.regionName())
                                .append("): ")
                                .append(replica.replicaStatus())
                                .append(" ");
                        }
                    }
                }
```

```
                // Check witness status
                if (response.table().globalTableWitnesses() != null) {
                    for (var witness : response.table().globalTableWitnesses()) {
                        if (!"ACTIVE".equals(witness.witnessStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Witness(")
                                .append(witness.regionName())
                                .append("): ")
                                .append(witness.witnessStatus())
                                .append(" ");
                        }
                    }
                }

                if (allActive) {
                    LOGGER.info("All MRSC replicas and witnesses are now active:
" + tableName);
                    return;
                }

                LOGGER.info("Waiting for MRSC components to become active.
Status: " + statusReport.toString());
                LOGGER.info("Next check in " + backoffSeconds + " seconds...");

                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, maxBackoffSeconds);
            }

            throw DynamoDbException.builder()
                .message("Timeout waiting for MRSC replicas to become active
after " + maxWaitTimeSeconds + " seconds")
                .build();

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("Failed to wait for MRSC replicas to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

使用 清除 MRSC 複本和見證 AWS SDK for Java 2.x。

```
   public static UpdateTableResponse cleanupMRSCReplicas(
       final DynamoDbClient dynamoDbClient,
       final String tableName,
       final Region replicaRegion,
       final Region witnessRegion) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (replicaRegion == null) {
           throw new IllegalArgumentException("Replica region cannot be null");
       }
       if (witnessRegion == null) {
           throw new IllegalArgumentException("Witness region cannot be null");
       }

       try {
           LOGGER.info("Cleaning up MRSC replicas and witnesses for table: " +
tableName);

           // Remove replica using ReplicationGroupUpdate
           ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
               .delete(DeleteReplicationGroupMemberAction.builder()
                   .regionName(replicaRegion.id())
                   .build())
               .build();

           // Remove witness
           GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
               .delete(DeleteGlobalTableWitnessGroupMemberAction.builder()
                   .regionName(witnessRegion.id())
                   .build())
               .build();
```

```
            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC cleanup initiated - removing replica and witness.
Response: " + response);

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to cleanup MRSC replicas: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to cleanup MRSC replicas: " + tableName)
                .cause(e)
                .build();
        }
    }
```

使用 完成 MRSC 工作流程示範 AWS SDK for Java 2.x。

```
    public static void demonstrateCompleteMRSCWorkflow(
        final DynamoDbClient primaryClient,
        final DynamoDbClient replicaClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion)
        throws InterruptedException {

        if (primaryClient == null) {
            throw new IllegalArgumentException("Primary DynamoDB client cannot be
null");
        }
        if (replicaClient == null) {
            throw new IllegalArgumentException("Replica DynamoDB client cannot be
null");
        }
```

```
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("=== Starting Complete MRSC Workflow Demonstration ===");

            // Step 1: Create an empty single-Region table
            LOGGER.info("Step 1: Creating empty single-Region table");
            createRegionalTable(primaryClient, tableName);

            // Use the existing GlobalTableOperations method for basic table
waiting
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 2: Convert to MRSC with replica and witness
            LOGGER.info("Step 2: Converting to MRSC with replica and witness");
            convertToMRSCWithWitness(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for MRSC conversion to complete using MRSC-specific waiter
            LOGGER.info("Waiting for MRSC conversion to complete...");
            waitForMRSCReplicasActive(primaryClient, tableName);

            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 3: Verify MRSC configuration
            LOGGER.info("Step 3: Verifying MRSC configuration");
            describeMRSCTable(primaryClient, tableName);

            // Step 4: Test strong consistency with data operations
            LOGGER.info("Step 4: Testing strong consistency with data
operations");
```

```
            // Add test item to primary region
            putTestItem(primaryClient, tableName, "The Beatles", "Hey Jude", "The
Beatles 1967-1970", "1968");

            // Immediately read from replica region (no wait needed with MRSC)
            LOGGER.info("Reading from replica region immediately (strong
consistency):");
            GetItemResponse getResponse =
                getItemWithConsistentRead(replicaClient, tableName, "The
Beatles", "Hey Jude");

            if (getResponse.hasItem()) {
                LOGGER.info("# Strong consistency verified - item immediately
available in replica region");
            } else {
                LOGGER.warning("# Item not found in replica region");
            }

            // Test conditional update from replica region
            LOGGER.info("Testing conditional update from replica region:");
            performConditionalUpdate(replicaClient, tableName, "The Beatles",
"Hey Jude", "5");
            LOGGER.info("# Conditional update successful - demonstrates strong
consistency");

            // Step 5: Cleanup
            LOGGER.info("Step 5: Cleaning up resources");
            cleanupMRSCReplicas(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for cleanup to complete using basic table waiter
            LOGGER.info("Waiting for replica cleanup to complete...");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // "Halt" until replica/witness cleanup is complete
            DescribeTableResponse cleanupVerification =
describeMRSCTable(primaryClient, tableName);
            int backoffSeconds = 5; // Start with 5 second intervals
            while (cleanupVerification.table().multiRegionConsistency() != null)
{
                LOGGER.info("Waiting additional time (" + backoffSeconds + "
seconds) for MRSC cleanup to complete...");
                tempWait(backoffSeconds);
```

```
                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, 30);
                cleanupVerification = describeMRSCTable(primaryClient,
tableName);
            }

            // Delete the primary table
            deleteTable(primaryClient, tableName);

            LOGGER.info("=== MRSC Workflow Demonstration Complete ===");
            LOGGER.info("");
            LOGGER.info("Key benefits of Multi-Region Strong Consistency
(MRSC):");
            LOGGER.info("- Immediate consistency across all regions (no eventual
consistency delays)");
            LOGGER.info("- Simplified application logic (no need to handle
eventual consistency)");
            LOGGER.info("- Support for conditional writes and transactions across
regions");
            LOGGER.info("- Consistent read operations from any region without
waiting");

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("MRSC workflow failed: " + e.getMessage());
            throw e;
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 建立和管理示範 MREC 的 DynamoDB 全域資料表

下列程式碼範例示範如何跨多個區域建立和管理具有複本的 DynamoDB 全域資料表。

- 建立具備全域次要索引和 DynamoDB Streams 的資料表。
- 在不同的區域中新增複本以建立全域資料表。
- 從全域資料表移除複本。
- 新增測試項目以驗證跨區域的複寫。
- 描述全域資料表組態和複本狀態。

Java

適用於 Java 2.x 的 SDK

使用 建立具有全域次要索引和 DynamoDB Streams 的資料表 AWS SDK for Java 2.x。

```
   public static CreateTableResponse createTableWithGSI(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
indexName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating table: " + tableName + " with GSI: " +
indexName);

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
```

```
                .attributeDefinitions(
                    AttributeDefinition.builder()
                        .attributeName("Artist")
                        .attributeType(ScalarAttributeType.S)
                        .build(),
                    AttributeDefinition.builder()
                        .attributeName("SongTitle")
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .globalSecondaryIndexes(GlobalSecondaryIndex.builder()
                    .indexName(indexName)
                    .keySchema(KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.HASH)
                        .build())
                    .projection(

Projection.builder().projectionType(ProjectionType.ALL).build())
                    .build())
                .streamSpecification(StreamSpecification.builder()
                    .streamEnabled(true)
                    .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
                    .build())
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
```

```
            LOGGER.severe("Failed to create table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

等待資料表使用 變成作用中 AWS SDK for Java 2.x。

```
    public static void waitForTableActive(final DynamoDbClient dynamoDbClient,
final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Waiting for table to become active: " + tableName);

            try (DynamoDbWaiter waiter =
                DynamoDbWaiter.builder().client(dynamoDbClient).build()) {
                DescribeTableRequest request =
                    DescribeTableRequest.builder().tableName(tableName).build();

                waiter.waitUntilTableExists(request);
                LOGGER.info("Table is now active: " + tableName);
            }

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to wait for table to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

新增複本以使用 建立或擴展全域資料表 AWS SDK for Java 2.x。

```
    public static UpdateTableResponse addReplica(
```

```
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final String indexName,
        final Long readCapacity) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (indexName == null || indexName.trim().isEmpty()) {
            throw new IllegalArgumentException("Index name cannot be null or
empty");
        }
        if (readCapacity == null || readCapacity <= 0) {
            throw new IllegalArgumentException("Read capacity must be a positive
number");
        }

        try {
            LOGGER.info("Adding replica in region: " + replicaRegion.id() + " for
table: " + tableName);

            // Create a ReplicationGroupUpdate for adding a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .create(builder -> builder.regionName(replicaRegion.id())
                    .globalSecondaryIndexes(ReplicaGlobalSecondaryIndex.builder()
                        .indexName(indexName)

.provisionedThroughputOverride(ProvisionedThroughputOverride.builder()
                            .readCapacityUnits(readCapacity)
                            .build())
                        .build())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
```

```
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica addition initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

使用 從全域資料表移除複本 AWS SDK for Java 2.x。

```
    public static UpdateTableResponse removeReplica(
        final DynamoDbClient dynamoDbClient, final String tableName, final Region
replicaRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }

        try {
            LOGGER.info("Removing replica in region: " + replicaRegion.id() + "
for table: " + tableName);

            // Create a ReplicationGroupUpdate for removing a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
```

```
                .delete(builder ->
builder.regionName(replicaRegion.id()).build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica removal initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to remove replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

新增測試項目以使用 驗證複寫 AWS SDK for Java 2.x。

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
```

```
        }

        try {
            LOGGER.info("Adding test item to table: " + tableName);

            Map<String,
software.amazon.awssdk.services.dynamodb.model.AttributeValue> item = new
HashMap<>();
            item.put(
                "Artist",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(artist)
                    .build());
            item.put(
                "SongTitle",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(songTitle)
                    .build());

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw e;
        }
    }
```

使用 描述全域資料表組態和複本 AWS SDK for Java 2.x。

```
    public static DescribeTableResponse describeTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
```

```
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);

            LOGGER.info("Table status: " + response.table().tableStatus());
            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

使用 完成全域資料表操作的範例 AWS SDK for Java 2.x。

```
   public static void exampleUsage(final Region sourceRegion, final Region
replicaRegion) {

       String tableName = "Music";
       String indexName = "SongTitleIndex";
       Long readCapacity = 15L;

       // Create DynamoDB client for the source region
       try (DynamoDbClient dynamoDbClient =
           DynamoDbClient.builder().region(sourceRegion).build()) {

           try {
               // Step 1: Create the initial table with GSI and streams
               LOGGER.info("Step 1: Creating table in source region: " +
sourceRegion.id());
               createTableWithGSI(dynamoDbClient, tableName, indexName);

               // Step 2: Wait for table to become active
               LOGGER.info("Step 2: Waiting for table to become active");
               waitForTableActive(dynamoDbClient, tableName);

               // Step 3: Add replica in destination region
               LOGGER.info("Step 3: Adding replica in region: " +
replicaRegion.id());
               addReplica(dynamoDbClient, tableName, replicaRegion, indexName,
readCapacity);

               // Step 4: Wait a moment for replica creation to start
               Thread.sleep(5000);

               // Step 5: Describe table to view replica information
               LOGGER.info("Step 5: Describing table to view replicas");
               describeTable(dynamoDbClient, tableName);

               // Step 6: Add a test item to verify replication
               LOGGER.info("Step 6: Adding test item to verify replication");
               putTestItem(dynamoDbClient, tableName, "TestArtist", "TestSong");

               LOGGER.info("Global table setup completed successfully!");
               LOGGER.info("You can verify replication by checking the item in
region: " + replicaRegion.id());

               // Step 7: Remove replica and clean up table
```

```
                LOGGER.info("Step 7: Removing replica from region: " +
replicaRegion.id());
                removeReplica(dynamoDbClient, tableName, replicaRegion);
                DeleteTableResponse deleteTableResponse =
dynamoDbClient.deleteTable(
                    DeleteTableRequest.builder().tableName(tableName).build());
                LOGGER.info("MREC global table demonstration completed
successfully!");

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Thread was interrupted", e);
            } catch (DynamoDbException e) {
                LOGGER.severe("DynamoDB operation failed: " + e.getMessage());
                throw e;
            }
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - CreateTable
  - DescribeTable
  - PutItem
  - UpdateTable

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 PartiQL DELETE 陳述式搭配 AWS SDK 刪除 DynamoDB 資料

下列程式碼範例示範如何使用 PartiQL DELETE 陳述式刪除資料。

JavaScript

適用於 JavaScript (v3) 的 SDK

搭配 PartiQL DELETE 陳述式從 DynamoDB 資料表刪除項目 適用於 JavaScript 的 AWS SDK。

```
/**
 * 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");
};
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的下列主題。
  - BatchExecuteStatement
  - ExecuteStatement

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 偵測 Amazon Rekognition 影像中的個人防護裝備

以下程式碼範例說明如何建置可使用 Amazon Rekognition 在映像中偵測個人防護裝備 (PPE) 的應用程式。

Java

適用於 Java 2.x 的 SDK

示範如何建立 AWS Lambda 函數，以偵測具有個人防護設備的映像。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

透過 AWS SDK 使用 PartiQL INSERT 陳述式插入 DynamoDB 資料

下列程式碼範例示範如何使用 PartiQL INSERT 陳述式插入資料。

JavaScript

適用於 JavaScript (v3) 的 SDK

搭配 PartiQL INSERT 陳述式將項目插入 DynamoDB 資料表 適用於 JavaScript 的 AWS SDK。

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

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的下列主題。
  - BatchExecuteStatement
  - ExecuteStatement

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

從瀏覽器調用 Lambda 函式

下列程式碼範例示範如何從瀏覽器叫用 AWS Lambda 函數。

JavaScript

適用於 JavaScript (v2) 的 SDK

您可以建立瀏覽器型應用程式，該應用程式使用 AWS Lambda 函數以使用者選擇更新 Amazon DynamoDB 資料表。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- DynamoDB
- Lambda

適用於 JavaScript (v3) 的開發套件

您可以建立瀏覽器型應用程式，該應用程式使用 AWS Lambda 函數以使用者選擇更新 Amazon DynamoDB 資料表。此應用程式使用 適用於 JavaScript 的 AWS SDK v3。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- DynamoDB
- Lambda

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS Command Line Interface v2 管理 DynamoDB 全域次要索引

下列程式碼範例示範如何管理全域次要索引的完整生命週期。

- 建立具有全域次要索引的資料表。
- 將新 GSI 新增至現有資料表。
- 更新 (增加) GSI 暖輸送量。
- 使用 GSI 查詢資料。
- 刪除 GSI。

Bash

AWS CLI 使用 Bash 指令碼

建立具有全域次要索引的資料表。

```
# 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}"
```

將新的 (隨需) GSI 新增至現有的資料表。

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

更新 (增加) GSI 暖輸送量。

```
# 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}}}]"
```

使用 GSI 查詢資料。

```
# 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"}}'
```

刪除 GSI。

```
# Delete a GSI from a table
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
        "[{\"Delete\":{\"IndexName\":\"GenreIndex\"}}]"
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - CreateTable
  - DeleteTable
  - Query
  - UpdateTable

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS Command Line Interface v2 管理 DynamoDB 資源型政策

下列程式碼範例示範如何管理 DynamoDB 資料表之資源型政策的完整生命週期。

- 建立具有資源政策的資料表。
- 取得資源政策。
- 更新資源政策。
- 刪除資源政策。

Bash

AWS CLI 使用 Bash 指令碼

建立具有資源政策的資料表。

```
# 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
```

取得資源政策。

```
# 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
```

更新資源政策。

```
# 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
```

刪除資源政策。

```
# Delete the resource-based policy from a table
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - CreateTable
  - DeleteResourcePolicy
  - GetResourcePolicy
  - PutResourcePolicy

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 監控 Amazon DynamoDB 的效能

下列程式碼範例示範如何設定應用程式使用 DynamoDB 來監控效能。

Java

適用於 Java 2.x 的 SDK

此範例示範如何設定 Java 應用程式來監控 DynamoDB 的效能。應用程式會將指標資料傳送至 CloudWatch，您可以在其中監控效能。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- CloudWatch
- DynamoDB

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 執行進階 DynamoDB 查詢操作

下列程式碼範例示範如何在 DynamoDB 中執行進階查詢操作。

- 使用各種篩選和條件技術查詢資料表。
- 為大型結果集實作分頁。
- 使用全域次要索引作為替代存取模式。
- 根據應用程式需求套用一致性控制。

Java

適用於 Java 2.x 的 SDK

使用 進行強式一致讀取的查詢 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;
        }
    }
```

搭配 使用全域次要索引進行查詢 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);
        }
    }
```

使用 查詢分頁 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;
        }
    }
```

使用 查詢複雜篩選條件 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);
    }
```

使用 查詢動態建構的篩選條件表達式 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);
        }
    }
```

使用 查詢篩選條件表達式和限制 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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 進行強式一致讀取的查詢 適用於 JavaScript 的 AWS SDK。

```
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;
  }
}
```

搭配 使用全域次要索引進行查詢 適用於 JavaScript 的 AWS SDK。

```
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;
  }
}
```

使用 查詢分頁 適用於 JavaScript 的 AWS SDK。

```
/**
 * 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 };
```

使用 查詢複雜篩選條件 適用於 JavaScript 的 AWS SDK。

```
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;
  }
}
```

使用 查詢動態建構的篩選條件表達式 適用於 JavaScript 的 AWS SDK。

```
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;
  }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

使用 進行強式一致讀取的查詢 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

搭配 使用全域次要索引進行查詢 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

使用 查詢分頁 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

使用 查詢複雜篩選條件 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

使用 查詢動態建構的篩選條件表達式 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

使用 查詢篩選條件表達式和限制 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 在 DynamoDB 中執行清單操作

下列程式碼範例示範如何在 DynamoDB 中執行清單操作。

- 將元素新增至清單屬性。
- 從清單屬性中移除元素。
- 依索引更新清單中的特定元素。
- 使用清單附加和清單索引函數。

Java

適用於 Java 2.x 的 SDK

使用 示範清單操作 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();
        }
    }
```

清單操作的範例使用方式 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();
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 示範清單操作 適用於 JavaScript 的 AWS SDK。

```
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;
}
```

清單操作的範例使用方式 適用於 JavaScript 的 AWS SDK。

```
/**
 * 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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateItem。

Python

適用於 Python 的 SDK (Boto3)

使用 示範清單操作 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

清單操作的範例使用方式 適用於 Python (Boto3) 的 AWS SDK。

```
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")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 UpdateItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 在 DynamoDB 中執行映射操作

下列程式碼範例示範如何在 DynamoDB 中執行映射操作。

- 在映射結構中新增和更新巢狀屬性。
- 從映射中移除特定欄位。
- 使用深度巢狀映射屬性。

Java

適用於 Java 2.x 的 SDK

使用 示範地圖操作 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();
        }
    }
```

搭配 的映射操作使用範例 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();
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 示範地圖操作 適用於 JavaScript 的 AWS SDK。

```
/**
 * 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();
}
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateItem。

Python

適用於 Python 的 SDK (Boto3)

使用 示範地圖操作 適用於 Python (Boto3) 的 AWS SDK。

```
"""
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()
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 UpdateItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 在 DynamoDB 中執行設定操作

下列程式碼範例示範如何在 DynamoDB 中執行集合運算。

- 將元素新增至集合屬性。
- 從集合屬性中移除元素。
- 搭配集合使用 ADD 和 DELETE 操作。

Java

適用於 Java 2.x 的 SDK

使用 示範設定操作 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();
        }
    }
```

使用 設定操作的範例用量 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();
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 示範設定操作 適用於 JavaScript 的 AWS SDK。

```
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;
}
```

使用 設定操作的範例用量 適用於 JavaScript 的 AWS SDK。

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

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateItem。

Python

適用於 Python 的 SDK (Boto3)

使用 示範設定操作 適用於 Python (Boto3) 的 AWS SDK。

```
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", {}),
    }
```

使用 設定操作的範例用量 適用於 Python (Boto3) 的 AWS SDK。

```
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")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 UpdateItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 PartiQL 陳述式批次和 AWS SDK 查詢 DynamoDB 資料表

下列程式碼範例示範如何：

- 透過執行多個 SELECT 陳述式取得一批項目。
- 透過執行多個 INSERT 陳述式新增一批項目。
- 透過執行多個 UPDATE 陳述式更新一批項目。
- 透過執行多個 DELETE 陳述式刪除一批項目。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// 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;
        }
```

- 如需 API 的詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 BatchExecuteStatement。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        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;
}
```

- 如需 API 的詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 BatchExecuteStatement。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

執行一個情境，該情境建立資料表並執行多批 PartiQL 查詢。

```
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))
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

建立結構和執行 PartiQL 陳述式的方法。

```
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
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 BatchExecuteStatement。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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);
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 BatchExecuteStatement。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

執行批次 PartiQL 陳述式。

```
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.");
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 BatchExecuteStatement。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun main() {
    val ddb = DynamoDbClient.fromEnvironment { region = "us-east-1" }
    val tableName = "MoviesPartiQBatch"
    println("Creating an Amazon DynamoDB table named $tableName with a key named
 id and a sort key named title.")
    createTablePartiQLBatch(ddb, tableName, "year")
    putRecordBatch(ddb)
    updateTableItemBatchBatch(ddb)
    deleteItemsBatch(ddb)
    deleteTablePartiQLBatch(tableName)
}

suspend fun createTablePartiQLBatch(
    ddb: DynamoDbClient,
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
```

```
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists {
        // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
}

suspend fun putRecordBatch(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE {'year':?,
 'title' : ?, 'info' : ?}"

    // Create three movies to add to the Amazon DynamoDB table.
    val parametersMovie1 = mutableListOf<AttributeValue>()
    parametersMovie1.add(AttributeValue.N("2022"))
    parametersMovie1.add(AttributeValue.S("My Movie 1"))
    parametersMovie1.add(AttributeValue.S("No Information"))

    val statementRequestMovie1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie1
        }

    // Set data for Movie 2.
    val parametersMovie2 = mutableListOf<AttributeValue>()
```

```
    parametersMovie2.add(AttributeValue.N("2022"))
    parametersMovie2.add(AttributeValue.S("My Movie 2"))
    parametersMovie2.add(AttributeValue.S("No Information"))

    val statementRequestMovie2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie2
        }

    // Set data for Movie 3.
    val parametersMovie3 = mutableListOf<AttributeValue>()
    parametersMovie3.add(AttributeValue.N("2022"))
    parametersMovie3.add(AttributeValue.S("My Movie 3"))
    parametersMovie3.add(AttributeValue.S("No Information"))

    val statementRequestMovie3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestMovie1)
    myBatchStatementList.add(statementRequestMovie2)
    myBatchStatementList.add(statementRequestMovie3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }
    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: " + response.toString())
    println("Added new movies using a batch command.")
}

suspend fun updateTableItemBatchBatch(ddb: DynamoDbClient) {
    val sqlStatement =
        "UPDATE MoviesPartiQBatch SET info = 'directors\":[\"Merian C. Cooper\",
\"Ernest B. Schoedsack' where year=? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))
```

```
    val statementRequestRec1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec1
        }

    // Update record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec2
        }

    // Update record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }

    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: $response")
    println("Updated three movies using a batch command.")
    println("Items were updated!")
}

suspend fun deleteItemsBatch(ddb: DynamoDbClient) {
```

```
    // Specify three records to delete.
    val sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year = ? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))

    val statementRequestRec1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec1
        }

    // Specify record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec2
        }

    // Specify record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }

    ddb.batchExecuteStatement(batchRequest)
```

```
    println("Deleted three movies using a batch command.")
}

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

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

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 BatchExecuteStatement。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 BatchExecuteStatement。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立一個類別，該類別可以執行多批 PartiQL 陳述式。

```
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
```

執行一個情境，該情境建立資料表並批次執行 PartiQL 查詢。

```
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}")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 BatchExecuteStatement。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

執行一個情境，該情境會建立資料表並執行批次 PartiQL 查詢。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 BatchExecuteStatement。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 PartiQL 和 AWS SDK 查詢 DynamoDB 資料表

下列程式碼範例示範如何：

- 透過執行 SELECT 陳述式取得項目。
- 透過執行 INSERT 陳述式新增項目。
- 透過執行 UPDATE 陳述式更新項目。
- 透過執行 DELETE 陳述式刪除項目。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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;
        }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 ExecuteStatement。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        //  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;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 ExecuteStatement。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

執行一個情境，該情境建立資料表並執行 PartiQL 查詢。

```
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))
}
```

定義此範例中使用的 Movie 結構。

```
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"])
}
```

建立結構和執行 PartiQL 陳述式的方法。

```
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
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 ExecuteStatement。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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());
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 ExecuteStatement。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

執行單一 PartiQL 陳述式。

```
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.");
};
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 ExecuteStatement。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun main() {
    val ddb = DynamoDbClient.fromEnvironment { region = "us-east-1" }
    val tableName = "MoviesPartiQ"
    val fileName = "../../../resources/sample_files/movies.json"
    println("Creating an Amazon DynamoDB table named MoviesPartiQ with a key
 named id and a sort key named title.")
    createTablePartiQL(ddb, tableName, "year")
    loadDataPartiQL(ddb, fileName)

    println("******* Getting data from the MoviesPartiQ table.")
    getMoviePartiQL(ddb)

    println("******* Putting a record into the MoviesPartiQ table.")
    putRecordPartiQL(ddb)

    println("******* Updating a record.")
    updateTableItemPartiQL(ddb)

    println("******* Querying the movies released in 2013.")
    queryTablePartiQL(ddb)

    println("******* Deleting the MoviesPartiQ table.")
    deleteTablePartiQL(tableName)
}

suspend fun createTablePartiQL(
```

```
    ddb: DynamoDbClient,
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists {
        // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
}
```

```
suspend fun loadDataPartiQL(
    ddb: DynamoDbClient,
    fileName: String,
) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode
    var t = 0

    while (iter.hasNext()) {
        if (t == 200) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()

        val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
        parameters.add(AttributeValue.N(year.toString()))
        parameters.add(AttributeValue.S(title))
        parameters.add(AttributeValue.S(info))

        executeStatementPartiQL(ddb, sqlStatement, parameters)
        println("Added Movie $title")
        parameters.clear()
        t++
    }
}

suspend fun getMoviePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and title=?"
    val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
    parameters.add(AttributeValue.N("2012"))
    parameters.add(AttributeValue.S("The Perks of Being a Wallflower"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("ExecuteStatement successful: $response")
}
```

```
suspend fun putRecordPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2020"))
    parameters.add(AttributeValue.S("My Movie"))
    parameters.add(AttributeValue.S("No Info"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Added new movie.")
}

suspend fun updateTableItemPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":[\"Merian C.
 Cooper\",\"Ernest B. Schoedsack\' where year=? and title=?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    parameters.add(AttributeValue.S("The East"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Item was updated!")
}

// Query the table where the year is 2013.
suspend fun queryTablePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year = ?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("ExecuteStatement successful: $response")
}

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

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

suspend fun executeStatementPartiQL(
    ddb: DynamoDbClient,
```

```
    statementVal: String,
    parametersVal: List<AttributeValue>,
): ExecuteStatementResponse {
    val request =
        ExecuteStatementRequest {
            statement = statementVal
            parameters = parametersVal
        }

    return ddb.executeStatement(request)
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 ExecuteStatement。

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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,
        ]);
    }
```

- 如需 API 詳細資訊，請參閱《適用於 PHP 的 AWS SDK API 參考》中的 ExecuteStatement。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立一個類別，該類別可以執行 PartiQL 陳述式。

```
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
```

執行一個情境，該情境建立資料表並執行 PartiQL 查詢。

```
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}")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 ExecuteStatement。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

執行一個情境，該情境建立資料表並執行 PartiQL 查詢。

```
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
```

- 如需 API 詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 ExecuteStatement。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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(())
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 ExecuteStatement。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

透過 AWS SDK 使用全域次要索引查詢 DynamoDB 資料表

下列程式碼範例示範如何使用全域次要索引查詢資料表。

- 使用其主索引鍵查詢 DynamoDB 資料表。
- 查詢全域次要索引 (GSI) 是否有替代存取模式。
- 比較資料表查詢和 GSI 查詢。

Java

適用於 Java 2.x 的 SDK

使用其主索引鍵和全域次要索引 (GSI) 來查詢 DynamoDB 資料表 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);
        }
    }
```

比較直接查詢資料表與使用 查詢 GSI 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);
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用主索引鍵查詢 DynamoDB 資料表 適用於 JavaScript 的 AWS SDK。

```
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;
  }
}
```

使用 查詢 DynamoDB 全域次要索引 (GSI) 適用於 JavaScript 的 AWS SDK。

```
/**
 * 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;
  }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

使用其主索引鍵和全域次要索引 (GSI) 來查詢 DynamoDB 資料表 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

透過 SDK 使用 start_with 條件查詢 DynamoDB 資料表 AWS

下列程式碼範例示範如何使用 begins_with 條件查詢資料表。

- 在索引鍵條件表達式中使用 begins_with 函數。
- 根據排序索引鍵中的字首模式篩選項目。

Java

適用於 Java 2.x 的 SDK

在具有 AWS SDK for Java 2.x的排序索引鍵上使用 begins_with 條件來查詢 DynamoDB 資料表。

```
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;
        }
    }
```

使用具有不同字首長度的 begins_with 搭配 AWS SDK for Java 2.x進行示範。

```
   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);
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

在具有 適用於 JavaScript 的 AWS SDK的排序索引鍵上使用 begins_with 條件來查詢 DynamoDB 資料表。

```
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;
  }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

在具有 適用於 Python (Boto3) 的 AWS SDK的排序索引鍵上使用 begins_with 條件來查詢 DynamoDB 資料表。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 SDK 排序索引鍵中的日期範圍查詢 DynamoDB 資料表 AWS

下列程式碼範例示範如何使用排序索引鍵中的日期範圍查詢資料表。

- 查詢特定日期範圍內的項目。
- 在日期格式的排序索引鍵上使用比較運算子。

Java

適用於 Java 2.x 的 SDK

查詢 DynamoDB 資料表，尋找日期範圍內的項目 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;
        }
    }
```

示範如何使用日期範圍篩選條件來查詢 DynamoDB 資料表。

```
    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);
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

查詢 DynamoDB 資料表，尋找日期範圍內的項目 適用於 JavaScript 的 AWS SDK。

```
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;
  }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

查詢 DynamoDB 資料表，尋找日期範圍內的項目 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 查詢具有複雜篩選條件表達式的 DynamoDB 資料表

下列程式碼範例示範如何使用複雜篩選條件表達式查詢資料表。

- 將複雜的篩選條件表達式套用至查詢結果。
- 使用邏輯運算子結合多重條件。
- 根據非索引鍵屬性篩選項目。

Java

適用於 Java 2.x 的 SDK

使用 查詢具有複雜篩選條件表達式的 DynamoDB 資料表 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);
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 查詢具有複雜篩選條件表達式的 DynamoDB 資料表 適用於 JavaScript 的 AWS SDK。

```
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;
  }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

使用 查詢具有複雜篩選條件表達式的 DynamoDB 資料表 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 查詢具有動態篩選條件表達式的 DynamoDB 資料表

下列程式碼範例示範如何使用動態篩選條件表達式查詢資料表。

- 在執行時期動態建置篩選條件表達式。
- 根據使用者輸入或應用程式狀態建構篩選條件。
- 依條件新增或移除篩選條件。

Java

適用於 Java 2.x 的 SDK

使用 查詢具有動態建構篩選條件表達式的 DynamoDB 資料表 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();
            }
        }
    }
```

示範如何使用動態篩選條件表達式 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);
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 查詢具有動態建構篩選條件表達式的 DynamoDB 資料表 適用於 JavaScript 的 AWS SDK。

```
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;
  }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

使用 查詢具有動態建構篩選條件表達式的 DynamoDB 資料表 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

示範如何使用動態篩選條件表達式 適用於 Python (Boto3) 的 AWS SDK。

```
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}")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 SDK 查詢具有篩選條件表達式和限制的 DynamoDB 資料表 AWS

下列程式碼範例示範如何使用篩選條件表達式和限制查詢資料表。

- 套用篩選條件表達式至評估項目存在限制的查詢結果。
- 了解限制如何影響篩選的查詢結果。
- 控制查詢中處理的項目數量上限。

Java

適用於 Java 2.x 的 SDK

使用 查詢具有篩選條件表達式和限制的 DynamoDB 資料表 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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

使用 查詢具有篩選條件表達式和限制的 DynamoDB 資料表 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

示範如何使用具有 限制的篩選條件表達式 適用於 Python (Boto3) 的 AWS SDK。

```
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.")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 查詢具有巢狀屬性的 DynamoDB 資料表

下列程式碼範例示範如何查詢具有巢狀屬性的資料表。

- 依 DynamoDB 項目中的巢狀屬性存取和篩選。

- 使用文件路徑表達式來參考巢狀元素。

Java

適用於 Java 2.x 的 SDK

使用 查詢具有巢狀屬性的 DynamoDB 資料表 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;
        }
    }
```

示範如何查詢具有巢狀屬性的 DynamoDB 資料表。

```
    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);
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 查詢具有巢狀屬性的 DynamoDB 資料表 適用於 JavaScript 的 AWS SDK。

```
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;
  }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

使用 查詢具有巢狀屬性的 DynamoDB 資料表 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 查詢具有分頁的 DynamoDB 資料表

下列程式碼範例示範如何查詢具有分頁的資料表。

- 實作 DynamoDB 查詢結果的分頁。
- 使用 LastEvaluatedKey 擷取後續頁面。
- 使用限制參數控制每頁的項目數量。

Java

適用於 Java 2.x 的 SDK

使用 查詢具有分頁的 DynamoDB 資料表 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;
        }
    }
```

示範如何查詢具有分頁的 DynamoDB 資料表。

```
    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);
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 查詢具有分頁的 DynamoDB 資料表 適用於 JavaScript 的 AWS SDK。

```
/**
 * 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 };
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

使用 查詢具有分頁的 DynamoDB 資料表 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 查詢具有高度一致讀取的 DynamoDB 資料表

下列程式碼範例示範如何查詢具有高度一致性讀取的資料表。

- 設定 DynamoDB 查詢的一致性層級。
- 使用高度一致性讀取以取得最新的資料。
- 了解最終一致性與強式一致性之間的權衡。

Java

適用於 Java 2.x 的 SDK

使用 查詢具有可設定讀取一致性的 DynamoDB 資料表 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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 查詢具有可設定讀取一致性的 DynamoDB 資料表 適用於 JavaScript 的 AWS SDK。

```
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;
  }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

使用 的強式一致讀取選項查詢 DynamoDB 資料表 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

透過 AWS SDK 使用 PartiQL SELECT 陳述式查詢 DynamoDB 資料

下列程式碼範例示範如何使用 PartiQL SELECT 陳述式查詢資料。

JavaScript

適用於 JavaScript (v3) 的 SDK

搭配 PartiQL SELECT 陳述式從 DynamoDB 資料表查詢項目 適用於 JavaScript 的 AWS SDK。

```
/**
 * 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");
};
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的下列主題。
  - BatchExecuteStatement
  - ExecuteStatement

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 查詢 DynamoDB 資料表的 TTL 項目

下列程式碼範例示範如何查詢 TTL 項目。

Java

適用於 Java 2.x 的 SDK

查詢篩選表達式，以使用 在 DynamoDB 資料表中收集 TTL 項目 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;
        }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

查詢篩選表達式，以使用 在 DynamoDB 資料表中收集 TTL 項目 適用於 JavaScript 的 AWS SDK。

```
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');
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

查詢篩選表達式，以使用 在 DynamoDB 資料表中收集 TTL 項目 適用於 Python (Boto3) 的 AWS SDK。

```
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")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

透過 AWS SDK 使用日期和時間模式查詢 DynamoDB 資料表

下列程式碼範例示範如何使用日期和時間模式查詢資料表。

- 在 DynamoDB 中儲存和查詢日期/時間值。
- 使用排序索引鍵實作日期範圍查詢。
- 格式化日期字串以進行有效的查詢。

Java

適用於 Java 2.x 的 SDK

使用排序索引鍵中的日期範圍進行查詢 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;
        }
    }
```

搭配 使用日期時間變數進行查詢 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;
        }
    }
```

使用具有 AWS SDK for Java 2.x的 Unix epoch 時間戳記在日期範圍內進行查詢。

```
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;
        }
    }
```

搭配 LocalDateTime 物件在日期範圍內查詢 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);
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 Query。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用排序索引鍵中的日期範圍進行查詢 適用於 JavaScript 的 AWS SDK。

```
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;
  }
}
```

搭配 使用日期時間變數進行查詢 適用於 JavaScript 的 AWS SDK。

```
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;
```

```
  }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 Query。

Python

適用於 Python 的 SDK (Boto3)

使用排序索引鍵中的日期範圍進行查詢 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

搭配 使用日期時間變數進行查詢 適用於 Python (Boto3) 的 AWS SDK。

```
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}")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 Query。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 儲存 EXIF 和其他影像資訊

以下程式碼範例顯示做法：

- 從 JPG、JPEG 或 PNG 檔案中取得 EXIF 資訊。
- 將映像檔案上傳至 Amazon S3 儲存貯體。
- 使用 Amazon Rekognition 識別檔案中的三個主要屬性 (標籤)。

- 將 EXIF 和標籤資訊新增至區域中的 Amazon DynamoDB 資料表。

Rust

適用於 Rust 的 SDK

從 JPG、JPEG 或 PNG 檔案獲取 EXIF 資訊，將映像檔案上傳至 Amazon S3 儲存貯體，使用 Amazon Rekognition 識別三個主要屬性 (Amazon Rekognition 中的標籤)，然後將 EXIF 和標籤資訊新增至區域中的 Amazon DynamoDB 資料表中。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS Command Line Interface v2 設定 DynamoDB 的屬性型存取控制

下列程式碼範例示範如何實作 DynamoDB 的屬性型存取控制 (ABAC)。

- 建立 ABAC 的 IAM 政策。
- 為不同部門建立具有標籤的資料表。
- 根據標籤列出和篩選資料表。

Bash

AWS CLI 搭配 Bash 指令碼

建立 ABAC 的 IAM 政策。

```
# 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
```

為不同部門建立具有標籤的資料表。

```
# 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
```

根據標籤列出和篩選資料表。

```
# 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
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - CreatePolicy
  - CreateTable
  - ListTables

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

了解使用 AWS SDK 在 DynamoDB 中更新表達式順序

下列程式碼範例示範如何了解更新表達式順序。

- 了解 DynamoDB 如何處理更新表達式。
- 了解更新表達式中的操作順序。
- 了解表達式評估以避免意外結果。

Java

適用於 Java 2.x 的 SDK

使用 示範更新表達式順序 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();
        }
    }
```

使用 更新表達式順序的範例用量 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();
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 示範更新表達式順序 適用於 JavaScript 的 AWS SDK。

```
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
  };
}
```

使用 更新表達式順序的範例用量 適用於 JavaScript 的 AWS SDK。

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

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateItem。

Python

適用於 Python 的 SDK (Boto3)

使用 示範更新表達式順序 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

使用 更新表達式順序的範例用量 適用於 Python (Boto3) 的 AWS SDK。

```
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")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 UpdateItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 更新具有暖輸送量的 DynamoDB 資料表設定

下列程式碼範例示範如何更新資料表的暖輸送量設定。

Java

適用於 Java 2.x 的 SDK

使用 AWS SDK for Java 2.x更新現有 DynamoDB 資料表上的暖輸送量設定。

```
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);
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateTable。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 適用於 JavaScript 的 AWS SDK更新現有 DynamoDB 資料表上的暖輸送量設定。

```
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
);
*/
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateTable。

Python

適用於 Python 的 SDK (Boto3)

使用 適用於 Python (Boto3) 的 AWS SDK更新現有 DynamoDB 資料表上的暖輸送量設定。

```
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
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Python (Boto3) API 參考》中的 UpdateTable。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 使用 TTL 更新 DynamoDB 項目

下列程式碼範例示範如何更新項目的 TTL。

Java

適用於 Java 2.x 的 SDK

更新資料表中現有 DynamoDB 項目上的 TTL。

```
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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

```
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');
```

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateItem。

Python

適用於 Python (Boto3) 的 SDK

```
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"
```

```
)
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 UpdateItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

透過 AWS SDK 使用 PartiQL UPDATE 陳述式更新 DynamoDB 資料

下列程式碼範例示範如何使用 PartiQL UPDATE 陳述式更新資料。

JavaScript

適用於 JavaScript (v3) 的 SDK

搭配 PartiQL UPDATE 陳述式更新 DynamoDB 資料表中的項目 適用於 JavaScript 的 AWS SDK。

```
/**
 * 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(),
    },
  ]);
};
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的下列主題。
  - BatchExecuteStatement
  - ExecuteStatement

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 API Gateway 來調用 Lambda 函式

下列程式碼範例示範如何建立 Amazon API Gateway 調用的 AWS Lambda 函數。

Java

適用於 Java 2.x 的 SDK

示範如何使用 Lambda Java 執行時間 API 建立 AWS Lambda 函數。此範例會叫用不同的 AWS 服務來執行特定的使用案例。此範例示範如何建立 Amazon API Gateway 調用的 Lambda 函式，該函數會掃描 Amazon DynamoDB 資料表中的工作週年紀念日，並使用 Amazon Simple Notification Service (Amazon SNS) 傳送文字訊息給您的員工，在他們的週年紀念日向他們道賀。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

適用於 JavaScript (v3) 的 SDK

示範如何使用 Lambda JavaScript 執行時間 API 建立 AWS Lambda 函數。此範例會叫用不同的 AWS 服務來執行特定的使用案例。此範例示範如何建立 Amazon API Gateway 調用的 Lambda 函式，該函數會掃描 Amazon DynamoDB 資料表中的工作週年紀念日，並使用 Amazon Simple Notification Service (Amazon SNS) 傳送文字訊息給您的員工，在他們的週年紀念日向他們道賀。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例也可在 適用於 JavaScript 的 AWS SDK v3 開發人員指南中取得。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Python

適用於 Python (Boto3) 的開發套件

此範例顯示如何建立和使用目標為 AWS Lambda 函數的 Amazon API Gateway REST API。Lambda 處理常式會展示如何根據 HTTP 方法來路由；如何從查詢字串、標頭和本文中取得資料；以及如何傳回 JSON 回應。

- 部署 Lambda 函式。
- 建立 API Gateway REST API。
- 建立目標為 Lambda 函式的 REST 資源。
- 授與許可讓 API Gateway 調用 Lambda 函式。
- 使用 Request 套件來將請求傳送到 REST API。
- 清理示範期間建立的所有資源。

這個範例在 GitHub 上的檢視效果最佳。如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 Step Functions 調用 Lambda 函式

下列程式碼範例示範如何建立依序叫用 AWS Lambda 函數 AWS Step Functions 的狀態機器。

Java

適用於 Java 2.x 的 SDK

顯示如何使用 AWS Step Functions 和 建立無 AWS 伺服器工作流程 AWS SDK for Java 2.x。每個工作流程步驟都是使用 AWS Lambda 函數實作。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- DynamoDB
- Lambda
- Amazon SES
- 步驟函數

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 為 DynamoDB 使用文件模型

下列程式碼範例示範如何使用 DynamoDB 和 AWS SDK 的文件模型來執行建立、讀取、更新和刪除 (CRUD) 和批次操作。

如需詳細資訊，請參閱文件模型。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用文件模型執行 CRUD 操作。

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

使用文件模型執行批次寫入操作。

```
    /// <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();
        }
    }
```

使用文件模型掃描資料表。

```
    /// <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}");
            }
```

```
        }
    }
```

使用文件模型搜尋和掃描資料表。

```
    /// <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}");
```

```
            }
        }
    }
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 為 DynamoDB 使用高階物件持久性模型

下列程式碼範例示範如何使用 DynamoDB 和 AWS SDK 的物件持久性模型來執行建立、讀取、更新和刪除 (CRUD) 和批次操作。

如需詳細資訊，請參閱物件持久性模型。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用高階物件持久性模型執行 CRUD 操作。

```
/// <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");
            }
        }
    }
```

使用高階物件持久性模型執行批次寫入操作。

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

使用高階物件持久性模型將任意資料映射至資料表。

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

使用高階物件持久性模型搜尋和掃描資料表。

```
    /// <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}");
            }
        }
    }
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭配 AWS SDK 在 DynamoDB 中使用原子計數器操作

下列程式碼範例示範如何在 DynamoDB 中使用原子計數器操作。

- 使用 ADD 和 SET 操作以原子方式遞增計數器。
- 安全地遞增可能不存在的計數器。
- 實作計數器操作的樂觀鎖定。

Java

適用於 Java 2.x 的 SDK

使用 示範原子計數器操作 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;
    }
```

使用 進行原子計數器操作的範例用量 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();
        }
    }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 UpdateItem。

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 示範原子計數器操作 適用於 JavaScript 的 AWS SDK。

```
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;
}
```

使用 進行原子計數器操作的範例用量 適用於 JavaScript 的 AWS SDK。

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

- 如需 API 的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 UpdateItem。

Python

適用於 Python 的 SDK (Boto3)

使用 示範原子計數器操作 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

使用 進行原子計數器操作的範例用量 適用於 Python (Boto3) 的 AWS SDK。

```
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")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 UpdateItem。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭配 AWS SDK 在 DynamoDB 中使用條件式操作

下列程式碼範例示範如何在 DynamoDB 中使用條件式操作。

- 實作條件式寫入以防止覆寫資料。
- 使用條件表達式來強制執行業務規則。
- 從容地處理條件式檢查失敗。

Java

適用於 Java 2.x 的 SDK

使用 示範條件式操作 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;
    }
```

使用 的條件式操作範例 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();
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - DeleteItem
  - PutItem
  - UpdateItem

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 示範條件式操作 適用於 JavaScript 的 AWS SDK。

```
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;
}
```

使用 的條件式操作範例 適用於 JavaScript 的 AWS SDK。

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

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的下列主題。
  - DeleteItem
  - PutItem
  - UpdateItem

Python

適用於 Python 的 SDK (Boto3)

使用 示範條件式操作 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

使用 的條件式操作範例 適用於 Python (Boto3) 的 AWS SDK。

```
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"
    )
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的下列主題。
  - DeleteItem
  - PutItem
  - UpdateItem

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭配 AWS SDK 在 DynamoDB 中使用表達式屬性名稱

下列程式碼範例示範如何在 DynamoDB 中使用表達式屬性名稱。

- 在 DynamoDB 表達式中使用保留字。
- 使用表達式屬性名稱預留位置。
- 處理屬性名稱中的特殊字元。

Java

適用於 Java 2.x 的 SDK

使用 示範表達式屬性名稱 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);
    }
```

表達式屬性名稱的範例使用方式 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();
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - Query
  - UpdateItem

JavaScript

適用於 JavaScript (v3) 的 SDK

使用 示範表達式屬性名稱 適用於 JavaScript 的 AWS SDK。

```
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;
}
```

表達式屬性名稱的範例使用方式 適用於 JavaScript 的 AWS SDK。

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

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的下列主題。
  - Query
  - UpdateItem

Python

適用於 Python 的 SDK (Boto3)

使用 示範表達式屬性名稱 適用於 Python (Boto3) 的 AWS SDK。

```
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
```

表達式屬性名稱的範例使用方式 適用於 Python (Boto3) 的 AWS SDK。

```
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")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的下列主題。
  - Query
  - UpdateItem

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用排程事件來調用 Lambda 函式

下列程式碼範例示範如何建立由 Amazon EventBridge 排程事件呼叫的 AWS Lambda 函數。

Java

適用於 Java 2.x 的 SDK

示範如何建立叫用 AWS Lambda 函數的 Amazon EventBridge 排程事件。將 EventBridge 設定為在調用 Lambda 函式時使用 Cron 表達式來進行排程。在此範例中，您會使用 Lambda Java 執行時期 API 建立 Lambda 函式。此範例會叫用不同的 AWS 服務來執行特定的使用案例。此範例示範如何建立應用程式，將行動裝置文字訊息傳送給員工，在他們的週年紀念日向他們道賀。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

適用於 JavaScript (v3) 的 SDK

示範如何建立叫用 AWS Lambda 函數的 Amazon EventBridge 排程事件。將 EventBridge 設定為在調用 Lambda 函式時使用 Cron 表達式來進行排程。在此範例中，您會使用 Lambda JavaScript 執行時期 API 建立 Lambda 函式。此範例會叫用不同的 AWS 服務來執行特定的使用案例。此範例示範如何建立應用程式，將行動裝置文字訊息傳送給員工，在他們的週年紀念日向他們道賀。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例也可在 適用於 JavaScript 的 AWS SDK v3 開發人員指南中取得。

此範例中使用的服務

- CloudWatch Logs
- DynamoDB
- EventBridge

- Lambda
- Amazon SNS

Python

適用於 Python 的 SDK (Boto3)

此範例說明如何將 AWS Lambda 函數註冊為排程 Amazon EventBridge 事件的目標。Lambda 處理常式會將合適的訊息和完整的事件資料寫入 Amazon CloudWatch Logs 中以供日後擷取。

- 部署 Lambda 函式。
- 建立一個 EventBridge 排程事件，並將 Lambda 函式做為目標。
- 授予許可讓 EventBridge 調用 Lambda 函式。
- 列印 CloudWatch Logs 中的最新資料，以顯示排程調用的結果。
- 清理示範期間建立的所有資源。

這個範例在 GitHub 上的檢視效果最佳。如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS Command Line Interface v2 使用 DynamoDB 本機次要索引

下列程式碼範例示範如何使用本機次要索引建立和查詢資料表。

- 使用本機次要索引 (LSI) 建立資料表。
- 使用具有不同投影類型的多個 LSI 建立資料表。
- 使用 LSI 查詢資料。

Bash

AWS CLI 使用 Bash 指令碼

使用本機次要索引建立資料表。

```
# 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
```

建立具有多個 LSI 的資料表。

```
# 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
```

使用 LSI 查詢資料。

```
# 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"}
    }'
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - CreateTable
  - Query

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS Command Line Interface v2 處理 DynamoDB 串流和Time-to-Live

下列程式碼範例示範如何管理 DynamoDB 串流和存留時間功能。

- 建立啟用串流的資料表。
- 描述串流。
- 建立 Lambda 函式以處理串流。
- 啟用資料表上的 TTL。
- 新增具有 TTL 屬性的項目。
- 描述 TTL 設定。

Bash

AWS CLI 使用 Bash 指令碼

建立啟用串流的資料表。

```
# 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
```

描述串流。

```
# 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
```

為串流建立 Lambda 函式。

```
# 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"
```

啟用資料表上的 TTL。

```
# 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"
```

新增具有 TTL 屬性的項目。

```
# 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'"}
    }'
```

描述 TTL 設定。

```
# Describe TTL settings for a table
aws dynamodb describe-time-to-live \
    --table-name TTLDemo
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - AttachRolePolicy
  - CreateRole
  - CreateTable
  - DescribeTable
  - DescribeTimeToLive
  - PutItem
  - UpdateTimeToLive

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 v AWS Command Line Interface 2 搭配 DynamoDB 全域資料表和具有最終一致性 (MREC) 的多區域複寫

下列程式碼範例示範如何使用具有最終一致性 (MREC) 的多區域複寫，來管理 DynamoDB 全域資料表。

- 建立具有多區域複寫 (MREC) 的資料表。
- 將項目放置於複本資料表，並從其中取得項目。
- 逐一移除複本。
- 藉由刪除資料表進行清除。

Bash

AWS CLI 使用 Bash 指令碼

利用多區域複寫來建立資料表。

```
# Step 1: Create a new table (MusicTable) in US East (Ohio), with DynamoDB
 Streams enabled (NEW_AND_OLD_IMAGES)
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2

# Step 2: Create an identical MusicTable table in US East (N. Virginia)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
```

```
    }
  ]
}' \
--region us-east-2

# Step 3: Create a table in Europe (Ireland)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2
```

描述多區域資料表。

```
# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

將項目放置於複本資料表中。

```
# Step 5: To verify that replication is working, add a new item to the Music
 table in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

從複本資料表取得項目。

```
# Step 6: Wait for a few seconds, and then check to see whether the item has
 been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

移除複本。

```
# Step 7: Delete the replica table in Europe (Ireland) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2

# Delete the replica table in US East (N. Virginia) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2
```

透過刪除資料表來進行清除作業。

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateTable

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS Command Line Interface v2 使用 DynamoDB 資源標記

以下程式碼範例示範如何管理 DynamoDB 資源的標籤。

- 建立具有標籤的資料表。
- 列出資源的標籤。
- 將標籤新增至資源。
- 從資源移除標籤。
- 依標籤篩選資料表。

Bash

AWS CLI 使用 Bash 指令碼

建立具有標籤的資料表。

```
# 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
```

列出資源的標籤。

```
# 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
```

將標籤新增至資源。

```
# Add tags to an existing table
aws dynamodb tag-resource \
    --resource-arn $TABLE_ARN \
    --tags \
        Key=CostCenter,Value=12345 \
        Key=BackupSchedule,Value=Daily
```

從資源移除標籤。

```
# Remove tags from a table
aws dynamodb untag-resource \
```

```
    --resource-arn $TABLE_ARN \
    --tag-keys Owner BackupSchedule
```

依標籤篩選資料表。

```
# 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
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。

- CreateTable
- ListTagsOfResource
- TagResource
- UntagResource

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS Command Line Interface v2 使用 DynamoDB 資料表加密

下列程式碼範例示範如何管理 DynamoDB 資料表的加密選項。

- 建立具預設加密的儲存貯體。
- 利用客戶管理的 CMK 建立資料表。
- 更新資料表加密設定。
- 描述資料表加密。

Bash

AWS CLI 使用 Bash 指令碼

建立具預設加密的儲存貯體。

```
# 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
```

利用客戶管理的 CMK 建立資料表。

```
# 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
```

更新資料表加密。

```
# 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
```

描述資料表加密。

```
# Describe the table to see encryption settings
aws dynamodb describe-table \
    --table-name CustomerData \
    --query "Table.SSEDescription"
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - CreateKey
  - CreateTable
  - DescribeTable
  - UpdateTable

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

DynamoDB 的無伺服器範例

下列程式碼範例示範如何使用 DynamoDB AWS SDKs。

範例

- 使用 DynamoDB 觸發條件調用 Lambda 函式
- 使用 DynamoDB 觸發條件報告 Lambda 函式的批次項目失敗

使用 DynamoDB 觸發條件調用 Lambda 函式

下列程式碼範例示範如何實作 Lambda 函式，以便接收在收到來自 DynamoDB 串流的訊息時觸發的事件。函數會擷取 DynamoDB 承載並記下記錄內容。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 .NET 搭配 Lambda 來使用 DynamoDB 事件。

```
// 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 for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Go 搭配 Lambda 來使用 DynamoDB 事件。

```
// 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

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Java 搭配 Lambda 來使用 DynamoDB 事件。

```
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

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 JavaScript 搭配 Lambda 來使用 DynamoDB 事件。

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

使用 TypeScript 搭配 Lambda 來使用 DynamoDB 事件。

```
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

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 PHP 搭配 Lambda 來使用 DynamoDB 事件。

```
<?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

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Python 搭配 Lambda 來使用 DynamoDB 事件。

```
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 for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Ruby 搭配 Lambda 來使用 DynamoDB 事件。

```
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

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 DynamoDB 事件。

```
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(())

}
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 DynamoDB 觸發條件報告 Lambda 函式的批次項目失敗

下列程式碼範例示範如何針對接收來自 DynamoDB 串流之事件的 Lambda 函式，實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 .NET 搭配 Lambda 報告 DynamoDB 批次項目失敗。

```
// 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 for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Go 搭配 Lambda 報告 DynamoDB 批次項目失敗。

```
// 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

適用於 Java 2.x 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Java 搭配 Lambda 報告 DynamoDB 批次項目失敗。

```
// 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

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 JavaScript 搭配 Lambda 報告 DynamoDB 批次項目失敗。

```
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: [] };
};
```

使用 TypeScript 搭配 Lambda 報告 DynamoDB 批次項目失敗。

```
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

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 PHP 搭配 Lambda 報告 DynamoDB 批次項目失敗。

```
<?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

適用於 Python (Boto3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Python 搭配 Lambda 報告 DynamoDB 批次項目失敗。

```
# 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 for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Ruby 搭配 Lambda 報告 DynamoDB 批次項目失敗。

```
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

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Rust 搭配 Lambda 報告 DynamoDB 批次項目失敗。

```
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
}
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

AWS DynamoDB 的社群貢獻

AWS 社群貢獻是由多個團隊所建立和維護的範例 AWS。若要提供意見回饋，請使用連結儲存庫中提供的機制。

範例

- 建置和測試無伺服器應用程式

建置和測試無伺服器應用程式

下列程式碼範例示範如何搭配 Lambda 和 DynamoDB 使用 API Gateway 來建置和測試無伺服器應用程式

.NET

適用於 .NET 的 SDK

示範如何使用 .NET SDK 建置和測試無伺服器應用程式，而該應用程式是由具有 Lambda 和 DynamoDB 的 API Gateway 組成。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda

Go

SDK for Go V2

示範如何使用 Go SDK 建置和測試無伺服器應用程式，而該應用程式是由具有 Lambda 和 DynamoDB 的 API Gateway 組成。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda

Java

適用於 Java 2.x 的 SDK

示範如何使用 Java SDK 建置和測試無伺服器應用程式，而該應用程式是由具有 Lambda 和 DynamoDB 的 API Gateway 組成。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda

Rust

適用於 Rust 的 SDK

示範如何使用 Rust SDK 建置和測試無伺服器應用程式，而該應用程式是由具有 Lambda 和 DynamoDB 的 API Gateway 組成。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用 DynamoDB。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

Amazon DynamoDB 中的安全與合規

的雲端安全性 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，這些架構專為滿足最安全敏感組織的需求而建置。

安全性是 AWS 與您之間的共同責任。共同責任模型 將此描述為雲端的安全和雲端內的安全：

- 雲端的安全性 – AWS 負責保護在 AWS Cloud 中執行 AWS 服務的基礎設施。 AWS 也為您提供可安全使用的服務。第三方稽核人員定期檢測及驗證安全的效率也是我們 AWS 合規計劃的一部分。若要進一步了解適用於 DynamoDB 的合規計劃，請參閱合規計劃範圍內的AWS 服務。
- 雲端的安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對資料敏感度、組織要求，以及適用法律和法規等其他因素負責。

本文件會協助您了解使用 DynamoDB 時共同的責任模型的適用情形。下列主題將示範如何設定 DynamoDB 以達到您的安全和合規目標。您也將了解如何使用其他 AWS 服務，協助您監控和保護 DynamoDB 資源。

主題

- AWS Amazon DynamoDB 的 受管政策
- 對 DynamoDB 使用資源型政策
- 搭配 DynamoDB 使用屬性型存取控制
- DynamoDB 中的資料保護
- AWS Identity and Access Management (IAM) 和 DynamoDB
- DynamoDB 的業界合規驗證
- Amazon DynamoDB 中的資料復原及災難復原功能
- Amazon DynamoDB 中的基礎設施安全性
- AWS PrivateLink 適用於 DynamoDB
- Amazon DynamoDB 中的組態與漏洞分析
- Amazon DynamoDB 的安全最佳實務

AWS Amazon DynamoDB 的 受管政策

DynamoDB 使用 AWS 受管政策來定義服務執行特定動作所需的一組許可。DynamoDB 會維護和更新其 AWS 受管政策。您無法變更 AWS 受管政策中的許可。如需 AWS 受管政策的詳細資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策。

DynamoDB 可能會偶爾將其他許可新增至 AWS 受管政策，以支援新功能。此類型的更新會影響已連接政策的所有身分識別 (使用者、群組和角色)。當新功能啟動或新操作可用時，最有可能更新 AWS 受管政策。DynamoDB 不會從 AWS 受管政策中移除許可，因此政策更新不會破壞您現有的許可。如需 AWS 受管政策的完整清單，請參閱 AWS 受管政策。

AWS 受管政策：DynamoDBReplicationServiceRolePolicy

您無法將 DynamoDBReplicationServiceRolePolicy 政策附加至 IAM 實體。此政策會連接到服務連結角色，而此角色可讓 DynamoDB 代表您執行動作。如需詳細資訊，請參閱搭配全域資料表使用 IAM。

此政策會授予許可，允許服務連結角色執行全域資料表複本之間的資料。也會授與管理許可，以代表您管理全域資料表複本。

許可詳細資訊

此政策會授予下列許可：

- dynamodb – 執行資料複寫並管理資料表複本。
- application-autoscaling – 擷取與管理資料表自動擴展設定
- account – 擷取區域狀態以評估複本可存取性。
- iam – 若尚未存在服務連結角色，則為應用程式自動擴展建立該服務連結角色。

您可以在這裡找到此受管政策的定義。

AWS 受管政策：AmazonDynamoDBFullAccess_v2

範圍受限的 AmazonDynamoDBFullAccess_v2 政策會授與使用者特定的存取權限。您可將 AmazonDynamoDBFullAccess_v2 政策連接到 IAM 身分。此政策會授予 Amazon DynamoDB 資源的管理存取權，並授予 IAM 身分 （例如使用者、群組或角色） 存取權給與 DynamoDB 整合 AWS 服務 的 ，以使用所有 DynamoDB 功能。使用此政策可存取 AWS 管理主控台中所有可用的 DynamoDB 功能。

許可詳細資訊

此政策包含以下許可：

- Amazon DynamoDB
- DynamoDB Accelerator
- AWS KMS
- AWS Resource Groups Tagging
- Lambda
- Application Auto Scaling
- CloudWatch
- Amazon Kinesis
- Amazon EC2
- IAM

若要以 JSON 格式檢視此政策，請參閱 AmazonDynamoDBFullAccess_v2。

AWS 受管政策：AmazonDynamoDBReadOnlyAccess

您可將 AmazonDynamoDBReadOnlyAccess 政策連接到 IAM 身分。

此政策授與對 Amazon DynamoDB 的唯讀存取權。

許可詳細資訊

此政策包含以下許可：

- Amazon DynamoDB – 提供對 Amazon DynamoDB 的唯讀存取權。
- Amazon DynamoDB Accelerator (DAX) – 提供對 Amazon DynamoDB Accelerator (DAX) 的唯讀存取權。
- Application Auto Scaling – 允許主體檢視來自 Application Auto Scaling 的設定。這是必要的，讓使用者能檢視附加於資料表的自動擴展政策。
- CloudWatch – 允許主體檢視於 CloudWatch 中設定的指標資料與警示。這是必要的，以便使用者檢視資料表的計費大小與 CloudWatch 警示。
- AWS Data Pipeline – 允許主體檢視 AWS Data Pipeline 和相關聯的物件。
- Amazon EC2：允許主體檢視 Amazon EC2 VPC、子網路與安全群組。

- IAM – 允許主體檢視 IAM 角色。
- AWS KMS – 允許主體檢視其中設定的金鑰 AWS KMS。這是必要的 AWS KMS keys ，以便使用者可以檢視他們在其帳戶中建立和管理 。
- Amazon SNS – 允許主體列出 Amazon SNS 的主題與其訂閱。
- AWS Resource Groups – 允許主體檢視資源群組及其查詢。
- AWS Resource Groups Tagging – 允許主體列出區域內所有已標記或先前標記過的資源。
- Kinesis – 允許主體檢視 Kinesis 資料串流的描述資訊。
- Amazon CloudWatch Contributor Insights – 允許主體檢視由 Contributor Insights 規則收集的時間序列資料。

若要以 JSON 格式檢視此政策，請參閱 AmazonDynamoDBReadOnlyAccess。

AWS 受管政策的 DynamoDB 更新

此資料表顯示 DynamoDB AWS 存取管理政策的更新。

變更	描述	變更日期
AmazonDynamoDBFullAccess – 已棄用	此政策已由名為 AmazonDynamoDBFullAccess_v2 的範圍受限政策取代。 2025 年 4 月之後，您將無法將 AmazonDynamoDBFullAccess 政策附加至任何新的使用者、群組或角色。如需詳細資訊，請參閱AWS 受管政策：AmazonDynamoDBFullAccess_v2。	2025 年 4 月 28 日
AmazonDynamoDBReadOnlyAccess 更新現有政策	AmazonDynamoDBReadOnlyAccess 已新增下列權限：dynamodb:GetAbacStatus 與 dynamodb:UpdateAbacStatus 。這些許可可讓您檢視 ABAC 狀態，並在 AWS 帳戶 目前區域中為 啟用 ABAC。	2024 年 11 月 18 日

變更	描述	變更日期
AmazonDynamoDBReadOnlyAccess 更新現有政策	AmazonDynamoDBReadOnlyAccess 新增權限 dynamodb:GetResourcePolicy 。此權限可讓您讀取附加於 DynamoDB 資源的資源型政策。	2024 年 3 月 20 日
DynamoDBReplicationServiceRolePolicy 更新現有政策	DynamoDBReplicationServiceRolePolicy 新增權限 dynamodb:GetResourcePolicy 。此權限允許服務連結角色讀取附加於 DynamoDB 資源的資源型政策。	2023 年 12 月 15 日
DynamoDBReplicationServiceRolePolicy 更新現有政策	DynamoDBReplicationServiceRolePolicy 新增權限 account:ListRegions 。此權限允許服務連結角色評估複本的存取	2023 年 5 月 10 日
DynamoDBReplicationServiceRolePolicy 已新增至受管政策清單	已新增有關 DynamoDB 全域資料表服務連結角色使用之受管政策 DynamoDBReplicationServiceRolePolicy 的資訊。	2023 年 5 月 10 日
DynamoDB 全域資料表已開始追蹤其變更	DynamoDB 全域資料表開始追蹤其 AWS 受管政策的變更。	2023 年 5 月 10 日

對 DynamoDB 使用資源型政策

DynamoDB 支援資料表、索引與串流的資源型政策。資源型政策可讓您指定有權存取各項資源的人員，以及允許前述人員對各項資源執行的動作，藉此定義存取許可權限。

您可以將資源型政策附加至 DynamoDB 資源，例如資料表或串流。在此政策中，您可為身分與存取管理 (IAM) 的主體指定可於這些 DynamoDB 資源上執行的特定動作權限。例如，附加於資料表的政策將包含存取該資料表及其索引的權限。因此，資源型政策可透過在資源層級定義權限，協助您簡化對 DynamoDB 資料表、索引與串流的存取控制。可附加至 DynamoDB 資源的政策大小上限為 20 KB。

使用資源型政策的主要優點之一，是簡化跨帳戶存取控制，讓您能為不同 AWS 帳戶中的 IAM 主體授與跨帳戶存取權。如需詳細資訊，請參閱跨帳戶存取權資源型政策。

資源型政策亦支援與 IAM 存取分析器外部存取分析器及封鎖公開存取 (BPA)功能整合。IAM 存取分析器會回報資源型政策中所指定之外部實體的跨帳戶存取情況。它同時提供可見性，協助您調整權限並符合最低權限政策。BPA 可協助您防止 DynamoDB 資料表、索引與串流遭公開存取，並於資源型政策的建立與修改流程中自動啟用。

主題

- 使用資源型政策建立資料表
- 將資源型政策連接至現有 DynamoDB 資料表
- 將資源型政策附加至 DynamoDB 串流
- 從 DynamoDB 資料表移除資源型政策
- DynamoDB 中使用資源型政策的跨帳戶存取權
- 使用 DynamoDB 資源型政策封鎖公開存取
- 支援資源型政策的 DynamoDB API 操作
- 使用 IAM 身分型政策與 DynamoDB 資源型政策進行授權
- DynamoDB 資源型政策範例
- DynamoDB 資源型政策考量事項
- DynamoDB 資源型政策最佳實務

使用資源型政策建立資料表

您可以使用 DynamoDB 主控台、CreateTable API、 AWS CLIAWS SDK 或 CloudFormation 範本，在建立資料表時新增資源型政策。

AWS CLI

下列範例會使用 `create-table` AWS CLI 命令建立名為 *`MusicCollection`* 的資料表。此指令同時包含 `resource-policy` 參數，用以將資源型政策新增至資料表。此政策允許使用者 *`John`* 對該資料表執行 RestoreTableToPointInTime、GetItem 與 PutItem API 動作。

請以資源特定資訊取代*##*文字。

```
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 管理主控台

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在儀表板上，選擇建立資料表。
3. 在資料表詳細資料中，輸入資料表名稱、分割區索引鍵與排序索引鍵的詳細資料。
4. 在資料表設定中，選擇自訂設定。
5. (選用) 可指定資料表類別、容量計算機、讀取/寫入容量設定、輔助索引、靜態加密與刪除保護等選項。
6. 在資源型政策中，新增政策以定義資料表及其索引的存取權限。在此政策中，您可指定哪些使用者可存取這些資源，以及他們可對各資源執行的動作。若要新增政策，請執行以下任一操作：

- 輸入或貼上 JSON 政策文件。如需 IAM 政策語言的詳細資訊，請參閱《IAM 使用者指南》中的使用 JSON 編輯器建立政策。

> **Tip**
>
> 若要查看 Amazon DynamoDB 開發人員指南中的資源型政策範例，請選取政策範例。

- 選擇新增陳述式，以在提供的欄位中輸入資訊並新增新的陳述式。針對您想要新增的任意數量陳述式重複此步驟。

> **Important**
>
> 儲存政策前，請務必先解決所有安全性警告、錯誤或建議。

下列 IAM 政策範例允許使用者 *John* 對資料表 *MusicCollection* 執行 RestoreTableToPointInTime、GetItem 與 PutItem API 動作。

請以資源特定資訊取代*##*文字。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/username"
      },
      "Action": [
        "dynamodb:RestoreTableToPointInTime",
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection"
    }
  ]
```

```
}
```

7. (選用) 選擇右下角的 Preview external access (預覽外部存取)，以預覽新政策會如何影響資源的公開和跨帳戶存取權。在儲存政策之前，您可以檢查它是否引入新的 IAM Access Analyzer 問題清單，或是解決現有的問題清單。如果您沒有看到作用中的分析器，請選擇 Go to Access Analyzer (移至 Access Analyzer)，以在 IAM Access Analyzer 中建立帳戶分析器。如需詳細資訊，請參閱預覽存取。
8. 選擇 Create Table (建立資料表)。

AWS CloudFormation 範本

Using the AWS::DynamoDB::Table resource

下列 CloudFormation 範本會使用 AWS::DynamoDB::Table 資源建立帶有串流的資料表。此範本也包含資源型政策，分別連接至資料表與串流。

```
{
    "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

下列 CloudFormation 範本會使用 AWS::DynamoDB::GlobalTable 資源建立資料表，並將資源型政策連接至該資料表及其串流。

```
{
    "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": "*"
                                    }]
                                }
                            }
                        }
                    ]
                }
            }
        }
```

將資源型政策連接至現有 DynamoDB 資料表

您可以使用 DynamoDB 主控台、PutResourcePolicy API、 AWS CLI、 AWS SDK 或 CloudFormation 範本，將資源型政策連接至現有資料表或修改現有政策。

AWS CLI 連接新政策的範例

下列 IAM 政策範例使用 `put-resource-policy` AWS CLI 命令，將資源型政策連接至現有資料表。此範例允許使用者 *John* 對名為 *MusicCollection* 的現有資料表執行 GetItem、PutItem、UpdateItem 與 UpdateTable API 動作。

請以資源特定資訊取代*##*文字。

```
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 以條件式更新現有政策的範例

若要有條件地更新資料表現有的資源型政策，您可以使用選用的 expected-revision-id 參數。下列範例僅當政策存在於 DynamoDB 且其目前修訂版 ID 符合提供的 expected-revision-id 參數時，才會更新政策。

```
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 管理主控台

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在儀表板中選取現有資料表。
3. 前往權限分頁，然後選取建立資料表政策。
4. 在資源型政策編輯器中，新增要附加的政策，然後選取建立政策。

   下列 IAM 政策範例允許使用者 *John* 對名為 *MusicCollection* 的現有資料表執行 GetItem、PutItem、UpdateItem 與 UpdateTable API 動作。

   請以資源特定資訊取代*##*文字。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/username"
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:UpdateTable"
      ],
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

AWS SDK for Java 2.x

下列 IAM 政策範例使用 putResourcePolicy 方法，將資源型政策附加到現有資料表。此政策允許使用者在現有資料表上執行 GetItem API 操作。

```
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" +
                "}";
    }
```

```
}
```

將資源型政策附加至 DynamoDB 串流

您可以使用 DynamoDB 主控台、PutResourcePolicy API、 AWS CLI、 AWS SDK 或 CloudFormation 範本，將資源型政策連接至現有資料表的串流或修改現有政策。

> **Note**
>
> 使用 CreateTable 或 UpdateTable API 建立時，無法將政策附加至串流。不過，資料表刪除後仍可修改或刪除政策。您也可以修改或刪除停用串流的政策。

AWS CLI

下列 IAM 政策範例使用 `put-resource-policy` AWS CLI 命令，將資源型政策連接至名為 *MusicCollection* 的資料表串流。此範例允許使用者 *John* 對該串流執行 GetRecords、GetShardIterator 與 DescribeStream API 操作。

請以資源特定資訊取代*##*文字。

```
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 管理主控台

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在 DynamoDB 主控台的儀表板中，選取資料表，然後選取現有資料表。

   請確認所選資料表已啟用串流功能。如需啟用資料表串流的詳細資訊，請參閱 啟用串流。
3. 選擇許可索引標籤。
4. 在作用中串流的資源型政策中，選取建立串流政策。
5. 在資源型政策編輯器中，新增政策以定義串流的存取權限。在此政策中，您可指定可存取串流的使用者，以及他們可執行的操作。若要新增政策，請執行以下任一操作：

   - 輸入或貼上 JSON 政策文件。如需 IAM 政策語言的詳細資訊，請參閱《IAM 使用者指南》中的使用 JSON 編輯器建立政策。

     > **Tip**
     >
     > 若要查看 Amazon DynamoDB 開發人員指南中的資源型政策範例，請選取政策範例。

   - 選擇新增陳述式，以在提供的欄位中輸入資訊並新增新的陳述式。針對您想要新增的任意數量陳述式重複此步驟。

   > **Important**
   >
   > 儲存政策前，請務必先解決所有安全性警告、錯誤或建議。

6. (選用) 選擇右下角的 Preview external access (預覽外部存取)，以預覽新政策會如何影響資源的公開和跨帳戶存取權。在儲存政策之前，您可以檢查它是否引入新的 IAM Access Analyzer 問題清單，或是解決現有的問題清單。如果您沒有看到作用中的分析器，請選擇 Go to Access Analyzer (移至 Access Analyzer)，以在 IAM Access Analyzer 中建立帳戶分析器。如需詳細資訊，請參閱預覽存取。
7. 選擇建立政策。

下列 IAM 政策範例會將資源型政策附加至名為 *MusicCollection* 的資料表串流。此範例允許使用者 *John* 對該串流執行 GetRecords、GetShardIterator 與 DescribeStream API 操作。

請以資源特定資訊取代*##*文字。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/username"
      },
      "Action": [
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:DescribeStream"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

從 DynamoDB 資料表移除資源型政策

您可以使用 DynamoDB 主控台、DeleteResourcePolicy API、 AWS CLI、 AWS SDK 或 CloudFormation 範本，從現有資料表刪除資源型政策。

AWS CLI

下列範例使用 `delete-resource-policy` AWS CLI 命令，從名為 *MusicCollection* 的資料表中移除資源型政策。

請以資源特定資訊取代*##*文字。

```
aws dynamodb delete-resource-policy \
```

```
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

AWS 管理主控台

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在 DynamoDB 主控台的儀表板中，選取資料表，然後選取現有資料表。
3. 選擇許可。
4. 在管理政策下拉式清單中，選取刪除政策。
5. 在刪除資料表資源型政策對話方塊中，輸入 **`confirm`** 以確認刪除操作。
6. 選擇 刪除。

DynamoDB 中使用資源型政策的跨帳戶存取權

您可以使用資源型政策，提供位於不同 AWS 帳戶中可用資源的跨帳戶存取權。如果您具有 AWS 區域與資源相同的分析器，則資源型政策允許的所有跨帳戶存取將透過 IAM Access Analyzer 外部存取調查結果進行報告。IAM Access Analyzer 會比對 IAM 政策文法和最佳實務來執行政策檢查，以驗證您的政策。這些檢查會產生問題清單並提供可行的建議，協助您撰寫具有功能性且符合安全最佳實務的政策。您可以在 DynamoDB 主控台的權限分頁中，檢視 IAM 存取分析器的有效調查結果。

如需使用 IAM 存取分析器驗證政策的詳細資訊，請參閱 IAM 使用者指南中的 IAM 存取分析器政策驗證。若要檢視 IAM Access Analyzer 傳回的警告、錯誤和建議清單，請參閱 IAM Access Analyzer 政策檢查參考。

若要授予帳戶 A 的使用者 A GetItem 權限，以便存取帳戶 B 的資料表 B，請依下列步驟操作：

1. 將資源型政策附加至資料表 B，以授予使用者 A 執行 `GetItem` 操作的權限。
2. 將身分型政策附加至使用者 A，以授予其對資料表 B 執行 `GetItem` 操作的權限。

您可以使用 DynamoDB 主控台中的預覽外部存取選項，預覽新政策對資源公有及跨帳戶存取權的影響。在儲存政策之前，您可以檢查它是否引入新的 IAM Access Analyzer 問題清單，或是解決現有的問題清單。如果您沒有看到作用中的分析器，請選擇 Go to Access Analyzer (移至 Access Analyzer)，以在 IAM Access Analyzer 中建立帳戶分析器。如需詳細資訊，請參閱預覽存取。

DynamoDB 資料平面與控制平面 API 中的資料表名稱參數可接受完整的 Amazon Resource Name (ARN)，以支援跨帳戶作業。若僅提供資料表名稱參數 (未包含完整 ARN)，API 作業將在請求者所屬帳戶的資料表上執行。如需使用跨帳戶存取的政策範例，請參閱 跨帳戶存取權資源型政策。

即使其他帳戶的主體讀取或寫入擁有者帳戶中的 DynamoDB 資料表，費用仍由資源擁有者帳戶承擔。若資料表採用預先佈建的輸送量，則來自擁有者與其他帳戶請求者的所有請求總和，將決定請求是否遭到限流 (若停用自動調整)，或進行擴增/縮減 (若啟用自動調整)。

所有請求將記錄於擁有者與請求者帳戶的 CloudTrail 日誌中，使雙方均可追蹤資料的存取來源。

與跨帳戶 AWS Lambda 函數共用存取權

帳戶 A 中的 Lambda 函數

1. 前往 IAM 主控台建立 IAM 角色，該角色將用作帳戶 A 中 Lambda 函數的 AWS Lambda 執行角色。新增`AWSLambdaDynamoDBExecutionRole`具有所需 DynamoDB Streams 和 Lambda 調用許可的受管 IAM 政策。此政策也會授予您可能在帳戶 A 中存取的所有潛在 DynamoDB Streams 資源的存取權。
2. 在 Lambda 主控台中，建立 AWS Lambda 函數來處理 DynamoDB 串流中的記錄，並在設定執行角色期間，選擇您在上一個步驟中建立的角色。
3. 將 Lambda 函數執行角色提供給帳戶 B 的 DynamoDB Streams 擁有者，以設定跨帳戶讀取存取的資源型政策。
4. 完成 Lambda 函數的設定。

帳戶 B 中的 DynamoDB 串流

1. 從將叫用 Lambda 函數的帳戶 A 取得跨帳戶 Lambda 執行角色。
2. 在帳戶 B 的 Amazon DynamoDB 主控台上，選擇 Lambda 跨帳戶觸發條件的資料表。在匯出和串流索引標籤下，找到您的 DynamoDB 串流 ARN。確保 DynamoDB 串流狀態為開啟，並記下完整的串流 ARN，因為您需要它才能執行資源政策。
3. 在許可索引標籤下，按一下建立串流政策按鈕以啟動視覺化政策編輯器。如果已存在，請按一下新增陳述式按鈕或編輯政策。
4. 建立政策，指定帳戶 A 中的 Lambda 執行角色做為委託人，並授予必要的 DynamoDB Stream 動作。請務必包含動作 `dynamodb:DescribeStream`、`dynamodb:GetRecords`、`dynamodb:GetShardIterator`和 `dynamodb:ListShards`。如需 DynamoDB Streams 資源政策範例的詳細資訊，請參閱 DynamoDB 資源型政策範例。

> **Note**
>
> 控制平面 API 的跨帳戶存取權每秒交易數 (TPS) 上限為 500 個請求。

使用 DynamoDB 資源型政策封鎖公開存取

封鎖公開存取 (BPA) 是一項功能，可偵測並防止附加授予公開存取權的資源型政策，避免其在您的 Amazon Web Services (AWS) 帳戶中影響 DynamoDB 資料表、索引或串流。透過 BPA，您可以防止對 DynamoDB 資源的公開存取。BPA 會在建立或修改資源型政策時執行檢查，協助提升您在 DynamoDB 中的安全防護狀態。

BPA 會使用自動推理技術分析資源型政策授予的存取權，若在管理政策時偵測到此類權限，系統會發出警示通知。分析會驗證所有以資源為基礎的政策陳述、動作，以及您政策中所使用的條件金鑰集合的存取權限。

> **Important**
>
> BPA 透過防止透過直接附加至 DynamoDB 資源 (如資料表、索引和串流) 的資源型政策授予公開存取權，來協助保護您的資源。除了使用 BPA 外，請仔細檢查下列政策，確認這些政策未授予公開存取權：
>
> - 連接到相關聯 AWS 主體的身分型政策 （ 例如 IAM 角色 ）
> - 連接至相關聯 AWS 資源的資源型政策 （ 例如 AWS Key Management Service (KMS) 金鑰 ）

您必須確保主體不包含 * 項目，或由指定的條件金鑰之一限制主體對資源的存取。如果以資源為基礎的政策授予對資料表、索引或串流的公開存取權 AWS 帳戶，則 DynamoDB 會阻止您建立或修改政策，直到政策中的規格更正並視為非公開。

您可以在 `Principal` 區塊內指定一個或多個主體，將政策設為非公開。下列資源型政策範例透過指定兩個主體來阻止公開存取。

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dynamodb:*",
  "Resource": "*"
```

```
}
```

指定特定條件金鑰以限制存取的政策，也不會被視為公開。除了評估資源型政策中指定的主體外，下列受信任條件金鑰也用於完成非公開存取的政策評估。

- 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

此外，若要使資源型政策為非公開，Amazon Resource Name (ARN) 與字串金鑰的值不得包含萬用字元或變數。若您的資源型政策使用 aws:PrincipalIsAWSService 金鑰，您必須確保已將該金鑰值設為 true。

下列政策限制指定帳戶中使用者 John 的存取權。該條件使 Principal 受限，因此不被視為公開。

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dynamodb:*",
  "Resource": "*",
```

```
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/John"
    }
  }
}
```

下列非公開的資源型政策範例使用 StringEquals 運算子來限制 sourceVPC。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dynamodb:*",
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/
MusicCollection",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

支援資源型政策的 DynamoDB API 操作

本主題列出受資源型政策支援的 API 操作。不過，進行跨帳戶存取時，您只能透過資源型政策使用特定一組 DynamoDB API。您無法將資源型政策附加至某些資源類型，例如備份與匯入。對應於這些資源類型 API 的 IAM 動作不在資源型政策支援範圍內。由於資料表管理員會在同一帳戶內設定資料表的內部參數，因此 UpdateTimeToLive 和 DisableKinesisStreamingDestination 等 API 不支援透過資源型政策進行跨帳戶存取。

支援跨帳戶存取的 DynamoDB 資料平面與控制平面 API 也支援資料表名稱多載，允許您指定資料表 ARN 以取代資料表名稱。您可以在這些 API 的 `TableName` 參數中指定資料表 ARN。不過，並非所有這些 API 都支援跨帳戶存取。

主題

- 資料平面 API 操作
- PartiQL API 操作
- 控制平面 API 操作
- 2019.11.21 版 (目前) 全域資料表 API 操作
- 2017.11.29 版 (舊版) 全域資料表 API 操作
- 標籤 API 操作
- 備份與還原 API 操作
- 持續備份/還原 (PITR) API 操作
- Contributor Insights API 操作
- 匯出 API 操作
- 匯入 API 操作
- Amazon Kinesis Data Streams API 操作
- 資源型政策 API 操作
- 存留時間 API 操作
- 其他 API 操作
- DynamoDB Streams API 操作

資料平面 API 操作

下表列出資料平面 API 操作針對資源型政策與跨帳戶存取所提供的 API 層級支援。

資料平面 - 資料表/索引 API	資源型政策支援	跨帳戶支援
DeleteItem	是	是
GetItem	是	是
PutItem	是	是

資料平面 - 資料表/索引 API	資源型政策支援	跨帳戶支援
Query	是	是
Scan	是	是
UpdateItem	是	是
TransactGetItems	是	是
TransactWriteItems	是	是
BatchGetItem	是	是
BatchWriteItem	是	是

PartiQL API 操作

下表列出 PartiQL API 操作針對資源型政策與跨帳戶存取權所提供的 API 層級支援。

PartiQL API	資源型政策支援	跨帳戶支援
BatchExecuteStatement	是	否
ExecuteStatement	是	否
ExecuteTransaction	是	否

控制平面 API 操作

下表列出控制平面 API 操作針對資源型政策與跨帳戶存取所提供的 API 層級支援。

控制平面 - 資料表 API	資源型政策支援	跨帳戶支援
CreateTable	否	否
DeleteTable	是	是
DescribeTable	是	是

控制平面 - 資料表 API	資源型政策支援	跨帳戶支援
UpdateTable	是	是

2019.11.21 版 (目前) 全域資料表 API 操作

下表列出 2019.11.21 版 (目前) 全域資料表 API 操作針對資源型政策與跨帳戶存取權所提供的 API 層級支援。

2019.11.21 版 (目前) 全域資料表 API	資源型政策支援	跨帳戶支援
DescribeTableReplicaAutoScaling	是	否
UpdateTableReplicaAutoScaling	是	否

2017.11.29 版 (舊版) 全域資料表 API 操作

下表列出 2017.11.29 版 (舊版) 全域資料表 API 操作針對資源型政策與跨帳戶存取權所提供的 API 層級支援。

2017.11.29 版 (舊版) 全域資料表 API	資源型政策支援	跨帳戶支援
CreateGlobalTable	否	否
DescribeGlobalTable	否	否
DescribeGlobalTableSettings	否	否
ListGlobalTables	否	否
UpdateGlobalTable	否	否
UpdateGlobalTableSettings	否	否

標籤 API 操作

下表列出與資源型政策及跨帳戶存取權之標籤相關的 API 操作所提供的 API 層級支援。

標籤 API	資源型政策支援	跨帳戶支援
ListTagsOfResource	是	是
TagResource	是	是
UntagResource	是	是

備份與還原 API 操作

下表列出與資源型政策及跨帳戶存取權相關的備份與還原 API 操作所提供的 API 層級支援。

備份與還原 API	資源型政策支援	跨帳戶支援
CreateBackup	是	否
DescribeBackup	否	否
DeleteBackup	否	否
RestoreTableFromBackup	否	否

持續備份/還原 (PITR) API 操作

下表列出與資源型政策及跨帳戶存取權相關的持續備份/還原 (PITR) API 操作所提供的 API 層級支援。

持續備份/還原 (PITR) API	資源型政策支援	跨帳戶支援
DescribeContinuousBackups	是	否
RestoreTableToPointInTime	是	否
UpdateContinuousBackups	是	否

Contributor Insights API 操作

下表列出與資源型政策及跨帳戶存取權相關的持續備份/還原 (PITR) API 操作所提供的 API 層級支援。

Contributor Insights API	資源型政策支援	跨帳戶支援
DescribeContributorInsights	是	否
ListContributorInsights	否	否
UpdateContributorInsights	是	否

匯出 API 操作

下表列出匯出 API 操作針對資源型政策與跨帳戶存取權所提供的 API 層級支援。

匯出 API	資源型政策支援	跨帳戶支援
DescribeExport	否	否
ExportTableToPointInTime	是	否
ListExports	否	否

匯入 API 操作

下表列出匯入 API 操作針對資源型政策與跨帳戶存取權所提供的 API 層級支援。

匯入 API	資源型政策支援	跨帳戶支援
DescribeImport	否	否
ImportTable	否	否
ListImports	否	否

Amazon Kinesis Data Streams API 操作

下表列出 Kinesis Data Streams API 操作針對資源型政策與跨帳戶存取權所提供的 API 層級支援。

Kinesis API	資源型政策支援	跨帳戶支援
DescribeKinesisStreamingDestination	是	否
DisableKinesisStreamingDestination	是	否
EnableKinesisStreamingDestination	是	否
UpdateKinesisStreamingDestination	是	否

資源型政策 API 操作

下表列出資源型政策 API 操作針對資源型政策與跨帳戶存取權所提供的 API 層級支援。

資源型政策 API	資源型政策支援	跨帳戶支援
GetResourcePolicy	是	否
PutResourcePolicy	是	否
DeleteResourcePolicy	是	否

存留時間 API 操作

下表列出針對資源型政策與跨帳戶存取權，存留時間 (TTL) API 操作所提供的 API 層級支援。

TTL API	資源型政策支援	跨帳戶支援
DescribeTimeToLive	是	否

TTL API	資源型政策支援	跨帳戶支援
UpdateTimeToLive	是	否

其他 API 操作

下表列出其他 API 操作針對資源型政策與跨帳戶存取權所提供的 API 層級支援。

其他 API	資源型政策支援	跨帳戶支援
DescribeLimits	否	否
DescribeEndpoint	否	否
ListBackups	否	否
ListTables	否	否

DynamoDB Streams API 操作

下表列出 DynamoDB Streams API 針對資源型政策與跨帳戶存取權所提供的 API 層級支援。

DynamoDB Streams API	資源型政策支援	跨帳戶支援
DescribeStream	是	是
GetRecords	是	是
GetShardIterator	是	是
ListStreams	否	否

使用 IAM 身分型政策與 DynamoDB 資源型政策進行授權

身分型政策會附加至 IAM 身分 (IAM 使用者、群組或角色)。這些 IAM 政策文件可控制身分能在何種條件下，對哪些資源執行哪些操作。身分型政策可分為受管政策或內嵌政策。

資源型政策是附加至資源 (例如 DynamoDB 資料表) 的 IAM 政策文件。這些政策會授予指定的主體許可，允許在該資源上執行特定的動作，並且定義資源所適用的條件。例如，DynamoDB 資料表的資源型政策同時包含與該資料表關聯的索引。資源型政策是內嵌政策。不存在受管的資源型政策。

如需更多資訊，請參閱《IAM 使用者指南》中的身分型政策與資源型政策。

若 IAM 主體與資源擁有者屬於同一帳戶，資源型政策即可充分定義資源的存取權限。您仍可同時使用 IAM 身分型政策與資源型政策。對於跨帳戶存取權，您必須在身分政策和資源政策中明確允許存取，如 DynamoDB 中使用資源型政策的跨帳戶存取權 中所指定。當您同時使用這兩種類型的政策時，系統會依據判斷帳戶內是否允許或拒絕請求中的說明來進行評估。

> **Important**
>
> 如果身分型政策授予 DynamoDB 資料表 （ 例如，`dynamodb:GetItem`沒有條件 ） 的無條件存取權，則允許具有 條件的存取權的資源型政策`dynamodb:Attributes`不會限制該存取權。身分型政策的無條件允許優先，且資源型政策的條件不會套用為限制。若要限制對特定屬性的存取，請使用明確`Deny`陳述式，而不是僅依賴資源型政策中的條件式`Allow`陳述式。

DynamoDB 資源型政策範例

當您在資源型政策的 `Resource` 欄位中指定 ARN 時，僅當該 ARN 與其所附加的 DynamoDB 資源 ARN 相符時，政策才會生效。

> **Note**
>
> 請以資源特定資訊取代`##`文字。

資料表的資源型政策

下列資源型政策附加至名為 *`MusicCollection`* 的 DynamoDB 資料表，授予 IAM 使用者 *`John`* 和 *`Jane`* 在 *`MusicCollection`* 資源上執行 GetItem 與 BatchGetItem 動作的權限。

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
        "Effect": "Allow",
        "Principal": {
          "AWS": [
            "arn:aws:iam::111122223333:user/username",
            "arn:aws:iam::111122223333:user/Jane"
          ]
        },
        "Action": [
          "dynamodb:GetItem",
          "dynamodb:BatchGetItem"
        ],
        "Resource": [
          "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
        ]
    }
  ]
}
```

串流的資源型政策

下列資源型政策附加至名為 2024-02-12T18:57:26.492 的 DynamoDB 串流，授予 IAM 使用者 *John* 和 *Jane* 在 2024-02-12T18:57:26.492 資源上執行 GetRecords、GetShardIterator 與 DescribeStream API 動作的權限。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:user/username",
```

```
          "arn:aws:iam::111122223333:user/Jane"
        ]
      },
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

資源型政策：允許對指定資源執行所有動作的存取權

若要允許使用者對資料表及其所有相關索引執行所有動作，您可以使用萬用字元 (*) 來表示與該資料表相關的動作和資源。將萬用字元用於資源時，使用者即可存取 DynamoDB 資料表及其所有相關索引，包括尚未建立的索引。例如，下列政策將授予使用者 *John* 對 *MusicCollection* 資料表及其所有索引執行任何動作的權限，包括未來建立的索引。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/role-name"
      },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
index/index-name"
```

```
      ]
    }
  ]
}
```

跨帳戶存取權資源型政策

您可以為跨帳戶的 IAM 身分指定存取 DynamoDB 資源的權限。例如，您可能需要讓受信任帳戶的使用者讀取資料表內容，但僅能存取特定項目及其特定屬性。下列政策允許使用者 *John* 從信任 AWS 帳戶 ID *111111111111* 存取，以使用 GetItem API 從帳戶 *123456789012* 中的資料表存取資料。此政策可確保使用者僅能存取主索引鍵為 *Jane* 的項目，並且只能擷取屬性 `Artist` 和 `SongTitle`，無法讀取其他屬性。

> **Important**
>
> 如果未指定 `SPECIFIC_ATTRIBUTES` 條件，系統將傳回項目的所有屬性。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountTablePolicy",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:user/John"
            },
            "Action": "dynamodb:GetItem",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
```

```
                    ]
                },
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
}
```

除了上述資源型政策外，附加於使用者 *John* 的身分型政策也必須允許相關的 GetItem API 動作，才能使跨帳戶存取權運作。以下範例顯示您必須附加至使用者 *John* 的身分型政策。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountIdentityBasedPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
                    ]
                },
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
```

```
}
```

使用者 John 可以在 `table-name` 參數中指定資料表 ARN，提出 `GetItem` 請求以存取帳戶 *123456789012* 中的 *MusicCollection* 資料表。

```
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
```

含 IP 位址條件的資源型政策

您可以套用條件以限制來源 IP 位址、虛擬私有雲端 (VPC)，以及 VPC 端點 (VPCE)。您可以根據請求的來源 IP 位址指定權限。例如，您可能僅允許使用者從特定 IP 來源 (如公司 VPN 端點) 存取 DynamoDB 資源。請在 `Condition` 陳述式中指定這些 IP 位址。

下列範例允許使用者 *John* 在來源 IP 位址為 `54.240.143.0/24` 和 `2001:DB8:1234:5678::/64` 時，存取任何 DynamoDB 資源。

JSON

```
{
  "Id":"PolicyId2",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AllowIPmix",
      "Effect":"Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:user/username"
      },
      "Action":"dynamodb:*",
      "Resource":"*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "54.240.143.0/24",
            "2001:DB8:1234:5678::/64"
          ]
```

```
        }
      }
    }
  ]
}
```

您也可以拒絕所有對 DynamoDB 資源的存取，除非來源是特定的 VPC 端點，例如 *vpce-1a2b3c4d*。

> **Important**
>
> 當您將 DAX 與IPv6-only環境中具有 IP 資源政策的 DynamoDB 資料表搭配使用時，您必須設定其他存取規則。如果您的資源政策限制存取資料表`0.0.0.0/0`上的 IPv4 地址空間，您必須允許存取與 DAX 叢集相關聯的 IAM 角色。將 `ArnNotEquals`條件新增至您的政策，以確保 DAX 維持對 DynamoDB 資料表的存取。如需詳細資訊，請參閱 DAX 和 IPv6。

JSON

```
{
  "Id":"PolicyId",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AccessToSpecificVPCEOnly",
      "Principal": "*",
      "Action": "dynamodb:*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringNotEquals":{
          "aws:sourceVpce":"vpce-1a2b3c4d"
        }
      }
    }
  ]
}
```

使用 IAM 角色的資源型政策

您也可以在資源型政策中指定 IAM 服務角色。取得此角色的 IAM 實體，將受限於該角色中所指定的允許動作，以及資源型政策中定義的特定資源集合。

下列範例允許 IAM 實體在 *MusicCollection* 與 *MusicCollection* 兩個 DynamoDB 資源上執行所有 DynamoDB 動作。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/role-name" },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/*"
      ]
    }
  ]
}
```

DynamoDB 資源型政策考量事項

當您為 DynamoDB 資源定義資源型政策時，請注意以下考量事項：

一般考量事項

- 資源型政策文件的最大支援大小為 20 KB。DynamoDB 在依此限制計算政策大小時，會將空白字元納入計算。
- 成功更新某資源的政策後，該資源的後續政策更新會在 15 秒內被暫時封鎖。
- 目前，您只能將資源型政策附加至現有的串流。您無法在建立串流時同時附加政策。

全域資料表考量事項

- 全域資料表版本 2017.11.29 (舊版) 的副本不支援資源型政策。
- 若在資源型政策中拒絕 DynamoDB 服務連結角色 (SLR) 用於全域資料表資料複寫的動作，新增或刪除複本時將會失敗並產生錯誤。
- AWS::DynamoDB::GlobalTable 資源不支援在非部署堆疊更新所在的區域中，在同一次堆疊更新內建立複本並同時新增資源型政策。

跨帳戶考量事項

- 使用資源型政策的跨帳戶存取不支援具有 AWS 受管金鑰的加密資料表，因為您無法授予 AWS 受管 KMS 政策的跨帳戶存取權。

CloudFormation 考量事項

- 資源型政策不支援偏離偵測。如果您在 AWS CloudFormation 堆疊範本之外更新資源型政策，則需要使用變更來更新 CloudFormation 堆疊。
- 資源型政策不支援頻外變更。若您在 CloudFormation 範本之外新增、更新或刪除政策，而範本內未變更該政策，則這些變更不會被覆寫。

  例如，假設您的範本包含一項資源型政策，而您稍後在範本之外更新該政策。若未在範本中變更政策，DynamoDB 中更新的政策將不會與範本內的政策同步。

  反之，若範本不包含資源型政策，但您在範本之外新增了政策。只要您未將該政策新增至範本，它就不會從 DynamoDB 中被移除。當您將政策新增至範本並更新堆疊時，DynamoDB 中的現有政策會自動更新，使其與範本中定義的政策一致。

DynamoDB 資源型政策最佳實務

本主題說明定義 DynamoDB 資源存取權限及允許動作的最佳實務。

簡化 DynamoDB 資源的存取控制

如果需要存取 DynamoDB 資源的 AWS Identity and Access Management 主體與 AWS 帳戶 資源擁有者屬於相同部分，則每個主體不需要 IAM 身分型政策。附加至指定資源的資源型政策即可。此設定方式可簡化存取控制。

使用資源型政策保護您的 DynamoDB 資源

對所有 DynamoDB 資料表與串流建立資源型政策，以強制執行這些資源的存取控制。資源型政策可讓您在資源層級集中管理權限，簡化對 DynamoDB 資料表、索引及串流的存取控制，並降低管理負擔。若未為資料表或串流指定資源型政策，則除非與 IAM 主體關聯的身分型政策允許存取，否則系統會隱含拒絕存取該資料表或串流。

套用最低權限許可

當您使用 DynamoDB 資源的資源型政策設定權限時，僅授與執行動作所需的權限。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。探索工作負載或使用案例所需的許可時，您可能會從廣泛許可開始。隨著使用案例的成熟，您可以設法減少授予的許可，以便朝向最低權限的目標邁進。

分析跨帳戶存取活動以產生最低權限政策

IAM 存取分析器會報告資源型政策中指定之外部實體的跨帳戶存取，並提供可視化資訊，協助您最佳化權限設定並符合最低權限政策。如需政策產生的詳細資訊，請參閱 IAM Access Analyzer 政策產生。

使用 IAM 存取分析器根據存取活動產生最低權限政策

若只授予執行任務所需的許可，您可以根據在 AWS CloudTrail中記錄的存取活動產生政策。IAM 存取分析器會分析您的政策中所使用的服務與動作。

搭配 DynamoDB 使用屬性型存取控制

屬性型存取控制 (ABAC) 是一種授權策略，可根據身分型政策或其他 AWS 政策中的標籤條件定義存取許可，例如資源型政策和組織 IAM 政策。您可以將標籤附加至 DynamoDB 資料表，系統會根據標籤式條件進行評估。資料表所關聯的索引會繼承您新增至該資料表的標籤。每個 DynamoDB 資料表最多可新增 50 個標籤。資料表中所有標籤支援的大小上限為 10 KB。如需有關標記 DynamoDB 資源及標記限制的詳細資訊，請參閱在 DynamoDB 中標記資源與 the section called “DynamoDB 中的標記限制”。

如需使用標籤控制 AWS 資源存取的詳細資訊，請參閱《IAM 使用者指南》中的下列主題：

- 什麼是 ABAC AWS
- 使用標籤控制對 AWS 資源的存取

使用 ABAC，您可以為團隊與應用程式設定不同的存取層級，並以更少的政策在 DynamoDB 資料表上執行操作。您可以在 IAM 政策的條件元素中指定標籤，以控制對 DynamoDB 資料表或索引的存取。這些條件決定 IAM 主體、使用者或角色對 DynamoDB 資料表與索引的存取層級。當 IAM 主體向

DynamoDB 提出存取請求時，系統會根據 IAM 政策中的標籤條件評估資源與身分的標籤。之後，僅在符合標籤條件時，該政策才會生效。這可讓您建立能實現以下任一條件的 IAM 政策：

- 允許使用者僅管理具有標籤鍵*X*與值 *Y* 標籤的資源。
- 拒絕所有使用者存取具有標籤鍵 *X* 的資源。

例如，您可以建立政策，僅允許使用者更新具有下列標籤鍵值對的資料表：`"environment": "staging"`。您可以使用 aws：ResourceTag 條件鍵，根據附加至該資料表的標籤來允許或拒絕存取。

您可以在建立政策時或稍後使用 AWS 管理主控台、 AWS API、 AWS Command Line Interface (AWS CLI)、 AWS SDK 或 包含屬性型條件 AWS CloudFormation。

下列範例允許在名為 `MusicTable` 的資料表上執行 UpdateItem 動作，前提是該資料表具有名稱為 `environment`、值為 `production` 的標籤鍵值。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/MusicTable",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "production"
        }
      }
    }
  ]
}
```

主題

- 為什麼要使用 ABAC？
- 使用 DynamoDB 實作 ABAC 的條件鍵

- 搭配 DynamoDB 使用 ABAC 的注意事項
- 在 DynamoDB 中啟用 ABAC
- 在 DynamoDB 資料表與索引中使用 ABAC
- 使用 ABAC 搭配 DynamoDB 資料表與索引的範例
- 疑難排解 DynamoDB 資料表與索引的常見 ABAC 錯誤

為什麼要使用 ABAC？

- 更簡單的政策管理：您可以減少使用的政策數量，因為不需為每個 IAM 主體建立不同的政策來定義其存取層級。
- 可擴展的存取控制：使用 ABAC 可讓存取控制更容易擴展，因為在建立新的 DynamoDB 資源時，無需更新政策。您可以使用標籤授權存取那些擁有與資源標籤相符標籤的 IAM 主體。您可以新增 IAM 主體或 DynamoDB 資源，套用適當的標籤，即可自動授予必要的權限，無需修改任何政策。
- 細粒度權限管理：在建立政策時，最佳實務是授與最低權限。使用 ABAC，您可以為 IAM 主體建立標籤，並依據這些標籤授與符合的特定動作與資源的存取權限。
- 與公司目錄對應：您可以將標籤對應至公司目錄中的員工屬性，使存取控制政策與組織結構保持一致。

使用 DynamoDB 實作 ABAC 的條件鍵

您可以在 AWS 政策中使用下列條件索引鍵來控制 DynamoDB 資料表和索引的存取層級：

- aws:ResourceTag/tag-key：根據 DynamoDB 資料表或索引上的標籤鍵值組是否與政策中的標籤鍵和值相符來控制存取。此條件鍵適用於所有針對現有資料表或索引運作的 API。

  這些 `dynamodb:ResourceTag` 條件的評估方式與資源未附加任何標籤時相同。
- aws:RequestTag/tag-key：可用於比較請求中傳遞的標籤鍵值組與政策中指定的標籤鍵值組。此條件鍵適用於在請求承載中包含標籤的 API。包含這些 API：CreateTable 與 TagResource。
- aws:TagKeys：用於比較請求中的標籤鍵與政策中指定的標籤鍵。此條件鍵適用於在請求承載中包含標籤的 API。相關 API 包含 `CreateTable`、`TagResource` 與 `UntagResource`。

搭配 DynamoDB 使用 ABAC 的注意事項

當您將 ABAC 用於 DynamoDB 資料表或索引時，請注意以下事項：

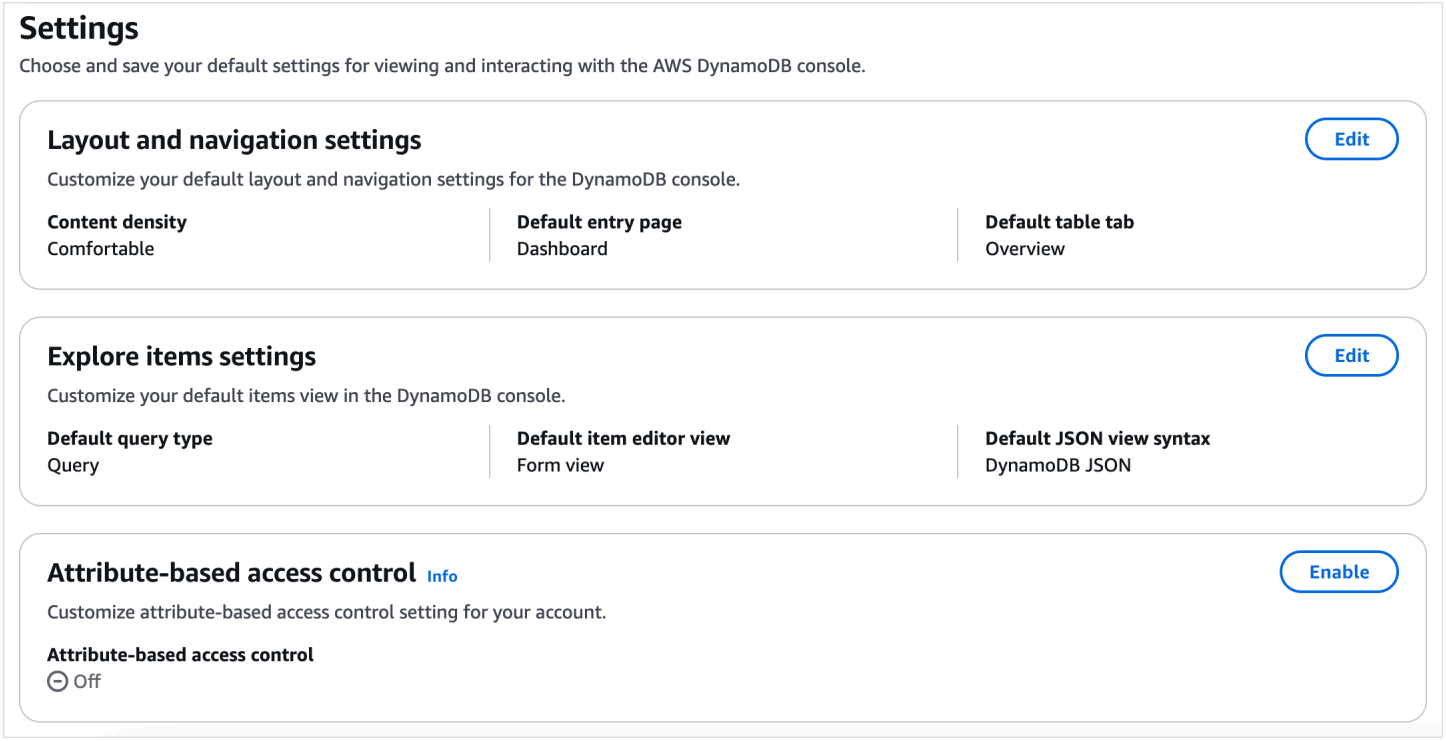
- DynamoDB Streams 不支援標籤與 ABAC。
- DynamoDB 備份不支援標籤與 ABAC。若要在備份中使用 ABAC，建議您使用 AWS Backup。
- 標籤不會在還原的資料表中保留。您必須先將標籤新增至還原的資料表，才能在政策中使用標籤式條件。

在 DynamoDB 中啟用 ABAC

對於大多數 AWS 帳戶，預設為啟用 ABAC。使用 DynamoDB 主控台，您可以確認帳戶是否已啟用 ABAC。若要執行此操作，請使用具有 dynamodb：GetAbacStatus 權限的角色開啟 DynamoDB 主控台。接著，開啟 DynamoDB 主控台的設定頁面。

若您未看到屬性型存取控制卡片，或卡片顯示狀態為開啟，即表示您的帳戶已啟用 ABAC。不過，如果您看到屬性型存取控制卡片顯示狀態為關閉 (如下圖所示)，表示帳戶未啟用 ABAC。

屬性型存取控制 – 未啟用



ABAC 不會 AWS 帳戶 為其身分型政策或其他政策中指定的標籤型條件啟用。若您的帳戶未啟用 ABAC，政策中針對 DynamoDB 資料表或索引設計的標籤式條件，將會以資源或 API 請求不含任何標籤的狀態進行評估。當您為帳戶啟用 ABAC 時，系統會依據附加至資料表或 API 請求的標籤，評估帳戶中政策內的標籤式條件。

若要啟用帳戶的 ABAC，我們建議您先依 政策稽核 一節所述稽核您的政策。接著，在 IAM 政策中加入 ABAC 所需的權限。最後，依 在主控台啟用 ABAC 所述步驟，在目前區域為帳戶啟用 ABAC。啟用 ABAC 後，您可在加入後七個日曆天內選擇退出。

主題

- 啟用 ABAC 前稽核政策
- 啟用 ABAC 所需的 IAM 權限
- 在主控台啟用 ABAC

啟用 ABAC 前稽核政策

在啟用帳戶的 ABAC 前，請稽核政策，確認帳戶內的標籤式條件均已按預期設定。稽核政策可協助避免啟用 ABAC 後，DynamoDB 工作流程中出現授權變更的意外情況。若要查看使用屬性型條件與標籤的範例，以及 ABAC 實作前後的行為變化，請參閱 使用 ABAC 搭配 DynamoDB 資料表與索引的範例。

啟用 ABAC 所需的 IAM 權限

您需要 dynamodb:UpdateAbacStatus 權限，才能在目前區域為帳戶啟用 ABAC。若要確認帳戶是否已啟用 ABAC，您也必須具備 dynamodb:GetAbacStatus 權限。具備此權限後，您可在任何區域檢視帳戶的 ABAC 狀態。除了存取 DynamoDB 主控台所需的權限外，您還需要這些額外權限。

下列 IAM 政策授予權限，可啟用 ABAC 並檢視目前區域內帳戶的狀態。

```
{
"version": "2012-10-17",        &TCX5-2025-waiver;
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateAbacStatus",
                "dynamodb:GetAbacStatus"
             ],
            "Resource": "*"
        }
    ]
}
```

在主控台啟用 ABAC

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 從頂端導覽窗格選取要啟用 ABAC 的區域。
3. 在左側的導覽窗格中，選擇設定。
4. 在設定頁面執行以下動作：

   a. 在屬性型存取控制卡片中，選取啟用。

   b. 在確認屬性型存取控制設定對話方塊中，選取啟用以確認設定。

   這將在目前區域啟用 ABAC，且屬性型存取控制卡片會顯示狀態為開啟。

   若您在主控台啟用 ABAC 後想退出，可在加入後七個日曆天內執行。若要退出，請前往設定頁面，於屬性型存取控制卡片中選取停用。

   > **Note**
   >
   > 更新 ABAC 狀態屬於非同步作業。若政策中的標籤未立即生效，可能需等待一段時間，因為變更的套用會最終一致。

在 DynamoDB 資料表與索引中使用 ABAC

以下步驟示範如何使用 ABAC 設定權限。在此範例中，您將新增標籤至 DynamoDB 資料表，並建立包含標籤式條件的 IAM 角色。接著，您將透過比對標籤條件，測試 DynamoDB 資料表上可用的權限。

主題

- 步驟 1：將標籤新增至 DynamoDB 資料表
- 步驟 2：建立包含標籤式條件的 IAM 角色與政策
- 步驟 3：測試允許的權限

步驟 1：將標籤新增至 DynamoDB 資料表

您可以使用 AWS 管理主控台、 AWS API、 AWS Command Line Interface (AWS CLI)、 AWS SDK 或 將標籤新增至新的或現有的 DynamoDB 資料表 AWS CloudFormation。例如，下列 tag-resource CLI 指令會將標籤新增至名為 `MusicTable` 的資料表。

```
aws dynamodb tag-resource –resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/
MusicTable –tags Key=environment,Value=staging
```

步驟 2：建立包含標籤式條件的 IAM 角色與政策

使用 aws：ResourceTag/tag-key 條件金鑰建立 IAM 政策，以比較政策中指定的標籤鍵值對與附加至資料表的鍵值對。下列範例政策允許使用者在資料表中放入或更新項目，前提是該資料表包含下列標籤鍵值對："environment": "staging"。若資料表未具指定的標籤鍵值對，系統將拒絕執行這些操作。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": "staging"
                }
            }
        }
    ]
}
```

步驟 3：測試允許的權限

1. 將 IAM 政策附加至 AWS 帳戶中的測試使用者或角色。確認您使用的 IAM 主體未經由其他政策擁有對 DynamoDB 資料表的存取權限。
2. 確認您的 DynamoDB 資料表包含值為 "staging" 的 "environment" 標籤鍵。
3. 在已標記的資料表上執行 dynamodb:PutItem 和 dynamodb:UpdateItem 動作。若 "environment": "staging" 標籤鍵值對存在，這些動作應可成功執行。

若在不含 "environment": "staging" 標籤鍵值對的資料表上執行這些動作，請求將失敗並返回 AccessDeniedException。

您也可以檢閱下一節中說明的其他範例使用案例，以實作 ABAC 並進行更多測試。

使用 ABAC 搭配 DynamoDB 資料表與索引的範例

下列範例展示使用標籤實作屬性式條件的多種使用案例。

主題

- 範例 1：使用 aws:ResourceTag 允許動作
- 範例 2：使用 aws:RequestTag 允許動作
- 範例 3：使用 aws:TagKeys 拒絕動作

範例 1：使用 aws:ResourceTag 允許動作

使用 aws:ResourceTag/tag-key 條件鍵，您可以比較 IAM 政策中指定的標籤鍵值對與附加在 DynamoDB 資料表中的鍵值對。例如，若 IAM 政策與資料表的標籤條件相符，即可允許執行特定動作，如 PutItem。請依下列步驟操作：

Using the AWS CLI

1. 建立資料表。下列範例使用 create-table AWS CLI 命令來建立名為 的資料表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. 為此資料表新增一個標籤。下列 tag-resource AWS CLI 命令範例會將標籤鍵值對新增至 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. 建立內嵌政策並將其新增至已連接 AmazonDynamoDBReadOnlyAccess AWS 受管政策的角色，如下列範例所示。

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": "dynamodb:PutItem",
         "Resource": "arn:aws:dynamodb:*:*:table/*",
         "Condition": {
           "StringEquals": {
             "aws:ResourceTag/Title": "ProductManager"
           }
         }
       }
     ]
   }
   ```

   當附加至資料表的標籤鍵和值與政策中指定的標籤相符時，此政策允許在該資料表上執行 PutItem 動作。

4. 依步驟 3 中說明的政策假設該角色。

5. 使用 put-item AWS CLI 命令將項目放入 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. 掃描資料表以確認項目是否已成功新增。

```
aws dynamodb scan --table-name myMusicTable  --region us-east-1
```

Using the AWS SDK for Java 2.x

1. 建立資料表。下列範例使用 CreateTable API 建立名為 myMusicTable 的資料表。

```
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. 為此資料表新增一個標籤。下列範例中，TagResource API 會將標籤鍵值對 Title: ProductManager 新增至 myMusicTable。

```
TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
```

```
    .resourceArn(tableArn)
    .tags(
        Arrays.asList(
            Tag.builder()
            .key("Title")
            .value("ProductManager")
            .build()
        )
    )
    .build();
dynamoDB.tagResource(tagResourceRequest);
```

3. 建立內嵌政策並將其新增至已連接 AmazonDynamoDBReadOnlyAccess AWS 受管政策的角色，如下列範例所示。

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": "dynamodb:PutItem",
         "Resource": "arn:aws:dynamodb:*:*:table/*",
         "Condition": {
           "StringEquals": {
             "aws:ResourceTag/Title": "ProductManager"
           }
         }
       }
     ]
   }
   ```

   當附加至資料表的標籤鍵和值與政策中指定的標籤相符時，此政策允許在該資料表上執行 `PutItem` 動作。

4. 依步驟 3 中說明的政策假設該角色。

5. 使用 PutItem API 將項目新增至 `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. 掃描資料表以確認項目是否已成功新增。

```
ScanRequest scanRequest = ScanRequest.builder()
                .tableName(tableName)
                .build();

ScanResponse scanResponse = dynamoDB.scan(scanRequest);
```

Using the 適用於 Python (Boto3) 的 AWS SDK

1. 建立資料表。下列範例使用 CreateTable API 建立名為 myMusicTable 的資料表。

```
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. 為此資料表新增一個標籤。下列範例中，TagResource API 會將標籤鍵值對 `Title: ProductManager` 新增至 `myMusicTable`。

```
tag_resouce_response = ddb_client.tag_resource(
    ResourceArn=table_arn,
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ]
)
```

3. 建立內嵌政策並將其新增至已連接 AmazonDynamoDBReadOnlyAccess AWS 受管政策的角色，如下列範例所示。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Action": "dynamodb:PutItem",
        "Resource": "arn:aws:dynamodb:*:*:table/*",
        "Condition": {
            "StringEquals": {
            "aws:ResourceTag/Title": "ProductManager"
            }
        }
        }
    ]
    }
```

當附加至資料表的標籤鍵和值與政策中指定的標籤相符時，此政策允許在該資料表上執行 PutItem 動作。

4. 依步驟 3 中說明的政策假設該角色。
5. 使用 PutItem API 將項目新增至 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. 掃描資料表以確認項目是否已成功新增。

```
scan_response = client.scan(
    TableName='myMusicTable'
)
```

未啟用 ABAC

如果您的 未啟用 ABAC AWS 帳戶，則 IAM 政策和 DynamoDB 資料表中的標籤條件不相符。因此，PutItem 動作會因 AmazonDynamoDBReadOnlyAccess 政策的影響而返回 AccessDeniedException。

```
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.
```

啟用 ABAC

如果您的 已啟用 ABAC AWS 帳戶，則put-item動作會成功完成，並將新項目新增至您的資料表。這是因為當 IAM 政策與資料表中的標籤條件相符時，資料表中的內嵌政策會允許執行 PutItem 動作。

範例 2：使用 aws:RequestTag 允許動作

使用 aws：RequestTag/tag-key 條件金鑰，您可以比較請求中傳遞的標籤鍵值對與 IAM 政策中指定的鍵值對。例如，若標籤條件不相符，您可以使用 aws:RequestTag 來允許特定動作 (如 CreateTable)。請依下列步驟操作：

Using the AWS CLI

1. 建立內嵌政策並將其新增至已連接 AmazonDynamoDBReadOnlyAccess AWS 受管政策的角色，如下列範例所示。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
    ]
}
```

2. 建立一個包含 "Owner": "John" 標籤鍵值對的資料表。

```
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 適用於 Python (Boto3) 的 AWS SDK

1. 建立內嵌政策並將其新增至已連接 AmazonDynamoDBReadOnlyAccess AWS 受管政策的角色，如下列範例所示。

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "dynamodb:CreateTable",
                   "dynamodb:TagResource"
               ],
               "Resource": "arn:aws:dynamodb:*:*:table/*",
               "Condition": {
                   "StringEquals": {
                       "aws:RequestTag/Owner": "John"
                   }
               }
           }
       ]
   }
   ```

2. 建立一個包含 "Owner": "John" 標籤鍵值對的資料表。

   ```
   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'
        },
    ],
)
```

未啟用 ABAC

如果您的 未啟用 ABAC AWS 帳戶，則內嵌政策和 DynamoDB 資料表中的標籤條件不相符。因此，CreateTable 請求會失敗，資料表將不會建立。

```
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.
```

啟用 ABAC

如果您的 已啟用 ABAC AWS 帳戶，則您的資料表建立請求會成功完成。由於 "Owner": "John" 的標籤鍵值對存在於 CreateTable 請求中，內嵌政策允許使用者 John 執行 CreateTable 動作。

範例 3：使用 aws:TagKeys 拒絕動作

使用 aws：TagKeys 條件金鑰，您可以比較請求中的標籤金鑰與 IAM 政策中指定的金鑰。例如，若請求中未包含特定標籤鍵 not，您可以使用 aws:TagKeys 來拒絕特定動作 (如 CreateTable)。請依下列步驟操作：

Using the AWS CLI

1. 將客戶受管政策新增至已連接 AmazonDynamoDBFullAccess AWS 受管政策的角色，如下列範例所示。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "ForAllValues:StringNotEquals": {
                    "aws:TagKeys": "CostCenter"
                }
            }
        }
    ]
}
```

2. 以附加該政策的角色身分建立含有標籤鍵 `Title` 的資料表。

```
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 適用於 Python (Boto3) 的 AWS SDK

1. 將客戶受管政策新增至已連接 AmazonDynamoDBFullAccess AWS 受管政策的角色，如下列範例所示。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "ForAllValues:StringNotEquals": {
                    "aws:TagKeys": "CostCenter"
                }
            }
        }
    ]
}
```

2. 以附加該政策的角色身分建立含有標籤鍵 `Title` 的資料表。

```
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'
        },
    ],
)
```

未啟用 ABAC

如果您的 未啟用 ABAC AWS 帳戶，DynamoDB 不會將create-table命令中的標籤金鑰傳送至 IAM。Null 條件可確保當請求中沒有標籤鍵時，條件的評估結果為 false。由於 Deny 政策不相符，create-table 指令將成功完成。

啟用 ABAC

如果您的 已啟用 ABAC AWS 帳戶，則 create-table命令中傳遞的標籤金鑰會傳遞給 IAM。標籤鍵 Title 會根據 Deny 政策中所設定的條件型標籤鍵 CostCenter 進行比對。由於 StringNotEquals 運算子，Title 標籤鍵與 Deny 政策中的標籤鍵不相符。因此，CreateTable 動作將失敗，資料表不會建立。執行 create-table 指令時會傳回 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.
```

疑難排解 DynamoDB 資料表與索引的常見 ABAC 錯誤

本主題提供在 DynamoDB 資料表或索引中實作 ABAC 時，可能遇到的常見錯誤與問題的疑難排解建議。

政策中使用服務特定條件金鑰會導致錯誤

服務特定條件金鑰不被視為有效的條件金鑰。若在政策中使用此類金鑰，將導致錯誤。若要解決此問題，請將服務特定條件金鑰替換為合適的條件金鑰，以在 DynamoDB 中實作 ABAC。

例如，假設您在執行 PutItem 請求的內嵌政策中使用了 dynamodb:ResourceTag 條件金鑰。假設請求因 AccessDeniedException 而失敗。下列範例顯示包含 dynamodb:ResourceTag 條件金鑰的錯誤內嵌政策。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "dynamodb:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

若要解決此問題，請將 dynamodb:ResourceTag 條件金鑰替換為 aws:ResourceTag，如下例所示。

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

無法停用 ABAC

如果您的帳戶已透過 啟用 ABAC 支援，您將無法透過 DynamoDB 主控台選擇退出 ABAC。若要停用，請聯絡 支援。

僅當 only if 下列條件成立時，您才可自行停用 ABAC：

- 您已使用透過 DynamoDB 主控台選擇加入的自助式流程。
- 您在選擇加入後七天內停用。

DynamoDB 中的資料保護

Amazon DynamoDB 提供高耐用性儲存基礎設施，專為任務關鍵型及主要資料儲存體所設計。資料會以備援方式存放在 Amazon DynamoDB 區域中多個設施的多部裝置上。

DynamoDB 會保護靜態儲存的使用者資料，以及內部部署用戶端與 DynamoDB 之間，以及 DynamoDB 與相同 AWS 區域內其他 AWS 資源之間傳輸中的資料。

主題

- DynamoDB 靜態加密
- 使用 VPC 端點與 IAM 政策強化 DynamoDB 連線安全性

DynamoDB 靜態加密

儲存在 Amazon DynamoDB 中的所有使用者資料都會完全靜態加密。DynamoDB 靜態加密使用存放在 AWS Key Management Service (AWS KMS) 的加密金鑰來加密所有靜態資料，藉此提供加強的安全性。此功能協助降低了保護敏感資料所涉及的操作負擔和複雜性。您可以透過靜態加密，建立符合嚴格加密合規和法規要求，而且對安全性要求甚高的應用程式。

每當資料存放到耐用的媒體時，DynamoDB 靜態加密就會透過保護加密資料表中的資料：包括其主索引鍵、本機及全域次要索引、串流、全域資料表、備份和 DynamoDB Accelerator (DAX) 叢集，來提供另一層資料保護。組織政策、行業或政府法規，以及合規要求可能會經常需要使用靜態加密來增加應用程式的資料安全性。如需有關資料庫應用程式加密的詳細資訊，請參閱 AWS 資料庫加密 SDK。

靜態加密與 整合 AWS KMS ，用於管理用來加密資料表的加密金鑰。如需金鑰類型與狀態的詳細資訊，請參閱AWS Key Management Service 開發人員指南中的 AWS Key Management Service 概念。

建立新資料表時，您可以選擇下列其中一種 AWS KMS key 類型來加密資料表。您可以隨時在這些金鑰類型之間切換。

- AWS 擁有的金鑰 – 預設加密類型。此金鑰是由 DynamoDB 所擁有 (不需額外費用)。
- AWS 受管金鑰 – 金鑰會存放在您的帳戶中，並由 管理 AWS KMS （需AWS KMS 付費）。
- 客戶自管金鑰 – 金鑰會存放在您的帳戶中，並且由您建立、擁有且管理。您可以完全控制 KMS 金鑰（需支付AWS KMS 費用）。

如需金鑰類型的詳細資訊，請參閱客戶金鑰和 AWS 金鑰。

> **Note**
>
> - 建立啟用靜態加密的新 DAX 叢集時， AWS 受管金鑰 會用來加密叢集中的靜態資料。
> - 如果您的資料表有排序索引鍵，則某些標示範圍界限的排序索引鍵將會以純文字的格式存放在資料表中繼資料中。

當您存取加密的資料表，DynamoDB 會以透明方式解密資料表資料。您不必變更任何代碼或應用程式來使用或管理加密的資料表。DynamoDB 會繼續提供您預期的個位數毫秒的相同延遲，且所有 DynamoDB 查詢會在加密資料上順暢地運作。

您可以使用 AWS 管理主控台、 AWS Command Line Interface (AWS CLI) 或 Amazon DynamoDB API，在建立新資料表或切換現有資料表上的加密金鑰時指定加密金鑰。若要了解作法，請參閱在 DynamoDB 中管理加密資料表。

使用 AWS 擁有的金鑰 進行靜態加密是免費的。不過， AWS KMS AWS 受管金鑰 和 適用於客戶受管金鑰。如需定價的詳細資訊，請參閱 AWS KMS 定價。

DynamoDB 靜態加密適用於所有 AWS 區域，包括 AWS 中國（北京）和 AWS 中國（寧夏）區域和 AWS GovCloud（美國）區域。如需詳細資訊，請參閱DynamoDB 靜態加密運作原理及DynamoDB 靜態加密使用須知。

DynamoDB 靜態加密運作原理

Amazon DynamoDB 靜態加密功能會使用 256 位元的進階加密標準 (AES-256) 來加密您的資料，此項標準有助於防止對底層儲存未經授權的存取，進而保護您的資料。

靜態加密與 AWS Key Management Service (AWS KMS) 整合，用於管理用來加密資料表的加密金鑰。

> **Note**
>
> 在 2022 年 5 月， 的輪換排程 AWS 受管金鑰 從每三年（約 1，095 天） AWS KMS 變更為每年（約 365 天）。
> 新的 AWS 受管金鑰 會在建立後自動輪換一年，之後大約每年輪換一次。
> 現有的 AWS 受管金鑰 會在最近一次輪換後一年自動輪換，之後每年輪換一次。

AWS 擁有的金鑰

AWS 擁有的金鑰 不會存放在您的帳戶中 AWS 。它們是 KMS 金鑰集合的一部分，其 AWS 擁有和管理用於多個 AWS 帳戶。 AWS 服務可以使用 AWS 擁有的金鑰 來保護您的資料。DynamoDB AWS 擁有的金鑰 使用的 會每年輪換（約 365 天）。

您無法檢視、管理或使用 AWS 擁有的金鑰或稽核其使用方式。不過，您不需要執行任何工作或變更任何程式，即可保護加密您的資料的金鑰。

您不需要支付每月費用或使用費 AWS 擁有的金鑰，也不會計入您帳戶的 AWS KMS 配額。

AWS 受管金鑰

AWS 受管金鑰 是您帳戶中的 KMS 金鑰，由與 整合的 AWS 服務代表您建立、管理和使用 AWS KMS。您可以檢視您的帳戶中的 AWS 受管金鑰 ，檢視其金鑰政策，以及在 AWS CloudTrail 日誌中稽核其使用方式。不過，您無法管理這些 KMS 金鑰或變更其許可。

靜態加密會自動與 整合 AWS KMS ，以管理用於加密資料表 AWS 受管金鑰 的 DynamoDB (aws/dynamodb) 專用 。如果在建立加密的 DynamoDB 資料表時 AWS 受管金鑰 不存在 ， AWS KMS 會自動為您建立新的金鑰。此金鑰會與未來建立的加密資料表搭配使用。 AWS KMS 結合安全、高可用性的硬體和軟體，以提供針對雲端擴展的金鑰管理系統。

如需管理 許可的詳細資訊 AWS 受管金鑰，請參閱《 開發人員指南》中的授權使用 AWS 受管金鑰 。AWS Key Management Service

客戶自管金鑰

客戶受管金鑰是您帳戶中建立、擁有和管理 AWS 的 KMS 金鑰。您可以完全控制這些 KMS 金鑰，包括建立和維護其金鑰政策、IAM 政策和授與、啟用和停用它們、輪換其密碼編譯材料、新增標籤、建立參考別名，以及排程供刪除。如需管理客戶自管金鑰權限的詳細資訊，請參閱客戶自管金鑰。

當您指定一個客戶自管金鑰作為資料表層級加密金鑰時，DynamoDB 資料表、本機和全域次要索引及串流皆會以相同的客戶自管金鑰加密。隨需備份會以於備份建立時指定的資料表層級加密金鑰加密。更新資料表層級加密金鑰並不會變更與現有隨需備份相關的加密金鑰。

將客戶自管金鑰的狀態設為停用，或排定其刪除時間，可避免所有使用者和 DynamoDB 服務可在資料表上加密或解密資料，以及執行讀取或寫入操作。DynamoDB 必須能夠存取您的加密金鑰，才能確保您可以繼續存取資料表並避免資料遺失。

如果您停用客戶自管金鑰或排定其刪除的時間，您的資料表狀態就會成為 Inaccessible (無法存取)。為確保您可以持續使用資料表進行作業，您必須在七天內為指定的加密金鑰提供 DynamoDB 存取權限。只要服務偵測到您的加密金鑰無法存取，DynamoDB 就會寄送電子通知提醒您。

> **Note**
>
> - 如果您的客戶自管金鑰仍然無法由 DynamoDB 服務存取且時間超過七天，資料表就會存檔並無法再存取。DynamoDB 會建立資料表的隨需備份，並會向您收取費用。您可以使用隨需備份還原您的資料至新的資料表。若要開始還原，最後一個在資料表中的客戶自管金鑰必須啟用，且 DynamoDB 必須可以存取該資料表。

- 如果用來加密全域資料表複本的客戶自管金鑰無法存取，DynamoDB 會從複寫群組中移除此複本。將不會刪除複本，且複寫將會在偵測到客戶自管金鑰為無法存取後 20 小時停止進出此區域。

如需更多詳細資訊，請參閱啟用金鑰和刪除金鑰。

使用 的注意事項 AWS 受管金鑰

除非 Amazon DynamoDB 可存取存放在您 AWS KMS 帳戶中的 KMS 金鑰，否則無法讀取您的資料表資料。DynamoDB 使用封套加密和金鑰階層來加密資料。您的 AWS KMS 加密金鑰用於加密此金鑰階層的根金鑰。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的封套加密。

DynamoDB 不會 AWS KMS 呼叫每個 DynamoDB 操作。對於每個具活躍流量的用戶端連線，金鑰會每 5 分鐘重新整理一次。

請確定您已設定好 SDK，來重複使用連線。否則，您會遇到來自 DynamoDB 的延遲，必須為每個 DynamoDB 操作重新建立新的 AWS KMS 快取項目。此外，您可能必須面對更高的 AWS KMS 和 CloudTrail 成本。例如，若要使用 Node.js SDK 來執行這項動作，您可以在 `keepAlive` 開啟時，建立新的 HTTPS 代理。如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》中的在 Node.js 中設定 keepAlive。

DynamoDB 靜態加密使用須知

在您使用 Amazon DynamoDB 中靜態加密時請考量以下項目。

所有資料表資料都會加密

已對所有 DynamoDB 資料表資料啟用伺服器端靜態加密，且無法停用。您無法僅加密資料表中的項目子集。

只有持久性儲存媒體上的資料是靜態 (靜止) 時，靜態加密才會加密資料。如果對傳輸中資料或使用中資料有資料安全性的疑慮，您可能需要進行額外的措施：

- 傳輸中的資料：DynamoDB 中的所有資料都是在傳輸中加密的。根據預設，與 DynamoDB 之間的通訊會使用 HTTPS 協定，其使用 Secure Sockets Layer (SSL)/Transport Layer Security (TLS) 加密保護網路流量。
- 使用中資料：在將資料傳送至 DynamoDB 前使用用戶端加密來保護資料。如需詳細資訊，請參閱《Amazon DynamoDB Encryption Client 開發人員指南》中的用戶端與伺服器端加密。

您可以將串流與加密的資料表搭配著使用。DynamoDB Streams 一律會以資料表層級加密金鑰加密。如需詳細資訊，請參閱 DynamoDB Streams 的變更資料擷取。

DynamoDB 的備份會受到加密，且從備份恢復的資料表也已啟用加密。您可以使用 AWS 擁有的金鑰 AWS 受管金鑰或客戶受管金鑰來加密備份資料。如需詳細資訊，請參閱DynamoDB 的備份與還原。

本機次要索引和全域次要索引會使用與基礎資料表相同的金鑰加密。

加密類型

> **Note**
>
> Global Table 版本 2017 不支援客戶自管金鑰。如果要在 DynamoDB Global Table 中使用客戶自管金鑰，則需要將資料表升級為 Global Table 版本 2019，然後將其啟用。

在 上 AWS 管理主控台，加密類型是當您使用 AWS 受管金鑰 或客戶受管金鑰來加密資料KMS時。使用 AWS 擁有的金鑰時，加密類型為 DEFAULT。在 Amazon DynamoDB API 中，加密類型是當您使用 AWS 受管金鑰 或客戶受管金鑰KMS時。在缺少加密類型的情況下，會使用 AWS 擁有的金鑰來加密資料。您可以隨時在 AWS 擁有的金鑰 AWS 受管金鑰和客戶受管金鑰之間切換。您可以使用 主控台、AWS Command Line Interface (AWS CLI) 或 Amazon DynamoDB API 來切換加密金鑰。

請注意，使用客戶自管金鑰時有下列限制：

- 您無法搭配 DynamoDB Accelerator (DAX) 叢集使用客戶自管金鑰。如需詳細資訊，請參閱 DAX 靜態加密。
- 您可以使用客戶自管金鑰來加密使用交易的資料表。不過，為了確保交易傳播的持久性，服務會暫時儲存交易請求的副本，並使用 AWS 擁有的金鑰進行加密。資料表和次要索引中已遞交的資料一律會使用客戶自管金鑰進行靜態加密。
- 您可以使用客戶自管金鑰來加密使用 Contributor Insights 的資料表。不過，傳輸至 的資料 Amazon CloudWatch 會使用 加密 AWS 擁有的金鑰。
- 當您轉換到新的客戶受管金鑰時，請務必保持啟用原始金鑰，直到程序完成為止。 AWS 仍然需要原始金鑰來解密資料，才能使用新的金鑰進行加密。當資料表的 SSE 描述狀態為「啟用」，且顯示新客戶自管金鑰的 KMSMasterKeyArn 時，程序就會完成。此時可以停用或排程刪除原始金鑰。
- 顯示新的客戶自管金鑰後，資料表和任何新的隨需備份都會使用新的金鑰加密。
- 任何現有的隨需備份都會使用建立備份時使用的客戶自管金鑰進行加密。您將需要相同的金鑰來還原這些備份。您可以使用 DescribeBackup API 來檢視該備份的 SSE 描述，識別建立每個備份的期間的金鑰。

- 如果您停用客戶自管金鑰或排定其刪除的時間，所有在 DynamoDB Streams 中的資料仍有 24 小時的生命週期。任何未擷取的活動資料如果超過 24 小時，則符合裁剪的資格。
- 如果您停用客戶自管金鑰或排定其刪除的時間，存留時間 (TTL) 刪除就會持續 30 分鐘。這些 TTL 刪除會持續發出至 DynamoDB Streams，並符合標準裁剪/保留間隔。

如需更多詳細資訊，請參閱啟用金鑰和刪除金鑰。

使用 KMS 金鑰和資料金鑰

DynamoDB 靜態加密功能使用 AWS KMS key 和資料金鑰階層來保護您的資料表資料。當 DynamoDB 串流、全域資料表和備份寫入到持久性媒體時，DynamoDB 會使用相同的金鑰階層來保護 DynamoDB 串流、全域資料表和備份。

我們建議您先規劃加密策略，再於 DynamoDB 中使用資料表。如果您將敏感或機密資料儲存在 DynamoDB，請考慮在計劃中加入用戶端加密。如此一來，您就能盡量靠近資料來源進行加密，並確保資料在整個生命週期受到保護。如需詳細資訊，請參閱 DynamoDB 加密客戶端文件。

AWS KMS key

靜態加密會保護 AWS KMS key下的 DynamoDB 資料表。根據預設，DynamoDB 會使用 AWS 擁有的金鑰，這是一種在 DynamoDB 服務帳戶中建立和管理的多租用戶加密金鑰。但是，您可以使用客戶自管金鑰或 AWS 帳戶中 DynamoDB (aws/dynamodb) 加密您的 DynamoDB 資料表。您可以為每個資料表選取不同的 KMS 金鑰。您為資料表選取的 KMS 金鑰也會用來加密其本機和全域次要索引、串流和備份。

建立或更新資料表時，您可以選取資料表的 KMS 金鑰。您可以隨時在 DynamoDB 主控台或使用 UpdateTable 操作變更資料表的 KMS 金鑰。切換金鑰的程序是無縫的，且不需要停機時間或降低服務效能。

> **Important**
>
> DynamoDB 僅支援對稱 KMS 金鑰。您無法使用非對稱 KMS 金鑰來加密您的 DynamoDB 資料表。

使用客戶自管金鑰來取得下列功能：

- 您可以建立和管理 KMS 金鑰，包括設定金鑰政策、IAM 政策和授予來控制對 KMS 金鑰的存取。您可以啟用和停用 KMS 金鑰、啟用和停用自動金鑰輪換，以及於不再使用時刪除 KMS 金鑰。

- 您可以搭配匯入的金鑰材料使用客戶自管金鑰，或是使用位於您所擁有及管理自訂金鑰存放區中的客戶自管金鑰。
- 您可以在 AWS CloudTrail 日誌 AWS KMS 中檢查對 的 DynamoDB API 呼叫，以稽核 DynamoDB 資料表的加密和解密。

AWS 受管金鑰 如果您需要以下任何功能，請使用 ：

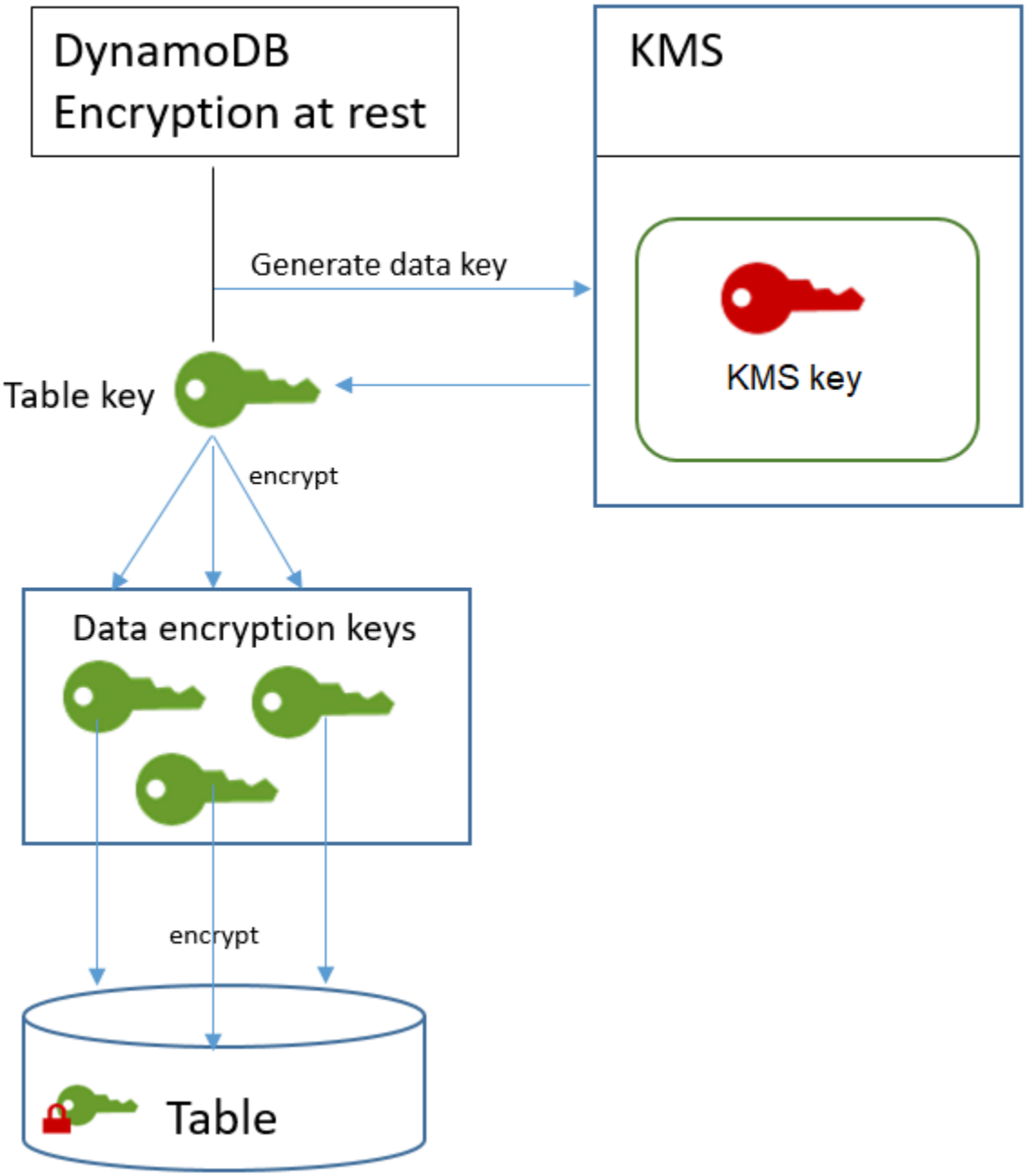
- 您可以檢視 KMS 金鑰和檢視其金鑰政策。(您無法變更金鑰政策。)
- 您可以在AWS CloudTrail 日誌 AWS KMS 中檢查對 的 DynamoDB API 呼叫，以稽核 DynamoDB 資料表的加密和解密。

不過， AWS 擁有的金鑰 是免費的，其使用不會計入AWS KMS 資源或請求配額。客戶受管金鑰和會針對每個 API AWS KMS 呼叫 AWS 受管金鑰 產生費用，而配額適用於這些 KMS 金鑰。

資料表金鑰

DynamoDB 對資料表使用 KMS 金鑰來為資料表產生和加密唯一的資料金鑰，稱為資料表金鑰。資料表金鑰會在加密資料表的生命週期內持續存在。

資料表金鑰用作金鑰加密金鑰。DynamoDB 使用此資料表金鑰來保護用於加密資料表資料的資料加密金鑰。DynamoDB 會為資料表中的每個基礎結構產生唯一的資料加密金鑰，但多個資料表項目可能受到相同的資料加密金鑰保護。



當您第一次存取加密的資料表時，DynamoDB 會傳送請求給 AWS KMS ，以使用 KMS 金鑰來解密資料表金鑰。然後，它使用純文字資料表金鑰來解密資料加密金鑰，並使用純文字資料加密金鑰來解密資料表資料。

DynamoDB 會存放並使用 外部的資料表金鑰和資料加密金鑰 AWS KMS。它使用進階加密標準 (AES) 加密和 256 位元加密金鑰來保護所有金鑰。然後，它會將加密金鑰與加密資料一起存放，以便這些資料可用來隨需解密資料表。

如果您變更了資料表的 KMS 金鑰，DynamoDB 便會產生新的資料表金鑰。然後，使用新的資料表金鑰來重新加密資料加密金鑰。

資料表金鑰快取

為了避免 AWS KMS 針對每個 DynamoDB 操作呼叫 ，DynamoDB 會快取記憶體中每個呼叫者的純文字資料表金鑰。如果 DynamoDB 在閒置五分鐘後收到快取資料表金鑰的請求，則會將新的請求傳送至 AWS KMS 以解密資料表金鑰。此呼叫將擷取自上次請求解密資料表金鑰以來，對 或 AWS Identity and Access Management (IAM) 中 AWS KMS KMS 金鑰存取政策所做的任何變更。

授權使用您的 KMS 金鑰

如果您使用客戶自管金鑰或帳戶中的 AWS 受管金鑰 保護您的 DynamoDB 資料表，該 KMS 金鑰的政策必須允許 DynamoDB 代您使用該金鑰。 AWS 受管金鑰 適用於 DynamoDB 的 上的授權內容包含其金鑰政策和授予，以委派使用它的許可。

您可以完全控制客戶自管金鑰的政策和授予，因為 AWS 受管金鑰 在您的帳戶中，您可以檢視其政策和授予。但是，由於由 管理 AWS，您無法變更政策。

DynamoDB 不需要額外的授權，即可使用預設值AWS 擁有的金鑰來保護您 中的 DynamoDB 資料表 AWS 帳戶。

主題

- 的金鑰政策 AWS 受管金鑰
- 客戶自管金鑰的金鑰政策
- 使用授予來授權 DynamoDB

的金鑰政策 AWS 受管金鑰

當 DynamoDB 在密碼編譯操作中使用適用於 DynamoDB (aws/dynamodb) 的 AWS 受管金鑰 時，它是代表正在存取 DynamoDB 資源的使用者進行這項操作。上的金鑰政策 AWS 受管金鑰 提供帳戶中所有使用者使用 AWS 受管金鑰 進行指定操作的許可。但是，只有在 DynamoDB 代表使用者提出請求時才會授予許可。除非請求源自 DynamoDB 服務， AWS 受管金鑰 否則金鑰政策中的 ViaService 條件不允許任何使用者使用 。

此金鑰政策與所有政策一樣 AWS 受管金鑰，是由 建立 AWS。您不能改變它，但可以隨時檢視它。如需詳細資訊，請參閱檢視金鑰政策。

金鑰政策中的政策陳述式具有下列效果：

- 當請求來自 DynamoDB 時，允許帳戶中的使用者在密碼編譯操作中使用 AWS 受管金鑰 適用於 DynamoDB 的 。此政策也允許使用者為 KMS 金鑰建立授予。
- 允許帳戶中授權的 IAM 身分檢視 DynamoDB AWS 受管金鑰 的屬性，並撤銷允許 DynamoDB 使用 KMS 金鑰的授予。DynamoDB 使用授予進行持續的維護操作。
- 允許 DynamoDB 執行唯讀操作，在您的帳戶中尋找 AWS 受管金鑰 適用於 DynamoDB 的 。

JSON

```
{
  "Version":"2012-10-17",
  "Id" : "auto-dynamodb-1",
  "Statement" : [ {
    "Sid" : "Allow access through Amazon DynamoDB for all principals in the
 account that are authorized to use Amazon DynamoDB",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "*"
    },
    "Action" : [ "kms:Encrypt", "kms:Decrypt", "kms:ReEncrypt*",
 "kms:GenerateDataKey*", "kms:CreateGrant", "kms:DescribeKey" ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:CallerAccount" : "111122223333",
        "kms:ViaService" : "dynamodb.us-west-2.amazonaws.com"
      }
    }
  }, {
    "Sid" : "Allow direct access to key metadata to the account",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*", "kms:RevokeGrant" ],
    "Resource" : "*"
  }, {
    "Sid" : "Allow DynamoDB Service with service principal name
 dynamodb.amazonaws.com to describe the key directly",
    "Effect" : "Allow",
    "Principal" : {
      "Service" : "dynamodb.amazonaws.com"
```

```
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*" ],
    "Resource" : "*"
  } ]
}
```

客戶自管金鑰的金鑰政策

當您選取客戶自管金鑰來保護 DynamoDB 資料表時，DynamoDB 會取得許可，代替進行選取的主體使用 KMS 金鑰。該主體 (使用者或角色) 必須在 KMS 金鑰上擁有 DynamoDB 需要的許可。您可以在金鑰政策、IAM 政策或授予中提供這些許可。

至少，DynamoDB 在客戶自管金鑰上需要具備下列許可：

- kms:Encrypt
- kms:Decrypt
- kms:ReEncrypt* (適用於 kms:ReEncryptFrom and kms:ReEncryptTo)
- kms:GenerateDataKey* (適用於 kms:GenerateDataKey 和 kms:GenerateDataKeyWithoutPlaintext)
- kms:DescribeKey
- kms:CreateGrant

例如，以下範例金鑰政策只會提供必要許可。政策具有下列效果：

- 允許 DynamoDB 在密碼編譯操作中使用 KMS 金鑰並建立授予，但只有在其代替具備使用 DynamoDB 許可帳戶中的主體時才能進行。如果政策陳述式中指定的主體沒有使用 DynamoDB 的許可，呼叫便會失敗，即使呼叫是來自 DynamoDB 服務也一樣。
- kms:ViaService 條件索引鍵只會在請求來自 DynamoDB，且其是代替政策陳述式中列出的主體時，才會允許許可。這些主體無法直接呼叫這些操作。請注意，`kms:ViaService` 值 (`dynamodb.*.amazonaws.com`) 在區域位置中有星號 (*)。DynamoDB 需要獨立於任何特定 的許可， AWS 區域 才能進行跨區域呼叫以支援 DynamoDB 全域資料表。
- 給予 KMS 金鑰管理員 (可取得 `db-team` 角色的使用者) 對 KMS 金鑰的唯讀存取權以及撤銷授予的許可，包括 DynamoDB 需要用來保護資料表的授予。

使用範例金鑰政策之前，請將範例主體取代為來自您的實際主體 AWS 帳戶。

JSON

```
{
  "Id": "key-policy-dynamodb",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid" : "Allow access through Amazon DynamoDB for all principals in the
 account that are authorized to use Amazon DynamoDB",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:user/db-lead"},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
         "StringLike": {
           "kms:ViaService" : "dynamodb.*.amazonaws.com"
         }
      }
    },
    {
      "Sid":  "Allow administrators to view the KMS key and revoke grants",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/db-team"
       },
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
      ],
      "Resource": "*"
    }
  ]
}
```

使用授予來授權 DynamoDB

除了金鑰政策外，DynamoDB 會使用授予在客戶自管金鑰或 AWS 受管金鑰 for DynamoDB (aws/dynamodb) 上設定許可。若要檢視您帳戶中 KMS 金鑰的授予，請使用 ListGrants 操作。DynamoDB 不需要授予或任何其他許可，即可使用 AWS 擁有的金鑰 來保護您的資料表。

DynamoDB 在執行背景系統維護和持續資料保護任務時使用授予許可。它還使用授與來產生資料表金鑰。

每個授與都專屬於一個資料表。如果帳戶包含使用相同 KMS 金鑰加密的多個資料表，則每個資料表有每一種類型的授予。授予受到 DynamoDB 加密內容的限制，其中包含資料表名稱和 AWS 帳戶 ID，並包含不再需要的淘汰授予的許可。

如要建立授予，DynamoDB 必須擁有代替建立加密資料表使用者呼叫 `CreateGrant` 的許可。對於 AWS 受管金鑰，DynamoDB 會從金鑰政策取得`kms:CreateGrant`許可，這允許帳戶使用者只有在 DynamoDB 代表授權使用者提出請求時，才對 KMS 金鑰呼叫 CreateGrant。

金鑰政策也會允許帳戶撤銷 KMS 金鑰上的授予。不過，如果您撤銷作用中加密資料表的授予，DynamoDB 將無法保護和維護資料表。

DynamoDB 加密內容

加密內容是一組金鑰/值對，其中包含任意非私密資料。當您在加密資料的請求中包含加密內容時，會以 AWS KMS 加密方式將加密內容繫結至加密的資料。若要解密資料，您必須傳遞相同的加密內容。

DynamoDB 在所有 AWS KMS 密碼編譯操作中使用相同的加密內容。如果使用客戶自管金鑰或 AWS 受管金鑰 來保護您的 DynamoDB 資料表，您可以使用加密內容來識別在稽核記錄和日誌中使用 KMS 金鑰的情況。它還會在日誌中以純文字顯示，例如 AWS CloudTrail 和 Amazon CloudWatch Logs。

加密內容也可以用作政策和授予中的授權條件。DynamoDB 使用加密內容來限制允許存取客戶受管金鑰或您 AWS 受管金鑰 帳戶和區域中的授予。

在其對 的請求中 AWS KMS，DynamoDB 使用具有兩個鍵/值對的加密內容。

```
"encryptionContextSubset": {
    "aws:dynamodb:tableName": "Books"
    "aws:dynamodb:subscriberId": "111122223333"
}
```

- 資料表 – 第一個鍵值對會識別 DynamoDB 正在加密的資料表。金鑰為 `aws:dynamodb:tableName`。值是資料表的名稱。

```
"aws:dynamodb:tableName": "<table-name>"
```

例如：

```
"aws:dynamodb:tableName": "Books"
```

- 帳戶 – 第二個鍵值對會識別 AWS 帳戶。金鑰為 `aws:dynamodb:subscriberId`。值是帳戶 ID。

```
"aws:dynamodb:subscriberId": "<account-id>"
```

例如：

```
"aws:dynamodb:subscriberId": "111122223333"
```

監控 DynamoDB 與 的互動 AWS KMS

如果您使用客戶受管金鑰或 AWS 受管金鑰來保護 DynamoDB 資料表，您可以使用 AWS CloudTrail 日誌來追蹤 DynamoDB AWS KMS 代表您傳送到 的請求。

本節將討論 `GenerateDataKey`、`Decrypt` 和 `CreateGrant` 請求。此外，DynamoDB 會使用 DescribeKey 操作來判斷您選取的 KMS 金鑰是否存在於帳戶和區域內。當您刪除資料表時，它也使用 RetireGrant 操作來移除授與。

GenerateDataKey

當您在資料表上啟用靜態加密時，DynamoDB 會建立唯一的資料表金鑰。它會將 GenerateDataKey 請求傳送至 AWS KMS ，以指定資料表的 KMS 金鑰。

記錄 `GenerateDataKey` 操作的事件類似於以下範例事件。使用者是 DynamoDB 服務帳戶。參數包括秘密 KMS 金鑰的 Amazon Resource Name (ARN)、需要 256 位元金鑰的金鑰規範，以及識別資料表和 AWS 帳戶的加密內容。

```
{
    "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"
}
```

解密

當您存取加密的 DynamoDB 資料表時，DynamoDB 需要解密資料表金鑰以便解密階層下方的金鑰。然後解密資料表中的資料。解密資料表金鑰。DynamoDB 會將解密請求傳送至 AWS KMS ，以指定資料表的 KMS 金鑰。

記錄 `Decrypt` 操作的事件類似於以下範例事件。使用者是您存取資料表的委託 AWS 帳戶 人。這些參數包括加密的資料表金鑰 （ 做為加密文字 Blob)，以及識別資料表和 的加密內容 AWS 帳戶。會從加密文字 AWS KMS 衍生 KMS 金鑰的 ID。

```
{
    "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

當您使用客戶自管金鑰或 AWS 受管金鑰 保護您的 DynamoDB 資料表時，DynamoDB 會使用授權，以允許服務執行持續資料保護和維護及耐用性任務。AWS 擁有的金鑰 上不需要這些授予。

DynamoDB 建立的授予專屬於特定資料表。CreateGrant 請求中的主體是建立資料表的使用者。

記錄 `CreateGrant` 操作的事件類似於以下範例事件。參數包括資料表的 KMS 金鑰的 Amazon Resource Name (ARN)、承授者主體和淘汰主體 (DynamoDB 服務)，以及授予涵蓋的操作。它還包含一個限制，要求所有加密操作使用指定的加密內容。

```
{
    "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"
}
```

在 DynamoDB 中管理加密資料表

您可以使用 AWS 管理主控台 或 AWS Command Line Interface (AWS CLI) 在新資料表上指定加密金鑰，並更新 Amazon DynamoDB 中現有資料表上的加密金鑰。

主題

- 指定新資料表的加密金鑰
- 更新加密金鑰

指定新資料表的加密金鑰

請依照這些步驟，使用 Amazon DynamoDB 主控台或 AWS CLI在新資料表上指定加密金鑰。

建立加密資料表 (主控台)

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇 Create Table (建立資料表)。針對資料表名稱，輸入 **`Music`**。針對主索引鍵，輸入 **`Artist`**，對於排序索引鍵，輸入 **`SongTitle`**，兩者都做為字串。
4. 在 Settings (設定) 中，確認選取 Customize settings (自訂設定)。

   > **Note**
   >
   > 如果選取使用預設設定，則會使用 AWS 擁有的金鑰 靜態加密資料表，無需額外費用。

5. 在靜態加密下，選擇加密類型 - AWS 擁有的金鑰 AWS 受管金鑰、 或客戶受管金鑰。
   - 由 Amazon DynamoDB AWS 擁有的金鑰，由 DynamoDB 特別擁有和管理。使用此金鑰不會向您收取額外費用。
   - AWS 受管金鑰。金鑰別名：`aws/dynamodb`。金鑰會存放在您的帳戶中，並由 AWS Key Management Service (AWS KMS) 管理。需支付 AWS KMS 費用。
   - 儲存在您的帳戶中，且由您擁有和管理。客戶自管金鑰。金鑰會存放在您的帳戶中，並由 AWS Key Management Service (AWS KMS) 管理。需支付 AWS KMS 費用。

> **Note**
>
> 如果您選擇擁有和管理自己的金鑰，請確保已正確設定 KMS 金鑰政策。如需詳細資訊及範例，請參閱客戶自管金鑰的金鑰政策。

6. 選擇 Create table (建立資料表) 以建立加密資料表。若要確認加密類型，選取 Overview (概觀) 標籤中的資料表詳細資訊並檢閱 Additional details (其他詳細資訊) 區段。

建立加密資料表 (AWS CLI)

使用 AWS CLI 為 Amazon DynamoDB 建立具有預設 AWS 擁有的金鑰、 AWS 受管金鑰或客戶受管金鑰的資料表。

使用預設值建立加密的資料表 AWS 擁有的金鑰

- 建立加密的 `Music` 資料表，如下所示。

```
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**
>
> 此資料表現在使用 AWS 擁有的金鑰 DynamoDB 服務帳戶中的預設值加密。

使用 AWS 受管金鑰 適用於 DynamoDB 的 建立加密資料表

- 建立加密的 `Music` 資料表，如下所示。

```
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
```

資料表描述的 SSEDescription 狀態會設為 ENABLED，而 SSEType 設為 KMS。

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

使用 DynamoDB 由客戶自管金鑰建立加密資料表

- 建立加密的 Music 資料表，如下所示。

```
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
```

> **Note**
>
> 對於 KMSMasterKeyId，您可以使用金鑰 ID、金鑰 ARN 或金鑰別名。如果您使用金鑰別名（例如 alias/my-key)，DynamoDB 會解析別名，並將基礎 AWS KMS 金鑰

與資料表建立關聯。在資料表描述中， KMSMasterKeyArn一律會顯示已解析金鑰的金鑰 ARN，而不是別名。如需金鑰識別符的詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的金鑰識別符 (KeyId)。

資料表描述的 SSEDescription 狀態會設為 ENABLED，而 SSEType 設為 KMS。

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

更新加密金鑰

您也可以使用 DynamoDB 主控台或 AWS CLI ，隨時更新 AWS 擁有的金鑰 AWS 受管金鑰和客戶受管金鑰之間現有資料表的加密金鑰。

更新加密金鑰 (主控台)

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇您要更新的資料表。
4. 選取 Actions (動作) 下拉式清單，然後選取 Update settings (更新設定) 選項。
5. 轉至 Additional settings (其他設定) 索引標籤。
6. 在 Encryption (加密) 下，選擇 Manage encryption (管理加密)。
7. 選擇加密類型：

   - 由 Amazon DynamoDB 擁有。 AWS KMS 金鑰由 DynamoDB 擁有和管理。使用此金鑰不會向您收取額外費用。
   - AWS 受管金鑰金鑰別名：aws/dynamodb。金鑰會存放在您的帳戶中，並由 管理 AWS Key Management Service。 (AWS KMS)。需支付 AWS KMS 費用。
   - 儲存在您的帳戶中，且由您擁有和管理。金鑰會存放在您的帳戶中，並由 管理 AWS Key Management Service。 (AWS KMS)。需支付 AWS KMS 費用。

> **Note**
>
> 如果您選擇擁有和管理自己的金鑰，請確保已正確設定 KMS 金鑰政策。如需詳細資訊，請參閱客戶自管金鑰的金鑰政策。

然後，選擇 Save (儲存) 以更新加密資料表。若要確認加密類型，檢查 Overview (概觀) 索引標籤下的資料表詳細資訊。

更新加密金鑰 (AWS CLI)

下列範例顯示如何使用 AWS CLI更新加密資料表。

使用預設值更新加密的資料表 AWS 擁有的金鑰

- 更新加密的 `Music` 資料表，如下列範例所示。

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=false
```

> **Note**
>
> 此資料表現在使用 AWS 擁有的金鑰 DynamoDB 服務帳戶中的預設值加密。

使用 AWS 受管金鑰 適用於 DynamoDB 的 更新加密的資料表

- 更新加密的 `Music` 資料表，如下列範例所示。

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true
```

資料表描述的 `SSEDescription` 狀態會設為 `ENABLED`，而 `SSEType` 設為 `KMS`。

```
"SSEDescription": {
  "SSEType": "KMS",
```

```
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

使用 DynamoDB 由客戶自管金鑰更新加密資料表

- 更新加密的 Music 資料表，如下列範例所示。

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

> **Note**
>
> 對於 KMSMasterKeyId，您可以使用金鑰 ID、金鑰 ARN 或金鑰別名。如果您使用金鑰別名（例如 alias/my-key)，DynamoDB 會解析別名，並將基礎 AWS KMS 金鑰與資料表建立關聯。在資料表描述中， KMSMasterKeyArn一律會顯示已解析金鑰的金鑰 ARN，而不是別名。

資料表描述的 SSEDescription 狀態會設為 ENABLED，而 SSEType 設為 KMS。

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

使用 VPC 端點與 IAM 政策強化 DynamoDB 連線安全性

Amazon DynamoDB 和內部部署應用程式之間，以及 DynamoDB 和相同 AWS 區域內其他 AWS 資源之間的連線都會受到保護。

端點所需的政策

Amazon DynamoDB 提供 DescribeEndpoints API，可讓您列舉區域端點資訊。對於公用 DynamoDB 端點的請求，即使 IAM 或 VPC 端點政策中設定了明確或隱含的拒絕條件，API 仍會回應，而不受 DynamoDB IAM 政策影響。這是因為 DynamoDB 會刻意略過 `DescribeEndpoints` API 的授權程序。

對於來自 VPC 端點的請求，IAM 和虛擬私有雲端 (VPC) 端點政策都必須使用 IAM `dynamodb:DescribeEndpoints` 動作針對請求身分和存取管理 (IAM) 主體授權 `DescribeEndpoints` API 呼叫。否則，將拒絕存取 `DescribeEndpoints` API。

以下是端點政策的範例。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "dynamodb:DescribeEndpoints",
            "Resource": "*"
        }
    ]
}
```

服務和內部部署用戶端與應用程式之間的流量。

您的私有網路與 之間有兩個連線選項 AWS：

- AWS Site-to-Site VPN 連線。如需詳細資訊，請參閱《 AWS Site-to-Site VPN使用者指南》中的什麼是AWS Site-to-Site VPN ？。
- Direct Connect 連線。如需詳細資訊，請參閱《 Direct Connect使用者指南》中的什麼是Direct Connect ？。

透過網路存取 DynamoDB 是透過 AWS 發佈APIs。用戶端必須支援 Transport Layer Security (TLS) 1.2。我們建議使用 TLS 1.3。用戶端還必須支援具備完全正向加密 (PFS) 功能的密碼套件，例如臨時 Diffie-Hellman (DHE) 或橢圓曲線臨時 Diffie-Hellman (ECDHE)。現代系統 (如 Java 7 和更新版本) 大多會支援這些模式。此外，您必須使用存取金鑰 ID，以及與 IAM 主體相關聯的私密存取金鑰來簽署請求，或者您可以使用 AWS Security Token Service (STS) 來產生臨時安全憑證來簽署請求。

相同區域中 AWS 資源之間的流量

適用於 DynamoDB 的 Amazon Virtual Private Cloud (Amazon VPC) 端點是 VPC 中的邏輯實體，僅允許連線到 DynamoDB。Amazon VPC 會將請求路由至 DynamoDB，並將回應路由回 VPC。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 VPC 端點。如需可從 VPC 端點控制存取權限的政策範例，請參閱使用 IAM 政策控制對 DynamoDB 的存取。

> **Note**
>
> Amazon VPC 端點無法透過 AWS Site-to-Site VPN 或 存取 Direct Connect。

AWS Identity and Access Management (IAM) 和 DynamoDB

AWS Identity and Access Management 是一種 AWS 服務，可協助管理員安全地控制對 AWS 資源的存取。管理員可以控制透過驗證 (已登入) 和授權 (具有許可) 來使用 Amazon DynamoDB 和 DynamoDB Accelerator 資源的對象。您可以使用 IAM 來管理存取許可，並為 Amazon DynamoDB 和 DynamoDB Accelerator 實作安全政策。IAM 是一項服務 AWS ，您可以免費使用。

主題

- Amazon DynamoDB 的 Identity and Access Management
- 使用 IAM 政策條件進行精細定義存取控制

Amazon DynamoDB 的 Identity and Access Management

AWS Identity and Access Management (IAM) 是一種 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可以控制誰能「完成身分驗證」(已登入) 和「獲得授權」(具有許可) 而得以使用 DynamoDB 資源。IAM 是您可以免費使用 AWS 服務 的 。

主題

- 目標對象
- 使用身分驗證
- 使用政策管理存取權
- Amazon DynamoDB 如何搭配 IAM 運作
- Amazon DynamoDB 以身分為基礎的政策範例
- Amazon DynamoDB 身分識別和存取疑難排解
- 避免購買 DynamoDB 預留容量的 IAM 政策

目標對象

使用方式 AWS Identity and Access Management (IAM) 會根據您的角色而有所不同：

- 服務使用者 — 若無法存取某些功能，請向管理員申請所需許可 (請參閱 Amazon DynamoDB 身分識別和存取疑難排解)
- 服務管理員 — 負責設定使用者存取權並提交相關許可請求 (請參閱 Amazon DynamoDB 如何搭配 IAM 運作)
- IAM 管理員 — 撰寫政策以管理存取控制 (請參閱 Amazon DynamoDB 以身分為基礎的政策範例)

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者或擔任 IAM 角色進行身分驗證。

您可以使用身分來源的登入資料，例如 AWS IAM Identity Center (IAM Identity Center)、單一登入身分驗證或 Google/Facebook 登入資料，以聯合身分的形式登入。如需有關登入的詳細資訊，請參閱《AWS 登入 使用者指南》中的如何登入您的 AWS 帳戶。

對於程式設計存取， AWS 提供 SDK 和 CLI 以密碼編譯方式簽署請求。如需詳細資訊，請參閱《IAM 使用者指南》中的 API 請求的AWS 第 4 版簽署程序。

AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個名為 AWS 帳戶 theroot 使用者的登入身分開始，該身分可完整存取所有 AWS 服務 和 資源。強烈建議不要使用根使用者來執行日常任務。有關需要根使用者憑證的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

聯合身分

最佳實務是要求人類使用者使用聯合身分提供者，以 AWS 服務 使用臨時憑證存取 。

聯合身分是來自您的企業目錄、Web 身分提供者的使用者，或使用來自身分來源的 AWS 服務 憑證存取 Directory Service 的使用者。聯合身分會擔任角色，而該角色會提供臨時憑證。

若需集中化管理存取權限，建議使用 AWS IAM Identity Center。如需詳細資訊，請參閱 AWS IAM Identity Center 使用者指南中的什麼是 IAM Identity Center？。

IAM 使用者和群組

IAM 使用者https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html是一種身分具備單人或應用程式的特定許可權。建議以臨時憑證取代具備長期憑證的 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的要求人類使用者使用聯合身分提供者來 AWS 使用臨時憑證存取 。

IAM 群組會指定 IAM 使用者集合，使管理大量使用者的許可權更加輕鬆。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

IAM 角色

IAM 角色https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html的身分具有特定許可權，其可以提供臨時憑證。您可以透過從使用者切換到 IAM 角色（主控台）或呼叫 AWS CLI 或 AWS API 操作來擔任角色。如需詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

IAM 角色適用於聯合身分使用者存取、臨時 IAM 使用者許可、跨帳戶存取權與跨服務存取，以及在 Amazon EC2 執行的應用程式。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策定義與身分或資源相關聯的許可。當委託人提出請求時 AWS ， 會評估這些政策。大多數政策會以 JSON 文件 AWS 形式存放在 中。如需進一步了解 JSON 政策文件，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員會使用政策，透過定義哪些主體可在哪些條件下對哪些資源執行動作，以指定可存取的範圍。

預設情況下，使用者和角色沒有許可。IAM 管理員會建立 IAM 政策並將其新增至角色，供使用者後續擔任。IAM 政策定義動作的許可，無論採用何種方式執行。

身分型政策

身分型政策是附加至身分 (使用者、使用者群組或角色) 的 JSON 許可政策文件。這類政策控制身分可對哪些資源執行哪些動作，以及適用的條件。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

身分型政策可分為內嵌政策 (直接內嵌於單一身分) 與受管政策 (可附加至多個身分的獨立政策)。如需了解如何在受管政策及內嵌政策之間做選擇，請參閱《IAM 使用者指南》中的在受管政策與內嵌政策之間選擇。

資源型政策

資源型政策是附加到資源的 JSON 政策文件。範例包括 IAM 角色信任政策與 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。您必須在資源型政策中指定主體。

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用來自 IAM 的 AWS 受管政策。

其他政策類型

AWS 支援其他政策類型，可設定更多常見政策類型授予的最大許可：

- 許可界限 — 設定身分型政策可授與 IAM 實體的最大許可。如需詳細資訊，請參閱《 IAM 使用者指南》中的 IAM 實體許可界限。
- 服務控制政策 (SCP) — 為 AWS Organizations中的組織或組織單位指定最大許可。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的服務控制政策。
- 資源控制政策 (RCP) — 設定您帳戶中資源可用許可的上限。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的資源控制政策 (RCP)。
- 工作階段政策 — 在以程式設計方式為角色或聯合身分使用者建立臨時工作階段時，以參數形式傳遞的進階政策。如需詳細資訊，請參《IAM 使用者指南》中的工作階段政策。

多種政策類型

當多種類型的政策適用於請求時，產生的許可會更複雜而無法理解。若要了解如何 AWS 在涉及多個政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

Amazon DynamoDB 如何搭配 IAM 運作

在您使用 IAM 管理 DynamoDB 的存取權之前，請了解搭配 DynamoDB 使用的 IAM 功能有哪些。

IAM 功能	DynamoDB 支援
身分型政策	是
資源型政策	是
政策動作	是
政策資源	是
政策條件索引鍵	是
ACL	否
ABAC (政策中的標籤)	是
臨時憑證	是
主體許可	是
服務角色	是
服務連結角色	是

若要全面了解 DynamoDB 和其他 AWS 服務如何與大多數 IAM 功能搭配使用，請參閱《AWS IAM 使用者指南》中的與 IAM 搭配使用的 服務。

適用於 DynamoDB 的以身分為基礎的政策

支援身分型政策：是

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。如要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素參考。

DynamoDB 以身分為基礎的政策範例

若要檢視 DynamoDB 以身分為基礎的政策範例，請參閱 Amazon DynamoDB 以身分為基礎的政策範例。

DynamoDB 內以資源為基礎的政策

支援資源型政策：是

資源型政策是附加到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

如需啟用跨帳戶存取權，您可以在其他帳戶內指定所有帳戶或 IAM 實體作為資源型政策的主體。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

DynamoDB 的政策動作

支援政策動作：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。政策會使用動作來授予執行相關聯動作的許可。

若要查看 DynamoDB 動作的清單，請參閱《服務授權參考》中的 Amazon DynamoDB 定義的動作。

DynamoDB 中的政策動作會在動作之前使用以下字首：

```
aws
```

若要在單一陳述式中指定多個動作，請用逗號分隔。

```
"Action": [
      "aws:action1",
      "aws:action2"
         ]
```

若要檢視 DynamoDB 以身分為基礎的政策範例，請參閱 Amazon DynamoDB 以身分為基礎的政策範例。

DynamoDB 的政策資源

支援政策資源：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

`Resource` JSON 政策元素可指定要套用動作的物件。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。若動作不支援資源層級許可，使用萬用字元 (*) 表示該陳述式適用於所有資源。

```
"Resource": "*"
```

若要查看 DynamoDB 資源類型清單及其 ARN，請參閱《服務授權參考》中的 Amazon DynamoDB 定義的資源。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱 Amazon DynamoDB 定義的動作。

若要檢視 DynamoDB 以身分為基礎的政策範例，請參閱 Amazon DynamoDB 以身分為基礎的政策範例。

DynamoDB 的政策條件索引鍵

支援服務特定政策條件金鑰：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

`Condition` 元素會根據定義的條件，指定陳述式的執行時機。您可以建立使用條件運算子的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的AWS 全域條件內容索引鍵。

如要查看 DynamoDB 條件索引鍵的清單，請參閱《服務授權參考》中的 Amazon DynamoDB 的條件索引鍵。若要了解您可以搭配哪些動作和資源使用條件索引鍵，請參閱 Amazon DynamoDB 定義的動作。

若要檢視 DynamoDB 以身分為基礎的政策範例，請參閱 Amazon DynamoDB 以身分為基礎的政策範例。

DynamoDB 中的存取控制清單 (ACL)

支援 ACL：否

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

搭配 DynamoDB 的屬性型存取控制 (ABAC)

支援 ABAC (政策中的標籤)：是

屬性型存取控制 (ABAC) 是一種授權策略，依據稱為標籤的屬性來定義許可。您可以將標籤連接至 IAM 實體 AWS 和資源，然後設計 ABAC 政策，以便在主體的標籤符合資源上的標籤時允許操作。

如需根據標籤控制存取，請使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 條件索引鍵，在政策的條件元素中，提供標籤資訊。

如果服務支援每個資源類型的全部三個條件金鑰，則對該服務而言，值為 Yes。如果服務僅支援某些資源類型的全部三個條件金鑰，則值為 Partial。

如需 ABAC 的詳細資訊，請參閱《IAM 使用者指南》中的使用 ABAC 授權定義許可。如要查看含有設定 ABAC 步驟的教學課程，請參閱《IAM 使用者指南》中的使用屬性型存取控制 (ABAC)。

將臨時憑證與 DynamoDB 搭配使用

支援臨時憑證：是

臨時登入資料提供 AWS 資源的短期存取權，當您使用聯合或切換角色時，會自動建立。 AWS 建議您動態產生臨時登入資料，而不是使用長期存取金鑰。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的臨時安全憑證與可與 IAM 搭配運作的AWS 服務。

DynamoDB 的跨服務主體許可

支援轉寄存取工作階段 (FAS)：是

轉送存取工作階段 (FAS) 使用呼叫 的委託人許可 AWS 服務，並結合 AWS 服務 請求向下游服務提出請求。如需提出 FAS 請求時的政策詳細資訊，請參閱轉發存取工作階段。

DynamoDB 的服務角色

支援服務角色：是

服務角色是服務擔任的 IAM 角色，可代您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可權給 AWS 服務。

> **Warning**
>
> 變更服務角色的許可有可能會讓 DynamoDB 功能出現故障。只有 DynamoDB 提供指引時，才能編輯服務角色。

DynamoDB 的服務連結角色

支援服務連結角色：是

服務連結角色是連結至 的一種服務角色 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的 中 AWS 帳戶 ，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

如需建立或管理服務連結角色的詳細資訊，請參閱可搭配 IAM 運作的AWS 服務。在資料表中尋找服務，其中包含服務連結角色欄中的 Yes。選擇是連結，以檢視該服務的服務連結角色文件。

DynamoDB 支援的服務連結角色

DynamoDB 支援下列服務連結角色。

- DynamoDB 使用服務連結角色 AWSServiceRoleForDynamoDBReplication 來執行跨 AWS 區域的全域資料表複寫。如需 AWSServiceRoleForDynamoDBReplication 服務連結角色的詳細資訊，請參閱 DynamoDB 全域資料表安全性。
- DynamoDB Accelerator (DAX) 使用服務連結角色 AWSServiceRoleForDAX 來設定與維護 DAX 叢集。如需 AWSServiceRoleForDAX 服務連結角色的詳細資訊，請參閱 the section called “使用 DAX 的服務連結角色”。

除了這些 DynamoDB 服務連結角色外，DynamoDB 也會使用 Application Auto Scaling 服務自動管理佈建容量模式資料表的輸送量設定。Application Auto Scaling 服務會使用服務連結角色 AWSServiceRoleForApplicationAutoScaling_DynamoDBTable，以管理已啟用自動調整的 DynamoDB 資料表之輸送量設定。如需更多詳細資訊，請參閱 Application Auto Scaling 的服務連結角色。

Amazon DynamoDB 以身分為基礎的政策範例

根據預設，使用者和角色不具備建立或修改 DynamoDB 資源的許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。

如需了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策 (主控台)。

如需 DynamoDB 所定義之動作和資源類型的詳細資訊，包括每種資源類型的 ARN 格式，請參閱《服務授權參考》中 Amazon DynamoDB 適用的動作、資源和條件索引鍵。

主題

- 政策最佳實務
- 使用 DynamoDB 主控台
- 允許使用者檢視他們自己的許可
- 搭配 Amazon DynamoDB 使用身分型政策

政策最佳實務

以身分為基礎的政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除 DynamoDB 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱《IAM 使用者指南》中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 等使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 CloudFormation。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM Access Analyzer 驗證政策。
- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的透過 MFA 的安全 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱《IAM 使用者指南》中的 IAM 安全最佳實務。

使用 DynamoDB 主控台

若要存取 Amazon DynamoDB 主控台，您必須擁有最基本的一組許可。這些許可必須允許您列出和檢視 中 DynamoDB 資源的詳細資訊 AWS 帳戶。如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (使用者或角色) 而言，主控台就無法如預期運作。

對於僅呼叫 AWS CLI 或 AWS API 的使用者，您不需要允許最低主控台許可。反之，只需允許存取符合他們嘗試執行之 API 操作的動作就可以了。

為了確保使用者和角色仍然可以使用 DynamoDB 主控台，請將 DynamoDB `ConsoleAccess`或 `ReadOnly` AWS 受管政策連接到實體。如需詳細資訊，請參閱《IAM 使用者指南》中的新增許可到使用者。

允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視附加到他們使用者身分的內嵌及受管政策。此政策包含在主控台或使用 或 AWS CLI AWS API 以程式設計方式完成此動作的許可。

```
{
    "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": "*"
        }
    ]
}
```

搭配 Amazon DynamoDB 使用身分型政策

本主題涵蓋搭配 Amazon DynamoDB 使用身分型 AWS Identity and Access Management (IAM) 政策，並提供範例。這些範例會示範帳戶管理員如何將許可政策連接至 IAM 身分 (使用者、群組和角色)，並藉此授與許可，以取得在 Amazon DynamoDB 資源上執行操作的許可。

本主題中的各節涵蓋下列內容：

- 使用 Amazon DynamoDB 主控台所需的 IAM 許可
- AWS Amazon DynamoDB 的受管（預先定義）IAM 政策
- 客戶管理政策範例

以下是許可政策的範例。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Books"
        }
```

```
    ]
}
```

上述政策有一個陳述式，可授予 us-west-2 AWS 區域中資料表上三個 DynamoDB 動作 (dynamodb:DescribeTable、 dynamodb:Query和 dynamodb:Scan) 的許可，該動作由 指定的 AWS 帳戶所擁有*account-id*。Resource 值中的 Amazon Resource Name (ARN) 會指定許可適用的資料表。

使用 Amazon DynamoDB 主控台所需的 IAM 許可

若要使用 DynamoDB 主控台，使用者必須擁有一組最低許可，允許使用者使用其 AWS 帳戶的 DynamoDB 資源。除了這些 DynamoDB 許可之外，主控台還需要以下許可：

- 顯示指標和圖形的 Amazon CloudWatch 許可。
- AWS Data Pipeline 匯出和匯入 DynamoDB 資料的許可。
- AWS Identity and Access Management 存取匯出和匯入所需角色的許可。
- 只要觸發 CloudWatch 警示就通知您的 Amazon Simple Notification Service 許可。
- AWS Lambda 處理 DynamoDB Streams 記錄的許可。

如果您建立比最基本必要許可更嚴格的 IAM 政策，則對於採取該 IAM 政策的使用者而言，主控台就無法如預期運作。為了確保這些使用者仍然可以使用 DynamoDB 主控台，也請將 AmazonDynamoDBReadOnlyAccess AWS 受管政策連接至使用者，如中所述AWS Amazon DynamoDB 的受管（預先定義）IAM 政策。

對於僅呼叫 AWS CLI 或 Amazon DynamoDB API 的使用者，您不需要允許最低主控台許可。

> **Note**
>
> 若您參考 VPC 端點，您也需使用 IAM 動作 (dynamodb:DescribeEndpoints)，對請求的 IAM 主體授權 DescribeEndpoints API 呼叫。如需更多資訊，請參閱端點所需的政策。

AWS Amazon DynamoDB 的受管（預先定義）IAM 政策

AWS 提供由 建立和管理的獨立 IAM 政策，以解決一些常見的使用案例 AWS。這些 AWS 受管政策會授予常見使用案例的必要許可，讓您不必調查需要哪些許可。如需詳細資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策。

下列 AWS 受管政策可連接至您帳戶中的使用者，其專屬於 DynamoDB，並依使用案例案例分組：

- AmazonDynamoDBReadOnlyAccess – 透過 授予 DynamoDB 資源的唯讀存取權 AWS 管理主控台。
- AmazonDynamoDBFullAccess – 透過 授予 DynamoDB 資源的完整存取權 AWS 管理主控台。

您可以登入 IAM 主控台並在其中搜尋特定政策，以檢閱這些 AWS 受管許可政策。

> **Important**
>
> 最佳實務是建立自訂 IAM 政策，授予最低權限給需要的使用者、角色或群組。

客戶管理政策範例

在本節中，您可以找到授予各種 DynamoDB 動作許可之政策範例。當您使用 AWS SDKs或 時，這些政策即可運作 AWS CLI。在您使用主控台時，需要授予特定的其他許可給主控台。如需詳細資訊，請參閱使用 Amazon DynamoDB 主控台所需的 IAM 許可。

> **Note**
>
> 下列所有政策範例都使用其中一個 AWS 區域，並包含虛構的帳戶 IDs 和資料表名稱。

範例：

- 將許可授與資料表上所有 DynamoDB 動作的 IAM 政策
- 用來授予 DynamoDB 資料表項目唯讀許可的 IAM 政策
- 授予特定 DynamoDB 資料表及其索引存取權限的 IAM 政策
- 讀取、寫入、更新和刪除 DynamoDB 資料表存取權限的 IAM 政策
- 在相同 AWS 帳戶中分隔 DynamoDB 環境的 IAM 政策
- 避免購買 DynamoDB 預留容量的 IAM 政策
- 僅授予 DynamoDB 串流讀取存取權限的 IAM 政策 (不適用於資料表)
- 允許 AWS Lambda 函數存取 DynamoDB 串流記錄的 IAM 政策
- 允許對 DynamoDB Accelerator (DAX) 叢集進行讀取和寫入的 IAM 政策

《IAM 使用者指南》包含三個其他 DynamoDB 範例：

- Amazon DynamoDB：允許存取特定資料表
- Amazon DynamoDB：允許存取特定資料欄
- Amazon DynamoDB：允許根據 Amazon Cognito ID 對 DynamoDB 進行資料列層級存取

將許可授與資料表上所有 DynamoDB 動作的 IAM 政策

下列政策會對資料表上名為 Books 的所有 DynamoDB 動作授予許可。中指定的資源 ARN Resource可識別特定 AWS 區域中的資料表。如果以萬用字元 (*) 代替 Resource ARN 中的資料表名稱 Books，則允許對帳戶中的所有資料表執行所有 DynamoDB 動作。在此或任何 IAM 政策上使用萬用字元之前，請仔細考慮可能發生的安全性影響。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": "dynamodb:*",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

> **Note**
>
> 這是使用萬用字元 (*) 允許所有動作的範例，包括管理、資料操作、監控和購買 DynamoDB 預留容量。反之，最佳實務是明確指定要授予的每個動作，以及只指定該使用者、角色或群組所需的動作。

用來授予 DynamoDB 資料表項目唯讀許可的 IAM 政策

下列許可政策只會授予 GetItem、BatchGetItem、Scan、Query 和 ConditionCheckItem DynamoDB 動作的許可，進而在 Books 資料表上設定唯讀存取權。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

授予特定 DynamoDB 資料表及其索引存取權限的 IAM 政策

下列政策會針對名為 Books 的 DynamoDB 資料表上的資料修改動作和該表所有的索引授予許可。如需索引運作方式的詳細資訊，請參閱 在 DynamoDB 中使用次要索引來改善資料存取。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AccessTableAllIndexesOnBooks",
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
              "dynamodb:BatchWriteItem",
              "dynamodb:GetItem",
              "dynamodb:BatchGetItem",
```

```
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

讀取、寫入、更新和刪除 DynamoDB 資料表存取權限的 IAM 政策

如果您需要允許應用程式建立、讀取、更新和刪除 Amazon DynamoDB 資料表、索引和串流中的資料，請使用此政策。視需要替換 AWS 區域名稱、您的帳戶 ID，以及資料表名稱或萬用字元 (*)。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBIndexAndStreamAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetShardIterator",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/stream/*"
            ]
        },
        {
            "Sid": "DynamoDBTableAccess",
            "Effect": "Allow",
            "Action": [
```

```
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:DescribeTable",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        },
        {
            "Sid": "DynamoDBDescribeLimitsAccess",
            "Effect": "Allow",
            "Action": "dynamodb:DescribeLimits",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

若要展開此政策以涵蓋此帳戶所有 AWS 區域中的所有 DynamoDB 資料表，請針對區域和資料表名稱使用萬用字元 (*)。例如：

```
"Resource":[
               "arn:aws:dynamodb:*:123456789012:table/*",
               "arn:aws:dynamodb:*:123456789012:table/*/index/*"
               ]
```

在相同 AWS 帳戶中分隔 DynamoDB 環境的 IAM 政策

假設您有不同的環境，且每個環境都會維護各自版本的資料表 (名為 ProductCatalog)。如果您在同一個 AWS 帳戶中建立兩個ProductCatalog資料表，由於許可的設定方式，在一個環境中工作可能會影響另一個環境。例如，在 AWS 帳戶層級設定並行控制平面操作數量的配額 （ 例如 CreateTable)。

因此，一個環境中的每個動作都會減少另一個環境中可用的操作數目。還有一種風險是，一個環境中的程式碼可能會在另一個環境中意外存取資料表。

> **Note**
>
> 若想區隔生產和測試工作負載來協助控制事件的潛在「爆炸範圍」，最佳實務是為測試和生產工作負載建立不同的 AWS 帳戶。如需詳細資訊，請參閱 AWS 帳戶管理與區隔。

進一步假設您有兩位開發人員 (Amit 和 Alice) 正在測試 `ProductCatalog` 資料表。您的開發人員可以共用相同的測試 AWS 帳戶，而不是每個需要個別 AWS 帳戶的開發人員。在此測試帳戶中，您可以建立相同資料表的複本以供每位開發人員處理 (例如 `Alice_ProductCatalog` 和 `Amit_ProductCatalog`)。在這種情況下，您可以在為測試環境建立的 AWS 帳戶中建立使用者 Alice 和 Amit。您接著可以將許可授予這些使用者，以對他們擁有的資料表執行 DynamoDB 動作。

若要授予這些 IAM 使用者許可，您可以執行下列其中一項：

- 為每位使用者建立不同的政策，然後分別將每個政策連接至其使用者。例如，您可以將下列政策連接至使用者 Alice，允許她存取 `Alice_ProductCatalog` 資料表上的所有 DynamoDB 動作：

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnAliceTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:DeleteItem",
              "dynamodb:DescribeContributorInsights",
              "dynamodb:RestoreTableToPointInTime",
              "dynamodb:ListTagsOfResource",
              "dynamodb:CreateTableReplica",
              "dynamodb:UpdateContributorInsights",
              "dynamodb:CreateBackup",
              "dynamodb:DeleteTable",
              "dynamodb:UpdateTableReplicaAutoScaling",
              "dynamodb:UpdateContinuousBackups",
              "dynamodb:TagResource",
              "dynamodb:DescribeTable",
              "dynamodb:GetItem",
```

```
                "dynamodb:DescribeContinuousBackups",
                "dynamodb:BatchGetItem",
                "dynamodb:UpdateTimeToLive",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem",
                "dynamodb:UntagResource",
                "dynamodb:PutItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteTableReplica",
                "dynamodb:DescribeTimeToLive",
                "dynamodb:RestoreTableFromBackup",
                "dynamodb:UpdateTable",
                "dynamodb:DescribeTableReplicaAutoScaling",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:DescribeLimits",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
Alice_ProductCatalog/*"
        }
    ]
}
```

然後，您可以為使用者 Amit 建立具有不同資源 (Amit_ProductCatalog 資料表) 的類似政策。

- 您可以使用 IAM 政策變數來撰寫單一政策，並將它連接至群組，而不是將政策連接至個別使用者。您需要建立群組，而在此範例中，會將使用者 Alice 和 Amit 新增至群組。下列範例授予對 ${aws:username}_ProductCatalog 資料表執行所有 DynamoDB 動作的許可。評估政策時，政策變數 ${aws:username} 會取代為申請者的使用者名稱。例如，如果 Alice 傳送新增項目的請求，則只有在 Alice 將項目新增至 Alice_ProductCatalog 資料表時，才允許此動作。

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ActionsOnUserSpecificTable",
            "Effect": "Allow",
```

```
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
              "dynamodb:BatchWriteItem",
              "dynamodb:GetItem",
              "dynamodb:BatchGetItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
${aws:username}_ProductCatalog"
        },
        {
            "Sid": "AdditionalPrivileges",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables",
                "dynamodb:DescribeTable",
                "dynamodb:DescribeContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/*"
        }
    ]
}
```

> **Note**
>
> 使用 IAM 政策變數時，您必須在政策中明確地指定 2012-10-17 版本的 IAM 政策語言。IAM 政策語言 (2008-10-17) 的預設版本不支援政策變數。

您可以對資料表使用萬用字元 (*) 授予許可 (而不是如同一般作法將特定資料表識別為資源)，在這些資料表中，資料表名稱的前面會加上提出請求的使用者，如下範例所示。

```
"Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/${aws:username}_*"
```

避免購買 DynamoDB 預留容量的 IAM 政策

使用 Amazon DynamoDB 預留容量，您會支付一次性預付費用，並承諾在一段時間支付最低消費額，以大幅降低成本。您可以使用 AWS 管理主控台 檢視和購買預留容量。不過，您可能不希望組織中的所有使用者都可以購買預留容量。如需預留容量的詳細資訊，請參閱 Amazon DynamoDB 定價。

DynamoDB 提供下列 API 操作，以控制對預留容量管理的存取：

- `dynamodb:DescribeReservedCapacity`：傳回目前生效的預留容量購買。
- `dynamodb:DescribeReservedCapacityOfferings`：傳回 AWS目前所提供之預留容量方案的詳細資訊。
- `dynamodb:PurchaseReservedCapacityOfferings`：執行實際購買的預留容量。

AWS 管理主控台 使用這些 API 動作來顯示預留容量資訊並進行購買。您無法從應用程式呼叫這些操作，因為只有從主控台才能加以存取。不過，您可以在 IAM 許可政策中允許或拒絕存取這些操作。

下列政策允許使用者使用 檢視預留容量購買和優惠 AWS 管理主控台 ，但拒絕新購買。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

請注意，此政策使用萬用字元 (*) 來允許所有、和描述許可，以拒絕為所有人購買 DynamoDB 預留容量。

僅授予 DynamoDB 串流讀取存取權限的 IAM 政策 (不適用於資料表)

當您為資料表啟用 DynamoDB Streams 時，會擷取資料表中每個資料項目的修改資訊。如需詳細資訊，請參閱 DynamoDB Streams 的變更資料擷取。

在部分情況下，建議您防止應用程式讀取 DynamoDB 資料表中的資料，但同時仍然允許存取該資料表的串流。例如，您可以 AWS Lambda 設定 在偵測到項目更新時輪詢串流和叫用 Lambda 函數，然後執行其他處理。

下列動作可用來控制對 DynamoDB Streams 的存取：

- `dynamodb:DescribeStream`
- `dynamodb:GetRecords`
- `dynamodb:GetShardIterator`
- `dynamodb:ListStreams`

下列範例政策會將存取名為 `GameScores` 之資料表上串流的許可授予使用者。ARN 中的萬用字元 (*) 可符合與該資料表建立關聯的任何串流。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AccessGameScoresStreamOnly",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/*"
        }
```

```
    ]
}
```

請注意，此政策會授予存取 GameScores 資料表串流的權限，但不允許存取資料表本身。

允許 AWS Lambda 函數存取 DynamoDB 串流記錄的 IAM 政策

如果您想要根據 DynamoDB 串流中的事件執行特定動作，您可以撰寫由這些事件觸發的 AWS Lambda 函數。這類 Lambda 函式需要從 DynamoDB Streams 讀取資料的許可。如需有關搭配使用 Lambda 和 DynamoDB Streams 的詳細資訊，請參閱 DynamoDB 串流和 AWS Lambda 觸發。

若要將許可授予 Lambda，請使用與 Lambda 函式 IAM 角色相關聯的許可政策 (也稱為執行角色)。在您建立 Lambda 函式時，請指定此政策。

例如，您可以將下列許可政策與執行角色建立關聯，以授予執行所列 DynamoDB Streams 動作的 Lambda 許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "APIAccessForDynamoDBStreams",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/*"
        }
    ]
}
```

如需詳細資訊，請參閱《AWS Lambda 開發人員指南》中的 AWS Lambda 許可。

允許對 DynamoDB Accelerator (DAX) 叢集進行讀取和寫入的 IAM 政策

以下政策允許對 DynamoDB Accelerator (DAX) 叢集 (而非相關聯的 DynamoDB 資料表)，執行讀取、寫入、更新和刪除。若要使用此政策，請取代 AWS 區域名稱、您的帳戶 ID 和 DAX 叢集的名稱。

> **Note**
>
> 此政策允許存取 DAX 叢集，但不允許存取相關聯的 DynamoDB 資料表。請確定您的 DAX 叢集具有正確政策，可代表您在 DynamoDB 資料表上執行這些相同操作。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonDynamoDBDAXDataOperations",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:PutItem",
                "dax:ConditionCheckItem",
                "dax:BatchGetItem",
                "dax:BatchWriteItem",
                "dax:DeleteItem",
                "dax:Query",
                "dax:UpdateItem",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:eu-west-1:123456789012:cache/MyDAXCluster"
        }
    ]
}
```

若要展開此政策以涵蓋帳戶所有 AWS 區域的 DAX 存取，請針對區域名稱使用萬用字元 (*)。

```
"Resource": "arn:aws:dax:*:123456789012:cache/MyDAXCluster"
```

Amazon DynamoDB 身分識別和存取疑難排解

請使用以下資訊來協助您診斷和修正使用 DynamoDB 和 IAM 時發生的常見問題。

主題

- 我未獲授權，不得在 DynamoDB 中執行動作
- 我未獲得執行 iam:PassRole 的授權
- 我想要允許 以外的人員 AWS 帳戶 存取我的 DynamoDB 資源

我未獲授權，不得在 DynamoDB 中執行動作

如果 AWS 管理主控台 告訴您無權執行 動作，則必須聯絡您的管理員尋求協助。您的管理員是為您提供使用者名稱和密碼的人員。

下列範例錯誤會在 `mateojackson` 使用者嘗試使用主控台檢視一個虛構 *`my-example-widget`* 資源的詳細資訊，但卻無虛構 `aws:`*`GetWidget`* 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

在此情況下，Mateo 會請求管理員更新他的政策，允許他使用 *`my-example-widget`* 動作存取 `aws:`*`GetWidget`* 資源。

我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您無權執行 `iam:PassRole` 動作，您的政策必須更新，允許您將角色傳遞給 DynamoDB。

有些 AWS 服務 可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 `marymajor` 的 IAM 使用者嘗試使用主控台在 DynamoDB 中執行動作時，發生下列範例錯誤。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞給服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 `iam:PassRole` 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我想要允許 以外的人員 AWS 帳戶 存取我的 DynamoDB 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 DynamoDB 是否支援這些功能，請參閱 Amazon DynamoDB 如何搭配 IAM 運作。
- 若要了解如何 AWS 帳戶 在您擁有的 資源間提供存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 IAM 使用者中提供存取權。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方 AWS 帳戶 擁有。
- 如需了解如何透過聯合身分提供存取權，請參閱《IAM 使用者指南》中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

避免購買 DynamoDB 預留容量的 IAM 政策

使用 Amazon DynamoDB 預留容量，您會支付一次性預付費用，並承諾在一段時間支付最低消費額，以大幅降低成本。您可以使用 AWS 管理主控台 檢視和購買預留容量。不過，您可能不希望組織中的所有使用者都可以購買預留容量。如需預留容量的詳細資訊，請參閱 Amazon DynamoDB 定價。

DynamoDB 提供下列 API 操作，以控制對預留容量管理的存取：

- `dynamodb:DescribeReservedCapacity`：傳回目前生效的預留容量購買。
- `dynamodb:DescribeReservedCapacityOfferings`：傳回 AWS目前所提供之預留容量方案的詳細資訊。
- `dynamodb:PurchaseReservedCapacityOfferings`：執行實際購買的預留容量。

AWS 管理主控台 使用這些 API 動作來顯示預留容量資訊並進行購買。您無法從應用程式呼叫這些操作，因為只有從主控台才能加以存取。不過，您可以在 IAM 許可政策中允許或拒絕存取這些操作。

下列政策允許使用者使用 檢視預留容量購買和優惠 AWS 管理主控台 ，但拒絕新購買。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

請注意，此政策使用萬用字元 (*) 來允許所有、和描述許可，以拒絕為所有人購買 DynamoDB 預留容量。

使用 IAM 政策條件進行精細定義存取控制

您在 DynamoDB 中授予許可時，可以指定條件，以決定許可政策的生效方式。

概觀

在 DynamoDB 中，您可以選擇在使用 IAM 政策授予許可時指定條件 (請參閱 Amazon DynamoDB 的 Identity and Access Management)。例如，您可以：

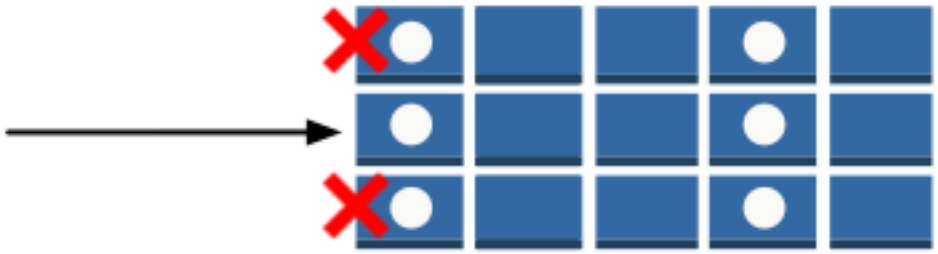
- 授予許可，允許使用者唯讀存取資料表或次要索引中的特定項目和屬性。
- 授予許可，允許使用者根據該使用者的身分唯寫存取資料表中的特定屬性。

在 DynamoDB 中，您可以使用條件金鑰在 IAM 政策中指定條件，如下節中的使用案例所述。

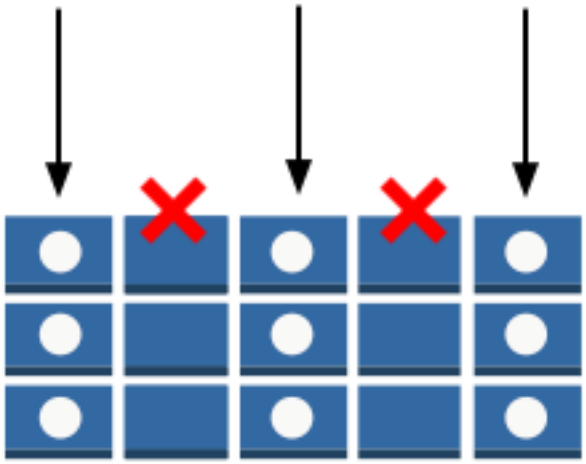
許可使用案例

除了控制對 DynamoDB API 動作的存取之外，您也可以控制對個別資料項目和屬性的存取。例如，您可以執行下列動作：

- 授予資料表的許可，但根據特定主索引鍵值來限制存取該資料表中的特定項目。範例可能是遊戲的社交聯網應用程式，而所有使用者的已儲存遊戲資料都會存放在單一資料表中，但使用者無法存取他們未擁有的資料項目，如下圖所示：



- 隱藏資訊，讓使用者只能看到一部分的屬性。範例可能是根據使用者位置來顯示附近機場之航班資料的應用程式。航空公司名稱、到達和起飛時間，以及航班編號全部會顯示。不過，會隱藏駕駛員名稱或乘客數目這類屬性，如下圖所示：



若要實作這類精細定義存取控制，您可以撰寫 IAM 許可政策，以指定條件來存取安全憑證和相關聯許可。您接著將政策套用至使用 IAM 主控台所建立的使用者、群組或角色。您的 IAM 政策可以限制對資料表中個別項目的存取、對這些項目中屬性的存取，或同時限制兩者的存取。

您可以選擇性地使用 Web 聯合身分來控制透過 Login with Amazon、Facebook 或 Google 進行身分驗證之使用者的存取。如需詳細資訊，請參閱 使用 Web 聯合身分。

您可以使用 IAM Condition 元素來實作精細定義存取控制政策。將 Condition 元素新增至許可政策，即可根據特定商業需求來允許或拒絕存取 DynamoDB 資料表和索引中的項目和屬性。

以下影片說明如何在 DynamoDB 中透過 IAM 政策條件實作精細存取控制。

了解 DynamoDB 中的精細存取控制

DynamoDB 中的精細存取控制可讓您在多個層級建立精確的許可界限：

1. 項目層級存取控制：限制使用者只能存取包含特定金鑰值的項目，通常符合其身分或許可範圍。
2. 屬性層級存取控制：限制使用者可以檢視或修改哪些屬性（資料欄），可讓您保護敏感資訊，同時允許存取相同項目中的非敏感資料。
3. 操作特定的控制項：根據正在執行的操作類型，套用不同的許可規則。

這些控制項是透過使用 DynamoDB 特定條件金鑰的 IAM 政策實作。

指定條件：使用條件金鑰

AWS 為支援 IAM 進行存取控制的所有 AWS 服務提供一組預先定義的條件金鑰 (AWS全條件金鑰）。例如，您可以使用 aws:SourceIp 條件金鑰先檢查申請者的 IP 位址，再允許執行動作。如需詳細資訊和 AWS全局金鑰清單，請參閱《IAM 使用者指南》中的可用條件金鑰。

以下是適用於 DynamoDB 的 DynamoDB 服務特定條件金鑰。

dynamodb:LeadingKeys

代表資料表的第一個索引鍵屬性，換言之，即分割區索引鍵。索引鍵名稱 LeadingKeys 是複數，即使索引鍵與單一項目動作一起使用也是一樣。此外，在條件中使用 ForAllValues 時，您必須使用 LeadingKeys 修飾詞。

dynamodb:Select

代表請求的 Select 參數。Select 可以是下列的任何值：

- ALL_ATTRIBUTES
- ALL_PROJECTED_ATTRIBUTES
- SPECIFIC_ATTRIBUTES
- COUNT

> **Note**
>
> 雖然通常與查詢和掃描操作相關聯，但此條件索引鍵適用於傳回項目屬性的所有 DynamoDB 操作，且對於控制所有 API 動作的屬性存取至關重要。在此條件金鑰上使用 StringEqualsIfExists 或類似限制條件，會對套用此條件金鑰的操作施加限制，同時忽略不適用的操作。

dynamodb:Attributes

代表請求存取的最上層屬性清單。如果請求或其包含的任何巢狀屬性是在請求參數中指定，則會由請求存取頂層屬性。例如，指定 ProjectionExpression的GetItem請求"Name, Address.City"，dynamodb:Attributes清單將包含 "Name" 和 "Address"。如果在精細存取控制政策中列舉 Attributes 參數，請考慮同時限制 ReturnValues和 Select 參數，以確保限制存取多個 API 動作中的指定屬性Query，例如 GetItem、 和 Scan。

> **Note**
>
> 此條件只會根據請求中指定的屬性 （ 例如 ProjectionExpression) 進行評估，而不是回應中的屬性。如果請求中未提供 ProjectionExpression，無論政策中的任何屬性限制，都會傳回所有屬性。如需如何正確保護屬性存取的詳細資訊，請參閱以下「強制執行屬性型限制」一節。

dynamodb:ReturnValues

代表請求的 ReturnValues 參數。根據 API 動作， ReturnValues可以是下列任何值：

- ALL_OLD
- UPDATED_OLD
- ALL_NEW
- UPDATED_NEW
- NONE

dynamodb:ReturnConsumedCapacity

代表請求的 ReturnConsumedCapacity 參數。ReturnConsumedCapacity 可以是下列的一個值：

- TOTAL

- NONE

dynamodb:FirstPartitionKeyValues

代表資料表的第一個索引鍵屬性，也就是第一個分割區索引鍵。索引鍵名稱 FirstPartitionKeyValues 是複數，即使索引鍵與單一項目動作一起使用也是一樣。此外，在條件FirstPartitionKeyValues中使用 ForAllValues 時，您必須使用 修飾詞。FirstPartitionKeyValues和 LeadingKeys可以使用可交換。

dynamodb:SecondPartitionKeyValues

類似於 dynamodb:FirstPartitionKeyValues。代表資源的第二個分割區索引鍵。索引鍵名稱 SecondPartitionKeyValues 是複數，即使索引鍵與單一項目動作一起使用也是一樣。

dynamodb:ThirdPartitionKeyValues

類似於 dynamodb:FirstPartitionKeyValues。代表資源的第三個分割區索引鍵。索引鍵名稱 ThirdPartitionKeyValues 是複數，即使索引鍵與單一項目動作一起使用也是一樣。

dynamodb:FourthPartitionKeyValues

類似於 dynamodb:FirstPartitionKeyValues。代表資源的第四個分割區索引鍵。索引鍵名稱 FourthPartitionKeyValues 是複數，即使索引鍵與單一項目動作一起使用也是一樣。

確保強制執行屬性型限制

使用屬性型條件來限制對特定屬性的存取時，請務必了解這些條件的評估方式：

- 屬性條件只會對請求中指定的屬性進行評估，而不是對回應中的屬性進行評估。
- 對於沒有 ProjectionExpression (GetItem、Query、Scan 等） 的讀取操作，無論政策中的屬性限制為何，都會傳回所有屬性。 GetItem 為了防止暴露這種潛在的敏感資料，必須同時實作屬性條件 (dynamodb:Attributes) 和需要特定屬性的條件 (dynamodb:Select)。
- 對於寫入操作 (PutItem、UpdateItem、DeleteItem)，ReturnValues 參數可以傳回完整項目，即使寫入操作本身符合您的政策，也可能會公開受限屬性。若要防止此暴露，請在政策中對 ReturnValues (dynamodb:Attributes) 實作屬性條件 (dynamodb:ReturnValues) 和限制。

限制使用者存取

許多 IAM 許可政策可讓使用者只存取資料表中分割區索引鍵值符合使用者識別符的項目。例如，遊戲應用程式先前限制是使用此方式所存取，因此使用者只能存取與其使用者 ID 建立關聯的遊戲資料。IAM 替換變數 ${www.amazon.com:user_id}、${graph.facebook.com:id} 和

`${accounts.google.com:sub}` 包含用於 Login with Amazon、Facebook 及 Google 的使用者識別符。若要了解應用程式如何登入其中一個身分提供者，並取得這些識別符，請參閱 使用 Web 聯合身分。

> **Important**
>
> 不支援使用精細存取控制來限制全域資料表的複寫。將精細存取控制的原則條件套用至用於全域資料表複寫的 DynamoDB 服務主體或服務連結角色，可能導致全域資料表內的複寫中斷。

> **Note**
>
> 下節中的每個範例都會將 `Effect` 子句設定為 `Allow`，而且只指定允許的動作、資源及參數。只允許存取 IAM 政策中明確列出的項目。
> 在某些情況下，有可能會重新撰寫這些政策，讓它們成為拒絕類型 (即，將 `Effect` 子句設定為 `Deny`，並反轉政策中的所有邏輯)。不過，建議您避免在 DynamoDB 中使用以拒絕為基礎的原則，因為這類原則比以允許為基礎的原則更難正確撰寫。此外，DynamoDB API 的未來變更 (或現有 API 輸入的變更) 也可能會導致無效的拒絕類型政策。

範例政策：使用條件進行精細定義存取控制

本節顯示數個政策，來實作 DynamoDB 資料表和索引上的精細定義存取控制。

> **Note**
>
> 所有範例都會使用 us-west-2 區域，並且包含虛構帳戶 ID。

範例 1. 具有屬性限制的基本分割區金鑰型存取控制

例如，試想有一個能讓玩家選取和玩各種不同遊戲的手機遊戲應用程式。應用程式使用名為 的 DynamoDB 資料表`GameScores`來追蹤高分和其他使用者資料。資料表中的每個項目都是依使用者 ID 和使用者所玩遊戲的名稱進行唯一識別。`GameScores` 資料表的主索引鍵包含分割區索引鍵 (`UserId`) 與排序索引鍵 (`GameTitle`)。使用者只可存取與其使用者 ID 建立關聯的遊戲資料。想要玩遊戲的使用者必須屬於名為 `GameRole` 且已連接安全政策的 IAM 角色。

若要管理應用程式中的使用者許可，您可以撰寫下列這類許可政策：

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
                  "TopScoreDateTime"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
}
```

除了授予 GameScores 資料表 (Action 元素) 上特定 DynamoDB 動作 (Resource 元素) 的許可之外，Condition 元素還會使用 DynamoDB 特有的下列條件金鑰來限制許可，如下所示：

- dynamodb:LeadingKeys：此條件金鑰可讓使用者只存取分割區索引鍵值符合其使用者 ID 的項目。此 ID ${www.amazon.com:user_id} 是替換變數。如需替換變數的詳細資訊，請參閱 使用 Web 聯合身分。
- dynamodb:Attributes：此條件金鑰會限制對所指定屬性的存取，因此只有許可政策中所列的動作才能傳回這些屬性的值。此外，StringEqualsIfExists 子句可以確保應用程式必須一律提供要處理的特定屬性清單，以及確保應用程式無法請求所有屬性。

評估 IAM 政策時，結果一律是 true (允許存取) 或 false (拒絕存取)。如果 Condition 元素的任一部分是 false，則整個政策會評估為 false，並拒絕存取。

> **Important**
>
> 如果您使用 dynamodb:Attributes，則必須指定政策中所列資料表和任何次要索引之所有主索引鍵和索引鍵屬性的名稱。否則，DynamoDB 無法使用這些索引鍵屬性來執行所請求的動作。

IAM 政策文件只能包含下列 Unicode 字元：水平定位字元 (U+0009)、換行字元 (U+000A)、歸位字元 (U+000D)，以及 U+0020 到 U+00FF 範圍內的字元。

範例 2：授予許可，以限制對具有特定分割區索引鍵值之項目的存取

下列許可政策授予許可，以允許 GamesScore 資料表上的一組 DynamoDB 動作。它會使用 dynamodb:LeadingKeys 條件金鑰，只限制下列項目的使用者動作：其 UserID 主索引鍵值符合此應用程式之 Login with Amazon 唯一使用者 ID 的項目。

> **Important**
>
> 動作清單不包含 Scan 的許可，因為 Scan 會傳回所有項目，不論前導索引鍵為何。

JSON

```
{
   "Version":"2012-10-17",
```

```
   "Statement":[
      {
         "Sid":"FullAccessToUserItems",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
         }
      }
   ]
}
```

> **Note**
>
> 使用政策變數時，您必須在政策中明確指定 2012-10-17 版。存取政策語言的預設版本 2008-10-17，不支援政策變數。

若要實作唯讀存取權，您可以移除任何可修改資料的動作。在下列政策中，條件中只會包含提供唯讀存取權的動作。

JSON

```
{
   "Version":"2012-10-17",
```

```
   "Statement":[
      {
         "Sid":"ReadOnlyAccessToUserItems",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
         }
      }
   ]
}
```

> **Important**
>
> 如果您使用 dynamodb:Attributes，則必須指定政策中所列資料表和任何次要索引之所有主索引鍵和索引鍵屬性的名稱。否則，DynamoDB 無法使用這些金鑰屬性來執行所請求的動作。

範例 3：授予許可，限制對資料表中特定屬性的存取

下列許可政策透過新增 dynamodb:Attributes 條件金鑰，來允許只存取資料表中的兩個特定屬性。在條件式寫入或掃描篩選條件中，可以讀取、寫入或評估這些屬性。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
```

```
      {
         "Sid":"LimitAccessToSpecificAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem",
            "dynamodb:Scan"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "UserId",
                  "TopScore"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

> **Note**
>
> 政策會採用允許清單方式，以允許存取一組指定的屬性。您可以撰寫同等的政策，改為拒絕存取其他屬性。我們不建議使用此拒絕清單方式。使用者可以決定這些受拒絕的屬性，請遵循最低權限政策，如 Wikipedia (網址為 http://en.wikipedia.org/wiki/Principle_of_least_privilege) 中所述，並使用允許清單方式來列舉所有允許值，而不是指定拒絕屬性。

此政策不允許 PutItem、DeleteItem 或 BatchWriteItem。這些動作一律會取代整個先前項目，以允許使用者刪除不允許它們存取之屬性的先前值。

許可政策中的 StringEqualsIfExists 子句確保下列狀況：

- 如果使用者指定 Select 參數，則其值必須是 SPECIFIC_ATTRIBUTES。此需求防止 API 動作傳回任何不允許的屬性，例如來自索引投影。
- 如果使用者指定 ReturnValues 參數，則其值必須是 NONE、UPDATED_OLD 或 UPDATED_NEW。這是必要項目，因為 UpdateItem 動作也會執行隱含讀取操作來檢查項目在取代前是否存在，因此，請求時可以傳回先前的屬性值。以這種方式限制 ReturnValues 可確定使用者只能讀取或寫入允許的屬性。
- StringEqualsIfExists 子句可確保在允許動作內容中，一個請求只能使用其中一個參數：Select 或 ReturnValues。

以下是此政策的某些變化：

- 若只要允許讀取動作，您可以從允許動作清單中移除 UpdateItem。因為其餘動作不接受 ReturnValues 動作，所以您可以從條件中移除 ReturnValues。您也可以將 StringEqualsIfExists 變更為 StringEquals，因為 Select 參數一律會有值 (除非特別指定，否則為 ALL_ATTRIBUTES)。
- 若只要允許寫入動作，您可以從允許動作清單中移除所有項目，但 UpdateItem 除外。因為 UpdateItem 未使用 Select 參數，所以您可以從條件中移除 Select。您也必須將 StringEqualsIfExists 變更為 StringEquals，因為 ReturnValues 參數一律會有值 (除非特別指定，否則為 NONE)。
- 若要允許名稱符合某模式的所有屬性，請使用 StringLike，而非 StringEquals，並使用多字元模式相符萬用字元 (*)。

範例 4：授予許可以防止更新特定屬性

下列許可政策限制只更新 dynamodb:Attributes 條件金鑰所識別之特定屬性的使用者存取。StringNotLike 條件防止應用程式更新使用 dynamodb:Attributes 條件金鑰所指定的屬性。

JSON

```
{
```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"PreventUpdatesOnCertainAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
         "Condition":{
            "ForAllValues:StringNotLike":{
               "dynamodb:Attributes":[
                  "FreeGamesAvailable",
                  "BossLevelUnlocked"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

注意下列事項：

- UpdateItem 動作 (如同其他寫入動作) 需要項目的讀取存取，以便其傳回更新前後的值。在政策中，您會限制只存取特定屬性的動作，且這些屬性僅限於允許藉由指定 dynamodb:ReturnValues 條件金鑰來進行更新者。條件金鑰會限制請求中的 ReturnValues，只指定 NONE、UPDATED_OLD 或 UPDATED_NEW，而且未包含 ALL_OLD 或 ALL_NEW。
- StringEqualsIfExists 運算子可確保如果 dynamodb:Select或 dynamodb:ReturnValues 存在於請求中，則必須符合指定的值。這可防止操作傳回完整項目。
- 限制屬性更新時，您也應該控制可以傳回哪些資料，以防止受保護屬性的資訊洩露。
- PutItem 和 DeleteItem 動作會取代整個項目，因此允許應用程式修改任何屬性。因此，限制應用程式只更新特定屬性時，您不應該授予這些 API 的許可。

範例 5：授予僅查詢索引中投影屬性的許可

下列許可政策使用 dynamodb:Attributes 條件金鑰，允許對次要索引 (TopScoreDateTimeIndex) 進行查詢。政策也會限制只請求已投影到索引之特定屬性的查詢。

若需要應用程式在查詢中指定屬性清單，政策也會指定 dynamodb:Select 條件金鑰需要 DynamoDB Query 動作的 Select 參數是 SPECIFIC_ATTRIBUTES。屬性清單限制為使用 dynamodb:Attributes 條件金鑰所提供的特定清單。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"QueryOnlyProjectedIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "TopScoreDateTime",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "Attempts"
               ]
            },
            "StringEquals":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
}
```

下列許可政策類似，但查詢必須請求所有已投影到索引的屬性。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"QueryAllIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:Select":"ALL_PROJECTED_ATTRIBUTES"
            }
         }
      }
   ]
}
```

範例 6：授予許可，以限制對特定屬性和分割區索引鍵值的存取

下列許可政策允許資料表和資料表索引 (指定於 `Action` 元素) 的特定 DynamoDB 動作 (指定於 `Resource` 元素)。此政策使用 `dynamodb:LeadingKeys`條件索引鍵來限制許可，僅限分割區索引鍵值符合使用者 Facebook ID 的項目。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"LimitAccessToCertainAttributesAndKeyValues",
         "Effect":"Allow",
```

```
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${graph.facebook.com:id}"
               ],
               "dynamodb:Attributes":[
                  "attribute-A",
                  "attribute-B"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

注意下列事項：

- 政策 (UpdateItem) 允許的寫入動作只能修改屬性 A 或屬性 B。
- 因為政策允許 UpdateItem，所以應用程式可以插入新項目，而隱藏屬性在新項目中會是 Null。如果這些屬性投影到 TopScoreDateTimeIndex，則政策將有助於防止會致使從資料表擷取的查詢。
- 應用程式無法讀取 dynamodb:Attributes 中所列屬性以外的任何屬性。在此政策下，應用程式必須將讀取請求中的 Select 參數設定為 SPECIFIC_ATTRIBUTES，而且只能請求列於允許清單上的

屬性。對於寫入請求，應用程式無法將 ReturnValues 設定為 ALL_OLD 或 ALL_NEW，而且無法根據任何其他屬性來執行條件式寫入操作。

範例 7：拒絕限制存取資料表中特定屬性的許可

下列政策拒絕存取敏感屬性，並確保無法透過省略投影表達式來略過此限制。它允許對CustomerData資料表的一般存取，同時明確拒絕對 SSN和 CreditCardNumber 屬性的存取。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:Scan"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/CustomerData"
      },
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:Scan"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/CustomerData",
         "Condition":{
            "ForAnyValue:StringEquals":{
               "dynamodb:Attributes":[
                  "SSN",
                  "CreditCardNumber"
               ]
            }
         }
      },
      {
         "Effect":"Deny",
         "Action":[
```

```
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:Scan"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/CustomerData",
         "Condition":{
            "StringNotEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
}
```

相關主題

- Amazon DynamoDB 的 Identity and Access Management
- DynamoDB API 許可：動作、資源和條件參考

使用 Web 聯合身分

如果您要撰寫以大量使用者為目標的應用程式，也可以選擇使用 web 聯合身分進行身分驗證和授權。Web 聯合身分讓您不再需要建立個別的使用者。反之，使用者可以登入身分提供者，然後從 AWS Security Token Service () 取得臨時安全登入資料AWS STS。然後，應用程式可以使用這些登入資料來存取 AWS 服務。

Web 聯合身分支援下列身分提供者：

- 登入 Amazon
- Facebook
- Google

Web 聯合身分的其他資源

下列資源可協助您進一步了解 Web 聯合身分：

- AWS 開發人員部落格中的使用 適用於 .NET 的 AWS SDK的 Web 聯合身分文章，會介紹如何使用 Facebook 的 Web 聯合身分。它包含 C# 中的程式碼片段，示範如何擔任具有 Web 身分的 IAM 角色，以及如何使用臨時安全登入資料來存取 AWS 資源。

- AWS Mobile SDK for iOS 和 適用於 Android 的 AWS Mobile SDK 包含範例應用程式。這些應用程式所包含的程式碼示範如何調用身分提供者，接著示範如何使用這些供應商中的資訊來取得和使用暫時安全憑證。
- 與行動應用程式的 Web 聯合身分一文討論 Web 聯合身分，並顯示如何使用 Web 聯合身分存取 AWS 資源的範例。

Web 聯合身分的範例政策

若要顯示如何使用 DynamoDB 的 Web 聯合身分，請重新造訪 使用 IAM 政策條件進行精細定義存取控制 中介紹的 GameScores 資料表。以下是 GameScores 的主索引鍵。

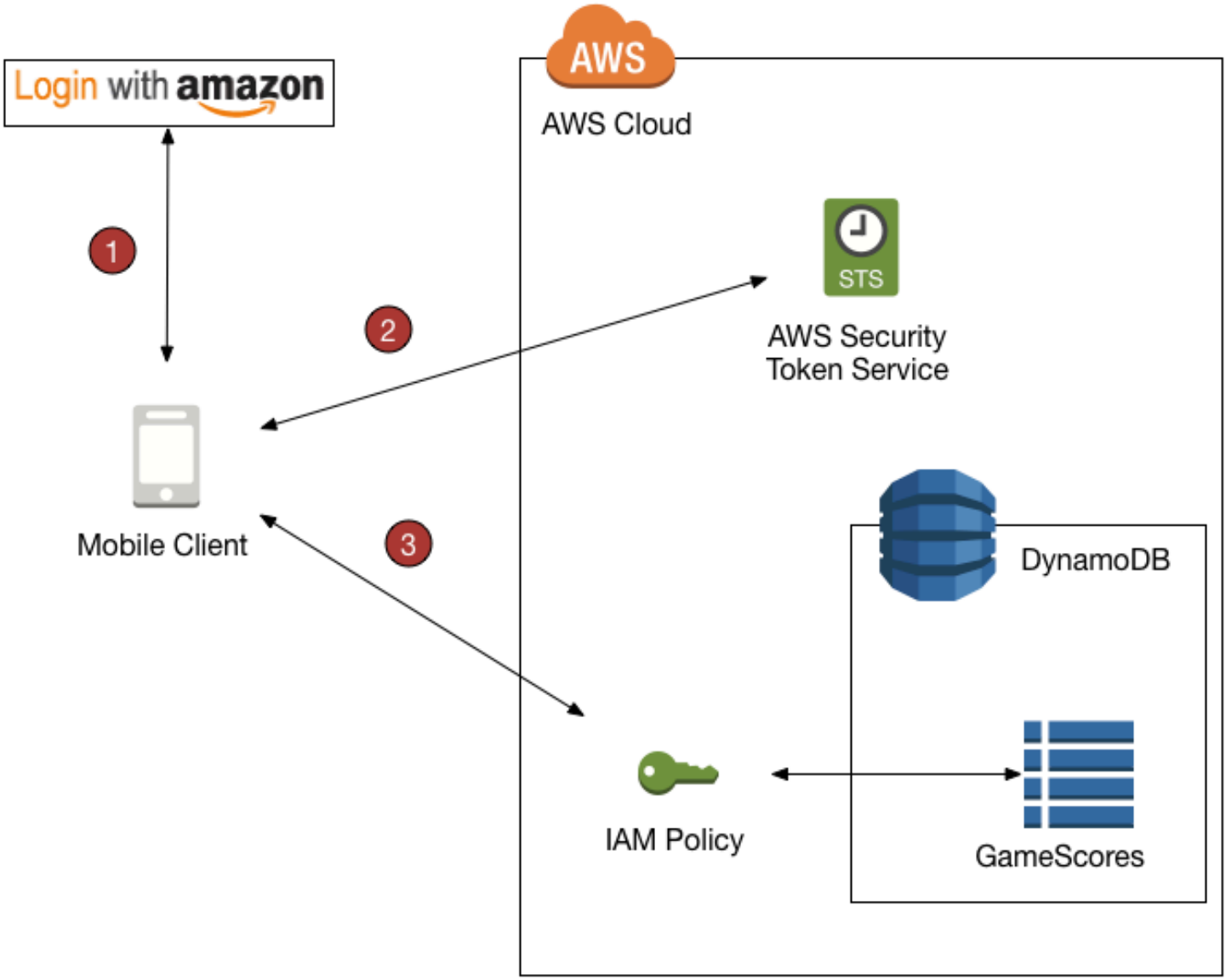
資料表名稱	主索引鍵類型	分割區索引鍵名稱和類型	排序索引鍵名稱和類型
GameScores (UserId、GameTitle ...)	複合	屬性名稱：UserId 類型：字串	屬性名稱：GameTitle 類型：字串

現在假設手機遊戲應用程式使用此資料表，而且應用程式需要支援數千或甚至數百萬使用者。就此規模而言，會變得很難管理個別應用程式使用者，且很難保證每位使用者只能存取 GameScores 資料表中的個別資料。好消息是，許多使用者都擁有第三方身分提供者 (例如 Facebook、Google 或 Login with Amazon) 的帳戶。因此可以合理地運用其中一個提供者來進行身分驗證任務。

若要使用 Web 聯合身分來執行這個操作，則應用程式開發人員必須向身分提供者 (例如 Login with Amazon) 註冊應用程式，並取得唯一應用程式 ID。接下來，開發人員需要建立 IAM 角色。(在此範例中，此角色的名稱為 GameRole。) 此角色必須已連接 IAM 政策文件、指定應用程式可存取 GameScores 資料表的條件。

使用者想要玩遊戲時，會從遊戲應用程式登入其 Login with Amazon 帳戶。應用程式接著會呼叫 AWS Security Token Service (AWS STS)，提供 Login with Amazon 應用程式 ID 並在 GameRole 中請求成員資格。 AWS STS 會傳回臨時 AWS 登入資料給應用程式，並允許其存取 GameScores 資料表，但需遵循 GameRole 政策文件。

下圖顯示這些部分如何一起使用。



Web 聯合身分概觀

1. 應用程式會呼叫第三方身分提供者，以對使用者和應用程式進行身分驗證。身分提供者會將 Web 身分字符傳回給應用程式。
2. 應用程式會呼叫 AWS STS 並將 Web 身分字符做為輸入傳遞。 會 AWS STS 授權應用程式並提供其暫時 AWS 存取憑證。應用程式可以擔任 IAM 角色 (GameRole)，並根據角色的安全政策存取 AWS 資源。
3. 應用程式呼叫 DynamoDB 來存取 GameScores 資料表。因為應用程式已擔任 GameRole，所以受限於與該角色建立關聯的安全政策。政策文件會防止應用程式存取不屬於使用者的資料。

再說明一次，以下是顯示在 使用 IAM 政策條件進行精細定義存取控制 中的 GameRole 安全性政策：

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
                  "TopScoreDateTime"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
```

```
}
```

Condition 子句決定 GameScores 在應用程式中可看到的項目。作法是比較 Login with Amazon ID 與 UserId 中的 GameScores 分割區索引鍵值。只有使用此政策中所列的其中一個 DynamoDB 動作才能處理屬於目前使用者的項目。但無法存取資料表中的其他項目。甚至，只能存取政策中所列的特定屬性。

準備使用 Web 聯合身分

如果您是應用程式開發人員，並想要針對應用程式使用 Web 聯合身分，請遵循下列步驟進行：

1. 向第三方身分提供者註冊為開發人員。下列外部連結提供向所支援之身分提供者註冊的資訊：
   - Login with Amazon 開發人員中心
   - Facebook 網站上的註冊
   - 在 Google 網站上使用 OAuth 2.0 存取 Google API
2. 向身分提供者註冊應用程式。當您這麼做時，提供者會將您應用程式的唯一 ID 提供給您。如果您想要應用程式使用多個身分提供者，則需要向每個供應商取得應用程式 ID。
3. 建立一或多個 IAM 角色。每個應用程式的各個身分供應商都需要一個角色。例如，您可能會建立使用者使用 Login with Amazon 登入之應用程式可擔任的角色、使用者使用 Facebook 登入之相同應用程式可擔任的次要角色，以及使用者使用 Google 登入之應用程式可擔任的第三個角色。

   在建立角色程序期間，您需要將 IAM 政策連接至此角色。您的政策文件應該定義應用程式所需的 DynamoDB 資源，以及存取這些資源的許可。

如需詳細資訊，請參閱《IAM 使用者指南》中的關於 Web 聯合身分。

> **Note**
>
> 或者 AWS Security Token Service，您可以使用 Amazon Cognito。Amazon Cognito 是管理行動應用程式臨時憑證的偏好服務。如需詳細資訊，請參閱《Amazon Cognito 開發人員指南》中的取得憑證。

使用 DynamoDB 主控台產生 IAM 政策

DynamoDB 主控台可協助您建立 IAM 政策，以與 Web 聯合身分搭配使用。若要執行此操作，您可以選擇 DynamoDB 資料表，並指定要包含在政策中的身分提供者、動作和屬性。DynamoDB 主控台接著會產生可連接至 IAM 角色的政策。

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在導覽窗格中，選擇 Tables (資料表)。
3. 在資料表清單中，選擇想要用於建立 IAM 政策的資料表。
4. 選取動作按鈕，然後選擇建立存取控制政策。
5. 選擇政策的身分提供者、動作和屬性。

   完成您要的設定後，選擇產生政策。即會出現產生的政策。
6. 選擇請參閱文件，然後遵循所需的步驟將產生的政策附加到 IAM 角色。

撰寫應用程式使用 Web 聯合身分

若要使用 Web 聯合身分，您的應用程式必須假設您建立的 IAM 角色。完成後，應用程式會遵守您連接到角色的存取政策。

在執行時期，如果您的應用程式使用 Web 聯合身分，則必須遵循下列步驟進行：

1. 使用第三方身分提供者進行身分驗證。您的應用程式必須使用身分提供者所提供的介面來呼叫身分提供者。對使用者進行身分驗證的確切方式取決於供應商以及執行應用程式的平台。一般而言，如果使用者尚未登入，則身分供應商會負責顯示該供應商的登入頁面。

   身分提供者對使用者進行身分驗證之後，供應商會將 Web 身分字符傳回給應用程式。此字符的格式取決於供應商，但通常是極長的字元字串。
2. 取得臨時 AWS 安全登入資料。若要執行此操作，您的應用程式會將 `AssumeRoleWithWebIdentity` 請求傳送至 AWS Security Token Service (AWS STS)。此請求包含以下項目：
   - 先前步驟中的 Web 身分字符
   - 來自身分提供者的應用程式 ID
   - 針對此應用程式的這個身分提供者所建立之 IAM 角色的 Amazon Resource Name (ARN)

   AWS STS 會傳回一組在特定時間後過期 AWS 的安全登入資料 （ 預設為 3，600 秒 ）。

以下是 AWS STS中 AssumeRoleWithWebIdentity 動作的範例請求和回應。Web 身分字符取自 Login with Amazon 身分提供者。

```
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. 存取 AWS 資源。 AWS STS 的回應包含應用程式存取 DynamoDB 資源所需的資訊：

   - AccessKeyID、SecretAccessKey 和 SessionToken 欄位包含只適用於此使用者和此應用程式的安全憑證。
   - Expiration 欄位表示這些憑證的時間限制，在此時間之後就不再有效。

- `AssumedRoleId` 欄位包含應用程式已擔任之工作階段特定 IAM 角色的名稱。應用程式會在此工作階段期間遵守 IAM 政策文件中的存取控制。
- `SubjectFromWebIdentityToken` 欄位包含出現在此特定身分提供者的 IAM 政策變數中的唯一 ID。以下是所支援供應商的 IAM 政策變數，及其範例值：

政策變數	範例值
`${www.amazon.com:user_id}`	`amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE`
`${graph.facebook.com:id}`	`123456789`
`${accounts.google.com:sub}`	`123456789012345678901`

如需使用這些政策變數的範例 IAM 政策，請參閱 範例政策：使用條件進行精細定義存取控制。

如需如何 AWS STS 產生臨時存取登入資料的詳細資訊，請參閱《IAM 使用者指南》中的請求臨時安全登入資料。

DynamoDB API 許可：動作、資源和條件參考

當您設定 Amazon DynamoDB 的 Identity and Access Management 並撰寫可連接至 IAM 身分 (身分類型政策) 的許可政策時，可以參考《IAM 使用者指南》中的 Amazon DynamoDB 的動作、資源和條件金鑰。此頁面列出每個 DynamoDB API 操作、您可以授予執行動作許可的對應動作，以及您可以授予許可 AWS 的資源。您在政策的 `Action` 欄位中指定動作，然後在政策的 `Resource` 欄位中指定資源值。

您可以在 DynamoDB 政策中使用 AWS整個條件索引鍵來表達條件。如需 AWS全系列金鑰的完整清單，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素參考。

除了 AWS全局條件金鑰之外，DynamoDB 也有自己的特定金鑰，您可以在條件中使用。如需詳細資訊，請參閱使用 IAM 政策條件進行精細定義存取控制。

相關主題

- Amazon DynamoDB 的 Identity and Access Management
- 使用 IAM 政策條件進行精細定義存取控制

DynamoDB 的業界合規驗證

若要了解 是否 AWS 服務 在特定合規計劃範圍內，請參閱AWS 服務 合規計劃範圍內然後選擇您感興趣的合規計劃。如需一般資訊，請參閱AWS 合規計劃。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載報告下載 AWS Artifact。

您使用 時的合規責任 AWS 服務 取決於資料的敏感度、您公司的合規目標，以及適用的法律和法規。如需使用 時合規責任的詳細資訊 AWS 服務，請參閱 AWS 安全文件。

Amazon DynamoDB 中的資料復原及災難復原功能

AWS 全球基礎設施是以 AWS 區域和可用區域為基礎建置。 AWS 區域提供多個實體分隔和隔離的可用區域，這些可用區域以低延遲、高輸送量和高備援聯網連接。透過可用區域，您所設計與操作的應用程式和資料庫，就能夠在可用區域之間自動容錯移轉，而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如果您需要在更大的地理距離內複製資料或應用程式，請使用 AWS 本地區域。 AWS 本機區域是單一資料中心，旨在補充現有的 AWS 區域。如同所有 AWS 區域， AWS 本機區域與其他 AWS 區域完全隔離。

如需 AWS 區域和可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

Amazon DynamoDB 會自動跨區域中的三個可用區域複寫您的資料，提供內建的高耐用性和 99.99% 的可用性 SLA。此外，DynamoDB 提供數種功能，可協助支援您的資料彈性和備份需求。

隨需備份與還原

DynamoDB 提供隨需備份功能。它可讓您建立資料表的完整備份來長期保留與封存。如需詳細資訊，請參閱 DynamoDB 的隨需備份與還原。

時間點復原

時間點復原有助於保護您的 DynamoDB 資料表免遭意外寫入或刪除操作。有了時間點復原，就無需為建立、維護或排程隨需備份而煩惱。如需詳細資訊，請參閱 DynamoDB 的時間點復原。

跨 AWS 區域同步的全域資料表

DynamoDB 會自動將資料表的資料與傳輸流分散到足夠數量的伺服器上，以處理您的輸送量和儲存需求，同時保持快速且一致的效能。所有資料都存放在固態硬碟 (SSDs) 上，並自動複寫到

AWS 區域中的多個可用區域，提供內建的高可用性和資料耐用性。您可以使用全域資料表來讓 DynamoDB 資料表跨 AWS 區域保持同步。

Amazon DynamoDB 中的基礎設施安全性

Amazon DynamoDB 是受管服務，受到位於 AWS Well-Architected Framework 的基礎設施保護中所述的 AWS 全球網路安全程序的保護。

您可以使用 AWS 發佈的 API 呼叫，透過網路存取 DynamoDB。用戶端可使用 TLS (Transport Layer Security) 1.2 或 1.3 版本。用戶端還必須支援具備完全正向加密 (PFS) 功能的密碼套件，例如臨時 Diffie-Hellman (DHE) 或橢圓曲線臨時 Diffie-Hellman (ECDHE)。現代系統 (如 Java 7 和更新版本) 大多會支援這些模式。此外，請求必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署。或者，您可以使用 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

您也可以使用 DynamoDB 的虛擬私有雲端 (VPC) 端點，讓 VPC 中的 Amazon EC2 執行個體能夠使用其私有 IP 位址來存取 DynamoDB，卻不會暴露於公有網際網路。如需詳細資訊，請參閱使用 Amazon VPC 端點來存取 DynamoDB。

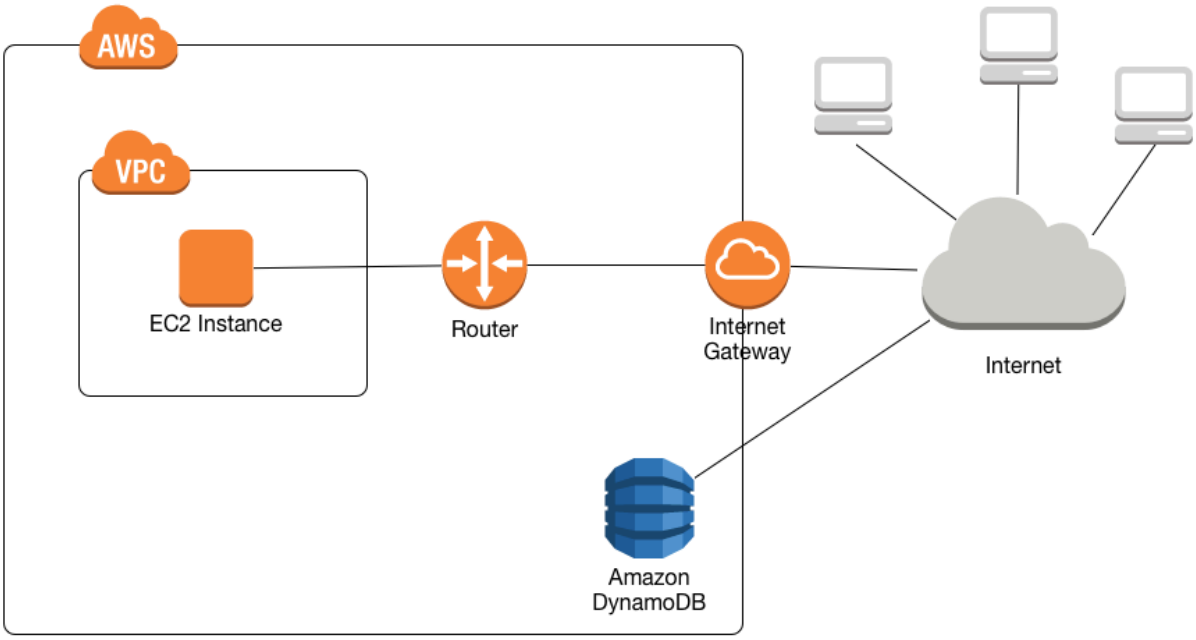
使用 Amazon VPC 端點來存取 DynamoDB

基於安全考量，許多 AWS 客戶會在 Amazon Virtual Private Cloud 環境 (Amazon VPC) 中執行其應用程式。運用 Amazon VPC，您可以將 Amazon EC2 執行個體啟動到 Virtual Private Cloud 上，此 Virtual Private Cloud 與其他網路 (包括公有網際網路) 在邏輯上隔離。使用 Amazon VPC，您可以控制其 IP 位址範圍、子網路、路由表、網路閘道及安全設定。

> **Note**
>
> 如果您在 2013 年 12 月 4 日 AWS 帳戶 之後建立 ，則您已在每個 中擁有預設 VPC AWS 區域。預設的 VPC 已可供使用：您可以立即開始使用，不需執行任何其他設定步驟。
> 如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的預設 VPC 和預設子網路。

若要存取公有網際網路，您的 VPC 必須擁有網際網路閘道，也即將 VPC 連線到網際網路的虛擬路由器。這可讓在 VPC 中的應用程式在 Amazon EC2 上執行，以便存取 Amazon DynamoDB 等網際網路資源。

根據預設，與 DynamoDB 之間的通訊會使用 HTTPS 協定，其使用 SSL/TLS 加密保護網路流量。下列圖示說明 VPC 中的 EC2 執行個體如何透過網際網路閘道而非 VPC 端點來存取 DynamoDB。



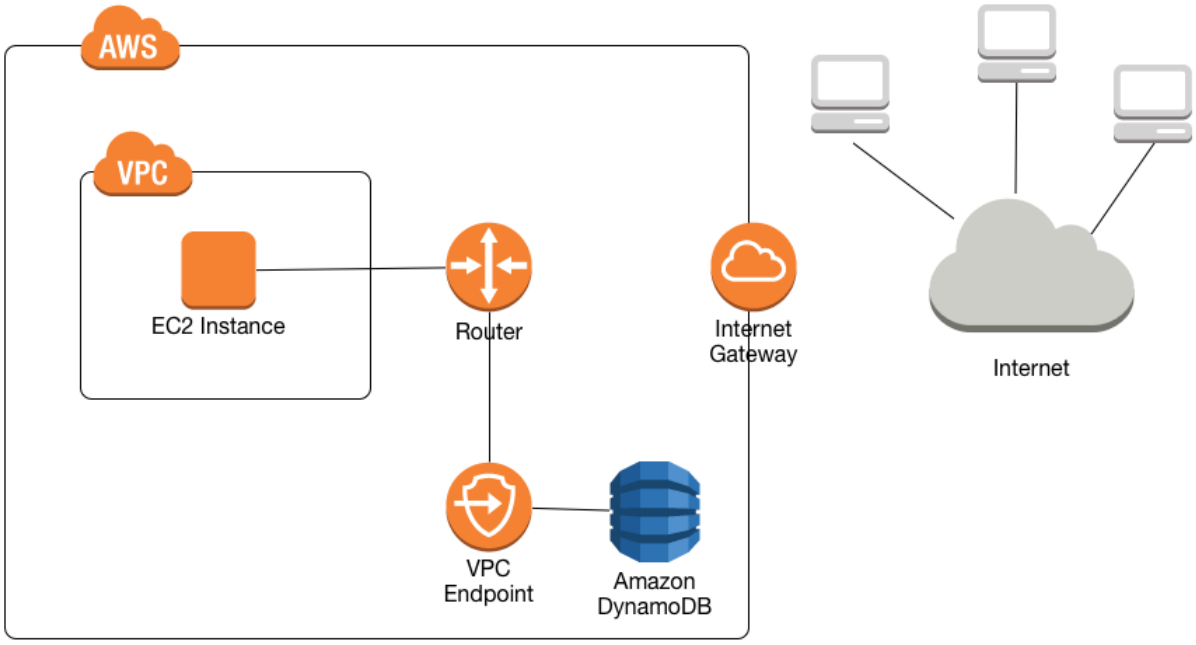
許多客戶對於透過公有的網際網路來傳送和接收資料，都會有合法的隱私與安全性疑慮。您可以使用虛擬私有網路 (VPN)，來轉傳通過您自己企業網路基礎設施的所有 DynamoDB 網路傳輸資料，以解決前述的問題。但是這個方法可能會帶來頻寬和可用性的挑戰。

DynamoDB 的 VPC 端點可以減輕這些挑戰的阻礙。DynamoDB 的 VPC 端點可讓 VPC 中的 Amazon EC2 執行個體使用私有 IP 位址來存取 DynamoDB，卻不需暴露於公有網際網路。您的 EC2 執行個體不需公有 IP 位址，您也不需要網際網路閘道、NAT 裝置或 VPC 中的虛擬私有閘道。您可以使用端點政策來控制對 DynamoDB 的存取。您的 VPC 與 AWS 服務之間的流量不會離開 Amazon 網路。

> **Note**
>
> 即使您使用公有 IP 地址，在 中託管的執行個體和服務之間的所有 VPC 通訊 AWS 都會在 AWS 網路中保持私有。來自 AWS 網路上具有目的地 AWS 的網路的封包會保留在全球 AWS 網路上，但往返 AWS 中國區域的流量除外。

當您建立 DynamoDB 使用的 VPC 端點時，區域 (例如 dynamodb.us-west-2.amazonaws.com) 內傳送到 DynamoDB 端點的所有要求，都會轉傳到 Amazon 網路中的私有 DynamoDB 端點。您不需要修改 VPC 中在 EC2 執行個體上執行的應用程式。端點的名稱會保持不變，但 DynamoDB 的路由會完全在 Amazon 網路中，不會存取公有網際網路。

下圖顯示 VPC 中的 EC2 執行個體如何使用 VPC 端點來存取 DynamoDB。



如需詳細資訊，請參閱the section called “教學課程：針對 DynamoDB 使用 VPC 端點”。

共用 Amazon VPC 端點和 DynamoDB

若要允許透過 VPC 子網路的閘道端點存取 DynamoDB 服務，您必須擁有該 VPC 子網路的擁有者帳戶許可。

一旦 VPC 子網路的閘道端點取得 DynamoDB 存取權，任何具有該子網路存取權的 AWS 帳戶都可以使用 DynamoDB。這表示 VPC 子網路中的所有帳戶使用者，都可以使用他們擁有存取權的任何 DynamoDB 資料表。這包括與 VPC 子網路以外不同帳戶關聯的 DynamoDB 資料表。VPC 子網路擁有者仍可自行決定限制子網路中任何特定使用者透過閘道端點使用 DynamoDB 服務。

教學課程：針對 DynamoDB 使用 VPC 端點

此節會逐步說明如何設定並使用 DynamoDB 的 VPC 端點。

主題

- 步驟 1：啟動 Amazon EC2 執行個體
- 步驟 2：設定您的 Amazon EC2 執行個體

- 步驟 3：建立 DynamoDB 的 VPC 端點
- 步驟 4：(選用) 清除

步驟 1：啟動 Amazon EC2 執行個體

在此步驟中，您會在預設的 Amazon VPC 中啟動 Amazon EC2 執行個體。您可以接著為 DynamoDB 建立並使用 VPC 端點。

1. 前往 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 選擇 Launch Instance (啟動執行個體) 並執行下列作業：

   步驟 1：選擇 Amazon Machine Image (AMI)

   - 在 AMI 清單的第一項上，前往 Amazon Linux AMI 並選擇 Select (選取)。

   步驟 2：選擇執行個體類型

   - 在執行個體類型清單的第一項上，選擇 t2.micro。
   - 選擇 Next: Configure Instance Details (下一步：設定執行個體詳細資訊)。

   步驟 3：設定執行個體詳細資訊

   - 前往 Network (網路)，然後選擇您的預設 VPC。

     選擇 Next: Add Storage (下一步：新增儲存體)。

   步驟 4：新增儲存體

   - 選擇 Next: Tag Instance (下一步：為執行個體新增標籤) 以跳過此步驟。

   步驟 5：將執行個體加上標籤

   - 選擇 Next: Configure Security Group (下一步：設定安全群組) 以跳過此步驟。

   步驟 6：設定安全群組

   - 選擇 Select an existing security group (選取現有的安全群組)。

- 在安全群組清單中，選擇 default (預設)。這是您 VPC 的預設安全群組。
- 選擇 Next: Review and Launch (下一步：檢閱和啟動)。

步驟 7：檢閱執行個體啟動

- 選擇啟動。

3. 在 Select an existing key pair or create a new key pair (選取現有的金鑰對或建立新的金鑰對) 視窗中，執行下列其中一項作業：
   - 如果您沒有 Amazon EC2 金鑰對，請選擇 Create a new key pair (建立新的金鑰對) 並依照指示進行。系統會要求您下載私有金鑰檔案 (.pem 檔案)，您之後登入 Amazon EC2 執行個體時會需要此檔案。
   - 若您已擁有 Amazon EC2 金鑰對，請前往 Select a key pair (選取金鑰對)，然後從清單中選擇您的金鑰對。您必須已具備可用的私有金鑰檔案 (.pem 檔案) 才能登入您的 Amazon EC2 執行個體。
4. 當您設定好金鑰對後，請選擇 Launch Instances (啟動執行個體)。
5. 返回 Amazon EC2 主控台首頁，然後選擇您啟動的執行個體。在下方窗格的 Description (描述) 標籤上，找到您執行個體的 Public DNS (公有 DNS)。例如：`ec2-00-00-00-00.us-east-1.compute.amazonaws.com`。

   請記下此公有 DNS 名稱，因為在此教學課程 (步驟 2：設定您的 Amazon EC2 執行個體) 的下一個步驟中您會需要此名稱。

> **Note**
>
> Amazon EC2 執行個體會需要幾分鐘的時間才會變成可用。在您繼續進行下一個步驟之前，請先確定 Instance State (執行個體狀態) 為 `running`，並已通過其所有 Status Checks (狀態檢查)。

步驟 2：設定您的 Amazon EC2 執行個體

在 Amazon EC2 執行個體可用時，您將能夠登入該執行個體並準備第一次使用。

> **Note**
>
> 下列步驟假設您從執行 Linux 的電腦連線至 Amazon EC2 執行個體。如需其他連線方式，請參閱《Amazon EC2 使用者指南》中的連線至 Linux 執行個體。

1. 您需要允許 SSH 流量傳入 Amazon EC2 執行個體。若要執行此動作，您需要建立新的 EC2 安全群組，然後將安全群組指派給 EC2 執行個體。

   a. 在導覽窗格中，選擇 Security Groups (安全群組)。

   b. 選擇 Create Security Group (建立安全群組)。在 Create Security Group (建立安全群組) 視窗中，執行下列動作：

      - Security group name (安全群組名稱)：為您的安全群組輸入名稱。例如：`my-ssh-access`
      - Description (描述)：為安全群組輸入簡短描述。
      - VPC：請選擇預設 VPC。
      - 在 Security group rules (安全群組規則) 區段中，選擇 Add Rule (新增規則)，並執行下列動作：
        - Type (類型)：選擇 SSH。
        - Source (來源)：選擇 My IP (我的 IP)。

      當您滿意設定後，請選擇 Create (建立)。

   c. 在導覽窗格中，選擇執行個體。

   d. 選擇您在 步驟 1：啟動 Amazon EC2 執行個體 中啟動的 Amazon EC2 執行個體。

   e. 選擇 Actions (動作) --> Networking (聯網) --> Change Security Groups (變更安全群組)。

   f. 在 Change Security Groups (變更安全群組) 下，選取您在前述步驟中建立的安全群組 (例如：`my-ssh-access`)。此外，還應選擇現有的 `default` 安全群組。完成設定後，請選擇 Assign Security Groups (指派安全群組)。

2. 使用 `ssh` 命令登入您的 Amazon EC2 執行個體，如下列範例所示。

   ```
   ssh -i my-keypair.pem ec2-user@public-dns-name
   ```

   您會需要指定您的私有金鑰檔案 (.pem 檔案) 和您執行個體的公有 DNS 名稱。(請參閱 步驟 1：啟動 Amazon EC2 執行個體)。

登入 ID 為 ec2-user。不需要任何密碼。

3. 設定您的 AWS 登入資料，如下列範例所示。根據提示輸入您的 AWS 存取金鑰 ID、秘密金鑰和預設區域名稱。

```
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]:
```

您現在可以開始建立 DynamoDB 的 VPC 端點。

步驟 3：建立 DynamoDB 的 VPC 端點

在此步驟中，您要建立 DynamoDB 的 VPC 端點並對其進行測試，確保其可以正常運作。

1. 在開始之前，請確認您可以使用 DynamoDB 的公有端點來與 DynamoDB 通訊。

```
aws dynamodb list-tables
```

輸出會顯示您目前擁有的 DynamoDB 資料表清單。(如果沒有任何資料表，清單將是空白。)

2. 確認 DynamoDB 是在目前 AWS 區域中建立 VPC 端點的可用服務。(命令會以粗體文字顯示，後面接著輸出範例。)

```
aws ec2 describe-vpc-endpoint-services
```

```
{
    "ServiceNames": [
        "com.amazonaws.us-east-1.s3",
        "com.amazonaws.us-east-1.dynamodb"
    ]
}
```

在範例輸出中，DynamoDB 是其中一個可用的服務，因此您可以繼續為其建立 VPC 端點。

3. 確定您的 VPC 識別碼。

```
aws ec2 describe-vpcs
```

```
{
    "Vpcs": [
        {
            "VpcId": "vpc-0bbc736e",
            "InstanceTenancy": "default",
            "State": "available",
            "DhcpOptionsId": "dopt-8454b7e1",
            "CidrBlock": "172.31.0.0/16",
            "IsDefault": true
        }
    ]
}
```

在範例輸出中，VPC ID 為 vpc-0bbc736e。

4. 建立 VPC 端點。對於 --vpc-id 參數，請指定上一個步驟的 VPC ID。使用 --route-table-ids 參數，將端點與路由表相關聯。

```
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. 確認您可以透過 VPC 端點存取 DynamoDB。

```
aws dynamodb list-tables
```

如果需要，您可以為 DynamoDB 嘗試其他一些 AWS CLI 命令。如需詳細資訊，請參閱《AWS CLI 命令參考》。

步驟 4：(選用) 清除

如果您要刪除自己在此教學課程中建立的資源，請按下列步驟執行：

若要移除 DynamoDB 的 VPC 端點

1. 登入 Amazon EC2 執行個體。
2. 決定 VPC 端點 ID。

```
aws ec2 describe-vpc-endpoints
```

```
{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

在範例輸出中，VPC 端點 ID 為 `vpce-9b15e2f2`。

3. 刪除 VPC 端點。

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2
```

```
{
    "Unsuccessful": []
}
```

空白陣列 [] 表示成功 (沒有未成功的請求)。

若要終止 Amazon EC2 執行個體

1. 前往 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 在導覽窗格中，選擇執行個體。
3. 選擇 Amazon EC2 執行個體。
4. 依序選擇 Actions (動作)、Instance State (執行個體狀態) 和 Terminate (終止)。
5. 在確認視窗中，請選擇 Yes, Terminate (是，終止)。

AWS PrivateLink 適用於 DynamoDB

使用 AWS PrivateLink for DynamoDB，您可以在虛擬私有雲端 (Amazon VPC) 中佈建介面 Amazon VPC 端點 （介面端點）。這些端點可直接透過 VPN 和內部部署的應用程式存取 Direct Connect，或 AWS 區域 是透過 Amazon VPC 對等互連的不同應用程式存取。使用 AWS PrivateLink 與介面端點，您可以簡化應用程式至 DynamoDB 的私有網路連線。

位於 VPC 中的應用程式無需公有 IP 位址，即可透過 VPC 介面端點與 DynamoDB 進行操作通訊。介面端點由一或多個彈性網路介面 (ENI) 所組成，這些介面會從 Amazon VPC 的子網路中指派私有 IP 位址。透過介面端點傳送至 DynamoDB 的請求會保留在 Amazon 網路內。您也可以透過 AWS Direct Connect 或 AWS Virtual Private Network ()，從內部部署應用程式存取 Amazon VPC 中的介面端點Site-to-Site VPN。如需如何將 Amazon VPC 與內部部署網路連線的詳細資訊，請參閱《Direct Connect 使用者指南》與《AWS Site-to-Site VPN 使用者指南》。

如需介面端點的一般資訊，請參閱《 指南》中的介面 Amazon VPC 端點 (AWS PrivateLink)。Amazon DynamoDB Streams 端點 AWS PrivateLink 也支援 。 AWS PrivateLink 如需詳細資訊，請參閱the section called “AWS PrivateLink 適用於 DynamoDB Streams”。

主題

- 適用於 Amazon DynamoDB 的 Amazon VPC 端點類型
- 使用 AWS PrivateLink for Amazon DynamoDB 時的考量事項
- 建立 Amazon VPC 端點
- 存取 Amazon DynamoDB 介面端點
- 從 DynamoDB 介面端點存取 DynamoDB 資料表及控制 API 操作

- 更新內部部署 DNS 組態
- 為 DynamoDB 建立 Amazon VPC 端點政策
- 搭配 AWS 管理主控台 私有存取使用 DynamoDB 端點
- AWS PrivateLink 適用於 DynamoDB Streams
- 使用 AWS PrivateLink for DynamoDB Accelerator (DAX)

適用於 Amazon DynamoDB 的 Amazon VPC 端點類型

您可以使用兩種類型的 Amazon VPC 端點來存取 Amazon DynamoDB：閘道端點和介面端點 （使用 AWS PrivateLink)。閘道端點是您在路由表中指定的閘道，可透過 AWS 網路從 Amazon VPC 存取 DynamoDB。介面端點使用私有 IP 地址，透過使用 Amazon VPC 對等互連或 ，將請求從 Amazon VPC 內部、內部部署或另一個 Amazon VPC 路由至 DynamoDB， AWS 區域 藉此擴充閘道端點的功能 AWS Transit Gateway。如需詳細資訊，請參閱什麼是 Amazon VPC 對等互連？和 Transit Gateway 與 Amazon VPC 對等互連。

介面端點與閘道端點相容。若您的 Amazon VPC 中已有現有的閘道端點，則可在同一個 Amazon VPC 中同時使用這兩種類型的端點。

DynamoDB 的閘道端點	DynamoDB 的介面端點
在這兩種情況下，您的網路流量都會保留在 AWS 網路上。	
使用 Amazon DynamoDB 公有 IP 位址	使用 Amazon VPC 中的私有 IP 位址以存取 Amazon DynamoDB
不允許從內部部署存取	允許從內部部署存取
不允許從另一個 存取 AWS 區域	AWS 區域 使用 Amazon VPC 對等互連或 允許從另一個 中的 Amazon VPC 端點存取 AWS Transit Gateway
不計費	計費

如需有關閘道端點的詳細資訊，請參閱《AWS PrivateLink 指南》中的閘道 Amazon VPC 端點。

使用 AWS PrivateLink for Amazon DynamoDB 時的考量事項

Amazon VPC 考量適用於 AWS PrivateLink Amazon DynamoDB。如需詳細資訊，請參閱《AWS PrivateLink 指南》中的介面端點考量事項和 AWS PrivateLink 配額。此外，適用下列限制。

AWS PrivateLink for Amazon DynamoDB 不支援下列項目：

- Transport Layer Security (TLS) 1.1
- 私有與混合網域名稱系統 (DNS) 服務

> **Important**
>
> 請勿建立私有託管區域來覆寫 DynamoDB 端點 DNS 名稱（例如 `dynamodb.region.amazonaws.com` 或 `*.region.amazonaws.com`），以將流量路由到您的介面端點。DynamoDB DNS 組態可能會隨著時間而變更。
> 自訂 DNS 覆寫與這些變更不相容，可能會導致請求意外透過公有 IP 地址路由，而不是您的介面端點。
> 若要透過 存取 DynamoDB AWS PrivateLink，請將您的用戶端設定為直接使用 Amazon VPC 端點 URL（例如 `https://vpce-1a2b3c4d-5e6f.dynamodb.region.vpce.amazonaws.com`）。

您啟用的每個 AWS PrivateLink 端點每秒最多可以提交 50，000 個請求。

> **Note**
>
> AWS PrivateLink 端點的網路連線逾時不在 DynamoDB 錯誤回應範圍內，需要由連線至 PrivateLink 端點的應用程式妥善處理。

建立 Amazon VPC 端點

若要建立 Amazon VPC 介面端點，請參閱《AWS PrivateLink 指南》中的建立 Amazon VPC 端點。

存取 Amazon DynamoDB 介面端點

建立介面端點時，DynamoDB 會產生兩種類型的專用 DynamoDB DNS 名稱：區域和可用區。

- 區域 DNS 名稱在其名稱`vpce.amazonaws.com`中包含唯一的 Amazon VPC 端點 ID、服務識別符 AWS 區域、 和 。例如，對於 Amazon VPC 端點 ID *vpce-1a2b3c4d*，產生的 DNS 名稱可能類似於 *vpce-1a2b3c4d-5e6f*`.dynamodb.us-east-1.vpce.amazonaws.com`。
- 地區 DNS 名稱包含可用區域，例如 *vpce-1a2b3c4d-5e6f*`-us-east-1a.dynamodb.us-east-1.vpce.amazonaws.com`。如果您的架構可隔離可用區域，則可以使用此選項。例如，您可以將其用於故障遏止或降低區域資料傳輸成本。

> **Note**
>
> 為了獲得最佳可靠性，建議您將服務部署到至少三個可用區域。

從 DynamoDB 介面端點存取 DynamoDB 資料表及控制 API 操作

您可以使用 AWS CLI AWS SDKs來存取 DynamoDB 資料表，並透過 DynamoDB 介面端點控制 API 操作。

AWS CLI 範例

若要透過 AWS CLI 命令中的 DynamoDB 介面端點存取 DynamoDB 資料表或 DynamoDB 控制 API 操作，請使用 `--region`和 `--endpoint-url` 參數。 DynamoDB

範例：建立 VPC 端點

```
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
```

範例：修改 VPC 端點

```
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
```

範例：使用端點 URL 列出資料表

在下列範例中，請以您自己的資訊取代區域 us-east-1 和 VPC 端點 ID vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com 的 DNS 名稱。

```
aws dynamodb --region us-east-1 --endpoint https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com list-tables
```

AWS 開發套件範例

若要在使用 AWS SDKs 時透過 DynamoDB 介面端點存取 DynamoDB 資料表或 DynamoDB 控制 API 操作，請將 SDK SDKs 更新至最新版本。 DynamoDB 接著，設定您的用戶端使用端點 URL，透過 DynamoDB 介面端點來存取資料表或 DynamoDB 控制 API 操作。

SDK for Python (Boto3)

範例：使用端點 URL 存取 DynamoDB 資料表

在下列範例中，將區域 us-east-1 和 VPC 端點 ID https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com 取代為您的資訊。

```
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

範例：使用端點 URL 存取 DynamoDB 資料表

在下列範例中，將區域 us-east-1 和 VPC 端點 ID https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com 取代為您的資訊。

```
//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

範例：使用端點 URL 存取 DynamoDB 資料表

在下列範例中，將區域 us-east-1 和 VPC 端點 ID https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com 取代為您的資訊。

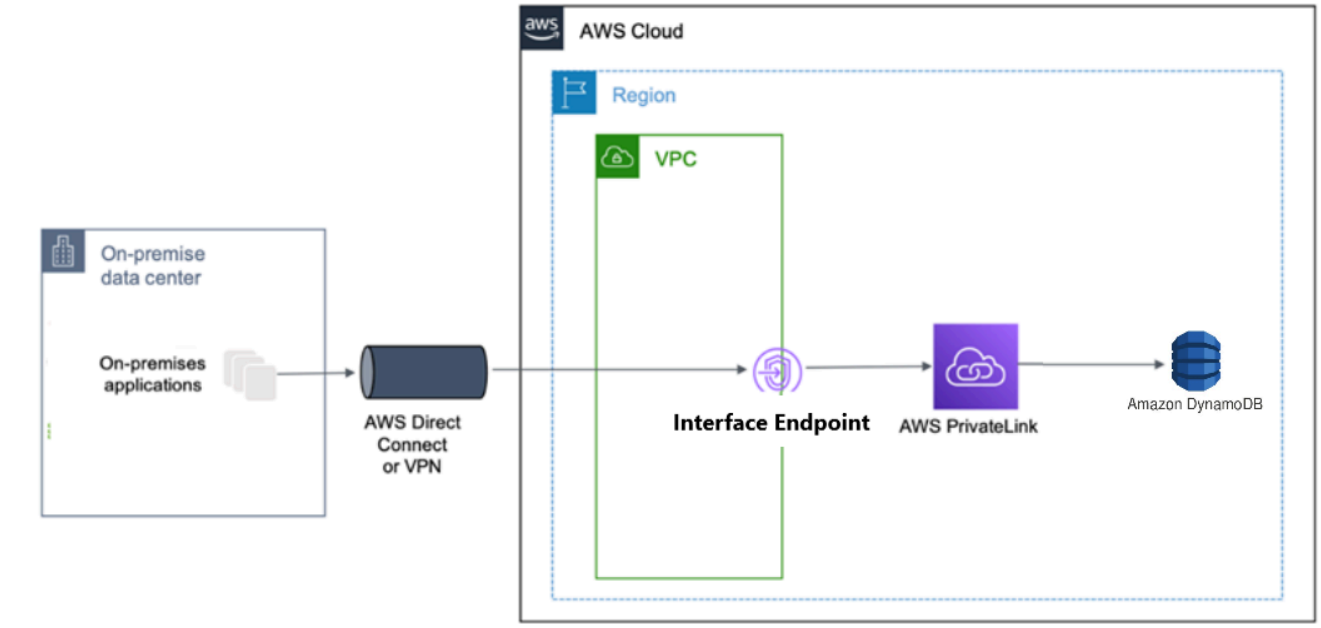
```
Region region = Region.US_EAST_1;
dynamoDbClient = DynamoDbClient.builder().region(region)
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com"))
.build()
```

更新內部部署 DNS 組態

當使用端點特定的 DNS 名稱來存取 DynamoDB 介面端點時，無需更新內部部署 DNS 解析器。您可以從公有 DynamoDB DNS 網域中，使用介面端點的私有 IP 位址解析端點特定的 DNS 名稱。

在 Amazon VPC 中，使用介面端點即可存取 DynamoDB，無需閘道端點或網際網路閘道。

您的 Amazon VPC 中的介面端點可透過 Amazon 網路，將 VPC 內及內部部署應用程式路由至 DynamoDB，如下圖所示。

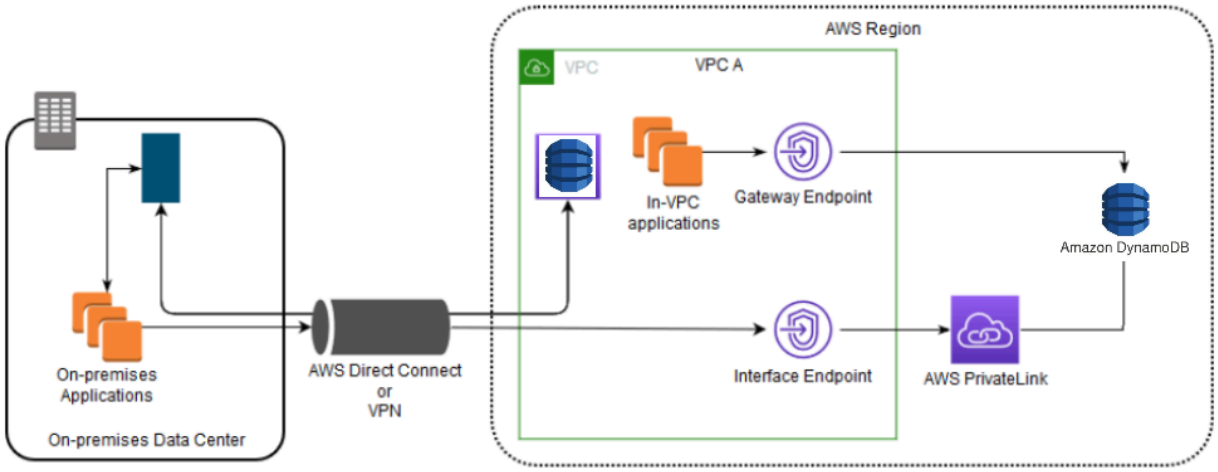


此圖展示了以下要點：

- 您的內部部署網路會使用 Direct Connect 或 Site-to-Site VPN 連線到 Amazon VPC A。
- 您的內部部署及 VPC A 中的應用程式使用端點特定的 DNS 名稱，透過 DynamoDB 介面端點存取 DynamoDB。
- 內部部署應用程式會透過 Direct Connect （或 Site-to-Site VPN) 將資料傳送至 Amazon VPC 中的介面端點。 會透過 AWS 網路將資料從介面端點 AWS PrivateLink 移至 DynamoDB。
- Amazon VPC 應用程式也會將流量傳送至介面端點。 AWS PrivateLink 會透過 AWS 網路將資料從介面端點移至 DynamoDB。

在同一個 Amazon VPC 中，同時使用閘道端點與介面端點來存取 DynamoDB。

您可以在同一個 Amazon VPC 中建立介面端點，並保留現有的閘道端點，如下圖所示。採用此方式後，VPC 內應用程式可繼續透過閘道端點存取 DynamoDB，且無需付費。之後，僅內部部署應用程式會使用介面端點存取 DynamoDB。若要這樣存取，必須更新內部部署應用程式以使用 DynamoDB 的端點特定 DNS 名稱。



此圖展示了以下要點：

- 內部部署應用程式會使用端點特定的 DNS 名稱，透過 Direct Connect （或 Site-to-Site VPN) 將資料傳送至 Amazon VPC 內的介面端點。 會透過 AWS 網路將資料從介面端點 AWS PrivateLink 移至 DynamoDB。
- 使用預設的區域 DynamoDB 名稱，Amazon VPC 內應用程式會將資料傳送至透過 AWS 網路連線至 DynamoDB 的閘道端點。

如需閘道端點的詳細資訊，請參閱《Amazon VPC 使用者指南》中的閘道 VPC 端點。

為 DynamoDB 建立 Amazon VPC 端點政策

您可以將端點政策附加至 Amazon VPC 端點，以控制對 DynamoDB 的存取。此政策會指定下列資訊：

- 可執行動作的 AWS Identity and Access Management (IAM) 委託人
- 可執行的動作
- 可在其中執行動作的資源

主題

- 範例：限制從 Amazon VPC 端點對特定資料表的存取

範例：限制從 Amazon VPC 端點對特定資料表的存取

您可以建立端點政策，以限制僅能存取特定的 DynamoDB 資料表。如果您在 Amazon VPC AWS 服務 中有使用資料表的其他 ，這種類型的政策很有用。下列資料表政策僅允許存取 *DOC-EXAMPLE-TABLE*。若要使用此端點政策，請將 *DOC-EXAMPLE-TABLE* 取代為您的資料表名稱。

JSON

```
{
"Version":"2012-10-17",
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-table-only",
      "Principal": "*",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:us-east-1:111122223333:table/DOC-EXAMPLE-
TABLE",
                   "arn:aws:dynamodb:us-east-1:111122223333:table/DOC-EXAMPLE-
TABLE/*"]
    }
  ]
}
```

搭配 AWS 管理主控台 私有存取使用 DynamoDB 端點

在 AWS 管理主控台 私有存取中使用 VPC 端點搭配 DynamoDB 主控台時，您必須設定 DynamoDB 與 DynamoDB Streams 的 DNS 組態。

若要將 DynamoDB 設定為可在 AWS 管理主控台 私有存取中存取，您必須建立下列兩個 VPC 端點：

- `com.amazonaws.<region>.dynamodb`
- `com.amazonaws.<region>.dynamodb-streams`

建立 VPC 端點時，請前往 Route 53 主控台，使用區域端點 `dynamodb.us-east-1.amazonaws.com` 為 DynamoDB 建立私有託管區域。

在私有託管區域中建立以下兩筆別名記錄：

- `dynamodb.<region>.amazonaws.com`：將流量導向至 VPC 端點 `com.amazonaws.<region>.dynamodb`。
- `streams.dynamodb.<region>.amazonaws.com`：將流量導向至 VPC 端點 `com.amazonaws.<region>.dynamodb-streams`。

AWS PrivateLink 適用於 DynamoDB Streams

使用 AWS PrivateLink for Amazon DynamoDB Streams，您可以在虛擬私有雲端 (Amazon VPC) 中佈建介面 Amazon VPC 端點（介面端點）。這些端點可直接透過 VPN 和內部部署的應用程式存取 Direct Connect，或 AWS 區域 是透過 Amazon VPC 對等互連的不同應用程式存取。使用 AWS PrivateLink 和 介面端點，您可以簡化從應用程式到 DynamoDB Streams 的私有網路連線。

位於 Amazon VPC 中的應用程式無需公有 IP 位址，即可透過 Amazon VPC 介面端點執行 DynamoDB Streams 操作。介面端點由一或多個彈性網路介面 (ENI) 所組成，這些介面會從 Amazon VPC 的子網路中指派私有 IP 位址。透過介面端點傳送至 DynamoDB Streams 的請求，會保留在 Amazon 網路中。您也可以透過 Direct Connect 或 AWS Virtual Private Network (AWS VPN)，從內部部署應用程式存取 Amazon VPC 中的介面端點。如需如何將 AWS Virtual Private Network 連線至內部部署網路的詳細資訊，請參閱 Direct Connect 使用者指南和 AWS Site-to-Site VPN 使用者指南。

如需介面端點的一般資訊，請參閱 Amazon VPC 介面端點 (AWS PrivateLink)。

> **Note**
>
> DynamoDB Streams 僅支援介面端點。不支援閘道端點。

主題

- 使用 AWS PrivateLink for Amazon DynamoDB Streams 時的考量事項
- 建立 Amazon VPC 端點
- 存取 Amazon DynamoDB Streams 介面端點
- 從 DynamoDB Streams 介面端點存取 DynamoDB Streams API 操作
- AWS 開發套件範例
- 為 DynamoDB Streams 建立 Amazon VPC 端點政策
- 搭配 AWS 管理主控台 私有存取使用 DynamoDB 端點

使用 AWS PrivateLink for Amazon DynamoDB Streams 時的考量事項

Amazon VPC 考量適用於 AWS PrivateLink Amazon DynamoDB Streams。如需詳細資訊，請參閱介面端點考量事項與 AWS PrivateLink 配額。適用以下限制。

AWS PrivateLink for Amazon DynamoDB Streams 不支援下列項目：

- Transport Layer Security (TLS) 1.1
- 私有與混合網域名稱系統 (DNS) 服務

> **Important**
>
> 請勿建立私有託管區域來覆寫 DynamoDB Streams 端點 DNS 名稱，以將流量路由到您的介面端點。DynamoDB DNS 組態可能會隨著時間而變更，而自訂 DNS 覆寫可能會導致請求意外透過公有 IP 地址路由，而不是您的介面端點。
> 若要透過 存取 DynamoDB Streams AWS PrivateLink，請將您的用戶端設定為直接使用 Amazon VPC 端點 URL（例如 `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.region.vpce.amazonaws.com`）。

> **Note**
>
> AWS PrivateLink 端點的網路連線逾時不在 DynamoDB Streams 錯誤回應範圍內，需要由連線至 AWS PrivateLink 端點的應用程式妥善處理。

建立 Amazon VPC 端點

若要建立 Amazon VPC 介面端點，請參閱 AWS PrivateLink 指南中的建立 Amazon VPC 端點。

存取 Amazon DynamoDB Streams 介面端點

建立介面端點時，DynamoDB 會產生兩種類型的專用 DynamoDB Streams DNS 名稱：區域和可用區。

- 區域 DNS 名稱在其名稱`vpce.amazonaws.com`中包含唯一的 Amazon VPC 端點 ID、服務識別符 AWS 區域、 和 。例如，對於 Amazon VPC 端點 ID *`vpce-1a2b3c4d`*，產生的 DNS 名稱可能類似於 *`vpce-1a2b3c4d-5e6f`*`.streams.dynamodb.us-east-1.vpce.amazonaws.com`。

- 地區 DNS 名稱包含可用區域，例如 *vpce-1a2b3c4d-5e6f*-**us-east-1a.**streams.dynamodb.us-east-1.vpce.amazonaws.com。如果您的架構可隔離可用區域，則可以使用此選項。例如，您可以將其用於故障遏止或降低區域資料傳輸成本。

從 DynamoDB Streams 介面端點存取 DynamoDB Streams API 操作

您可以使用 AWS CLI AWS SDKs 透過 DynamoDB Streams 介面端點存取 DynamoDB Streams API 操作。

AWS CLI 範例

若要透過 AWS CLI 命令中的 DynamoDB Streams 介面端點存取 DynamoDB Streams 或 API 操作，請使用 `--region`和 `--endpoint-url` 參數。

範例：建立 VPC 端點

```
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
```

範例：修改 VPC 端點

```
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
```

範例：使用端點 URL 列出串流

在下列範例中，請以您自己的資訊取代區域 `us-east-1` 和 VPC 端點 ID `vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` 的 DNS 名稱。

```
aws dynamodbstreams --region us-east-1 –endpoint https://
vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com list-streams
```

AWS 開發套件範例

若要在使用 AWS SDK 時，透過 DynamoDB Streams 介面端點存取 Amazon DynamoDB Streams API 操作，請將 SDK 更新至最新版本。然後，將用戶端設定為透過 DynamoDB Streams 介面端點，使用 DynamoDB Streams API 操作的端點 URL。

SDK for Python (Boto3)

範例：使用端點 URL 存取 DynamoDB 串流

在下列範例中，將區域 `us-east-1` 和 VPC 端點 ID `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` 取代為您的資訊。

```
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

範例：使用端點 URL 存取 DynamoDB 串流

在下列範例中，將區域 `us-east-1` 和 VPC 端點 ID `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` 取代為您的資訊。

```
//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

範例：使用端點 URL 存取 DynamoDB 串流

在下列範例中，將區域 us-east-1 和 VPC 端點 ID https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com 取代為您的資訊。

```
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()
```

為 DynamoDB Streams 建立 Amazon VPC 端點政策

您可以將端點政策附加至 Amazon VPC 端點，以控制對 DynamoDB Streams 的存取。此政策會指定下列資訊：

- 可執行動作的 AWS Identity and Access Management (IAM) 委託人
- 可執行的動作
- 可在其中執行動作的資源

主題

- 範例：限制從 Amazon VPC 端點存取特定串流

範例：限制從 Amazon VPC 端點存取特定串流

您可以建立端點政策，以限制僅存取特定的 DynamoDB Streams。如果您的 Amazon VPC AWS 服務 中有使用 DynamoDB Streams 的其他 ，這種類型的政策很有用。下列串流政策限制僅能存取附加至 *DOC-EXAMPLE-TABLE* 的串流 *2025-02-20T11:22:33.444*。若要使用此端點政策，請將 *DOC-EXAMPLE-TABLE* 取代為資料表名稱，並將 *2025-02-20T11:22:33.444* 取代為串流標籤。

JSON

```
{
"Version":"2012-10-17",
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-stream-only",
      "Principal": "*",
```

```
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:us-east-1:111122223333:table/table-name/
stream/2025-02-20T11:22:33.444"]
    }
  ]
}
```

> **Note**
>
> DynamoDB Streams 不支援閘道端點。

搭配 AWS 管理主控台 私有存取使用 DynamoDB 端點

在 AWS 管理主控台 私有存取中使用 VPC 端點搭配 DynamoDB 主控台時，您必須設定 DynamoDB 與 DynamoDB Streams 的 DNS 組態。

若要將 DynamoDB 設定為可在 AWS 管理主控台 私有存取中存取，您必須建立下列兩個 VPC 端點：

- `com.amazonaws.<region>.dynamodb`
- `com.amazonaws.<region>.dynamodb-streams`

建立 VPC 端點時，請前往 Route 53 主控台，使用區域端點 `dynamodb.us-east-1.amazonaws.com` 為 DynamoDB 建立私有託管區域。

在私有託管區域中建立以下兩筆別名記錄：

- `dynamodb.<region>.amazonaws.com`：將流量導向至 VPC 端點 `com.amazonaws.<region>.dynamodb`。
- `streams.dynamodb.<region>.amazonaws.com`：將流量導向至 VPC 端點 `com.amazonaws.<region>.dynamodb-streams`。

使用 AWS PrivateLink for DynamoDB Accelerator (DAX)

AWS PrivateLink for DynamoDB Accelerator (DAX) 可讓您`DeleteCluster`透過虛擬私有雲端 (VPC) 中的私有 IP 地址，安全地存取 DAX 管理 APIs`CreateCluster`，例如 `DescribeClusters`、 和 。此功能可讓您從應用程式私下存取 DAX 服務，而無需向公有網際網路公開流量。

DAX PrivateLink 支援雙堆疊端點 (`dax.{region}.api.aws`)，同時啟用 IPv4 和 IPv6 連線。使用 AWS PrivateLink for DAX，客戶可以使用私有 DNS 名稱存取服務。雙堆疊端點支援可確保透明連線，同時維護網路隱私權。這可讓您透過公有網際網路和 VPC 端點存取 DAX，而無需對 SDK 組態進行任何變更。

使用 AWS PrivateLink for DynamoDB Accelerator (DAX) 時的考量事項

實作 AWS PrivateLink DynamoDB Accelerator (DAX) 時，必須考量幾項重要考量。

在您設定 DAX 的介面端點之前，請考慮下列事項：

- DAX 介面端點僅支援存取相同 內的 DAX APIs AWS 區域。您無法使用介面端點來存取其他 區域中的 DAX APIs。
- 若要 AWS 管理主控台 私下存取 以進行 DAX 管理，您可能需要為 `com.amazonaws.region.console`和相關服務等服務建立額外的 VPC 端點。
- 您需要支付建立和使用界面端點至 DAX 的費用。如需定價資訊，請參閱 AWS PrivateLink 定價。

AWS PrivateLink 如何使用 DAX

當您為 DAX 建立介面端點時：

1. AWS 在您為介面端點啟用的每個子網路中建立端點網路介面。
2. 這些是請求者管理的網路介面，可做為目的地為 DAX 之流量的進入點。
3. 然後，您可以透過 VPC 中的私有 IP 地址存取 DAX。
4. 此架構可讓您使用 VPC 安全群組來管理對端點的存取。
5. 應用程式可以透過 VPC 內各自的界面端點存取 DynamoDB 和 DAX，同時允許內部部署應用程式透過 Direct Connect 或 VPN 連接。
6. 這可在這兩個服務之間提供一致的連線模式、簡化架構，並透過將流量保持在 AWS 網路中來改善安全性。

建立 DAX 的界面端點

您可以使用 CloudFormation、 AWS 管理主控台 AWS SDK 或 AWS API 建立介面端點以連線至 DAX。

使用主控台建立 DAX 的介面端點

1. 導覽至 Amazon VPC 主控台，網址為 https：//https://console.aws.amazon.com/vpc/。
2. 在導覽窗格中選擇端點。
3. 選擇建立端點。
4. 針對服務類別，選擇 AWS 服務 ，針對服務名稱選擇 ，搜尋並選擇 `com.amazonaws.region.dax`。
5. 對於 VPC，選取您要從中存取 DAX 的 VPC，對於子網路，選取 AWS 將在其中建立端點網路介面的子網路。
6. 針對安全群組，選取或建立要與端點網路介面建立關聯的安全群組。
7. 對於政策，保留預設的完整存取或視需要自訂。
8. 選取啟用 DNS 名稱以啟用端點的私有 DNS。保持啟用私有 DNS 名稱，以防止 SDK 組態的變更。啟用時，您的應用程式可以繼續使用標準服務 DNS 名稱 （ 例如：`dax.region.amazonaws.com`)。 會在 VPC 中 AWS 建立私有託管區域，將此名稱解析為端點的私有 IP 地址。

   > **Note**
   >
   > 如有需要，請使用區域 DNS 名稱。不建議使用區域 DNS 名稱。此外，從 3 個或多個AZs 選取子網路，以確保透過 PrivateLink 獲得最大的可用性。

9. 選擇建立端點。

使用 建立 DAX 的介面端點 AWS CLI

使用 `create-vpc-endpoint`命令，並將 `vpc-endpoint-type` 參數設定為 ，`Interface`並將 `service-name` 參數設定為 `com.amazonaws.region.dax`。

```
aws ec2 create-vpc-endpoint \
    --vpc-id vpc-ec43eb89 \
    --vpc-endpoint-type Interface \
    --service-name com.amazonaws.us-east-1.dax \
```

```
    --subnet-ids subnet-abcd1234 subnet-1a2b3c4d \
    --security-group-ids sg-1a2b3c4d \
    --private-dns-enabled
```

其他資源

如需 AWS PrivateLink 和 VPC 端點的詳細資訊，請參閱下列資源：

- AWS PrivateLink 適用於 DynamoDB
- AWS PrivateLink 適用於 DynamoDB 串流
- 使用 將您的 VPC 連接到 服務 AWS PrivateLink
- 使用 簡化 DynamoDB 的私有連線 AWS PrivateLink
- AWS PrivateLink 白皮書

Amazon DynamoDB 中的組態與漏洞分析

AWS 處理基本安全任務，例如訪客作業系統 (OS) 和資料庫修補、防火牆組態和災難復原。這些程序已由適當的第三方進行檢閱並認證。如需詳細資訊，請參閱以下資源：

- Amazon DynamoDB 的合規驗證
- 共同責任模型
- Amazon Web Services：安全程序概觀 (白皮書)

以下安全最佳實務也可處理 Amazon DynamoDB 中的組態和漏洞分析：

- 使用 監控 DynamoDB 合規 AWS Config 規則
- 使用 監控 DynamoDB 組態 AWS Config

Amazon DynamoDB 的安全最佳實務

在您開發和實作自己的安全政策時，可考慮使用 Amazon DynamoDB 提供的多種安全功能。以下最佳實務為一般準則，並不代表完整的安全解決方案。這些最佳實務可能不適用或無法滿足您的環境需求，因此請將其視為實用建議就好，而不要當作是指示。

主題

- DynamoDB 預防性安全最佳實務

- DynamoDB 偵測性安全最佳實務

DynamoDB 預防性安全最佳實務

下列最佳實務有助於預測並預防 Amazon DynamoDB 中的安全性事件。

靜態加密

DynamoDB 使用儲存在 AWS Key Management Service (AWS KMS) 中的加密金鑰，為資料表、索引、串流和備份中的所有使用者資料進行靜態加密。如此可透過保護您的資料免於發生未經授權的基礎儲存體存取，為資料提供另一層保護。

您可以指定 DynamoDB 是否應使用 AWS 擁有的金鑰 （預設加密類型） AWS 受管金鑰、 或客戶受管金鑰來加密使用者資料。如需詳細資訊，請參閱 Amazon DynamoDB 靜態加密。

使用 IAM 角色驗證對 DynamoDB 的存取權

對於存取 DynamoDB 的使用者、應用程式和其他 AWS 服務，他們必須在其 AWS API 請求中包含有效的 AWS 登入資料。您不應將 AWS 登入資料直接存放在應用程式或 EC2 執行個體中。這些是不會自動輪換的長期憑證，因此如果遭到盜用，可能會對業務造成嚴重的影響。IAM 角色可讓您取得可用來存取 AWS 服務和資源的臨時存取金鑰。

如需詳細資訊，請參閱Amazon DynamoDB 的 Identity and Access Management。

使用 IAM 政策進行 DynamoDB 基礎授權

授予許可時，會決定誰可以取得這些許可、可以取得哪些 DynamoDB API 的許可，以及可以對這些資源進行的特定動作。對降低錯誤或惡意意圖所引起的安全風險和影響而言，實作最低權限是其中關鍵。

將許可政策連接至 IAM 身分 (即使用者、群組和角色)，藉此授予可在 DynamoDB 資源上執行操作的許可。

您可以使用下列內容執行這項作業：

- AWS 受管 （預先定義） 政策
- 客戶管理政策

使用 IAM 政策條件進行精細定義存取控制

您在 DynamoDB 中授予許可時，可以指定條件，以決定許可政策的生效方式。對降低錯誤或惡意意圖所引起的安全風險和影響而言，實作最低權限是其中關鍵。

您可以指定使用 IAM 政策授予許可時的條件。例如，您可以執行下列動作：

- 授予許可，允許使用者唯讀存取資料表或次要索引中的特定項目和屬性。
- 授予許可，允許使用者根據該使用者的身分唯寫存取資料表中的特定屬性。

如需詳細資訊，請參閱使用 IAM 政策條件精細定義存取控制。

使用 VPC 端點和政策來存取 DynamoDB

如果只需要從虛擬私有雲端 (VPC) 內存取 DynamoDB，則應使用 VPC 端點來限制僅來自所需 VPC 的存取。這樣做可防止流量周遊開放式網際網路，也可防止流量受到該環境的限制。

使用 DynamoDB 的 VPC 端點可讓您使用下列內容來控制和限制存取：

- VPC 端點政策：這些政策會套用在 DynamoDB VPC 端點上。它們可讓您控制和限制 API 對 DynamoDB 資料表的存取權限。
- IAM 政策：利用使用連接至使用者、群組或角色的 `aws:sourceVpce` 條件，您可以強制要求對 DynamoDB 資料表進行的所有存取都要透過指定的 VPC 端點。

如需詳細資訊，請參閱 Amazon DynamoDB 端點。

考慮用戶端加密

我們建議您先規劃加密策略，再於 DynamoDB 中使用資料表。如果您將敏感或機密資料儲存在 DynamoDB，請考慮在計劃中加入用戶端加密。如此一來，您就能盡量靠近資料來源進行加密，並確保資料在整個生命週期受到保護。將您傳輸中和靜態的敏感資料加密，有助於確保您的明文資料不會被任何第三方取得。

AWS Database Encryption SDK for DynamoDB 是一個軟體程式庫，可協助您保護在將資料表資料傳送至 DynamoDB 之前的安全性。其會加密、簽署、驗證和解密 DynamoDB 資料表項目。您可以控制要加密和簽署的屬性。

主索引鍵設計考量

請勿在資料表與全域次要索引的主索引鍵中使用敏感名稱或純文字資料。主索引鍵名稱會顯示於資料表定義中。例如，任何擁有呼叫 DescribeTable 權限的人都能查看主索引鍵名稱。主索引鍵值可能出現在 AWS CloudTrail 及其他日誌中。此外，DynamoDB 會使用金鑰值來分佈資料和路由請求， AWS 管理員可能會觀察這些值來維護服務的運作狀態。

若需在資料表或 GSI 主索引鍵值中使用敏感資料，建議採用端到端用戶端加密。這可讓您執行鍵值查詢，同時確保資料在任何與 DynamoDB 相關的日誌中都不會以未加密形式出現。一種可行方法是使用 AWS Database Encryption SDK for DynamoDB，但這並非必要。若使用自訂解決方案，應一律採用足夠安全的加密演算法。請勿使用雜湊等非加密選項，因其在多數情境下安全性不足。

若主索引鍵名稱包含敏感資訊，建議改用 `pk` 和 `sk`。這是通用的最佳實務，可讓您的分割區索引鍵設計保持彈性。

如果您擔心什麼是正確的選擇，請務必諮詢您的安全專家或 AWS 客戶團隊。

DynamoDB 偵測性安全最佳實務

下列 Amazon DynamoDB 最佳實務能幫助您偵測潛在安全弱點與事件。

使用 AWS CloudTrail 監控 AWS 受管 KMS 金鑰用量

如果您使用 AWS 受管金鑰進行靜態加密，則會登入此金鑰的使用 AWS CloudTrail。CloudTrail 透過記錄對您帳戶所採取的動作，來提供使用者活動的可見性。CloudTrail 會記錄每個動作的重要資訊，包括提出請求的人員、使用的服務、執行的動作、動作的參數，以及 AWS 服務傳回的回應元素。此資訊可協助您追蹤對 AWS 資源所做的變更，並對操作問題進行疑難排解。CloudTrail 可讓您更輕鬆地確保內部政策和法規標準的合規性。

您可以使用 CloudTrail 來稽核金鑰的使用情況。CloudTrail 會建立日誌檔案，其中包含您帳戶的 AWS API 呼叫和相關事件的歷史記錄。這些日誌檔案包括使用 AWS 管理主控台、 AWS SDKs和命令列工具提出的所有 AWS KMS API 請求，以及透過整合 AWS 服務提出的請求。您可以使用這些日誌檔來取得使用 KMS 金鑰的時間、所請求的操作、請求者的身分、請求的來源 IP 位址等資訊。如需詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的使用 AWS CloudTrail記錄 AWS KMS API 呼叫。

使用 CloudTrail 監控 DynamoDB 操作

CloudTrail 可以監控控制平面事件和資料平面事件。控制平面操作可讓您建立及管理 DynamoDB 資料表。它們也可讓您使用索引、串流，以及相依於資料表的其他物件。資料平面操作可讓您對資料表中的資料執行建立、讀取、更新與刪除 (也稱為 CRUD) 動作。某些資料平面操作也可讓您從次要索引中讀取資料。若要在 CloudTrail 中啟用資料平面事件的記錄功能，您需要在 CloudTrail 中啟用資料平面 API 活動的記錄功能。如需詳細資訊，請參閱記錄追蹤的資料事件。

當 DynamoDB 中發生活動時，該活動會與事件歷史記錄中的其他 AWS 服務事件一起記錄在 CloudTrail 事件中。如需詳細資訊，請參閱使用 AWS CloudTrail來記錄 DynamoDB 操作。您可以在 AWS 帳戶中檢視、搜尋和下載最近的事件。如需詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的使用 CloudTrail 事件歷史記錄檢視事件。

若要持續記錄您 AWS 帳戶中的事件，包括 DynamoDB 的事件，請建立追蹤。線索能讓 CloudTrail 將日誌檔案交付至 Amazon Simple Storage Service (Amazon S3) 儲存貯體。根據預設，當您在主

控台上建立線索時，線索會套用至所有 AWS 區域。該追蹤會記錄來自 AWS 分割區中所有區域的事件，並將日誌檔案交付到您指定的 S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。

使用 DynamoDB Streams 監控資料平面操作

DynamoDB 與 整合 AWS Lambda ，因此您可以建立觸發，也就是自動回應 DynamoDB Streams 中事件的程式碼片段。您可以利用觸發條件建立對 DynamoDB 資料表資料修改做出反應的應用程式。

如果您在資料表中啟用 DynamoDB Streams，就可以建立串流 Amazon Resource Name (ARN) 與您撰寫之 Lambda 函式的關聯。修改資料表中的項目後，資料表的 stream. AWS Lambda polls 會立即顯示新記錄，並在偵測到新的串流記錄時同步叫用 Lambda 函數。Lambda 函式可以執行您指定的任何動作，例如傳送通知或啟動工作流程。

如需範例，請參閱教學課程：搭配 Amazon DynamoDB Streams 使用 AWS Lambda。此範例會接收 DynamoDB 事件輸入、處理其包含的訊息，並將部分傳入事件資料寫入 Amazon CloudWatch Logs。

使用 監控 DynamoDB 組態 AWS Config

您可以使用 AWS Config 持續監控並記錄 AWS 資源的變更。您也可以使用 AWS Config 清查您的 AWS 資源。當偵測到先前狀態的變更時，Amazon Simple Notification Service (Amazon SNS) 會通知您檢閱並採取行動。遵循AWS Config 使用主控台設定中的指示，確保包含 DynamoDB 資源類型。

您可以設定 AWS Config 將組態變更和通知串流至 Amazon SNS 主題。例如，更新資源時，您可以收到傳送至您電子郵件的通知，因此您可以檢視變更。當 根據您的資源 AWS Config 評估您的自訂或受管規則時，您也會收到通知。

如需範例，請參閱《 AWS Config 開發人員指南》中的AWS Config 傳送至 Amazon SNS 主題的通知。

監控 DynamoDB 是否符合 AWS Config 規則

AWS Config 會持續追蹤 資源中發生的組態變更。其會檢查這些變更是否違反您規則中的任何條件。如果資源違反規則， 會將資源和規則 AWS Config 標記為不合規。

透過使用 AWS Config 評估您的資源組態，您可以評估資源組態符合內部實務、產業準則和法規的程度。 AWS Config 提供AWS 受管規則，這些是預先定義的可自訂規則， AWS Config 可用來評估您的 AWS 資源是否符合常見的最佳實務。

為您的 DynamoDB 資源加上標籤，以便進行識別和自動化

您可以將中繼資料以標籤形式指派給 AWS 資源。每個標記都是由客戶定義金鑰和選用值組成的簡單標籤，能夠更輕鬆地管理、搜尋和篩選資源。

標記允許實現分組控制。雖然標籤不具固有類型，但能讓您依用途、擁有者、環境或其他條件分類資源。下列是一些範例：

- 安全性：用於確定加密等需求。
- 機密性：資源支援的特定資料機密等級識別符。
- 環境：用來區分開發、測試和生產基礎設施。

如需詳細資訊，請參閱 AWS 標記策略和 DynamoDB 的標記。

監控 Amazon DynamoDB 的使用量，因為它與使用 的安全最佳實務相關 AWS Security Hub CSPM。

Security Hub CSPM 使用安全控制來評估資源組態和安全標準，以協助您符合各種合規架構。

如需使用 Security Hub CSPM 評估 DynamoDB 資源的詳細資訊，請參閱AWS Security Hub CSPM 《 使用者指南》中的 Amazon DynamoDB 控制項。

DynamoDB 中的記錄和監控

監控是維護 DynamoDB 及您 AWS 解決方案之可靠性、可用性和效能的重要部分。您應該收集 AWS 解決方案中所有部分的監控資料，以便在出現多點故障時更輕鬆地偵錯。

主題

- 監控計畫
- 效能基準
- 整合服務
- 自動化監控工具
- 使用 Amazon CloudWatch 來監控 DynamoDB 指標
- 使用 記錄 DynamoDB 操作 AWS CloudTrail
- 使用 DynamoDB 專用 CloudWatch Contributor Insights 分析資料存取

監控計畫

在開始監控 DynamoDB 之前，應先建立監控計畫，其中包括下列問題的答案：

- 監控目標是什麼？
- 要監控哪些資源？
- 監控這些資源的頻率為何？
- 要使用哪些監控工具？
- 誰將執行監控任務？
- 發生問題時應該通知誰？

效能基準

下一步是在各個時間點和不同的負載條件下測量效能，以便在您的環境中確立 DynamoDB 正常效能的基準。當您監控 DynamoDB 時，應該考慮存放歷史監控資料。這個存放的資料會提供基準，讓您與目前的效能資料比較，識別出正常的效能模式和效能異常狀況，再規劃方式來處理問題。若要建立基準，您至少必須監控下列項目：

- 在指定時段使用的讀取或寫入容量單位數目，讓您可追蹤已使用多少佈建的輸送量。

- 在指定時段內超出資料表佈建寫入容量或佈建讀取容量的請求，讓您可判斷哪些請求超出資料表的佈建輸送量配額。
- 系統錯誤，可讓您判斷是否有任何請求導致錯誤。

整合服務

DynamoDB 代表您自動監視資料表，並透過 Amazon CloudWatch 回報指標。此外，DynamoDB 會與下列 AWS 服務 整合，以協助您監控 DynamoDB 資源並進行疑難排解。

- AWS CloudTrail 擷取您 AWS 帳戶 發出或代表發出的 API 呼叫和相關事件，並傳送日誌檔案至您指定的 Amazon S3 儲存貯體。如需更多詳細資訊，請參閱 使用 記錄 DynamoDB 操作 AWS CloudTrail。
- Contributor Insights 為分析工具，可快速找到您資料表或索引中最常存取和被限流的索引鍵。如需更多詳細資訊，請參閱 使用 DynamoDB 專用 CloudWatch Contributor Insights 分析資料存取。

自動化監控工具

AWS 提供可讓您用來監控 DynamoDB 的工具。建議您盡可能自動化監控任務。您可以使用下列自動化監控工具來監看 DynamoDB，並在發生錯誤時進行回報：

- Amazon CloudWatch 警示：監看指定時段內的單一指標，並根據與多個時段內給定之閾值相對的指標值來執行一或多個動作。

  此動作是傳送到 Amazon Simple Notification Service (Amazon SNS) 主題或 Amazon EC2 Auto Scaling 政策的通知。Amazon CloudWatch 警示不會只因處於特定狀態就調用動作，狀態必須已變更並已維持一段指定的時間。如需更多詳細資訊，請參閱 使用 Amazon CloudWatch 來監控 DynamoDB 指標。
- AWS CloudTrail 日誌監控：在帳戶之間共享日誌檔案、將 AWS CloudTrail 日誌檔案傳送至 AWS CloudTrail 日誌，以對其進行即時監控、以 Java 撰寫日誌處理應用程式，以及驗證日誌檔案在由 AWS CloudTrail 交付之後沒有發生變更。如需更多詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的什麼是 Amazon CloudWatch Logs。

使用 Amazon CloudWatch 來監控 DynamoDB 指標

您可以使用 CloudWatch 來監控 DynamoDB；該服務會收集並處理來自 DynamoDB 的原始資料，進而將這些資料轉換為便於讀取且幾近即時的指標。這些統計數字會保留一段時間，讓您可以存取歷史

資訊，以更清楚 Web 應用程式或服務的執行效能。根據預設，系統會自動將 DynamoDB 指標資料傳送至 CloudWatch。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的什麼是 Amazon CloudWatch？以及指標保留。

主題

- 如何使用 DynamoDB 指標？
- 在 CloudWatch 主控台中檢視指標
- 在 中檢視指標 AWS CLI
- DynamoDB 指標和維度
- 在 DynamoDB 中建立 CloudWatch 警示

如何使用 DynamoDB 指標？

DynamoDB 回報的指標可提供能讓您以不同方式分析的資訊。下列清單顯示一些常見的指標用途。這些是協助您開始的建議，而不是完整清單。

如何使用 DynamoDB 指標？

如何？	相關指標
如何監控資料表上的 TTL 刪除率？	您可以監控指定時段內的 `TimeToLiveDeletedItemCount` ，以追蹤您資料表的 TTL 刪除率。如需使用 `TimeToLiveDeletedItemCount` 指標的無伺服器應用程式範例，請參閱使用 DynamoDB 存留時間 (TTL) 搭配 AWS Lambda 和 Amazon Data Firehose 自動將項目封存至 S3。
如何判斷使用了多少佈建輸送量？	您可以監控指定時段內的 `ConsumedReadCapacityUnits` 或 `ConsumedWriteCapacityUnits` ，以追蹤佈建輸送量的使用數量。
如果判斷哪些請求超過資料表的佈建輸送量配額？	如果請求內的任何事件超過佈建輸送量的配額，則 `ThrottledRequests` 會增加 1。然後，若要深入了解哪些事件限流了請求，請將 `ThrottledRequests` 與資料表和其索引的 `ReadThrottleEvents` 與 `WriteThrottleEvents` 指標比較。

如何？	相關指標
如何判斷是否發生任何系統錯誤？	您可以監控 `SystemErrors` ，以判斷是否有任何請求導致 HTTP 500 (伺服器錯誤) 代碼。這項指標通常應該等於零。否則建議您予以調查。
如何監控資料表操作的延遲值？	您可以透過百分比指標 (p50) 追蹤平均延遲和中位數延遲來監控 `SuccessfulRequestLatency` 。偶爾的延遲峰值無需擔心。不過，如果平均延遲或 p50 (中位數) 很高，則可能有您必須解決的基礎問題。如需詳細資訊，請參閱對 Amazon DynamoDB 中的延遲問題進行疑難排解。

在 CloudWatch 主控台中檢視指標

指標會先依服務命名空間分組，再依各命名空間內不同的維度組合分類。

若要在 CloudWatch 主控台中檢視指標

1. 在 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在導覽窗格中，選擇指標，所有指標。
3. 選取 DynamoDB 命名空間。您也可以選擇用途命名空間來查看 DynamoDB 使用量指標。如需使用量指標的詳細資訊，請參閱AWS 使用量指標。
4. Browse (瀏覽) 索引標籤會顯示命名空間中該維度的所有指標。
5. (選用) 若要將指標圖表新增至 CloudWatch 儀表板，請選擇 Actions, Add to dashboard (動作，新增至儀表板)。

在 中檢視指標 AWS CLI

若要使用 取得指標資訊 AWS CLI，請使用 CloudWatch 命令 `list-metrics`。以下範例會列出 `AWS/DynamoDB` 命名空間中的所有指標。

```
aws cloudwatch list-metrics --namespace "AWS/DynamoDB"
```

若要取得指標統計數字，請使用 get-metric-statistics 命令。以下命令會取得資料表 ProductCatalog 在指定 24 小時期間的 ConsumedReadCapacityUnits 統計資料，精細程度為 5 分鐘。

```
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
```

範例輸出如下所示：

```
{
    "Datapoints": [
        {
            "Timestamp": "2023-11-01T 09:18:00+00:00",
            "Average": 20,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 04:36:00+00:00",
            "Average": 22.5,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 15:12:00+00:00",
            "Average": 20,
            "Unit": "Count"
        }, ...
        {
            "Timestamp": "2023-11-01T 17:30:00+00:00",
            "Average": 25,
            "Unit": "Count"
        }
    ],
    "Label": " ConsumedReadCapacityUnits "
}
```

DynamoDB 指標和維度

當您與 DynamoDB 互動時，其會將指標與維度傳送至 CloudWatch。

DynamoDB 會以一分鐘為週期，輸出耗用的佈建輸送量。當耗用容量連續兩分鐘超過設定的目標使用率時，自動擴展即會觸發。CloudWatch 警示在觸發自動擴展前，可能會有短暫數分鐘的延遲。此延遲可確保 CloudWatch 指標的準確評估。若耗用輸送量峰值間隔超過一分鐘，自動擴展可能不會觸發。同理，當連續 15 個資料點低於目標使用率時，會觸發縮減規模事件。無論哪種情況，自動擴展觸發後，會調用 UpdateTable API。接著，系統可能需要數分鐘時間來更新資料表或索引的佈建容量。在此期間，所有超出資料表先前佈建容量的請求都會受到限流。

檢視指標和維度

CloudWatch 顯示下列 DynamoDB 指標：

DynamoDB 指標

> **Note**
>
> Amazon CloudWatch 以一分鐘為間隔彙總這些指標：
>
> - ConditionalCheckFailedRequests
> - ConsumedReadCapacityUnits
> - ConsumedWriteCapacityUnits
> - ReadAccountLimitThrottleEvents
> - ReadKeyRangeThroughputThrottleEvents
> - ReadMaxOnDemandThroughputThrottleEvents
> - ReadProvisionedThroughputThrottleEvents
> - ReadThrottleEvents
> - ReturnedBytes
> - ReturnedItemCount
> - ReturnedRecordsCount
> - SuccessfulRequestLatency
> - SystemErrors
> - TimeToLiveDeletedItemCount
> - ThrottledRequests

- TransactionConflict
- UserErrors
- WriteAccountLimitThrottleEvents
- WriteKeyRangeThroughputThrottleEvents
- WriteMaxOnDemandThroughputThrottleEvents
- WriteProvisionedThroughputThrottleEvents
- WriteThrottleEvents
- FaultInjectionServiceInducedErrors

對於所有其他 DynamoDB 指標，彙總的間隔為五分鐘。

並非所有統計數字，例如 Average 或 Sum，皆適用於所有指標。不過，所有這些值都可以透過 Amazon DynamoDB 主控台，或使用 CloudWatch 主控台、 AWS CLI或 AWS SDKs來取得所有指標。

在以下列表中，每個指標皆有適用於該指標的一組有效統計數字。

可用指標的列表

- AccountMaxReads
- AccountMaxTableLevelReads
- AccountMaxTableLevelWrites
- AccountMaxWrites
- AccountProvisionedReadCapacityUtilization
- AccountProvisionedWriteCapacityUtilization
- AgeOfOldestUnreplicatedRecord
- ConditionalCheckFailedRequests
- ConsumedChangeDataCaptureUnits
- ConsumedReadCapacityUnits
- ConsumedWriteCapacityUnits
- FailedToReplicateRecordCount
- MaxProvisionedTableReadCapacityUtilization
- MaxProvisionedTableWriteCapacityUtilization

- OnDemandMaxReadRequestUnits
- OnDemandMaxWriteRequestUnits
- OnlineIndexConsumedWriteCapacity
- OnlineIndexPercentageProgress
- OnlineIndexThrottleEvents
- PendingReplicationCount
- ProvisionedReadCapacityUnits
- ProvisionedWriteCapacityUnits
- ReadAccountLimitThrottleEvents
- ReadKeyRangeThroughputThrottleEvents
- ReadMaxOnDemandThroughputThrottleEvents
- ReadProvisionedThroughputThrottleEvents
- ReadThrottleEvents
- ReplicationLatency
- ReturnedBytes
- ReturnedItemCount
- ReturnedRecordsCount
- SuccessfulRequestLatency
- SystemErrors
- TimeToLiveDeletedItemCount
- ThrottledPutRecordCount
- ThrottledRequests
- TransactionConflict
- UserErrors
- WriteAccountLimitThrottleEvents
- WriteKeyRangeThroughputThrottleEvents
- WriteMaxOnDemandThroughputThrottleEvents
- WriteProvisionedThroughputThrottleEvents
- WriteThrottleEvents
- 用量指標

- FaultInjectionServiceInducedErrors

AccountMaxReads

帳戶可以使用的讀取容量單位數目上限。此限制不適用於隨需資料表或全域次要索引。

單位：Count

有效的統計數字：

- Maximum：帳戶可以使用的讀取容量單位數目上限。

AccountMaxTableLevelReads

帳戶的資料表或全域次要索引可以使用的讀取容量單位數目上限。若是隨需資料表，此限制會限制資料表或全域次要索引可以使用的讀取請求單位上限。

單位：Count

有效的統計數字：

- Maximum：帳戶的資料表或全域次要索引可以使用的讀取容量單位數目上限。

AccountMaxTableLevelWrites

帳戶的資料表或全域次要索引可以使用的寫入容量單位數目上限。若是隨需資料表，此限制會限制資料表或全域次要索引可以使用的寫入請求單位上限。

單位：Count

有效的統計數字：

- Maximum：帳戶的資料表或全域次要索引可以使用的寫入容量單位數目上限。

AccountMaxWrites

帳戶可以使用的寫入容量單位數目上限。此限制不適用於隨需資料表或全域次要索引。

單位：Count

有效的統計數字：

- Maximum：帳戶可以使用的寫入容量單位數目上限。

AccountProvisionedReadCapacityUtilization

帳戶可以使用的佈建讀取容量單元百分比。

單位：Percent

有效的統計數字：

- Maximum：帳戶使用的佈建讀取容量單元百分比上限。
- Minimum：帳戶使用的佈建讀取容量單元百分比下限。
- Average：帳戶使用的佈建讀取容量單元平均百分比。指標會每隔五分鐘發布一次。因此，如果您快速調整佈建的讀取容量單位，此統計數字可能無法反映真實的平均值。

AccountProvisionedWriteCapacityUtilization

帳戶使用的佈建寫入容量單元百分比。

單位：Percent

有效的統計數字：

- Maximum：帳戶使用的佈建寫入容量單元百分比上限。
- Minimum：帳戶使用的佈建寫入容量單元百分比下限。
- Average：帳戶使用的佈建寫入容量單元平均百分比。指標會每隔五分鐘發布一次。因此，如果您快速調整佈建的寫入容量單位，此統計數字可能無法反映真實的平均值。

AgeOfOldestUnreplicatedRecord

自尚未複製到 Kinesis Data Streams 的紀錄首次出現在 DynamoDB 資料表中以來經過的時間。

單位：Milliseconds

維度：TableName, DelegatedOperation

有效的統計數字：

- Maximum.
- Minimum.

- Average.

ConditionalCheckFailedRequests

執行條件式寫入的失敗嘗試次數。PutItem、UpdateItem 以及 DeleteItem 操作可讓您提供邏輯條件，該條件必須評估為 true，才能繼續操作。若此條件評估為 false，ConditionalCheckFailedRequests 會遞增 1。對於 PartiQL Update 和 Delete 陳述式 (其中提供了一個評估為 false 的邏輯條件)，ConditionalCheckFailedRequests 也會遞增 1。

> **Note**
>
> 失敗的條件式寫入會導致 HTTP 400 錯誤 (錯誤的請求)。這些事件會反映在 ConditionalCheckFailedRequests 指標而非 UserErrors 指標中。

單位：Count

維度：TableName

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount
- Sum

ConsumedChangeDataCaptureUnits

已耗用的變更資料擷取單位數目。

單位：Count

維度：TableName, DelegatedOperation

有效的統計數字：

- Minimum
- Maximum

- Average

ConsumedReadCapacityUnits

在指定時段內，使用的佈建與隨需讀取容量單位數目，可讓您追蹤已使用多少佈建的輸送量。您可以擷取資料表及其所有全域次要索引或特定全域次要索引的總消耗讀取容量。如需詳細資訊，請參閱讀取/寫入容量模式。

此 TableName 維度會針對資料表傳回 ConsumedReadCapacityUnits，但不會針對任何全域次要索引傳回。若要檢視全域次要索引的 ConsumedReadCapacityUnits，您必須指定 TableName 和 GlobalSecondaryIndexName。

> **Note**
>
> 這表示，容量使用當中僅持續一秒的短暫而密集的峰值，可能不會正確反映在 CloudWatch 圖表中，進而可能導致該分鐘的使用率明顯偏低。
> 使用 Sum 統計數字來計算消耗的輸送量。例如，取完整一分鐘的 Sum 值，再將該值除以一分鐘的秒數 (60)，得出每秒平均 ConsumedReadCapacityUnits。您可以將計算的數值與您提供給 DynamoDB 的佈建輸送量數值進行比較。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- Minimum：對資料表或索引的任何個別請求所消耗的讀取容量單位數目下限。
- Maximum：對資料表或索引的任何個別請求消耗的讀取容量單位數目上限。
- Average：每個請求消耗的平均讀取容量。

  > **Note**
  >
  > 此 Average 數值受樣本數值會為零的閒置時段所影響。

- Sum：所耗用的讀取容量單位總數。這是 ConsumedReadCapacityUnits 指標最實用的統計數字。
- SampleCount – 代表發出指標的頻率。即使是流量為零的資料表也會定期發出 SampleCount，但範例值一律為零。

> **Note**
>
> 此 SampleCount 數值受樣本數值會為零的閒置時段所影響。

ConsumedWriteCapacityUnits

在指定時段內，使用的佈建與隨需寫入容量單位數目，可讓您追蹤已使用多少佈建的輸送量。您可以擷取資料表及其所有全域次要索引或特定全域次要索引的消耗總計寫入容量。如需詳細資訊，請參閱讀取/寫入容量模式。

此 TableName 維度會針對資料表傳回 ConsumedWriteCapacityUnits，但不會針對任何全域次要索引傳回。若要檢視全域次要索引的 ConsumedWriteCapacityUnits，您必須指定 TableName 和 GlobalSecondaryIndexName。Source 維度可以傳回兩個值的其中之一： Customer和 GlobalTable。複寫的寫入將ConsumedWriteCapacityUnits具有來源 GlobalTable，但區域資料表寫入將ConsumedWriteCapacityUnits具有來源 Customer。

> **Note**
>
> 使用 Sum 統計數字來計算消耗的輸送量。例如，獲取一分鐘的 Sum 數值，然後除以一分鐘內的秒數 (60) 來計算每秒平均 ConsumedWriteCapacityUnits (請注意，這個平均值不會醒目提示該分鐘內發生的寫入活動中，任何大而短暫的峰值)。您可以將計算的數值與您提供給 DynamoDB 的佈建輸送量數值進行比較。

單位：Count

維度：TableName, GlobalSecondaryIndexName, Source

有效的統計數字：

- Minimum：對資料表或索引的任何個別請求消耗的寫入容量單位數目下限。
- Maximum：對資料表或索引的任何個別請求消耗的寫入容量單位數目上限。
- Average：每個請求消耗的平均寫入容量。

> **Note**
>
> 此 Average 數值受樣本數值會為零的閒置時段所影響。

- Sum：所耗用的寫入容量單位總數。這是 ConsumedWriteCapacityUnits 指標最實用的統計數字。
- SampleCount – 代表發出指標的頻率。即使是流量為零的資料表也會定期發出 SampleCount，但範例值一律為零。

> **Note**
>
> 此 SampleCount 數值受樣本數值會為零的閒置時段所影響。

FailedToReplicateRecordCount

DynamoDB 無法複製到您的 Kinesis 資料串流的記錄數目。

單位：Count

維度：TableName、DelegatedOperation

有效的統計數字：

- Sum

MaxProvisionedTableReadCapacityUtilization

帳戶最高佈建讀取資料表或全域次要索引所使用的佈建讀取容量單位百分比。

單位：Percent

有效的統計數字：

- Maximum – 帳戶最高佈建讀取資料表或全域次要索引所使用的佈建讀取容量單位最大百分比。
- Minimum – 帳戶最高佈建讀取資料表或全域次要索引所使用的佈建讀取容量單位最小百分比。
- Average – 帳戶最高佈建讀取資料表或全域次要索引所使用的平均佈建讀取容量單位百分比。指標會每隔五分鐘發布一次。因此，如果您快速調整佈建的讀取容量單位，此統計數字可能無法反映真實的平均值。

MaxProvisionedTableWriteCapacityUtilization

帳戶最高佈建寫入資料表或全域次要索引所使用的佈建寫入容量百分比。

單位：Percent

有效的統計數字：

- Maximum：帳戶最高佈建寫入資料表或全域次要索引所使用的佈建寫入容量單位百分比上限。
- Minimum：帳戶最高佈建寫入資料表或全域次要索引所使用的佈建寫入容量單位百分比下限。
- Average：帳戶最高佈建寫入資料表或全域次要索引所使用的平均佈建寫入容量單位百分比。指標會每隔五分鐘發布一次。因此，如果您快速調整佈建的寫入容量單位，此統計數字可能無法反映真實的平均值。

OnDemandMaxReadRequestUnits

資料表或全域次要索引的指定隨需讀取請求單位數目。

若要檢視資料表的 OnDemandMaxReadRequestUnits，您必須指定 TableName。若要檢視全域次要索引的 OnDemandMaxReadRequestUnits，您必須指定 TableName 和 GlobalSecondaryIndexName。

單位：Count

維度：TableName、GlobalSecondaryIndexName

有效的統計數字：

- Minimum – 隨需讀取請求單位的最低設定。如果使用 UpdateTable 增加讀取請求單位，此指標會顯示此時段內的隨需 ReadRequestUnits 最低數值。
- Maximum – 隨需讀取請求單位的最高設定。如果使用 UpdateTable 減少讀取請求單位，此指標會顯示此時段內的隨需 ReadRequestUnits 最高數值。
- Average – 平均隨需讀取請求單位。OnDemandMaxReadRequestUnits 指標會每隔五分鐘發布一次。因此，如果您快速調整隨需讀取請求單位，此統計數字可能無法反映真實的平均值。

OnDemandMaxWriteRequestUnits

資料表或全域次要索引的指定隨需寫入請求單位數目。

若要檢視資料表的 OnDemandMaxWriteRequestUnits，您必須指定 TableName。若要檢視全域次要索引的 OnDemandMaxWriteRequestUnits，您必須指定 TableName 和 GlobalSecondaryIndexName。

單位：Count

維度：TableName、GlobalSecondaryIndexName

有效的統計數字：

- Minimum – 隨需寫入請求單位的最低設定。如果使用 UpdateTable 增加寫入請求單位，此指標會顯示此時段內的隨需 WriteRequestUnits 最低數值。
- Maximum – 隨需寫入請求單位的最高設定。如果使用 UpdateTable 減少寫入請求單位，此指標會顯示此時段內的隨需 WriteRequestUnits 最高數值。
- Average – 平均隨需寫入請求單位。OnDemandMaxWriteRequestUnits 指標會每隔五分鐘發布一次。因此，如果您快速調整隨需寫入請求單位，此統計數字可能無法反映真實的平均值。

OnlineIndexConsumedWriteCapacity

此指標預期在索引建置期間會顯示 0。此指標先前顯示將新的全域次要索引新增至資料表時所使用的寫入容量單位數量。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount
- Sum

OnlineIndexPercentageProgress

新的全域次要索引新增至資料表時的完成百分比。DynamoDB 必須先為新索引分配資源，然後將資料表中的屬性回填至索引。對於大型資料表，此程序可能需要很長的時間。您應該監控此統計數字，以便在 DynamoDB 建置索引時檢視相對進度。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount
- Sum

OnlineIndexThrottleEvents

此指標預期在索引建置期間會顯示 0。此指標先前顯示將新的全域次要索引新增至資料表時發生的寫入調節事件數。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount
- Sum

PendingReplicationCount

全域資料表 2017.11.29 版 (舊版) 指標 (僅限全域資料表)。寫入一個複本列表，但尚未寫入全域資料表中的另一個複本的項目更新數目。

單位：Count

維度：TableName, ReceivingRegion

有效的統計數字：

- Average
- Sample Count

- Sum

ProvisionedReadCapacityUnits

資料表或全域次要索引佈建的讀取容量單位數目。此 TableName 維度會針對資料表傳回 ProvisionedReadCapacityUnits，但不會針對任何全域次要索引傳回。若要檢視全域次要索引的 ProvisionedReadCapacityUnits，您必須指定 TableName 和 GlobalSecondaryIndexName。

單位：Count

維度：TableName、GlobalSecondaryIndexName

有效的統計數字：

- Minimum：佈建讀取容量的最低設定。如果使用 UpdateTable 增加讀取容量，此指標會顯示在此時段內佈建的 ReadCapacityUnits 最低數值。
- Maximum：佈建讀取容量的最高設定。如果使用 UpdateTable 減少讀取容量，此指標會顯示在此時段內佈建的 ReadCapacityUnits 最高數值。
- Average：平均佈建讀取容量。ProvisionedReadCapacityUnits 指標會每隔五分鐘發布一次。因此，如果您快速調整佈建的讀取容量單位，此統計數字可能無法反映真實的平均值。

ProvisionedWriteCapacityUnits

資料表或全域次要索引佈建的寫入容量單位數目。

此 TableName 維度會針對資料表傳回 ProvisionedWriteCapacityUnits，但不會針對任何全域次要索引傳回。若要檢視全域次要索引的 ProvisionedWriteCapacityUnits，您必須指定 TableName 和 GlobalSecondaryIndexName。

單位：Count

維度：TableName、GlobalSecondaryIndexName

有效的統計數字：

- Minimum：佈建寫入容量的最低設定。如果使用 UpdateTable 增加寫入容量，此指標會顯示在此時段內佈建的 WriteCapacityUnits 最低數值。
- Maximum：佈建寫入容量的最高設定。如果使用 UpdateTable 減少寫入容量，此指標會顯示在此時段內佈建的 WriteCapacityUnits 最高數值。

- Average：佈建平均的寫入容量。ProvisionedWriteCapacityUnits 指標會每隔五分鐘發布一次。因此，如果您快速調整佈建的寫入容量單位，此統計數字可能無法反映真實的平均值。

ReadAccountLimitThrottleEvents

因帳戶限制而被限流的讀取請求數。

單位：Count

維度：TableName、GlobalSecondaryIndexName

有效的統計數字：

- Sum – 限流事件的總數。
- SampleCount – 限流發生次數。
- Minimum – 任何指定範例中的限流事件數目下限。
- Maximum – 任何指定範例中的限流事件數目上限。

ReadKeyRangeThroughputThrottleEvents

因分割區限制而被限流的讀取請求數。

單位：Count

維度：TableName、GlobalSecondaryIndexName

有效的統計數字：

- Sum – 限流事件的總數。
- SampleCount – 限流發生次數。
- Minimum – 任何指定範例中的限流事件數目下限。
- Maximum – 任何指定範例中的限流事件數目上限。

ReadMaxOnDemandThroughputThrottleEvents

因隨需模式達最大輸送量而被限流的讀取請求數。

單位：Count

維度：TableName、GlobalSecondaryIndexName

有效的統計數字：

- Sum – 限流事件的總數。
- SampleCount – 限流發生次數。
- Minimum – 任何指定範例中的限流事件數目下限。
- Maximum – 任何指定範例中的限流事件數目上限。

ReadProvisionedThroughputThrottleEvents

因佈建輸送量限制而被限流的讀取請求數。

單位：Count

維度：TableName、GlobalSecondaryIndexName

有效的統計數字：

- Sum – 限流事件的總數。
- SampleCount – 限流發生次數。
- Minimum – 任何指定範例中的限流事件數目下限。
- Maximum – 任何指定範例中的限流事件數目上限。

ReadThrottleEvents

對 DynamoDB 的請求超過資料表或全域次要索引的佈建讀取容量單位。

單一請求可能會導致多個事件。例如，BatchGetItem 讀取 10 個項目會作為 10 個 GetItem 事件處理。針對每個事件，如果該事件經限流，則 ReadThrottleEvents 會遞增 1。除非全部 10 個 GetItem 事件經過限流，否則整個 BatchGetItem 的 ThrottledRequests 指標不會遞增。

此 TableName 維度會針對資料表傳回 ReadThrottleEvents，但不會針對任何全域次要索引傳回。若要檢視全域次要索引的 ReadThrottleEvents，您必須指定 TableName 和 GlobalSecondaryIndexName。

單位：Count

維度：TableName、GlobalSecondaryIndexName

有效的統計數字：

- SampleCount
- Sum

ReplicationLatency

(此指標適用於 DynamoDB 全域資料表。) 在 DynamoDB Streams 中顯示一個複本列表的更新項目到該項目出現在全域資料表的另一個複本中的經過時間。

單位：Milliseconds

維度：TableName, ReceivingRegion

有效的統計數字：

- Average
- Minimum
- Maximum

ReturnedBytes

GetRecords 操作 (Amazon DynamoDB Streams) 在指定時段內傳回的位元組數目。

單位：Bytes

維度：Operation, StreamLabel, TableName

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount
- Sum

ReturnedItemCount

在指定時段內由 Query、Scan 或 ExecuteStatement (選取) 操作所傳回的項目數。

傳回的項目數量不一定與評估的項目數量相同。例如，假設您請求的是資料表上的 Scan 或具有 100 個項目的索引，但指定了縮小結果的 FilterExpression，故只返回 15 個項目。在這種情況下，來自 Scan 的回應會包含 100 個 ScanCount 和 15 個返回項目的 Count。

單位：Count

維度：TableName, Operation

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount
- Sum

ReturnedRecordsCount

GetRecords 操作 (Amazon DynamoDB Streams) 在指定時段內傳回的串流紀錄數目。

單位：Count

維度：Operation, StreamLabel, TableName

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount
- Sum

SuccessfulRequestLatency

在指定時段內對 DynamoDB 或 Amazon DynamoDB Streams 發起的成功請求延遲。SuccessfulRequestLatency 可以提供兩種不同類型的資訊：

- 成功請求的經過時間 (Minimum、Maximum、Sum、Average 或 Percentile)。

- 成功請求的數量 (SampleCount)。

SuccessfulRequestLatency 僅會反映 DynamoDB 或 Amazon DynamoDB Streams 中的活動，不會考慮網路延遲或用戶端活動。

> **Note**
>
> 若要分析自訂百分位數值 (例如 p99.9)，可在 CloudWatch 指標統計資料欄位中手動輸入所需百分位數。這可讓您評估超出下拉選單預設百分位數範圍的延遲分布。

單位：Milliseconds

維度：TableName, Operation, StreamLabel

有效的統計數字：

- Minimum
- Maximum
- Sum
- Average
- Percentile
- SampleCount

SystemErrors

對 DynamoDB 或 Amazon DynamoDB Streams 發起的能在指定時段內產生 HTTP 500 狀態碼的請求。HTTP 500 通常表示內部服務錯誤。

> **Note**
>
> 當 DynamoDB 傳回系統錯誤 (HTTP 500) 時，大多數 AWS SDKs 會自動執行可設定的重試次數。如果重試期間解決了問題，則應用程式會繼續，而不會顯示錯誤，但您可能會發現用戶端感知延遲增加。如果全部重試後仍存在錯誤，則該錯誤會傳播到應用程式的程式碼。

單位：Count

維度：TableName, Operation

有效的統計數字：

- Sum
- SampleCount

TimeToLiveDeletedItemCount

存留時間 (TTL) 在指定時段內刪除的項目數量。此指標可協助您監控資料表上的 TTL 刪除率。

單位：Count

維度：TableName

有效的統計數字：

- Sum

ThrottledPutRecordCount

因 Kinesis Data Streams 容量不足而受到限流的 Kinesis Data Streams 的紀錄數目。

單位：Count

維度：TableName、DelegatedOperation

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount

ThrottledRequests

對 DynamoDB 的請求超過資源 (例如資料表或索引) 的佈建輸送量限制。

如果請求內的任何事件超過佈建輸送量的限額，則 ThrottledRequests 會遞增 1。例如，如果使用全域次要索引更新資料表中的項目，則會有多個事件：寫入資料表和寫入每個索引。如果其中一或多個事件經過限流，則 ThrottledRequests 會遞增 1。

> **Note**
>
> 在批次處理請求中 (BatchGetItem 或 BatchWriteItem)，ThrottledRequests 只有在每個批次中的請求進行限流時才會遞增。
> 如果批次內的任何個別請求經過限流，則會遞增下列一個指標：
>
> - ReadThrottleEvents：在 BatchGetItem 內限流的 GetItem 事件。
> - WriteThrottleEvents：在 BatchWriteItem 內限流的 PutItem 或 DeleteItem 事件。

若要深入了解哪些事件限流了請求，請將 ThrottledRequests 與資料表和其索引的 ReadThrottleEvents 和 WriteThrottleEvents 進行比較。

> **Note**
>
> 經限流的請求會導致 HTTP 400 狀態碼。所有這些事件都會反映在 ThrottledRequests 指標而非 UserErrors 指標中。

單位：Count

維度：TableName, Operation

有效的統計數字：

- Sum
- SampleCount

TransactionConflict

因同一項目上並行請求之間的交易衝突而拒絕的項目層級請求。如需詳細資訊，請參閱在 DynamoDB 中處理交易衝突。

單位：Count

維度：TableName

有效的統計數字：

- Sum：因交易衝突而拒絕的項目層級請求數目。

> **Note**
>
> 如果對 TransactWriteItems 或 TransactGetItems 的呼叫中有多個項目層級請求遭拒絕，則對於每個項目層級的 Put、Update、Delete 或 Get 請求，Sum 會遞增 1。

- SampleCount：因交易衝突而拒絕的請求數目。

> **Note**
>
> 如果對 TransactWriteItems 或 TransactGetItems 的呼叫中有多個項目層級請求遭拒絕，則 SampleCount 只會遞增 1。

- Min：對 TransactWriteItems、TransactGetItems、PutItem、UpdateItem 或 DeleteItem 的呼叫中遭拒絕的項目層級請求數目下限。
- Max：對 TransactWriteItems、TransactGetItems、PutItem、UpdateItem 或 DeleteItem 的呼叫中被拒絕的項目層級請求數目上限。
- Average：對 TransactWriteItems、TransactGetItems、PutItem、UpdateItem 或 DeleteItem 的呼叫中被拒絕的項目層級請求平均數。

UserErrors

對 DynamoDB 或 Amazon DynamoDB Streams 發起的能在指定時段內產生 HTTP 400 狀態碼的請求。HTTP 400 通常表示用戶端錯誤，例如參數組合無效、嘗試更新不存在的資料表或請求簽章不正確。

有些例外狀況會記錄關於 UserErrors 的指標，範例如下：

- ResourceNotFoundException
- ValidationException
- TransactionConflict

所有這些事件都會反映在 UserErrors 指標中，但下列項目除外：

- ProvisionedThroughputExceededException：請參閱本節的 ThrottledRequests 指標。
- ConditionalCheckFailedException：請參閱本節的 ConditionalCheckFailedRequests 指標。

UserErrors 代表目前 AWS 區域和目前 AWS 帳戶的 DynamoDB 或 Amazon DynamoDB Streams 請求的 HTTP 400 錯誤彙總。

單位：Count

有效的統計數字：

- Sum
- SampleCount

WriteAccountLimitThrottleEvents

因帳戶限制而被限流的寫入請求數。

單位：Count

維度：TableName

有效的統計數字：

- Sum – 限流事件的總數。
- SampleCount – 限流發生次數。
- Minimum – 任何指定範例中的限流事件數目下限。
- Maximum – 任何指定範例中的限流事件數目上限。

WriteKeyRangeThroughputThrottleEvents

因分割區限制而被限流的寫入請求數。

單位：Count

維度：TableName、GlobalSecondaryIndexName

有效的統計數字：

- Sum – 限流事件的總數。
- SampleCount – 限流發生次數。
- Minimum – 任何指定範例中的限流事件數目下限。
- Maximum – 任何指定範例中的限流事件數目上限。

WriteMaxOnDemandThroughputThrottleEvents

因隨需模式達最大輸送量而被限流的寫入請求數。

單位：Count

維度：TableName、GlobalSecondaryIndexName

有效的統計數字：

- Sum – 限流事件的總數。
- SampleCount – 限流發生次數。
- Minimum – 任何指定範例中的限流事件數目下限。
- Maximum – 任何指定範例中的限流事件數目上限。

WriteProvisionedThroughputThrottleEvents

因佈建輸送量限制而被限流的寫入請求數。

單位：Count

維度：TableName、GlobalSecondaryIndexName

有效的統計數字：

- Sum – 限流事件的總數。
- SampleCount – 限流發生次數。
- Minimum – 任何指定範例中的限流事件數目下限。
- Maximum – 任何指定範例中的限流事件數目上限。

WriteThrottleEvents

對 DynamoDB 的請求超過資料表或全域次要索引的佈建寫入容量單位。

單一請求可能會導致多個事件。例如，在附有三個全域次要索引的資料表上提出 PutItem 請求會導致四個事件：一個資料表寫入事件、三個索引寫入事件。針對每個事件，如果該事件經限流，則 WriteThrottleEvents 指標會遞增 1。針對單一 PutItem 請求，如果有任何事件經過限流，則 ThrottledRequests 也會遞增 1。針對 BatchWriteItem，除非全部個別 PutItem 或 DeleteItem 事件經過調解，否則整個 BatchWriteItem 的 ThrottledRequests 指標不會遞增。

此 TableName 維度會針對資料表傳回 WriteThrottleEvents，但不會針對任何全域次要索引傳回。若要檢視全域次要索引的 WriteThrottleEvents，您必須指定 TableName 和 GlobalSecondaryIndexName。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- Sum
- SampleCount

用量指標

CloudWatch 中的用量指標可讓您主動管理用量，方法是將 CloudWatch 主控台中的指標視覺化、建立自訂儀表板、使用 CloudWatch 異常偵測偵測活動中的變更，以及設定在用量達到閾值時提醒您的警示。

DynamoDB 也將這些使用量指標與 Service Quotas 整合。您可以使用 CloudWatch 來管理帳戶對服務配額的使用情況。如需詳細資訊，請參閱視覺化您的 Service Quotas 和設定警報。

可用使用量指標的清單

- AccountProvisionedWriteCapacityUnits
- AccountProvisionedReadCapacityUnits
- TableCount

AccountProvisionedWriteCapacityUnits

帳戶所有資料表或全域次要索引佈建的寫入容量單位數目總和。

單位：Count

有效的統計數字：

- Minimum：在一時間段內佈建的最低寫入容量單位數目。
- Maximum：在一時間段內佈建的最大寫入容量單位數目。
- Average – 在一段時間內佈建的寫入容量單位帳戶的平均數量。

此指標會每隔五分鐘發布一次。因此，如果您快速調整佈建的寫入容量單位，此統計數字可能無法反映真實的平均值。

AccountProvisionedReadCapacityUnits

所有資料表或全域次要索引佈建的讀取容量單位數目。

單位：Count

有效的統計數字：

- Minimum：在一時間段內佈建的最低讀取容量單位數目。
- Maximum：在一時間段內佈建的最大讀取容量單位數目。
- Average – 在一段時間內佈建的讀取容量單位帳戶的平均數量。

此指標會每隔五分鐘發布一次。因此，如果您快速調整佈建的讀取容量單位，此統計數字可能無法反映真實的平均值。

TableCount

帳戶中作用中資料表的數目。

單位：Count

有效的統計數字：

- Minimum：在一時間段內的最小資料表數目。
- Maximum：在一時間段內的最大資料表數目。
- Average – 在一時間段內的平均資料表數目。

FaultInjectionServiceInducedErrors

對 DynamoDB 提出的請求，會在指定的期間和補足期間產生模擬 HTTP 500 狀態碼，因為 AWS FIS 實驗。

單位：Count

維度：TableName、Operation

有效的統計數字：

- `Sum`
- `SampleCount`

了解 DynamoDB 的指標和維度

DynamoDB 的指標條件由帳戶的數值、資料表名稱、全域次要索引名稱或操作進行限定。您可以使用 CloudWatch 主控台擷取 DynamoDB 資料以及下表中的任何維度。

可用維度的列表

- DelegatedOperation
- GlobalSecondaryIndexName
- 作業
- OperationType
- 動詞
- ReceivingRegion
- StreamLabel
- TableName

DelegatedOperation

此維度將資料限制在 DynamoDB 代表您執行的操作中。擷取下列操作：

- Kinesis Data Streams 的變更資料擷取。

GlobalSecondaryIndexName

此維度將資料限制為資料表上的全域次要索引。若您指定 `GlobalSecondaryIndexName`，您也必須指定 `TableName`。

作業

此維度將資料限制為下列其中一項 DynamoDB 操作：

- `PutItem`
- `DeleteItem`
- `UpdateItem`

- GetItem
- BatchGetItem
- Scan
- Query
- BatchWriteItem
- TransactWriteItems
- TransactGetItems
- ExecuteTransaction
- BatchExecuteStatement
- ExecuteStatement

此外，您可以將資料限制為下列 Amazon DynamoDB Streams 操作：

- GetRecords

OperationType

此維度將資料限制為下列其中一種操作類型：

- Read
- Write

此維度針對回應 ExecuteTransaction 和 BatchExecuteStatement 請求發出。

動詞

此維度將資料限制為下列其中一個 DynamoDB PartiQL 動詞：

- Insert (插入)：PartiQLInsert
- Select (選取)：PartiQLSelect
- Update (更新)：PartiQLUpdate
- Delete (刪除)：PartiQLDelete

此維度針對回應 ExecuteStatement 操作發出。

ReceivingRegion

此維度會將資料限制在特定 AWS 區域。它可與源自 DynamoDB 全域資料表內複本列表的指標搭配使用。

StreamLabel

此維度將資料限制為特定串流標籤。它與源自 Amazon DynamoDB Streams `GetRecords` 操作的指標搭配使用。

TableName

此維度將資料限制為特定資料表。此值可以是目前區域和目前 AWS 帳戶中的任何資料表名稱。

在 DynamoDB 中建立 CloudWatch 警示

CloudWatch 警示會監看指定時段內的單一指標，並在一段時間內根據指標相對於閾值的值來執行一或多個指定動作。此動作是傳送到 Amazon SNS 主題或 Auto Scaling 政策的通知。您也可以將警示新增至儀表板，以便監控和接收跨多個區域的AWS資源和應用程式的提醒。您可建立的警示數量沒有限制。CloudWatch 警示不會只因處於特定狀態就調用動作，狀態必須已變更並已維持一段指定的時間。如需建議的 DynamoDB 警示清單，請參閱建議的警示。

> **Note**
>
> 建立 CloudWatch 警示時，您必須指定所有必要的維度，因為 CloudWatch 不會彙總遺失維度的指標。在建立警示時，使用遺失維度建立 CloudWatch 警示不會造成錯誤。

假設您有一個佈建的資料表，其中包含五個讀取容量單位。您希望在用盡整個佈建的讀取容量之前收到通知，因此您決定建立 CloudWatch 警示，以便在使用的容量達到您為資料表佈建的 80% 時收到通知。您可以在 CloudWatch 主控台中或使用AWS CLI建立警示。

在 CloudWatch 主控台中建立警示

若要在 CloudWatch 主控台中建立警示

1. 登入AWS 管理主控台，並在 https：//https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在導覽窗格中，選擇 Alarms (警示)、All alarms (所有警示)。
3. 選擇 Create alarm (建立警示)。

4. 尋找包含您要監控之資料表的資料列，並在指標名稱欄中尋找 **ConsumeReadCapacityUnits**。選取此資料列旁的核取方塊，然後選擇 Select metric (選取指標)。
5. 在指定指標和條件下，針對統計資料選擇總和。對於期間，選擇 1 分鐘。
6. 在 Conditions (條件) 下，指定以下內容：
   a. 對於閾值類型，選擇靜態。
   b. 對於每當 **ConsumedReadCapacityUnits** 為，選擇大於/等於，然後輸入 240 作為閾值。
7. 選擇 Next (下一步)。
8. 在通知 下，選擇 **In alarm**，然後選取當警示處於 ALARM 狀態時要通知的 SNS 主題。
9. 完成時，請選擇下一步。
10. 輸入警示的名稱和說明，然後選擇 Next (下一步)。
11. 在 Preview and create (預覽及建立) 下，請確認資訊和條件都是您希望的內容，然後選擇 Create alarm (建立警示)。

在 中建立警示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
```

測試警示。

```
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
```

更多AWS CLI範例

下列程序說明如何在請求超過資料表的佈建調節配額時收到通知。

1. 建立 Amazon SNS 主題 `arn:aws:sns:us-east-1:123456789012:requests-exceeding-throughput`。如需詳細資訊，請參閱設定 Amazon Simple Notification Service。
2. 建立警示。

```
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. 測試警示。

```
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
```

下列程序說明如何在發生系統錯誤時收到通知。

1. 建立 Amazon SNS 主題 `arn:aws:sns:us-east-1:123456789012:notify-on-system-errors`。如需詳細資訊，請參閱設定 Amazon Simple Notification Service。
2. 建立警示。

```
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. 測試警示。

```
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
```

使用 記錄 DynamoDB 操作 AWS CloudTrail

DynamoDB 已與 服務整合 AWS CloudTrail，此服務提供 DynamoDB AWS 中使用者、角色或服務所採取動作的記錄。CloudTrail 會將 DynamoDB 的所有 API 呼叫擷取為事件。擷取的呼叫包括來自 DynamoDB 主控台的呼叫，以及對 DynamoDB API 操作 (同時使用 PartiQL 和傳統 API) 的程式碼呼叫。如果建立線索，則可將 CloudTrail 事件 (包括 DynamoDB 的事件) 持續交付到 Amazon S3 儲存貯體。即使您未設定追蹤，依然可以透過 CloudTrail 主控台中的 Event history (事件歷史記錄) 檢視最新事件。您可以利用 CloudTrail 所收集的資訊來判斷向 DynamoDB 發出的請求，以及發出請求的 IP 位址、人員、時間和其他詳細資訊。

如需強大的監控和提醒功能，您也可以整合 CloudTrail 事件與 Amazon CloudWatch Logs。若要增強對 DynamoDB 服務活動的分析，並識別 AWS 帳戶活動的變更，您可以使用 Amazon Athena 查詢 AWS CloudTrail 日誌。例如，您可以使用查詢來識別趨勢，並依屬性 (例如來源 IP 位址或使用者) 進一步隔離活動。

若要進一步了解 CloudTrail，包括如何設定及啟用，請參閱《AWS CloudTrail 使用者指南》。

主題

- CloudTrail 中的 DynamoDB 資訊

- 了解資料 DynamoDB 日誌檔案項目

CloudTrail 中的 DynamoDB 資訊

當您建立 AWS 帳戶時，會在您的帳戶上啟用 CloudTrail。當 DynamoDB 中發生支援的事件活動時，該活動會與事件歷史記錄中的其他 AWS 服務事件一起記錄在 CloudTrail 事件中。您可以在 AWS 帳戶中檢視、搜尋和下載最近的事件。如需詳細資訊，請參閱使用 CloudTrail 事件歷史記錄。

若要持續記錄您 AWS 帳戶中的事件，包括 DynamoDB 的事件，請建立追蹤。線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。根據預設，當您在主控台中建立線索時，線索會套用至所有 AWS 區域。線索會記錄 AWS 分割區中所有區域的事件，並將日誌檔案傳送到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中收集的事件資料。如需詳細資訊，請參閱下列內容：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知
- 接收多個區域的 CloudTrail 日誌檔案和接收多個帳戶的 CloudTrail 日誌檔案

CloudTrail 中的控制平面事件

下列 API 動作記錄預設為 CloudTrail 檔案中的事件：

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

CloudTrail 中的 DynamoDB 資料平面事件

若要在 CloudTrail 檔案中啟用下列 API 動作的日誌紀錄，您需要在 CloudTrail 中啟用資料平面 API 活動的日誌紀錄。如需詳細資訊，請參閱記錄追蹤的資料事件。

您可以依資源類型篩選資料平面事件，以精細控制您要在 CloudTrail 中選擇性地記錄和付費的 DynamoDB API 呼叫。例如，藉由指定 `AWS::DynamoDB::Stream` 作為資源類型，您就能只記錄 DynamoDB 串流 API 的呼叫。對於已啟用串流的資料表，資料平面事件中的資源欄位會同時包含 `AWS::DynamoDB::Stream` 和 `AWS::DynamoDB::Table`。如果您指定 `AWS::DynamoDB::Table` 作為資源類型，則會根據預設同時記錄 DynamoDB 資料表和 DynamoDB 串流事件。如果您不想要記錄串流事件，可以新增額外的篩選條件來排除串流事件。如需詳細資訊，請參閱 AWS CloudTrail API 參考中的 DataResource。

Amazon DynamoDB

- BatchExecuteStatement
- BatchGetItem
- BatchWriteItem
- DeleteItem

- ExecuteStatement
- ExecuteTransaction
- GetItem
- PutItem
- Query
- Scan
- TransactGetItems
- TransactWriteItems
- UpdateItem

> **Note**
>
> CloudTrail 不會記錄 DynamoDB 的存留時間資料平面動作

DynamoDB Streams

- GetRecords
- GetShardIterator

> **Important**
>
> 當您記錄`GetRecords`資料事件時，您可能會看到來自 DynamoDB 內部操作的`GetRecords`呼叫，例如全域資料表複寫。雖然 DynamoDB 不會向您收取這些`GetRecords`呼叫的費用，但 CloudTrail 會向您收取資料事件日誌的費用。這可能會導致非預期的 CloudTrail 費用。
> 若要避免意外的 CloudTrail 費用，請執行下列其中一項操作：
>
> - 使用排除 AWS 服務起始的事件日誌選取器範本。
> - 新增進階事件選取器篩選條件，並將 `userIdentity.arn`設定為 `NotStartsWith` `AWSServiceRoleFor`。
>
> 如需詳細資訊，請參閱 AWS CloudTrail 《使用者指南》中的記錄資料事件。

了解資料 DynamoDB 日誌檔案項目

追蹤是一種組態，能讓事件以日誌檔案的形式交付到您指定的 Amazon S3 儲存貯體。CloudTrail 日誌檔案包含一或多個日誌專案。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否使用根或使用者憑證提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。

> **Note**
>
> 非索引鍵屬性值會在使用 PartiQL API 的 CloudTrail 動作日誌中進行修訂，而不會出現在使用傳統 API 的動作日誌中。

如需詳細資訊，請參閱 CloudTrail userIdentity 元素。

下列範例會示範這些事件類型的 CloudTrail 日誌：

Amazon DynamoDB

- UpdateTable
- DeleteTable
- CreateCluster
- PutItem (成功)
- UpdateItem (不成功)
- TransactWriteItems (成功)
- TransactWriteItems (與 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 (成功)

```
{
    "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 (不成功)

```
{
    "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` (成功)

```
{
    "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` (與 `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"
        }
    ]
```

```
}
```

使用 DynamoDB 專用 CloudWatch Contributor Insights 分析資料存取

Amazon DynamoDB 專用 Amazon CloudWatch Contributor Insights 為分析工具，可快速找到您資料表或索引中最常存取和限流的索引鍵。此工具會使用 CloudWatch Contributor Insights。

在資料表或全域次要索引中啟用 DynamoDB 專用 CloudWatch Contributor Insights，您就可以在這些資源中檢視最常存取和限流的項目。

> **Note**
>
> 使用 DynamoDB 專用 Contributor Insights 須支付 CloudWatch 費用。如需定價的詳細資訊，請參閱 Amazon CloudWatch 定價。

主題

- DynamoDB 專用 CloudWatch Contributor Insights：運作方式
- DynamoDB 專用 CloudWatch Contributor Insights 入門
- 將 IAM 與 DynamoDB 專用 CloudWatch Contributor Insights 搭配使用

DynamoDB 專用 CloudWatch Contributor Insights：運作方式

Amazon DynamoDB 已與 CloudWatch Contributor Insights 整合，可提供資料表或全域次要索引中最常存取和限流的項目資訊。DynamoDB 透過 CloudWatch Contributor Insights 規則、報告以及報告資料的圖形將此資訊提供給您。

DynamoDB 專用 CloudWatch Contributor Insights 的設計不會影響 DynamoDB 資料表的效能。

如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的使用 CloudWatch Contributor Insights 分析高基數資料。

以下章節說明 DynamoDB 專用 CloudWatch Contributor Insights 核心概念及行為。

主題

- DynamoDB 專用 CloudWatch Contributor Insights 規則
- DynamoDB 專用 CloudWatch Contributor Insights 規則
- 了解 DynamoDB 專用 CloudWatch Contributor Insights 圖表
- 與其他 DynamoDB 功能互動
- DynamoDB 專用 CloudWatch Contributor Insights 帳單

DynamoDB 專用 CloudWatch Contributor Insights 規則

DynamoDB 專用 CloudWatch Contributor Insights 提供兩種不同的模式，以滿足不同的監控需求。

限流索引鍵模式

此模式僅針對限流請求，僅在發生限流時處理事件。其提供效能問題的洞見，而沒有追蹤所有存取模式的負擔。在此模式中，DynamoDB 只會追蹤：

- 最常限流的項目 – 遇到最多限流事件的項目

此模式適合用於：

- 您的主要考量是識別和解決限流問題時
- 您想要持續啟用 Contributor Insights 以進行即時限流偵測時
- 您希望採用成本最佳化方法來監控限流問題時

> **Note**
>
> 限流索引鍵模式只會在限流發生時處理事件，因此在持續監控方面具有成本效益。這種針對性的方法可讓您永久啟用此功能，並將對成本的影響降到最小，同時仍能在發生限流問題時立即查看。

如果您的資料表沒有發生限流，您就不會在 Contributor Insights 圖形中看到任何資料，這表示效能良好。偵測到限流時，產生的圖形可協助您識別導致效能問題的特定存取模式。此資訊可協助您實作策略，以解決不統一的存取模式。

如需全方位監控策略，您可以將這些限流洞見與其他 CloudWatch 指標整合，以建立統一儀表板，將限流事件與整體資料表效能建立關聯。

存取和限流索引鍵模式

此模式提供存取和限流項目的完整監控。在此模式中，DynamoDB 會追蹤：

- 最常存取的項目 – 消耗最多讀取和寫入容量的項目
- 最常限流的項目 – 遇到最多限流事件的項目

當您需要完整查看資料表的存取模式，且想要同時了解高流量項目和限流問題時，此模式是最理想的選擇。

在模式之間切換

您可以隨時使用 DynamoDB 主控台 AWS CLI或 APIs 切換模式。切換模式時：

- 現有的 CloudWatch 規則會更新以符合新模式
- 限流金鑰 CloudWatch 規則會保持不變，以維護您用於限流指標的持續歷史資料：
  - 從限流索引鍵模式切換為存取和限流索引鍵模式時，會保留現有的限流索引鍵規則，並建立新的存取索引鍵規則
  - 從存取和限流索引鍵模式切換為限流索引鍵模式時，只會保留限流索引鍵規則，並移除存取索引鍵規則
- 立即調整帳單以反映新模式的事件處理

DynamoDB 專用 CloudWatch Contributor Insights 規則

在資料表或全域次要索引中啟用 DynamoDB 專用 CloudWatch Contributor Insights 時，DynamoDB 會代表您根據所選模式建立規則。

> **Note**
>
> 在 DynamoDB 資料表上啟用 Contributor Insights 時，會受到 Contributor Insights 規則限制。如需詳細資訊，請參閱 CloudWatch 服務配額。

存取和限流索引鍵模式的規則

在存取和限流索引鍵模式中，DynamoDB 會建立下列規則：

- 最常存取的項目 （分割區索引鍵） — 識別資料表或全域次要索引中最常存取項目的分割區索引鍵。

CloudWatch 規則名稱格式：DynamoDBContributorInsights-PKC-[resource_name]-[creationtimestamp]

- 大多數節流索引鍵（分割區索引鍵）— 識別資料表或全域次要索引中節流項目的分割區索引鍵。

  CloudWatch 規則名稱格式：DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]

如果您的資料表或全域次要索引具有排序索引鍵，DynamoDB 也會建立下列專屬於排序索引鍵的規則：

- 最常存取項目 (分割區和排序索引鍵)：在您的資料表或全域次要索引中辨識最常存取項目的分割區和排序索引鍵。

  CloudWatch 規則名稱格式：DynamoDBContributorInsights-SKC-[resource_name]-[creationtimestamp]

- 最常限流項目 (分割區和排序索引鍵)：在您的資料表或全域次要索引中辨識最常限流項目的分割區和排序索引鍵。

  CloudWatch 規則名稱格式：DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]

限流索引鍵模式的規則

在限流索引鍵模式中，DynamoDB 只會建立限流相關規則：

- 最常限流項目 (分割區索引鍵)：在您的資料表或全域次要索引中辨識最常限流項目的分割區索引鍵。

  CloudWatch 規則名稱格式：DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]

如果您的資料表或全域次要索引具有排序索引鍵，DynamoDB 也會建立：

- 最常限流項目 (分割區和排序索引鍵)：在您的資料表或全域次要索引中辨識最常限流項目的分割區和排序索引鍵。

  CloudWatch 規則名稱格式：DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]

這種重點方法可減少作用中規則的數量，並減少處理的事件量，以更有效地診斷限流事件。

> **Note**
>
> - 使用 CloudWatch 主控台或 API 檢視 DynamoDB 的 CloudWatch Contributor Insights 時，您只會看到與所選模式對應的規則。
> - 您無法使用 CloudWatch 主控台或 API 直接修改或刪除 DynamoDB 專用 CloudWatch Contributor Insights 所建立的規則。在資料表或全域次要索引中，停用 DynamoDB 專用 CloudWatch Contributor Insights，系統就會自動刪除針對該資料表或全域次要索引建立的規則。
> - 在搭配 DynamoDB 建立的 CloudWatch Contributor Insights 規則使用 GetInsightRuleReport 操作時，只有 `MaxContributorValue` 和 `Maximum` 會傳回有用的統計數字。此清單中的其他統計數字不會傳回有意義的值。
> - DynamoDB 專用 CloudWatch Contributor Insights 有 25 個貢獻因子的限制。請求超過 25 個參與者會傳回錯誤。

您可以使用 DynamoDB 專用 CloudWatch Contributor Insights 規則建立 CloudWatch 警示。這可讓您在任何項目超過或達到 `ConsumedThroughputUnits` 或 `ThrottleCount` 的特定門檻時收到通知。如需詳細資訊，請參閱在 Contributor Insights 指標資料上設定警示。

了解 DynamoDB 專用 CloudWatch Contributor Insights 圖表

DynamoDB 專用 CloudWatch Contributor Insights 會根據所選模式在 DynamoDB 和 CloudWatch 主控台中顯示不同類型的圖表。

依模式的圖形可用性

顯示的圖形取決於您選取的 Contributor Insights 模式。

- 存取和限流索引鍵模式會顯示最常存取的項目和最常限流的項目圖表
- 限流索引鍵模式只會顯示最常限流的項目圖表

最常存取項目

此圖形僅適用於存取和限流索引鍵模式。使用此圖表在資料表或全域次要索引中辨識最常存取的項目。圖表的 Y 軸代表 `ConsumedThroughputUnits`，而 X 軸代表時間。每個主要 N 金鑰都會以其各自的顏色顯示，而且在 X 軸下方也有圖例說明。

DynamoDB 會使用 ConsumedThroughputUnits 評量金鑰的存取頻率，且此評量數值結合讀取和寫入流量。ConsumedThroughputUnits 的定義如下：

- 佈建：(3 x 已消耗寫入容量單位) + 已消耗讀取容量單位
- 隨需：(3 x 寫入請求單位) + 讀取請求單位

在 DynamoDB 主控台上，圖表中的每個資料點都代表 1 分鐘內 ConsumedThroughputUnits 的最大值。例如，圖表值 180,000 ConsumedThroughputUnits 代表該項目持續存取的頻率為在 1 分鐘內，每 60 秒的每個項目最大輸送量 1,000 個寫入要求單位或 3,000 個讀取要求單位 (3,000 x 60 秒)。也就是說，圖表中呈現的數值代表每 1 分鐘內最高的每分鐘流量。您可以在 CloudWatch 主控台上變更 ConsumedThroughputUnits 指標的時間精細程度 (例如：檢視 5 分鐘指標而不是 1 分鐘)。

如果您看到多條緊密聚集的線，沒有任何明顯的異常，就表示您於特定時期內部同項目的工作負載相對來說還算平衡。如果圖表中出現孤立點而不是相連的線，就表示有一個項目僅於短時間內經常存取。

如果您的資料表或全域次要索引具有排序索引鍵，DynamoDB 會建立兩個圖形：一個用於最常存取的分割區索引鍵，另一個用於最常存取的分割區 + 排序索引鍵對。您可以在僅限分割區索引鍵的圖形中，查看分割區索引鍵層級的流量。您可以在分割區 + 排序索引鍵圖形中的項目層級看到流量。

最常限流項目

此圖形在兩種模式中皆可使用。使用此圖表在資料表或全域次要索引中辨識最常限流的項目。圖表的 Y 軸代表 ThrottleCount，而 X 軸代表時間。每個主要 N 金鑰都會以其各自的顏色顯示，而且在 X 軸下方也有圖例說明。

DynamoDB 會使用 ThrottleCount 來評估限流頻率，也就是 ProvisionedThroughputExceededException、ThrottlingException 和 RequestLimitExceeded 錯誤計數。

不會測量因全域次要索引的寫入容量不足而造成的寫入限流。您可以使用全域次要索引的最常存取項目圖形，識別可能會造成寫入限流的不平衡存取模式。如需詳細資訊，請參閱全域次要索引的佈建輸送量考量。

在 DynamoDB 主控台上，圖形中的每個資料點代表 1 分鐘內的限流事件計數。

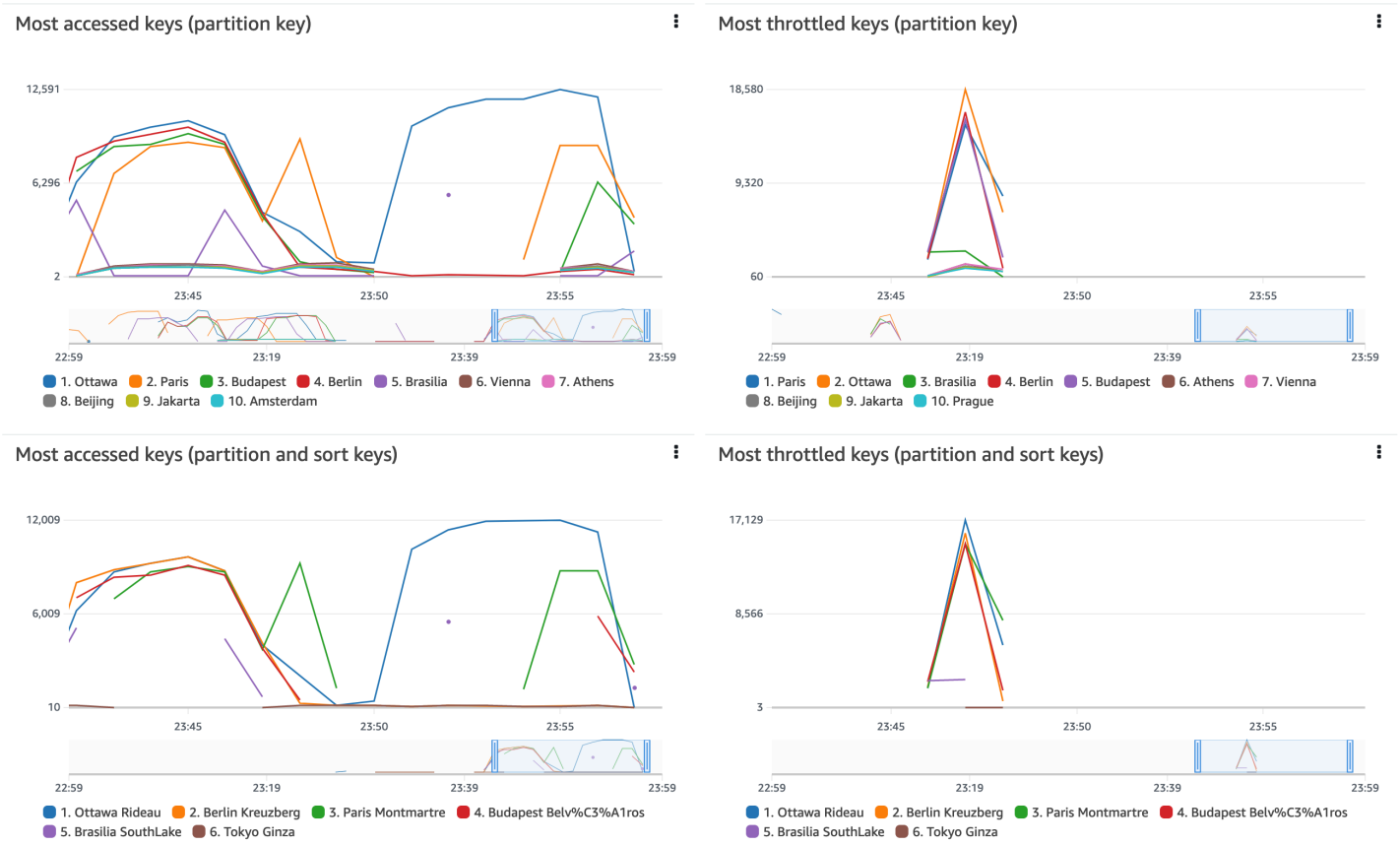
如果此圖表沒有顯示任何資料，就表示您的要求沒有受到限流。如果圖表中出現孤立點而不是相連的線，表示有一個項目僅於短時間內經常限流。

如果您的資料表或全域次要索引具有排序索引鍵，DynamoDB 會建立兩個圖形：一個用於大多數調節的分割區索引鍵，另一個用於大多數調節的分割區 + 排序索引鍵對。您可以在分割區索引鍵限定圖形的分割區索引鍵層級查看限流計數，並在分割區 + 排序索引鍵圖形的項目層級查看限流計數。

> **Note**
>
> 在限流索引鍵模式中，這是您將看到的唯一圖形類型。這些圖表中未顯示任何資料，表示資料表效能良好，未發生限流。

報告範例

下列範例顯示針對具有分割區索引鍵和排序索引鍵的資料表，在存取和限流索引鍵模式中產生的報告。在限流索引鍵模式中，您只會看到此報告的限流相關部分。



與其他 DynamoDB 功能互動

下列各節說明 DynamoDB 專用 CloudWatch Contributor Insights 如何運作，以及如何與 DynamoDB 中的數種其他功能互動。除非另有說明，否則這些行為適用於這兩種模式。

全域資料表

DynamoDB 專用 CloudWatch Contributor Insights 監控全域資料表複本作為不同的資料表。一個 AWS 區域中複本的 Contributor Insights 圖形可能不會顯示與另一個區域相同的模式。這是因為寫入資料於所有全域資料表中的複本複製，但每個複本都可做為區域性讀取流量。

每個複本都可以獨立設定不同的 Contributor Insights 模式。例如，您可以在主要區域中使用存取和限流索引鍵模式以進行全面監控，同時在次要區域中使用限流索引鍵模式以維持效能問題的可見性。

DynamoDB Accelerator (DAX)

DynamoDB 專用 CloudWatch Contributor Insights 不會顯示 DAX 快取回應。它只會顯示存取資料表或全域次要索引的回應。

> **Note**
>
> DynamoDB CloudWatch Contributor Insights 不支援 PartiQL 請求。

靜態加密

DynamoDB 專用 CloudWatch Contributor Insights 不會影響 DynamoDB 中的加密運作方式。CloudWatch 中發布的主索引鍵資料會用 AWS 擁有的金鑰加密。不過，DynamoDB 也支援 AWS 受管金鑰 和客戶受管金鑰。

DynamoDB 專用 CloudWatch Contributor Insights 圖表會以純文字顯示最常存取項目和最常限流項目的分割區索引鍵和排序索引鍵 (如適用)。雖然 CloudWatch Contributor Insights 可與加密的 DynamoDB 資料表搭配使用，但請務必注意，其使用自己的 Amazon 擁有的加密內容，這與資料表設定的加密不同。

如果您的 DynamoDB 資料表的主索引鍵包含敏感資訊，且您組織的安全政策需要完全控制加密程序，可能不合適啟用 CloudWatch Contributor Insights。

精細定義存取控制

DynamoDB 專用 CloudWatch Contributor Insights 與有精細定義存取控制 (FGAC) 的資料表功能相同。也就是說，任何有 CloudWatch 相關許可的使用者都可以在 CloudWatch Contributor Insights 圖表中檢視 FGAC 保護的主索引鍵。

如果資料表的主索引鍵包含 FGAC 保護的資料，且您不希望在 CloudWatch 中發布該資料，建議您不要針對資料表啟用 DynamoDB 專用 CloudWatch Contributor Insights。

存取控制

您可以透過限制 DynamoDB 控制平面許可和 CloudWatch 資料平面許可，使用 AWS Identity and Access Management (IAM) 控制對 DynamoDB 的 CloudWatch Contributor Insights 的存取。如需詳細資訊，請參閱搭配 DynamoDB 專用 CloudWatch Contributor Insights 使用 IAM。

DynamoDB 專用 CloudWatch Contributor Insights 帳單

DynamoDB 專用 CloudWatch Contributor Insights 費用會出現在您每月帳單的 CloudWatch 區段中。這些費用是根據處理的 DynamoDB 事件數和選擇的模式來計算。

依模式計費

這兩種 Contributor Insights 模式具有不同的計費特性。

- 存取和限流索引鍵模式計費 - 在此模式中，透過資料平面操作寫入或讀取的每個項目都代表一個事件，無論請求是否成功或受到限流。如果資料表或全域次要索引包含排序索引鍵，則讀取或寫入的每個項目代表兩個事件。這是因為 DynamoDB 正在識別來自不同時間序列的主要參與者：一個僅用於分割區索引鍵，另一個用於分割區和排序索引鍵對。
- 限流索引鍵模式計費 - 在此模式中，只有限流請求會產生計費事件。只有在請求導致 `ProvisionedThroughputExceededException`、`ThrottlingException` 或 `RequestLimitExceeded` 錯誤時，才會產生事件。如果資料表或全域次要索引包含排序索引鍵，則每個限流項目代表兩個事件（分割區索引鍵追蹤和分割區 + 排序索引鍵追蹤）。

計費範例

例如，假設您的應用程式要執行下列 DynamoDB 操作：`GetItem`、`PutItem` 和放置 5 個項目的 `BatchWriteItem`。也假設 `PutItem` 操作受到限流，但所有其他操作都成功。

- 存取和限流索引鍵模式
  - 如果您的資料表或全域次要索引只有分割區索引鍵，則會產生 7 個事件 (1 個用於 `GetItem`，1 個用於 `PutItem`，5 個用於 `BatchWriteItem`)。
  - 如果您的資料表或全域次要索引具有分割區索引鍵和排序索引鍵，則會產生 14 個事件 (2 個用於 `GetItem`，2 個用於 `PutItem`，10 個用於 `BatchWriteItem`)。
- 限流索引鍵模式
  - 如果您的資料表或全域次要索引只有分割區索引鍵，則會產生 1 個事件（僅適用於調節的 `PutItem`)。

- 如果您的資料表或全域次要索引具有分割區索引鍵和排序索引鍵，則會產生 2 個事件 （調節的 為 2PutItem)。

成功的 GetItem 和 BatchWriteItem 操作在限流索引鍵模式中不會產生任何事件。

常見計費因素

無論模式或傳回的項目數為何，Query 操作一律產生 1 個事件。

與其他 DynamoDB 功能不同，DynamoDB 專用 CloudWatch Contributor Insights 計費不會隨下列條件變化：

- 容量模式 (佈建與隨需)
- 是否執行讀取或寫入請求
- 讀取或寫入項目的大小 (KB)

DynamoDB 專用 CloudWatch Contributor Insights 入門

本節說明如何在不同模式下啟用和使用 Amazon CloudWatch Contributor Insights，以使用 Amazon DynamoDB 主控台或 AWS Command Line Interface () 滿足您的監控需求AWS CLI。

在下列範例中，您會用到 DynamoDB 入門教學課程中定義的 DynamoDB 資料表。

主題

- 選擇 Contributor Insights 模式
- 使用 Contributor Insights (主控台)
- 使用 Contributor Insights (AWS CLI)

選擇 Contributor Insights 模式

在啟用 Contributor Insights 之前，您應該了解兩種可用的模式。檢閱模式比較，以選取最符合您特定需求的選項。

面向	存取和限流索引鍵模式	限流索引鍵模式
監視器	所有請求 (成功和限流)	僅限流的請求

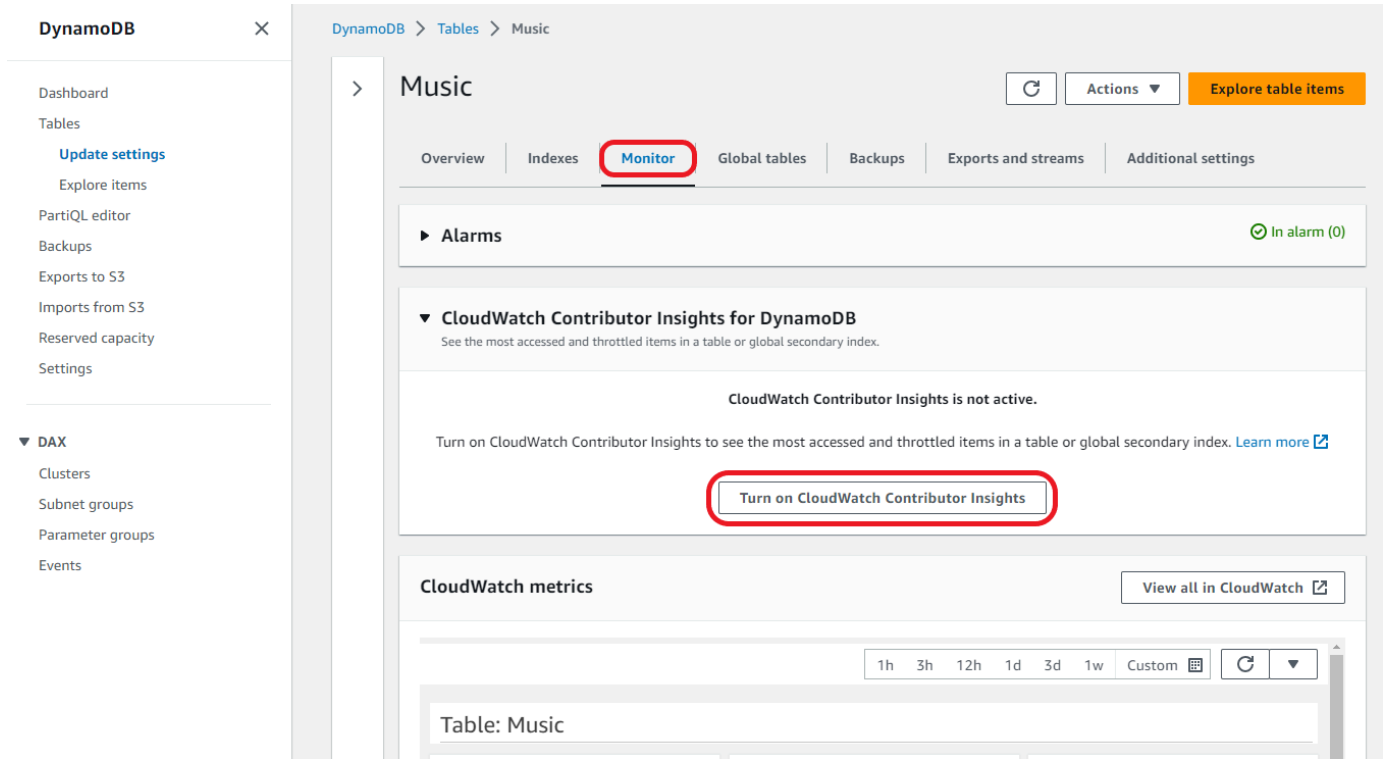
面向	存取和限流索引鍵模式	限流索引鍵模式
圖表	最常存取的項目 + 最常限流的項目	最常限流的項目
最適合	目標分析和最佳化	限流監控
使用時機	您需要完全了解存取模式。您正在執行短期分析或偵錯。	您的主要考量是識別和解決限流問題。您想要持續啟用 Contributor Insights 以提供即時限流警示。

使用 Contributor Insights (主控台)

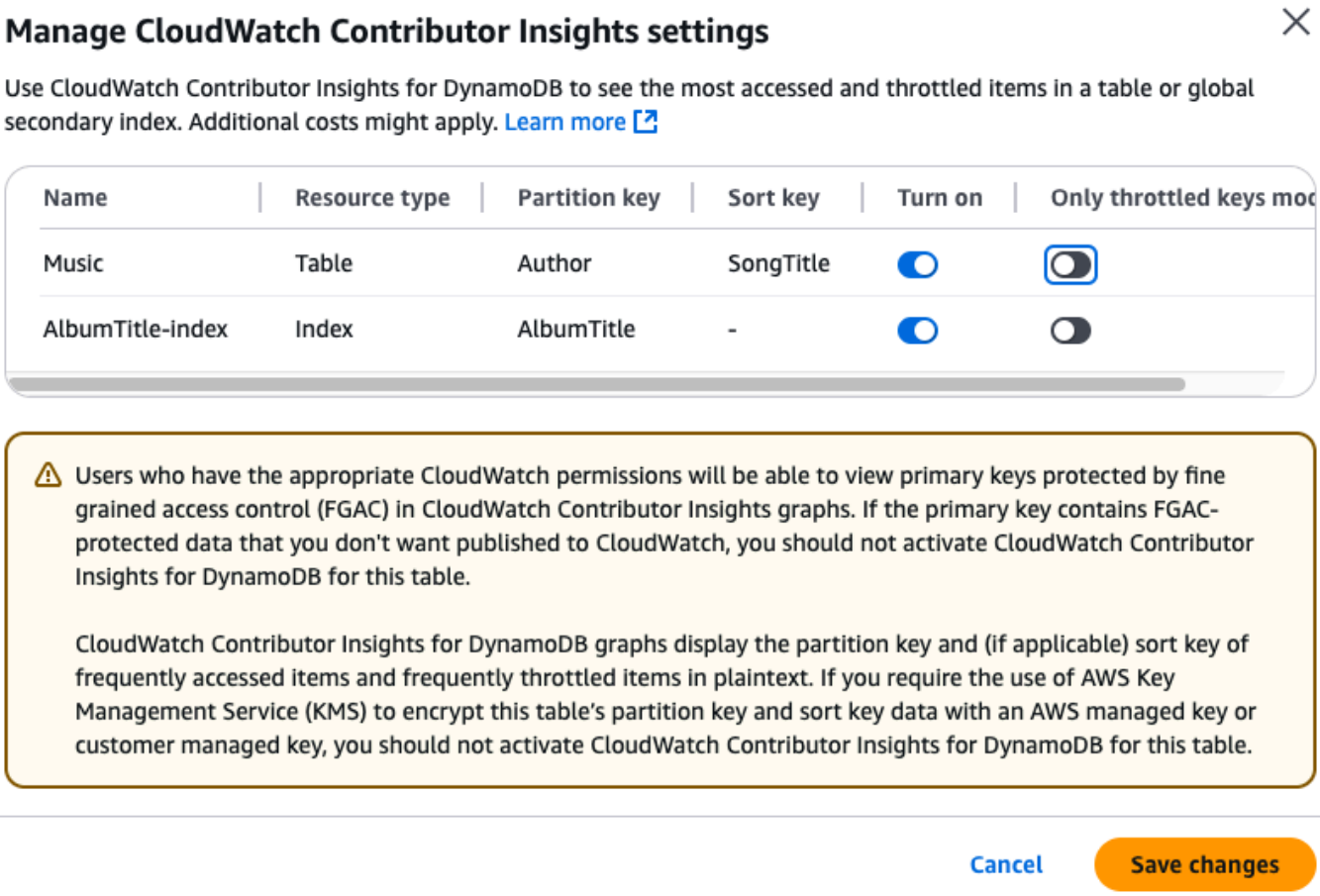
主控台提供直覺式的方式來啟用 Contributor Insights，並為您的監控需求選取適當的模式。

在主控台中使用 Contributor Insights

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇 `Music` 資料表。
4. 選擇 Monitor (監控) 索引標籤。
5. 選擇開啟 CloudWatch Contributor Insights。



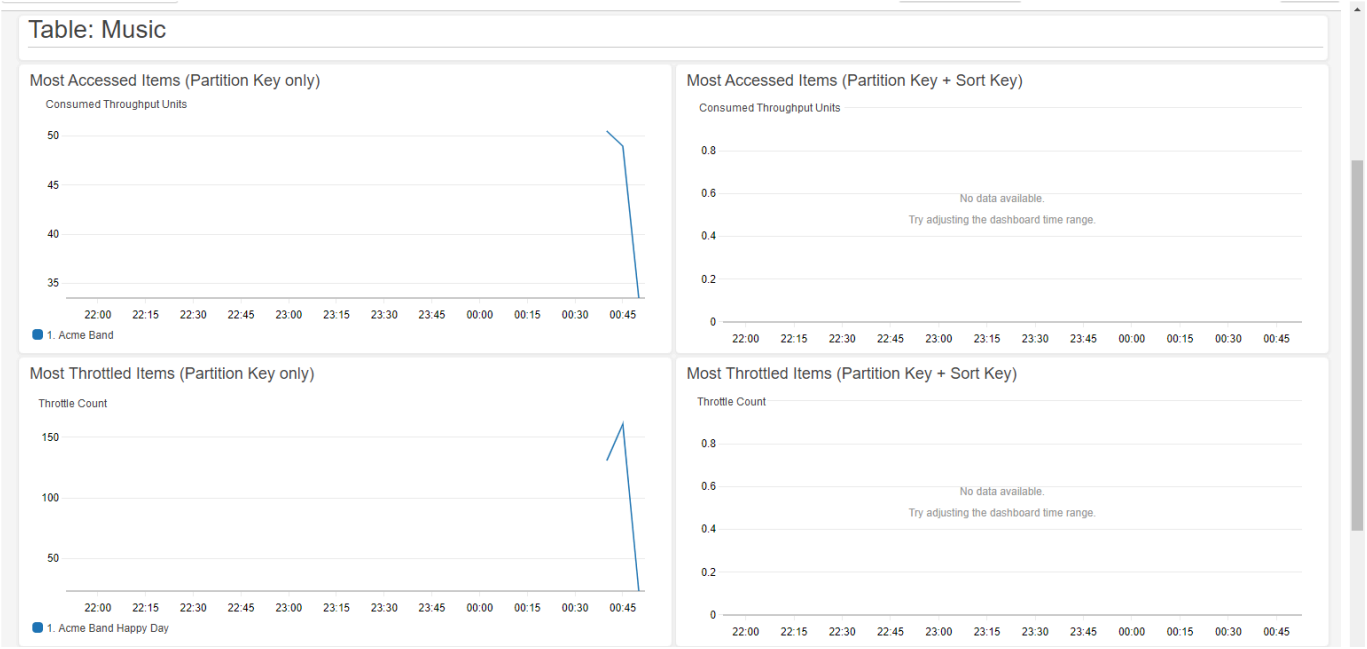
6. 在管理 CloudWatch Contributor Insights 設定對話方塊中，將 `Music` 基礎資料表和 `AlbumTitle-index` 全域次要索引切換為開啟。
7. 將兩者的僅限流索引鍵模式開關保留在關閉位置，然後選擇儲存變更。



接著將啟用資料表和 GSI 的預設存取和限流索引鍵模式，提供存取和限流項目的監控。將僅限流索引鍵模式開關切換為開啟位置會啟用限流索引鍵模式。

如果操作失敗，請參閱《Amazon DynamoDB API 參考》中的 DescribeContributorInsights FailureException 找出可能的原因。

8. CloudWatch Contributor Insights 圖表現在可以在 `Music` 資料表的監控索引標籤找到。由於已啟用存取和限流索引鍵模式，因此可以看到存取和限流項目圖表。



在模式之間切換

您可以隨時在模式之間切換，而無需停用 Contributor Insights。

切換 Contributor Insights 模式

1. 在資料表的監控索引標籤上，選擇管理 CloudWatch Contributor Insights。
2. 在管理 Contributor Insights 設定對話方塊中，針對每個基礎資料表或 GSI：
   - 開啟或關閉僅限流索引鍵模式，以啟用限流索引鍵模式，或返回預設的存取和限流索引鍵模式。
   - 切換為關閉，以停用資料表或 GSI 的 CloudWatch Contributor Insight。
3. 選擇儲存變更。

   完成後，圖形將反映新模式。

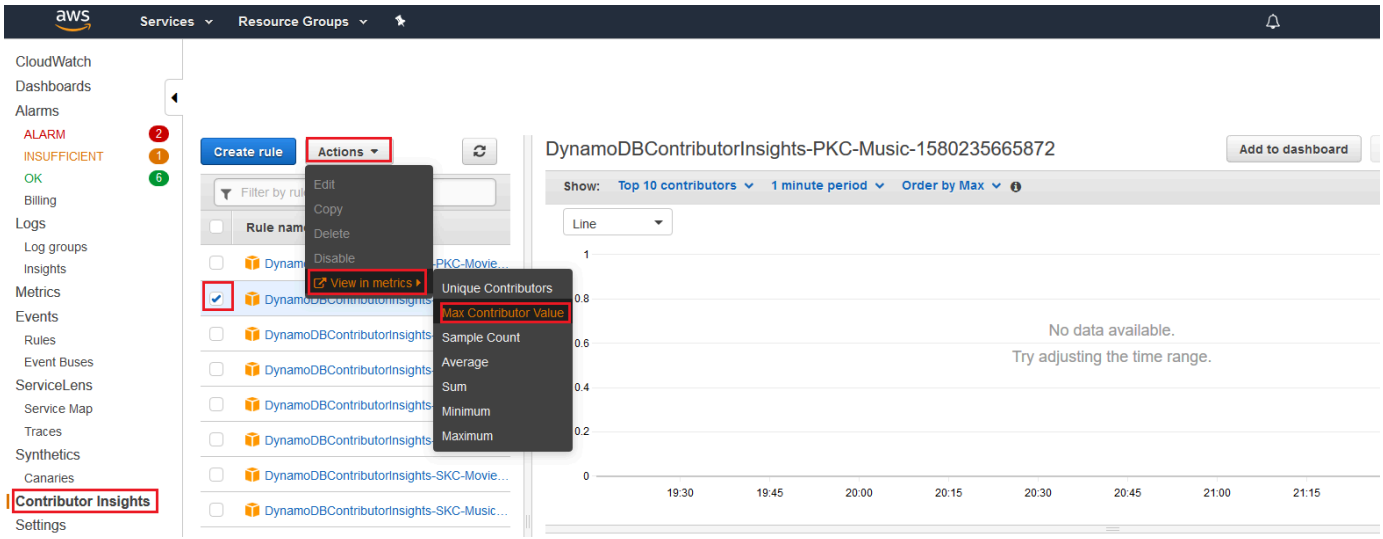
建立 CloudWatch 警示

請遵循下列步驟建立 CloudWatch 警示，並在任何分割區索引鍵耗用超過 50,000 個 ConsumedThroughputUnits 或遇到限流時收到通知。

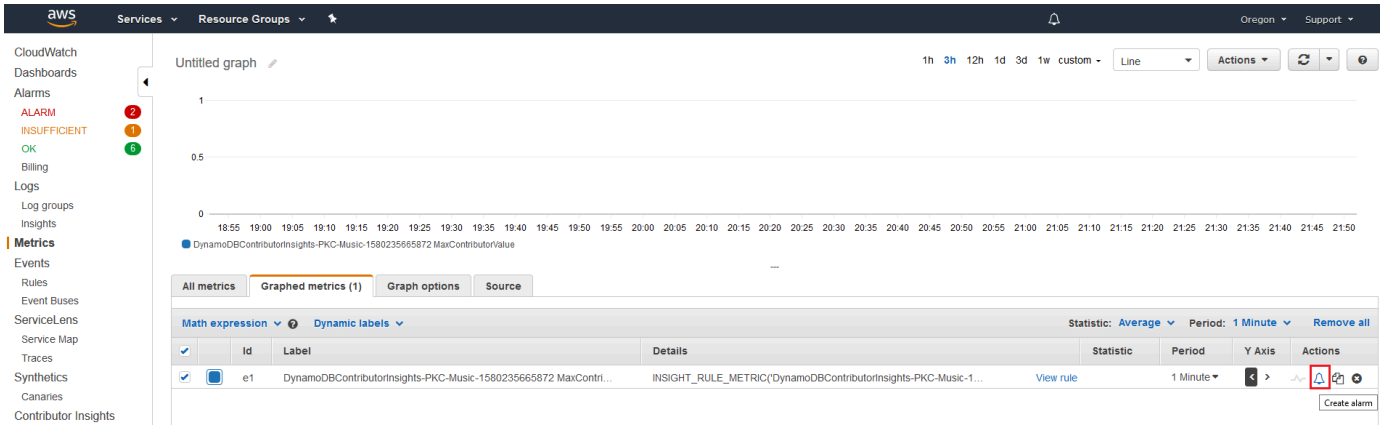
1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台

2. 在主控台左側的導覽窗格中，選擇 Contributor Insights。
3. 根據您的模式和您要監控的內容，選擇適當的規則：
   - 對於存取項目監控 (僅限存取和限流索引鍵模式)：選擇 DynamoDBContributorInsights-PKC-Music
   - 對於限流項目監控 (兩種模式)：選擇 DynamoDBContributorInsights-PKT-Music
4. 選擇 Actions (動作) 下拉式功能表。
5. 選擇 View in metrics (在指標中檢視)。
6. 選擇 Max Contributor Value (作者值上限)。

> **Note**
>
> 只有 `Max Contributor Value` 和 `Maximum` 會傳回有用的統計數字。此清單中的其他統計數字不會傳回有意義的值。

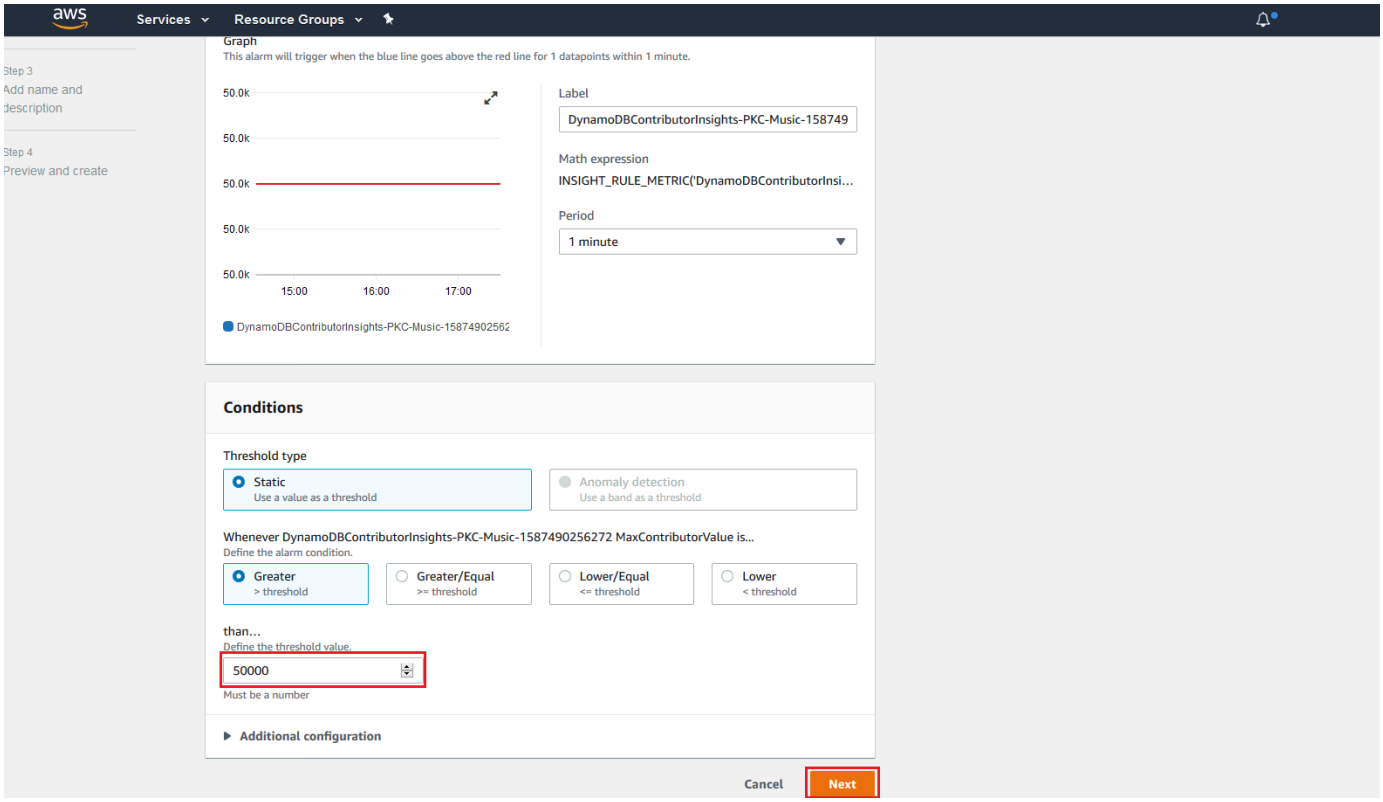


7. 在 Actions (動作) 欄中，選擇 Create Alarm (建立警示)。



8. 輸入適當的閾值並選擇下一步：

   - 對於存取項目 (PKC 規則)：在 `ConsumedThroughputUnits` 輸入 50000
   - 對於限流項目 (PKT 規則)：在 `ThrottleCount` 輸入 1，在有任何限流時收到警示



9. 如需如何設定警示通知的詳細資訊，請參閱使用 Amazon CloudWatch 警示。

使用 Contributor Insights (AWS CLI)

AWS CLI 提供對 Contributor Insights 的程式設計存取，並完整支援這兩種模式。您可以在啟用 Contributor Insights 或稍後切換模式時指定模式。

預設模式的基本操作

使用 Contributor Insights 的預設設定

1. 在具備存取和限流索引鍵模式的 `Music` 基礎資料表上啟用 DynamoDB 專用 CloudWatch Contributor Insights。由於 `ACCESSED_AND_THROTTLED_KEYS` 為預設模式，因此您可以省略 `--contributor-insights-mode=ACCESSED_AND_THROTTLED_KEYS` 參數。

```
aws dynamodb update-contributor-insights \
                    --table-name Music \
                    --contributor-insights-action=ENABLE
```

2. 在 `AlbumTitle-index` 全域次要索引上啟用 DynamoDB 專用 Contributor Insights。

```
aws dynamodb update-contributor-insights \
                    --table-name Music \
                    --index-name AlbumTitle-index \
                    --contributor-insights-action=ENABLE
```

3. 取得 `Music` 資料表及其所有索引的狀態和規則。

```
aws dynamodb describe-contributor-insights
                    --table-name Music
```

   回應將包含顯示 `ACCESSED_AND_THROTTLED_KEYS` 的 `ContributorInsightsMode` 欄位。

4. 列出 `Music` 資料表及其所有索引的狀態。

```
aws dynamodb list-contributor-insights --table-name Music
```

啟用限流索引鍵模式

在限流索引鍵模式中啟用 Contributor Insights

1. 在具備限流索引鍵模式的 `Music` 基礎資料表上啟用 DynamoDB 專用 CloudWatch Contributor Insights。

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

2. 在 AlbumTitle-index 全域次要索引的限流索引鍵模式中啟用 Contributor Insights。

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --index-name AlbumTitle-index \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

3. 透過描述 Contributor Insights 組態來驗證模式。

```
aws dynamodb describe-contributor-insights --table-name Music
```

相較於預設模式，回應會將 ContributorInsightsMode 顯示為 THROTTLED_KEYS 並顯示較少的規則。

在模式之間切換

切換 Contributor Insights 模式

1. 從限流索引鍵模式切換為存取和限流索引鍵模式。

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=ACCESSED_AND_THROTTLED_KEYS
```

2. 從存取和限流索引鍵模式切換為限流索引鍵模式。

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

3. 在轉換期間檢查狀態。

```
aws dynamodb describe-contributor-insights --table-name Music
```

在模式切換期間，會將 ContributorInsightsStatus 顯示為 ENABLING。完成後，將顯示為 ENABLED 的新模式。

管理 Contributor Insights

管理 Contributor Insights 設定

1. 在 AlbumTitle-index 全域次要索引上停用 DynamoDB 專用 CloudWatch Contributor Insights。

```
aws dynamodb update-contributor-insights \
                    --table-name Music --index-name AlbumTitle-index \
                    --contributor-insights-action=DISABLE
```

2. 列出您帳戶中的所有 Contributor Insights 組態。

```
aws dynamodb list-contributor-insights
```

將顯示已啟用 Contributor Insights 的所有資料表和索引，以及其模式。

3. 取得特定組態的詳細資訊。

```
aws dynamodb describe-contributor-insights \
                    --table-name Music \
                    --index-name AlbumTitle-index
```

回應範例

以下是顯示模式之間差異的回應範例：

存取和限流索引鍵模式回應

```
{
    "TableName": "Music",
    "ContributorInsightsRuleList": [
        "DynamoDBContributorInsights-PKC-Music-1234567890123",
        "DynamoDBContributorInsights-PKT-Music-1234567890123",
```

```
        "DynamoDBContributorInsights-SKC-Music-1234567890123",
        "DynamoDBContributorInsights-SKT-Music-1234567890123"
    ],
    "ContributorInsightsStatus": "ENABLED",
    "ContributorInsightsMode": "ACCESSED_AND_THROTTLED_KEYS",
    "LastUpdateDateTime": "2024-01-15T10:30:00.000Z"
}
```

限流索引鍵模式回應

```
{
    "TableName": "Music",
    "ContributorInsightsRuleList": [
        "DynamoDBContributorInsights-PKT-Music-1234567890123",
        "DynamoDBContributorInsights-SKT-Music-1234567890123"
    ],
    "ContributorInsightsStatus": "ENABLED",
    "ContributorInsightsMode": "THROTTLED_KEYS",
    "LastUpdateDateTime": "2024-01-15T10:35:00.000Z"
}
```

請注意，限流索引鍵模式的規則較少 (僅 PKT 和 SKT)，對應於更集中的監控。

將 IAM 與 DynamoDB 專用 CloudWatch Contributor Insights 搭配使用

您第一次為 Amazon DynamoDB 啟用 Amazon CloudWatch Contributor Insights 時，DynamoDB 會自動為您建立 AWS Identity and Access Management (IAM) 服務連結角色。 DynamoDB 此角色 `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` 會允許 DynamoDB 代表您管理 CloudWatch Contributor Insights 規則。請勿刪除此服務連結角色。如果刪除，您所有管理的角色都不會再於您刪除資料表或全域次要索引時清理。

如需服務連結角色的詳細資訊，請參閱《IAM 使用者指南》中的使用服務連結角色。

需要以下許可：

- 若要啟用或停用 DynamoDB 專用 CloudWatch Contributor Insights，您必須有資料表或索引的 `dynamodb:UpdateContributorInsights` 許可。
- 若要檢視 DynamoDB 專用 CloudWatch Contributor Insights 圖表，您必須有 `cloudwatch:GetInsightRuleReport` 許可。
- 若要針對某個特定 DynamoDB 資料表或索引描述 DynamoDB 專用 CloudWatch Contributor Insights，您必須有 `dynamodb:DescribeContributorInsights` 許可。

- 若要針對每個資料表和全域次要索引列出 DynamoDB 專用 CloudWatch Contributor Insights 狀態，您必須有 `dynamodb:ListContributorInsights` 許可。

範例：啟用或停用 DynamoDB 專用 CloudWatch Contributor Insights

下列 IAM 政策會授予啟用或停用 DynamoDB 專用 CloudWatch Contributor Insights 的許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {"StringLike": {"iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        }
    ]
}
```

對於由 KMS 金鑰加密的資料表，使用者必須擁有 kms:Decrypt 許可，才能更新 Contributor Insights。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        },
        {
            "Effect": "Allow",
            "Resource": "arn:aws:kms:*:*:key/*",
            "Action": [
                "kms:Decrypt"
            ]
        }
    ]
}
```

範例：擷取 CloudWatch Contributor Insights 規則報告

下列 IAM 政策授予擷取 CloudWatch Contributor Insights 規則報告的許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport"
```

```
            ],
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/
DynamoDBContributorInsights*"
        }
    ]
}
```

範例：根據資源選擇性地套用 DynamoDB 許可專用 CloudWatch Contributor Insights

下列 IAM 政策會授予允許 `ListContributorInsights` 和 `DescribeContributorInsights` 動作的許可，並拒絕特定全域次要索引的 `UpdateContributorInsights` 動作。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListContributorInsights",
                "dynamodb:DescribeContributorInsights"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books/
index/Author-index"
        }
    ]
}
```

使用 DynamoDB 專用 CloudWatch Contributor Insights 的服務連結角色

適用於 DynamoDB 的 CloudWatch Contributor Insights 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是一種獨特的 IAM 角色類型，可直接連結至 DynamoDB 專用

CloudWatch Contributor Insights。CloudWatch Contributor Insights for DynamoDB 會預先定義服務連結角色，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓設定 DynamoDB 專用 CloudWatch Contributor Insights 更為簡單，因為您不必手動新增必要的許可。DynamoDB 專用 CloudWatch Contributor Insights 會定義其服務連結角色的許可，除非另有定義，否則僅有 DynamoDB 專用 CloudWatch Contributor Insights 可以擔任其角色。定義的許可包括信任政策和許可政策，並且該許可政策不能連接到任何其他 IAM 實體。

如需關於支援服務連結角色的其他服務的資訊，請參閱可搭配 IAM 運作的AWS 服務，並尋找 Service-Linked Role (服務連結角色) 欄顯示為 Yes (是) 的服務。選擇具有連結的 Yes (是)，以檢視該服務的服務連結角色文件。

DynamoDB 專用 CloudWatch Contributor Insights 的服務連結角色許可

DynamoDB 專用 CloudWatch Contributor Insights 會使用名為 AWSServiceRoleForDynamoDBCloudWatchContributorInsights 的服務連結角色。服務連結角色的目的是讓 Amazon DynamoDB 代表您管理針對 DynamoDB 資料表和全域次要索引建立的 Amazon CloudWatch Contributor Insights 規則。

`AWSServiceRoleForDynamoDBCloudWatchContributorInsights` 服務連結角色信任下列服務以擔任角色：

- `contributorinsights.dynamodb.amazonaws.com`

此角色許可政策允許 DynamoDB 專用 CloudWatch Contributor Insights 對指定資源完成下列動作：

- 動作：`DynamoDBContributorInsights` 上的 `Create and manage Insight Rules`

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的服務連結角色許可。

建立 DynamoDB 專用 CloudWatch Contributor Insights 的服務連結角色

您不需要手動建立服務連結角色，當您在 AWS 管理主控台、 AWS CLI或 AWS API 中啟用 Contributor Insights 時，DynamoDB 的 CloudWatch Contributor Insights 會為您建立服務連結角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。在啟用 Contributor Insights 時，DynamoDB 專用 CloudWatch Contributor Insights 會再次為您建立服務連結角色。

編輯 DynamoDB 專用 CloudWatch Contributor Insights 的服務連結角色

DynamoDB 專用 CloudWatch Contributor Insights 不允許您編輯 `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

刪除 DynamoDB 專用 CloudWatch Contributor Insights 的服務連結角色

您不需要手動刪除 `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` 角色。當您在 AWS 管理主控台、 AWS CLI或 AWS API 中停用 Contributor Insights 時，適用於 DynamoDB 的 CloudWatch Contributor Insights 會清除資源。

您也可以使用 IAM 主控台、 AWS CLI 或 AWS API 手動刪除服務連結角色。若要執行此操作，您必須先手動清除服務連結角色的資源，然後才能手動刪除它。

> **Note**
>
> 若 DynamoDB 專用 CloudWatch Contributor Insights 服務在您試圖刪除資源時正在使用該角色，刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

使用 IAM 手動刪除服務連結角色

使用 IAM 主控台 AWS CLI、 或 AWS API 來刪除`AWSServiceRoleForDynamoDBCloudWatchContributorInsights`服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

使用 DynamoDB 進行設計和架構的最佳實務

使用本節可快速查閱在使用 DynamoDB 時，提升效能並降低輸送量成本的建議。

主題

- DynamoDB 的 NoSQL 設計
- 使用 DynamoDB Well-Architected Lens 來最佳化您的 DynamoDB 工作負載
- 在 DynamoDB 中設計與有效運用分割區索引鍵的最佳實務
- 使用排序索引鍵組織 DynamoDB 資料的最佳實務
- 使用 DynamoDB 中次要索引的最佳實務
- 在 DynamoDB 中儲存大型項目與屬性的最佳實務
- 在 DynamoDB 中處理時間序列資料的最佳實務
- 管理 DynamoDB 資料表中多對多關係的最佳實務
- 在 DynamoDB 中查詢與掃描資料的最佳實務
- DynamoDB 資料表設計的最佳實務
- 使用 DynamoDB 全域資料表
- 在 DynamoDB 中管理控制平面的最佳實務
- 在 DynamoDB 中使用大量資料操作的最佳實務
- 在 DynamoDB 中處理並行更新的最佳實務
- 了解 DynamoDB 中 AWS 帳單和用量報告的最佳實務
- 將 DynamoDB 資料表從一個帳戶移轉至另一帳戶
- 整合 DAX 與 DynamoDB 應用程式的設計指南

- 使用 AWS PrivateLink for Amazon DynamoDB 時的考量事項

DynamoDB 的 NoSQL 設計

如 Amazon DynamoDB 這類的 NoSQL 資料庫會使用備用模型來管理資料，例如鍵值的配對或文件儲存。當您從關聯式資料庫管理系統切換至如 DynamoDB 這類的 NoSQL 資料庫時，請務必了解主要差異和特定設計方法。

主題

- 關聯式資料設計與 NoSQL 間的差異
- NoSQL 設計的兩個主要概念
- NoSQL 設計方法
- DynamoDB 專用 NoSQL Workbench

關聯式資料設計與 NoSQL 間的差異

關聯式資料庫管理系統 (RDBMS) 和 NoSQL 資料庫各有優劣：

- RDBMS 可以彈性地查詢資料，但查詢相對昂貴，而且在高流量的狀況下擴展不易 (請參閱 在 DynamoDB 中製作關聯式資料模型的第一步)。
- 在如 DynamoDB 這類的 NoSQL 資料庫中，可以少數方式有效地查詢資料，但在外部的查詢就非常昂貴而且速度緩慢。

這些差異導致兩種系統之間的資料庫設計不同：

- 在 RDBMS 中，您會針對靈活性而進行設計，而不需擔心實行詳細資訊或效能。查詢最佳化通常不會影響結構描述設計，但標準化相當重要。
- 在 DynamoDB 中，您會特別設計結構描述，盡可能以最快最便宜的方式進行常用與重要的查詢。系統會打造您的資料結構，以符合企業使用案例的特定要求。

NoSQL 設計的兩個主要概念

NoSQL 設計思維與 RDBMS 設計不同。針對 RDBMS，您可以直接建立標準化的資料模型，而不需考量存取模式。您可以在稍後有新問題與查詢要求時擴展此模型。您可以將每種資料整理至其資料表。

NoSQL 設計的不同之處

- 相反地，針對 DynamoDB，您不應開始設計結構描述，除非您知道其將需要回答的問題。必須事先了解企業問題和應用程式使用案例。
- 您在 DynamoDB 應用程式中維護的資料表應越少越好。擁有較少的資料表可讓事物更具擴展性，需要較少的許可管理，並減少 DynamoDB 應用程式的額外負荷。它還可以幫助降低整體備份成本。

NoSQL 設計方法

設計 DynamoDB 應用程式的第一步即是辨識系統必須滿足的特定查詢模式。

特別是，請務必了解應用程式存取模式的三種基本屬性，再開始進行：

- 資料大小：了解資料一次儲存與要求的方式，是協助判斷分割資料的最有效方法。
- 資料形狀：NoSQL 資料庫會組織資料，而非在資料處理時重新改造資料 (如 RDBMS 系統)，如此其在資料庫中的狀態就會與將受到查詢的項目相對應。此為提升速度與可擴展性的關鍵因素。
- 資料速度：DynamoDB 會隨著增加程序查詢可用的實體分割區數與有效地在這些分割區間散佈資料來擴展。事先了解尖峰查詢負載，將可能有助於判斷如何分割資料以有效使用輸入/輸出容量。

在您識別特定查詢要求後，您可以根據管理效能的一般原則來組織資料：

- 將相關的資料保持在一起。 研究顯示，「參照區域性」原則 (將相關資料集中存放) 是提升 NoSQL 系統效能與回應時間的關鍵，同樣在早期最佳化路由表時也被視為重要因素。

  作為一般規則，您在 DynamoDB 應用程式中維護的資料表應越少越好。

  例外狀況包含大量時間序列資料涉及其中的案例，或存取模式極為不同的資料集。含反轉索引的單一資料表通常可以啟用簡單查詢，來建立並擷取您應用程式所需的複雜階層資料結構。
- 使用排序。 若相關項目的索引鍵設計為能一起排序，則可以一起分組，以更有效地進行查詢。此為重要的 NoSQL 設計策略。
- 發布佇列。 也應避免將大量查詢集中於資料庫的單一部分，以免超出 I/O 容量。相反地，應盡可能設計資料索引鍵，使流量平均分散至各分割區，以避免產生熱點。
- 使用全域次要索引。 透過建立特定全域次要索引，您可以啟用主要資料表可支援的不同查詢，這能維持速度且費用相對便宜。

這些一般原則會轉換為常見設計模式，讓您可以在 DynamoDB 使用它們來有效打造資料模型。

DynamoDB 專用 NoSQL Workbench

DynamoDB 專用 NoSQL Workbench 是跨平台用戶端 GUI 應用程式，可用於現代資料庫開發和操作。適用於 Windows、macOS 和 Linux。NoSQL Workbench 是視覺化開發工具，提供了資料模型建立、資料視覺化、範例資料產生和查詢開發功能，協助您設計、建立、查詢及管理 DynamoDB 資料表。借助 DynamoDB 專用 NoSQL Workbench，您可以使用滿足您應用程式資料存取模式的現有資料模型，來建置新的資料模型或設計模型。您也可以在程序結束時，匯入及匯出設計好的資料模型。如需詳細資訊，請參閱使用 NoSQL Workbench 建立資料模型。

使用 DynamoDB Well-Architected Lens 來最佳化您的 DynamoDB 工作負載

本節描述 Amazon DynamoDB Well-Architected Lens，其收集了用於設計架構良好 DynamoDB 工作負載的設計原則和指引。

最佳化 DynamoDB 資料表上的成本

本節涵蓋如何為現有 DynamoDB 資料表最佳化成本的最佳實務。您應查看以下策略，了解哪一項成本最佳化策略最適合您的需求，並反覆採用這些策略。每個策略都會概述可能會影響您成本的原因、應查看的跡象，以及如何降低成本的規範式引導。

主題

- 在資料表層級評估您的成本
- 評估 DynamoDB 資料表的容量模式
- 評估您的 DynamoDB 資料表自動擴展設定
- 評估您的 DynamoDB 資料表類別選擇
- 識別您在 DynamoDB 中未使用的資源
- 評估您的 DynamoDB 資料表使用模式
- 評估您的 DynamoDB Streams 使用情況
- 評估您的 DynamoDB 資料表佈建容量是否合適

在資料表層級評估您的成本

在 中找到的 Cost Explorer 工具 AWS 管理主控台 可讓您查看依類型細分的成本，例如讀取、寫入、儲存和備份費用。您還可以查看依期間 (例如月份或日期) 彙總的這些成本。

管理員面臨的挑戰之一，在於只需要檢閱個別特定資料表成本的情況。部分資料可透過 DynamoDB 主控台或呼叫 `DescribeTable` API 取得，但 Cost Explorer 依預設不允許按照特定資料表相關的成本進行篩選或分組。本節將說明如何使用標記功能，在 Cost Explorer 中執行個別資料表的成本分析。

主題

- 如何檢視單一 DynamoDB 資料表的成本
- Cost Explorer 的預設檢視
- 如何在 Cost Explorer 中使用和套用資料表標籤

如何檢視單一 DynamoDB 資料表的成本

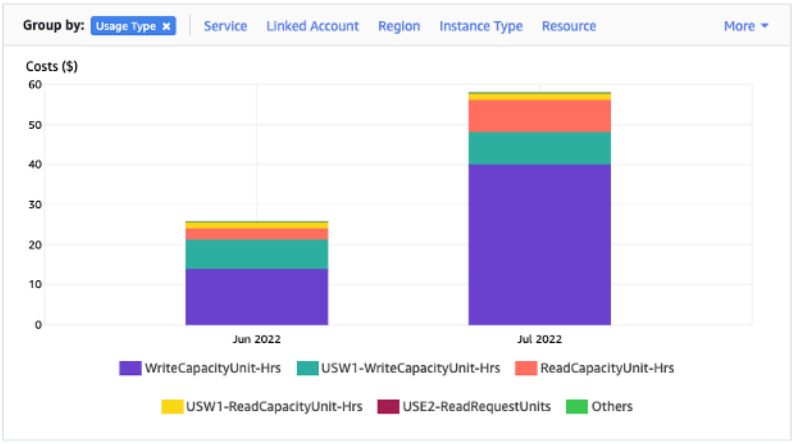
Amazon DynamoDB AWS 管理主控台 和 `DescribeTable` API 都會顯示單一資料表的相關資訊，包括主索引鍵結構描述、資料表上的任何索引，以及資料表和任何索引的大小和項目計數。將資料表的大小加上索引的大小，即可計算資料表的每月儲存成本。例如，在 us-east-1 區域中，每 GB 為 0.25 美元。

如果資料表處於佈建容量模式，也會傳回目前的 RCU 和 WCU 設定值。這些資訊可用於計算資料表目前的讀取和寫入成本，但這些成本可能變動，尤其是在設定了資料表的 Auto Scaling 功能後。

> **Note**
>
> 如果資料表處於隨需容量模式，則 `DescribeTable` 對於估計輸送量成本將沒有幫助，因為這些成本是根據任一期間內的實際用量而非佈建用量進行計費。

Cost Explorer 的預設檢視

Cost Explorer 的預設檢視會提供圖表以顯示耗用資源 (例如輸送量和儲存體) 的成本。您可以選擇按期間分組成本，例如按月或按日總計。儲存、讀取、寫入和其他功能的成本也可以進行細分和比較。



如何在 Cost Explorer 中使用和套用資料表標籤

根據預設，Cost Explorer 不會提供任何一個特定資料表的成本摘要，因為它會將多個資料表的成本合併為一項總數。不過，您可以使用 AWS 資源標記，以中繼資料標籤來識別各個資料表。標籤是一種鍵值對，可用於各種用途，例如找出所有屬於特定專案或部門的資源。在此範例中，我們假設您有一個名為「MyTable」的資料表。

1. 設定一項具有「table_name」索引鍵和「MyTable」值的標籤。
2. 在 Cost Explorer 中啟用此標籤，然後篩選標籤值，以深入了解各個資料表的成本。

> **Note**
>
> 標籤可能需要一到兩天的時間才會開始出現在 Cost Explorer 中

您可以在 主控台中自行設定中繼資料標籤，或透過自動化，例如 CLI AWS 或 AWS SDK。請考慮在組織的新標籤建立程序中要求設定 table_name 標籤。對於現有的資料表，可使用一項 Python 公用程式來尋找這些標記，並將這些標記套用到您帳戶中特定區域內全部現有的資料表。如需詳細資訊，請參閱 GitHub 上的 Eponymous Table Tagger (同名資料表標記工具)。

評估 DynamoDB 資料表的容量模式

本節概述如何為 DynamoDB 資料表選取適當的容量模式。每種模式都經過調整，以滿足不同工作負載在回應輸送量變化以及該用量計費方式方面的需求。在做出決策時，必須平衡考量這些因素。

主題

- 有哪些可用的資料表容量模式
- 何時選取隨需容量模式
- 何時選取佈建容量模式
- 選擇資料表容量模式時應考慮的其他因素

有哪些可用的資料表容量模式

建立 DynamoDB 資料表時，您必須選取隨需或佈建容量模式。

您可在 24 小時滾動期間內，最多將資料表從佈建容量模式切換至隨需模式四次。您可隨時將資料表從隨需模式切換回佈建容量模式。

隨需容量模式

隨需容量模式旨在讓使用者無須規劃或佈建 DynamoDB 資料表容量。在此模式下，資料表不需擴充或縮減任何資源，就能立即容納資料表收到的請求 (最多可達到先前資料表尖峰輸送量的兩倍)。

DynamoDB 隨需模式提供按請求付費的定價機制，讓您僅需為實際用量付費。

佈建容量模式

佈建容量模式是一種較傳統的模式，使用者必須直接或透過自動擴展功能，定義資料表可用的請求容量。由於在任一時間點都會為資料表佈建固定容量，因此計費依據為總佈建容量，而非實際請求次數。超過配置的容量也可能導致資料表拒絕請求，並降低應用程式使用者的體驗。

佈建容量模式需持續監控，以在避免過度或不足佈建間取得平衡，確保限流最小化並最佳化成本。

何時選取隨需容量模式

在進行成本最佳化時，若您的工作負載與下列圖表所示相似，隨需模式即為最佳選擇。

下列因素會造成此類型的工作負載：

- 隨時間變化的流量模式
- 變動的請求量 (由批次工作負載導致)
- 無法預測的請求時機 (導致流量尖峰)
- 在特定時段流量降至零或低於峰值的 30%

對於具上述特性的工作負載，若透過自動擴展維持足夠容量以應對流量尖峰，可能導致資料表過度佈建、成本增加，或因佈建不足造成不必要的限流。隨需容量模式更為理想，因其可自動應對流量波動，無需預測或手動調整容量。

採用隨需模式的「按請求付費」定價模型時，您無需擔心閒置容量，因為僅需為實際使用的輸送量付費。系統會依每次讀取或寫入請求向您收費，因此成本能直接反映實際用量，讓您更容易在成本與效能之間取得平衡。您也可以選擇為各個隨需資料表與全域次要索引設定每秒最大讀取或寫入 (或兩者) 輸送量，以協助控制成本與用量界限。如需詳細資訊，請參閱隨需資料表的最大輸送量。

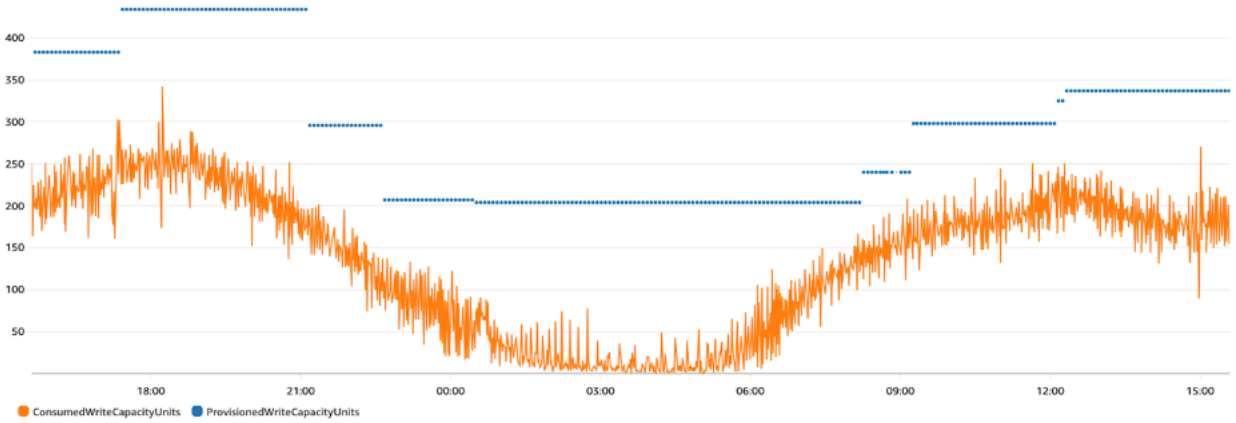
何時選取佈建容量模式

對於佈建容量模式而言，理想的工作負載應具備穩定且可預測的使用模式，如下圖所示。

> **Note**
>
> 我們建議在對佈建容量進行調整前，先檢視細分期間 (例如 14 天) 的指標。

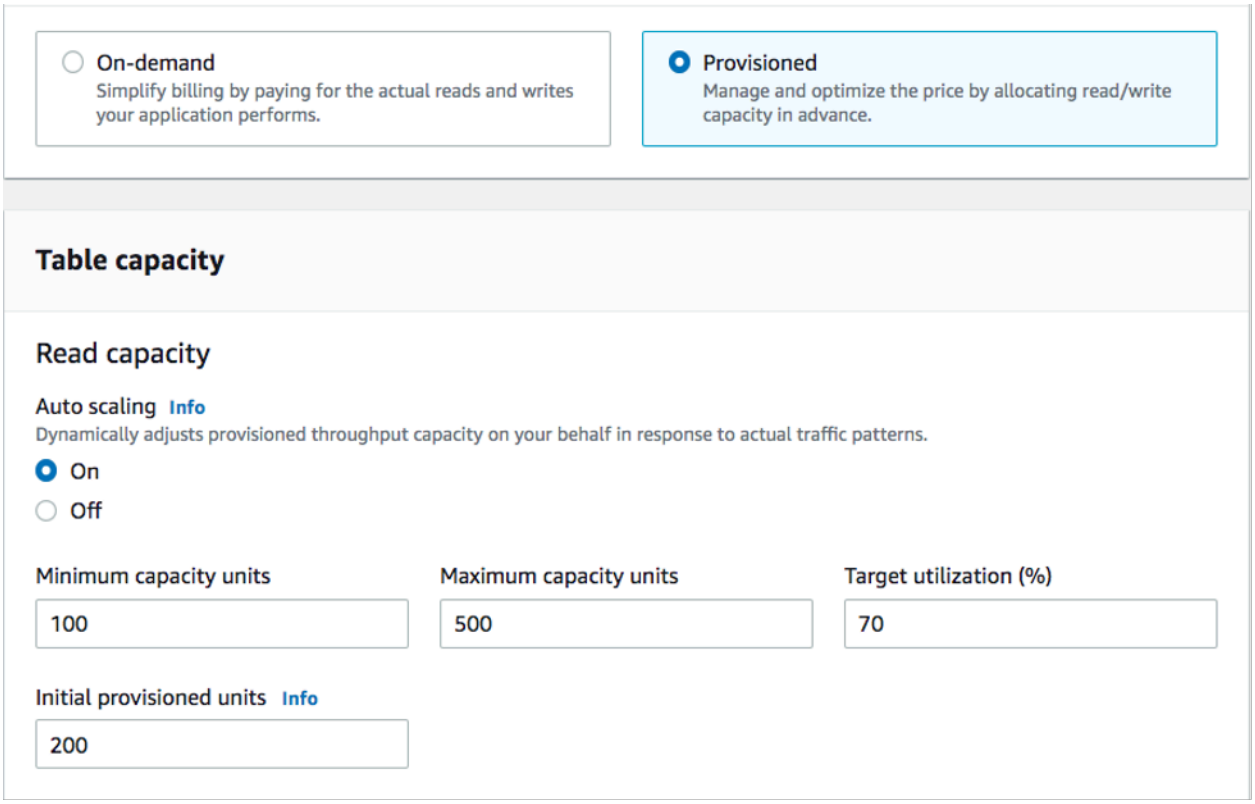
下列因素會造成此類型的工作負載：

- 在特定時段 (如每小時或每日) 的穩定、可預測且具循環性的流量
- 限制性的短期流量高峰



由於在特定時段內流量相對穩定，因此可將資料表的佈建容量設定得接近實際消耗容量。基本上，佈建容量資料表的成本最佳化必須藉由實際操作，盡量使佈建容量 (藍線) 接近實際使用容量 (橘線)，同時不增加資料表上的 `ThrottledRequests`。兩條線之間的空間表示浪費的容量，但同時也是預防因限流而影響使用者體驗的保障。若您能預測應用程式的輸送量需求，並希望透過控制讀寫容量維持成本的可預測性，建議持續使用佈建資料表。

DynamoDB 為佈建容量資料表提供自動擴展功能，將代表您自動平衡佈建容量。如此可讓您追蹤全天的使用容量，並根據少數幾項變數來設定資料表的容量。使用自動擴展時，資料表可能出現過度佈建，您需微調限流事件數與過度佈建容量單位的比例，以符合工作負載需求。



容量單位下限

您可以設定資料表的容量下限，藉此限制限流的程度，但這麼做不會降低資料表的成本。如果資料表會在低用量期間之後出現突發高用量，則可以設定用量下限以預防自動擴展功能將資料表容量設定得過低。

容量單位上限

您可以設定資料表的容量上限，以避免資料表的規模調整到高於預期的程度。如果不需要進行大規模負載測試，請考慮為開發或測試資料表套用上限。您可以為任何資料表設定上限，但是在生產環境中使用時，請務必定期依據資料表基準評估此設定，以預防意外限流的狀況。

目標使用率

對於佈建容量資料表，設定資料表的目標使用率是實現成本最佳化的主要方法。對此設定較低的百分比值，將會增加資料表過度佈建程度而提高成本，但同時降低限流的風險。對此設定較高的百分比值，將會降低資料表過度佈建程度，但同時提高限流的風險。

選擇資料表容量模式時應考慮的其他因素

在兩種模式之間進行抉擇時，還有一些其他因素值得納入考量。

佈建容量使用率

若要判斷隨需模式的成本是否低於佈建容量，建議檢視佈建容量使用率，該指標反映已配置 (或「佈建」) 資源的使用效率。若工作負載的平均佈建容量使用率低於 35%，隨需模式的成本通常更低。在許多情況下，即使佈建容量使用率高於 35%，若工作負載包含低活動期與偶發高峰，採用隨需模式仍可能更具成本效益。

預留容量

對於佈建容量資料表，DynamoDB 提供預留容量的購買功能，適用於讀取和寫入容量 (目前不適用於複寫寫入容量單位 (rWCU) 和標準 - IA 資料表)。預留容量可比標準佈建容量定價享有更高折扣。

在兩種資料表模式之間進行抉擇時，請考慮這項額外的折扣因素對資料表成本的影響程度。在部分情況下，即使是難以預測的工作負載，若於具預留容量的過度佈建資料表上執行，成本仍可能更低。

改善工作負載的可預測性

在某些情況下，工作負載可能顯得同時具有可預測和無法預測的模式。雖然隨需資料表可以輕鬆支援這種情況，但若可改善無法預測的工作負載模式，可能會有助於提高成本效益。

造成這些模式的最常見原因之一是批次匯入。這種類型的流量可能經常超過資料表的基準容量，導致執行時期間發生限流。若要在佈建容量資料表上執行這類工作負載，請考慮下列選項：

- 如果批次發生在排程時間，則可排程在其執行之前增加自動擴展的容量。
- 如果批次是隨機發生，請考慮嘗試延長執行時期，而不是盡速執行
- 為匯入程序新增「逐步提升期」，使匯入速率從低速開始，並在數分鐘內逐漸增加，直到自動擴展開始調整資料表容量。

評估您的 DynamoDB 資料表自動擴展設定

本節概述如何評估 DynamoDB 資料表上的自動擴展設定。Amazon DynamoDB 自動擴展是一款功能，可根據應用程式流量和目標使用率指標，管理資料表和全域次要索引 (GSI) 輸送量。如此可確保您的資料表或 GSI 具有應用程式模式所需的容量。

AWS 自動擴展服務將監控您目前的資料表使用率，並將其與目標使用率值進行比較：`TargetValue`。若需增加或減少分配的容量，您會收到通知。

主題

- 了解您的自動擴展設定
- 如何識別目標使用率低的資料表 (<=50%)
- 如何處理具有季節性差異的工作負載

- 如何處理具有未知模式的尖峰工作負載
- 如何處理具有連結應用程式的工作負載

了解您的自動擴展設定

您的營運團隊需定義目標使用率的正確值、初始步驟和最終值。這可讓您根據歷史應用程式使用情況來定義值，用來觸發 AWS 自動擴展政策。使用率目標是套用自動擴展規則之前的一段時間內，需要達成的總容量百分比。

當您設定高使用率目標 (目標約 90%) 時，表示您的流量須在一段時間內高於 90%，才會啟用自動擴展。除非您的應用程式狀態穩定，且不會接收到流量尖峰，否則不應使用高使用率目標。

當您設定非常低使用率 (目標小於 50%) 時，表示您的應用程式須達到佈建容量的 50%，才會觸發自動擴展政策。除非您的應用程式流量成長快速，否則這通常會變成未使用容量和資源浪費。

如何識別目標使用率低的資料表 (<=50%)

您可以使用 AWS CLI 或 AWS 管理主控台 來監控和識別 DynamoDB 資源中自動擴展政策`TargetValues`的 ：

AWS CLI

1. 執行下列命令，傳回完整的資源清單：

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb
   ```

   此命令將傳回發給所有 DynamoDB 資源的自動擴展政策完整清單。若您只想從特定資料表擷取資源，您可以新增 `–resource-id parameter`。例如：

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb --resource-id "table/<table-name>”
   ```

2. 執行下列命令，僅傳回特定 GSI 的自動擴展政策

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb --resource-id "table/<table-name>/index/<gsi-name>”
   ```

   我們對自動擴展政策感興趣的值於下面強調顯示。我們希望確保目標值大於 50%，以避免過度佈建。您應該會獲得類似以下的結果：

```
{
    "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"
        }
    ]
```

```
}
```

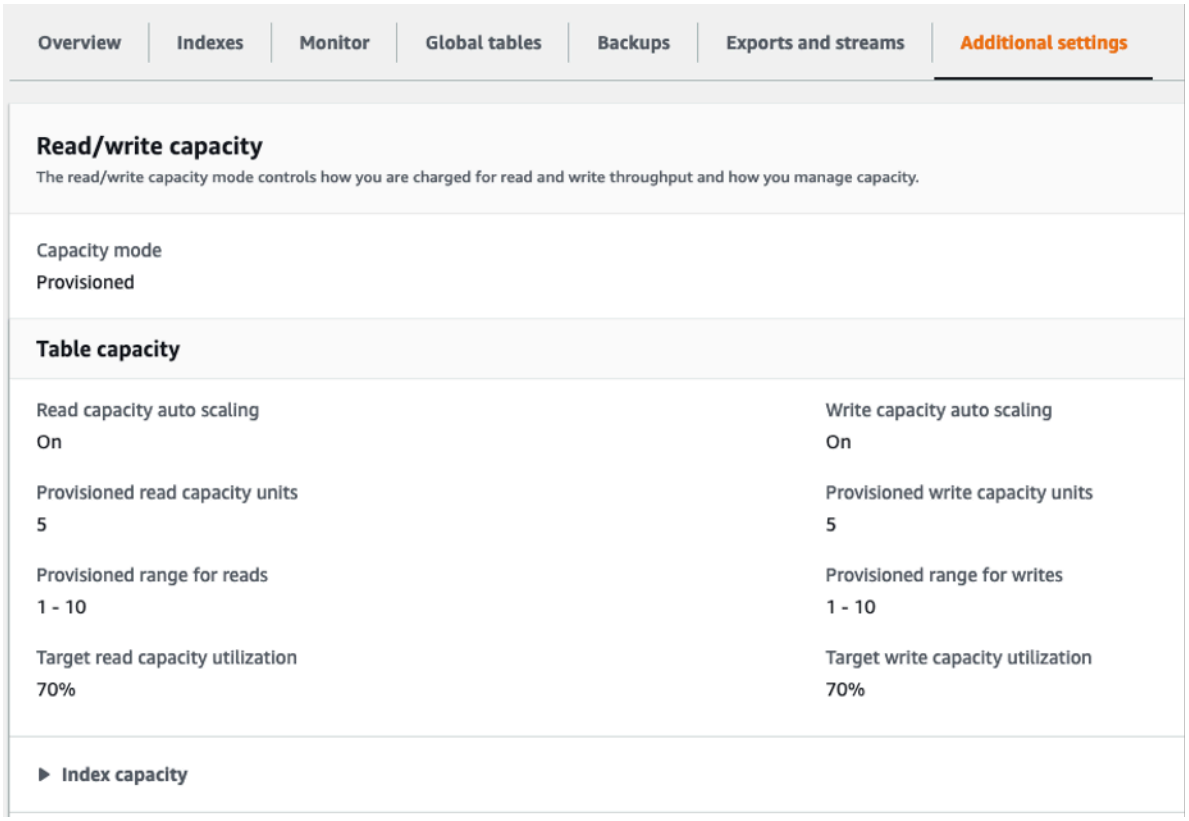
AWS 管理主控台

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。

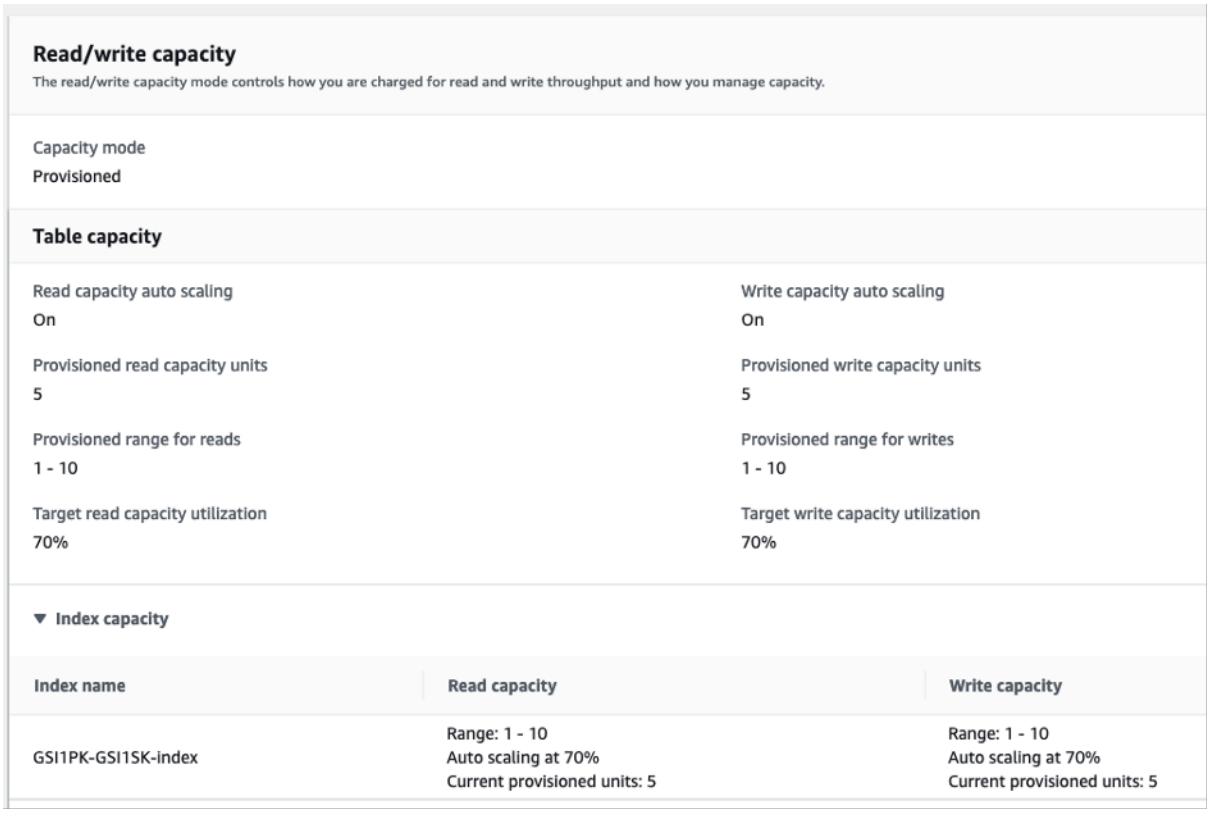
   AWS 區域 如有必要，請選取適當的 。

2. 在左側導覽列中，選取 Tables (資料表)。在 Tables (資料表) 頁面上，選取資料表 Name (名稱)。

3. 在資料表詳細資訊，選取其他設定，然後檢閱資料表的自動擴展設定。



對於索引，展開索引容量區段以檢視索引的自動擴展設定。



若您的目標使用率值小於或等於 50%，您應探索資料表使用率指標，查看指標是否佈建不足或過度佈建。

如何處理具有季節性差異的工作負載

請考慮下列情況：您的應用程式大部分時間都以最小平均值運作，但使用率目標很低，因此應用程式可以快速回應特定時間發生的事件，且您擁有足夠容量避免受到限流。應用程式在正常辦公時間 (上午 9 點至下午 5 點) 非常忙碌，但下班時間僅在基礎層級運作時，這種情況就很常見。由於部分使用者在上午 9 點前開始連線，因此應用程式會使用此低閾值快速提升，以便在尖峰時段達到所需容量。

此情況可能如下所示：

- 下午 5 點至上午 9 點之間，`ConsumedWriteCapacity` 單位停留在 90 和 100 之間
- 使用者在上午 9 點前開始連線到應用程式，且容量單位大幅增加 (您看到的最大值為 1500 WCU)
- 平均而言，在工作期間，應用程式使用量介於 800 到 1200

若上一個情況適用於您，請考慮使用排程的自動擴展，您的資料表仍可設定應用程式自動擴展規則，但因目標使用率較低，只會在您需要的特定間隔佈建額外容量。

您可以使用 AWS CLI 來執行下列步驟，以建立排程的自動擴展規則，該規則將根據一天中的時間和一週中的一天執行。

1. 將您的 DynamoDB 資料表或 GSI 向 Application Auto Scaling註冊為可擴展的目標。可擴展的目標是 Application Auto Scaling 可橫向擴充和縮減的資源。

```
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. 根據您的需求設定排定的動作。

   我們需要兩個規則來處理此情況：一個用於向上擴展，另一個用於縮減規模。第一個規則向上擴展排程的動作：

```
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"
```

   第二個規則縮減規模排程的動作：

```
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"
```

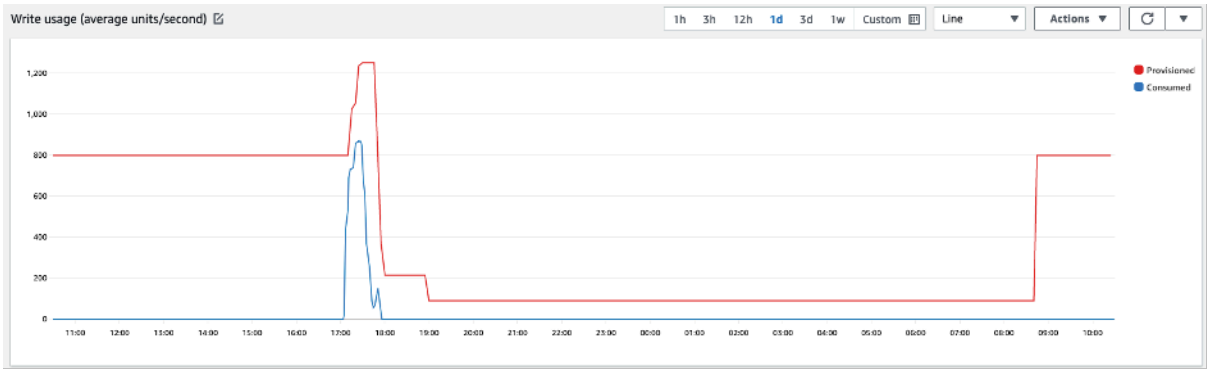
3. 執行以下命令，驗證這兩個規則皆已啟用。

```
aws application-autoscaling describe-scheduled-actions --service-namespace dynamodb
```

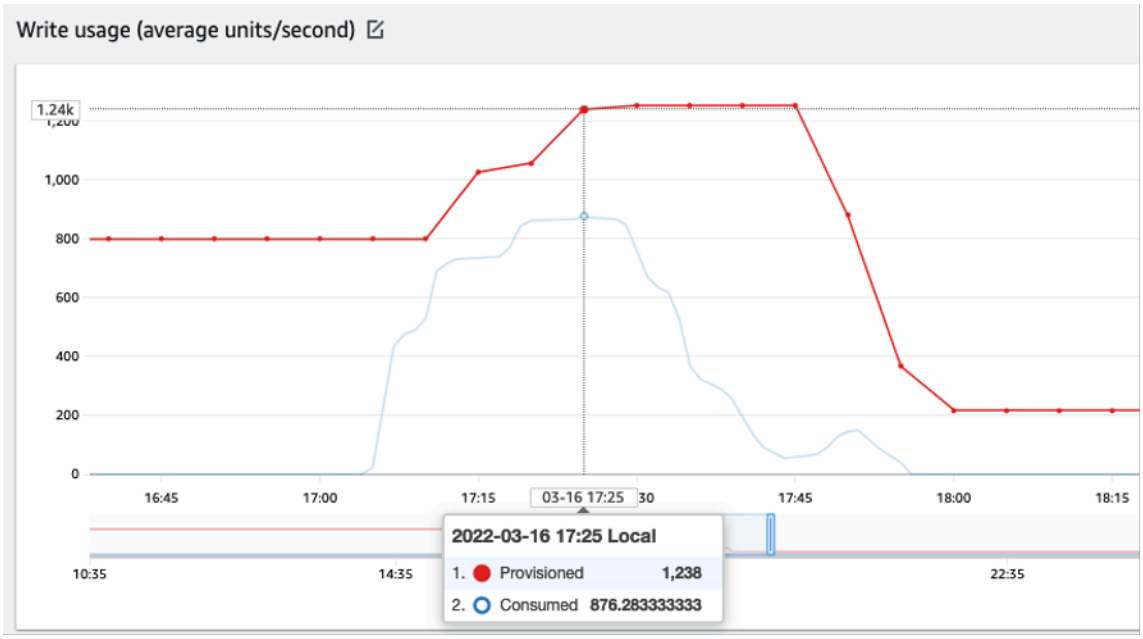
您應該會取得如下結果：

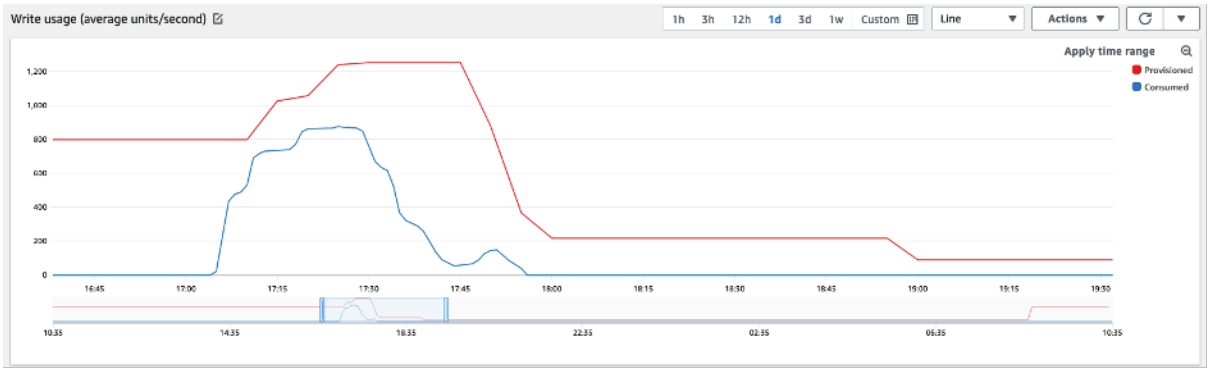
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-5-8-scheduled-down-action",
            "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"
        }
    ]
}
```

下圖顯示一律保持 70% 目標使用率的範例工作負載。請注意持續套用自動擴展規則的方式，同時輸送量不會降低。



放大觀察可見應用程式出現尖峰，觸發了 70% 的自動擴展閾值，促使系統啟動自動擴展機制，以提供資料表所需的額外容量。排程的自動擴展動作會影響最大值與最小值，您須設定這些值。





如何處理具有未知模式的尖峰工作負載

在此情況中，應用程式會使用非常低的使用率目標，因為您仍不確定應用程式模式，同時要確保工作負載不受限流。

請考慮改用隨需容量模式。隨需資料表非常適合流量模式不明的尖峰工作負載。使用隨需容量模式，您可以按請求支付應用程式在資料表上執行的資料讀取和寫入費用。您不需要指定預期應用程式執行的讀取和寫入輸送量，因為 DynamoDB 會立即因應工作負載的升降。

如何處理具有連結應用程式的工作負載

在此情況中，應用程式會依賴其他系統，像是在批次處理的情況，您可能會根據應用程式邏輯中的事件，遇到流量出現大峰值。

請考慮開發自訂自動擴展邏輯，以回應這些事件，讓您可以根據特定需要，提升資料表容量與 `TargetValues`。您可以受益於 Lambda Amazon EventBridge 和 Step Functions 等 AWS 服務，並使用其組合來回應您的特定應用程式需求。

評估您的 DynamoDB 資料表類別選擇

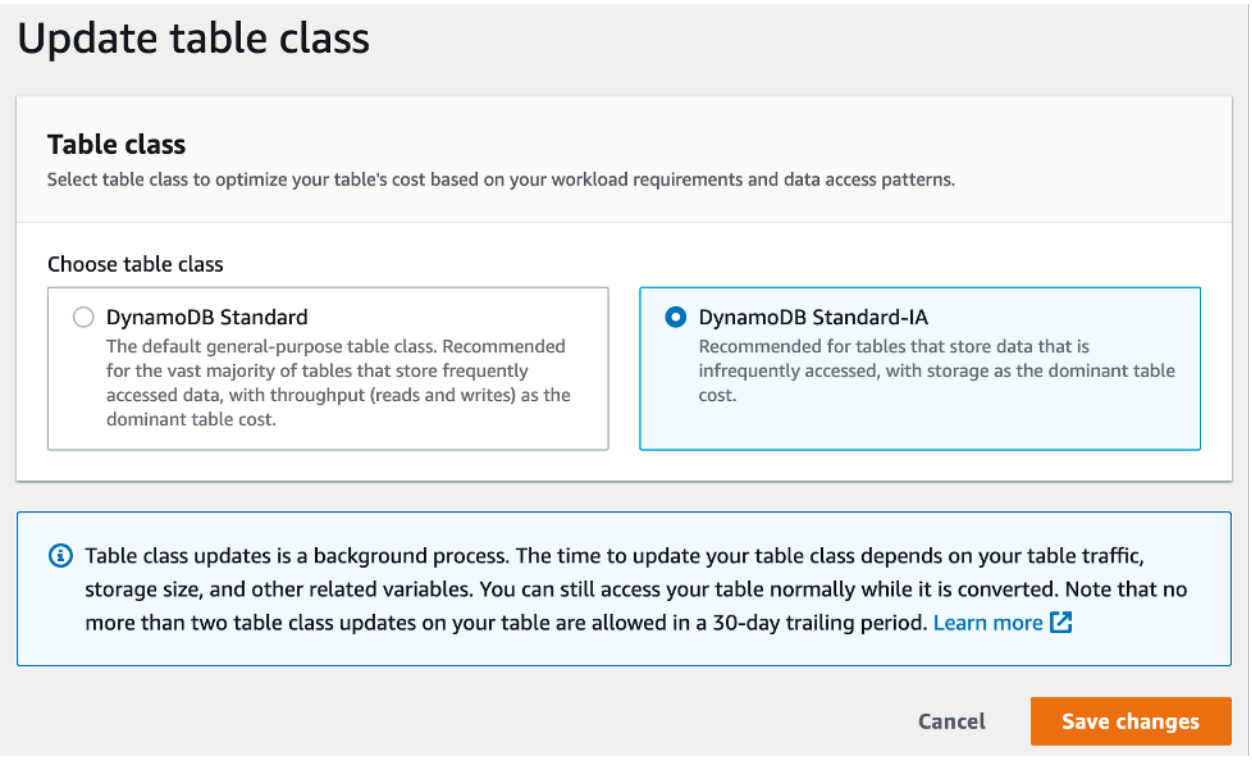
本節概述如何為 DynamoDB 資料表選取適當的資料表類別。隨著標準不常存取 (Standard Infrequent-Access，標準 - IA) 資料表類別的推出，您現在可利用此功能將資料表最佳化，實現較低的儲存成本或輸送量成本。

主題

- 有哪些可用的資料表類別
- 何時選取 DynamoDB 標準資料表類別
- 何時選取 DynamoDB 標準 - IA 資料表類別
- 選擇資料表類別時應考慮的其他因素

有哪些可用的資料表類別

建立 DynamoDB 資料表時，您必須選取 DynamoDB 標準或 DynamoDB 標準 - IA 資料表類別。資料表類別可以在 30 天內變更兩次，因此可以在日後隨時變更。選取任一資料表類別，都不會影響資料表的效能、可用性、可靠性或耐久性。



標準資料表類別

新資料表預設的選項是標準資料表類別。對於具有頻繁存取資料的資料表，此選項可維持 DynamoDB 的原始計費餘額，有利於在輸送量和儲存成本之間取得平衡。

標準 - IA 資料表類別

對於需要長期儲存資料且不常更新或讀取的工作負載，標準 - IA 資料表類別可提供較低的儲存成本 (約降低 60%)。由於此類別已針對非頻繁的存取模式進行最佳化，因此讀取和寫入會以稍高於標準資料表類別的費率 (約提高 25%) 來計費。

何時選取 DynamoDB 標準資料表類別

如果資料表的儲存成本佔資料表每月總成本約 50% 以下，則最適合選擇 DynamoDB 標準資料表類別。這項成本結餘表示一項工作負載定期存取或更新 DynamoDB 內儲存項目的情況。

何時選取 DynamoDB 標準 - IA 資料表類別

如果資料表的儲存成本佔資料表每月總成本約 50% 以上，則最適合選擇 DynamoDB 標準 - IA 資料表類別。這項成本結餘表示一項工作負載每月建立或讀取項目少於儲存數量的情況。

標準 - IA 資料表類別的常見用途是將不常存取的資料移至個別的標準 - IA 資料表。如需進一步資訊，請參閱使用 Amazon DynamoDB 標準 - IA 資料表類別來最佳化工作負載的儲存成本。

選擇資料表類別時應考慮的其他因素

在兩種資料表類別之間進行抉擇時，還有一些其他因素值得納入決策考量。

預留容量

目前不支援使用標準 - IA 資料表類別來購買資料表的預留容量。在從具有預留容量的標準資料表轉換為不含預留容量的標準 - IA 資料表時，可能不會為您帶來可見的成本效益。

識別您在 DynamoDB 中未使用的資源

本節概述如何定期評估未使用的資源。隨著應用程式需求演變，您應確保沒有任何資源仍未使用且因此產生不必要的 Amazon DynamoDB 成本。在以下說明的程序中，將使用 Amazon CloudWatch 指標來識別未使用的資源，進協助您找出這些資源，並採取降低成本的行動。

您可以使用 CloudWatch 來監控 DynamoDB；該服務會收集並處理來自 DynamoDB 的原始資料，進而將這些資料轉換為便於讀取且幾近即時的指標。這些統計資料會保留一段時間，以便您存取歷史資訊，並更清楚了解自己的使用率。根據預設，系統會自動將 DynamoDB 指標資料傳送至 CloudWatch。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的什麼是 Amazon CloudWatch？以及指標保留。

主題

- 如何識別未使用資源
- 識別未使用的資料表資源
- 清除未使用的資料表資源
- 識別未使用的 GSI 資源
- 清除未使用的 GSI 資源
- 清除未使用的全域資料表
- 清除未使用的備份或時間點復原 (PITR)

如何識別未使用資源

為了識別未使用的資料表或索引，我們會查看 30 天內的下列 CloudWatch 指標，以了解是否有對資料表的任何作用中讀取或寫入，或對全域次要索引 (GSI) 的任何讀取：

ConsumedReadCapacityUnits

在指定時段使用的讀取容量單位數目，可讓您追蹤已使用多少使用容量。您可以擷取資料表及其所有全域次要索引或特定全域次要索引的總消耗讀取容量。

ConsumedWriteCapacityUnits

在指定時段使用的寫入容量單位數目，可讓您追蹤已使用多少使用容量。您可以擷取資料表及其所有全域次要索引或特定全域次要索引的消耗總計寫入容量。

識別未使用的資料表資源

Amazon CloudWatch 是一項監控和可觀測性服務，提供 DynamoDB 資料表指標，以利您識別未使用的資源。您可以透過 AWS 管理主控台 和 AWS Command Line Interface來檢視 CloudWatch 指標。

AWS Command Line Interface

若要透過 檢視資料表指標 AWS Command Line Interface，您可以使用下列命令。

1. 首先評估資料表的讀取：

```
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>
```

為了避免將資料表誤認為未使用，請評估較長期間內的指標。選擇適當的開始時間和結束時間範圍 (例如 30 天) 及適當的期間 (例如 86400)。

在傳回的資料中，任何大於 0 的總和都表示您所評估的資料表在該期間內曾經接收讀取流量。

下列結果顯示在評估期間接收讀取流量的資料表：

```
        {
            "Timestamp": "2022-08-25T19:40:00Z",
            "Sum": 36023355.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-12T19:40:00Z",
            "Sum": 38025777.5,
            "Unit": "Count"
        },
```

下列結果顯示在評估期間未接收讀取流量的資料表：

```
        {
```

```
            "Timestamp": "2022-08-01T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-20T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

2. 接下來，評估資料表的寫入數：

```
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>
```

為了避免將資料表誤認為未使用，建議您評估較長期間內的指標。選擇適當的開始時間和結束時間範圍 (例如 30 天) 及適當的期間 (例如 86400)。

在傳回的資料中，任何大於 0 的總和都表示您所評估的資料表在該期間內曾經接收讀取流量。

下列結果顯示在評估期間接收寫入流量的資料表：

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 41014457.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-18T20:15:00Z",
            "Sum": 40048531.0,
            "Unit": "Count"
        },
```

下列結果顯示在評估期間曾接收寫入流量的資料表：

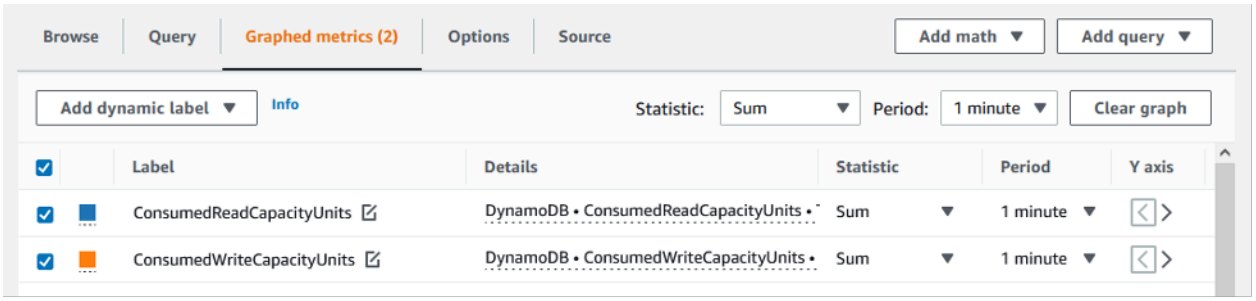
```
        {
            "Timestamp": "2022-07-31T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

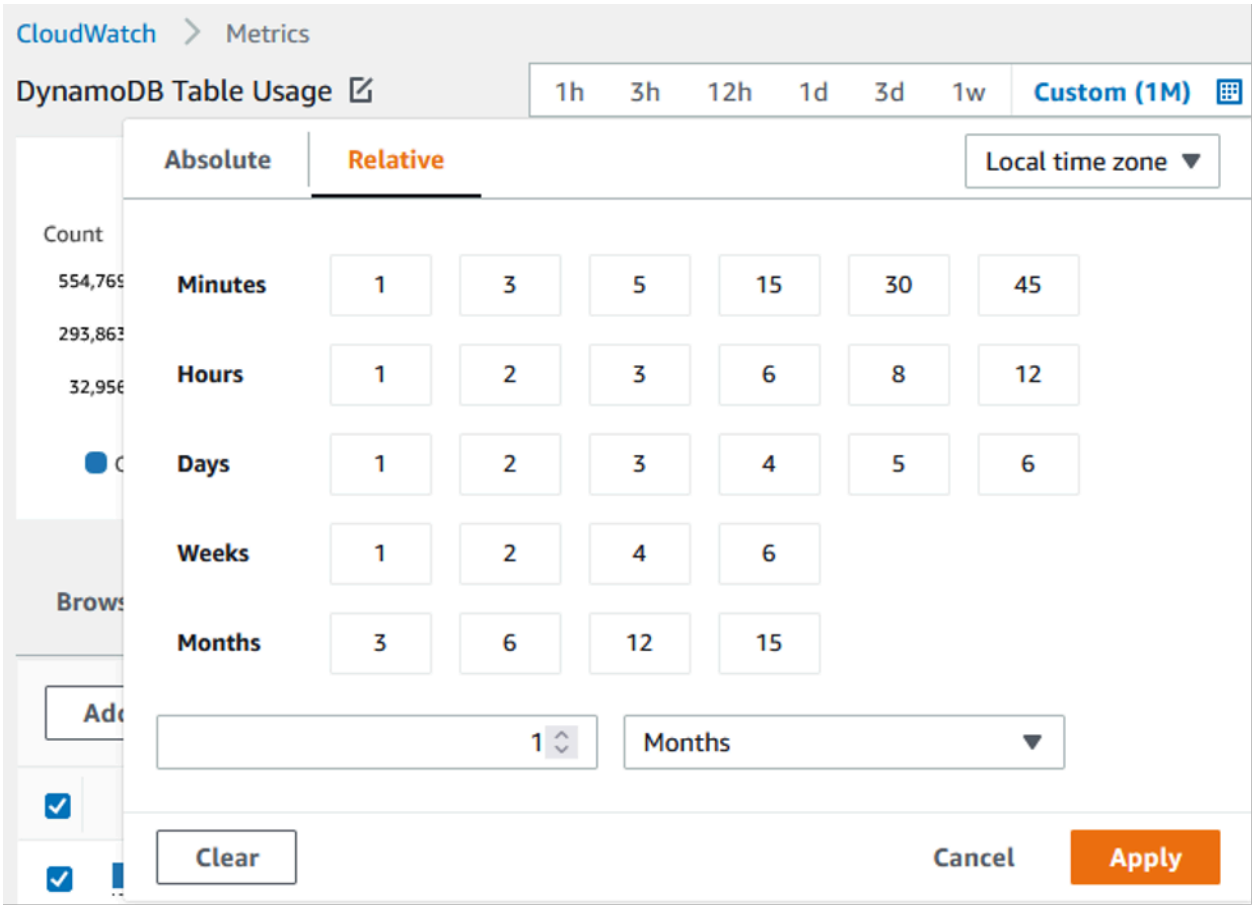
AWS 管理主控台

採用下列步驟，即可透過 AWS 管理主控台評估資源使用率。

1. 登入 AWS 主控台並導覽至位於 https：//https://console.aws.amazon.com/cloudwatch/ 的 CloudWatch 服務頁面。如有必要，請在主控台的右上角選取適當的 AWS 區域。
2. 在左側導覽窗格中，找到 Metrics (指標) 區段並選取 All metrics (所有指標)。
3. 這個動作會開啟一個具有兩個面板的儀表板。在頂部面板中，您會看到目前的圖表化指標。在底部，您將能選取可供進行圖表化的指標。在底部面板中選取 DynamoDB。
4. 在 DynamoDB 指標選取面板中，選取 Table Metrics (資料表指標) 類別，以顯示目前區域中資料表的指標。
5. 向下捲動選單找出您的資料表名稱，然後為您的資料表選取指標 `ConsumedReadCapacityUnits` 和 `ConsumedWriteCapacityUnits`。
6. 選取 Graphed metrics (2) (圖表化指標 (2)) 索引標籤，然後將 Statistic (統計數據) 欄調整為 Sum (總和)。

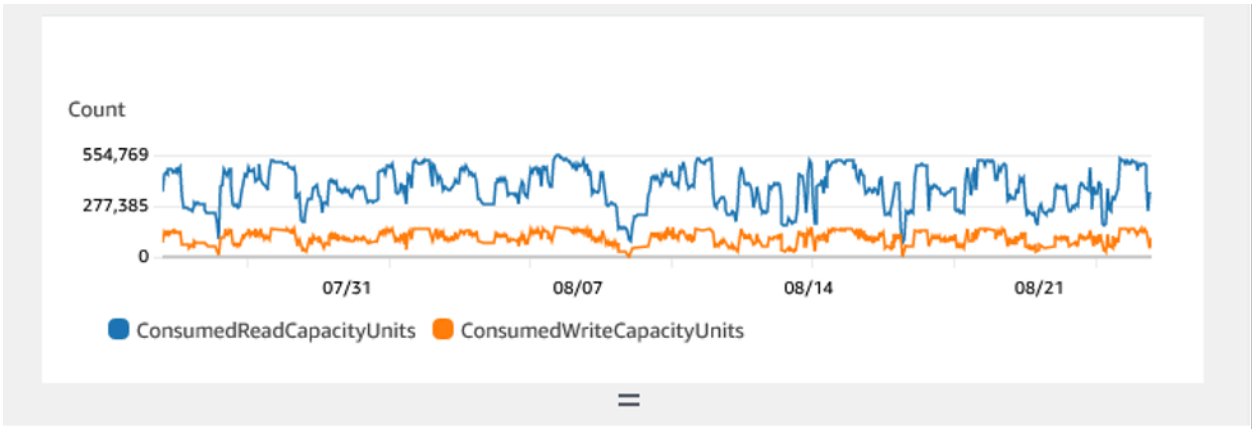


7. 為了避免將資料表誤認為未使用，建議您評估較長期間內的指標。在圖表面板頂端，選擇適當的時間範圍 (例如 1 個月) 來評估您的資料表。選取 Custom (自訂)，在下拉式清選中選取 1 Months (1 個月)，然後選擇 Apply (套用)。

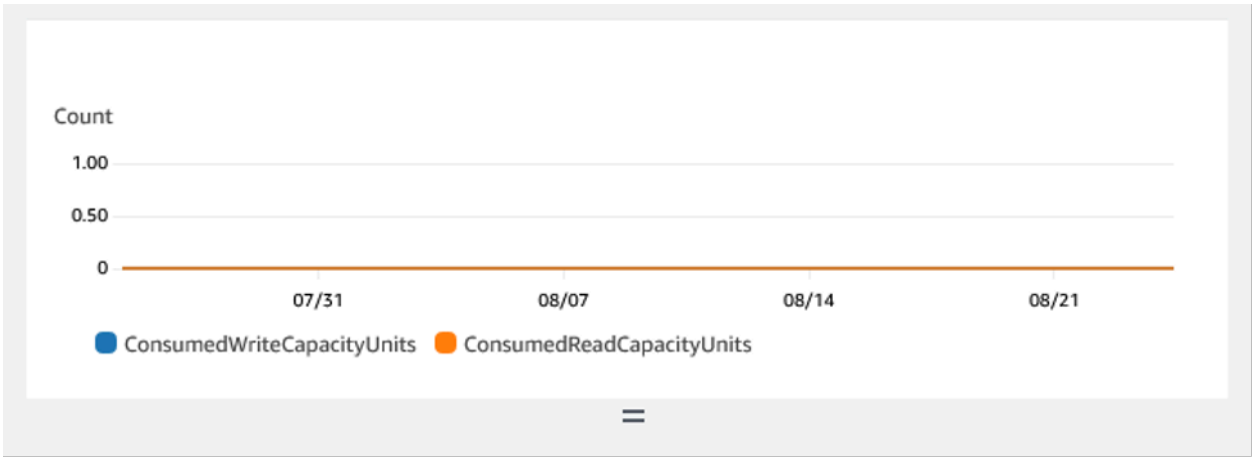


8. 評估資料表的圖表化指標，判斷是否已使用該資料表。如果指標超過 0，就表示在評估期間內已使用該資料表。讀取和寫入均為 0 的平坦圖形，表示該資料表尚未使用。

   下圖顯示具有讀取流量的資料表：



   下圖顯示沒有讀取流量的資料表：



清除未使用的資料表資源

如果找出未使用的資料表資源，您可以透過下列方式降低其持續產生的成本。

> **Note**
>
> 如果找出未使用的資料表，但仍希望保留以備日後需要時可供存取，請考慮將其轉換為隨需模式。否則，您可以考慮備份和刪除整個資料表。

容量模式

DynamoDB 會對 DynamoDB 資料表中讀取、寫入及儲存資料等活動收取費用。

DynamoDB 有「隨需」和「佈建」兩種容量模式，分別提供適用於處理資料表讀取和寫入的特定計費選項。讀取/寫入容量模式可控制您變更讀取與寫入輸送量以及管理容量的方式。

若為隨需模式資料表，不需要指定您預期應用程式將進行的讀取和寫入輸送量。DynamoDB 會根據應用程式在資料表上執行的讀取與寫入作業，依讀取請求單位與寫入請求單位計費。如果您的資料表/索引沒有任何活動，則不需要支付輸送量的費用，但仍會產生儲存費用。

資料表類別

DynamoDB 也提供兩種資料表類別，旨在協助您最佳化成本。預設值為 DynamoDB 標準資料表類別，建議大多數工作負載使用。DynamoDB 標準：不常存取 (DynamoDB 標準-IA) 資料表類別針對以儲存為主要成本的資料表進行最佳化。

如果您的資料表或索引沒有任何活動，儲存體可能成為主要成本來源，而變更資料表類別可大幅節省成本。

刪除資料表

如果您發現未使用的資料表並希望予以刪除，建議您先備份或匯出資料。

透過 Backup 取得的 AWS 備份可以利用冷儲存分層，進一步降低成本。有關如何透過 AWS Backup 啟用備份的資訊，請參閱 搭配 DynamoDB 使用 AWS Backup 文件，以及管理備份計劃文件，了解如何使用生命週期將備份移至冷儲存的資訊。

或者，您可以選擇將資料表的資料匯出到 S3。若要執行此操作，請參閱匯出至 Amazon S3 文件。資料匯出後，如果希望利用 S3 Glacier Instant Retrieval、S3 Glacier Flexile Retrieval 或 S3 Glacier Deep Archive 進一步降低成本，請參閱管理儲存生命週期。

備份資料表之後，即可選擇透過 AWS 管理主控台 或 AWS Command Line Interface加以刪除。

識別未使用的 GSI 資源

識別未使用全域次要索引的步驟類似於識別未使用資料表的步驟。如果寫入基礎資料表的項目包含用作 GSI 分割索引鍵的屬性，則 DynamoDB 會將這些項目複製到 GSI 中，因此在基礎資料表處於使用狀態時，未使用的 GSI 仍可能具有大於 0 的 `ConsumedWriteCapacityUnits`。因此，您只需評估 `ConsumedReadCapacityUnits` 指標即可判斷 GSI 是否未使用。

若要透過 檢視 GSI 指標 AWS AWS CLI，您可以使用下列命令來評估資料表讀取：

```
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>
```

為了避免將資料表誤認為未使用，建議您評估較長期間內的指標。選擇適當的開始時間和結束時間範圍 (例如 30 天) 及適當的期間 (例如 86400)。

在傳回的資料中，任何大於 0 的總和都表示您所評估的資料表在該期間內曾經接收讀取流量。

下列結果顯示 GSI 在評估期間內接收讀取流量：

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 36319167.0,
          "Unit": "Count"
```

```
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 1869136.0,
          "Unit": "Count"
        },
```

下列結果顯示 GSI 在評估期間內接收最低讀取流量：

```
        {
          "Timestamp": "2022-08-28T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-15T21:20:00Z",
          "Sum": 2.0,
          "Unit": "Count"
        },
```

下列結果顯示 GSI 在評估期間未接收讀取流量：

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
```

清除未使用的 GSI 資源

如果找出未使用的 GSI，您可以選擇將其刪除。由於 GSI 中的所有資料也位於基礎資料表中，因此在刪除 GSI 之前，不需進行額外的備份。如果日後再次需要 GSI，則可以將其重新加入資料表中。

如果找出不常使用的 GSI，則應考慮變更應用程式中的設計，以便將其刪除或降低其成本。例如，雖然應謹慎使用 DynamoDB 掃描，因為其可能消耗大量系統資源，但如果它們支援的存取模式很少使用，則可能比 GSI 更具成本效益。

此外，如果需要 GSI 來支援「不常存取」模式，請考慮投射一組限制程度較高的屬性。雖然這可能需要對基礎資料表進行後續查詢以支援「不常存取」模式，但可能有助於大幅降低儲存和寫入成本。

清除未使用的全域資料表

Amazon DynamoDB 全域資料表提供全受管解決方案，用以部署多個區域和多個作用中資料庫，而不需要建置和維護您自己的複寫解決方案。

對於使用者近處的資料以及用於災難復原的次要區域，理想的做法是使用全域資料表來提供低延遲的存取管道。

如果針對特定資源啟用全域資料表選項，藉以提供低延遲的資料存取管道，但其並不屬於您災難復原策略的一部分，則請評估它們的 CloudWatch 指標，以確認兩個複本是否都主動提供讀取流量。如果其中一個複本不提供讀取流量，則可能就是未使用的資源。

如果全域資料表是災難復原策略的一部分，則在作用中/待命模式下可能會有一個複本未接收讀取流量。

清除未使用的備份或時間點復原 (PITR)

DynamoDB 提供兩種備份樣式。時間點復原可提供長達 35 天的連續備份，協助防止意外寫入或刪除；隨需備份則可建立快照以進行長期保存。您可以將復原期間設為 1 到 35 天。兩種類型的備份都會產生相關成本。

請參閱有關 DynamoDB 的備份與還原 和 DynamoDB 的時間點備份 的說明文件，以判斷您的資料表是否啟用了可能不再需要的備份。

評估您的 DynamoDB 資料表使用模式

本節概述如何判斷您有效地使用 DynamoDB 資料表。某些用量模式對 DynamoDB 來說並非最理想的，且從效能和成本角度來看，這些模式仍有最佳化空間。

主題

- 執行較少高度一致性讀取操作
- 執行較少的讀取操作交易
- 執行較少掃描
- 縮短屬性名稱
- 啟用存留時間 (TTL)
- 以跨區域備份取代全域資料表

執行較少高度一致性讀取操作

DynamoDB 可讓您根據每個請求設定讀取一致性。根據預設，讀取請求最終會保持一致。最終一致讀取費用為 0.5 RCU，資料上限為 4 KB。

分散式工作負載的多數部分都具有彈性，可以容忍最終一致性。但是，有些存取模式要求高度一致性讀取。高度一致性讀取費用為 1 RCU，資料上限為 4 KB，實際上是讀取成本的兩倍。DynamoDB 讓您能在同一個資料表上使用這兩種一致性模式。

您可以評估工作負載和應用程式的程式碼，確定是否僅在需要時使用高度一致性讀取。

執行較少的讀取操作交易

DynamoDB 可讓您以全有或全無的方式，將部分動作分組，這表示您可以使用 DynamoDB 執行 ACID 交易。但是，如同關聯式資料庫，並非每個動作都需要遵循這種方法。

上限 4 KB 的交易讀取操作會耗用 2 RCU，而非以最終一致方式讀取相同數量資料的預設 0.5 RCU。寫入操作的成本會加倍，也就是說，最多 1 KB 的交易寫入等於 2 WCU。

若要判斷資料表上的所有操作是否皆為交易，可以將資料表的 CloudWatch 指標向下篩選為交易 API。若交易 API 是資料表 `SuccessfulRequestLatency` 指標下唯一可用的圖表，這就表示此資料表中每項操作都是交易。或者，若整體容量使用率趨勢符合交易 API 趨勢，請考慮重新檢視應用程式設計，因為該設計似乎由交易 API 主導。

執行較少掃描

會廣泛使用 `Scan` 操作，通常源於對 DynamoDB 資料執行分析查詢的需求。在大型資料表上執行頻繁 `Scan` 操作可能既低效又昂貴。

更好的替代方案是使用匯出至 S3 功能，並選擇時間點將資料表狀態匯出至 S3. AWS offers 服務，例如 Athena，然後可用於對資料執行分析查詢，而不會耗用資料表中的任何容量。

您可以使用 `Scan` 的 `SuccessfulRequestLatency` 指標下的 `SampleCount` 統計資料，決定 `Scan` 操作的頻率。若 `Scan` 操作確實十分頻繁，則應重新評估存取模式和資料模型。

縮短屬性名稱

DynamoDB 項目的總大小是其屬性名稱長度和值的總和。較長的屬性名稱不僅產生更多儲存成本，也導致更多 RCU/WCU 耗用。建議您選擇較短的屬性名稱，而不是較長的屬性名稱。較短的屬性名稱有助於限制項目大小少於 4KB/1KB，之後您將耗用額外的 RCU/WCU 來存取資料。

啟用存留時間 (TTL)

存留時間 (TTL) 可識別比您設定的到期時間更早的項目，並將其從資料表中移除。若您的資料已不重要，則在資料表上啟用 TTL 可協助您縮減資料並節省儲存成本。

TTL 的另一個益處為，過期項目發生在 DynamoDB 串流上，因此您不僅從資料中移除該資料，而是可以從串流中使用這些項目，並封存到成本較低的儲存層。此外，透過 TTL 刪除項目不需額外費用 – 它不會消耗容量，且設計清理應用程式也不會產生額外負荷。

以跨區域備份取代全域資料表

全域資料表可讓您在不同區域中維護多個使用中複本資料表 – 這些資料表皆可接受寫入操作，並彼此複寫資料。您可以輕鬆設定複本，系統會為您管理同步處理。使用最後一個寫入獲勝策略，將複本收斂到一致。

若您僅使用全域資料表做為容錯移轉或災難復原 (DR) 策略的一部分，您可以將資料表取代為跨區域備份複本，以達到相對較寬鬆的復原點目標，和復原時間點目標需求。若您不需要快速的本機存取和五個九的高可用性，維護全域資料表複本可能不是容錯移轉的最佳方法。

或者，請考慮使用 AWS Backup 來管理 DynamoDB 備份。與使用全域資料表相比，您可以排程定期備份並跨區域複製備份，以更符合成本效益的方式滿足 DR 需求。

評估您的 DynamoDB Streams 使用情況

本章節概述如何評估 DynamoDB Streams 使用量。某些用量模式對 DynamoDB 來說並非最理想的，且從效能和成本角度來看，這些模式仍有最佳化空間。

您有兩個原生串流整合，適用於串流和事件驅動使用案例：

- Amazon DynamoDB Streams
- Amazon Kinesis Data Streams

此頁面將著重於這些選項的成本最佳化策略。若您想了解如何在兩個選項間選擇，請參閱 變更資料擷取的串流選項。

主題

- 最佳化 DynamoDB Streams 的成本
- 最佳化 Kinesis Data Streams 的成本
- 兩種 Streams 使用量類型的成本最佳化策略

最佳化 DynamoDB Streams 的成本

如 DynamoDB Streams 的定價頁面所述，無論資料表的輸送容量模式為何，DynamoDB 都會根據對資料表 DynamoDB 串流發出的讀取請求量計費。對 DynamoDB 串流發出的讀取請求，與對 DynamoDB 資料表發出的讀取請求不同。

串流中的每個讀取請求都採用 `GetRecords` API 呼叫形式，可在回應中傳回最多 1000 筆記錄或 1 MB 的記錄，以先到達者為準。其他 DynamoDB 串流 API 不會收取任何費用，且 DynamoDB Streams 不會因閒置而收費。換句話說，若沒有對 DynamoDB 串流發出讀取請求，則在資料表上啟用 DynamoDB 串流將不會產生任何費用。

以下是 DynamoDB Streams 的部分取用者應用程式：

- AWS Lambda function(s)
- Amazon Kinesis Data Streams型應用程式
- 使用 AWS SDK 建置的客戶消費者應用程式

以 AWS Lambda 為基礎的 DynamoDB Streams 取用者提出的讀取請求是免費的，而任何其他類型的取用者提出的呼叫則會收費。每個月，非 Lambda 取用者發出的前 2,500,000 個讀取要求也是免費。這適用於對每個 AWS 區域 AWS 帳戶中任何 DynamoDB 串流提出的所有讀取請求。

監控您的 DynamoDB Streams 使用量

帳單主控台上的 DynamoDB Streams 費用會針對 AWS 帳戶中跨 AWS 區域的所有 DynamoDB Streams 分組在一起。目前不支援標記 DynamoDB Streams，因此無法使用成本分配標籤來識別 DynamoDB Streams 的精細成本。您可以在 DynamoDB 串流層級取得 `GetRecords` 磁碟區，以計算每個串流的費用。磁碟區由 DynamoDB 串流的 CloudWatch 指標 `SuccessfulRequestLatency` 及其 `SampleCount` 統計資料表示。此指標也將包含由全域資料表進行的`GetRecords`呼叫，以執行持續複寫，以及由 AWS Lambda 消費者進行的呼叫，這兩者都不會收費。如需詳細了解 DynamoDB Streams 發布的其他 CloudWatch 指標，請參閱 DynamoDB 指標和維度。

使用 AWS Lambda 做為消費者

評估使用 AWS Lambda 函數做為 DynamoDB Streams 的取用者是否可行，因為這可以消除從 DynamoDB Stream 讀取相關的成本。另一方面，DynamoDB Streams Kinesis 轉接器或以 SDK 為基礎的取用者應用程式將根據他們對 DynamoDB 串流發出的 `GetRecords` 呼叫量計費。

Lambda 函式調用將根據標準 Lambda 定價收費，但 DynamoDB Streams 不會產生任何費用。Lambda 會輪詢您 DynamoDB 串流中的碎片，其記錄的基本速率為每秒 4 次。當記錄可用

時，Lambda 會調用您的函數，並等待結果。如果處理成功，Lambda 會恢復輪詢，直到收到多筆記錄。

調校以 DynamoDB Streams Kinesis 轉接器為基礎的取用者應用程式

由於非 Lambda 取用者發出的讀取要求會針對 DynamoDB Streams 收費，因此需在近乎即時的要求與取用者應用程式須輪詢 DynamoDB 串流的次數間找出平衡。

使用以 DynamoDB Streams Kinesis 轉接器為基礎的應用程式輪詢 DynamoDB Streams 的頻率由設定的 `idleTimeBetweenReadsInMillis` 值決定。此參數決定取用者在處理碎片前應等待的時間 (以毫秒為單位)，以防先前對相同碎片進行的 `GetRecords` 呼叫未傳回任何記錄。此參數的預設值為 1000 ms。若不需要近乎即時的處理，則可以增加此參數，讓取用者應用程式在 DynamoDB Streams 呼叫上進行較少的 `GetRecords` 呼叫，並進行最佳化。

最佳化 Kinesis Data Streams 的成本

當 Kinesis 資料串流設定為目的地，為 DynamoDB 資料表提供變更資料擷取事件時，Kinesis 資料串流可能需單獨的大小管理，但這會影響整體成本。DynamoDB 會根據變更資料擷取單位 (CDU) 計費，其中每單位皆由最多 1 KB DynamoDB 項目大小所組成，該項目則為 DynamoDB 服務嘗試傳送至目的地 Kinesis 動資料串流的內容。

除了 DynamoDB 服務費用外，也會產生標準的 Kinesis 資料串流費用。如定價頁面所述，服務定價會根據容量模式 (佈建和隨需) 而不同，這些模式與 DynamoDB 資料表容量模式不同，且為使用者定義。在高階上，Kinesis Data Streams 會根據容量模式以及 DynamoDB 服務擷取到的串流資料，按小時計費。視 Kinesis 資料串流的使用者組態而定，可能產生額外費用，例如資料擷取 (針對隨需模式)、延長資料保留 (超過預設 24 小時)，以及強化廣發取用者擷取。

監控 Kinesis Data Streams 使用量

DynamoDB 專用 Kinesis Data Streams 除了標準的 Kinesis 資料串流 CloudWatch 指標外，還會從 DynamoDB 發布指標。DynamoDB 服務的`Put`嘗試可能會因為 Kinesis Data Streams 容量不足而受到 Kinesis 服務調節，或者可能設定為 AWS KMS 加密靜態 Kinesis Data Stream 資料的相依元件。

若要深入了解 DynamoDB 服務針對 Kinesis 資料串流發布的 CloudWatch 指標，請參閱 使用 Kinesis Data Streams 監控變更資料擷取。為避免因限流而產生額外的服務重試成本，請確定在佈建模式中，Kinesis 資料串流的大小適中。

為 Kinesis Data Streams 選擇合適的容量模式

Kinesis Data Streams 有兩種容量模式受支援：佈建模式和隨需模式。

- 如果涉及 Kinesis 資料串流的工作負載具有可預測或準確預測的應用程式流量 (流量一致或逐漸增加)，那麼 Kinesis Data Stream 的佈建模式就很適合，且更具成本效益
- 若該工作負載是新工作負載，有無法預測的應用程式流量，或您不想管理容量，那麼 Kinesis Data Streams 的隨需模式就很適合，且更具成本效益

最佳化成本的最佳實務是評估與 Kinesis 資料串流相關聯的 DynamoDB 資料表是否具有可預測的流量模式，該模式可利用 Kinesis Data Streams 的佈建模式。若工作負載是新工作負載，您可以在最初幾週使用 Kinesis Data Streams 的隨需模式，檢視 CloudWatch 指標以了解流量模式，再根據工作負載的性質將相同的串流切換至佈建模式。在佈建模式中，可以遵循 Kinesis Data Streams 的碎片管理考量事項，估算碎片數量。

使用 DynamoDB 專用 Kinesis Data Streams，評估您的取用者應用程式

由於 Kinesis Data Streams 不會對 DynamoDB Streams 等 `GetRecords` 呼叫數量計費，因此只要頻率低於 `GetRecords` 的限流限制，取用者應用程式就能盡量執行呼叫。針對 Kinesis Data Streams 的隨需模式，資料讀取按 GB 計費。針對 Kinesis Data Streams 的佈建模式，若資料保留時間少於 7 天，則不收取讀取費用。若 Lambda 函式為 Kinesis Data Streams 取用者，Lambda 會在 Kinesis 串流輪詢每個碎片，以每秒一次的基本速率記錄。

兩種 Streams 使用量類型的成本最佳化策略

AWS Lambda 消費者的事件篩選

Lambda 事件篩選可讓您根據篩選條件捨棄事件，使其無法在 Lambda 函式調用批次中使用。如此一來，便可最佳化 Lambda 成本，以在取用者函數邏輯中處理或捨棄不需要的串流記錄。若要深入了解如何設定事件篩選和寫入篩選條件，請參閱 Lambda 事件篩選。

調校 AWS Lambda 取用者

您可以調校 Lambda 組態參數，進一步最佳化成本，例如提升 `BatchSize` 以在每次調用處理更多內容、啟用 `BisectBatchOnFunctionError` 以防止處理重複項目 (會產生額外成本)，以及設定 `MaximumRetryAttempts` 以防產生太多重試次數。根據預設，失敗的取用者 Lambda 調用會無限重試，直到記錄從串流到期，DynamoDB Streams 的期限約為 24 小時，針對 Kinesis Data Streams 則可設定為 24 小時至最長 1 年。可在《Lambda AWS 開發人員指南》找到其他 Lambda 設定選項，包含上述適用於 DynamoDB 串流取用者的選項。

評估您的 DynamoDB 資料表佈建容量是否合適

本節概述如何評估 DynamoDB 資料表是否有適當大小的佈建。隨著工作負載的演變，您應該適時修改操作程序，尤其是在佈建模式下設定 DynamoDB 資料表時，可能會有過度佈建或佈建不足的風險。

以下描述的程序需要統計資訊，這些資訊應從支援您生產應用程式的 DynamoDB 資料表擷取。若要了解您的應用程式行為，您應定義足以從應用程式擷取資料季節性的時段。例如，若應用程式顯示每週模式，三週時間便應足夠來分析應用程式輸送量需求。

若您不知道從何下手，請使用至少一個月的資料用量進行以下計算。

評估容量時，DynamoDB 資料表可以獨立設定讀取容量單位 (RCU) 和寫入容量單位 (WCU)。如果資料表設定了任何全域次要索引 (GSI)，則需指定耗用輸送量，這也與基礎資料表的 RCU 和 WCU 無關。

> **Note**
>
> 本機次要索引 (LSI) 會耗用基礎資料表的容量。

主題

- 如何擷取 DynamoDB 資料表上的耗用指標
- 如何識別佈建不足的 DynamoDB 資料表
- 如何識別過度佈建的 DynamoDB 資料表

如何擷取 DynamoDB 資料表上的耗用指標

若要評估資料表和 GSI 容量，請監控下列 CloudWatch 指標，然後選取適當維度，擷取資料表或 GSI 資訊：

讀取容量單位	寫入容量單位
`ConsumedReadCapacityUnits`	`ConsumedWriteCapacityUnits`
`ProvisionedReadCapacityUnits`	`ProvisionedWriteCapacityUnits`
`ReadThrottleEvents`	`WriteThrottleEvents`

您可以透過 AWS CLI 或 執行此操作 AWS 管理主控台。

AWS CLI

在擷取資料表耗用指標前，我們需要先用 CloudWatch API 擷取一些歷史資料點。

首先要建立兩個檔案：write-calc.json 和 read-calc.json。這些檔案代表資料表或 GSI 的計算。您需要更新部分欄位 (如下表所示)，以符合您的環境。

欄位名稱	定義	範例
<table-name>	您要分析的資料表名稱	SampleTable
<period>	您將用來評估使用率目標的期間 (以秒為單位)	針對 1 小時的時間，您應指定：3600
<start-time>	評估間隔的開始時間，以 ISO8601 格式顯示	2022-02-21T23:00:00
<end-time>	評估間隔的結束時間，以 ISO8601 格式顯示	2022-02-22T06:00:00

寫入計算檔案將擷取指定日期範圍時段內佈建和耗用的 WCU 數。同時也會產生用於分析的使用率百分比。write-calc.json 檔案的完整內容應如下所示：

```
{
  "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
}
```

讀取計算檔案會使用類似檔案。此檔案將擷取指定日期範圍時段內佈建和耗用的 RCU 數。`read-calc.json` 檔案的內容應如下所示：

```
{
  "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
}
```

一旦您建立了檔案，就可以開始擷取使用率資料。

1. 若要擷取寫入使用率資料，請發出下列命令：

   ```
   aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
   ```

2. 若要擷取讀取使用率資料，請發出下列命令：

   ```
   aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
   ```

這兩個查詢結果將為一系列 JSON 格式的資料點，這些資料點會用於分析。您的結果將取決於您指定的資料點數量、期間以及您自己的特定工作負載資料。這可能看起來如下：

```
{
    "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": []
}
```

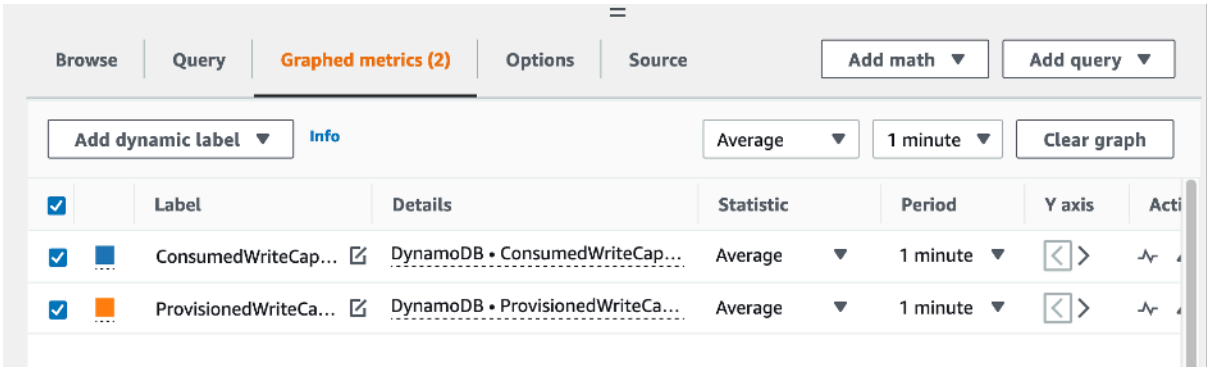
> **Note**
>
> 若您指定一個短期和一個長期時間範圍，您可能需要修改指令碼中預設為 24 的 `MaxDatapoints`。這代表每小時產生一個資料點，每天 24 個資料點。

AWS 管理主控台

1. 登入 AWS 管理主控台 並導覽至 CloudWatch 服務頁面。 AWS 區域 如有必要，請選取適當的 。
2. 在左側導覽窗格中找到指標區段，並選取所有指標。
3. 這會開啟一份儀表板，其中包含兩個面板。上方面板顯示圖表，下方面板顯示您要繪製的指標。選擇 DynamoDB。
4. 選擇資料表指標。接著會顯示目前區域中的資料表。
5. 使用搜尋方塊搜尋您的資料表名稱，然後選擇寫入操作指標：`ConsumedWriteCapacityUnits` 和 `ProvisionedWriteCapacityUnits`。

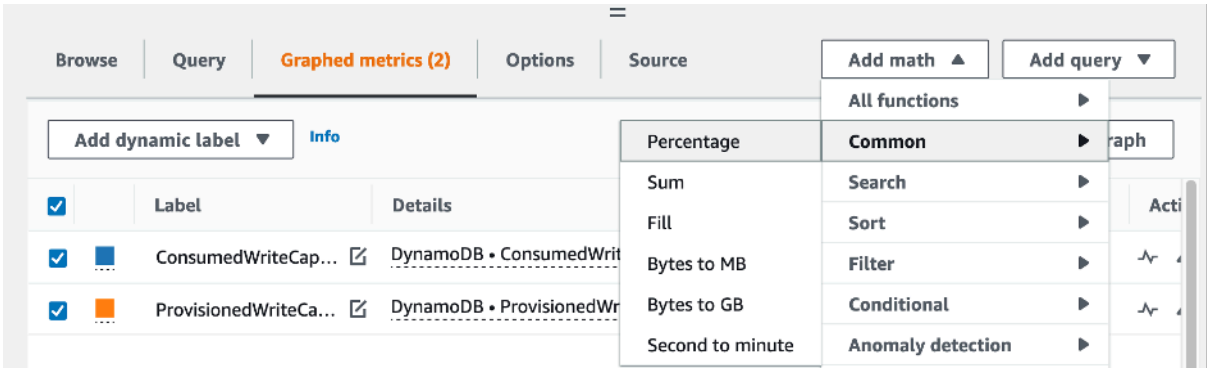
   > **Note**
   >
   > 此範例討論寫入操作指標，但您也可以用這些步驟來繪製讀取操作指標的圖形。

6. 選取圖表化指標 (2)分頁以修改公式。預設情況下，CloudWatch 會為圖表選取統計函數平均值。

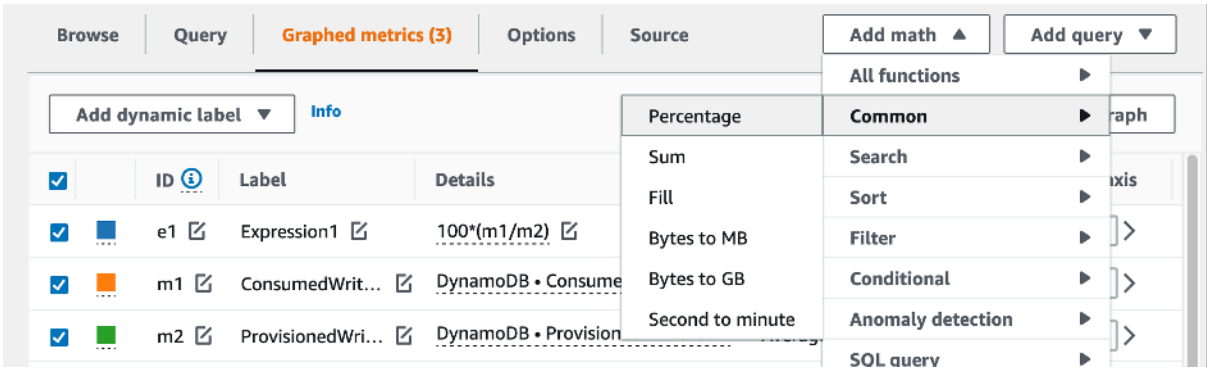


7. 同時選取兩個圖形化指標 (左側的核取方塊) 後，選取選單 Add math (新增數學)，再選取 Common (一般) 及 Percentage (百分比) 函數。重複該過程兩次。

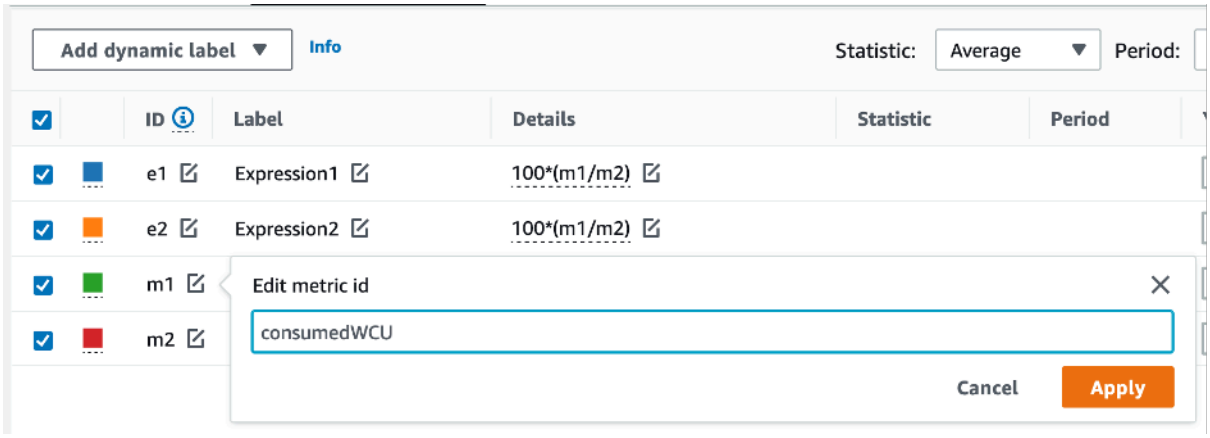
   第一次選取 Percentage (百分比) 函數：



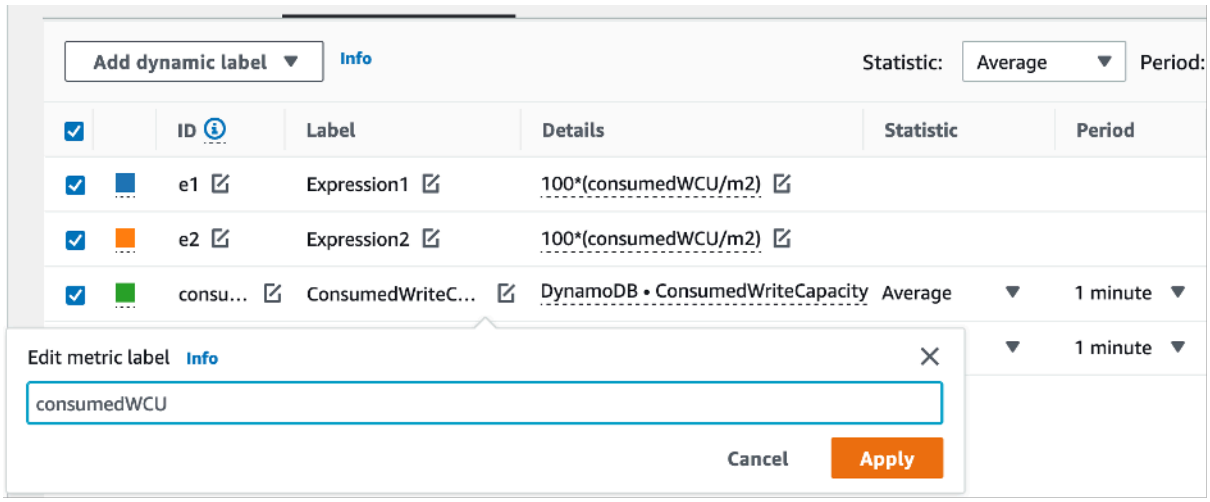
   第二次選取 Percentage (百分比) 函數：



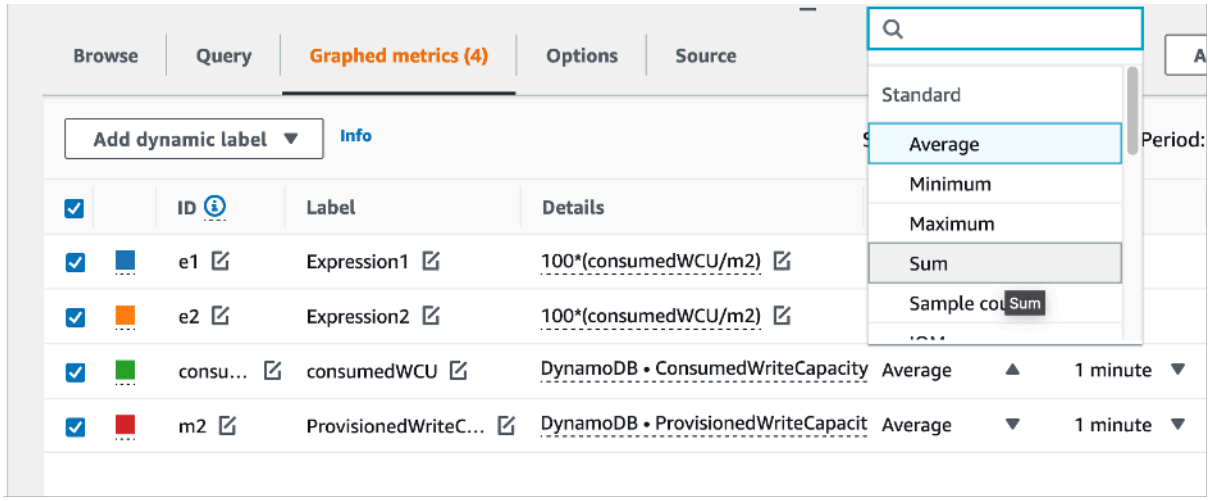
8. 此時，選單底部應有四個指標。接著進行 `ConsumedWriteCapacityUnits` 計算。為了保持一致，我們需要比對我們在 AWS CLI 區段中使用的名稱。按一下 m1 ID，並將此值變更為 consumedWCU。

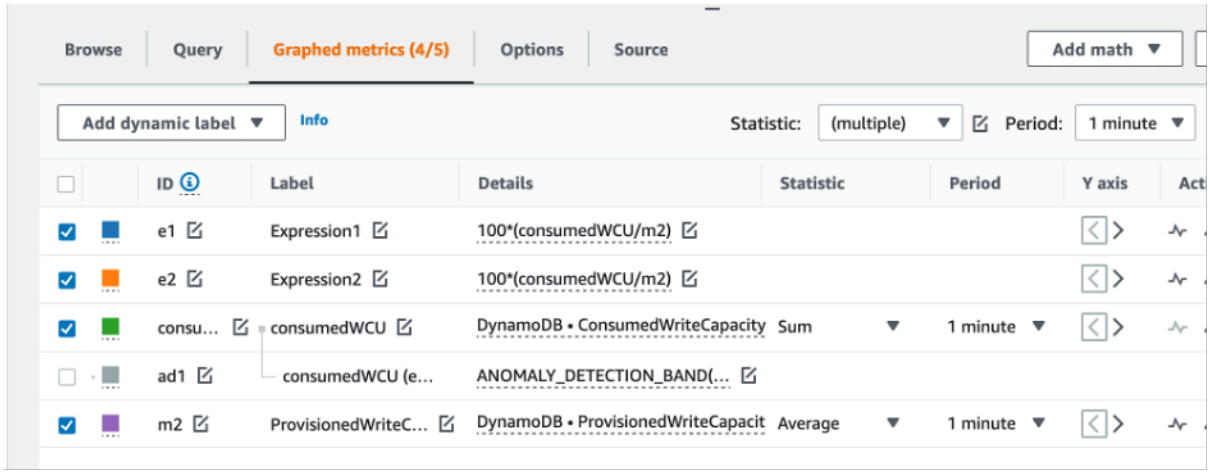


將 ConsumedWriteCapacityUnit 標籤重新命名為 **consumedWCU**。

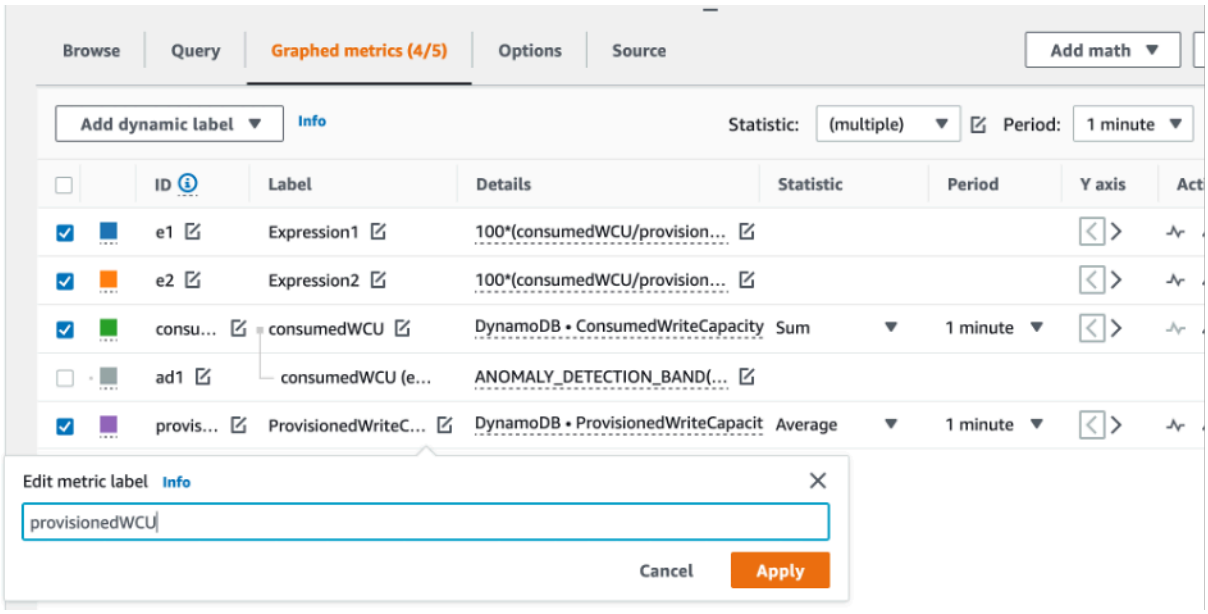


9. 將統計資料從 Average (平均值) 變更為 Sum (總和)。此動作會自動建立另一個稱為 ANOMALY_DETECTION_BAND 的指標。在本程序範圍內，可取消勾選新產生的 ad1 指標，以忽略該項。



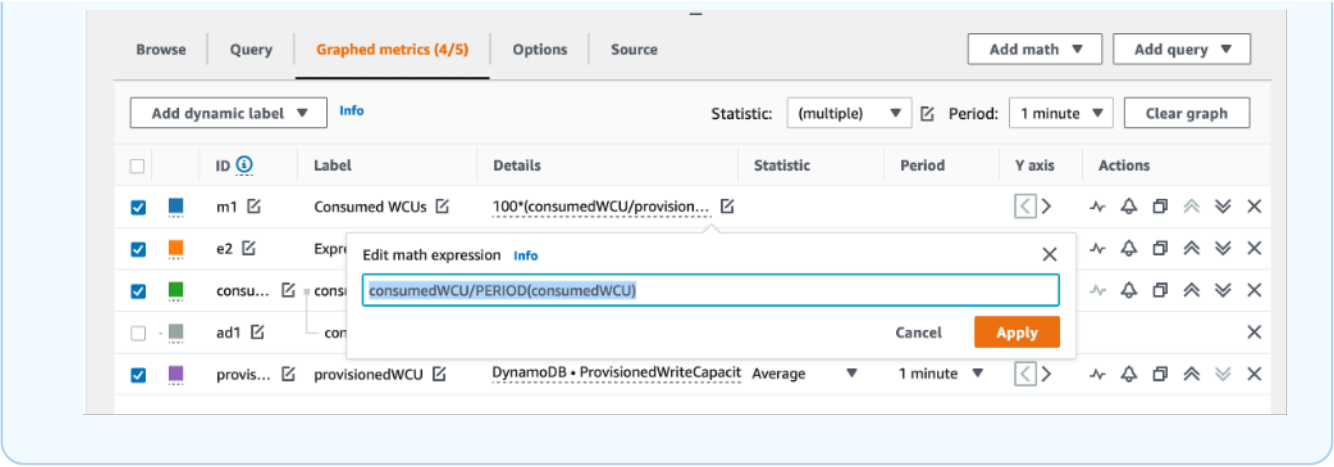


10. 重複步驟 8，將 m2 ID 重新命名為 provisionedWCU。將統計資料設定保留為 Average (平均值)。

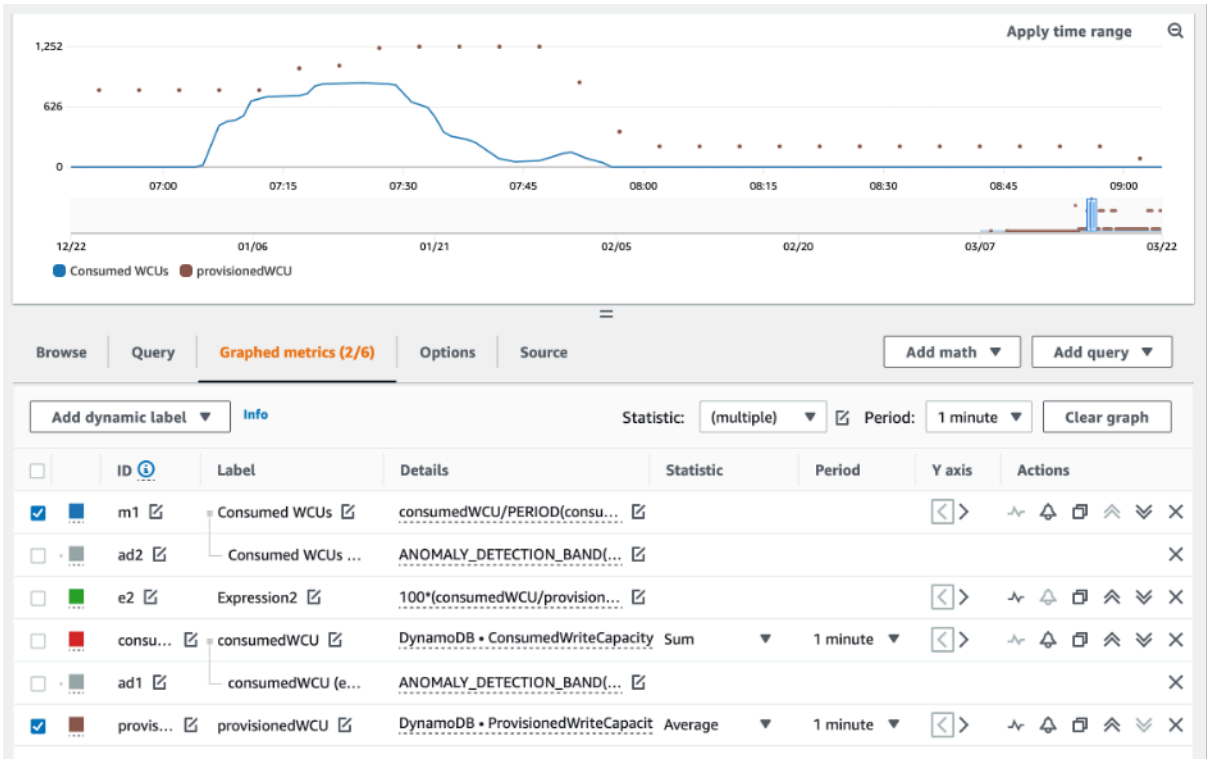


11. 選取 Expression1 標籤，並將值更新為 m1，將標籤更新為 Consumed WCUs。

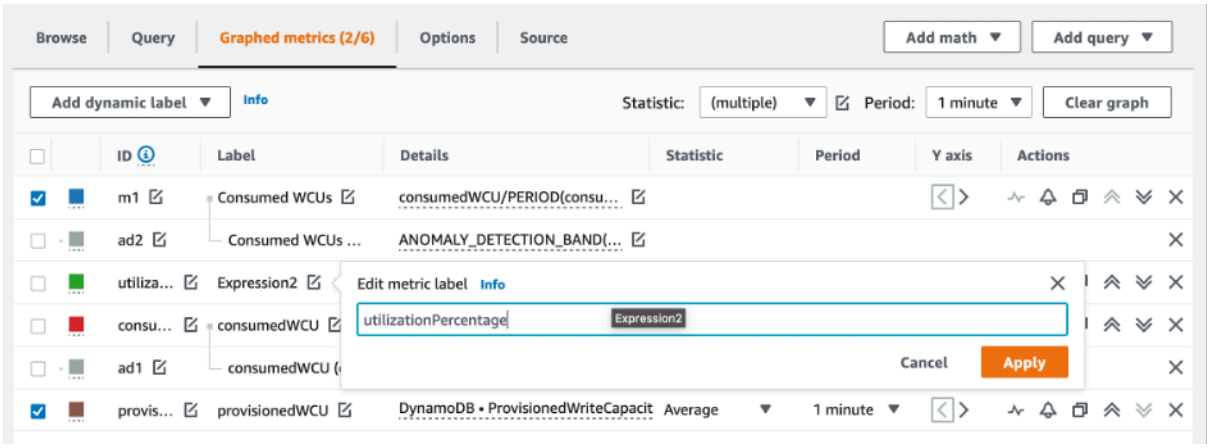
> **Note**
>
> 請確認您只選取 m1 (左側的核取方塊) 及 provisionedWCU，適當地視覺化資料。按一下 Details (詳細資料)，並將公式變更為 consumedWCU/PERIOD(consumedWCU)，以更新公式。此步驟也可能產生另一個 ANOMALY_DETECTION_BAND 指標，但針對此程序的範圍，我們可加以忽略。

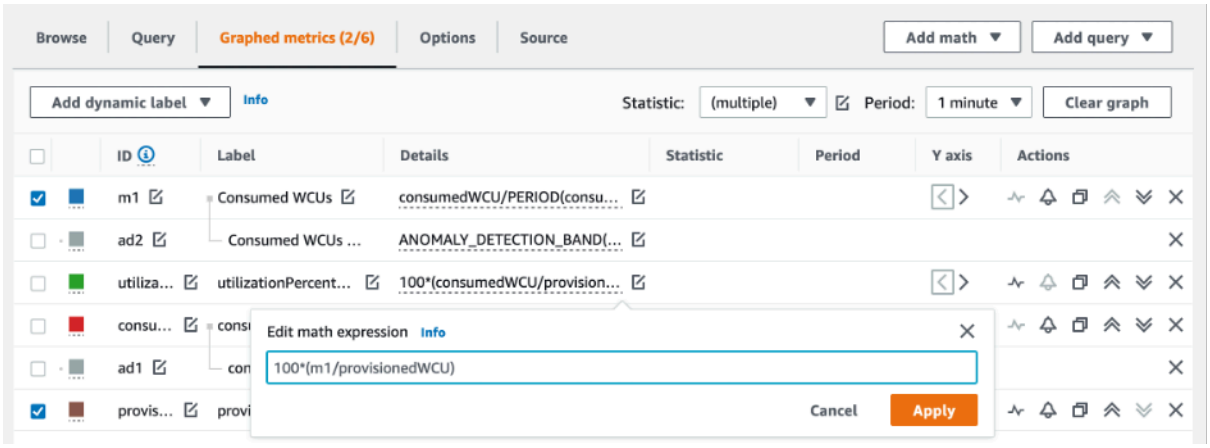


12. 您現在應有兩個圖形：一個表示資料表上已佈建的 WCU，另一個表示已耗用的 WCU。圖形的形狀可能與下方形狀不同，但您可以將其做為參考：



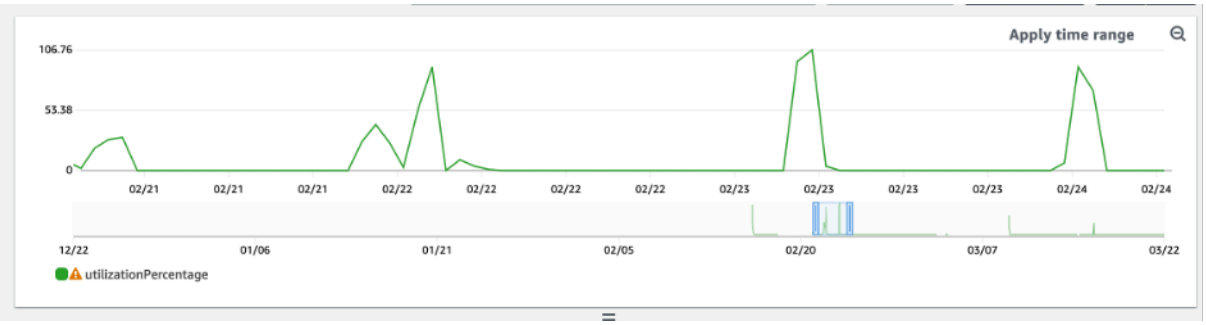
13. 選取 Expression2 圖形 (e2) 以更新百分比公式。將標籤和 ID 重新命名為 utilizationPercentage。將公式重新命名，以符合 100*(m1/provisionedWCU)。





14. 除了 utilizationPercentage 之外，從所有指標移除核取方塊，視覺化您的使用率模式。預設間隔設為 1 分鐘，但您可以根據需要修改。

這是一個更長時間及 1 小時的視圖。您可以看到有些間隔使用率高於 100%，但這個特定的工作負載具有較長的間隔，使用率為零。



您可能得到與此範例中的圖片不同的結果。取決於您工作負載中的資料。使用率超過 100% 的間隔容易發生限流。DynamoDB 提供高載容量，但是一旦高載容量完成，高於 100% 的任何項目都會受到限流。

如何識別佈建不足的 DynamoDB 資料表

針對多數工作負載，當資料表持續耗用超過其佈建容量的 80%，就會被視為佈建不足。

高載容量是一項 DynamoDB 功能，可讓客戶暫時耗用比原先佈建更多的 RCU/WCU (超過資料表定義的每秒佈建輸送量)。高載容量是為了吸收由於特殊事件或使用量尖峰而突增的流量。高載容量不會一直持續執行。一旦耗盡未使用的 RCU 和 WCU，若您嘗試耗用超過佈建的容量，就會受到限流。當應用程式流量接近 80% 使用率時，限流風險就會大幅提高。

80% 使用率規則會因資料的季節性和流量成長而有所不同。請考量下列情況：

- 如果您的流量在過去 12 個月內穩定保持約 90% 的使用率，那麼您的資料表容量剛好
- 如果您的應用程式流量在 3 個月內以每月 8% 的速度增加，您將達到 100%
- 如果您的應用程式流量在略多於 4 個月的期間以 5% 的速度增加，您仍會達到 100%

上述查詢結果能讓您得知使用率的情況。您可以使用這些結果做為指引，進一步評估其他指標，以協助您根據需要增加資料表容量 (例如：每月或每週成長率)。與您的營運團隊合作，為工作負載和資料表定義合理的百分比。

在部分特殊情況下，每天或每週進行分析時，資料會出現偏斜。例如，若季節性應用程式在工作時段使用量高峰 (但在非工作時段幾乎降至零)，您可透過排程的自動擴展指定每日特定時段 (與一週內哪幾天)，以調整佈建容量的增減。若應用季節性不明顯，與其提高容量以應對尖峰時段，不如使用 DynamoDB 資料表自動擴展組態以最佳化效能。

> **Note**
>
> 當您為基礎資料表建立 DynamoDB 自動擴展組態時，請記得將任何與資料表相關的 GSI 加入其他組態。

如何識別過度佈建的 DynamoDB 資料表

從以上指令碼獲得的查詢結果提供了執行部分初始分析所需的資料點。若您的資料集在數個間隔內顯示的使用率低於 20%，表示資料表可能過度佈建。若要進一步判斷是否需減少 WCU 和 RCU 數量，您應重新檢視間隔中的其他讀數。

當資料表出現多個低使用率時段時，您可透過排程或預設的使用率型自動擴展政策有效提升資源利用率。

若您的工作負載使用率低且限流率高 (間隔中 Max(ThrottleEvents)/Min(ThrottleEvents))，則當工作負載尖峰相當明顯，流量在部分日子 (或小時) 大幅增加，但通常流量一直很低時，可能會發生這種情況。在此類情境中，採用排程的自動擴展可發揮效益。

AWS Well-Architected Framework 可協助雲端架構師為各種應用程式和工作負載建置安全、高效能、彈性且高效率的基礎設施。AWS Well-Architected 以營運卓越性、安全性、可靠性、效能效率、成本最佳化和永續性六大支柱為基礎，為客戶提供一致的方法來評估架構並實作可擴展的設計。

AWS Well-Architected 鏡頭將 AWS Well-Architected 提供的指引擴展到特定的產業和技術領域。Amazon DynamoDB Well-Architected Lens 著重於 DynamoDB 工作負載。它提供最佳實務、設計原則和問題，以評估並檢閱 DynamoDB 工作負載。完成 Amazon DynamoDB Well-Architected Lens 檢閱，將為您提供有關建議設計原則的教育和指引，因為它與每個 AWS Well-Architected 支柱相關。本指引是以我們與不同產業、客群、規模和地區之客戶合作的經驗為基礎。

作為 Well-Architected Lens 檢閱的直接結果，您將收到可行建議的摘要，以最佳化並改善 DynamoDB 工作負載。

進行 Amazon DynamoDB Well-Architected Lens 檢閱

DynamoDB Well-Architected Lens 檢閱通常由 AWS 解決方案架構師與客戶一起執行，但也可以由客戶作為自助服務執行。儘管建議您檢閱全部六個 Well-Architected 支柱，作為 Amazon DynamoDB Well-Architected Lens 的一部分，但您也可以決定先將重點放在一或多個支柱上。

如需進行 Amazon DynamoDB Well-Architected Lens 檢閱的其他資訊和指示，請參閱本影片和 DynamoDB Well-Architected Lens GitHub 頁面。

Amazon DynamoDB Well-Architected Lens 的支柱

Amazon DynamoDB Well-Architected Lens 是以六個支柱為中心而建置的：

效能效率支柱

效能效率支柱包括能夠有效率地使用運算資源，以滿足系統需求，並隨著需求變更與技術發展來保持該效率需求。

此支柱的主要 DynamoDB 設計原則是以建模資料、選擇分割區索引鍵和排序索引鍵，以及根據應用程式存取模式定義次要索引為主。其他考量包括為工作負載選擇最佳輸送量模式、 AWS SDK 調校，以及適時使用最佳快取策略。若要進一步了解這些設計原則，請觀看這部有關 DynamoDB Well-Architected Lens 效能效率支柱的深入探討影片。

成本最佳化支柱

成本最佳化支柱著重於避免不必要的成本。

重要主題包括了解和控制花錢之處、選取最合適且正確數量的資源類型、分析一段時間後的花費、設計資料模型以最佳化應用程式特定存取模式的成本，以及擴展以符合業務需求而不會超支。

DynamoDB 的關鍵成本最佳化設計原則是以下列動作為主：為資料表選擇最合適的容量模式和資料表類別，以及使用隨需容量模式或佈建容量模式搭配自動擴展來避免過度佈建容量。其他考量包括高效資料建模與查詢以降低容量消耗、以折扣價保留部分使用容量、縮小項目大小、識別並移除閒置資源，以及使用 TTL 免費自動刪除過期資料。若要進一步了解這些設計原則，請觀看這部有關 DynamoDB Well-Architected Lens 成本最佳化支柱的深入探討影片。

如需 DynamoDB 成本最佳化最佳實務的其他資訊，請參閱成本最佳化。

卓越營運支柱

卓越營運支柱著重於執行和監控系統，以提供商業價值並持續改善流程和程序。重要主題包括自動化變更、回應事件，以及定義管理日常作業的標準。

DynamoDB 的主要卓越營運設計原則包括透過 Amazon CloudWatch 監控 DynamoDB 指標， AWS Config 並在違反預先定義的閾值或偵測到不合規規則時自動提醒和修復。其他考量為透過基礎架構即程式碼定義 DynamoDB 資源，以及利用標籤來改善 DynamoDB 資源的組織、識別和成本會計。若要進一步了解這些設計原則，請觀看這部有關 DynamoDB Well-Architected Lens 卓越營運支柱的深入探討影片。

可靠性支柱

可靠性支柱著重於確保工作負載如預期般正確且一致地執行其預期功能。彈性工作負載可快速從失敗中復原，以滿足業務和客戶需求。重要主題包括分散式系統設計、復原規劃，以及如何處理變更。

DynamoDB 的基本可靠性設計原則涉及根據您的 RPO 和 RTO 需求選擇備份策略和保留， 針對多區域工作負載使用 DynamoDB 全域資料表， 或具有低 RTO 的跨區域災難復原案例， 透過在 AWS SDK 中設定和使用這些功能，在應用程式中實作具有指數退避的重試邏輯， 透過 Amazon CloudWatch 監控 DynamoDB 指標，並在違反預先定義的閾值時自動提醒和修復。若要進一步了解這些設計原則，請觀看這部有關 DynamoDB Well-Architected Lens 可靠性支柱的深入探討影片。

安全支柱

安全支柱著重於保護資訊和系統。重要主題包括資料的機密性和完整性、識別和管理誰可以透過權限管理做什麼、保護系統，以及建立控制項來偵測安全事件。

DynamoDB 的主要安全設計原則是使用 HTTPS 加密傳輸中的資料、選擇靜態資料加密的金鑰類型，以及定義 IAM 角色和政策，以驗證、授權和提供 DynamoDB 資源的精細存取權。其他考量包括透過稽核 DynamoDB 控制平面和資料平面操作 AWS CloudTrail。若要進一步了解這些設計原則，請觀看這部有關 DynamoDB Well-Architected Lens 安全支柱的深入探討影片。

如需 DynamoDB 安全性的其他資訊，請參閱安全性。

永續性支柱

永續性支柱著重於將執行雲端工作負載時對環境的影響降到最低。重要主題包括永續性的共同責任模式、了解影響，以及充分利用以將所需資源降至最少，並減少下游影響。

DynamoDB 的主要永續性設計原則包括識別和移除未使用的 DynamoDB 資源、搭配自動擴展使用隨需容量模式或佈建容量模式來避免過度佈建容量、有效查詢以減少耗用的容量，以及透過壓縮資料和使用 TTL 刪除過時的資料來減少儲存體使用量。若要進一步了解這些設計原則，請觀看這部有關 DynamoDB Well-Architected Lens 永續性支柱的深入探討影片。

在 DynamoDB 中設計與有效運用分割區索引鍵的最佳實務

唯一識別 Amazon DynamoDB 資料表中每個項目的主索引鍵，可以是簡易 (僅分割區索引鍵) 或複合 (分割區索引鍵與排序索引鍵的組合)。

建議設計應用程式，使其在資料表內所有分割區索引鍵及其次要索引中均勻運作。您可以確定應用程式需要的存取模式，並估計每個資料表和次要索引所需的讀取和寫入單位。

> **Note**
>
> 動態調整容量適用於隨需模式與佈建容量。

每個 DynamoDB 資料表分割區設計為可提供每秒最多 3,000 個讀取單位與 1,000 個寫入單位。一個讀取容量單位代表每秒一次高度一致性讀取，或每秒兩次最終一致讀取，適用於大小上限 4 KB 的項目。一個寫入容量單位代表每秒一次寫入操作，適用於大小上限 1 KB 的項目。

評估資料表分割區輸送量限制時，必須考慮項目大小。例如，若資料表項目大小為 20 KB，單次一致性讀取操作將耗用 5 個讀取單位。這表示在達到分割區限制前，可同時對該項目每秒執行 600 次一致性讀取操作。所有分割區的總輸送量可能受佈建模式的設定輸送量或隨需模式下資料表層級限制所制約。如需詳細資訊，請參閱 Service Quotas。

主題

- 在 DynamoDB 中設計分割區索引鍵以有效分配工作負載
- 使用寫入碎片以在 DynamoDB 資料表中平均分配工作負載
- 在將資料上傳至 DynamoDB 時有效分配寫入活動

在 DynamoDB 中設計分割區索引鍵以有效分配工作負載

資料表主索引鍵的分割區索引鍵部分，決定了已儲存資料表資料的邏輯分割區。因此會影響底層的實體分割區。不能有效分配 I/O 請求的分割區索引鍵設計可能會造成「經常性」分割區，因此導致限流並且無法有效地使用已佈建的 I/O 容量。

資料表已佈建輸送量的最佳使用情況，不僅取決於各個項目的工作負載模式，還取決於分割區索引鍵設計。這並不表示您必須存取所有分割區索引鍵值才能達到有效率的輸送量層級，也不表示所存取之分割區索引鍵值的百分比必須很高。這表示您的工作負載存取的分割區索引鍵值越獨特，這些請求將越多地分佈在已分割區空間中。一般而言，隨著已存取分割區索引鍵值與總分割區索引鍵值的比例提升，您將更有效率地運用佈建輸送量。

以下是一些常用分割區索引鍵結構描述的已佈建輸送量效率比較。

分割區索引鍵值	一致性
使用者 ID，其中應用程式有許多使用者。	好
狀態碼，其中只有一些可能的狀態碼。	不好

分割區索引鍵值	一致性
項目建立日期，捨入到最接近的時間週期 (例如日期、小時、分鐘)。	不好
裝置 ID，其中每部裝置會依相當類似的間隔存取資料。	好
裝置 ID，其中即使追蹤多個裝置，其中一部裝置也會明顯比其他所有裝置更熱門。	不好

如果單一資料表只有少量的分割區索引鍵值，請考慮在更多相異分割區索引鍵值之間分佈您的寫入操作。換句話說，請結構化主索引鍵元素，以避免某個「經常性」(高度請求) 的分割區索引鍵值使整體效能變慢。

例如，請考慮使用具有複合主鍵的資料表。分割區索引鍵代表項目的建立日期，已捨入到最接近的日期。排序索引鍵是項目識別符。在指定的日期 (假設是 2014-07-09)，所有新項目都會寫入單一分割區索引鍵值 (和對應的實體分割區)。

如果資料表完全符合單一分割區 (考慮到資料會隨時間成長)，而且您應用程式的讀取與寫入輸送量需求不超過單一分割區的讀取與寫入能力，則您的應用程式應不會遇到由於分割所造成的非預期限流。

若要使用適用於 DynamoDB 的 NoSQL Workbench 來協助視覺化您的分割區索引鍵設計，請參閱 使用 NoSQL Workbench 建立資料模型。

使用寫入碎片以在 DynamoDB 資料表中平均分配工作負載

在 Amazon DynamoDB 中的分割區索引鍵空間分配寫入的一個好方法是拓展空間。您可以數種不同的方式來執行此動作：您可以在分割區索引鍵值中新增隨機數字，以便在分割區之間分配項目。或者，您可以使用根據您查詢內容計算的數字。

使用隨機尾碼的碎片

要在分割區索引鍵空間中更平均地分佈負載的方法，即是在分割區索引鍵值的結尾新增一個亂數。如此您就能在較大的空間中將寫入隨機化。

例如，對於代表今天日期的分割區索引鍵，您可以選擇 1 到 200 間的一個亂數，並將其串連到日期做為尾碼。這會產生分割區索引鍵值，例如 2014-07-09.1、2014-07-09.2... 到 2014-07-09.200。因為您隨機化了分割區索引鍵，每天對資料表的寫入會平均分配在多個分割區中。這可帶來更優良的平行處理與更高的整體輸送量。

然而，若要讀取指定日期的所有項目，您需要查詢所有尾碼的項目，接著合併結果。例如，您會先對分割區索引鍵值 2014-07-09.1 發出 Query 請求。再對 2014-07-09.2 發出另一個 Query，依此類推直到 2014-07-09.200。最後，您的應用程式需要合併所有該等 Query 請求的結果。

使用計算尾碼的碎片

隨機化的策略可大幅改善寫入輸送量。由於您不知道寫入項目時所使用的尾碼值，因此很難讀取特定項目。若要讓讀取個別項目變得比較容易，您可以使用不同的策略。不使用亂數以在分割區中分佈項目，而是使用您可以計算的數字 (根據您想要查詢之項目)。

考量先前的範例，其中資料表會使用分割區索引鍵中的今天日期。現在假設每個項目有可存取的 OrderId 屬性，且您經常需要依訂單 ID (除了日期) 來尋找項目。在您的應用程式將項目寫入資料表之前，可根據訂單 ID 計算雜湊尾碼並將其附加至分割區索引鍵日期。計算應會產生介於 1 到 200 之間的數字，此分佈非常平均，與隨機策略產生的結果相當類似。

一個簡單的計算便足以說明，例如訂單 ID 中字元之 UTF-8 字碼指標值的乘積：模數 200，+ 1。分割區索引鍵值會是與計算結果串聯的日期。

有了這項策略，即可在分割區索引鍵值之間，以及實體分割區之間平均分配寫入。您可以輕鬆地對特定項目與日期執行 GetItem 操作，因為您可以計算對特定 OrderId 值的分割區索引鍵值。

若要讀取指定日期的所有項目，您仍然需要 Query 每個 2014-07-09.N 索引鍵 (其中 N 是 1-200)，而且您的應用程式需要合併所有結果。好處是您能避免讓單一「經常性」分割區索引鍵值承受所有工作負載。

> **Note**
>
> 如需為了處理大量時間序列資料而設計的有效策略，請參閱 時間序列資料。

在將資料上傳至 DynamoDB 時有效分配寫入活動

通常，當您從其他資料來源載入資料時，Amazon DynamoDB 會將資料表資料分割在多個伺服器上。如果您同時將資料上傳到所有已配置的伺服器，就能獲得更好的效能。

例如，假設您想要將使用者訊息上傳至使用複合主鍵的 DynamoDB 資料表，該資料表將 UserID 作為分割區索引鍵並將 MessageID 作為排序索引鍵。

當您上傳資料時，您可以採取的一種方法是一個接一個地上傳每位使用者的所有訊息項目：

UserID	MessageID
U1	1
U1	2
U1	...
U1	…最多可達 100
U2	1
U2	2
U2	...
U2	…最多可達 200

這個案例中的問題是您沒有將寫入請求分配到跨分割區索引鍵值的 DynamoDB。您一次需要一個分割區索引鍵值並上傳其所有項目，再轉到下一個分割區索引鍵值並執行相同操作。

在幕後，DynamoDB 會跨多個伺服器分割資料表中的資料。若要充分使用為資料表佈建的所有輸送容量，您必須將工作負載分配至各分割區索引鍵值。若您將不均勻數量的上傳工作導向所有具有相同分割區索引鍵值的項目，便無法完全使用 DynamoDB 為資料表佈建的所有資源。

您可以使用排序索引鍵，從每個分割區索引鍵值載入一個項目，再從每個分割區索引鍵值載入另一個項目，依此類推：

UserID	MessageID
U1	1
U2	1
U3	1
...	...
U1	2

UserID	MessageID
U2	2
U3	2
...	...

此序列中的每次上傳都會使用不同的分割區索引鍵值，讓更多 DynamoDB 伺服器同時處於忙碌狀態，並改善您的輸送量效能。

使用排序索引鍵組織 DynamoDB 資料的最佳實務

在 Amazon DynamoDB 資料表中，每個項目的主索引鍵可由分割區索引鍵或排序索引鍵組成，以唯一識別該項目。

精心設計的排序索引鍵有兩個主要優點：

- 它們將相關資訊集中在一個可以有效率查詢的地方。仔細設計的排序索引鍵，讓您可以透過例如 `begins_with`、`between`、`>`、`<` 等運算子，使用範圍查詢來擷取經常需要的相關項目群組。
- 透過組合排序索引鍵，可讓您在資料中定義階層式 (一對多) 關係，您可以在階層中的任何層級查詢這些關係。

  例如，在列出地理位置的表中，您可以按以下方式建構排序索引鍵。

  ```
  [country]#[region]#[state]#[county]#[city]#[neighborhood]
  ```

  這可以讓您在任何一種此彙整層級的位置列表進行有效率的範圍查詢，從 `country` 到一個 `neighborhood`，以及其之間的所有內容。

使用排序索引鍵進行版本控制

許多應用程式需要維護項目層級修訂的歷史記錄，以用於稽核或合規目的，並能輕鬆擷取最新版本。有一種有效的設計模式，是使用排序索引鍵字首來完成此操作：

- 在每個新項目建立該項目的兩個複本：一個複本在排序索引鍵字首應有版本號的字首為零 (例如`v0_`)，並且應有一個版本號的字首為一 (`v1_`)。

- 每次更新項目時，都使用更新版本的排序索引鍵中更高版本的字首，並將更新的內容複製到版本字首為零的項目中。這代表任何項目的最新版本都可以使用零字首輕鬆定位。

例如，零件製造商可能會使用如下圖所示的結構描述。

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.

`Equipment_1` 項目會經過各種稽核人員的一系列稽核。每個新稽核的結果都會在資料表中的新項目中擷取，從第一號版本開始，然後遞增每個連續修訂的編號。

增加每個新修訂版本時，應用程式層會將零版本項目 (具有等於 `v0_Audit` 的排序索引鍵) 的內容替換為新版本的內容。

無論應用程式何時需要擷取最近的稽核狀態，都可以查詢排序索引鍵開頭的 `v0_`。

如果應用程式需要擷取整個修訂歷史記錄，可查詢該項目分割區索引鍵下的所有項目並篩選掉 `v0_` 項目。

如果您在排序索引鍵開頭之後的排序索引鍵中包括個別部分 ID，此設計也適用於跨設備多個部分的稽核。

使用 DynamoDB 中次要索引的最佳實務

次要索引對於支援您應用程式所需的查詢模式而言通常很重要。同時，過度使用次要索引或未有效地使用次要索引可能會增加非必要的成本並降低效能。

內容

- DynamoDB 中次要索引的一般準則
  - 有效率地使用索引

  - 謹慎選擇投影
  - 最佳化頻繁查詢，避免擷取
  - 在建立本機次要索引時請留意項目集合大小限制
- 利用疏鬆索引
  - DynamoDB 中疏鬆索引的範例
- 使用全域次要索引以支援 DynamoDB 中的具體化聚合查詢
  - 範例案例和存取模式
  - 為什麼要預先運算彙總
  - 資料表設計
  - 使用 Streams 和 彙總管道 AWS Lambda
  - 稀疏 GSI 設計
  - 查詢 GSI
  - 考量事項
- 在 DynamoDB 中多載全域次要索引
- 在 DynamoDB 中使用全域次要索引寫入碎片，以進行選擇性資料表查詢。
  - 模式設計
  - 碎片策略
  - 查詢已分碎片的 GSI
  - 平行查詢執行注意事項
  - 程式碼範例
- 使用全域次要索引在 DynamoDB 中建立最終一致複本

DynamoDB 中次要索引的一般準則

Amazon DynamoDB 支援兩種次要索引：

- 全域次要索引 (GSI)：是包含分割區索引鍵或排序索引鍵的索引，這些索引鍵可與基礎資料表上的索引鍵不同。全域次要索引之所以視為全域，是因為索引上的查詢可以跨越所有分割區之間基礎資料表中的所有資料。全域次要索引沒有大小限制，也有自己的讀取和寫入活動佈建輸送量設定，而且這些設定與資料表的佈建輸送量設定無關。
- 本機次要索引 (LSI)：與基礎資料表擁有相同分割區索引鍵但不同排序索引鍵的索引。本機次要索引的「本機」概念是指，本機次要索引的每個分割區都會列入基礎資料表分割區的範圍，其中分割區索

引鍵的值皆相同。因此，針對每個分割區索引鍵值，所有已索引項目的大小總計不得超過 10 GB。此外，本機次要索引與其建立索引的資料表共用適用於讀取和寫入活動的佈建輸送量設定。

DynamoDB 中的每個資料表最多可以有 20 個全域次要索引 (預設配額) 以及 5 個本機次要索引。

全域次要索引通常比本機次要索引更實用。決定要使用的索引類型也將取決於應用程式的需求。如需比較全域與本機次要索引並了解選擇準則的詳細資訊，請參閱 the section called “使用索引”。

以下是在 DynamoDB 中建立索引時需要謹記在心的一些常見原則與設計模式：

主題

- 有效率地使用索引
- 謹慎選擇投影
- 最佳化頻繁查詢，避免擷取
- 在建立本機次要索引時請留意項目集合大小限制

有效率地使用索引

將索引數量降到最低。請不要在您不常查詢的屬性上建立次要索引。不常使用的索引不僅會增加儲存體和 I/O 成本，而不會改善應用程式的效能。

謹慎選擇投影

由於次要索引會使用儲存體和佈建輸送量，建議您將索引的大小維持得越小越好。此外，相較於查詢完整資料表，索引的大小越小所能帶來的效能優勢越大。若您的查詢通常只會傳回一小部分的屬性，且這些屬性的總大小遠小於整個項目的大小，建議您只投影會定期請求的屬性。

相較於讀取，若您預期資料表上將會有許多寫入活動，遵循這些最佳實務：

- 考慮投影少量的屬性，以將寫入到索引的項目大小減至最小。但是，此作法僅適用於若投影屬性大小大於單一寫入容量單位 (1 KB)。例如，若一個索引項目的大小只有 200 位元組，DynamoDB 會將此大小向上四捨五入為 1 KB。換句話說，只要索引項目很小，您便可以投影更多屬性，而無須支付額外的成本。
- 避免投影您已知在查詢中很少會需要用到的屬性。每次您更新在索引中投影的屬性後，更新索引也不會產生額外的成本。您仍可以更高佈建輸送量成本在 `Query` 中擷取非投影屬性，但查詢成本可能會比經常更新索引的成本少很多。
- 只有在您希望查詢傳回不同排序索引鍵排序的整個資料表項目，才指定 `ALL`。為所有屬性建立投影之後便不再需要擷取資料表，但在大多數的案例中，這樣會讓您的儲存體和寫入活動成本加倍。

如下一區段所述，平衡盡可能將索引大小降到最低的需求來達成將擷取減到最低的需求。

最佳化頻繁查詢，避免擷取

若要在延遲盡可能最低的前提下取得最快速的查詢，請投影所有您預期該查詢要傳回的屬性。特別是，如果您對未投影的屬性查詢本機次要索引，DynamoDB 會自動從需要在資料表讀取整個項目的資料表擷取那些屬性。此引發的延遲與其他 I/O 操作是可以避免的。

請謹記「偶爾」的查詢可能經常是「必要」的查詢。若因為您預計偶爾才會查詢某些屬性，而不打算投影，請考慮可能的結果，否則您可能會後悔沒有投影那些屬性。

如需資料表擷取的詳細資訊，請參閱 本機次要索引的佈建輸送量考量。

在建立本機次要索引時請留意項目集合大小限制

項目集合是資料表和其本機次要索引中所有具有相同分割區索引鍵的項目。由於項目集合的大小不可超過 10 GB，可能會耗盡特定分割區索引鍵值的空間。

當您新增或更新資料表項目時，DynamoDB 會更新任何受到影響的本機次要索引。若建立索引的屬性已在資料表中定義，本機次要索引便會一同成長。

當您建立本機次要索引時，建議您思考會有多少資料寫入索引，以及那些資料項目中有多少具有相同的分割區索引鍵。若您預期具有特定分割區索引鍵值的資料表和索引項目總和可能會超過 10 GB，請考慮是否要避免建立索引。

若您無法避免建立本機次要索引，您必須預計項目集合的大小上限，並在超過之前採取動作。建議在撰寫項目時使用 `ReturnItemCollectionMetrics` 參數，以監控並警示項目集合大小接近 10GB 限制時的狀況。超出項目集合大小上限時，寫入嘗試將失敗。可在項目集合大小影響應用程式之前進行監控與警示，以預防相關問題。

> **Note**
>
> 建立後無法刪除本機次要索引。

如需取得在限制內操作及採取正確動作的策略，請參閱「項目集合大小限制」。

利用疏鬆索引

對於資料表中的任何項目，只有在索引鍵屬性存在於項目中時，DynamoDB 才會寫入對應的索引項目。 對於全域次要索引，這表示必須在項目上定義索引分割區索引鍵，如果索引也有排序索引鍵，則

該屬性也必須存在。如果某個項目缺少任一索引鍵屬性，則該項目不會出現在索引中。只顯示基礎資料表中項目子集的索引稱為稀疏索引。

稀疏索引在對小型子區塊的資料表進行查詢時很有用。例如，假設您有一個資料表，其中存放您所有的客戶訂單並內含以下索引鍵屬性：

- 分割區索引鍵：CustomerId
- 排序索引鍵：OrderId

若要追蹤開啟的訂單，您可以在尚未運送的項目訂單中插入名為 isOpen 的屬性。然後，隨著訂單運送時，您可以刪除該屬性。當您在 CustomerId (分割區索引鍵) 和 isOpen (排序索引鍵) 上建立索引時，只有已定義 isOpen 的那些訂單會顯示在其中。當您有上千筆訂單，但只有部分的訂單處於開啟狀態，對開啟訂單查詢索引會比掃描整個資料表快得多且節省更多費用。

相較於使用 isOpen 之類的類型屬性，您可以使用對索引排序順序有用的屬性值。例如，您可以使用設定為訂單下訂日期的 OrderOpenDate 屬性，接著在訂單履行後將其刪除。如此一來，在您查詢稀疏索引時，系統會傳回依訂單下訂的日期所排序的項目。

DynamoDB 中疏鬆索引的範例

全域次要索引依預設是稀疏的。在您建立全域次要索引時，指定一個分割區索引鍵和 (選擇性) 一個排序索引鍵。只有包含必要金鑰屬性的基礎資料表中的項目才會出現在索引中。如果項目缺少索引分割區索引鍵，或定義排序索引鍵時，該項目會從索引中排除。

透過將全域次要索引設計為稀疏，您可以使用低於基礎資料表的寫入輸送量來佈建該索引，同時仍達成優異的效能。

例如，遊戲應用程式可能會追蹤每位使用者的所有比分，但一般來說只需要查詢少數較高的計分。以下設計模式能有效處理這種情況：

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)*

在此範例中，Rick 玩了三款遊戲並在其中一款遊戲中達到 Champ 狀態。Padma 玩了四款遊戲並在其中兩款遊戲中達到 Champ 狀態。請注意，Award 屬性僅在使用者獲得獎項時的項目中才會出現。關聯的全域次要索引看起來應如下：

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

全域次要索引僅包含常受到查詢的高計分，也就是基礎資料表中的小子集項目。

使用全域次要索引以支援 DynamoDB 中的具體化聚合查詢

快速變更的資料中維護接近即時彙總和索引鍵指標，對於企業來說變得越來越重要。例如，音樂庫可能想要近乎即時地展示其下載次數最多的歌曲，或者電子商務平台可能需要按類別顯示趨勢產品。

由於 DynamoDB 本身不支援類似SUM或COUNT跨項目的彙總操作，因此在讀取時間運算這些值需要掃描大量項目，這可能會很慢且昂貴。相反地，您可以在資料變更時預先計算彙總，並將結果儲存為資料表中的一般項目。此模式稱為具體化彙總。

主題

- 範例案例和存取模式
- 為什麼要預先運算彙總
- 資料表設計
- 使用 Streams 和 彙總管道 AWS Lambda
- 稀疏 GSI 設計
- 查詢 GSI
- 考量事項

範例案例和存取模式

考慮具有下列要求的音樂程式庫應用程式：

- 應用程式會以大量（每秒數千首）記錄個別歌曲下載。
- 使用者需要查看特定月份中下載最多的單位數毫秒延遲歌曲。
- 應用程式也需要支援查詢，例如「本月前 10 首歌曲」和「特定月份下載的所有歌曲」。

掃描所有下載記錄的讀取時間運算下載計數在此規模下可能很昂貴。反之，您可以維護每次下載時更新的執行中計數，並以支援有效查詢的方式存放。

為什麼要預先運算彙總

有數種方法來運算彙總。下表比較常見的替代方案，並說明 DynamoDB 中的具體化彙總通常最適合這種使用案例的原因。

方法	取捨	使用情況
讀取時掃描和計數	需要讀取每個查詢的所有下載記錄。延遲會隨著資料量而增加，並消耗大量的讀取容量。	僅適用於不考慮延遲的極小型資料集。
外部彙總存放區（例如，Amazon ElastiCache)	使用要管理的個別服務新增操作複雜性。DynamoDB 和快取之間需要同步邏輯。	當您需要低於一毫秒的讀取，或超過簡單計數的複雜彙總邏輯時。

方法	取捨	使用情況
寫入時的應用程式層級彙總	將彙總邏輯耦合至寫入路徑。如果應用程式在記錄下載之後但在更新計數之前失敗，則彙總會變得不一致。	當您需要同步、高度一致的彙總時，可以容忍新增的寫入延遲。
使用 Streams 和 Lambda 進行具體化彙總	將彙總與寫入路徑分離。彙總最終一致（通常落後 秒）。新增 Lambda 調用成本。	當您需要近乎即時且讀取延遲低的彙總時， 可以容忍最終一致性。這是此頁面所述的方法。

具體化彙總方法可讓寫入路徑簡單（只要記錄下載）、將彙總卸載至非同步程序，並將結果存放在 DynamoDB 中，可在其中查詢單一位數毫秒延遲。

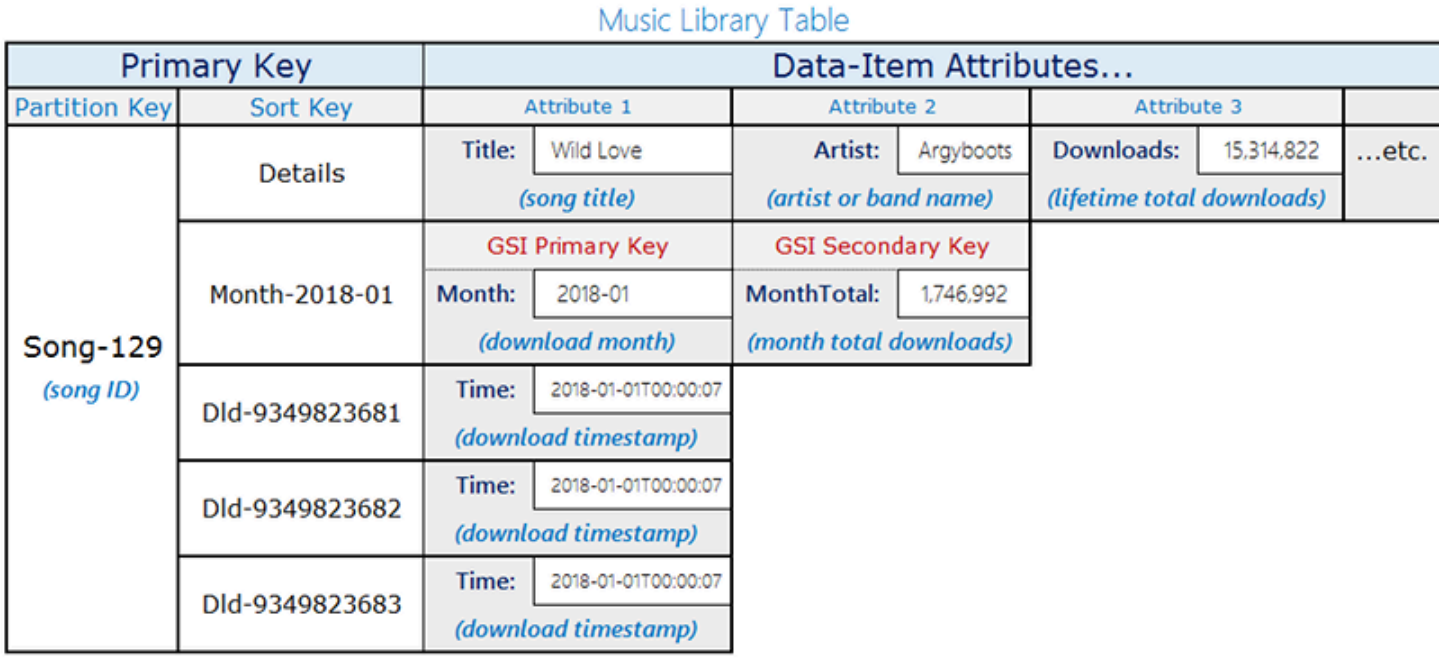
資料表設計

此設計使用具有兩個項目類型的單一資料表，這些項目類型共用相同的分割區索引鍵 (songID)，但使用不同的排序索引鍵模式來區分它們：

- 下載記錄 – 個別下載事件。排序索引鍵是 DownloadID（每次下載的唯一識別符）。
- 每月彙總項目 – 每月每首歌曲的預先計算下載計數。排序索引鍵是YYYY-MM格式的月份（例如 2018-01)。這些項目也包含執行中總數的DownloadCount屬性。

只有每月彙總項目包含 Month 屬性。此區別對於稍後描述的稀疏 GSI 設計很重要。

下圖顯示兩種項目類型的資料表配置：



項目類型	分割區索引鍵 (songID)	排序索引鍵	其他屬性
下載記錄	song1	download-abc123	UserID, Timestamp
每月彙總	song1	2018-01	Month=2018-01, DownloadCount =1,746,992

使用 Streams 和 彙總管道 AWS Lambda

彙總管道的運作方式如下：

1. 下載歌曲時，應用程式會使用 Partition-Key=songID和 將新項目寫入資料表Sort-Key=DownloadID。
2. DynamoDB Streams 會將此寫入擷取為串流記錄。
3. 連接至串流的 Lambda 函數會處理新記錄。它會識別 songID和目前月份，然後透過遞增 DownloadCount 屬性來更新對應的每月彙總項目。
4. 然後，更新的彙總項目可用於透過稀疏 GSI 進行查詢。

Lambda 函數使用具有 ADD表達式的 UpdateItem呼叫，以原子方式增加下載計數。這可避免read-modify-write競爭條件：

```
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('MusicLibrary')

def handler(event, context):
    for record in event['Records']:
        if record['eventName'] == 'INSERT':
            new_image = record['dynamodb']['NewImage']
            song_id = new_image['songID']['S']
            # Derive the month from the download timestamp
            timestamp = new_image['Timestamp']['S']
            month = timestamp[:7]  # Extract YYYY-MM

            table.update_item(
                Key={
                    'songID': song_id,
                    'SK': month
                },
                UpdateExpression='ADD DownloadCount :inc SET #m = :month',
                ExpressionAttributeNames={
                    '#m': 'Month'
                },
                ExpressionAttributeValues={
                    ':inc': 1,
                    ':month': month
                }
            )
```

> **Note**
>
> 如果 Lambda 執行在寫入更新的彙總值後失敗，可能會重試串流記錄。由於ADD操作會在每次執行時遞增計數，因此針對相同的下載，重試會多次遞增計數，讓您擁有近似值。對於大多數分析和排行榜使用案例，這個小範圍的錯誤是可接受的。如果您需要確切計數，請考慮新增冪等性邏輯，例如，使用檢查特定 是否已DownloadID處理的條件表達式。

稀疏 GSI 設計

若要有效率地查詢彙總結果，請使用下列索引鍵結構描述建立全域次要索引：

- GSI 分割區索引鍵：Month（字串）
- GSI 排序索引鍵：DownloadCount（數字）

此 GSI 是稀疏的，因為只有每月彙總項目包含 Month 屬性。個別下載記錄沒有此屬性，因此它們會自動從索引中排除。這表示 GSI 僅包含預先計算的彙總項目，這是資料表中項目總數的一小部分。

稀疏 GSI 提供兩個主要優點：

- 降低成本 – 因為只有彙總項目會複寫至索引，相較於包含資料表中每個項目的索引，您所使用的寫入容量和儲存體會大幅降低。
- 更快速的查詢 – 索引僅包含您需要查詢的資料，因此讀取效率高，並傳回單位數毫秒延遲的結果。

如需稀疏索引如何運作的詳細資訊，請參閱 利用疏鬆索引。

查詢 GSI

設置稀疏 GSI 後，您可以有效地回答多種類型的查詢：

取得特定月份下載次數最多的歌曲：

```
aws dynamodb query \
    --table-name "MusicLibrary" \
    --index-name "MonthDownloadsIndex" \
    --key-condition-expression "#m = :month" \
    --expression-attribute-names '{"#m": "Month"}' \
    --expression-attribute-values '{":month": {"S": "2018-01"}}' \
    --scan-index-forward false \
    --limit 1
```

設定 ScanIndexForward 依DownloadCount遞減順序false排序結果，並僅Limit=1傳回最熱門的歌曲。

取得指定月份的前 10 首歌曲：

```
aws dynamodb query \
```

```
    --table-name "MusicLibrary" \
    --index-name "MonthDownloadsIndex" \
    --key-condition-expression "#m = :month" \
    --expression-attribute-names '{"#m": "Month"}' \
    --expression-attribute-values '{":month": {"S": "2018-01"}}' \
    --scan-index-forward false \
    --limit 10
```

取得指定月份下載的所有歌曲（依下載計數排序）：

```
aws dynamodb query \
    --table-name "MusicLibrary" \
    --index-name "MonthDownloadsIndex" \
    --key-condition-expression "#m = :month" \
    --expression-attribute-names '{"#m": "Month"}' \
    --expression-attribute-values '{":month": {"S": "2018-01"}}' \
    --scan-index-forward false
```

考量事項

實作此模式時，請記住下列事項：

- 最終一致性 – 彙總值會透過 DynamoDB Streams 和 Lambda 非同步更新。在記錄下載和更新彙總之間，通常會有幾秒鐘的延遲。這表示 GSI 會反映近乎即時的資料，而不是即時的資料。
- Lambda 並行 – 如果您的資料表具有高寫入量，則多個 Lambda 調用可能會嘗試同時更新相同的彙總項目。原子ADD操作可安全地處理此問題，但您應該監控 Lambda 並行和限流指標，以確保您的函數可以跟上串流的進度。
- GSI 寫入容量 – 由於稀疏 GSI 僅包含彙總項目，因此需要的寫入容量遠低於基礎資料表。不過，您仍應佈建足夠的容量（或使用隨需模式）來處理彙總更新的速率。
- 約略計數 – 如前所述，Lambda 重試可能會導致計數稍微過度計數。對於需要確切計數的使用案例，請在 Lambda 函數中實作冪等性檢查。

在 DynamoDB 中多載全域次要索引

雖然 Amazon DynamoDB 每個資料表具有 20 個全域次要索引的預設配額，實際上，您可以在遠超過 20 個以上的資料欄位間進行索引。與在關聯式資料庫管理系統 (RDBMS) 中的資料表不同 (其中結構描述是統一的)，DynamoDB 中的資料表可以一次保留許多不同種類的資料項目。此外，在不同項目中的相同屬性可能包含完整不同種類的資訊。

考量 DynamoDB 資料表配置的下列範例，該資料表中會儲存各種不同的資料。

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.

Data 屬性 (這是所有項目共有的屬性) 會根據其父項目擁有不同的內容。如果您為使用資料表排序索引鍵做為其分割區索引鍵和 Data 屬性做為其排序索引鍵的資料表建立全域次要索引，您可以使用單一全域次要索引來進行多種不同的查詢。這些查詢可能包含以下項目：

- 使用 Employee_Name 作為分割區索引鍵值並將員工的姓名 (例如 Murphy, John) 作為排序索引鍵值，即可在全域次要索引中按姓名查找員工。
- 使用全域次要索引，透過搜尋倉庫 ID (像是 Warehouse_01) 來尋找在特定倉庫工作的所有員工。
- 取得最近的雇用清單，查詢在 HR_confidential 上的全域次要索引作為分割區索引鍵值，並在排序索引鍵值中使用日期範圍。

在 DynamoDB 中使用全域次要索引寫入碎片，以進行選擇性資料表查詢。

當需要在特定時間範圍內查詢最新資料時，DynamoDB 對大多數讀取作業必須提供分割區索引鍵的設計，可能會造成挑戰。為解決此情境，可使用寫入碎片與全域次要索引 (GSI) 的組合，實作高效查詢模式。

此方法可高效率地擷取並分析時間敏感資料，無需執行完整資料表掃描，避免資源消耗與高成本。透過策略性設計資料表結構與索引，可建立支援時間導向資料擷取的彈性解決方案，同時維持最佳效能。

主題

- 模式設計
- 碎片策略
- 查詢已分碎片的 GSI
- 平行查詢執行注意事項
- 程式碼範例

模式設計

使用 DynamoDB 時，可透過結合寫入碎片與全域次要索引的進階模式，在近期資料範圍內實現靈活且高效的查詢，以克服時間導向資料擷取挑戰。

資料表結構

- 分割區索引鍵 (PK)："Username"

GSI 結構

- GSI 分割區索引鍵 (PK_GSI)："ShardNumber#"
- GSI 排序索引鍵 (SK_GSI): ISO 8601 時間戳記 (例如 "2030-04-01T12:00:00Z")

[TABLE] - TimeBounded Metadata Actions

PK	SK_GSI	Address	PK_GSI
Trenton69	2023-12-05T06:40:31.312Z	117 Hudson Divide	ShardNumber#0
Scot_Langworth-Prosacco	2024-10-09T14:44:45.819Z	84021 Herzog Skyway	ShardNumber#6
Juliet.Morissette	2025-03-28T07:20:56.538Z	4871 N Broadway	ShardNumber#4
Kay.Von71	2024-11-23T16:19:26.763Z	14554 Odell Throughway	ShardNumber#2
Kiarra43	2023-11-15T16:21:57.078Z	8471 London Road	ShardNumber#1
Marcella91	2024-12-17T09:02:31.623Z	10271 Nick Crescent	ShardNumber#6
Bernard.Lemke36	2025-09-09T00:18:34.482Z	5438 Douglas Groves	ShardNumber#2
Daisha83	2024-01-24T06:38:08.482Z	8786 Armstrong Radial	ShardNumber#3
Marcos_Schmitt58	2024-04-06T17:38:58.551Z	510 S Center Street	ShardNumber#3
Jeremie_VonRueden-Mills	2025-04-06T04:24:23.701Z	157 Koch Drives	ShardNumber#1
Verna28	2025-10-10T18:42:21.059Z	70040 Baylee Trafficway	ShardNumber#1
Trycia.Doyle	2024-01-10T17:00:08.360Z	9511 Tillman Extensions	ShardNumber#2

碎片策略

假設使用 10 個碎片，碎片編號可設定為 0 至 9。記錄活動時，需計算碎片編號 (例如：以使用者 ID 經雜湊函數運算後取碎片數量的模數)，並將結果前置於 GSI 分割區索引鍵。此方法可將項目平均分佈至多個碎片，有效降低熱分割區風險。

查詢已分碎片的 GSI

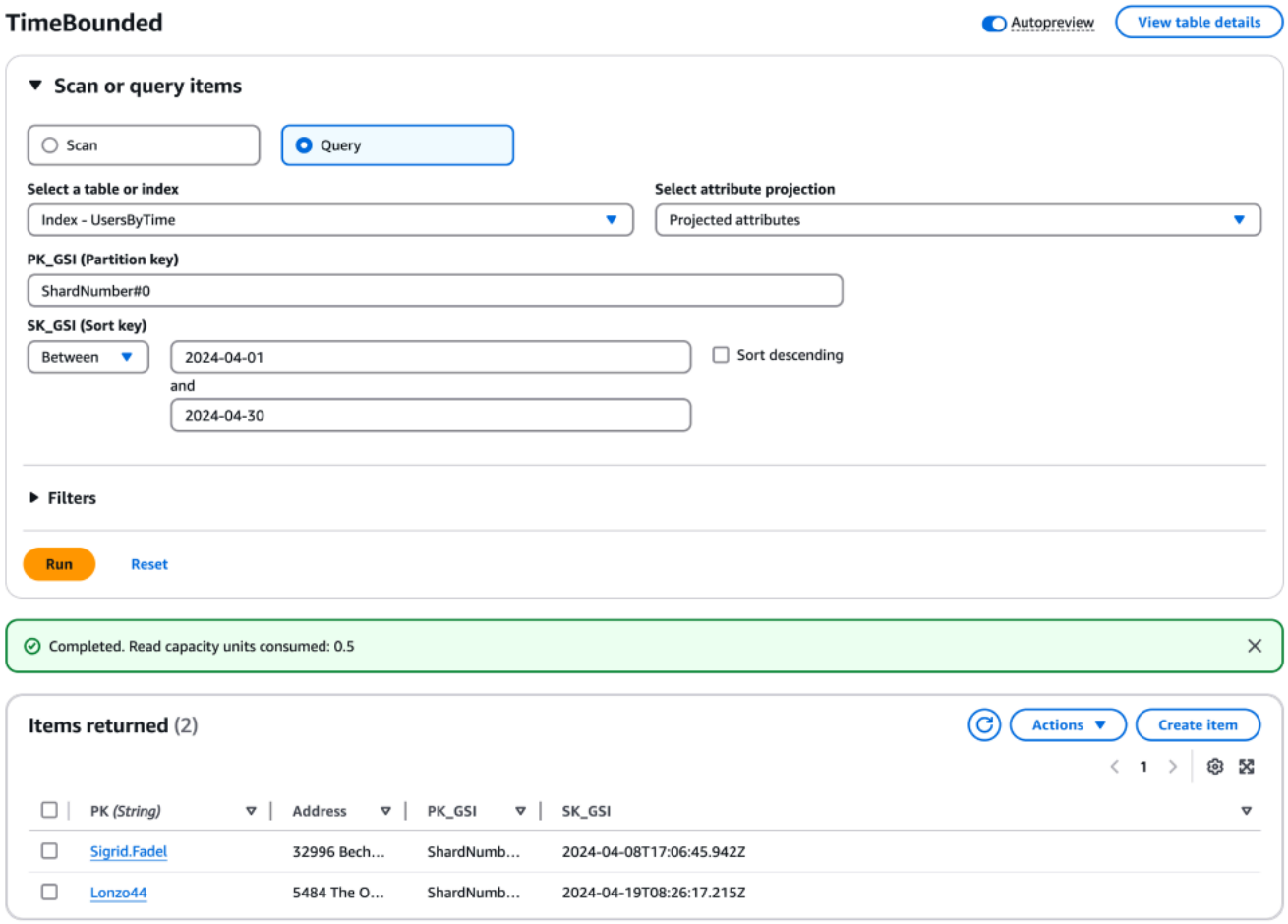
在 DynamoDB 資料表中，若需在特定時間範圍內查詢分散於多個碎片的項目，必須採用與單一分割區查詢不同的方式。由於 DynamoDB 查詢操作一次僅支援單一分割區索引鍵，因此無法以單一查詢跨多個碎片。不過，可透過應用程式層邏輯執行多次查詢，每次針對特定碎片，並在彙總階段整合結果以達成目標。以下程序說明具體執行步驟。

查詢與彙總碎片

1. 識別碎片策略中使用的碎片編號範圍。例如，若設有 10 個碎片，則編號範圍為 0–9。
2. 針對每個碎片建構並執行查詢，以擷取目標時間範圍內的資料項目。可平行執行查詢以提升整體效能。查詢時，使用包含碎片編號的分割區索引鍵，並以時間範圍設定排序索引鍵。以下示範單一碎片的查詢範例：

```
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"}
    }'
```



可為各碎片重複此查詢，並依序調整分割區索引鍵 (例如："ShardNumber#1"、"ShardNumber#2" ... "ShardNumber#9")。

3. 所有查詢完成後，彙總各碎片查詢結果。在應用程式程式碼中執行彙總，將各查詢結果合併成單一資料集，以呈現指定時間範圍內所有碎片項目。

平行查詢執行注意事項

每次查詢都會消耗資料表或索引的讀取容量單位。若使用預先佈建輸送量，請確保資料表具備足夠容量以應付多重平行查詢的尖峰負載。若您使用隨需容量，請留意可能產生的成本影響。

程式碼範例

若要使用 Python 在 DynamoDB 中跨碎片執行平行查詢，您可使用 boto3 函式庫，它是適用於 Python 的 Amazon Web Services SDK。此範例假設您已安裝 boto3 並使用適當的 AWS 登入資料進行設定。

下列 Python 程式碼示範如何在指定時間範圍內，跨多個碎片執行平行查詢。程式使用 concurrent.futures 模組以平行方式執行查詢，相較於循序執行可縮短整體執行時期。

```
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)}")
```

執行此程式碼前，請確保以 DynamoDB 設定中的實際資料表名稱與 GSI 名稱取代 `YourTableName` 和 `YourGSIName`。此外，請依您的具體需求調整 `total_shards`、`time_start` 和 `time_end` 變數。

此指令碼會查詢各碎片中指定時間範圍內的項目，並彙總結果。

使用全域次要索引在 DynamoDB 中建立最終一致複本

使用全域次要索引建立資料表的最終一致複本。建立複本可讓您執行下列動作：

- 為不同的讀取器設定不同的佈建讀取容量。例如，假設您有兩個應用程式：一個應用程式處理高優先順序查詢，並需要最高層級的讀取效能；另一個應用程式則處理低優先順序查詢，且容許限流讀取活動。

  如果這兩個應用程式都從同一個資料表讀取資料，則來自低優先順序應用程式的高讀取負載可能會耗用資料表的所有可用讀取容量。這將限流高優先級應用程式的讀取活動。

  相反地，您可以透過全域次要索引建立複本，其讀取容量可與資料表本身分開設定。然後，您可以讓低優先級的應用程式查詢複本而非資料表。

- 完全消除資料表中的讀取。例如，您可能有一個應用程式會從網站擷取大量點擊流活動，但您不想冒著讓讀取干擾該活動的風險。您可以隔離此資料表並避免其他應用程式讀取 (請參閱 使用 IAM 政策條件進行精細定義存取控制)，同時讓其他應用程式讀取使用全域次要索引建立的複本。

若要建立複本，請設定具有與資料表相同的索引鍵結構描述的全域次要索引，並將部分或全部的非索引鍵屬性投影到其中。在應用程式中，您可以將部分或所有讀取活動導向此全域次要索引，而不是父資料表。然後，您可以調整全域次要索引的佈建讀取容量，以便在不必變更父資料表的佈建讀取容量情況下處理該讀取。

寫入父資料表與寫入資料出現在索引中的時間之間，總是會有短暫的傳播延遲。換句話說，您的應用程式應該考慮到全域次要索引複本只是與父資料表有最終一致性。

您可以建立多個全域次要索引複本來支援不同的讀取模式。在建立複本時，只需投入每個讀取模式實際需要的屬性。然後，應用程式可以耗用較少佈建的讀取容量來僅取得所需的資料，而不必從父資料表讀取項目。隨著時間推移，這項最佳化可以大幅節省成本。

在 DynamoDB 中儲存大型項目與屬性的最佳實務

Amazon DynamoDB 限制您在資料表中儲存的每個項目大小為 400 KB (請參閱 項目大小)。若您的應用程式需要在項目中存放比 DynamoDB 大小限制所允許大小更多的資料，您可以嘗試壓縮一或多個大型屬性，或將項目分成多個項目 (即以排序索引鍵編製索引)。您也可以將項目作為物件存放在 Amazon Simple Storage Service (Amazon S3) 中，接著在您的 DynamoDB 項目中存放 Amazon S3 物件的識別符。

根據最佳實務，撰寫項目時應使用 `ReturnConsumedCapacity` 參數，以監控並警示項目大小是否接近 400 KB 上限。若超出項目大小上限，寫入作業將會失敗。DynamoDB 將回傳 ValidationException 錯誤。透過監控與警示項目大小，您可在問題影響應用程式前先行防範。

壓縮大型屬性值

壓縮大型屬性值可以讓它們符合 DynamoDB 中的項目限制，並減少您的儲存成本。壓縮演算法 (如 GZIP 或 LZO) 會產生二進位輸出，可儲存在項目中的 `Binary` 屬性類型中。

例如，可考慮一個儲存論壇使用者訊息的資料表。這類訊息通常包含長文字字串，適合作為壓縮對象。雖然壓縮可減少項目大小，但壓縮後的屬性值無法用於篩選。

如需示範如何在 DynamoDB 中壓縮這類訊息的範本程式碼，請參閱以下內容：

- 範例：使用 適用於 Java 的 AWS SDK 文件 API 處理二進位類型屬性

- 範例：使用 適用於 .NET 的 AWS SDK 低階 API 處理二進位類型屬性

垂直分割

處理大型項目的另一種方法是將其拆分為較小的資料區塊，並依分割區索引鍵值將所有相關項目建立關聯。接著，您可以使用排序索引鍵字串來識別與之相關的儲存資訊。透過此作法，將多個項目依相同的分割區索引鍵值分組時，您便建立了項目集合。

如需此方法的詳細資訊，請參閱：

- 使用垂直分割在 Amazon DynamoDB 中有效率地擴展資料
- 使用 在 Amazon DynamoDB 中實作垂直分割 AWS Glue

在 Amazon S3 中存放大型屬性值

如先前所述，您可以使用 Amazon S3 來存放在 DynamoDB 項目中不符合的大型屬性值。您可以將它們作為物件存放在 Amazon S3 中，接著在您的 DynamoDB 項目中存放物件的識別符。

您也可以使用 Amazon S3 中的物件中繼資料支援，將連結提供回 DynamoDB 中的父項目。在 Amazon S3 中以物件 Amazon S3 中繼資料的形式存放項目的主索引鍵值。此通常可協助維護 Amazon S3 物件。

例如，請參考 `ProductCatalog` 資料表。此資料表中的項目會存放項目價格、描述、書籍作者和其他產品維度的資訊。如果您希望存放的每個產品影像太大而無法在項目中容納，您可以將影像存放在 Amazon S3 而不是在 DynamoDB 中。

實作此策略時，請注意下列各項：

- DynamoDB 不支援跨 Amazon S3 和 DynamoDB 的交易。因此，應用程式能必須處理任何錯誤，其可能會包含清理遺棄的 Amazon S3 物件。
- Amazon S3 會限制物件識別碼的長度。因此，您必須以不會產生過長物件識別碼或違反其他 Amazon S3 限制的方法來整理您的資料。

有關如何使用 Amazon S3 的詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》。

在 DynamoDB 中處理時間序列資料的最佳實務

Amazon DynamoDB 中的一般設計原則會建議您盡量減少使用的資料表數量。對於大多數應用程式，您只需要一張資料表。不過，對於時間序列資料，您通常可以在每個期間針對每個應用程式使用一個資料表，這是最好的處理方式。

時間序列資料的設計模式

假設一個典型的時間序列情況，這時您想追蹤大量事件。您的寫入存取模式是所有正在記錄的事件都記有今天的日期。您的讀取存取模式可能是讀取今天最頻繁的活動，昨天的活動則較少讀存，然後更舊的活動則極少讀存。處理這種情況的一個方法，就是將目前的日期和時間內建到主索引鍵中。

以下設計模式通常能有效處理這種情況：

- 每個期間建立一個資料表，並且佈建所需的讀取和寫入容量，以及必要的索引。
- 在每個期間結束之前，先為下一個期間預先建立資料表。在目前時段結束時，將事件流量引導至新資料表。您可以為這些資料表指派名稱，來為其所記錄的期間命名。
- 一旦資料表不再被寫入，會將其佈建的寫入容量減少到較低的值 (例如，1 個 WCU)，並佈建適當的讀取容量。隨著舊資料表的存在時間拉長，逐漸地縮減其佈建的讀取容量。您可以針對內容極少用到或不再需要的資料表，選擇封存或刪除這些資料表。

這個概念是針對目前將會產生最高流量的期間，配置所需的資源，然後針對目前未經常使用的舊資料表，縮減佈建的容量，以節省成本。根據您的業務需求，您可能需要考慮寫入碎片的方法，來將流量平均地分配到邏輯分割區索引鍵。如需詳細資訊，請參閱使用寫入碎片以在 DynamoDB 資料表中平均分配工作負載。

時間序列資料表範例

下列是時間序列資料的範例，其中針對目前的資料表佈建了較高的讀取/寫入容量，並縮減舊資料表的佈建容量，因為這些資料表不常存取。

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	...
• • •	• • •	• • •	• • •	...

管理 DynamoDB 資料表中多對多關係的最佳實務

相鄰清單是一設計模式，在 Amazon DynamoDB 中建立多對多關係的模型時非常有用。整體來說，它們提供在 DynamoDB 中呈現圖形資料 (節點和邊緣) 的方式。

相鄰清單設計模式

當應用程式的不同實體彼此有多對多關係時，關係可以建立為相鄰清單的模型。在此模式中，所有頂層實體 (相當於圖形模型的節點) 會使用分割區索引鍵來呈現。任何具其他實體的關係 (圖形中的邊緣) 會透過將排序索引鍵值設為目標實體 ID (目標節點) 來以分割區中項目的形式呈現。

此模式的優點包含可將資料複製降到最低並簡化查詢模式，以尋找與目標實體相關 (有目標節點的邊緣) 的所有實體 (節點)。

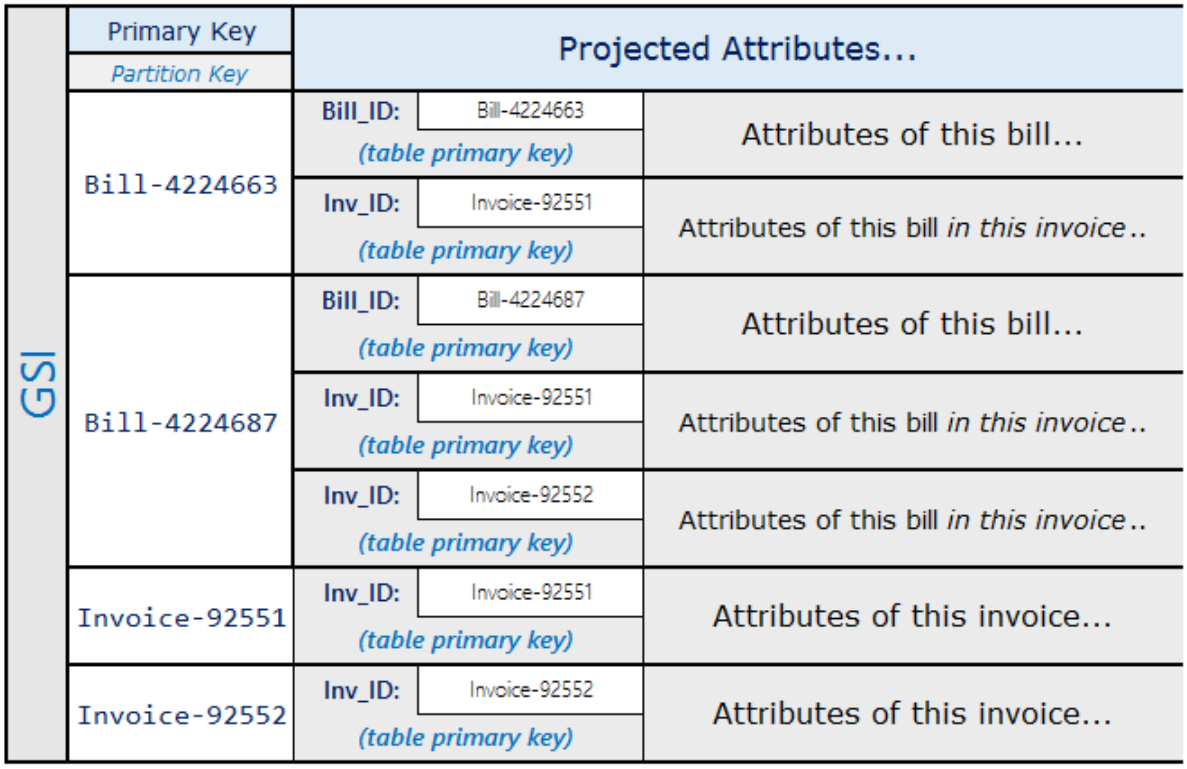
此模式的實用實際範例為可開立多帳單發票的發票系統。一個帳單可屬於多個發票。此範例中的分割區索引鍵是 InvoiceID 或 BillID。BillID 分割區擁有專屬於帳單的所有屬性。InvoiceID 分割區有儲存特定發票屬性的項目，且擁有累計至發票的每個 BillID 項目。

結構描述看起來類似如下。

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...

使用上述結構描述，您會發現可使用資料表上的主索引鍵來查詢發票的所有帳單。若要查詢包含部分帳單的所有發票，針對資料表的排序索引鍵建立全域次要索引。

全域次要索引的投影看起來類似如下。



具體化圖形模式

許多應用程式會根據對等間的排名、實體間的一般關係、鄰接實體狀態和圖形樣式流程的其他類型來建置。如需這些類型的應用程式，請考量以下結構描述設計模式。

TABLE	Primary Key		Attributes		
	PK (NodeId)	SK (TypeTarget, GSI 2 SK)	Data	GSI PK	Graph Projections
	1	DATE|2|BIRTH	1980-12-19	Hash(Person.Data)	
		PERSON|1	Data (GSI1 SK)	GSI PK	
			John Doe	Hash(Person.Data)	
		PERSON|5|FRIEND	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PLACE|4|BIRTH	Data	GSI PK	
			USA|Texas|Austin	Hash(Person.Data)	
		SKILL|6	Data	GSI PK	
			Java Developer|Senior	Hash(Person.Data)	
	2	DATE|2	Data	GSI PK	
			1980-12-19	0	
	3	PLACE|3	Data	GSI PK	
			UK|England|London	0	
	4	PLACE|4	Data	GSI PK	
			USA|Texas|Austin	0	...
	5	DATE|2|BIRTH	Data	GSI PK	
			1980-12-19	Hash(Person.Data)	
		PERSON|5	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PERSON|1|FRIEND	Data	GSI PK	
			John Doe	Hash(Person.Data)	
		PLACE|3|BIRTH	Data	GSI PK	
			UK|England|London	Hash(Person.Data)	
		SKILL|7	Data	GSI PK	
			Guitar|Advanced	Hash(Person.Data)	
	6	SKILL|6	Data	GSI PK	
			Java Developer	0	
	7	SKILL|7	Data	GSI PK	
			Guitar	0	

	Primary Key		Attributes		
	GSI PK	GSI 1 SK (Data)			
GSI 1	O-N	1980-12-19	NodeID	TypeTarget	Graph Projections
			2	DATE|2	...
			NodeID	TypeTarget	
			1	DATE|2|BIRTH	
			NodeID		
			5		
		Guitar	NodeID	TypeTarget	
			7	SKILL|7	
		Guitar|Advanced	NodeID		
			5		
		Jane Smith	NodeID	TypeTarget	
			5	Person|5	
			NodeID	TypeTarget	
			1	Person|5|FRIEND	
		Java Developer	NodeID	TypeTarget	
			6	SKILL|6	
		Java Developer|Senior	NodeID		
			1		
		John Doe	NodeID	TypeTarget	
			1	Person|1	
			NodeID	TypeTarget	
			5	Person|1|FRIEND	
		UK|England|London	NodeID	TypeTarget	
			3	PLACE|3	
			NodeID	TypeTarget	
			1	PLACE|3|BIRTH	
		USA|Texas|Austin	NodeID	TypeTarget	
			4	PLACE|4	
			NodeID	TypeTarget	
			5	PLACE|4|BIRTH	

GSI 2	Primary Key		Attributes		
	GSI PK	GSI 2 SK (TypeTarget)	NodeID	Data	Graph Projections
	O-N	DATE|2	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	

上述結構描述顯示的圖形資料結構會以內含項目的資料分割區組來定義，其內含項目會定義圖形的邊緣與節點。邊緣項目包含 `Target` 和 `Type` 屬性。這些屬性會用做為複合索引鍵「TypeTarget」的一部分，來識別在主要資料表或第二個全域次要索引中分割區中的項目。

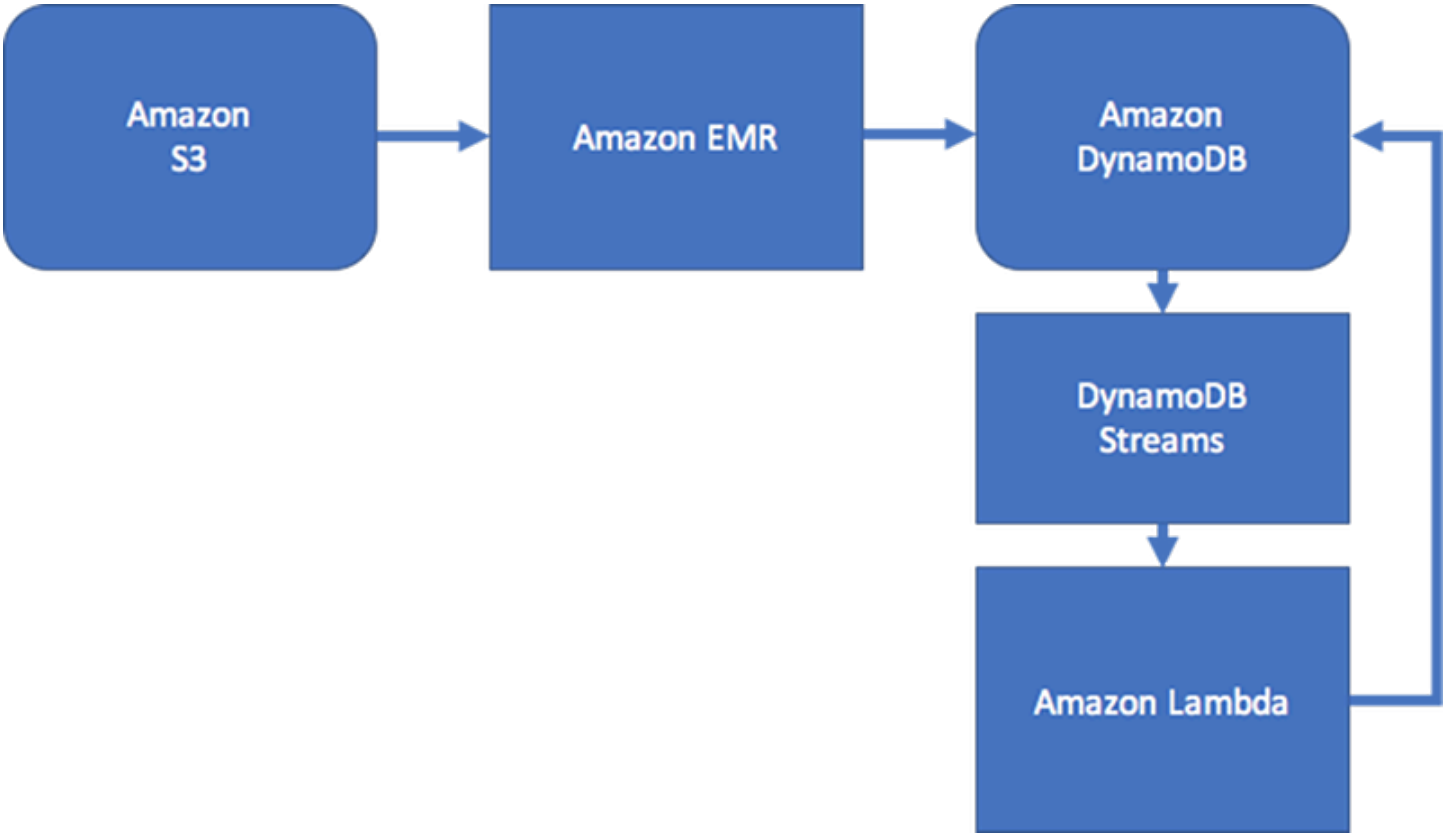
第一個全域次要索引會根據 `Data` 屬性建置。此屬性會如先前數個不同屬性類型 (也就是 `Dates`、`Names`、`Places` 和 `Skills`) 所述，使用全域次要索引多載。在這裡，一個全域次要索引可有效地對四個不同的屬性製作索引。

隨著您將項目插入資料表，您可以使用智慧碎片策略來使用大型彙總 (生日、技能)，在避免經常性讀取/寫入問題時所需的許多邏輯分割區中，對全域次要索引散佈項目。

結合設計模式後，您能得到一個功能扎實的資料儲存，可有效進行即時圖形流程。這些流程可以為建議的引擎、社交應用程式、節點排名、子樹系彙總和其他一般圖形使用者案例提供高效能鄰近實體狀態和邊緣彙總查詢。

如果您的使用案例對即時資料一致性並不敏感，您可以使用已排定的 Amazon EMR 程序，將適用於流程的相關圖形摘要彙總填入邊緣。如果您的應用程式不需要立即知道邊緣新增至圖形的時間，您可以使用排定的程序來彙總結果。

若要維持一定程度的一致性，設計可以包含 Amazon DynamoDB Streams 與 AWS Lambda 來處理邊緣更新。其可以使用 Amazon EMR 任務來驗證定期間隔的結果。下圖說明此方法。在社交應用程式會常常用到此方法，其中即時查詢的成本相當高，且立即知道個別使用者更新的需求很低。



IT 服務管理 (ITSM) 和安全性應用程式通常需要對由複雜邊際彙總構成的實體狀態變更回應做出即時回應。此類應用程式需要可以支援第二和第三層級關係之即時多節點彙總或複雜邊際尋訪的系統。若您的使用案例需要這些類型的即時圖形查詢工作流程，我們建議您使用 Amazon Neptune 來管理這些工作流程。

> **Note**
>
> 如果您需要查詢高度連線的資料集，或執行需要以毫秒延遲周遊多個節點 (也稱為多躍點查詢) 的查詢，則應考慮使用 Amazon Neptune。Amazon Neptune 是專為高效能圖形資料庫引擎所打造，該服務將存放的數十億筆關係最佳化，且查詢圖形時只會有數毫秒的延遲。

在 DynamoDB 中查詢與掃描資料的最佳實務

本節會介紹在 Amazon DynamoDB 中使用 `Query` 和 `Scan` 操作的一些最佳實務。

掃描的效能考量

一般而言，Scan 操作效率低於 DynamoDB 中的其他操作。Scan 操作會一律掃描整個資料表或次要索引。然後它會篩選數值來提供您想要的結果，基本上增加了從結果集刪除資料的額外步驟。

如果可行，您應該避免在大型資料表上或能移除許多結果的篩選條件上使用 Scan 操作。此外，隨著資料表或索引的增長，Scan 操作會變慢。所以此 Scan 操作會檢查每個項目的請求數值，並且可能會在單一操作中耗用大型資料表或索引的佈建輸送量。為了能擁有更快的回應時間，請設計資料表和索引，讓您的應用程式可以使用 Query 而非 Scan。(對於資料表，您也可以考慮使用 GetItem 和 BatchGetItem API。)

您也可以設計應用程式來使用 Scan 操作，最大限度地減少對請求率的影響。這可能包括比起 Scan 操作，使用全域次要索引更有效率的建模。有關此流程的更多資訊，請參閱以下影片。

建模低速存取模式

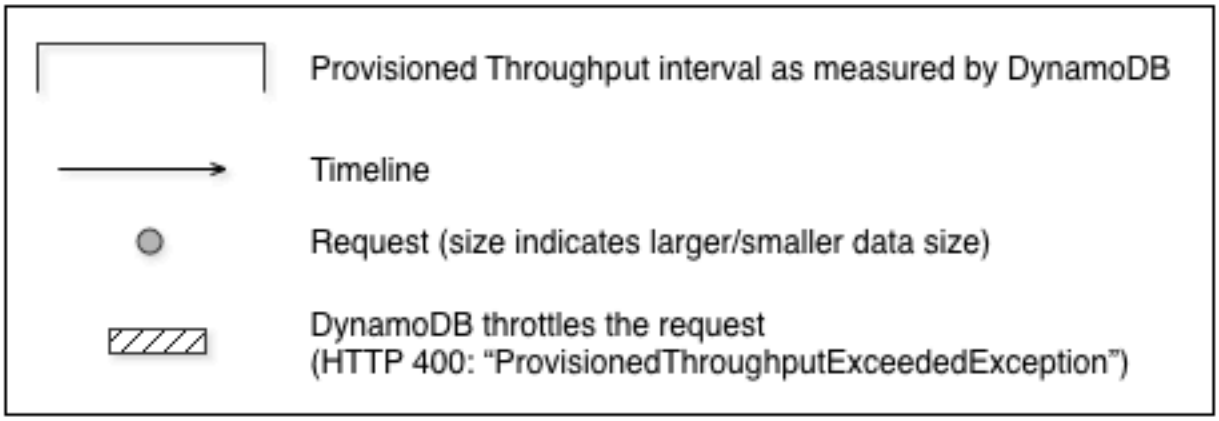
避免讀取活動突發尖峰

建立資料表時，您可以設定其讀取和寫入容量單位需求。針對讀取，容量單位會以每秒強烈一致的 4 KB 資料讀取請求數來表示。對於最終一致讀取，讀取容量單位為每秒兩個 4 KB 讀取請求。Scan 操作預設會執行最終一致讀取，而且最多可傳回 1 MB (一頁) 的資料。因此，單一 Scan 請求可能會消耗 (1 MB 頁面大小/4 KB 項目大小)/2 (最終一致讀取) = 128 個讀取操作。相對地，如果您請求高度一致性讀取，則 Scan 操作會耗用兩倍的佈建輸送量：256 個讀取操作。

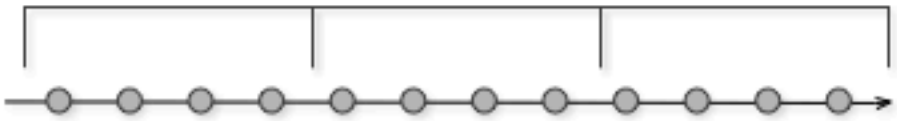
與資料表設定的讀取容量相比，這代表在使用時突發尖峰。此掃描使用容量單位可防止同一資料表的其他可能更重要的請求使用可用容量單位。因此對於那些請求，您可能會得到一個 ProvisionedThroughputExceeded 例外狀況。

問題不僅僅是 Scan 使用的容量單位突然增加。掃描也可能會耗用相同分割區的所有容量單位，因為掃描會請求讀取該分割區上彼此相鄰的項目。這表示請求會落在相同的分割區上，造成其所有容量單位被消耗，並限流對該分割區的其他請求。如果讀取資料的請求分散在多個分割區，則操作不會限流特定的分割區。

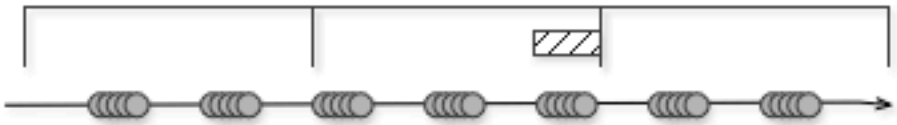
下圖說明 Query 和 Scan 操作的容量單位使用量激增的影響，以及其對同一資料表的其他請求的影響。



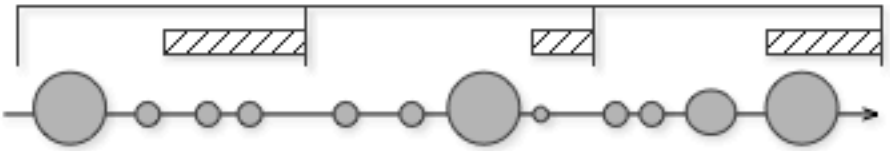
1. Good: Even distribution of requests and size



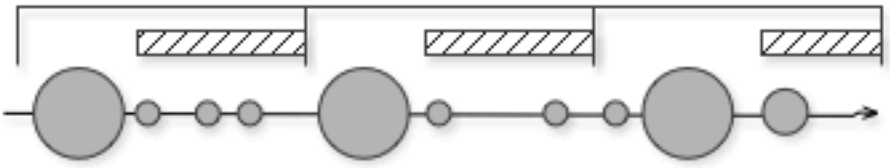
2. Not as Good: Frequent requests in bursts



3. Bad: A few random large requests



4. Bad: Large scan operations



如此處所示，使用量激增會以數種方式影響資料表的佈建輸送量：

1. 良好：請求和大小分配均勻

2. 不太好：頻繁的突發請求
3. 不良：幾個隨機的大型請求
4. 不良：大型掃瞄操作

除了使用大型 Scan 操作，您可以使用下列技巧，將該操作對資料表佈建輸送量的影響降至最低。

- 縮減頁面大小

  由於掃描操作會讀取整個頁面 (預設為 1 MB)，您可以設定較小的頁面大小來降低掃描操作的影響。Scan 操作提供了 Limit 參數，可讓您用來設定請求的頁面大小。每個擁有較小頁面大小的 Query 或 Scan 請求都會使用較少的讀取操作，並會在各請求之間建立「暫停」。例如，假設每個項目是 4 KB，而且您將頁面大小設定為 40 個項目。Query 請求將僅消耗 20 個最終一致讀取操作或 40 個高度一致性讀取操作。較大數量的較小 Query 或 Scan 操作會允許您的其他關鍵請求在不受限流的情況下成功進行。

- 隔離掃描操作

  DynamoDB 旨在輕鬆實現可擴展性。因此，應用程式可以為不同的目的建立資料表，甚至可能會跨多個資料表複製內容。您想要在未採用「任務關鍵型」流量的資料表上執行掃描。某些應用程式處理此負載的方式為每小時在兩個資料表 (一個用於關鍵流量，另一個則用於簿記) 之間輪換流量。其他應用程式可以透過在兩個資料表 (「任務關鍵型」和「陰影」資料表) 上執行每次寫入來處理負載。

將您的應用程式設定為重試接收表示您已超過佈建輸送量的回應代碼的任何請求。您也可以使用 UpdateTable 操作為資料表增加佈建的輸送量。如果工作負載有暫時的激增，導致輸送量偶爾超出佈建層級，則使用指數退避重試請求。如需實作指數退避的詳細資訊，請參閱 錯誤重試和指數退避。

利用平行掃描

許多應用程式可以從使用平行 Scan 操作而非循序掃描中獲益。例如，處理大型歷史資料表的應用程式，執行平行掃描的速度會比循序掃描的速度要快得多。背景「掃描程式」程序中的多個工作者執行緒，可以低優先順序掃描資料表，而不會影響生產流量。在每個這些例子中，平行 Scan 的使用方式使其不會剝奪佈建輸送量資源的其他應用程式。

雖然平行掃描可能很有幫助，但它們可能會對佈建輸送量造成大量需求。使用平行掃描時，您的應用程式有多個工作者都在同時執行 Scan 操作。這會快速消耗資料表的佈建讀取容量。在這種情況下，其他需要存取資料表的應用程式可能會受到限流。

如果符合下列條件，則平行掃描會是正確的選擇：

- 資料表大小為 20 GB 或更大。
- 該資料表的佈建讀取輸送量尚未完全使用。
- 循序 Scan 操作太慢。

選擇 TotalSegments

TotalSegments 的最佳設定取決於您的特定資料、資料表的佈建輸送量設定以及您的效能需求。您可能需要進行實驗，才能正確設定。我們建議您從簡單的比率開始，例如每 2 GB 資料一個區段。例如，對於 30 GB 的資料表，您可以設定 TotalSegments 為 15 個區段 (30 GB/2 GB)。您的應用程式將使用 15 個工作者，每個工作者掃描不同的區段。

您也可以根據用戶端資源為 TotalSegments 選擇一個數值。您可以將 TotalSegments 設定為 1 到 1000000 之間的任何數字，而 DynamoDB 則可讓您掃描該數目的區段。例如，如果用戶端限制可以同時運行的執行緒數量，您就可以逐漸增加 TotalSegments，直到應用程式得到最佳 Scan 效能。

監控平行掃描以最佳化佈建輸送量的使用，同時確保其他應用程式不會耗盡資源。如果您沒有消耗所有佈建的輸送量，但仍然在 Scan 請求中經歷限流，則增加 TotalSegments 的數值。如果 Scan 請求所耗用的佈建輸送量超過您想使用的量，則減少 TotalSegments 的數值。

DynamoDB 資料表設計的最佳實務

Amazon DynamoDB 中的一般設計原則會建議您盡量減少使用的資料表數量。大多數情況下，我們建議您考慮使用單一資料表。但是，如果單一或少量資料表不可行，則可能可以使用以下指引。

- 每個帳戶限制資料表不得增加超過每個帳戶 10,000 個。如果您的應用程式需要更多資料表，請在多個帳戶間分散資料表。如需詳細資訊，請參閱 Amazon DynamoDB 中的服務、帳戶和資料表配額。
- 對於可能影響資料表管理的並行控制平面操作，請考慮控制平面限制。
- 與 AWS 解決方案架構師合作，驗證多租戶設計的設計模式。

使用 DynamoDB 全域資料表

全域資料表以 Amazon DynamoDB 的全球佈局為基礎，提供全受管、多區域、多作用中的資料庫，能為大規模的全球應用程式提供快速且本機化的讀寫效能。全域資料表會自動複寫您選擇的 DynamoDB 資料表 AWS 區域。因為全域資料表使用現有的 DynamoDB API，所以不需要變更應用程式。使用全域資料表沒有前期成本或承諾，您僅需為使用的資源付費。

本指南說明如何有效使用 DynamoDB 全域資料表。本指南說明全域資料表的關鍵資訊、主要使用案例、兩種一致性模式、三種可考慮的寫入模型分類、四項可實作的請求路由選項、區域撤離 (不論線上或離線) 方法、輸送容量規劃考量，以及部署全域資料表時的核對清單。

本指南適用於 AWS 更大的多區域部署內容，如AWS 多區域基礎白皮書和含影片的資料彈性設計模式 AWS所述。

主題

- 有關 DynamoDB 全域資料表設計的關鍵事實
- 關於 MREC 的重點說明
- 關於 MRSC 的重點說明
- MREC DynamoDB 全域資料表使用案例
- 使用 DynamoDB 全域資料表的寫入模式
- DynamoDB 的路由策略
- 疏散程序
- DynamoDB 全域資料表的輸送容量規劃
- DynamoDB 全域資料表部署前檢查清單
- 結論和資源

有關 DynamoDB 全域資料表設計的關鍵事實

- 全域資料表共有兩個版本：目前版本全域資料表版本 2019.11.21 (目前版本) (又稱「V2」)，以及 全域資料表 2017.11.29 版 (舊版) (又稱「V1」)。本指南僅著重於目前版本。
- DynamoDB (不含全域資料表) 為區域型服務，具備高可用性，並在設計上對基礎架構故障 (包含整個可用區域 AZ 的中斷) 具備內建彈性。單區域 DynamoDB 資料表設計為可達 99.99% 可用性。如需詳細資訊，請參閱 DynamoDB 服務水準協議 (SLA)。
- DynamoDB 全域資料表會在兩個或多個區域間自動複寫資料。多區域 DynamoDB 資料表設計為可達 99.999% 可用性。透過適當規劃，全域資料表可協助建立具備區域故障復原能力的彈性架構。
- DynamoDB 並未提供全域端點。所有請求皆傳送至區域端點，以存取該區域的本機全域資料表執行個體。
- 不應跨區域對 DynamoDB 發出呼叫。最佳實務為：每個部署於單一區域的應用程式應僅存取該區域的本機 DynamoDB 端點。若在某區域偵測到問題 (無論位於 DynamoDB 層或相關堆疊)，應將終端使用者流量重新導向至位於其他區域的應用端點。全域資料表可確保各區域的應用程式均可存取相同資料。

一致性模式

建立全域資料表時，可設定其一致性模式。全域資料表支援兩種一致性模式：多區域最終一致性 (MREC) 與多區域高度一致性 (MRSC)，後者於 2025 年 6 月推出。

若您在建立全域資料表時未指定一致性模式，系統將預設使用 MREC。全域資料表不得包含使用不同一致性模式設定的副本。全域資料表建立後，其一致性模式無法再變更。

關於 MREC 的重點說明

- 採用 MRSC 的全域資料表同樣使用主動-主動式複寫模型。從 DynamoDB 的角度來看，每個區域中的資料表具有相同地位，可以接受讀取和寫入請求。本機複本資料表在接收到寫入請求後，會於背景作業中將寫入操作複製至其他參與區域。
- 單獨複製項目。同一交易內更新的項目可能不會同時被複寫。
- 來源區域內的每個資料表分割區，會與其他分割區並行執行寫入複寫。遠端區域中的寫入操作順序，可能與來源區域的寫入順序不一致。如需有關資料表分割區的詳細資訊，請參閱部落格文章擴展 DynamoDB：分割區、快捷鍵和熱分割會如何影響效能。
- 新寫入的項目通常會在一秒內傳播到所有複本列表。靠近區域的傳播速度往往更快。
- Amazon CloudWatch 為每個區域對提供一個 `ReplicationLatency` 指標。該指標是根據項目的到達時間與初始寫入時間的差異計算平均值而得。計時會儲存在來源區域的 CloudWatch 當中。檢視平均值與最大值有助於評估複寫延遲的平均與最差情境。此延遲沒有 SLA。
- 若相同項目在兩個不同區域幾乎同時 (於此 `ReplicationLatency` 視窗內) 更新，而第二次寫入發生於第一次寫入尚未複寫之前，則可能發生寫入衝突。採用 MREC 的全域資料表會根據寫入操作的時間戳記，以「最後寫入者獲勝」機制解決此類衝突。第一個操作將被第二個操作覆寫。這些衝突不會記錄於 CloudWatch 或 AWS CloudTrail。
- 每個項目都有一個作為私有系統屬性保留的最後寫入時間戳記。「最後寫入者獲勝」機制透過條件式寫入實現，要求新項目的時間戳記必須大於現有項目的時間戳記。
- 全域資料表會將所有項目複寫至所有參與的區域。若需不同複寫範圍，可建立多個全域資料表，並為各資料表指定不同參與區域。
- 即使複本區域離線或 `ReplicationLatency` 延遲增加，本機區域仍可接受寫入操作。本機資料表會繼續嘗試將項目複製到遠端資料表，直到每個項目成功為止。
- 萬一區域完全離線，當稍後重新上線時，所有待處理的輸出與輸入複寫都會自動重新嘗試。不需要特殊動作即可使資料表恢復同步。最後寫入者獲勝機制可確保資料最終達成一致狀態。
- 您可隨時將新區域新增至 DynamoDB MREC 資料表。DynamoDB 會自動處理初始同步與持續的複寫作業。您亦可移除某個區域 (包含原始區域)，系統將刪除該區域內的本機資料表。

關於 MRSC 的重點說明

- 採用 MRSC 的全域資料表同樣使用主動-主動式複寫模型。從 DynamoDB 的角度來看，每個區域中的資料表具有相同地位，可以接受讀取和寫入請求。在寫入操作傳回成功回應之前，MRSC 全域資料表複本中的項目變更會至少同步複寫到一個其他區域。
- 任何 MRSC 複本上的高度一致性讀取操作，一律會傳回項目的最新版本。條件式寫入操作一律會依據項目的最新版本來評估條件運算式。更新操作一律會以項目的最新版本為基準執行。
- MRSC 複本上的最終一致讀取操作，可能未包含最近在其他區域或同一區域中發生的變更。
- 當寫入操作嘗試修改另一個區域中已被修改的項目時，系統會傳回 `ReplicatedWriteConflictException` 例外狀況並導致操作失敗。若寫入操作因 `ReplicatedWriteConflictException` 例外狀況而失敗，可重試；當項目不再於其他區域被修改時，操作將成功。
- 使用 MRSC 時，寫入操作與高度一致性讀取操作的延遲較高。這些操作需要跨區域通訊。此類通訊會根據所存取區域與參與全域資料表中最近區域之間的往返延遲而增加額外延遲。如需詳細資訊，請參閱 AWS re：Invent 2024 簡報，多區域與 DynamoDB 全域資料表的高度一致性。最終一致讀取操作不會產生額外延遲。有一款開放原始碼的測試工具，可讓您針對各區域實驗性計算延遲。
- 單獨複製項目。採用 MRSC 的全域資料表不支援交易 API。
- MRSC 全域資料表必須剛好部署在三個區域中。您可將 MRSC 全域資料表設定為三個副本，或兩個副本搭配一個見證。見證是 MRSC 全域資料表中的一個元件，包含最近寫入各副本的資料。見證可作為完整複本的選用替代方案，並支援 MRSC 的高可用架構。見證不支援任何讀取或寫入操作。見證不會產生儲存或寫入的成本。見證部署於與兩個副本不同的區域中。
- 若要建立 MRSC 全域資料表，可新增一個副本與一個見證，或將兩個副本新增至不含資料的現有 DynamoDB 資料表。無法在現有 MRSC 全域資料表中新增額外副本。無法從 MRSC 全域資料表刪除單一複本或見證。可從 MRSC 全域資料表刪除兩個副本，或刪除一個副本與一個見證。第二種情境會將剩餘複本轉換為單一區域的 DynamoDB 資料表。
- 您可從 DescribeTable API 的輸出結果判斷 MRSC 全域資料表是否已設定見證，以及該見證所在的區域。見證由 DynamoDB 擁有和管理，不會出現在設定見證的區域中的 AWS 帳戶 中。
- MRSC 全域資料表可在下列區域集內使用：
  - 美國區域集：美國東部 (維吉尼亞北部)、美國東部 (俄亥俄)、美國西部 (奧勒岡)
  - 歐洲區域集：歐洲 (愛爾蘭)、歐洲 (倫敦)、歐洲 (巴黎)、歐洲 (法蘭克福)
  - AP 區域集：亞太區域 （東京）、亞太區域 （首爾） 和亞太區域 （大阪）
- MRSC 全域資料表無法跨越不同的區域集。例如，MRSC 全域資料表不能同時包含來自美國區域集與歐洲區域集的副本。
- MRSC 全域資料表不支援存活時間 (TTL)。

- MRSC 全域資料表不支援本機次要索引 (LSI)。
- CloudWatch Contributor Insights 資訊僅會針對執行操作的區域回報。
- 只要有第二個區域託管複本或見證以建立仲裁，本機區域便會接受所有讀取與寫入操作。若第二個區域無法使用，本機區域僅能提供最終一致讀取。
- 萬一某個區域完全離線，稍後重新上線時，將自動追上進度。在追上進度之前，寫入操作和強式一致讀取操作只會傳回錯誤，而對其他區域的請求會繼續正常運作。對趕上區域的最終一致讀取操作將傳回到目前為止傳播到該區域的資料，並在領導節點和本機複本之間具有一般的本機一致性行為。不需要特殊動作即可使資料表恢復同步。

MREC DynamoDB 全域資料表使用案例

MREC 全域資料表提供以下優點：

- 低延遲讀取操作。將資料複本部署於更接近終端使用者的位置，以降低讀取作業期間的網路延遲。資料會保持與 `ReplicationLatency` 值一致的新鮮度。
- 低延遲寫入操作。您可以寫入附近的區域，以減少網路延遲和寫入所花費的時間。必須小心路由寫入流量，以確保沒有衝突。路由技術在 DynamoDB 的路由策略 中有更詳細的討論。
- 無縫區域移轉。您可新增新區域並刪除舊區域，將部署從一個區域移轉至另一個區域，而不會造成資料層停機。

MREC 與 MRSC 全域資料表皆提供以下優點：

- 提高彈性和災難復原。若某區域效能降低或完全中斷，您可執行疏散程序。所謂疏散，指將部分或全部導向該區域的請求重新導向至其他區域。使用全域資料表可將 DynamoDB SLA 的月度正常運作時間百分比從 99.99% 提升至 99.999%。使用 MREC 可支援以秒為單位的復原點目標 (RPO) 與復原時間點目標 (RTO)。使用 MRSC 可支援 RPO 為零。

  例如，Fidelity 投資公司在 re:Invent 2022 中分享了其如何使用 DynamoDB 全域資料表實作訂單管理系統。他們的目標是在無法透過內部部署系統達成的規模下，實現可靠的低延遲處理，同時維持對可用區域及區域性故障的高彈性。

如果您的目標是提高系統韌性與災難復原能力，MRSC 資料表雖具有較高的寫入延遲與高度一致性讀取延遲，但可支援 RPO 為 0。MREC 全域資料表的 RPO 取決於副本之間的複寫延遲，通常為數秒，依各副本所在區域而異。

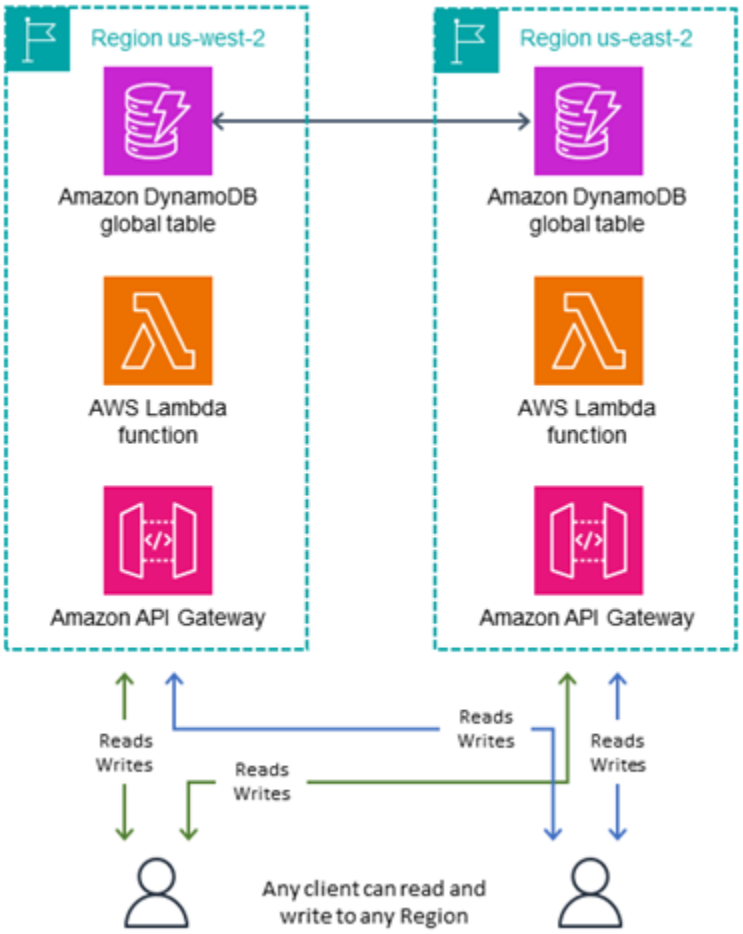
使用 DynamoDB 全域資料表的寫入模式

全域表在資料表永遠處於主動-主動資料表層級。不過，特別是對於 MREC 資料表，您可以透過控制寫入請求的路由方式，將其視為主動-被動配置。例如，您可以選擇將寫入請求路由至單一區域，以避免 MREC 資料表中可能發生的寫入衝突。

接下來三個章節將說明三種主要的受管寫入模式。您應該考慮哪種寫入模式適合您的使用案例。此選項會影響您的路由請求、疏散區域以及處理災難復原的方式。後續章節中的建議取決於應用程式所採用的寫入模式。

寫入任何區域模式 (無優先層級)

下圖所示的寫入任何區域模式採完全主動-主動架構，對寫入位置不施加任何限制。任何區域隨時可以接受寫入。這是最簡單的模式，但僅適用於特定類型的應用程式。此模式適用於所有 MRSC 資料表。當所有寫入端皆具冪等性時，此模式亦適用於 MREC 資料表，因為可安全地重複執行，確保跨區域的並行或重複寫入操作不會衝突。例如，使用者更新其聯絡資料。這種模式也適用於冪等性的特殊情況 (一個僅能新增的資料集)，其中所有寫入都是確定性主索引鍵下的唯一插入。最後，此模式也適用於在可容忍寫入衝突風險的 MREC 資料表。



寫入任何區域模式是最直覺式的實作架構。因為任何區域都可以隨時成為寫入目標，路由會更加容易。容錯移轉較為容易，因為 MRSC 資料表中的項目始終保持同步，且任何最近的寫入都可重播至任意次要區域，次數不限。盡可能之下，您應該以此為寫入模式進行設計。

例如，多數影音串流服務會使用全域資料表來追蹤書籤、評論與觀看狀態等資訊。這些部署採用 MREC 資料表，因需在全球多地配置副本，以提供低延遲的讀寫操作。只要確保每個寫入操作皆具冪等性，這些部署即可使用寫入至任何區域模式。當每次更新 (例如設定最新時間碼、指派新的評論或設定新的觀看狀態) 皆直接更新使用者的新狀態，且項目的下一個正確值不依賴其目前值時，即符合此情況。若使用者的寫入請求偶然被路由至不同區域，則最後一次寫入將保留，其全域狀態將依最終指派結果一致化。在此模式下，讀取操作最終將達成一致性，延遲時間取決於最新的 `ReplicationLatency` 值。

在另一個範例中，一家金融服務公司使用全域資料表作為系統的一部分來維護每個客戶使用借記卡購買的動作記錄，以計算該客戶的現金回饋獎勵。他們希望為每位客戶保留一個 `RunningBalance` 項目。此寫入模式本質上非等冪，因為當交易連續進入時，系統會使用 `ADD` 表達式根據目前值修改餘額，其

新值取決於當前值。透過使用 MRSC 資料表，他們仍可寫入任何區域，因為每個 ADD 呼叫都會對項目的最新值進行操作。

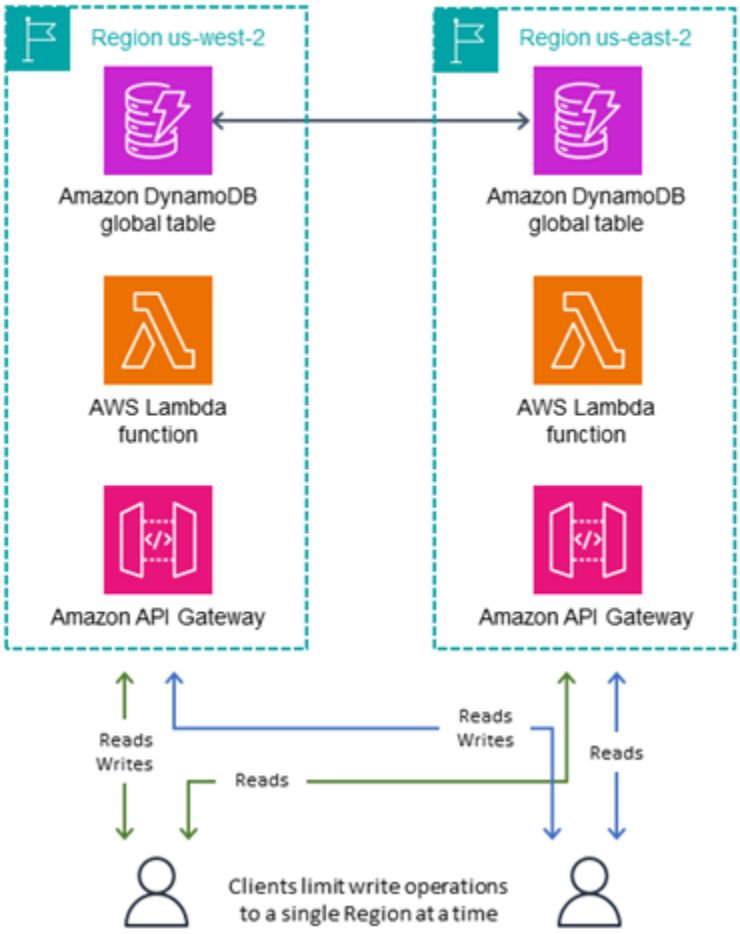
第三個範例是一家提供線上廣告投放服務的公司。該公司認為可接受低程度的資料遺失風險，以換取寫入任何區域模式帶來的設計簡化。當系統投放廣告時，僅有數毫秒可擷取足夠的中繼資料以判斷顯示哪則廣告，並記錄曝光，以避免短時間內重複播放相同廣告。他們透過全域資料表，為全球終端使用者提供低延遲的讀取與寫入操作。系統將每位使用者的所有廣告曝光記錄於單一項目中，並以持續增長的清單形式儲存。他們以單一項目取代附加至項目集合的方式，藉此在每次寫入操作中移除較舊的廣告曝光，而不需額外支付刪除操作費用。此寫入操作非冪等性；若同一使用者同時從多個區域接收廣告，可能發生某次廣告曝光寫入覆蓋另一筆記錄的情況。其風險在於使用者可能偶爾會看到重複的廣告。他們認為這樣的風險在可接受範圍內。

寫入單一區域 (單一主節點)

下圖所示的寫入單一區域模式採主動-被動架構，將所有資料表的寫入路由至單一主動區域。請注意，DynamoDB 沒有單一使用中區域的概念；DynamoDB 外部路由的應用程式會管理此功能。寫入單一區域模式適用於需避免寫入衝突的 MREC 資料表，因其可確保任一時間僅向單一區域執行寫入操作。當您需要使用條件式表達式但無法採用 MRSC，或需執行交易時，此寫入模式將特別有用。此類表達式僅在您確定操作的資料為最新時才可行，因此所有寫入請求必須傳送至擁有最新資料的單一區域。

使用 MRSC 資料表時，您可為方便起見選擇主要向單一區域進行寫入。例如，這有助於將 DynamoDB 以外的基礎設施擴建需求降至最低。寫入模式仍屬寫入任何區域，因為使用 MRSC 時，可於任何時間安全寫入任一區域，而無需進行衝突解決，也不會像 MREC 資料表那樣需選擇寫入單一區域。

最終一致讀取可以移至任何複本區域，以達到較低的延遲。高度一致性讀取必須由單一主要區域處理。



有時需因應區域性故障而切換作用中區域，這正是此模式的典型應用情境。部分使用者會定期切換作用中區域，例如採用「追日式」部署策略。這種方式會將作用中區域設置於使用活動最頻繁的地理位置附近 (通常為白天時區，因此得名)，以達到最低延遲的讀寫操作。此外，它還有每天呼叫區域切換程式碼的附加優點，可確保在災難復原前完成充分測試。

被動區域可能僅在 DynamoDB 週邊維持一組縮減版的基礎設施，僅在成為主動區域時才會擴建。本指南不涵蓋「指示燈」與「暖待命」架構設計。如需詳細資訊，請參閱 上的災難復原 (DR) 架構 AWS，第 III 部分：指示燈和暖待命。

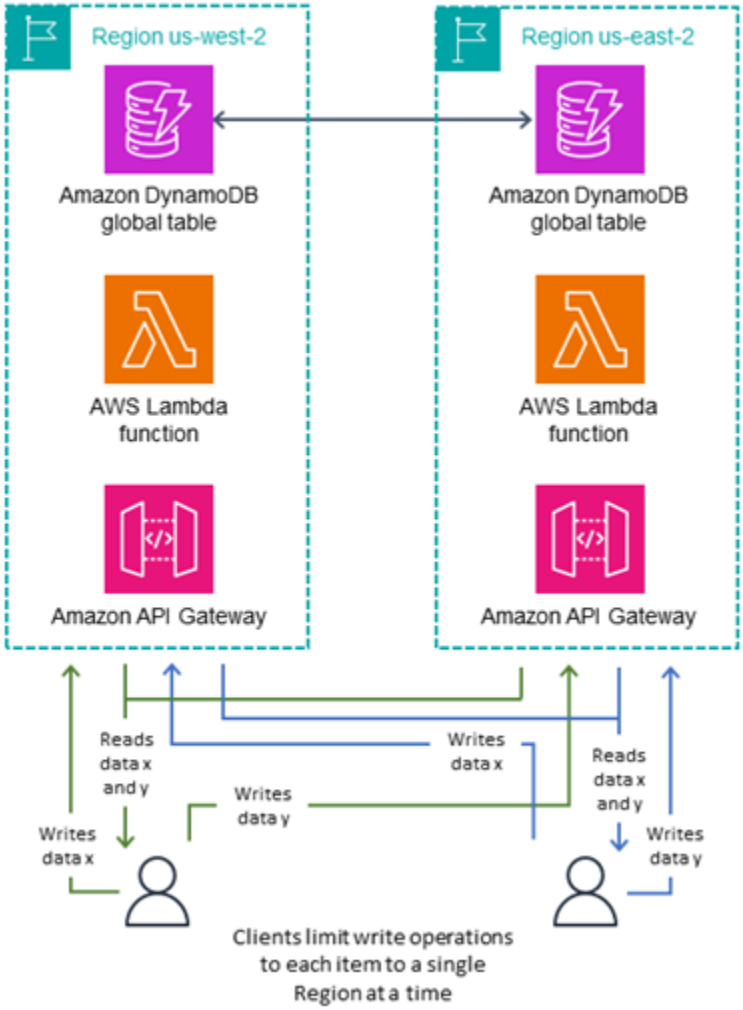
當使用全域資料表以實現低延遲的全球分散式讀取時，採用寫入單一區域模式效果極佳。例如，一家大型社群媒體公司需要在全球各區域提供相同的參考資料。該公司不常更新資料，但在更新時僅寫入單一區域，以避免潛在的寫入衝突。讀取操作始終可從任何區域執行。

另一個例子是前文提及的金融服務公司，該公司實作了每日現金回饋計算。該公司使用寫入任何區域模式計算餘額，而使用寫入單一區域模式追蹤現金回饋付款。這項作業需要交易功能，而 MRSC 資料表不支援此功能，因此使用獨立的 MREC 資料表並採用寫入單一區域模式會更合適。

混合主要層級 (寫入您的區域)

寫入您的區域模式 (如下圖所示) 適用於 MREC 資料表。系統會將不同資料子集分配至各自的主區域，並僅允許透過該主區域執行寫入操作。此模式屬主動-被動架構，但會依據項目決定作用中區域。每個區域皆為其專屬的非重疊資料集之主要區域，且必須限制寫入以確保正確的區域性。

此模式類似於寫入單一區域模式，但可提供更低延遲的寫入操作，因為每位使用者的資料可部署於距離其更近的網路環境中。此模式也能在各區域間更均勻地分配周邊基礎設施，並在容錯移轉情境中減少額外建置工作，因為所有區域均已有部分基礎設施處於運行狀態。



您可以透過多種方式判定項目的主區域：

- 內部：資料的特定屬性 (例如特殊屬性或內嵌於分割區鍵中的值) 可明確指出其主區域。此技術於使用區域固定功能為 Amazon DynamoDB 全域資料表項目設定主區域一文中說明。
- 協商：每個資料集的主區域會透過外部機制協商決定，例如由獨立的全域服務維護指派關係。指派可能是一個有限的持續時間，之後需要重新協商。

- 資料表導向：並非建立單一可複寫的全域資料表，而是根據複寫區域的數量建立相同數量的全域資料表。每個資料表的名稱都表示其主區域。標準操作中，所有資料都會寫入主區域，而其他區域則保留唯讀副本。在容錯移轉期間，另一個區域會暫時接管該資料表的寫入職能。

例如，假設您任職於一家遊戲公司。您需要為全球所有遊戲玩家提供低延遲的讀寫操作。您會將每位玩家指派至距離他最近的區域。該區域負責該玩家的所有讀寫操作，確保具備強一致的寫入後讀一致性。不過，若玩家外出旅行或其主區域發生中斷，替代區域中仍可使用其完整資料副本，並可重新指派新的主區域。

再舉一例，假設您任職於一家視訊會議公司。每場會議通話的中繼資料會被指派至特定區域。通話者可使用距離最近的區域，以獲得最低的通訊延遲。若發生區域中斷，透過全域資料表可迅速復原，系統能將通話處理移轉至已存在資料複本的其他區域。

摘要

- 寫入任何區域模式適用於 MRSC 資料表，以及對 MREC 資料表的等冪呼叫。
- 寫入單一區域模式適用於對 MREC 資料表的非等冪呼叫。
- 寫入您的區域模式適用於對 MREC 資料表的非等冪呼叫，並適用於需確保用戶端能寫入距離最近區域的情境。

DynamoDB 的路由策略

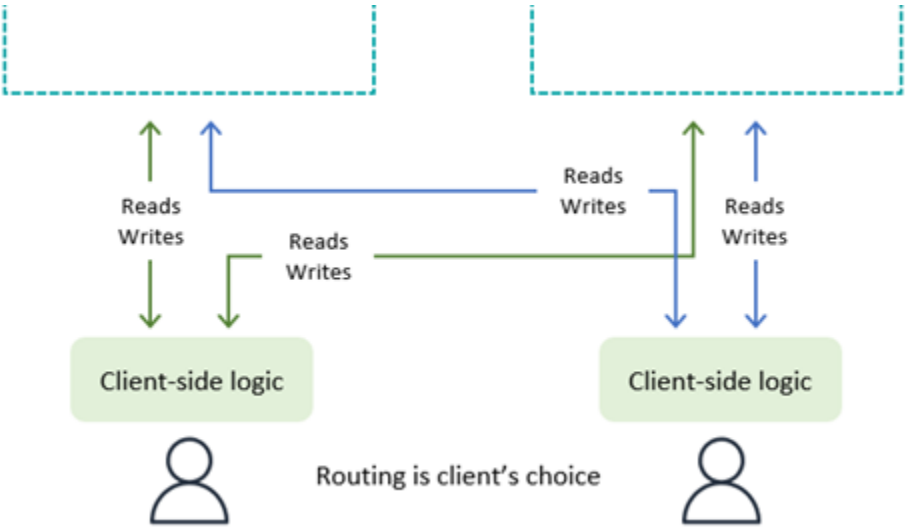
也許，全域資料表部署中最複雜的部分是管理請求路由。請求必須首先從終端使用者以某種方式選擇和路由到區域。請求在該區域中遇到一些服務堆疊，包括可能由 AWS Lambda函數、容器或 Amazon Elastic Compute Cloud (Amazon EC2) 節點支援的負載平衡器組成的運算層，以及其他可能包括另一個資料庫的服務。該運算層與 DynamoDB 通訊應該使用該區域的本機端點來執行此操作。全域資料表中的資料會複寫到所有其他參與的區域，而且每個區域在其 DynamoDB 資料表周圍都有類似的服務堆疊。

全域資料表會為不同區域中的每個堆疊提供相同資料的本機複本。如果本機 DynamoDB 資料表發生問題，您可以考慮為單一區域中的單一堆疊進行設計，並預期會對次要區域的 DynamoDB 端點進行遠端呼叫。這不是最佳實務。若某個區域出現由 DynamoDB 引起的問題 (或更可能由堆疊中其他元件或依賴 DynamoDB 的服務導致)，建議將終端使用者請求路由至其他區域處理，並使用該區域的運算層與其本機 DynamoDB 端點通訊。此方法可完全避開有問題的區域。為確保系統具備高彈性，需在多個區域間進行運算層與資料層的複寫。

有許多替代技術可將終端使用者請求路由到區域進行處理。最佳選擇取決於您的寫入模式和容錯移轉的考量。本節討論四個選項：用戶端驅動、運算層、Route 53 和 Global Accelerator。

用戶端驅動的請求路由

透過下圖所示的用戶端驅動請求路由，終端用戶端 (應用程式、含 JavaScript 的網頁或其他用戶端) 會追蹤有效的應用程式端點 (例如 Amazon API Gateway 端點，而非固定的 DynamoDB 端點)，並依自身內嵌邏輯選擇通訊區域。選擇依據可包含隨機抽樣、最低觀察延遲、最高觀察頻寬或本機執行的運作狀態檢查結果。



用戶端驅動請求路由的優勢在於，能依實際網際網路流量狀況自動調整，並在偵測到性能下降時切換至其他區域。用戶端必須了解所有可用端點，但啟動新的區域端點並不常見。

使用寫入任何區域模式時，用戶端可自行選擇偏好的端點。如果對某個區域的存取受損，用戶端可以路由到另一個端點。

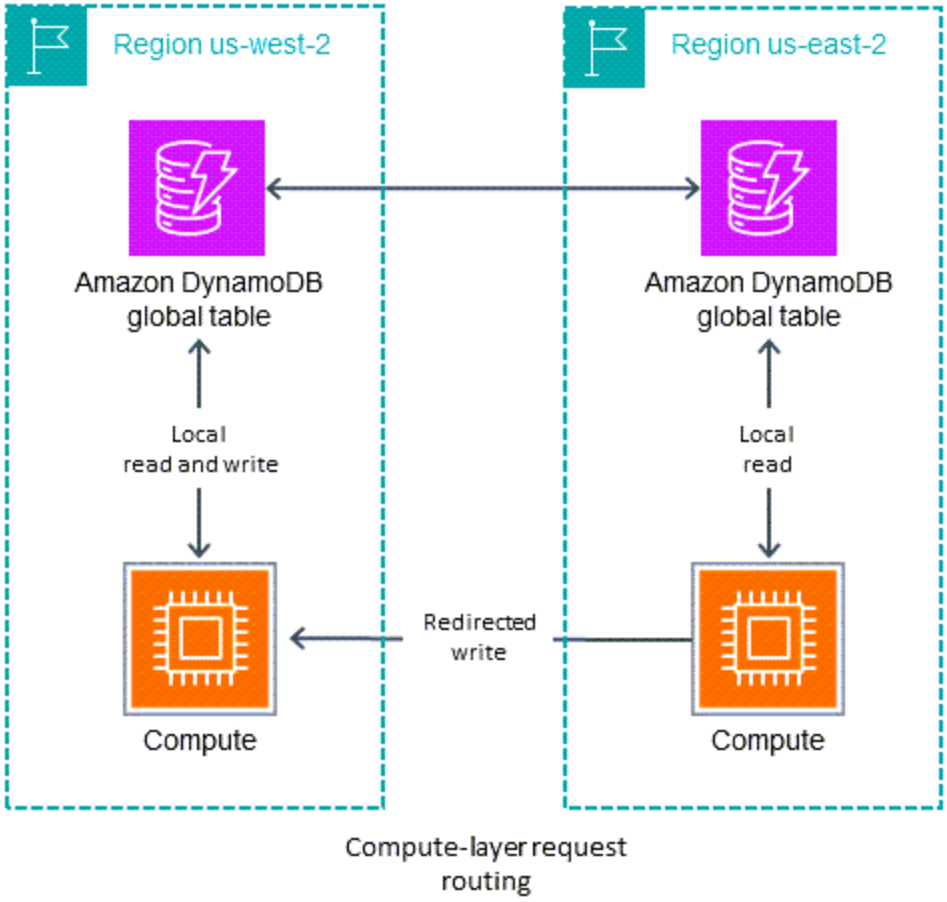
透過寫入單一區域模式，用戶端需要一種機制，將寫入路由到目前使用中的區域。這可以是簡單的實際測試，以找出目前可接受寫入的區域 (記錄任何寫入拒絕並回退至替代區域)；也可以是較複雜的方式，例如呼叫全域協調器以查詢目前的應用程式狀態 (可能建構於 Amazon 應用程式復原控制器 (ARC) 的路由控制之上，該控制提供一個以五區域仲裁機制維持全域狀態的系統，以支援此類需求)。用戶端可以決定讀取是否可以移轉到任何區域以取得最終一致性，還是必須路由到使用中的區域以取得高度一致性。如需進一步資訊，請參閱 Route 53 的運作方式。

使用寫入您的區域模式時，用戶端必須確定其處理之資料集的主區域。例如，如果用戶端對應一個使用者帳戶，且每個使用者帳戶都連結至一個區域，則用戶端可以從全域登入系統要求適當的端點。

例如，透過網路協助使用者管理其業務金融的金融服務公司，可以使用全域資料表和寫入您的區域模式。每個使用者都必須登入中央服務。該服務會傳回憑證，以及這些憑證將在該區域使用的端點。憑證僅在短時間內有效。之後，網頁會自動協商一個新的登入，提供一個可能將使用者活動重定導向新區域的機會。

運算層請求路由

如下圖所示，透過運算層請求路由，運算層執行的程式碼會決定是於本機處理請求，或將其轉交給在其他區域執行的相同程式實例。使用寫入單一區域模式時，運算層可能偵測自身非作用中區域，允許本機讀取操作，並將所有寫入操作轉送至其他區域。運算層程式碼必須了解資料拓撲與路由規則，並根據最新設定 (定義哪些區域為各資料的作用中區域) 可靠地強制執行。區域內的外層軟體堆疊無需了解微型服務如何路由讀寫請求。在穩健的設計中，接收區域會驗證其是否為寫入操作的目前主要項目。如果不是，則會產生錯誤，指出需要全域狀態需要修改。如果主要區域正在變更，則接收區域也可能會緩衝寫入操作一段時間。在所有情況下，區域中的運算堆疊只會寫入其本機 DynamoDB 端點，但運算堆疊可能會彼此通訊。

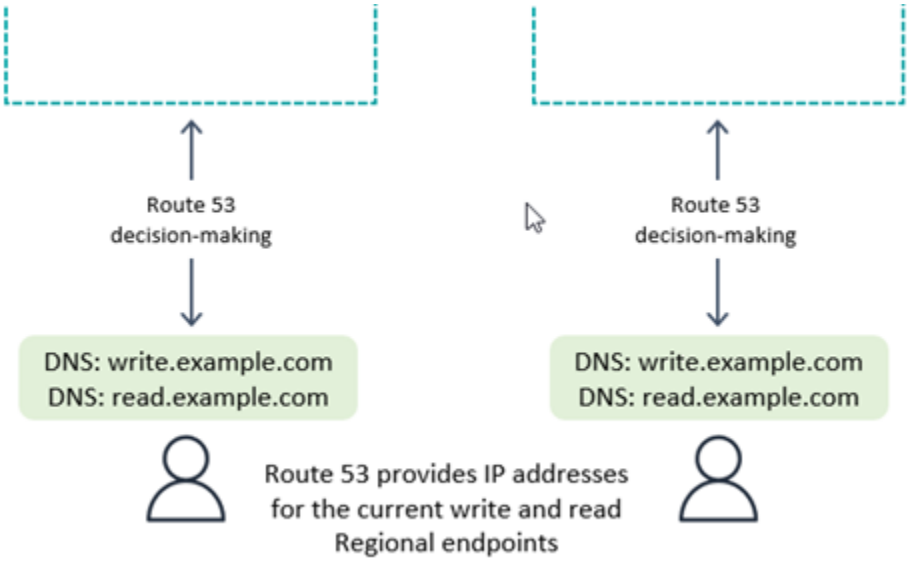


Vanguard 集團在 re:Invent 2022 中介紹了其使用的路由系統，包括 Global Orchestration and Status Tool (GOaST) 以及 Global Multi-Region Library (GMRlib)。他們採用「追日式」單一主要區域模型。GOaST 維護全域狀態，其運作方式類似於前節所述的 ARC 路由控制。該系統使用全域資料表來追蹤主要區域與下一次主要區域切換的排程時間。所有讀寫操作皆經由 GMRlib 處理，並與 GOaST 協調運作。GMRlib 允許在本機以低延遲執行讀取操作。對於寫入操作，GMRlib 會檢查本機區域是否為目前主要區域。如果是，則寫入操作會直接完成。若非主要區域，GMRlib 會將寫入任務轉送至主要區

域的 GMRlib。接收程式庫會確認它也會將自己視為主要區域，如果不是，則會引發錯誤，表示全域狀態的傳播延遲。此方法不直接寫入遠端 DynamoDB 端點，進而提供驗證優勢。

Route 53 請求路由

Amazon 應用程式復原控制器 (ARC) 是一種網域名稱服務 (DNS) 技術。使用 Route 53 時，用戶端會透過查詢已知的 DNS 網域名稱來請求其端點，Route 53 會傳回與其認為最合適的區域端點對應 IP 位址。這會在下列圖表中進行說明。Route 53 具備多種路由策略，用以決定最適合的區域。Route 53 亦可執行容錯移轉路由，將流量導離運作狀態檢查失敗的區域。



使用寫入任何區域模式，或者結合後端運算層請求路由使用，Route 53 可以取得完整存取權限，以根據任何複雜的內部規則 (例如，最近網絡鄰近的區域或最近的地理位置或任何其他選擇) 返回區域。

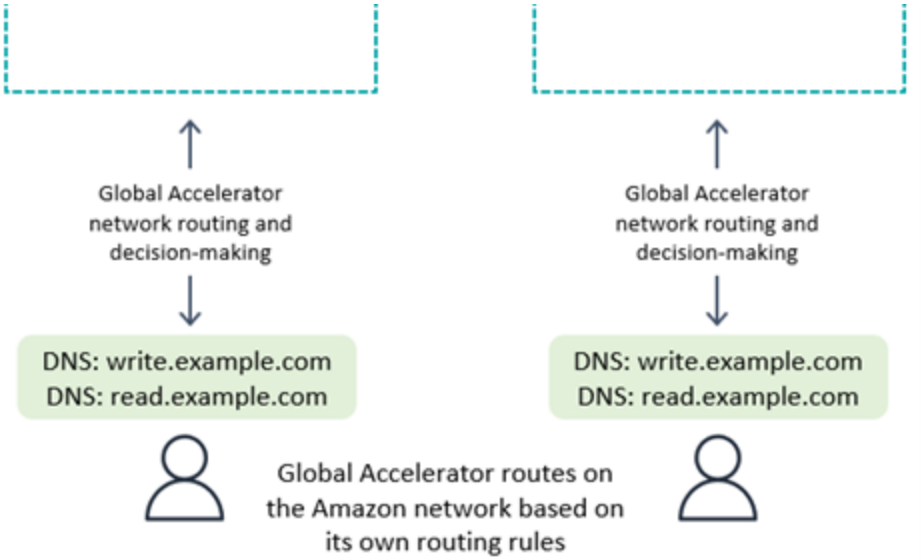
使用寫入單一區域模式，可設定 Route 53 返回目前作用中區域 (透過 Route 53 ARC)。若用戶端需連線至被動區域 (例如進行讀取操作)，可查詢不同的 DNS 名稱。

> **Note**
>
> 用戶端會快取 Route 53 回應中的 IP 位址一段時間，該時間由網域名稱上的存留時間 (TTL) 設定所指定的時間。較長的 TTL 會延長復原時間點目標 (RTO)，讓所有用戶端辨識新的端點。60 秒是容錯移轉使用的典型值。並非所有軟體都嚴格遵守 DNS TTL 到期設定，且可能存在多層 DNS 快取，例如作業系統層、虛擬機層及應用程式層。

寫入您的區域模式時，最好避免使用 Route 53，除非您還使用運算層請求路由。

Global Accelerator 請求路由

使用 AWS Global Accelerator，如下圖所示，用戶端會在 Route 53 中查詢知名網域名稱。不過，用戶端不會傳回對應至區域端點的 IP 地址，而是收到路由至最近 AWS 節點的任何廣播靜態 IP 地址。從該邊緣節點開始，所有流量都會路由到私有 AWS 網路，並透過路由至 Global Accelerator 內維護由路由規則所選擇區域中的某些端點 (例如負載平衡器或 API Gateway)。與使用 Route 53 規則為基礎的路由相比，Global Accelerator 請求路由的延遲較低，因為它可以減少公用網際網路上的流量。此外，由於 Global Accelerator 不依賴 DNS TTL 到期來更新路由規則，因此能更快速地調整流量路由。



透過寫入任何區域模式，或者如果與後端的計算層請求路由結合使用，Global Accelerator 可以無縫工作。用戶端會連線到最近的邊緣節點，而且不需要擔心由哪個區域接收請求。

使用寫入單一區域的 Global Accelerator 路由規則時，必須將請求傳送到目前使用中的區域。您可以使用運作狀態檢查，以人為方式報告任何區域的故障，而這些區域並未被您的全域系統視為使用中區域。與 DNS 一樣，如果請求可能來自任何區域，則可以使用替代 DNS 網域名稱來路由讀取請求。

使用寫入您的區域模式時，最好避免使用 Global Accelerator，除非您同時使用運算層請求路由。

疏散程序

區域疏散指的是將作業活動 (通常為讀取與寫入活動) 從該區域移轉出去的過程。

疏散即時區域

您可能會因多種原因決定疏散運作中的區域：例如作為日常業務流程的一部分 (如採用「追日式」或「寫入單一區域」模式)、因業務決策變更作用中區域、回應 DynamoDB 以外軟體堆疊的故障，或因遇到一般性問題 (例如區域延遲高於平常)。

通過寫入任何區域模式，疏散即時區域非常簡單。您可以使用任意路由系統將流量導向其他區域，並讓疏散區域中的寫入操作照常複製。

「寫入單一區域」與「寫入您區域」模式通常搭配 MREC 資料表使用。因此，在新作用中區域開始寫入作業前，您必須確保舊作用中區域的所有寫入已完整記錄、串流處理並全域傳播，以確保後續寫入作業基於最新資料版本。

假設區域 A 處於作用中狀態，而區域 B 是被動的 (無論是針對全域資料表或區域 A 中的項目來說)。執行疏散的典型機制是暫停寫入 A、等待充分的時間讓這些操作完全傳播到 B、更新架構堆疊以辨識 B 為使用中，然後繼續寫入操作至 B。沒有指標能夠保證區域 A 的資料已完全複製到區域 B。如果區域 A 狀況良好，請暫停寫入操作至區域 A，然後等待 10 倍 `ReplicationLatency` 最近指標的最大值，判斷複製是否完成。如果區域 A 狀況不佳並顯示延遲增加的其他區域，則您可以選擇較大的倍數作為等待時間。

疏散離線區域

有一種特殊情況需考量：若區域 A 在未預警的情況下完全離線，該怎麼處理？ 此情況極少發生，但仍應納入考量。

疏散離線 MRSC 資料表

若 MRSC 資料表出現此情況，您無需執行額外操作。MRSC 資料表支援復原點目標 (RPO) 為零。對離線區域中 MRSC 資料表的所有成功寫入作業，都會出現在其他區域資料表中，因此即使該區域完全離線，資料仍不會出現落差。業務可持續使用位於其他區域的資料副本。

疏散離線 MREC 資料表

若發生此情況，區域 A 中尚未傳播的寫入作業會暫存，並在該區域重新上線後傳播。寫操作不會遺失，但它們的傳播會無限期延遲。

在這種情況下如何進行應用程式的決策。對於業務連續性，寫入操作可能需要繼續進行新的主要區域 B。不過，如果區域 B 中的某個項目在區域 A 的寫入操作擱置傳播時收到更新，則會在最後一個寫入獲勝模型下抑制傳播。區域 B 中的任何更新都可能會抑制傳入的寫入請求。

在寫入任何區域模式下，區域 B 可持續執行讀寫作業，並假設區域 A 的項目最終會傳播至區域 B，同時需注意在區域 A 恢復上線前可能出現項目遺失。若可行 (例如針對具冪等性的寫入作業)，建議

重新播放近期寫入流量 (如使用上游事件來源)，以填補可能遺失的寫入作業，並透過「最後寫入優先」衝突解決機制抑制傳入寫入的最終傳播。

使用其他寫入模式，您必須考慮工作可以在稍微延時的環境下繼續進行的程度。區域 A 重新上線之前，將丟失一些持續時間較短的寫入操作 (如，ReplicationLatency 追蹤)。業務能夠繼續進行嗎？ 在某些使用案例中，能夠繼續進行，但在其他情況下，如果沒有額外的緩解機制，可能無法繼續進行。

例如，假設即使某個區域完全中斷，您仍必須不中斷地維持可用的信用餘額。您可以將餘額分為兩個獨立項目，一個配置在區域 A，另一個配置在區域 B，並讓每個項目各自擁有一半的可用餘額。這將使用寫入您的區域模式。每個區域中處理的交易更新會針對餘額的本機複本寫入。如果區域 A 完全離線，工作仍然可以在區域 B 中繼續進行交易處理，並且寫入操作將限制為區域 B 中所保留的餘額部分，像這樣拆分餘額會導致複雜性，但即使在不確定的待處理寫入操作下，可以提供一個安全業務復原的範例。

再舉一例，假設您正在擷取 Web 表單資料。您可以使用樂觀並行控制 (OCC)，為資料項目指派版本，並將最新版本嵌入 Web 表單中作為隱藏欄位。每次提交時，只有當資料庫中的版本仍與建置表單的版本相符時，寫入操作才會成功。如果版本不相符，則可以根據資料庫中目前版本重新整理 (或仔細合併) Web 表單，並且使用者可以再次進行操作。OCC 模型通常可以防止其他用戶端覆寫和產生新版本的資料，但也可以在用戶端遇到較舊版本資料的容錯移轉期間提供幫助。假設您是使用時間戳記作為版本。表單最初於 12:00 在區域 A 建置，但在容錯移轉後嘗試寫入區域 B，並發現資料庫中的最新版本時間為 11:59。在此情況中，用戶端可以等候 12:00 版本傳播到區域 B，然後在該版本之上寫入，或是在 11:59 建置並建立新的 12:01 版本 (寫入後，會在區域 A 復原之後抑制傳入版本)。

第三個例子是，某金融服務公司在 DynamoDB 資料庫中儲存客戶帳戶及其財務交易資料。若區域 A 完全中斷，他們希望確保與帳戶相關的所有寫入活動在區域 B 中完全可用；否則，將這些帳戶標示為「部分可用」並予以隔離，直到區域 A 恢復上線。他們沒有暫停所有業務，而是決定只對確定有未傳播交易的一小部分賬戶暫停業務。為了達成此目的，他們使用了第三個區域，我們將呼叫區域 C。在處理區域 A 中的任何寫入操作之前，他們在區域 C 中對那些暫緩操作 (例如，帳戶新的交易計數) 建立摘要彙總。此彙總足以讓區域 B 判斷其檢視是否完全為最新狀態。此動作會有效鎖定帳戶，從寫入區域 C 開始，直到區域 A 接受寫入操作且區域 B 收到為止。除非作為容錯移轉程序的一部分外，不會使用區域 C 中的資料，之後區域 B 可以和區域 C 交叉比對資料，以檢查其帳戶是否已過期。這些帳戶會維持「隔離」狀態，直到區域 A 的復原程序將部分資料同步至區域 B。若區域 C 發生故障，則可啟動新的區域 D 以替代使用。區域 C 中的資料非常短暫，幾分鐘後，區域 D 將擁有最新的使用中寫入操作記錄，以充分發揮作用。如果區域 B 當機，區域 A 可以繼續接受與區域 C 合作的寫入請求。這家公司願意接受更高的延遲寫入 (到兩個區域：C 和 A)，並且很高興擁有資料模型，可以摘要彙總帳戶狀態。

DynamoDB 全域資料表的輸送容量規劃

將流量從一個區域移轉到另一個區域需要仔細考慮與容量有關的 DynamoDB 資料表設定。

管理寫入容量時可考慮以下幾點：

- 全域資料表必須處於隨需模式，或啟用自動擴展的佈建。
- 如果使用自動擴展進行佈建，則會跨區域複寫寫入設定 (最小、最大和目標使用率)。雖然自動擴展設定會同步，但實際佈建的寫入容量可能在各區域間獨立變動。
- 造成佈建寫入容量差異的原因之一是 TTL 功能。當您在 DynamoDB 中啟用 TTL 時，您可以指定屬性名稱，其值表示項目的到期時間，以秒為單位的 Unix 紀元時間格式。之後，DynamoDB 可刪除項目，而不會產生寫入成本。有了全域資料表，您可以在任何區域設定 TTL，該設定便會自動複寫到另一個區域關聯的全域資料表。當某個項目符合通過 TTL 規則刪除的條件時，該工作可以在任何區域完成。執行刪除操作時不會耗用於源資料表上的寫入單位，但複本資料表會取得該刪除操作的複寫寫入操作，並將產生複製寫入單元成本。MRSC 資料表不支援 TTL。
- 如果您使用自動擴展，請確定佈建的寫入容量上限設定足以處理所有寫入操作以及所有潛在的 TTL 刪除操作。自動擴展會根據每個區域的寫入耗用進行調整。隨需資料表沒有最大佈建的寫入容量設定，但是資料表層級最大寫入輸送量限制會指定隨需資料表允許的最大持續寫入容量。預設限制為 40,000，但可以調整。我們建議您設定充足，以處理隨需資料表可能需要的所有寫入操作 (包括 TTL 寫入操作)。當您設定全域資料表時，所有參與區域的這個值必須相同。

管理讀取容量時可考慮以下幾點：

- 允許區域之間的讀取容量管理設定不同，因為需假設不同的區域可能具有獨立的讀取模式。當第一次將全域複本新增至資料表時，會傳播來源區域的容量。建立之後，您可以調整讀取容量設定，而這個設定不會傳輸到另一端。
- 使用 DynamoDB 自動擴展時，請確保佈建的最大讀取容量設定充足，以處理所有區域的所有讀取操作。在標準操作期間，讀取容量可能會分散到區域，但在容錯移轉期間，資料表應該能夠自動適應增加的讀取工作負載。隨需資料表沒有最大佈建的讀取容量設定，但是資料表層級最大讀取輸送量限制會指定隨需資料表允許的最大持續讀取容量。預設限制為 40,000，但可以調整。如果所有讀取操作都要路由到此單一區域，我們建議您將它設定充足，以處理資料表可能需要的所有讀取操作。
- 若某區域的資料表平時讀取流量較低，但在容錯移轉後可能需承受大量讀取流量，您可以預先暖機資料表容量，以因應較高的讀取需求。

Route 53 ARC 具有整備檢查功能，可用於確認 DynamoDB 各區域的資料表設定與帳戶配額是否一致，無論您是否使用 Route 53 進行請求路由。這些就緒性檢查還可以幫助調整帳戶層級的配額，以確保符合。

DynamoDB 全域資料表部署前檢查清單

部署全域資料表時，請使用下列檢查清單來制定決策和執行任務。

- 判斷您的使用案例更適合 MRSC 或 MREC 一致性模式。您是否需要高度一致性，即使必須承擔更高延遲與其他取捨？
- 決定有多少以及哪些區域應參與全域資料表。若您計劃使用 MRSC，請決定第三個區域要作為複本區域或見證區域。
- 決定應用程式的寫入模式。這與一致性模式並非相同概念。如需詳細資訊，請參閱使用 DynamoDB 全域資料表的寫入模式。
- 根據您的寫入模式計畫 DynamoDB 的路由策略 策略。
- 根據一致性模式、寫入模式與路由策略定義您的

區域疏散指的是將作業活動 (通常為讀取與寫入活動) 從該區域移轉出去的過程。

疏散即時區域

您可能會因多種原因決定疏散運作中的區域：例如作為日常業務流程的一部分 (如採用「追日式」或「寫入單一區域」模式)、因業務決策變更作用中區域、回應 DynamoDB 以外軟體堆疊的故障，或因遇到一般性問題 (例如區域延遲高於平常)。

通過寫入任何區域模式，疏散即時區域非常簡單。您可以使用任意路由系統將流量導向其他區域，並讓疏散區域中的寫入操作照常複製。

「寫入單一區域」與「寫入您區域」模式通常搭配 MREC 資料表使用。因此，在新作用中區域開始寫入作業前，您必須確保舊作用中區域的所有寫入已完整記錄、串流處理並全域傳播，以確保後續寫入作業基於最新資料版本。

假設區域 A 處於作用中狀態，而區域 B 是被動的 (無論是針對全域資料表或區域 A 中的項目來說)。執行疏散的典型機制是暫停寫入 A、等待充分的時間讓這些操作完全傳播到 B、更新架構堆疊以辨識 B 為使用中，然後繼續寫入操作至 B。沒有指標能夠保證區域 A 的資料已完全複製到區域 B。如果區域 A 狀況良好，請暫停寫入操作至區域 A，然後等待 10 倍 `ReplicationLatency` 最近指標的最大值，判斷複製是否完成。如果區域 A 狀況不佳並顯示延遲增加的其他區域，則您可以選擇較大的倍數作為等待時間。

疏散離線區域

有一種特殊情況需考量：若區域 A 在未預警的情況下完全離線，該怎麼處理？ 此情況極少發生，但仍應納入考量。

疏散離線 MRSC 資料表

若 MRSC 資料表出現此情況，您無需執行額外操作。MRSC 資料表支援復原點目標 (RPO) 為零。對離線區域中 MRSC 資料表的所有成功寫入作業，都會出現在其他區域資料表中，因此即使該區域完全離線，資料仍不會出現落差。業務可持續使用位於其他區域的資料副本。

疏散離線 MREC 資料表

若發生此情況，區域 A 中尚未傳播的寫入作業會暫存，並在該區域重新上線後傳播。寫操作不會遺失，但它們的傳播會無限期延遲。

在這種情況下如何進行應用程式的決策。對於業務連續性，寫入操作可能需要繼續進行新的主要區域 B。不過，如果區域 B 中的某個項目在區域 A 的寫入操作擱置傳播時收到更新，則會在最後一個寫入獲勝模型下抑制傳播。區域 B 中的任何更新都可能會抑制傳入的寫入請求。

在寫入任何區域模式下，區域 B 可持續執行讀寫作業，並假設區域 A 的項目最終會傳播至區域 B，同時需注意在區域 A 恢復上線前可能出現項目遺失。若可行 (例如針對具冪等性的寫入作業)，建議重新播放近期寫入流量 (如使用上游事件來源)，以填補可能遺失的寫入作業，並透過「最後寫入優先」衝突解決機制抑制傳入寫入的最終傳播。

使用其他寫入模式，您必須考慮工作可以在稍微延時的環境下繼續進行的程度。區域 A 重新上線之前，將丟失一些持續時間較短的寫入操作 (如，`ReplicationLatency` 追蹤)。業務能夠繼續進行嗎？ 在某些使用案例中，能夠繼續進行，但在其他情況下，如果沒有額外的緩解機制，可能無法繼續進行。

例如，假設即使某個區域完全中斷，您仍必須不中斷地維持可用的信用餘額。您可以將餘額分為兩個獨立項目，一個配置在區域 A，另一個配置在區域 B，並讓每個項目各自擁有一半的可用餘額。這將使用寫入您的區域模式。每個區域中處理的交易更新會針對餘額的本機複本寫入。如果區域 A 完全離線，工作仍然可以在區域 B 中繼續進行交易處理，並且寫入操作將限制為區域 B 中所保留的餘額部分，像這樣拆分餘額會導致複雜性，但即使在不確定的待處理寫入操作下，可以提供一個安全業務復原的範例。

再舉一例，假設您正在擷取 Web 表單資料。您可以使用樂觀並行控制 (OCC)，為資料項目指派版本，並將最新版本嵌入 Web 表單中作為隱藏欄位。每次提交時，只有當資料庫中的版本

仍與建置表單的版本相符時，寫入操作才會成功。如果版本不相符，則可以根據資料庫中目前版本重新整理 (或仔細合併) Web 表單，並且使用者可以再次進行操作。OCC 模型通常可以防止其他用戶端覆寫和產生新版本的資料，但也可以在用戶端遇到較舊版本資料的容錯移轉期間提供幫助。假設您是使用時間戳記作為版本。表單最初於 12:00 在區域 A 建置，但在容錯移轉後嘗試寫入區域 B，並發現資料庫中的最新版本時間為 11:59。在此情況中，用戶端可以等候 12:00 版本傳播到區域 B，然後在該版本之上寫入，或是在 11:59 建置並建立新的 12:01 版本 (寫入後，會在區域 A 復原之後抑制傳入版本)。

第三個例子是，某金融服務公司在 DynamoDB 資料庫中儲存客戶帳戶及其財務交易資料。若區域 A 完全中斷，他們希望確保與帳戶相關的所有寫入活動在區域 B 中完全可用；否則，將這些帳戶標示為「部分可用」並予以隔離，直到區域 A 恢復上線。他們沒有暫停所有業務，而是決定只對確定有未傳播交易的一小部分賬戶暫停業務。為了達成此目的，他們使用了第三個區域，我們將呼叫區域 C。在處理區域 A 中的任何寫入操作之前，他們在區域 C 中對那些暫緩操作 (例如，帳戶新的交易計數) 建立摘要彙總。此彙總足以讓區域 B 判斷其檢視是否完全為最新狀態。此動作會有效鎖定帳戶，從寫入區域 C 開始，直到區域 A 接受寫入操作且區域 B 收到為止。除非作為容錯移轉程序的一部分外，不會使用區域 C 中的資料，之後區域 B 可以和區域 C 交叉比對資料，以檢查其帳戶是否已過期。這些帳戶會維持「隔離」狀態，直到區域 A 的復原程序將部分資料同步至區域 B。若區域 C 發生故障，則可啟動新的區域 D 以替代使用。區域 C 中的資料非常短暫，幾分鐘後，區域 D 將擁有最新的使用中寫入操作記錄，以充分發揮作用。如果區域 B 當機，區域 A 可以繼續接受與區域 C 合作的寫入請求。這家公司願意接受更高的延遲寫入 (到兩個區域：C 和 A)，並且很高興擁有資料模型，可以摘要彙總帳戶狀態。the section called “疏散程序”。

- 擷取每個區域的運作狀態、延遲和錯誤的指標。如需 DynamoDB 指標清單，請參閱 AWS 部落格文章監控 Amazon DynamoDB 的操作意識，以取得要觀察的指標清單。您還應該使用 Synthetic Canaries (旨在檢測故障的人工請求，以煤礦中的金絲雀命名)，以及即時觀察客戶流量。並非所有問題都會出現在 DynamoDB 指標中。
- 若您使用 MREC，請針對 `ReplicationLatency` 的持續增加設定警示。增加可能表示意外設定錯誤，而全域資料表在不同區域中有不同的寫入設定，這會導致複寫請求失敗並增加延遲。這也可能表示存在區域中斷。一個很好的例子是如果最近的平均值超過 180,000 毫秒，則發出警示。您也應監控 `ReplicationLatency` 是否降為 0，此狀況表示複寫程序已停滯。
- 為每個全域資料表指派充足的最大讀取和寫入設定值。
- 事先確定疏散區域的原因。如果決定涉及人為判斷，請記錄所有考量因素。這項工作應提前仔細完成，而不是在壓力下進行。

- 為每個動作維護執行手冊，作為當疏散區域時必須採取的措施。通常，全域資料表涉及的工作很少，但移動堆疊的其餘部分可能很複雜。

> **Note**
>
> 在容錯移轉程序中，最佳實務是僅依賴資料平面操作，而非控制平面操作，因為在區域故障期間，部分控制平面操作可能會降級。

如需詳細資訊，請參閱 AWS 部落格文章使用 Amazon DynamoDB 全域資料表建置彈性應用程式：第 4 部分。

- 定期測試執行手冊的各個方面，包括區域疏散。未經測試的執行手冊是不可靠的執行手冊。
- 請考慮使用 AWS Resilience Hub 來評估整個應用程式 (包含全域資料表) 的韌性。它透過儀表板提供整體應用程式產品組合彈性狀態的全面檢視。
- 請考慮使用 ARC 的整備檢查來評估應用程式的目前組態，並追蹤與最佳實務的偏差。
- 當您為 Route 53 或 Global Accelerator 撰寫運作狀態檢查時，請設計一組涵蓋完整資料庫流程的呼叫。如果您限制檢查以確認 DynamoDB 端點已啟動，您將無法涵蓋許多失敗模式，例如 AWS Identity and Access Management (IAM) 組態錯誤、程式碼部署問題、DynamoDB 外部堆疊中的失敗、高於平均讀取或寫入延遲等。

部署全域資料表的常見問答集 (FAQ)

全域資料表的定價為何？

- 傳統的 DynamoDB 資料表寫入作業會依照佈建資料表的寫入容量單位 (WCU) 或隨需資料表的寫入請求單位 (WRU) 收費。如果您寫入一個 5 KB 的項目，則會產生 5 個單位的費用。全域資料表的寫入依複寫寫入容量單位 (rWCU，適用於佈建資料表) 或複寫寫入請求單位 (rWRU，適用於隨需資料表) 收費；rWCU 與 rWRU 的收費與 WCU、WRU 相同。
- 無論項目是直接寫入或透過複寫寫入，每個區域都會產生 rWCU 與 rWRU 費用。需支付跨區域資料傳輸費用。
- 寫入全域次要索引 (GSI) 會視為本機寫入作業，並使用常規寫入單位。
- 目前 rWCU 與 rWRU 均無可用的預留容量。若資料表中的 GSI 消耗大量寫入單位，購買 WCU 預留容量可能有助於節省成本。
- 當您將新區域加入全域資料表時，DynamoDB 會自動啟動新區域，並根據資料表的 GB 大小向您收費，就像是資料表還原一樣。此外，需支付跨區域資料傳輸費用。

全域資料表支援哪些區域？

Global Tables 2019.11.21 版（目前）支援 AWS 區域 MREC 資料表的所有 ，以及 MRSC 資料表的下列區域集：

- 美國區域集：美國東部 (維吉尼亞北部)、美國東部 (俄亥俄)、美國西部 (奧勒岡)
- 歐洲區域集：歐洲 (愛爾蘭)、歐洲 (倫敦)、歐洲 (巴黎)、歐洲 (法蘭克福)
- AP 區域集：亞太區域（東京）、亞太區域（首爾）和亞太區域（大阪）

全域資料表如何處理 GSI？

在全域資料表 2019.11.21 版 (目前)中，當您在某個區域建立 GSI 時，系統會在其他參與區域自動建立並回填。

如何停止全域資料表的複寫？

- 刪除複本資料表的方式，與刪除其他資料表相同。刪除全域資料表將停止複寫到該區域，並刪除保留在該區域中的資料表複本。不過，您無法在保留資料表複本為獨立實體的同時停止複寫，也無法暫停複寫。
- MRSC 資料表必須部署於三個區域中。若要刪除副本，您必須同時刪除所有副本與見證，使 MRSC 資料表轉為本機資料表。

DynamoDB 串流如何與全域資料表互動？

- 每個全域資料表都會根據其所有的寫入作業產生獨立串流，無論從何處開始皆然。您可以選擇在一個區域或所有區域中 (獨立) 使用此 DynamoDB 串流。如果您想要處理本機寫入，而不是複寫的寫入操作，可以將自己的區域屬性新增至每個項目。然後，您可以使用 Lambda 事件篩選條件，呼叫 Lambda 函式在本機區域中進行寫入。這有助於插入和更新操作，但不能用於刪除操作。
- 設定為多區域最終一致性 (MREC 資料表) 的全域資料表會透過從複本資料表上的 DynamoDB 串流讀取變更，並將其套用至其他所有複本資料表，以進行變更複寫。因此，MREC 全域資料表的所有副本預設均啟用 DynamoDB，且無法停用。MREC 複寫程序可在短時間內將多項變更整合為單一複寫寫入作業。因此，各複本的串流可能會包含略有差異的記錄。在 MREC 副本上，DynamoDB Streams 記錄一律依每個項目排序，但不同副本之間的項目順序可能會有所不同。
- 為多區域強式一致性 (MRSC 資料表) 設定的全域資料表不使用 DynamoDB Streams 進行複寫，因此 MRSC 副本預設不啟用此功能。您可以在 MRSC 複本上啟用 DynamoDB Streams。MRSC 副本上的 DynamoDB Streams 記錄在所有副本間完全相同，並一律依項目排序，但不同副本間的項目順序可能仍有差異。

全域資料表如何處理交易？

- 在 MRSC 資料表上執行交易作業會產生錯誤。
- MREC 資料表上的交易作業僅在最初執行寫入的區域內，提供原子性、一致性、隔離性與持久性 (ACID) 保證。全域資料表不支援跨區域交易。例如，若您擁有在美國東部 (俄亥俄) 與美國西部 (奧勒岡) 區域皆有副本的 MREC 全域資料表，並於美國東部 (俄亥俄) 區域執行 `TransactWriteItems` 操作，當變更複寫後，您可能會在美國西部 (奧勒岡) 區域觀察到部分完成的交易。當變更已在來源區域遞交後，這些變更才會複寫至其他區域。

全域資料表如何與 DynamoDB Accelerator (DAX) 快取互動？

全域資料表會透過直接更新 DynamoDB 來略過 DAX，因此 DAX 不會察覺其持有過時資料。當快取的 TTL 到期時，才會重新整理 DAX 快取。

是否會傳播資料表上的標籤？

否，標籤不會自動傳播。

我應該備份所有區域中的資料表還是只備份一個？

答案是取決於備份的目的。

- 如果您想要確保資料耐久性，則 DynamoDB 已適當提供該保護。該服務確保的就是耐久性。
- 如果您想要保留歷史記錄的快照 (例如，為了符合法規要求)，則在一個區域中進行備份應該就足夠了。您可以使用 將備份複製到其他區域 AWS Backup。
- 如果您想要復原錯誤刪除或修改的資料，請在一個區域中使用 DynamoDB 時間點復原 (PITR)。

如何使用 部署全域資料表 CloudFormation？

- CloudFormation 將 DynamoDB 資料表和一個全域資料表顯示為兩個獨立的資源：`AWS::DynamoDB::Table` 和 `AWS::DynamoDB::GlobalTable`。一種做法是使用 `GlobalTable` 建構方式，先建立為獨立資料表，若有需要再新增其他區域。
- 在 CloudFormation 中，無論複本數目多少，每個全域資料表都在單一區域中由單一堆疊控制。當您部署範本時，CloudFormation 會將所有複本建立和更新為單一堆疊操作的一部分。您不應該在多個區域中部署相同的 AWS::DynamoDB::GlobalTable 資源。這會導致錯誤且不支援。如果您在多個區域中部署應用程式範本，則可以使用條件在單一區域中建立 `AWS::DynamoDB::GlobalTable` 資源。您也可以選擇在與應用程式分開的堆疊中定義 `AWS::DynamoDB::GlobalTable` 資源，並確保其部署到單一區域。

- 若您希望在維持 CloudFormation 管理的情況下，將一般資料表轉換為全域資料表，請將刪除原則設定為 `Retain`，自堆疊移除該資料表後，於主控台將其轉換為全域資料表，再將該全域資料表作為新資源匯入堆疊。如需更多資訊，請參閱 AWS GitHub 儲存庫。
- 目前不支援跨帳戶複寫。

結論和資源

DynamoDB 全域資料表的控制項雖然不多，但仍需審慎規劃。您必須決定寫入模式、路由模型和疏散程序。您必須在每個區域檢測您的應用程式，並準備好調整路由或執行疏散以維護全域狀況。最終成果是可獲得具全球分散架構的資料集，提供低延遲的讀寫操作，並達到 99.999% 的可用性設計目標。

如需 DynamoDB 全域資料表的更多資訊，請參閱以下資源：

- DynamoDB 文件
- Amazon 應用程式復原控制器說明
- ARC 中的準備度檢查 (AWS 文件）
- Route 53 路由政策
- AWS Global Accelerator
- DynamoDB 服務水準協議
- AWS 多區域基礎 (AWS 白皮書）
- 使用 的資料彈性設計模式 AWS(AWS re：Invent 2022 簡報）
- Fidelity 投資和 Reltio 如何使用 Amazon DynamoDB 進行現代化 (AWS re：Invent 2022 簡報）
- 多區域設計模式和最佳實務 (AWS re：Invent 2022 簡報）
- 開啟災難復原 (DR) 架構 AWS，第 III 部分：指示燈和暖待命 (AWS 部落格文章）
- 使用區域鎖定為 Amazon DynamoDB 全域資料表中的項目設定主區域 (AWS 部落格文章）
- 監控 Amazon DynamoDB 的操作意識 (AWS 部落格文章）
- 擴展 DynamoDB：分割區、熱鍵和分割如何影響效能 (AWS 部落格文章）
- 多區域與 DynamoDB 全域資料表的高度一致性 (AWS re：Invent 2024 簡報）

在 DynamoDB 中管理控制平面的最佳實務

> **Note**
>
> DynamoDB 引入每秒 2,500 個請求的控制平面限流，並提供重試選項。請參閱以下更多詳細資訊。

DynamoDB 控制平面操作可讓您管理 DynamoDB 資料表，以及相依於索引等資料表的物件。如需有關這些操作的詳細資訊，請參閱 控制平面。

在某些情況下，您可能需要採取動作，並使用控制平面呼叫傳回的資料做為業務邏輯的一部分。例如，您可能需要知道 `DescribeTable` 返回的 `ProvisionedThroughput` 值。在這些情況下，請遵循下列最佳實務：

- 請勿過度查詢 DynamoDB 控制平面。
- 請勿在同一個程式碼中混用控制平面呼叫和資料平面呼叫。
- 處理控制平面請求上的限流，並使用輪詢重試。
- 從單一用戶端調用和追蹤特定資源的變更。
- 比起在短時間間隔內多次擷取相同資料表的資料，請快取資料以進行處理。

在 DynamoDB 中使用大量資料操作的最佳實務

DynamoDB 支援批次操作，例如使用 `BatchWriteItem`，可同時執行最多 25 個 `PutItem` 和 `DeleteItem` 請求。不過，`BatchWriteItem` 不支援 `UpdateItem` 操作。進行大量更新時，差異取決於需求與更新的性質。您可以使用其他 DynamoDB API，例如 `TransactWriteItems`，其批次大小上限為 100。當涉及更多項目時，您可以使用 Amazon EMR AWS Glue等服務， AWS Step Functions 或使用 DynamoDB-shell 等自訂指令碼和工具進行大量更新。

主題

- 條件式批次更新
- 高效大量操作

條件式批次更新

DynamoDB 支援批次操作，例如使用 `BatchWriteItem`，可在單一批次中同時執行最多 25 個 `PutItem` 與 `DeleteItem` 請求。不過，`BatchWriteItem` 不支援 `UpdateItem` 操作，也不支援條件運算式。可採用替代方案，例如使用其他 DynamoDB API (如 `TransactWriteItems`)，其批次大小上限為 100。

當涉及更多項目，且需要變更主要的資料區塊時，您可以使用 AWS Glue、Amazon EMR 等服務，AWS Step Functions 或使用自訂指令碼和 DynamoDB-shell 等工具來實現高效的大量更新。

適用情境

- DynamoDB-shell 不支援用於生產環境。
- `TransactWriteItems` – 最多可執行 100 個具或不具條件的個別更新，以全有或全無的 ACID 套件形式執行。若應用程式需要冪等性，可在 `TransactWriteItems` 呼叫中加入 `ClientRequestToken`，確保多次相同呼叫的結果與單次呼叫相同。這可確保不會重複執行相同交易，以避免產生錯誤的資料狀態。

  取捨 – 會消耗額外的輸送量。每 1 KB 寫入需耗用 2 個 WCU，而非標準的每 1 KB 寫入 1 個 WCU。
- PartiQL `BatchExecuteStatement` – 最多可執行 25 個具或不具條件的更新。`BatchExecuteStatement` 會一律傳回整體請求的成功回應，並附上保留執行順序的個別操作回應清單。

  取捨 – 對於較大批次，需在用戶端實作額外邏輯，以每 25 個為一組分批發送請求。需考量個別錯誤回應，以決定重試策略。

程式碼範例

這些程式碼範例使用 boto3 程式庫，這是適用於 Python 的 AWS SDK。範例假設您已安裝 boto3，並已使用正確的 AWS 憑證完成設定。

假設一個電器供應商的庫存資料庫，該供應商在歐洲多個城市設有倉儲。由於夏季結束，廠商想要清空桌上風扇庫存，為其他商品騰出空間。廠商希望對義大利倉儲出貨的所有桌上風扇提供價格折扣，但僅在預留庫存達 20 台時適用。DynamoDB 資料表名稱為 warehouse，其金鑰結構包括分割區金鑰 sku (每個產品的唯一識別碼) 與排序索引鍵 warehouse (倉儲識別碼)。

下列 Python 程式碼示範如何透過 `BatchExecuteStatement` API 呼叫執行條件式批次更新。

```
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'])
```

執行後，範例資料將產生以下輸出：

```
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'}}]
```

由於此為內部系統的受限操作，因此未納入冪等性需求考量。可加入額外防護機制，例如僅在價格大於 35 且小於 40 時才允許更新，以提升更新的穩健性。

或者，在冪等性與 ACID 要求更嚴格的情況下，可使用 `TransactWriteItems` 執行相同的批次更新操作。需注意，交易套件中的所有操作要麼全部成功，要麼全部失敗。

假設義大利出現熱浪，導致桌上風扇需求急遽上升。廠商希望將義大利各倉儲出貨的桌上風扇成本提高 20 歐元，但監管機構僅在全庫存中目前成本低於 70 歐元時允許此調漲。價格必須在整個庫存中同時且僅更新一次，且僅當各倉儲的成本低於 70 歐元時才執行更新。

下列 Python 程式碼示範如何透過 `TransactWriteItems` API 呼叫執行此批次更新。

```
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'])
```

執行後，範例資料將產生以下輸出：

```
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'}}]
```

DynamoDB 提供多種執行批次更新的方法。適合的方法取決於 ACID 與/或冪等性需求、更新項目數量，以及對 API 的熟悉程度。

高效大量操作

適用情境

這些模式可協助在 DynamoDB 項目上高效執行大量更新。

- DynamoDB-shell 不支援用於生產環境。
- `TransactWriteItems` – 最多可執行 100 個有或無條件的個別更新，並以「全有或全無」的 ACID 套件方式運行

  取捨 – 額外消耗輸送量，每 1 KB 寫入需耗用 2 WCUs。
- PartiQL `BatchExecuteStatement` – 最多可執行 25 個有或無條件的更新。

  取捨 – 需額外邏輯以 25 筆為一批分配請求。
- AWS Step Functions – 開發人員熟悉的速率限制大量操作 AWS Lambda。

取捨 – 執行時期與速率限制成反比。受限於 Lambda 函式的最大逾時時間。此功能意味著讀取與寫入之間的資料變更可能被覆寫。如需更多資訊，請參閱使用 Amazon EMR 回填 Amazon DynamoDB 存留時間屬性：第 2 部分。

- AWS Glue 和 Amazon EMR – 具有受管平行處理的速率限制大量操作。對於非即時的應用程式或更新，這些選項可在背景執行，僅耗用少量輸送量。兩項服務皆透過 emr-dynamodb-connector 執行 DynamoDB 操作。這些服務會先進行大量讀取，再進行大量寫入更新項目，並可設定速率限制。

  取捨 – 執行時期與速率限制成反比。功能包括可覆寫讀取與寫入期間發生的資料變更。您無法從全域次要索引 (GSIs) 讀取資料。請參閱使用 Amazon EMR 回填 Amazon DynamoDB 存留時間屬性：第 2 部分。
- DynamoDB Shell – 透過類似 SQL 的查詢執行具速率限制的大量操作。您可從 GSIs 讀取資料以提升效率。

  取捨 – 執行時期與速率限制成反比。請參閱在 DynamoDB Shell 中進行速率限制大量操作。

使用此模式

大量更新可能顯著影響成本，特別是在使用隨需輸送量模式時。使用佈建輸送量模式時，速度與成本之間需要權衡。將 rate-limit 參數設定得過於嚴格，可能會造成處理時間過長。您可以根據平均項目大小和速率限制，大致估算更新速度。

或者，您可以根據更新程序的預期持續時間與平均項目大小，估算所需的輸送量。每個模式附帶的部落格參考說明使用該模式的策略、實作方式與限制。如需更多資訊，請參閱使用 Amazon DynamoDB 進行具成本效益的大量處理。

可使用多種方法對即時 DynamoDB 資料表執行大量更新。適合的方法取決於 ACID 與/或冪等性需求、更新項目數量，以及對 API 的熟悉程度。必須同時考量成本與時間的權衡，上述多數方法皆可設定速率限制以控制大量更新任務的輸送量。

在 DynamoDB 中處理並行更新的最佳實務

在分散式系統中，多個程序或使用者可能會嘗試同時修改相同的資料。如果沒有並行控制，這些並行寫入可能會導致遺失更新、資料不一致或競爭條件。DynamoDB 提供多種機制，協助您管理並行存取和維護資料完整性。

> **Note**
>
> 等個別寫入操作`UpdateItem`是原子的，無論並行為何，一律在項目的最新版本上運作。當您的應用程式必須讀取項目，然後根據讀取值 (read-modify-write週期） 將項目寫回時，需要鎖定策略，因為另一個程序可能會修改讀取和寫入之間的項目。

處理並行更新有兩種主要策略：

- 樂觀鎖定 – 假設衝突很少見。它允許並行存取，並使用條件式寫入偵測寫入時的衝突。如果偵測到衝突，寫入會失敗，應用程式可以重試。
- 冪等鎖定 – 假設可能發生衝突。它可透過在修改資源之前取得資源的專屬存取權來防止並行存取。其他程序必須等到鎖定釋放。

下表摘要說明 DynamoDB 中可用的方法：

方法	Mechanism	最適合
樂觀鎖定	版本屬性 + 條件式寫入	爭用性低、成本低廉的重試
預估鎖定（交易）	`TransactWriteItems`	多項目原子性、中度爭用
冪等鎖定（鎖定用戶端）	具有租用和活動訊號的專用鎖定資料表	長時間執行的工作流程、分散式協調

含版本編號的樂觀鎖定

樂觀鎖定是一種策略，可偵測寫入時的衝突，而不是阻止衝突。每個項目都包含一個版本屬性，每次更新都會遞增。更新項目時，您會包含條件表達式，以檢查版本號碼是否符合應用程式上次讀取的值。如果另一個程序同時修改了項目，則條件會失敗，且 DynamoDB 會傳回`ConditionalCheckFailedException`。

何時使用樂觀鎖定

樂觀鎖定非常適合下列情況：

- 多個使用者或程序可能會更新相同的項目，但衝突不常發生。

- 重試失敗的寫入對您的應用程式來說並不昂貴。
- 您想要避免管理分散式鎖定的額外負荷和複雜性。

常見的範例包括電子商務庫存更新、協作編輯平台和金融交易記錄。

取捨

在高度爭用中重試額外負荷

在高並行環境中，衝突發生的可能性提高，可能造成更多重試與更高的寫入成本。

實作複雜度

將版本控制新增至項目和處理條件式檢查，會增加應用程式邏輯的複雜性。適用於 Java v2 的 AWS SDK 增強型用戶端透過 `@DynamoDbVersionAttribute`註釋提供內建支援，可自動為您管理版本編號。

模式設計

在每個項目中包含版本屬性。以下為簡單的結構設計：

- 分割區索引鍵 – 每個項目的唯一識別符（例如 `ItemId`)。
- 屬性：
  - `ItemId` – 項目的唯一識別碼。
  - `Version` – 表示項目版本編號的整數。
  - `QuantityLeft` – 項目的剩餘庫存。

首次建立項目時，`Version` 屬性會設為 1。每次更新時，版本編號會遞增 1。

ItemID（分割區索引鍵）	版本	QuantityLeft
香蕉	1	10
Apples	1	5
橘色	1	7

實作

若要實作樂觀鎖定，請遵循下列步驟：

1. 讀取項目的當前版本。

```
def get_item(item_id):
    response = table.get_item(Key={'ItemID': item_id})
    return response['Item']

item = get_item('Bananas')
current_version = item['Version']
```

2. 使用檢查版本編號的條件表達式更新項目。

```
def update_item(item_id, qty_bought, current_version):
    try:
        response = table.update_item(
            Key={'ItemID': item_id},
            UpdateExpression="SET QuantityLeft = QuantityLeft - :qty, Version
 = :new_v",
            ConditionExpression="Version = :expected_v",
            ExpressionAttributeValues={
                ':qty': qty_bought,
                ':new_v': current_version + 1,
                ':expected_v': current_version
            },
            ReturnValues="UPDATED_NEW"
        )
        return response
    except ClientError as e:
        if e.response['Error']['Code'] == 'ConditionalCheckFailedException':
            print("Version conflict: another process updated this item.")
        raise
```

3. 使用新讀取重試來處理衝突。

   每次重試都需要額外的讀取，因此請限制重試的總數。

```
def update_with_retry(item_id, qty_bought, max_retries=3):
    for attempt in range(max_retries):
        item = get_item(item_id)
        try:
```

```
            return update_item(item_id, qty_bought, item['Version'])
        except ClientError as e:
            if e.response['Error']['Code'] != 'ConditionalCheckFailedException':
                raise
            print(f"Retry {attempt + 1}/{max_retries}")
    raise Exception("Update failed after maximum retries.")
```

對於 Java 應用程式，適用於 Java v2 的 AWS SDK 增強型用戶端透過 `@DynamoDbVersionAttribute`註釋提供內建的樂觀鎖定支援，該註釋會自動為您管理版本編號。

如需條件表達式的詳細資訊，請參閱 DynamoDB 條件表達式 CLI 範例。

使用 DynamoDB 交易進行估算鎖定

DynamoDB 交易提供all-or-nothing方法。當您使用 時`TransactWriteItems`，DynamoDB 會監控交易中的所有項目。如果在交易期間由其他操作修改任何項目，則整個交易都會取消，且 DynamoDB 會傳回 `TransactionCanceledException`。這種行為提供一種可觀的並行控制形式，因為防止衝突並行修改，而不是在事實之後偵測到。

何時使用交易進行鎖定

在下列情況下，交易非常適合：

- 您需要以原子方式更新多個項目，無論是在同一資料表內或跨資料表。
- 您的商業邏輯需要all-or-nothing語意 - 所有變更都成功套用或不套用。

常見範例包括在帳戶之間轉移資金、下訂單同時更新庫存和訂單資料表，以及在遊戲中的玩家之間交換項目。

取捨

較高的寫入成本

對於高達 1 KB 的項目，相較於標準寫入的 1 WCUs，交易會耗用每個項目 2 個 WCU（一個用於準備，一個用於遞交）。

項目限制

單一交易最多可包含一或多個資料表中的 100 個動作。

衝突敏感度

如果交易中的任何項目被另一個操作修改，則整個交易會失敗。在高內容案例中，這可能會導致頻繁取消。

實作

下列範例使用 在兩個項目之間TransactWriteItems以原子方式傳輸庫存。如果另一個程序在交易期間修改任一項目，則會復原整個操作。

```
import boto3

client = boto3.client('dynamodb')

def transfer_inventory(source_id, target_id, quantity):
    try:
        client.transact_write_items(
            TransactItems=[
                {
                    'Update': {
                        'TableName': 'Inventory',
                        'Key': {'ItemID': {'S': source_id}},
                        'UpdateExpression': 'SET QuantityLeft = QuantityLeft - :qty',
                        'ConditionExpression': 'QuantityLeft >= :qty',
                        'ExpressionAttributeValues': {
                            ':qty': {'N': str(quantity)}
                        }
                    }
                },
                {
                    'Update': {
                        'TableName': 'Inventory',
                        'Key': {'ItemID': {'S': target_id}},
                        'UpdateExpression': 'SET QuantityLeft = QuantityLeft + :qty',
                        'ExpressionAttributeValues': {
                            ':qty': {'N': str(quantity)}
                        }
                    }
                }
            ]
        )
        return True
```

```
    except client.exceptions.TransactionCanceledException as e:
        print(f"Transaction canceled: {e}")
        return False
```

在此範例中，條件表達式會檢查是否存在足夠的庫存，但不需要版本屬性。如果準備和遞交階段之間的另一個操作修改了交易中的任何項目，DynamoDB 會自動取消交易。這就是提供樂觀並行控制的原因 – 交易本身會防止衝突並行修改。

> **Note**
>
> 您可以將版本檢查新增為其他條件表達式，以結合交易與樂觀鎖定。這提供額外的保護層，但並非交易偵測衝突的必要條件。

如需詳細資訊，請參閱管理 DynamoDB 交易的複雜工作流程。

使用 DynamoDB 鎖定用戶端進行分散式鎖定

對於需要傳統lock-acquire-release語意的應用程式，DynamoDB Lock Client 是一個開放原始碼程式庫，使用 DynamoDB 資料表做為鎖定存放區來實作分散式鎖定。當您需要在多個應用程式執行個體之間協調對外部資源（例如 S3 物件或共用組態）的存取時，此方法非常有用。

鎖定用戶端可作為開放原始碼 Java 程式庫使用。

運作方式

鎖定用戶端使用專用的 DynamoDB 資料表來追蹤鎖定。每個鎖定會以具有下列金鑰屬性的項目表示：

- 識別要鎖定之資源的分割區索引鍵。
- 指定鎖定有效時間的租用持續時間。如果鎖定持有者當機或變得沒有回應，鎖定會在租用期間之後自動過期。
- 鎖定持有者定期傳送的活動訊號，以延長租用。這可防止鎖定在持有人仍在主動處理時過期。

鎖定用戶端使用條件式寫入，以確保一次只有一個程序可以取得鎖定。如果鎖定已保留，發起人可以選擇等待並重試或立即失敗。

何時使用鎖定用戶端

鎖定用戶端非常適合下列情況：

- 您需要協調跨多個應用程式執行個體或微服務對共用資源的存取。
- 關鍵區段長時間執行（秒到分鐘），在衝突時重試整個操作會很昂貴。
- 您需要自動鎖定過期，才能正常處理程序失敗。

常見範例包括協調分散式工作流程、跨多個執行個體協調 Cron 任務，以及管理對共用外部資源的存取。

取捨

其他基礎設施

需要專用的 DynamoDB 資料表來進行鎖定管理，並針對鎖定操作和活動訊號提供額外的讀取和寫入容量。

時鐘相依性

鎖定到期取決於時間戳記。用戶端之間的重大時鐘扭曲可能會導致意外行為，尤其是在短期租用期間。

死鎖風險

如果您的應用程式在多個資源上取得鎖定，您必須以一致順序取得它們，以避免死結。租用期間會自動從沒有回應的持有人釋放鎖定，以提供安全網路。

實作

下列範例示範如何使用 DynamoDB 鎖定用戶端來取得和釋放鎖定：

```
import java.io.IOException;
import java.util.Optional;
import java.util.concurrent.TimeUnit;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

final DynamoDbClient dynamoDB = DynamoDbClient.builder()
    .region(Region.US_WEST_2)
    .build();

final AmazonDynamoDBLockClient lockClient = new AmazonDynamoDBLockClient(
    AmazonDynamoDBLockClientOptions.builder(dynamoDB, "Locks")
        .withTimeUnit(TimeUnit.SECONDS)
        .withLeaseDuration(10L)
```

```
        .withHeartbeatPeriod(3L)
        .withCreateHeartbeatBackgroundThread(true)
        .build());

// Try to acquire a lock on a resource
final Optional<LockItem> lock =
    lockClient.tryAcquireLock(AcquireLockOptions.builder("my-shared-
resource").build());

if (lock.isPresent()) {
    try {
        // Perform operations that require exclusive access
        processSharedResource();
    } finally {
        // Always release the lock when done
        lockClient.releaseLock(lock.get());
    }
} else {
    System.out.println("Failed to acquire lock.");
}

lockClient.close();
```

> **Important**
>
> 始終在finally區塊中釋放鎖定，以確保即使您的處理邏輯擲回例外狀況，也會釋放鎖定。未發行的鎖定會封鎖其他程序，直到租用過期為止。

您也可以直接使用條件式寫入實作簡單的鎖定機制，而無需鎖定用戶端程式庫。下列範例使用UpdateItem搭配條件表達式來取得鎖定，並加以DeleteItem釋放：

```
from datetime import datetime, timedelta
from boto3.dynamodb.conditions import Attr

def acquire_lock(table, resource_name, owner_id, ttl_seconds):
    """Attempt to acquire a lock. Returns True if successful."""
    expiry = (datetime.now() + timedelta(seconds=ttl_seconds)).isoformat()
    now = datetime.now().isoformat()
    try:
        table.update_item(
            Key={'LockID': resource_name},
```

```
            UpdateExpression='SET #owner = :owner, #expiry = :expiry',
            ConditionExpression=Attr('LockID').not_exists() |
 Attr('ExpiresAt').lt(now),
            ExpressionAttributeNames={'#owner': 'OwnerID', '#expiry': 'ExpiresAt'},
            ExpressionAttributeValues={':owner': owner_id, ':expiry': expiry}
        )
        return True
    except table.meta.client.exceptions.ConditionalCheckFailedException:
        return False

def release_lock(table, resource_name, owner_id):
    """Release a lock. Only succeeds if the caller is the lock owner."""
    try:
        table.delete_item(
            Key={'LockID': resource_name},
            ConditionExpression=Attr('OwnerID').eq(owner_id)
        )
        return True
    except table.meta.client.exceptions.ConditionalCheckFailedException:
        return False
```

此方法使用條件表達式，以確保只有在鎖定不存在或已過期時，才能取得鎖定，而且只能由取得鎖定的程序釋出。請考慮啟用鎖定資料表上的存留時間 (TTL)，以自動清除過期的鎖定項目。

選擇並行控制策略

使用下列準則為您的工作負載選擇正確的方法：

在下列情況下使用樂觀鎖定：

- 衝突不常發生。
- 重試失敗的寫入並不昂貴。
- 您一次更新單一項目。

在下列情況下使用交易：

- 您需要以原子方式更新多個項目。
- 您需要跨項目all-or-nothing語意。
- 您需要在單一操作中結合條件檢查與寫入。

在下列情況下使用鎖定用戶端：

- 您需要協調跨分散式程序對外部資源的存取。
- 關鍵區段長時間執行，在衝突上重試非常昂貴。

- 您需要自動鎖定過期，才能處理程序失敗。

> **Note**
>
> 如果您使用 DynamoDB 全域資料表，請注意全域資料表會使用「最後寫入器獲勝」對帳策略進行並行更新。具有版本號碼的樂觀鎖定在跨區域無法如預期運作，因為一個區域中的寫入可能會在沒有版本檢查的情況下覆寫另一個區域中的並行寫入。設計您的應用程式，以便在使用全域資料表時處理應用程式層級的衝突。

了解 DynamoDB 中 AWS 帳單和用量報告的最佳實務

本文件說明與 DynamoDB 費用相關的 `UsageType` 帳單代碼。

AWS 提供成本和用量報告 (CUR)，其中包含所使用服務的資料。您可以使用 AWS Cost and Usage Report 以 CSV 格式將帳單報告發佈至 Amazon S3。設定 CUR 時，您可選擇依小時、天或月份細分時段，並決定是否依資源 ID 細分用量。如需產生 CUR 的詳細資訊，請參閱建立成本與用量報告

在 CSV 匯出檔中，您可看到每行列出的相關屬性。以下為可能包含的屬性範例：

- lineitem/UsageStartDate：明細項目的開始日期與時間 (含)，以 UTC 表示。
- lineitem/UsageEndDate：對應明細項目的結束日期與時間 (不含)，以 UTC 表示。
- lineitem/ProductCode：對於 DynamoDB，此欄位值為 "AmazonDynamoDB"。
- lineitem/UsageType：用量類型的特定描述代碼，如本文件所述。
- lineitem/Operation：提供費用內容的名稱，例如產生費用的操作名稱 (選用)。
- lineitem/ResourceId：產生用量的資源識別符。若 CUR 包含依資源 ID 細分的明細，此欄位可用。
- lineitem/UsageAmount：指定時段內產生的用量。
- lineitem/UnblendedCost：此用量的成本。
- lineitem/LineItemDescription：明細項目的文字描述。

如需 CUR 資料字典的詳細資訊，請參閱成本與用量報告 (CUR) 2.0。請注意，實際名稱會依內容而異。

`UsageType` 是一個字串，其值可能為 `ReadCapacityUnit-Hrs`、`USW2-ReadRequestUnits`、`EU-WriteCapacityUnit-Hrs` 或 `USE1-TimedPITRStorage-ByteHrs`。

每個用量類型皆以可選的區域字首開頭。若未出現，則代表 us-east-1 區域。若存在，下表會對應短帳單區域代碼與標準區域代碼及名稱。

例如，名為 USW2-ReadRequestUnits 的用量表示在 us-west-2 區域中消耗的讀取請求單位。

帳單區域代碼	區域代碼	區域名稱
AFS1	af-south-1	非洲 (開普敦)
APE1	ap-east-1	亞太地區 (香港)
APN1	ap-northeast-1	亞太地區 (東京)
APN2	ap-northeast-2	亞太地區 (首爾)
APN3	ap-northeast-3	亞太地區 (大阪)
APS1	ap-southeast-1	亞太地區 (新加坡)
APS2	ap-southeast-2	亞太地區 (雪梨)
APS3	ap-south-1	亞太地區 (孟買)
APS4	ap-southeast-3	亞太地區 (雅加達)
APS5	ap-south-2	亞太地區 (海德拉巴)
APS6	ap-southeast-4	亞太地區 (墨爾本)
CAN1	ca-central-1	加拿大 (中部)
歐盟	eu-west-1	歐洲 (愛爾蘭)
EUC1	eu-central-1	歐洲 (法蘭克福)
EUC2	eu-central-2	歐洲 (蘇黎世)
EUN1	eu-north-1	歐洲 (斯德哥爾摩)
EUS1	eu-south-1	歐洲 (米蘭)
EUS2	eu-south-2	歐洲 (西班牙)

帳單區域代碼	區域代碼	區域名稱
EUW1	eu-west-1	歐洲 (愛爾蘭)
EUW2	eu-west-2	歐洲 (倫敦)
EUW3	eu-west-3	Europe (Paris)
ILC1	il-central-1	以色列 (特拉維夫)
MEC1	me-central-1	中東 (阿拉伯聯合大公國)
MES1	me-south-1	Middle East (Bahrain)
SAE1	sa-east-1	南美洲 (聖保羅)
USE1 (預設)	us-east-1	美國東部 (維吉尼亞北部)
USE2	us-east-2	美國東部 (俄亥俄)
UGE1	us-gov-east-1	美國政府東部區域
UGW1	us-gov-west-1	美國政府西部區域
USW1	us-west-1	美國西部 (加利佛尼亞北部)
USW2	us-west-2	美國西部 (奧勒岡)

在以下各節中，我們使用 REG-UsageType 模式說明 DynamoDB 的費用，其中 REG 表示用量發生的區域，usageType 為費用類型代碼。例如，若您在 CSV 檔案中看到 USW1- ReadCapacityUnit-Hrs 的明細項目，代表該筆用量為 US-West-1 的佈建讀取容量。在此情況下，清單會顯示 REG-ReadCapacityUnit-Hrs。

主題

- 輸送容量
- 串流
- 儲存
- 備份與還原
- 資料傳輸

- CloudWatch Contributor Insights
- DynamoDB Accelerator (DAX)

輸送容量

佈建容量的讀取與寫入

在佈建容量模式下建立 DynamoDB 資料表時，您需指定應用程式所需的讀寫容量。用量類型取決於資料表類別 (標準或標準不常存取)。您依每秒的耗用率佈建讀寫容量，但費用依佈建容量按小時計價。

UsageType	單位	精細程度	Description
REG-ReadCapacityUnit-Hrs	RCU 小時	小時	佈建容量模式下使用 Standard 資料表類別進行讀取的費用。
REG-IA-ReadCapacityUnit-Hrs	RCU 小時	小時	佈建容量模式下使用 Standard-IA 資料表類別進行讀取的費用。
REG-WriteCapacityUnit-Hrs	WCU 小時	小時	佈建容量模式下使用 Standard 資料表類別進行寫入的費用。
REG-IA-WriteCapacityUnit-Hrs	WCU 小時	小時	佈建容量模式下使用 Standard-IA 資料表類別進行寫入的費用。

預留容量的讀取與寫入

使用預留容量，您會支付一次性預付費用並承諾一段時間的最低佈建消費額。預留容量以折扣後的小時計費。任何超過預留容量的佈建容量都會依標準佈建容量費率計費。預留容量適用於使用 Standard 資料表類別的 DynamoDB 資料表中，單一區域的佈建讀寫容量單位 (RCU 和 WCU)。1 年與 3 年的預留容量均使用相同 SKU 計費。

UsageType	單位	精細程度	Description
REG-HeavyUsage：dynamodb.read	RCU 小時	先預付後每月計費	預留容量讀取費用包括一次性預付款與每月月初的月費，涵蓋該月所有折扣後承諾的 RCU 小時。將包含相符的零成本 REG-ReadCapacityUnit-Hrs 明細項。
REG-HeavyUsage：dynamodb.write	WCU 小時	先預付後每月計費	預留容量寫入費用包括一次性預付款與每月月初的月費，涵蓋該月所有折扣後承諾的 WCU 小時。將包含相符的零成本 REG-WriteCapacityUnit-Hrs 明細項。

隨需容量的讀取與寫入

在隨需容量模式下建立 DynamoDB 資料表時，您僅需支付應用程式實際執行的讀寫費用。讀取與寫入請求的價格取決於資料表類別。

UsageType	單位	精細程度	Description
REG-ReadRequestUnits	RRUs	單位	隨需容量模式下使用標準資料表類別進行讀取的費用。
REG-IA-ReadRequestUnits	RRUs	單位	隨需容量模式下使用標準不常存取資料表類別進行讀取的費用。

UsageType	單位	精細程度	Description
REG-WriteRequestUnits	WRUs	單位	隨需容量模式下使用標準資料表類別進行寫入的費用。
REG-IA-WriteRequestUnits	WRUs	單位	隨需容量模式下使用標準不常存取資料表類別進行寫入的費用。

全域資料表的讀取與寫入

DynamoDB 會依各複本資料表的資源使用情況收取全域資料表用量費用。對於佈建的全域資料表，寫入請求以複寫寫入容量單位 (rWCU) 計量，而非標準寫入容量單位 (WCU)；全域資料表中的全域次要索引寫入則以 WCU 計量。對於隨需全域資料表，寫入請求以複寫寫入請求單位 (rWRU) 計量，而非標準寫入請求單位 (WRU)。用於複寫的 rWCU 或 rWRU 數量取決於所使用的全域資料表版本。定價取決於資料表類別。

全域次要索引 (GSI) 的寫入以標準寫入單位 (WCU 與 WRU) 計費。讀取請求與資料儲存的計費方式與單一區域資料表相同。

當您在新區域中新增資料表複本以建立或擴展全域資料表時，DynamoDB 會依還原資料量 (每 GB) 向新增區域的資料表收取還原費用。還原資料依 REG-RestoreDataSize-Bytes 計費。如需詳細資訊，請參閱 DynamoDB 的備份與還原。跨區域複寫或將副本新增至含資料的資料表時，會產生資料傳出費用。

當您為 DynamoDB 全域資料表選擇隨需容量模式時，僅需支付應用程式在各複本資料表上實際使用的資源費用。

UsageType	單位	精細程度	Description
REG-ReplWriteCapacityUnit-Hrs	rWCU 小時	小時	全域資料表 (佈建，標準資料表類別)。
REG-IA-ReplWriteCapacityUnit-Hrs	rWCU 小時	小時	全域資料表 (佈建，標準不常存取資料表類別)。

UsageType	單位	精細程度	Description
REG-ReplWriteRequestUnits	rWRU	單位	全域資料表 (隨需，標準資料表類別)。
REG-IA-ReplWriteRequestUnits	rWRU	單位	全域資料表 (隨需，標準不常存取資料表類別)

串流

DynamoDB 提供兩種串流技術：DynamoDB Streams 與 Kinesis。兩者分別有獨立的定價。

DynamoDB Streams 依讀取請求單位計費。每個 `GetRecords` API 呼叫都會視為一次串流讀取請求並計費。在 DynamoDB 觸發程序 AWS Lambda 中呼叫的 `GetRecords` API 呼叫，或在複寫中由 DynamoDB 全域資料表呼叫的 API 呼叫，您不需要付費。

UsageType	單位	精細程度	Description
REG-Streams-RequestsCount	計數	單位	DynamoDB Streams 的讀取請求單位。

Amazon Kinesis Data Streams 依變更資料擷取單位計費。DynamoDB 會針對每次寫入 (最多 1 KB) 收取一個變更資料擷取單位費用。超過 1 KB 的項目需要額外的變更資料擷取單位。您只需為應用程式執行的寫入付費，無需管理資料表的輸送容量。

UsageType	單位	精細程度	Description
REG-ChangeDataCaptureUnits-Kinesis	CDC 單位	單位	Kinesis Data Streams 的變更資料擷取單位。

儲存

DynamoDB 會根據您啟用的功能，將資料的原始位元組大小與每個項目的儲存額外負荷相加，以計算可計費資料大小。

> **Note**
>
> 與使用 `DescribeTable` 時相比，CUR 中的儲存使用量會較高，因為 `DescribeTable` 不包含每個項目的儲存額外負荷。

儲存使用量以每小時計算，但價格依每月的每小時平均費用計算。

雖然儲存 `UsageType` 使用 `ByteHrs` 作為尾碼，但 CUR 中的儲存使用量以 GB 為單位，並以 GB-月計價。

UsageType	單位	精細程度	Description
REG-TimedStorage-ByteHrs	GB	月	DynamoDB 標準資料表類別的資料表與索引所使用的儲存量。
REG-IA-TimedStorage- ByteHrs	GB	月	DynamoDB Standard-IA 資料表類別的資料表與索引所使用的儲存量。

備份與還原

DynamoDB 提供兩種類型的備份：時間點復原 (PITR) 備份與隨需備份。使用者也能從這些備份還原至 DynamoDB 資料表。下列費用適用於備份與還原。

備份儲存費用會在每月一日產生，並隨備份新增或移除於整月期間進行調整。如需更多資訊，請參閱了解 Amazon DynamoDB 隨需備份與計費部落格。

UsageType	單位	精細程度	Description
REG-Timed BackupStorage-ByteHrs	GB	月	DynamoDB 資料表與本機次要索引的隨需備份所使用的儲存使用量。

UsageType	單位	精細程度	Description
TimedPITRStorage-ByteHrs	GB	月	時間點復原 (PITR) 備份所使用的儲存使用量。只要啟用 PITR，DynamoDB 會在整個月持續監控相關資料表大小，以計算備份與儲存費用。
REG-RestoreDataSize-Bytes	GB	大小	從 DynamoDB 備份中還原的資料總大小 (以 GB 計)，包括資料表資料、本機次要索引與全域次要索引。

AWS Backup

AWS Backup 是一種全受管備份服務，可讓您輕鬆集中和自動化雲端和內部部署中跨 AWS 服務的資料備份。 AWS Backup 會收取儲存 （暖儲存或冷儲存）、還原活動和跨區域資料傳輸的費用。下列 `UsageType` 費用會顯示於 “AWS Backup” ProductCode 下，而非 “AmazonDynamoDB”。

UsageType	單位	精細程度	Description
REG-WarmStorage-ByteHrs-DynamoDB	GB	月	DynamoDB 備份在整個月由 AWS Backup 管理所使用的儲存使用量 (以 GB-月計)。
REG-CrossRegion-WarmBytes-DynamoDB	GB	大小	資料會傳輸至相同 帳戶中的不同 AWS 區域，或傳輸至不同的 AWS 帳戶。當備份從一個區域複製至另一個區域時，會產生跨區域傳輸費用。費用

UsageType	單位	精細程度	Description
			一律向資料來源帳戶收取。
REG-Restore-WarmBytes-DynamoDB	GB	大小	從暖儲存還原的資料總大小 (以 GB 計)。
REG-ColdStorage-ByteHrs-DynamoDB	GB	月	由 AWS Backup 在整個月期間管理的 DynamoDB 備份所使用的冷儲存 (以 GB-月計)。
REG-Restore-ColdBytes-DynamoDB	GB	月	從冷儲存還原的資料總大小 (以 GB 計)。

匯出與匯入

您可將資料自 DynamoDB 匯出至 Amazon S3，或自 Amazon S3 匯入至新的 DynamoDB 資料表。

雖然 `UsageType` 使用 `Bytes` 作為尾碼，但 CUR 中的匯出與匯入用量是以 GB 為單位進行計量與定價。

UsageType	單位	精細程度	Description
REG-ExportDataSize-Bytes	GB	大小	將資料匯出至 S3 的費用。DynamoDB 會根據匯出建立時的指定時間點，依據 DynamoDB 基礎資料表 (資料表資料與本機次要索引) 的大小計費。
REG-ImportDataSize-Bytes	GB	大小	將資料自 S3 匯入的費用。資料大小根據

UsageType	單位	精細程度	Description
			Amazon S3 中未壓縮物件的大小計算。將資料匯入含有 GSI 的資料表不會產生額外費用。
REG-IncrementalExportDataSize-Bytes	GB	大小	針對從連續備份處理以產生增量匯出的資料大小所收取的費用。

資料傳輸

資料傳輸活動可能會顯示為與 DynamoDB 服務相關。DynamoDB 不會針對傳入資料傳輸收費，也不會針對 DynamoDB 與 AWS 相同 AWS 區域內其他服務之間的資料傳輸收費（亦即每 GB 0.00 美元）。跨 AWS 區域傳輸的資料（例如美國東部 【維吉尼亞北部】 區域的 DynamoDB 和歐洲 【愛爾蘭】 區域的 Amazon EC2 之間）會在傳輸的兩側收費。

UsageType	單位	精細程度	Description
REG-DataTransfer-In-Bytes	GB	單位	從網際網路傳入至 DynamoDB 的資料。
REG-DataTransfer-Out-Bytes	GB	單位	從 DynamoDB 傳出至網際網路的資料。

CloudWatch Contributor Insights

CloudWatch Contributor Insights for DynamoDB 是診斷工具，用於識別 DynamoDB 資料表中最常存取及受限流的索引鍵。下列 `UsageType` 費用會顯示於 “AmazonCloudWatch” ProductCode 下，而非 “AmazonDynamoDB”。

UsageType	單位	精細程度	Description
REG-CW：ContributorEventsManaged	已處理事件	單位	已處理的 DynamoDB 事件總數。例如，啟用 CloudWatch Contributor Insights 的資料表中，每次讀取或寫入項目都會計為一個事件。若資料表含有排序索引鍵，則會產生兩個事件的收費。
REG-CW：ContributorRulesManaged	規則數量	月	當您啟用 CloudWatch Contributor Insights 時，DynamoDB 會建立規則以識別最常被存取的項目與最常受限流的索引鍵。此費用是因為為每個設定用於記錄 CloudWatch 貢獻者洞察的實體 (資料表與 GSI) 新增規則而產生。

DynamoDB Accelerator (DAX)

DynamoDB Accelerator (DAX) 會依服務所選的執行個體類型按小時計費。下列費用與已佈建的 DynamoDB Accelerator 執行個體相關。下列 `UsageType` 費用會列於 “AmazonDAX” ProductCode 下，而非 “AmazonDynamoDB”。

UsageType	單位	精細程度	Description
REG-NodeUsage：dax-<INSTANCE TYPE>	節點小時	小時	特定執行個體類型的每小時使用量。定價以實際使用的節點小

UsageType	單位	精細程度	Description
			時為單位，計費期間自節點啟動至終止。任一部分節點小時皆以整小時計費。DAX 叢集中每個節點均會產生 DAX 費用。若叢集包含多個節點，帳單報告中將顯示多筆明細。

執行個體類型將為下列清單之一。如需節點類型詳細資訊，請參閱 節點。

- r3.2xlarge、r4.8xlarge 或 r5.8xlarge
- r3.4xlarge、r4.large 或 r5.large
- r3.8xlarge、r4.xlarge 或 r5.xlarge
- r3.2xlarge、r5.12xlarge 或 t2.medium
- r3.4xlarge、r4.large 或 r5.large
- r3.xlarge、r5.16xlarge 或 t2.small
- r4.16xlarge、r5.24xlarge 或 t3.medium
- r4.2xlarge、r5.2xlarge 或 t3.small
- r4.4xlarge 或 r5.4xlarge

將 DynamoDB 資料表從一個帳戶移轉至另一帳戶

您可以將 Amazon DynamoDB 資料表從一個帳戶移轉至另一帳戶，以實施多帳戶或備份策略。您也可因測試、偵錯或合規需求執行此操作。常見的使用案例是跨生產、預備、測試和開發環境複製 DynamoDB 資料表，其中每個環境都會使用不同的 AWS 帳戶。

DynamoDB 提供兩個選項，可將資料表從一個帳戶遷移到另一個 AWS 帳戶：

- AWS Backup for Cross-Account Backup and Restore： AWS Backup 是一項全受管備份服務，可讓您集中管理多個 AWS 服務的備份。使用其跨帳戶備份與還原功能，可在一帳戶中備份 DynamoDB 資料表，並將其還原至同一 AWS 組織內的另一帳戶。

- DynamoDB 匯出和匯入至 Amazon S3：使用 DynamoDB 匯出和匯入至 Amazon S3 功能可讓您完整匯出至 Amazon S3 儲存貯體，然後將該資料匯入至另一個 AWS 帳戶中的新資料表。當您需要在不屬於相同 AWS 組織的帳戶之間遷移，或者您不想使用 時，此方法很適合 AWS Backup。

> **Note**
>
> 從 Amazon S3 匯入不支援含本機次要索引 (LSI) 的資料表，但支援全域次要索引 (GSI)。如需 LSI 與 GSI 的詳細資訊，請參閱 在 DynamoDB 中使用次要索引來改善資料存取。

主題

- 使用 遷移資料表 AWS Backup 以進行跨帳戶備份和還原
- 使用匯出至 S3 並從 S3 匯入的方式移轉資料表

使用 遷移資料表 AWS Backup 以進行跨帳戶備份和還原

先決條件

- 來源和目標 AWS 帳戶必須屬於 Organizations AWS 服務中的相同組織
- 建立和使用 AWS Backup 保存庫的有效 AWS Identity and Access Management (IAM) 許可

如需設定跨帳戶備份的詳細資訊，請參閱跨 AWS 帳戶建立備份複本。

定價資訊

AWS 備份 （根據資料表大小）、區域間 AWS 的任何資料複製 （根據資料量）、還原 （根據資料量） 和任何持續儲存費用的費用。為避免持續收費，若還原後不再需要備份，可將其刪除。

如需定價的詳細資訊，請參閱 AWS Backup 定價。

步驟 1：啟用 DynamoDB 與跨帳戶備份的進階功能

1. 在來源和目標 AWS 帳戶中，存取 AWS 管理主控台並開啟 AWS Backup 主控台。
2. 選取設定選項。
3. 在 Amazon DynamoDB 備份的進階功能下，確認進階功能已啟用。若未啟用，請選取啟用。
4. 在跨帳戶管理下，針對跨帳戶備份，選取開啟。

步驟 2：在來源帳戶與目標帳戶中建立備份保存庫

1. 在來源 AWS 帳戶中，開啟 AWS Backup 主控台。
2. 選擇 Backup vaults (備份文件庫)。
3. 選擇 Create backup vault (建立備份文件庫)。
4. 複製並儲存已建立備份文件庫和目標 AWS 帳戶的 Amazon Resource Name (ARN)。
5. 在帳戶間複製 DynamoDB 資料表備份時，您將需要來源與目標備份保存庫的 ARN。

步驟 3：在來源帳戶中建立 DynamoDB 資料表備份

1. 在 AWS 備份儀表板頁面上，選取建立隨需備份。
2. 在設定區段中，將 DynamoDB 選為資源類型，然後選取資料表名稱。
3. 在備份保存庫下拉式清單中，選取您於來源帳戶建立的備份保存庫。
4. 選取所需的保留期間。
5. 選擇 Create on-demand backup (建立隨需備份)。
6. 在備份任務頁面的 AWS 備份任務分頁上監控備份任務狀態。

步驟 4：將 DynamoDB 資料表備份從來源帳戶複製至目標帳戶

1. 備份任務完成後，在來源帳戶中開啟 AWS Backup 主控台，然後選擇備份保存庫。
2. 在備份項目下，選擇 DynamoDB 資料表備份。選擇動作，再選擇複製。
3. 輸入目標帳戶的 AWS 區域。
4. 在外部保存庫 ARN 欄位中，輸入您於目標帳戶建立的備份保存庫 ARN。
5. 在目標帳戶的備份保存庫中，啟用來源帳戶的存取權限，以允許複製備份。

步驟 5：在目標帳戶中還原 DynamoDB 資料表備份

1. 在目標 AWS 帳戶中，開啟 AWS Backup 主控台，然後選擇備份保存庫
2. 在備份項目下，選取從來源帳戶複製的備份。選擇動作，再選擇還原。
3. 輸入新 DynamoDB 資料表名稱、加密設定、用於加密還原的金鑰與其他設定選項。
4. 還原完成後，資料表狀態將顯示為作用中。

使用匯出至 S3 並從 S3 匯入的方式移轉資料表

先決條件

- 您必須為資料表啟用時間點復原 (PITR)，才能執行匯出至 S3。如需詳細資訊，請參閱DynamoDB 的時間點復原。
- 具有執行匯出所需的有效 IAM 權限。如需詳細資訊，請參閱請求在 DynamoDB 中匯出資料表。
- 具有足以執行匯入的有效 IAM 權限。如需詳細資訊，請參閱請求在 DynamoDB 中匯入資料表。

定價資訊

AWS PITR 的費用 （ 根據資料表的大小和啟用 PITR 的時間長度 ）。若僅為匯出啟用 PITR，可在匯出完成後關閉。 AWS 也會針對 S3 請求、匯出資料儲存與匯入作業 (依未壓縮資料大小) 收費。

如需 DynamoDB 定價詳細資訊，請參閱 DynamoDB 定價。

> **Note**
>
> 從 S3 匯入至 DynamoDB 時，物件大小與數量皆有限制。如需詳細資訊，請參閱匯入配額。

請求將資料表匯出至 Amazon S3

1. 登入 AWS 管理主控台並開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Exports to S3 (匯出至 S3)。
3. 選擇來源資料表和目的地 S3 儲存貯體。使用 `s3://bucketname/prefix` 格式輸入目標帳戶的儲存貯體 URL。`/prefix` 是可協助整理目的地儲存貯體的選用資料夾。
4. 選取完整匯出。完整匯出會在您指定的時間點輸出資料表的完整快照。
   a. 選取目前時間以匯出最新的完整資料表快照。
   b. 對於匯出的檔案格式，選擇 DynamoDB JSON 和 Amazon Ion 兩者之一。預設為 DynamoDB JSON。
5. 按一下 Export (匯出) 按鈕開始匯出。
6. 小型資料表匯出通常在數分鐘內完成，但 TB 級資料表可能需超過一小時。

從 Amazon S3 請求匯入資料表

1. 登入 AWS 管理主控台並開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Import from S3 (從 S3 匯入)。
3. 在出現的頁面上，選取 Import from S3 (從 S3 匯入)。
4. 輸入 Amazon S3 來源 URL。您也可以使用瀏覽 S3 按鈕找到它。預期的路徑格式為 `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/data/`。
5. 指定您是否為 S3 bucket owner (S3 儲存貯體擁有者)。
6. 在匯入檔案壓縮中，選取 GZIP 以符合匯出。
7. 在匯入檔案格式中，選取 DynamoDB JSON 以符合匯出。
8. 選取下一步。在指定資料表詳細資訊中，選取要建立的新資料表選項以儲存您的資料。
9. 選取下一步。在設定資料表設定中，如有需要可自訂其他設定。
10. 選擇 Next (下一步) 以再次檢視您的匯入選項，然後按一下 Import (匯入)，以啟動匯入任務。您會在從 S3 匯入下看到新資料表，其狀態為匯入中。此期間無法存取資料表。小型匯入通常在數分鐘內完成，但 TB 級資料表可能需超過一小時。
11. 匯入完成後，狀態將顯示為啟用，即可開始使用資料表。

在移轉期間保持資料表同步

若能在移轉期間暫停來源資料表的寫入操作，來源與輸出將在移轉後完全一致。若無法暫停寫入操作，移轉後目標資料表通常會落後來源。若要使來源資料表同步，可使用串流 (DynamoDB Streams 或適用於 DynamoDB 的 Kinesis Data Streams) 重播自備份或匯出以來的寫入。

應在將來源資料表匯出至 S3 的時間戳記之前開始讀取串流記錄。例如，若匯出至 S3 發生於下午 2:00，而目標資料表匯入於下午 11:00 完成，應在下午 1:58 啟動 DynamoDB 串流讀取。「變更資料擷取」的串流選項表摘要說明各串流模型的功能。

使用 DynamoDB Streams 搭配 Lambda，可提供精簡的資料同步方法，讓來源與目標 DynamoDB 資料表保持一致。可使用 Lambda 函式重播目標資料表中的每筆寫入。

> **Note**
>
> 項目會在 DynamoDB Streams 中保留 24 小時，因此應規劃在此期間內完成備份與還原，或匯出與匯入作業。

整合 DAX 與 DynamoDB 應用程式的設計指南

DynamoDB Accelerator (DAX) 是與 DynamoDB 相容的快取服務，可為高效能需求的應用程式 (如讀取密集型應用程式) 提供快速的記憶體內處理效能。使用 DAX 可在存取常用資料時達成微秒級回應時間。本 DynamoDB Accelerator 指南提供整合 DAX 與 DynamoDB 應用程式的完整見解與最佳實務。

本指南提供 DAX 初學者及欲最佳化現有設定者的基礎知識。本指南涵蓋多項主題，例如 DAX 的使用時機與 DAX 叢集的建立。內容亦包含實際範例與詳細說明，協助您在專案中有效實作 DAX。最後，本指南提供可實作的進階策略，以充分發揮 DAX 快取效能，確保應用程式具備高速度與可擴展性。

主題

- 評估 DAX 是否適用於您的使用案例
- 設定 DAX 用戶端
- 設定 DAX 叢集
- 調整 DAX 叢集大小
- 部署叢集
- 管理叢集作業
- 監控 DAX

評估 DAX 是否適用於您的使用案例

本節說明 DAX 的使用時機與原因。使用本指南可協助您判斷，整合 DAX 與 DynamoDB 是否有助滿足應用程式的工作負載模式、效能與資料一致性需求。本指南亦說明 DAX 不適用的情境，例如寫入密集的工作負載或低頻存取資料。

本節內容

- 選擇 DAX 的時機與原因
- 何時不使用 DAX

選擇 DAX 的時機與原因

您可在多種情境下考慮將 DAX 新增至應用程式堆疊。例如，可使用 DAX 降低 DynamoDB 讀取請求的整體延遲，或減少資料表中相同資料的重複讀取。以下列出可透過整合 DAX 與 DynamoDB 取得效益的典型情境：

- 高效能需求

  - 低延遲讀取 – 若應用程式的最終一致讀取需達微秒級回應時間，建議考慮使用 DAX。DAX 也能顯著縮短存取常用資料的回應時間。
- 讀取密集型工作負載
  - 讀取密集型應用程式 – 對於讀寫比高的應用程式 (例如 10:1 或以上)，DAX 可帶來更多快取命中並減少過時資料。這可減少對資料表的讀取次數。若應用程式以寫入為主，為避免從快取讀取過時資料，請確保為快取設定較低的 使用 DynamoDB 的存留時間 (TTL) 功能。
  - 快取常見查詢 – 若應用程式頻繁讀取相同資料 (如電子商務平台上的熱門商品)，DAX 可直接從快取回應這些請求。
- 高載流量模式
  - 平滑資料表擴展 – DAX 有助於緩解突發流量高峰的影響。DAX 提供緩衝機制，可使 DynamoDB 資料表容量平順擴展，降低讀取限流風險。
  - 每項資料的較高讀取輸送量 – DynamoDB 會為每項資料配置獨立分割區。不過，當分割區達到 3,000 個讀取容量單位 (RCU) 時，會開始對項目的讀取進行限流。DAX 可讓單一項目的讀取擴展至超過 3,000 個 RCU。
- 成本最佳化
  - 降低 DynamoDB 成本 – 從 DAX 讀取可減少送往 DynamoDB 資料表的讀取請求，直接降低成本。當快取命中率高時，資料表讀取成本的節省可能超過 DAX 叢集成本，實現整體成本降低。
- 資料一致性需求
  - 最終一致性 – DAX 支援最終一致讀取。因此，DAX 適用於對即時一致性要求不高的使用案例。
  - 寫入快取 – 對 DAX 進行的寫入為寫入通過方式。當 DAX 確認項目已寫入 DynamoDB 後，會將該版本保留於項目快取中。此寫入通過機制有助於維持快取與資料庫間更高的資料一致性，但會消耗額外的 DAX 叢集資源。

何時不使用 DAX

雖然 DAX 功能強大，但並非適用於所有情境。以下列出不建議將 DAX 與 DynamoDB 整合的典型情境：

- 寫入密集型工作負載 – DAX 主要優勢在於加速讀取，但寫入會消耗比讀取更多的 DAX 資源。若應用程式以寫入為主，DAX 帶來的效益可能有限。
- 不常讀取資料 – 若應用程式不常存取資料，或需處理大量冷資料 (很少重複使用的資料)，可能會導致較低的 cache hit ratio。在此情況下，維護快取的額外負擔可能不足以抵銷效能提升的效益。
- 大量讀取或寫入 – 如果您的應用程式執行的大量寫入操作多於個別寫入，應在 DAX 外部直接進行寫入。另外，針對大量讀取，應直接對 DynamoDB 資料表執行完整資料表掃描。

- 高度一致性或交易需求 – DAX 會將高度一致性讀取及 TransactGetItems 呼叫傳送至 DynamoDB 資料表。這些讀取應繞過 DAX 叢集，以避免佔用叢集資源。以此方式讀取的項目不會被快取，因此將這類項目經由 DAX 路由沒有任何意義。
- 效能需求適中的簡單應用程式 – 若應用程式能容忍直接連線至 DynamoDB 所造成的延遲，則無需額外導入 DAX 的複雜性與成本。DynamoDB 本身可處理高輸送量，並提供個位數毫秒級效能。
- 超出金鑰值存取的複雜查詢需求 – DAX 已針對金鑰值存取模式進行最佳化。若應用程式需要具備複雜篩選的進階查詢功能，例如查詢或掃描操作，則 DAX 的快取效益可能有限。

  在此情況下，建議改用 Amazon ElastiCache (Redis OSS) 作為替代方案。ElastiCache (Redis OSS) 支援進階資料結構，例如清單、集合與雜湊。亦提供 pub/sub、地理空間索引及指令碼等功能。
- 合規需求 – DAX 目前尚未具備與 DynamoDB 相同層級的合規認證。例如，DAX 尚未通過 SOC 認證。

設定 DAX 用戶端

DAX 叢集為執行個體型叢集，可透過多種 DAX SDK 存取。各 SDK 提供可設定選項 (如 requestTimeout 與連線)，以滿足特定應用程式需求。

設定 DAX 用戶端時，需重點考量應用程式規模，特別是用戶端執行個體與 DAX 伺服器執行個體 (最多 11 個) 的比例。大規模用戶端執行個體機群可能產生大量連線至 DAX 伺服器執行個體，導致伺服器負載過重。本指南說明 DAX 用戶端設定的最佳實務。

最佳實務

1. 用戶端執行個體 – 實作單一執行個體模式，以確保請求間能重複使用。如需實作的詳細資訊，請參閱 the section called “步驟 4：執行範例應用程式”。
2. 請求逾時 – 雖然應用程式通常會設定較低的逾時值以降低上游系統延遲，但逾時過低可能造成問題。當 DAX 伺服器出現暫時性延遲高峰時，逾時設定過低可能導致頻繁重新連線至伺服器執行個體。當發生逾時時，DAX 用戶端會終止現有伺服器節點的連線，並建立新的連線。由於建立連線耗費大量資源，過多的連續連線可能導致 DAX 伺服器超載。我們建議下列作法：
   - 保持預設的請求逾時設定。
   - 若必須使用較短逾時，請建立逾時值較低的個別應用程式執行緒，並實作具指數退避的重試機制。
3. 連線逾時 – 對大多數應用程式，建議保持預設連線逾時設定。
4. 並行連線 – 某些 SDK (例如 JavaV2) 允許調整與 DAX 伺服器的並行連線數。關鍵考量事項：

- DAX 伺服器執行個體最多可處理 40,000 個並行連線。
- 預設設定適用於多數使用案例。
- 大型用戶端執行個體若搭配高並行連線，可能導致伺服器超載。
- 降低並行連線數可減少伺服器超載風險。
- 效能計算範例：
  - 假設請求延遲為 1ms，則每個連線理論上可處理每秒 1,000 個請求。
  - 在 3 節點叢集中，單一連線至所有節點的用戶端執行個體每秒可處理 3,000 個請求。
  - 以 10 個連線時，用戶端每秒約可處理 30,000 個請求。

    建議：先以較低的並行連線設定起始，並以效能測試驗證其在預期生產工作負載下的表現。

設定 DAX 叢集

DAX 叢集為受管型叢集，但仍可依應用程式需求調整其組態。由於 DAX 與 DynamoDB API 操作緊密整合，在整合應用程式時應考量以下面向。

本節內容

- DAX 定價
- 項目快取與查詢快取
- 選擇快取的 TTL 設定
- 使用 DAX 叢集以快取多個資料表
- DAX 與 DynamoDB 全域資料表中的資料複寫
- DAX 區域可用性
- DAX 快取行為

DAX 定價

叢集成本取決於其已佈建節點的數量與大小。每個節點在叢集中執行的每小時均會產生費用。如需詳細資訊，請參閱 Amazon DynamoDB 定價。

快取命中不會產生 DynamoDB 成本，但仍會佔用 DAX 叢集資源。快取未命中會產生 DynamoDB 讀取成本，並消耗 DAX 資源。寫入操作會產生 DynamoDB 寫入成本，且會佔用 DAX 叢集資源以代理該寫入。

項目快取與查詢快取

DAX 維護項目快取與查詢快取。了解這些快取的差異有助於判斷它們為應用程式提供的效能與一致性特性。

快取特性	項目快取	查詢快取
用途	儲存 GetItem 與 BatchGetItem API 操作的結果。	儲存查詢和掃描 API 操作的結果。這些操作可依查詢條件傳回多個項目，而非特定項目鍵值。
存取類型	使用鍵值型存取。 當應用程式使用 `GetItem` 或 `BatchGetItem` 請求資料時，DAX 會先以請求項目的主索引鍵檢查項目快取。若項目已快取且未過期，DAX 會立即回傳結果，無需存取 DynamoDB 資料表。	使用參數式存取。 DAX 會快取 `Query` 與 `Scan` API 操作的結果集。DAX 會以快取中的結果回應具相同參數的後續請求，包括相同查詢條件、資料表與索引。這可顯著降低回應時間與 DynamoDB 讀取輸送量的耗用。
快取失效	在下列情況下，DAX 會自動將更新後的項目複寫至 DAX 叢集中各節點的項目快取。 • 透過快取執行項目更新。 • 從資料表讀取更新後的項目版本。	查詢快取的失效管理比項目快取更具挑戰。項目更新可能無法直接對應至已快取的查詢或掃描。必須謹慎調整查詢快取的 TTL，以維持資料一致性。透過 DAX 或基礎資料表進行的寫入，不會反映於查詢快取中，直到 TTL 過期並由 DAX 對 DynamoDB 重新執行查詢。
全域次要索引	由於本機次要索引與全域次要索引不支援 `GetItem` API 操作，因此項目快取僅快取基礎資料表的讀取。	查詢快取會同時快取針對資料表與索引的查詢。

選擇快取的 TTL 設定

TTL 決定資料在變為過時前可儲存在快取中的時間長度。期間結束後，資料會於下次請求時自動更新。為 DAX 快取選擇合適的 TTL 設定需在應用程式效能最佳化與資料一致性間取得平衡。由於沒有適用於所有應用程式的通用 TTL 設定，最佳 TTL 應依應用程式的特性與需求而定。建議依本指南從保守的 TTL 設定起步。接著依應用程式效能資料與洞察逐步調整 TTL 設定。

DAX 會為項目快取維護一個最近最少使用 (LRU) 清單。LRU 清單會追蹤項目首次寫入或最近一次從快取讀取的時間。當 DAX 節點記憶體已滿時，DAX 會移出較舊項目 (無論是否過期)，以釋放空間存放新項目。LRU 演算法一律啟用，且無法由使用者設定。

若要設定適用於應用程式的 TTL 持續時間，請考量以下重點：

了解資料存取模式

- 大量讀取工作負載 – 對於讀取頻繁且資料更新較少的應用程式，建議設定較長的 TTL 持續時間，以降低快取未命中的次數。較長的 TTL 持續時間亦可減少對底層 DynamoDB 資料表的存取需求。
- 大量寫入工作負載 – 若應用程式頻繁更新且未經 DAX 寫入，應設定較短的 TTL 持續時間，以維持快取與資料庫一致性。較短的 TTL 持續時間亦可降低提供過時資料的風險。

評估應用程式效能需求

- 延遲敏感度 – 若應用程式重視延遲低於資料即時性，應設定較長的 TTL 持續時間。較長的 TTL 持續時間可最大化快取命中率，進而降低平均讀取延遲。
- 輸送量和可擴展性 – 較長的 TTL 持續時間可降低 DynamoDB 資料表負載，提升輸送量與可擴展性。然而，仍須在即時資料需求與上述因素之間取得平衡。

分析快取移出與記憶體使用量

- 快取記憶體限制 – 監控 DAX 叢集的記憶體使用量。較長的 TTL 持續時間可使快取儲存更多資料，可能導致達到記憶體上限並觸發 LRU 式移出。

使用監控指標調整 TTL

定期檢閱指標，例如快取命中與未命中次數，以及 CPU 與記憶體使用率。依據這些指標調整 TTL 設定，以取得效能與資料即時性間的最佳平衡。若快取未命中率高且記憶體使用率低，應延長 TTL 持續時間以提升快取命中率。

考量業務需求與合規要求

資料保留政策可能規定敏感或個人資訊可設定的 TTL 最長時限。

TTL 設為零時的快取行為

當 TTL 設為 0 時，項目快取與查詢快取會呈現以下行為：

- 項目快取 – 僅於發生 LRU 移出或寫入操作時，快取中的項目才會更新。
- 查詢快取 – 查詢回應不會被快取。

使用 DAX 叢集以快取多個資料表

若工作負載包含多個不需個別快取的小型 DynamoDB 資料表，可使用單一 DAX 叢集來快取這些資料表的請求。這讓 DAX 的使用更具彈性與效率，特別適用於需存取多個資料表並要求高效能讀取的應用程式。

與 DynamoDB 的資料平面 API 類似，DAX 請求需指定資料表名稱。若在同一個 DAX 叢集中使用多個資料表，無需額外設定。但必須確保叢集的安全權限允許存取所有快取資料表。

使用 DAX 搭配多個資料表時的考量事項

使用 DAX 搭配多個 DynamoDB 資料表時，應考量以下要點：

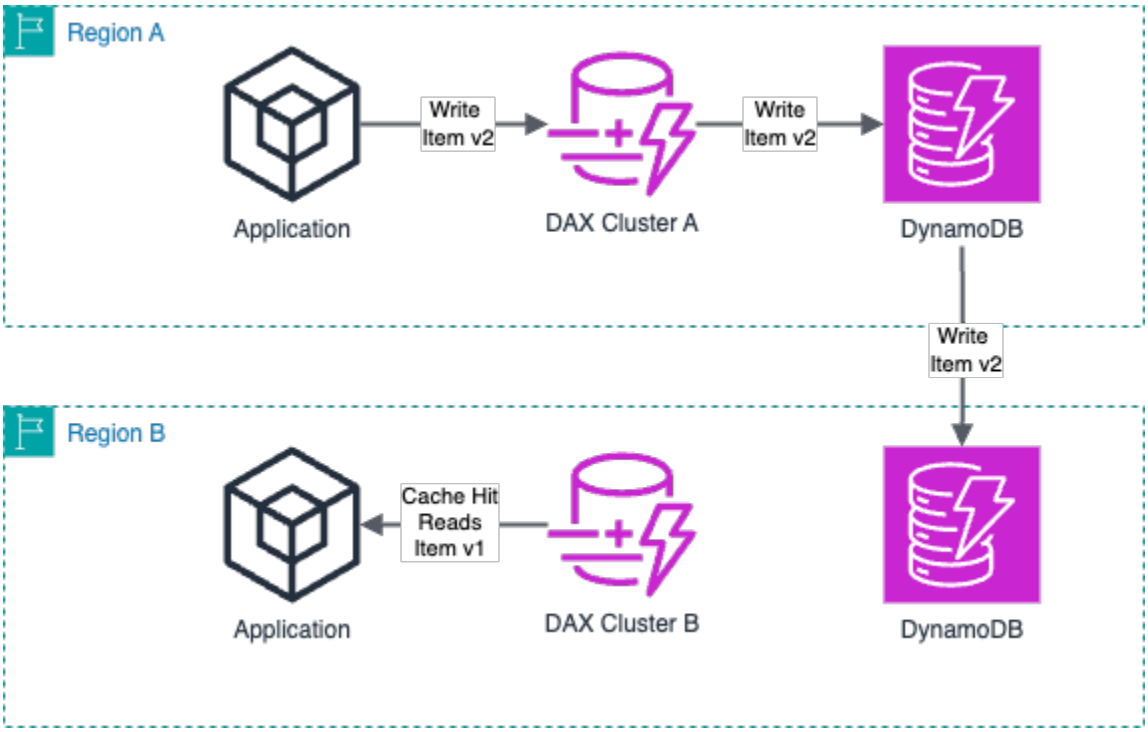
- 記憶體管理 – 使用 DAX 搭配多個資料表時，應評估工作資料集的總大小。資料集內所有資料表將共用所選節點類型的記憶體空間。
- 資源配置 – DAX 叢集的資源會由所有快取資料表共同使用。不過，高流量的資料表可能會迫使相鄰的小型資料表資料被移出。
- 規模經濟 – 將較小資源整合成較大的 DAX 叢集，以平衡流量並維持穩定模式。就 DAX 叢集的總讀取資源而言，配置三個以上節點較具成本效益。這同時能提升叢集中所有快取資料表的可用性。

DAX 與 DynamoDB 全域資料表中的資料複寫

DAX 是區域型服務，因此叢集僅能感知其 AWS 區域內的流量。全域資料表在從其他區域複寫資料時，會繞過快取進行寫入。

較長的 TTL 持續時間可能使過時資料在次要區域保留時間長於主要區域。這可能導致次要區域的本機快取發生遺漏。

下圖顯示來源區域 A 內全域資料表層級的資料複寫。區域 B 的 DAX 叢集不會立即察覺來源區域 A 新複寫的資料。



DAX 區域可用性

並非所有支援 DynamoDB 資料表的區域都支援 DAX 叢集部署。若應用程式需透過 DAX 以達低讀取延遲，請先檢閱支援 DAX 的區域清單。接著，為 DynamoDB 資料表選擇適當的區域。

DAX 快取行為

DAX 會執行中繼資料快取與負快取。了解這些快取行為有助於您有效管理快取項目及負快取項目的屬性中繼資料。

- 中繼資料快取 – DAX 叢集會無限期保留快取項目屬性名稱的中繼資料。即使項目過期或被從快取中移出，該中繼資料仍會保留。

  若應用程式使用無限制數量的屬性名稱，長期下來可能導致 DAX 叢集記憶體耗盡。此限制僅適用於頂層屬性名稱，不適用於巢狀屬性名稱。無限制屬性名稱的例子包括時間戳記、UUID 和工作階段 ID。雖然您可以將時間戳記和工作階段 ID 用作屬性值，但建議使用較短且可預測的屬性名稱。
- 負快取 – 當發生快取未命中，且從 DynamoDB 資料表讀取不到相符項目時，DAX 會在對應的項目或查詢快取中新增負快取項目。此項目會保留，直到快取 TTL 到期或發生寫入通過。DAX 會在後續請求中持續傳回此負快取項目。

若負快取行為不符合應用程式模式，當 DAX 傳回空結果時，直接讀取 DynamoDB 資料表。另建議設定較短的 TTL 快取時間，以避免快取中長期維持空結果，並提升與資料表的一致性。

調整 DAX 叢集大小

DAX 叢集的總容量與可用性取決於節點類型與數量。叢集中節點數越多，讀取容量越高，但寫入容量不變。較大型節點類型 (最高至 r5.8xlarge) 可處理更多寫入，但節點數過少在節點故障時會降低可用性。如需關於 DAX 叢集規模調整的詳細資訊，請參閱 DAX 叢集調整大小指南。

下列章節將說明如何在建立可擴展且具成本效益的叢集時，平衡節點類型與數量等各項規模因素。

本節內容

- 可用性規劃
- 寫入輸送量規劃
- 讀取輸送量規劃
- 資料集大小規劃
- 計算叢集的大致容量需求
- 依節點類型估算叢集輸送容量
- 在 DAX 叢集中擴展寫入容量

可用性規劃

在規劃 DAX 叢集規模時，應先確定目標可用性。對於 DAX 等叢集服務，可用性取決於叢集節點的總數。單節點叢集不具容錯能力，其可用性僅等同於單一節點。在 10 節點叢集中，單一節點失效對整體容量的影響最小。此節點失效不會直接影響可用性，因為其餘節點仍可處理讀取請求。為恢復寫入，DAX 會迅速指派新的主節點。

DAX 以 VPC 為基礎運作。DAX 使用子網路群組來判定可在哪些可用區域中執行節點，並決定要從哪些子網路配置 IP 位址。對於生產環境工作負載，強烈建議在不同可用區域中部署至少三個節點的 DAX。這可確保即使單一節點或可用區域故障，叢集仍保有多個節點可處理請求。叢集最多可包含 11 個節點，其中 1 個為主節點，10 個為讀取副本。

寫入輸送量規劃

所有 DAX 叢集皆具有用於寫入通過請求的主節點。節點類型的規模決定叢集的寫入容量。新增更多讀取副本不會提升叢集的寫入容量。因此，應在建立叢集時評估寫入容量，因為節點類型日後無法變更。

若應用程式需透過 DAX 寫入通過以更新項目快取，應預留更多叢集資源以支援寫入。對 DAX 的寫入操作耗用的資源約為快取命中讀取的 25 倍。因此可能需使用比唯讀叢集更大型的節點類型。

如需判斷「寫入通過」或「寫入繞過」哪種機制較適合應用程式的進一步指南，請參閱 寫入策略。

讀取輸送量規劃

DAX 叢集的讀取容量取決於工作負載的快取命中率。由於 DAX 在快取未命中時需從 DynamoDB 讀取資料，因此會耗用約為快取命中讀取 10 倍的叢集資源。若要提升快取命中率，請提高快取的 TTL 設定，以定義項目在快取中的儲存時間。但較長的 TTL 時間會提高讀取舊項目版本的機率，除非更新是透過 DAX 寫入通過的。

為確保叢集具足夠讀取容量，請依 水平擴展叢集 所述進行水平擴展。新增節點會將讀取副本加入資源集區，而移除節點則會降低讀取容量。在選擇叢集節點數量與大小時，應同時考量所需的最小與最大讀取容量。若無法以較小節點類型水平擴展叢集以滿足讀取需求，應改用較大型節點類型。

資料集大小規劃

每種可用節點類型皆具固定記憶體容量，供 DAX 用於快取資料。若節點類型過小，應用程式的工作資料集無法完整載入記憶體，將導致快取未命中。由於較大型節點支援更大的快取容量，請選用比預估需快取資料集更大的節點類型。較大的快取同時有助提升快取命中率。

若快取的項目重複讀取次數較少，快取效益可能遞減。計算經常存取項目的記憶體需求，並確保快取容量足以容納該資料集。

計算叢集的大致容量需求

您可以預估工作負載的總容量需求，以協助選擇適當的叢集節點規格與數量。若要進行此估算，請計算變數標準化每秒請求數 (Normalized RPS)。此變數代表應用程式要求 DAX 叢集支援的總工作量，包含快取命中、快取遺漏及寫入作業。若要計算標準化 RPS，請使用以下輸入參數：

- `ReadRPS_CacheHit` – 指定每秒導致快取命中的讀取請求數。
- `ReadRPS_CacheMiss` – 指定每秒導致快取遺漏的讀取請求數。
- `WriteRPS` – 指定每秒透過 DAX 執行的寫入請求數。
- `DaxNodeCount` – 指定 DAX 叢集中的節點數。
- `Size` – 指定寫入或讀取項目的大小 (以 KB 為單位)，並四捨五入至最接近整數 KB。
- `10x_ReadMissFactor` – 表示數值 10。發生快取遺漏時，DAX 會耗用比快取命中多約 10 倍的資源。
- `25x_WriteFactor` – 表示數值 25，因為 DAX 寫入作業耗用的資源約為快取命中的 25 倍。

請使用下列公式計算標準化 RPS。

```
Normalized RPS = (ReadRPS_CacheHit * Size) + (ReadRPS_CacheMiss * Size *
 10x_ReadMissFactor) + (WriteRequestRate * 25x_WriteFactor * Size * DaxNodeCount)
```

例如，請參考以下輸入參數值：

- ReadRPS_CacheHit = 50,000
- ReadRPS_CacheMiss = 1,000
- ReadMissFactor = 1
- Size = 2 KB
- WriteRPS = 100
- WriteFactor = 1
- DaxNodeCount = 3

將這些數值代入公式後，可得出標準化 RPS，計算如下。

```
Normalized RPS = (50,000 Cache Hits/Sec * 2KB) + (1,000 Cache Misses/Sec * 2KB * 10) +
 (100 Writes/Sec * 25 * 2KB * 3)
```

在此範例中，計算所得的標準化 RPS 為 135,000。不過，此標準化 RPS 值未考慮叢集使用率低於 100% 或節點遺失的情況。建議預留額外容量。為此，請判斷以下兩項倍增因子中的較大值：目標使用率或節點遺失容錯能力。接著，將標準化 RPS 乘以較大因子，以取得目標每秒請求數 (Target RPS)。

- 目標使用率

  因為效能下降會導致快取遺漏增加，因此不建議讓 DAX 叢集以 100% 使用率運行。理想情況下，叢集使用率應維持在 70% 或以下。為達此目標，請將標準化 RPS 乘以 1.43。

- 節點遺失容錯能力

  當節點發生故障時，應用程式必須能在其餘節點間分散請求。為確保節點使用率低於 100%，請選擇足夠大的節點類型，以便在故障節點恢復上線前吸收額外流量。節點較少的叢集在單一節點故障時，每個節點必須能承受更大的流量成長。

  若主節點發生故障，DAX 會自動容錯移轉至讀取複本，並將其指定為新的主節點。若複本節點發生故障，DAX 叢集中的其他節點仍可持續處理請求，直到故障節點復原。

例如，3 節點的 DAX 叢集若有一個節點故障，其餘兩個節點需額外承擔 50% 的容量。此情況需採用乘數係數 1.5。相對地，11 節點叢集若有節點故障，其餘節點需增加 10% 的容量，或採用乘數係數 1.1。

可使用下列公式計算目標 RPS。

```
Target RPS = Normalized RPS * CEILING(TargetUtilization, NodeLossTolerance)
```

例如，若要計算目標 RPS，請考慮下列數值：

- `Normalized RPS` = 135,000
- `TargetUtilization` = 1.43

  由於目標叢集使用率上限為 70%，因此將 `TargetUtilization` 設為 1.43。
- `NodeLossTolerance` = 1.5

  假設使用 3 節點叢集，因此將 `NodeLossTolerance` 設為 50% 的容量。

將這些數值代入公式，即可計算出目標 RPS，如下所示。

```
Target RPS = 135,000 * CEILING(1.43, 1.5)
```

在此範例中，由於 `NodeLossTolerance` 的值大於 `TargetUtilization`，因此以 `NodeLossTolerance` 計算目標 RPS。計算結果為目標 RPS = 202,500，即 DAX 叢集需支援的總容量。若要判斷叢集中所需節點數，請將目標 RPS 對應至下表中的適當欄位。以 202,500 的目標 RPS 為例，需使用三個節點的 dax.r5.large 節點類型。

依節點類型估算叢集輸送容量

可使用 Target RPS formula 估算不同節點類型的叢集容量。下表顯示 1、3、5 和 11 節點叢集的近似容量。這些容量估值無法取代使用自有資料與請求模式進行 DAX 負載測試的必要性。此外，這些容量不包含 t 類型執行個體，因其不具固定 CPU 容量。下表中所有數值的單位均為標準化 RPS。

節點類型 (記憶體)	1 個節點	3 個節點	5 個節點	11 個節點
dax.r5.24xlarge (768GB)	100 萬	300 萬	500 萬	1,100 萬
dax.r5.16xlarge (512GB)	100 萬	300 萬	500 萬	1,100 萬
dax.r5.12xlarge (384GB)	100 萬	300 萬	500 萬	1,100 萬
dax.r5.8xlarge (256GB)	100 萬	300 萬	500 萬	1,100 萬
dax.r5.4xlarge (128GB)	60 萬	180 萬	300 萬	660 萬
dax.r5.2xlarge (64GB)	30 萬	90 萬	150 萬	330 萬
dax.r5.xlarge (32GB)	15 萬	45 萬	75 萬	165 萬
dax.r5.large (16GB)	7.5 萬	22.5 萬	37.5 萬	82.5 萬

由於每個節點的 NPS (每秒網路操作) 上限為 100 萬，因此類型為 dax.r5.8xlarge 或更大的節點不會提供額外叢集容量。大於 8xlarge 的節點類型可能不會增加叢集的總輸送容量。不過，這類節點類型有助於在記憶體中容納更大的工作資料集。

在 DAX 叢集中擴展寫入容量

每次寫入 DAX 都會在每個節點上消耗 25 個標準化請求。由於每個節點的 RPS 上限為 100 萬，因此 DAX 叢集的寫入上限為每秒 40,000 次，不包含讀取用量。

若快取中的寫入需求超過每秒 40,000 次，則必須使用多個獨立 DAX 叢集，並將寫入資料碎片分散於各叢集。類似 DynamoDB，可對寫入快取的資料分割區索引鍵進行雜湊處理。接著，使用取模運算判定應將資料寫入哪個碎片。

下列範例計算輸入字串的雜湊值。接著，以 10 為基準計算該雜湊值的取模結果。

```
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}.")
```

部署叢集

建立新 DAX 叢集時，需進行超出 DynamoDB 組態範圍的額外設定。這些設定主要與網路相關，因為 DAX 架構於 Amazon VPC 之上。這讓使用者能完全掌控虛擬網路環境，包括資源配置、連線性與安全性。本節說明叢集建立過程中相關設定的最佳實務。

如需選擇叢集節點的相關資訊，請參閱 調整 DAX 叢集大小。

本節內容

- 設定網路
- 設定安全性
- 參數群組
- Maintenance window (維護時段)

設定網路

DAX 使用子網路群組來判定可在哪些可用區域執行節點，以及要從哪些子網路指派 IP 位址。為了將應用程式與 DAX 之間的延遲降至最低，應用程式伺服器與 DAX 叢集應位於相同的子網路與可用區域中。

建議將 DAX 節點分散部署於多個可用區域。預設的自動配置選項會自動完成此設定。

如需設定 VPC 的最佳實務，請參閱 Amazon VPC 使用者指南中的開始使用 Amazon VPC。

設定安全性

本節說明應為使用 DAX 的應用程式實施的安全措施。本節亦簡要說明 DAX 提供的資料加密支援。

IAM

DAX 和 DynamoDB 各自具有獨立的存取控制機制。DAX 需要 IAM 角色才能存取 DynamoDB 資料表。此角色應遵循最低權限原則，只授予特定資料表與 DynamoDB 操作 (如 GetItem、PutItem) 的存取權。如需 DAX 所提供的存取控制機制詳細資訊，請參閱 DAX 存取控制。

加密

建立 DAX 叢集時可設定靜態加密與傳輸中加密。這些加密選項預設為啟用。建議除非業務需求另有要求，否則保留預設加密設定。如需詳細資訊，請參閱DAX 靜態加密及DAX 傳輸中加密。

參數群組

DAX 會在叢集的每個節點上套用一組稱為參數群組的組態。可在建立叢集後變更此組態。

DAX 參數群組包含項目快取與查詢快取的 TTL 設定。預設 TTL 持續時間為 5 分鐘。可將 TTL 持續時間覆寫為大於或等於 1 毫秒的任意整數值。

當參數群組正供執行中的 DAX 執行個體使用時，無法修改。可在 DAX 叢集停機時間變更參數群組值。

Maintenance window (維護時段)

為了執行節點的軟體升級與修補，系統會為 DAX 叢集設定每週的維護時段。在此維護時段內，DAX 會對節點執行循環更新。多節點叢集在更新期間可維持可用性，但叢集容量會暫時降低，直到節點復原。若組織具有可預測的低負載時段，建議手動將維護時段設為該時間。

管理叢集作業

DAX 會自動管理叢集的維護與健康狀態。不過，仍需提供操作參數，以便依使用模式進行叢集的水平或垂直擴展。本節說明擴展 DAX 叢集的建議流程。

本節內容

- 水平擴展叢集
- 垂直擴展叢集

水平擴展叢集

擴展 DAX 叢集需調整容量以符合輸送量需求。此調整可在叢集運行期間，透過增加或減少節點 (副本) 數量來完成。此程序稱為水平擴展，可在需求高時分散工作負載，需求低時整併節點。

您可在 AWS CLI中使用 decrease-replication-factor 或 increase-replication-factor 指令，執行 DAX 叢集的水平擴展與縮減。

增加複寫係數 (橫向擴充)

增加 DAX 叢集的複寫係數可新增更多節點至叢集。以下範例顯示 increase-replication-factor 指令的用法。

```
aws dax increase-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- 在此指令中，cluster-name 參數指定叢集名稱。例如：*yourClusterName*。
- new-replication-factor 參數指定擴展後叢集中的節點總數。此數包含主節點與副本節點。例如，若叢集目前有 3 個節點且要再新增 2 個節點，請將 new-replication-factor 的值設為 5。

減少複寫係數 (向內縮減)

減少 DAX 叢集的複寫係數會移除部分節點。移除節點可在低需求期間降低成本。以下範例顯示 decrease-replication-factor 指令的用法。

```
aws dax decrease-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- 在此指令中，cluster-name 參數指定叢集名稱。例如：*yourClusterName*。
- new-replication-factor 參數指定擴展後叢集的節點總數 (縮減後)。此數值須低於目前複寫係數，且包含主節點。例如，若叢集有 5 個節點且要移除 2 個節點，請將 new-replication-factor 的值設為 3。

水平擴展注意事項

規劃水平擴展時，請考量以下事項：

- 主節點 – DAX 叢集包含主節點。複寫係數計入此主節點。例如，複寫係數為 3 時，代表 1 個主節點與 2 個副本節點。
- 可用性 – 新增或移除 DAX 節點會影響叢集的可用性與容錯能力。增加節點可提升可用性，但同時提高成本。
- 資料移轉 – 當複寫係數增加時，DAX 會自動在新節點間分配資料。當新節點開始處理流量時，其快取已預先載入。然而，在資料移轉過程中，可能會暫時影響系統效能。

請務必在擴展過程中及完成後密切監控 DAX 叢集的運作，以確保其效能符合預期，並依需求進行調整。

垂直擴展叢集

若要垂直擴展現有叢集的節點規格，需建立新叢集，並將應用程式流量切換至新叢集。移轉至具不同節點的新叢集需經多個步驟，以確保轉換順暢，並將對應用程式效能與可用性的影響降至最低。

若要建立新叢集以垂直擴展節點規格，請考慮以下事項：

- 存取目前設定 – 檢閱現有 DAX 叢集的效能指標，以評估所需的新節點規格與數量。根據此資訊作為依據，以定義叢集規模。如需相關資訊，請參閱調整 DAX 叢集大小。
- 設定新 DAX 叢集 – 依您所選的節點類型與數量建立新叢集。可沿用參數群組中的現有組態設定，除非需要修改。
- 同步資料 – 由於 DAX 為 DynamoDB 的快取層，因此無需直接移轉資料。但在將流量導向新 DAX 叢集之前，其記憶體中不會載入任何工作資料集。
- 更新應用程式組態 – 將應用程式組態更新為指向新的 DAX 叢集端點。視應用程式組態而定，您可能需要修改程式碼或更新環境變數。

  為降低切換至新叢集時的影響，請先從部分應用程式機群傳送 Canary 測試流量至新叢集。您可透過逐步發布應用程式更新，或在 DAX 端點前配置權重式路由 DNS 項目，以達成此操作。
- 監控與最佳化 – 切換至新 DAX 叢集後，請密切監控其效能指標與日誌，以偵測潛在問題。請根據最新的工作負載模式，隨時調整節點數量。

  在新叢集完成工作資料集快取前，可能會出現較高的快取未命中率與延遲。
- 停用舊叢集 – 確認新叢集運作正常後，請安全停用舊 DAX 叢集，以避免額外成本。

監控 DAX

可監控主要指標 (如快取命中率)，以確保 DAX 叢集維持最佳效能、協助問題診斷，並判斷何時需擴展叢集。定期檢查關鍵指標，可協助您依工作負載需求調整叢集規模，以維持效能、穩定性與成本效益。如需 DAX 監控的詳細資訊，請參閱 生產監控。

下列列出您應監控的主要指標：

- 快取命中率 – 顯示 DAX 提供快取資料的效率，並減少對底層 DynamoDB 資料表的存取需求。快取未命中次數偏低表示叢集快取效率良好。但若快取命中次數偏低，可能表示需重新檢視快取 TTL 設定，或工作負載不適合快取。

  使用 Amazon CloudWatch 計算 DAX 叢集的快取命中率。比較 `ItemCacheHits`、`ItemCacheMisses`、`QueryCacheHits` 及 `QueryCacheMisses` 等指標以取得此比率。下列公式說明快取命中率的計算方式。使用此公式計算時，請以快取命中次數除以快取命中與未命中次數的總和。

  ```
  Cache hit ratio = Cache hits / (Cache hits + Cache misses)
  ```

  快取命中率介於 0 與 1 之間，並以百分比顯示。百分比越高，表示整體快取使用率越佳。

- ErrorRequestCount – 統計節點或叢集回報使用者錯誤的請求數，其中 `ErrorRequestCount` 包含節點或叢集被限流的請求。監控使用者錯誤有助識別應用程式中的擴展設定錯誤，或熱門項目與分割區的存取模式。
- 操作延遲 – 監控進出 DAX 叢集的讀寫操作延遲，有助識別潛在效能瓶頸。延遲上升可能表示 DAX 叢集組態或網路存在問題，或需進行擴展。
- 網路耗用量 – 持續監控 `NetworkBytesIn` 與 `NetworkBytesOut` 等指標，以掌握 DAX 叢集的網路流量。網路輸送量異常增加可能表示用戶端請求量上升，或查詢模式效率不佳導致資料傳輸增加。

  監控網路使用量可協助您有效管理 DAX 叢集成本。同時可確保網路不成為叢集效能瓶頸。

- 移出率 – 顯示快取項目被移除以騰出空間給新項目的頻率。若移出率隨時間上升，可能表示快取容量不足，或快取策略不佳。

  在 CloudWatch 中監控 `EvictedSize` 指標，以判斷快取容量是否符合工作負載需求。若總移出量持續增加，您可能需要向上擴展 DAX 叢集以支援更大快取容量。

- CPU 使用率 – 表示節點或叢集的 CPU 使用百分比。這是任何資料庫或快取系統中需重點監控的關鍵指標。高 CPU 使用率可能表示 DAX 叢集已超載，需要擴展以因應增加的需求。

監控 DAX 叢集的 CPUUtilization 指標。若 CPU 使用率持續接近或超過 70–80%，請依下節說明考慮向上擴展 DAX 叢集。

若傳送至 DAX 的請求數超出節點容量，DAX 會限制其接受額外請求的速率。DAX 會透過傳回 ThrottlingException 來達成此行為。DAX 會持續評估叢集的 CPU 使用率，以判斷可處理的請求量，同時維持叢集的穩定運作狀態。

您可以監控 DAX 發布至 CloudWatch 的 ThrottledRequestCount 指標。如果您每隔一段時間就會看到這些例外狀況，請考慮擴展您的叢集。

使用監控資料進行 DAX 叢集擴展

您可透過監控 DAX 叢集的效能指標，判斷是否需向上擴展或縮減規模。

- 向上或橫向擴展 – 若 DAX 叢集出現高 CPU 使用率、快取命中率偏低 (在最佳化快取策略後)，或操作延遲過高，建議向上擴展叢集。新增節點 (亦稱橫向擴充) 有助於更均衡地分散負載。若工作負載的每秒寫入量持續增加，您可能需要選用更高效能的節點 (向上擴展)。
- 縮減規模 – 若 CPU 使用率與操作延遲持續低於設定閾值，可能表示資源配置過度。在此情況下，請縮減節點規模以降低成本。在低使用率期間，您可將節點數量減至 1，但無法完全停用叢集。

將 DynamoDB 與其他 AWS 服務搭配使用

Amazon DynamoDB 與其他 AWS 服務整合，可讓您自動重複任務或建置跨越多個 服務的應用程式。

主題

- 使用 Amazon Cognito for DynamoDB 設定 AWS 登入資料
- 與 Amazon Redshift 整合
- 在 Amazon EMR 上使用 Apache Hive 處理 DynamoDB 資料
- 將 DynamoDB 與 Amazon S3 整合
- DynamoDB 與 Amazon SageMaker 資料湖倉的零 ETL 整合
- DynamoDB 與 Amazon OpenSearch Service 的零 ETL 整合
- 整合 DynamoDB 與 Amazon EventBridge
- 將 DynamoDB 與 Amazon Managed Streaming for Apache Kafka 整合
- 與 DynamoDB 整合的最佳實務

使用 Amazon Cognito for DynamoDB 設定 AWS 登入資料

為您的 Web 和行動應用程式取得 AWS 登入資料的建議方法是使用 Amazon Cognito。Amazon Cognito 可協助您避免對檔案上的 AWS 登入資料進行硬式編碼。它使用 AWS Identity and Access Management (IAM) 角色為您的應用程式已驗證和未驗證的使用者產生臨時登入資料。

例如，若要設定您的 JavaScript 檔案以使用 Amazon Cognito 未驗證角色來存取 Amazon DynamoDB Web 服務，請遵循以下方法：

設定憑證來與 Amazon Cognito 互動

1. 建立允許未驗證身分的 Amazon Cognito 身分池。

```
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. 將下列政策複製到名為 myCognitoPolicy.json 的檔案。使用在上一個步驟取得的 IdentityPoolId，來取代身分池 ID (*us-west-2:12345678-1ab2-123a-1234-a12345ab12*)。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "us-
west-2:12345678-1ab2-123a-1234-a12345ab12"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "unauthenticated"
        }
      }
    }
  ]
}
```

3. 建立採用上述政策的 IAM 角色。如此一來，Amazon Cognito 會成為可擔任 Cognito_DynamoPoolUnauth 角色的信任實體。

```
aws iam create-role --role-name Cognito_DynamoPoolUnauth \
--assume-role-policy-document file://PathToFile/myCognitoPolicy.json --output json
```

4. 連接受管政策 (AmazonDynamoDBFullAccess)，將 DynamoDB 的完整存取權授予 Cognito_DynamoPoolUnauth 角色。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonDynamoDBFullAccess \
```

```
--role-name Cognito_DynamoPoolUnauth
```

> **Note**
>
> 或者，您也可以將精細定義存取權授予 DynamoDB。如需詳細資訊，請參閱使用 IAM 政策條件精細定義存取控制。

5. 取得並複製 IAM 角色 Amazon Resource Name (ARN)。

```
aws iam get-role --role-name Cognito_DynamoPoolUnauth --output json
```

6. 將 `Cognito_DynamoPoolUnauth` 角色新增至 `DynamoPool` 身分池。指定的格式是 `KeyName=string`，其中 `KeyName` 是 `unauthenticated`，而字串是在上一步中取得的角色 ARN。

```
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. 在您的檔案中指定 Amazon Cognito 憑證。並依照上一步的程式碼修改 `IdentityPoolId` 與 `RoleArn`。

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
IdentityPoolId: "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
RoleArn: "arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth"
});
```

您現在已可使用 Amazon Cognito 憑證，對 DynamoDB Web 服務執行您的 JavaScript 程式。如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 入門指南》中的在 Web 瀏覽器設定登入資料。

與 Amazon Redshift 整合

Amazon Redshift 是一項快速的全受管 PB 級資料倉儲服務，可讓您使用現有的商業智慧工具來高效分析所有資料，操作簡單且符合成本效益。

DynamoDB 和 Amazon Redshift 可以一起使用，以滿足應用程式或資料生態系統內不同的資料儲存和處理需求。

如需如何將 DynamoDB 與 Amazon Redshift 整合的詳細主題，請參閱下列主題。

主題

- 使用 CMK 時的跨帳戶整合考量
- 與 Amazon Redshift 的 DynamoDB 零 ETL 整合
- 使用 COPY 命令將資料從 DynamoDB 載入 Amazon Redshift

使用 CMK 時的跨帳戶整合考量

當您嘗試從 DynamoDB 整合到 Amazon Redshift 時，初始動作會從 Amazon Redshift 啟動。如果沒有適當的許可，此動作可能會導致靜默失效。下列各節詳細說明此跨帳戶整合所需的許可。

必要的 AWS KMS 政策和許可

取代下列範例中的預留位置：

- `111122223333`：託管 Amazon Redshift 的 AWS 帳戶 ID
- `444455556666`：託管 DynamoDB 的 AWS 帳戶 ID
- `REDSHIFT_ROLE_NAME`：Amazon Redshift 使用的 IAM 角色名稱
- `REGION`： AWS 區域 您的資源所在的 。
- `TABLE_NAME`：DynamoDB 資料表的名稱
- `KMS_KEY_ID`：KMS 金鑰的 ID

DynamoDB 帳戶中的 KMS 金鑰政策

下列 AWS KMS 金鑰政策可啟用 DynamoDB 和 Amazon Redshift 服務之間的跨帳戶存取。在此範例中，帳戶 444455556666 包含 DynamoDB 資料表和 AWS KMS 金鑰，而帳戶 111122223333 包含需要存取以解密資料的 Amazon Redshift 叢集。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
```

```
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow Redshift to use the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/REDSHIFT_ROLE_NAME"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Redshift 角色的 IAM 政策 (在 Amazon Redshift 帳戶中)

下列 IAM 政策允許 Amazon Redshift 服務在跨帳戶案例中存取 DynamoDB 資料表及其相關聯的 AWS KMS 加密金鑰。在此範例中，帳戶 444455556666 包含 Amazon Redshift 服務需要存取的 DynamoDB 資源和 AWS KMS 金鑰。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDynamoDBAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
```

```
                "dynamodb:Query",
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME",
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME/stream/*"
            ]
        },
        {
            "Sid": "AllowKMSAccess",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "arn:aws:kms:us-east-1:444455556666:key/KMS_KEY_ID"
        }
    ]
}
```

Amazon Redshift 角色的信任關係

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
```

```
}
```

DynamoDB 資料表政策 (如果使用資源型政策)

下列資源型政策允許帳戶 111122223333 中的 Amazon Redshift 服務存取帳戶 444455556666 中的 DynamoDB 資料表和串流。將此政策連接至您的 DynamoDB 資料表，以啟用跨帳戶存取。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRedshiftAccess",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/REDSHIFT_ROLE_NAME"
            },
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME",
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME/stream/*"
            ]
        }
    ]
}
```

重要考量

1. 確保 KMS 金鑰與您的 DynamoDB 資料表位於相同的區域。
2. KMS 金鑰必須是客戶受管金鑰 (CMK)，而不是 AWS 受管金鑰。
3. 如果您使用的是 DynamoDB 全域資料表，請設定所有相關區域的許可。
4. 考慮新增條件陳述式，以根據 VPC 端點或 IP 範圍限制存取。
5. 為了增強安全性，請考慮使用 `aws:PrincipalOrgID` 條件來限制對組織的存取。
6. 透過 CloudTrail 和 CloudWatch 指標監控 KMS 金鑰用量。

與 Amazon Redshift 的 DynamoDB 零 ETL 整合

Amazon DynamoDB 與 Amazon Redshift 的零 ETL 整合可對 DynamoDB 資料進行無縫分析，無需撰寫任何程式碼。此全受管功能會自動將 DynamoDB 資料表複寫到 Amazon Redshift 資料庫，讓使用者可以對其 DynamoDB 資料執行 SQL 查詢和分析，而無需設定複雜的 ETL 程序。整合的運作方式是從 DynamoDB 資料表將資料複寫到 Amazon Redshift 資料庫。

若要設定整合，只要將 DynamoDB 資料表指定為來源，並將 Amazon Redshift 資料庫指定為目標即可。啟用時，整合會匯出完整的 DynamoDB 資料表以填入 Amazon Redshift 資料庫。完成此初始程序所需的時間取決於 DynamoDB 資料表大小。然後，零 ETL 整合會使用 DynamoDB 增量匯出，每 15-30 分鐘將更新從 DynamoDB 增量複寫到 Amazon Redshift。這表示 Amazon Redshift 中複寫的 DynamoDB 資料會自動保持為最新。

設定完成後，使用者可以使用標準 SQL 用戶端和工具分析 Amazon Redshift 中的 DynamoDB 資料，而不會影響 DynamoDB 資料表效能。透過消除繁瑣的 ETL，此零 ETL 整合提供快速、簡單的方式，透過 Amazon Redshift 分析和機器學習功能從 DynamoDB 釋放洞見。

主題

- 與 Amazon Redshift 建立 DynamoDB 零 ETL 整合之前的先決條件
- DynamoDB 零 ETL 整合與 Amazon Redshift 搭配使用時的限制
- 建立與 Amazon Redshift 的 DynamoDB 零 ETL 整合
- 檢視與 Amazon Redshift 的 DynamoDB 零 ETL 整合
- 刪除與 Amazon Redshift 的 DynamoDB 零 ETL 整合

與 Amazon Redshift 建立 DynamoDB 零 ETL 整合之前的先決條件

1. 您必須先建立來源 DynamoDB 資料表和目標 Amazon Redshift 叢集，才能建立整合。此資訊涵蓋在 步驟 1：設定來源 DynamoDB 資料表 和 步驟 2：建立 Amazon Redshift 資料倉儲 中。
2. Amazon DynamoDB 和 Amazon Redshift 之間的零 ETL 整合需要您的來源 DynamoDB 資料表啟用時間點復原 (PITR)。
3. 對於資源型政策，零 ETL 整合需要直接連接到 DynamoDB 資料表的資源型政策。此內嵌政策授予 Amazon Redshift 服務存取資料表資料以進行複寫的許可。如需 DynamoDB 資源型政策的詳細資訊，請參閱使用 DynamoDB 資源型政策。

   如果您建立的整合是 DynamoDB 資料表和 Amazon Redshift 資料倉儲位於相同帳戶中，您可以在建立整合步驟期間使用 Fix it for me 選項，自動將所需的資源政策套用到 DynamoDB 和 Amazon Redshift。

   如果您建立 DynamoDB 資料表和 Amazon Redshift 資料倉儲位於不同 AWS 帳戶的整合，則需要在 DynamoDB 資料表上手動套用下列資源政策。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid":
 "StatementthatallowsAmazonRedshiftservicetoDescribeTableandExportTable",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": [
                "dynamodb:ExportTableToPointInTime",
                "dynamodb:DescribeTable"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
```

```
                }
            }
        },
        {
            "Sid":
 "StatementthatallowsAmazonRedshiftservicetoDescribeTableandExportTable",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "dynamodb:DescribeExport",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/table-
name/export/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
                }
            }
        }
    ]
}
```

您可能還需要在 Amazon Redshift 資料倉儲上設定資源政策。如需詳細資訊，請參閱使用 Amazon Redshift 資料 API 設定授權。

4. 身分型政策：

   a. 建立整合的使用者需要身分型政策，以授權下列動作：`GetResourcePolicy`、`PutResourcePolicy` 和 `UpdateContinuousBackups`。

> **Note**
>
> 下列政策範例會將資源顯示為 `arn:aws:redshift{-serverless}`。這是顯示 arn 可以是 `arn:aws:redshift` 或 `arn:aws:redshift-serverless` 的範例，取決於您的命名空間是 Amazon Redshift 叢集還是 Amazon Redshift Serverless 命名空間。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetResourcePolicy",
                "dynamodb:PutResourcePolicy",
                "dynamodb:UpdateContinuousBackups"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:111122223333:table/table-name"
            ]
        },
        {
            "Sid": "AllowRedshiftDescribeIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:DescribeIntegrations"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowRedshiftCreateIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateIntegration",
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
        },
        {
            "Sid": "AllowRedshiftModifyDeleteIntegration",
            "Effect": "Allow",
            "Action": [
```

```
                "redshift:ModifyIntegration",
                "redshift:DeleteIntegration"
            ],
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:integration:uuid"
        },
        {
            "Sid": "AllowRedshiftCreateInboundIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateInboundIntegration",
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:namespace:uuid"
        }
    ]
}
```

b. 負責設定目的地 Amazon Redshift 命名空間的使用者需要身分型政策，以授權下列動作：PutResourcePolicy、DeleteResourcePolicy 和 GetResourcePolicy。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "redshift:PutResourcePolicy",
                "redshift:DeleteResourcePolicy",
                "redshift:GetResourcePolicy"
            ],
            "Resource": [
                "arn:aws:redshift:us-east-1:111122223333:cluster:cluster-
name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "redshift:DescribeInboundIntegrations"
            ],
            "Resource": [
                "arn:aws:redshift:us-east-1:111122223333:cluster:cluster-
name"
```

```
            ]
        }
    ]
}
```

5. 金鑰加密許可

   如果來源 DynamoDB 資料表使用客戶自管 AWS KMS 金鑰加密，您將需要在 KMS 金鑰上新增下列政策。此政策可讓 Amazon Redshift 使用您的 KMS 金鑰從加密的資料表匯出資料。

```
{
    "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:*"
        }
    }
}
```

您也可以遵循 Amazon Redshift 管理指南中零 ETL 整合入門的步驟，以設定 Amazon Redshift 命名空間的許可。

DynamoDB 零 ETL 整合與 Amazon Redshift 搭配使用時的限制

下列一般限制適用於此整合的目前版本。這些限制在後續版本中可能會變更。

> **Note**
>
> 除了以下限制之外，也請參閱《Amazon Redshift 管理指南》中的使用零 ETL 整合與 Amazon Redshift 的考量。

- DynamoDB 資料表和 Amazon Redshift 叢集必須位於相同的區域。
- 來源 DynamoDB 資料表必須使用 Amazon 擁有或客戶管理的 AWS KMS 金鑰加密。來源 DynamoDB 資料表不支援 Amazon 受管加密。

建立與 Amazon Redshift 的 DynamoDB 零 ETL 整合

建立零 ETL 整合之前，您必須先設定來源 DynamoDB 資料表，然後設定目標 Amazon Redshift 資料倉儲。

步驟 1：設定來源 DynamoDB 資料表

若要建立與 Amazon Redshift 的零 ETL 整合，您需要在資料表上啟用時間點復原 (PITR)。如果您沒有開啟 PITR，主控台可以在整合設定過程中為您修正此問題。如需如何啟用 PITR 的詳細資訊，請參閱時間點復原。

步驟 2：建立 Amazon Redshift 資料倉儲

如果您還沒有 Amazon Redshift 資料倉儲，您可以建立一個。若要建立 Amazon Redshift Serverless 工作群組，請參閱建立包含命名空間的工作群組。若要建立 Amazon Redshift 叢集，請參閱建立叢集。

目標 Amazon Redshift 工作群組或叢集必須開啟 enable_case_sensitive_identifier 參數，才能成功整合。如需啟用區分大小寫的詳細資訊，請參閱《Amazon Redshift 管理指南》中的為您的資料倉儲開啟區分大小寫。

Amazon Redshift 工作群組或叢集設定完成後，您需要設定資料倉儲。如需詳細資訊，請參閱《Amazon Redshift 管理指南》中的零 ETL 整合。

步驟 3：建立 DynamoDB 零 ETL 整合

建立零 ETL 整合之前，請務必完成標題為 與 Amazon Redshift 建立 DynamoDB 零 ETL 整合之前的先決條件 區段中的任務。在 DynamoDB 和 Amazon Redshift 之間建立整合是一個兩步驟的程序。首先從 DynamoDB 建立整合，然後將 Amazon Redshift 資料庫連接至此新建立的整合。

建立零 ETL 整合

1. 登入 AWS 管理主控台，並在 開啟 Amazon DynamoDB 主控台https://console.aws.amazon.com/dynamodbv2。
2. 在導覽窗格中選擇整合。
3. 選取建立零 ETL 整合，然後選擇 Amazon Redshift。
4. 這將帶您前往 Amazon Redshift 主控台。若要繼續此程序，請參閱為 DynamoDB 建立零 ETL 整合中的 DynamoDB 一節。

檢視與 Amazon Redshift 的 DynamoDB 零 ETL 整合

您可以檢視零 ETL 整合的詳細資訊，以查看其組態資訊和目前狀態。

若要在 Amazon DynamoDB 主控台中檢視零 ETL 整合的詳細資訊：

1. 登入 AWS 管理主控台，並在 開啟 Amazon DynamoDB 主控台https://console.aws.amazon.com/dynamodbv2。
2. 在 DynamoDB 主控台中，選擇整合。
3. 在零 ETL 整合窗格中，選取您要檢視的零 ETL 整合。

若要在 Amazon Redshift 主控台中檢視零 ETL 整合的詳細資訊：

1. 登入 AWS 管理主控台，並在 開啟 Amazon Redshift 主控台https://console.aws.amazon.com/redshiftv2。
2. 請遵循檢視零 ETL 整合中的步驟。

> **Note**
>
> 與 Amazon Redshift 進行零 ETL 整合的可能狀態列於《Amazon Redshift 管理指南》中的檢視零 ETL 整合中。

刪除與 Amazon Redshift 的 DynamoDB 零 ETL 整合

當您刪除零 ETL 整合時，您的資料不會從 DynamoDB 或 Amazon Redshift 中刪除，但 DynamoDB 會停止將資料從來源資料表傳送至 Amazon Redshift 目標。

刪除零 ETL 整合

1. 登入 AWS 管理主控台，並在 開啟 Amazon DynamoDB 主控台https://console.aws.amazon.com/dynamodbv2。
2. 在 DynamoDB 主控台中，選擇整合。
3. 在零 ETL 整合窗格中，選取您要刪除的零 ETL 整合。
4. 選擇管理。這將帶您前往整合詳細資訊頁面。
5. 如要確認刪除，請選擇 Delete (刪除)。

使用 COPY 命令將資料從 DynamoDB 載入 Amazon Redshift

Amazon Redshift 可與 Amazon DynamoDB 搭配，並提供進階商業智慧功能和強大的 SQL 介面。當您將資料從 DynamoDB 資料表複製到 Amazon Redshift 時，可以對該資料執行複雜的資料分析查詢，包括與 Amazon Redshift 叢集中其他資料表的聯結。

在佈建輸送量方面，DynamoDB 資料表的複製操作會計入該資料表的讀取容量。資料複製之後，Amazon Redshift 中的 SQL 查詢無論如何都不會影響 DynamoDB。原因是您的查詢是處理 DynamoDB 中資料的複本，而不是 DynamoDB 本身。

您必須先建立 DynamoDB 資料表作為資料的目標，才能從 Amazon Redshift 資料表載入資料。請記住，您會將資料從 NoSQL 環境複製至 SQL 環境，而且某個環境中的特定規則不適用於另一個環境。以下是一些需要考量的差異：

- DynamoDB 資料表名稱最多可以包含 255 個字元，包括「.」(點) 和「-」(破折號) 字元，並且區分大小寫。Amazon Redshift 資料表名稱不得超過 127 個字元以，而且不能包含點或破折號，也不區分大小寫。此外，資料表名稱不能與任何 Amazon Redshift 保留字衝突。
- DynamoDB 不支援 SQL 的 NULL 概念。您必須指定 Amazon Redshift 如何解譯 DynamoDB 中的空屬性值或空白屬性值，將它們視為 NULL 或空欄位。
- DynamoDB 資料類型未直接與 Amazon Redshift 的資料類型相對應。您必須確定 Amazon Redshift 資料表中每個欄位的資料類型和大小都正確，才能容納 DynamoDB 的資料。

以下是 Amazon Redshift SQL 中的 COPY 命令範例：

```
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;
```

在此範例中，DynamoDB 中的來源資料表是 `my-favorite-movies-table`。Amazon Redshift 中的目標資料表是 `favoritemovies`。`readratio 50` 子句會調節使用的佈建輸送量百分比，在此情況下，COPY 命令不會使用超過 50% 為 `my-favorite-movies-table` 佈建的讀取容量單位。強烈建議將此比率設定為小於平均未用佈建輸送量的值。

如需將 DynamoDB 中的資料載入 Amazon Redshift 的詳細說明，請參閱《Amazon Redshift 資料庫開發人員指南》中的下列章節：

- 從 DynamoDB 資料表載入資料
- COPY 命令
- COPY 範例

在 Amazon EMR 上使用 Apache Hive 處理 DynamoDB 資料

Amazon DynamoDB 已與 Apache Hive 整合；Apache Hive 是一種在 Amazon EMR 上執行的資料倉儲應用程式。Hive 可以在 DynamoDB 資料表中讀取和寫入資料，讓您能夠：

- 使用類似 SQL 的語言 (HiveQL) 查詢即時 DynamoDB 資料。
- 將資料從 DynamoDB 資料表複製到 Amazon S3 儲存貯體，反之亦然。
- 將資料從 DynamoDB 資料表複製到 Hadoop 分散式檔案系統 (HDFS) 中，反之亦然。
- 在 DynamoDB 資料表上執行聯結操作。

主題

- 概觀
- 教學課程：使用 Amazon DynamoDB 和 Apache Hive
- 在 Hive 中建立外部資料表
- 處理 HiveQL 陳述式
- 查詢 DynamoDB 中的資料
- 在 Amazon DynamoDB 之中複製和貼入資料
- 效能調校

概觀

Amazon EMR 服務可讓您輕鬆快速地以符合成本效益的方式來處理相當大量的資料。若要使用 Amazon EMR，您可以啟動執行 Hadoop 開放原始碼架構的 Amazon EC2 執行個體受管叢集。Hadoop 為實作 MapReduce 演算法的分散式應用程式，其中任務會映射到叢集中的多個節點。每個節點會與其他節點平行處理其指定的工作。最後，在單個節點上會減少輸出，產生最終結果。

您可以選擇啟動 Amazon EMR 叢集，以便維持其持續性或暫時性：

- 持續性叢集會一直執行，直到您將其關閉為止。持續性叢集非常適合用於資料分析、資料倉儲或任何其他互動式用途。
- 暫時性叢集執行的時間足以處理任務流程，然後自動關閉。暫時性叢集非常適合用於定期處理任務，如執行指令碼。

如需 Amazon EMR 架構和管理的相關資訊，請參閱《Amazon EMR 管理指南》。

在啟動 Amazon EMR 叢集時，請指定 Amazon EC2 執行個體的初始編號和類型。您還可以指定要在叢集上執行的其他分散式應用程式 (除了 Hadoop 本身)。這些應用程式包含 Hue、Mahout、Pig、Spark。

如需 Amazon EMR 應用程式的相關資訊，請參閱《Amazon EMR 版本指南》。

根據叢集組態，您可能具有下列一或多種節點類型：

- 領導節點：管理叢集，協調 MapReduce 可執行文件和原始資料子集到核心和任務執行個體群組的分佈。該節點也會追蹤每個已執行任務的狀態，並監控執行個體群組的運作狀態。一個叢集中只有一個領導節點。
- 核心節點：會執行 MapReduce 任務，然後使用 Hadoop 分散式檔案系統 (HDFS) 存放資料。
- 任務節點 (選用)：執行 MapReduce 任務。

教學課程：使用 Amazon DynamoDB 和 Apache Hive

在本教學課程中，您將啟動 Amazon EMR 叢集，然後使用 Apache Hive 處理存放在 DynamoDB 資料表中的資料。

Hive 是 Hadoop 的資料倉儲應用程式，允許您用其處理和分析來自多個來源的資料。Hive 提供類似 SQL 的語言 (HiveQL)，可讓您處理本機存放在 Amazon EMR 叢集或外部資料來源 (如 Amazon DynamoDB) 中的資料。

如需詳細資訊，請參閱 Hive 教學課程。

主題

- 開始之前
- 步驟 1：建立 Amazon EC2 金鑰對
- 步驟 2：啟動 Amazon EMR 叢集
- 步驟 3：連接到領導節點
- 步驟 4：將資料載入 HDFS
- 步驟 5：將資料複製到 DynamoDB
- 步驟 6：查詢 DynamoDB 資料表中的資料
- 步驟 7：(選用) 清除

開始之前

在此教學課程中，您將需執行下列項目：

- AWS 帳戶。如果您沒有帳戶，請參閱 註冊 AWS。
- SSH 用戶端 (Secure Shell)。您可以使用 SSH 用戶端連線到 Amazon EMR 叢集的領導節點並執行互動式命令。預設情況下，大多數的 Linux、Unix 和 Mac OS X 裝置都可以使用 SSH 用戶端。Windows 使用者可以下載並安裝支援 SSH 的 PuTTY 用戶端。

下一步驟

步驟 1：建立 Amazon EC2 金鑰對

步驟 1：建立 Amazon EC2 金鑰對

在此步驟中，您要建立連線到 Amazon EMR 領導節點並執行 Hive 命令所需的 Amazon EC2 金鑰對。

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 選擇區域 (例如 US West (Oregon))。這應與 DynamoDB 資料表所在的區域相同。
3. 在導覽窗格中，選擇 Key Pairs (金鑰對)。
4. 選擇 Create Key Pair (建立金鑰對)。
5. 在 Key pair name (金鑰對名稱) 中，為金鑰對輸入名稱 (例如 mykeypair)，然後選擇 Create (建立)。

6. 下載私有金鑰檔案。檔案名稱以 `.pem` 結尾 (例如 `mykeypair.pem`)。將此私有金鑰檔案保存在安全的地方。您需要它來存取使用此金鑰對啟動的任何 Amazon EMR 叢集。

   > **Important**
   >
   > 如果遺失金鑰對，您便無法連線到 Amazon EMR 叢集的領導節點。

   如需金鑰對的詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 金鑰對。

下一步驟

步驟 2：啟動 Amazon EMR 叢集

步驟 2：啟動 Amazon EMR 叢集

在此步驟中，您將設定並啟動 Amazon EMR 叢集。DynamoDB 的 Hive 和儲存處理常式已安裝在叢集上。

1. 在 https：//https://console.aws.amazon.com/emr 開啟 Amazon EMR 主控台。
2. 選擇 Create Cluster (建立叢集)。
3. 在 Create Cluster：Quick Options (建立叢集：快速選項) 頁面上，執行下列動作：

   a. 在 Cluster name (叢集名稱) 中，為叢集輸入名稱 (例如 `My EMR cluster`)。
   b. 在 EC2 key pair (EC2 金鑰對) 中，選擇您之前建立的金鑰對。

   將其他設定保留為各自的預設設定。
4. 選擇 Create cluster (建立叢集)。

啟動叢集需要幾分鐘的時間。您可以使用 Amazon EMR 主控台中的 Cluster Details (叢集詳細資訊) 頁面監控此流程。

當狀態變更為 `Waiting` 時，即叢集已準備就緒。

叢集日誌檔案和 Amazon S3

Amazon EMR 叢集會產生日誌檔案，其中包含叢集狀態和偵錯資訊的相關資訊。Create Cluster：Quick Options (建立叢集：快速選項) 的預設設定包含設定 Amazon EMR 記錄。

如果尚未存在， AWS 管理主控台 會建立 Amazon S3 儲存貯體。儲存貯體名稱為 `aws-logs-account-id-region`，其中 *account-id*是 AWS 您的帳戶號碼，而 *region*是您啟動叢集的區域（例如 `aws-logs-123456789012-us-west-2`)。

> **Note**
>
> 您可以使用 Amazon S3 主控台來檢視日誌檔案。如需詳細資訊，請參閱《Amazon EMR 管理指南》中的檢視日誌檔案。

除了日誌記錄之外，您還可以將此儲存貯體用於其他目的。例如，您可以使用儲存貯體作為儲存 Hive 指令碼的位置，或將資料從 Amazon DynamoDB 匯出到 Amazon S3 時作為目的地。

下一步驟

步驟 3：連接到領導節點

步驟 3：連接到領導節點

當 Amazon EMR 叢集狀態變更為 `Waiting` 時，您將能夠使用 SSH 連線到領導節點並執行命令列操作。

1. 在 Amazon EMR 主控台中，選擇叢集的名稱以檢視其狀態。
2. 在 Cluster Details (叢集詳細資訊) 頁面中，尋找 Leader public DNS (領導公有 DNS) 欄位。這是 Amazon EMR 叢集領導節點的公有 DNS 名稱。
3. 在 DNS 名稱右側，選擇 SSH 連結。
4. 請按使用 SSH 連線到領導節點中的說明進行。

   根據您的作業系統，選擇 Windows 索引標籤或 Mac/Linux 索引標籤，然後按照連線到領導節點的說明進行。

使用 SSH 或 PuTTY 連線到領導節點後，您應該會看到類似下列內容的命令提示：

```
[hadoop@ip-192-0-2-0 ~]$
```

下一步驟

步驟 4：將資料載入 HDFS

步驟 4：將資料載入 HDFS

在此步驟中，您要將資料檔案複製到 Hadoop 分散式檔案系統 (HDFS)，然後建立會映射到資料檔案的外部 Hive 資料表。

下載範例資料

1. 下載範例資料封存 (`features.zip`)：

   ```
   wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   features.zip
   ```

2. 從封存擷取 `features.txt` 檔案：

   ```
   unzip features.zip
   ```

3. 檢視 `features.txt` 檔案的前幾列：

   ```
   head features.txt
   ```

   結果看起來會與下列類似：

   ```
   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` 檔案包含來自美國地名委員會 (http://geonames.usgs.gov/domestic/download_data.htm) 的資料子集。每一列中的欄位代表下列項目：

   - 功能 ID (唯一識別碼)
   - 名稱
   - 類別 (湖泊、森林、溪流等)
   - State

- 緯度 (度數)
- 經度 (度數)
- 高度 (英尺)

4. 在命令提示中，輸入以下命令：

```
hive
```

命令提示會變更為此項目：hive>

5. 輸入下列 HiveQL 陳述式來建立原生 Hive 資料表：

```
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. 輸入下列 HiveQL 陳述式來載入包含資料的資料表：

```
LOAD DATA
LOCAL
INPATH './features.txt'
OVERWRITE
INTO TABLE hive_features;
```

7. 您現在有一個原生 Hive 資料表已填入來自 features.txt 檔案的資料。若要進行驗證，請輸入下列 HiveQL 陳述式：

```
SELECT state_alpha, COUNT(*)
FROM hive_features
GROUP BY state_alpha;
```

輸出應顯示州清單以及每個州的地理特徵數量。

下一步驟

[步驟 5：將資料複製到 DynamoDB]

步驟 5：將資料複製到 DynamoDB

在此步驟中，您會將資料從 Hive 資料表 (`hive_features`) 複製到 DynamoDB 中的新資料表。

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 選擇 Create Table (建立資料表)。
3. 在 Create DynamoDB table (建立 DynamoDB 資料表) 頁面中，執行下列作業：

   a. 在 Table (資料表) 中輸入 **`Features`**。

   b. 對於 Primary key (主索引鍵)，在 Partition key (分割區索引鍵) 欄位中輸入 **`Id`**。將資料類型設定為 Number (數字)。

      清除 Use default settings (使用預設設定)。為 Provisioned Capacity (佈建容量)，輸入下列內容：

      - 讀取容量單位：`10`
      - 寫入容量單位：`10`

   選擇 Create (建立)。
4. 在 Hive 提示中輸入下列 HiveQL 陳述式：

```
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
);
```

您已經在 DynamoDB 中的 Hive 和 Features (特徵) 資料表之間建立了映射。

5. 輸入下列 HiveQL 陳述式，將資料匯入到 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 會提交將由 Amazon EMR 叢集處理的 MapReduce 任務。完成任務需要幾分鐘的時間。

6. 驗證資料已載入到 DynamoDB：

   a. 在 DynamoDB 主控台的導覽窗格中，選擇 Tables (資料表)。

   b. 選擇 Features (特徵) 資料表，然後選擇 Items (項目) 索引標籤來檢視資料。

下一步驟

步驟 6：查詢 DynamoDB 資料表中的資料

步驟 6：查詢 DynamoDB 資料表中的資料

在此步驟中，您將使用 HiveQL 來查詢 DynamoDB 中的 Features (特徵) 資料表。嘗試下列 Hive 查詢：

1. 所有功能類型 (`feature_class`) 依字母順序排序：

```
SELECT DISTINCT feature_class
FROM ddb_features
ORDER BY feature_class;
```

2. 所有以字母「M」開頭的湖泊：

```
SELECT feature_name, state_alpha
FROM ddb_features
```

```
WHERE feature_class = 'Lake'
AND feature_name LIKE 'M%'
ORDER BY feature_name;
```

3. 至少三個特徵高於一英里 (5,280 英尺) 的州：

```
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;
```

下一步驟

步驟 7：(選用) 清除

步驟 7：(選用) 清除

您已完成教學課程，可以繼續閱讀本節，進一步了解如何在 Amazon EMR 中使用 DynamoDB 資料。如果繼續閱讀，您可以決定讓 Amazon EMR 叢集保持正常運作。

如果不再需要叢集，您應該終止它並移除任何相關聯的資源。這將協助您避免為不需要的資源付費。

1. 終止 Amazon EMR 叢集：
   a. 在 https：//https://console.aws.amazon.com/emr 開啟 Amazon EMR 主控台。
   b. 選擇 Amazon EMR 叢集，選擇 Terminate (終止)，然後確認。
2. 刪除 DynamoDB 中的 Features (特徵) 資料表：
   a. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
   b. 在導覽窗格中，選擇 Tables (資料表)。
   c. 選擇 Features (特徵) 資料表。在 Actions (動作) 選單中，選擇 Delete Table (刪除資料表)。
3. 刪除含有 Amazon EMR 日誌檔案的 Amazon S3 儲存貯體：
   a. 開啟位於 https://console.aws.amazon.com/s3/ 的 Amazon S3 主控台。
   b. 從儲存貯體清單中，選擇 `aws-logs-` *`accountID-region`*，其中 *`accountID`* 是 AWS 您的帳戶號碼，*`# region`* 是您啟動叢集的區域。
   c. 在 Action (動作) 選單中，選擇 Delete (刪除)。

在 Hive 中建立外部資料表

在 教學課程：使用 Amazon DynamoDB 和 Apache Hive 中，您已建立映射到 DynamoDB 資料表的外部 Hive 資料表。在針對外部資料表發出 HiveQL 陳述式時，讀取和寫入操作會傳遞到 DynamoDB 資料表。

您可以將外部資料表視為指向在別處管理和存放的資料來源的指標。在此情況下，基礎資料來源即為 DynamoDB 資料表。(資料表必須已存在。您無法從 Hive 內建立、更新或刪除 DynamoDB 資料表。) 您可以使用 CREATE EXTERNAL TABLE 陳述式建立外部資料表。之後，您可以使用 HiveQL 來處理 DynamoDB 中的資料，就好像資料本機存放在 Hive 中一樣。

> **Note**
>
> 您可以使用 INSERT 陳述式將資料插入到外部資料表中，並使用 SELECT 陳述式從中選取資料。不過，您無法使用 UPDATE 或 DELETE 陳述式來運用資料表中的資料。

如果不再需要外部資料表，您可以使用 DROP TABLE 陳述式將其移除。在本案例中，DROP TABLE 僅移除 Hive 中的外部資料表。它不會影響基礎 DynamoDB 資料表或其中的任何資料。

主題

- 建立外部資料表語法
- 資料類型映射

建立外部資料表語法

以下內容顯示用於建立映射到 DynamoDB 資料表的外部 Hive 資料表的 HiveQL 語法：

```
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..."
);
```

第 1 列為 CREATE EXTERNAL TABLE 陳述式的開始，可讓您在其中提供要建立的 Hive 資料表 (hive_table)。

第 2 列指定 hive_table 的資料欄和資料類型。您需要定義與 DynamoDB 資料表中屬性對應的資料欄和資料類型。

第 3 列是 STORED BY 子句，可讓您在其中指定類別來處理 Hive 和 DynamoDB 資料表之間的資料管理。針對 DynamoDB，STORED BY 應設定為 `'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'`。

第 4 列為 TBLPROPERTIES 子句的開始，可讓您在其中為 DynamoDBStorageHandler 定義下列參數：

- `dynamodb.table.name`：DynamoDB 資料表的名稱。
- `dynamodb.column.mapping`：Hive 資料表中資料欄名稱的配對，以及其在 DynamoDB 資料表中的對應屬性。每個配對形式為 hive_column_name:dynamodb_attribute_name，且配對以逗號分隔。

注意下列事項：

- Hive 資料表名稱的名稱不一定要與 DynamoDB 資料表名稱相同。
- Hive 資料表資料欄名稱不一定要與 DynamoDB 資料表中的名稱相同。
- 由 `dynamodb.table.name` 指定的資料表必須存在於 DynamoDB 中。
- 在 `dynamodb.column.mapping` 中：
  - 您必須映射 DynamoDB 資料表的金鑰結構描述屬性。這包含分割區索引鍵和排序索引鍵 (若有)。
  - 您不需要映射 DynamoDB 資料表的非索引鍵屬性。不過，在查詢 Hive 資料表時，您不會看到這些屬性的任何資料。
  - 如果 Hive 資料表資料欄的資料類型與 DynamoDB 屬性的資料類型不相容，在查詢 Hive 資料表時，您會在這些資料欄中看到 NULL。

> **Note**
>
> 此 CREATE EXTERNAL TABLE 陳述式不會對 TBLPROPERTIES 子句執行任何驗證。您為 `dynamodb.table.name` 和 `dynamodb.column.mapping` 提供的值僅在您嘗試存取資料表時由 DynamoDBStorageHandler 類別評估。

資料類型映射

以下資料表顯示 DynamoDB 資料類型和相容的 Hive 資料類型：

DynamoDB 資料類型	Hive 資料類型
String	STRING
數字	BIGINT 或 DOUBLE
二進位	BINARY
String Set	ARRAY<STRING>
Number Set	ARRAY<BIGINT> 或 ARRAY<DOUBLE>
Binary Set	ARRAY<BINARY>

> **Note**
>
> 下列 DynamoDB 資料類型不受 DynamoDBStorageHandler 類別支援，因此無法與 dynamodb.column.mapping 搭配使用：
>
> - Map
> - 清單
> - Boolean
> - Null
>
> 但是，如果您需要使用這些資料類型，則可以建立名為 item 的單一實體來代表整個 DynamoDB 項目，做為對應中索引鍵和值的字串對應。如需詳細資訊，請參閱在無資料欄映射的情況下複製資料

如果要映射 Number 類型的 DynamoDB 屬性，則必須選擇適當的 Hive 類型：

- Hive BIGINT 類型用於 8 位元組帶正負號的整數。該類型與 Java 中的 long 資料類型相同。
- Hive DOUBLE 類型用於 8 位元雙精度浮點數。該類型與 Java 中的 double 類型相同。

如果 DynamoDB 中儲存的數值資料精確度高於您選擇的 Hive 資料類型，則存取 DynamoDB 資料可能會導致精確度損失。

如果您將 Binary 類型的資料從 DynamoDB 匯出到 (Amazon S3) 或 HDFS，則資料會以 Base64-encoded 的字串存放。如果您將資料以 Binary 類型從 Amazon S3 或 HDFS 匯入 DynamoDB，則必須確保資料已編碼為 Base64 字串。

處理 HiveQL 陳述式

Hive 是在 Hadoop 上執行的應用程式；Hadoop 是用於執行 MapReduce 任務的批次導向框架。在發出 HiveQL 陳述式時，Hive 會確定它是否可以立即返回結果，或者它是否必須提交 MapReduce 任務。

例如，考量 ddb_features 資料表 (從 教學課程：使用 Amazon DynamoDB 和 Apache Hive)。下列 Hive 查詢會列印州縮寫和各州境內山峰的數量：

```
SELECT state_alpha, count(*)
FROM ddb_features
WHERE feature_class = 'Summit'
GROUP BY state_alpha;
```

Hive 不會立即返回結果。相反，它會提交由 Hadoop 架構處理的 MapReduce 任務。Hive 將等待任務完成，然後才會顯示查詢結果：

```
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)
```

監控和取消任務

當 Hive 啟動 Hadoop 任務時，它會列印該任務的輸出。任務完成狀態隨著任務的進行而更新。在某些情況下，狀態可能會長時間無法進行更新。(在查詢具有較低佈建讀取容量設定的大型 DynamoDB 資料表時，可能會發生這種情況。)

如需在完成之前取消任務，您可以在任何時候輸入 **Ctrl+C**。

查詢 DynamoDB 中的資料

以下範例顯示使用 HiveQL 查詢存放在 DynamoDB 中的資料的一些方法。

這些範例參考教學課程 (步驟 5：將資料複製到 DynamoDB) 中的 ddb_features 資料表。

主題

- 使用彙總函數
- 使用 GROUP BY 和 HAVING 子句
- 加入兩個 DynamoDB 資料表
- 聯結來自不同來源的資料表

使用彙總函數

HiveQL 提供用於摘要資料值的內建函數。例如，您可以使用 MAX 函數尋找所選資料欄的最大值。下列範例會傳回科羅拉多州最高山峰的海拔。

```
SELECT MAX(elev_in_ft)
FROM ddb_features
WHERE state_alpha = 'CO';
```

使用 GROUP BY 和 HAVING 子句

您可以使用 GROUP BY 子句跨多筆記錄收集資料。這常會與 SUM、COUNT、MIN 或 MAX 等彙總函數搭配著使用。您也可以使用 HAVING 子句捨棄任何不符合特定條件的結果。

以下範例返回 ddb_features 資料表中具有五個以上特徵的州的最高海拔清單。

```
SELECT state_alpha, max(elev_in_ft)
FROM ddb_features
GROUP BY state_alpha
HAVING count(*) >= 5;
```

加入兩個 DynamoDB 資料表

以下範例示範將另一個 Hive 資料表 (east_coast_states) 映射到 DynamoDB 中的資料表。此 SELECT 陳述式是這兩個資料表之間的聯結。聯結會在叢集上運算並傳回。聯結不會在 DynamoDB 中發生。

考慮名為 EastCoastStates 的 DynamoDB 資料表，其中包含下列資料：

```
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
```

假設該資料表可用作名為 east_coast_states 的 Hive 外部資料表：

```
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");
```

下列聯結會傳回美國東海岸至少具有三個特徵的州：

```
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;
```

聯結來自不同來源的資料表

在以下範例中，s3_east_coast_states 是與 Amazon S3 中存放的 CSV 檔案相關聯的 Hive 資料表。此 ddb_features 資料表與 DynamoDB 中的資料相關聯。下列範例會聯結這兩個資料表，傳回名稱以「New」開頭的州的地理特徵。

```
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%';
```

在 Amazon DynamoDB 之中複製和貼入資料

在 教學課程：使用 Amazon DynamoDB 和 Apache Hive 中，您會將資料從原生 Hive 資料表複製到外部 DynamoDB 資料表，然後查詢外部 DynamoDB 資料表。該資料表為外部資料表，因為它存在於 Hive 之外。即使捨棄映射到它的 Hive 資料表，DynamoDB 中的資料表也不會受到影響。

Hive 是在 DynamoDB 資料表、Amazon S3 儲存貯體、原生 Hive 資料表和 Hadoop 分散式檔案系統 (HDFS) 之間複製資料的絕佳解決方案。本節將提供這些操作的範例。

主題

- 在 DynamoDB 和原生 Hive 資料表之間複製資料
- 在 DynamoDB 與 Amazon S3 之間複製資料
- 在 DynamoDB 與 HDFS 之間複製資料

- 使用資料壓縮
- 讀取不可列印的 UTF-8 字元資料

在 DynamoDB 和原生 Hive 資料表之間複製資料

如果 DynamoDB 資料表中有資料，您可以將這些資料複製到原生 Hive 資料表。這會為您提供截至資料複製時間的資料快照。

如果您需要執行許多 HiveQL 查詢，但又不想消耗 DynamoDB 佈建的輸送容量，則可以決定這樣做。由於原生 Hive 資料表中的資料是 DynamoDB 資料的複本，而不是「即時」資料，您不應期望查詢可獲得最新資料。

> **Note**
>
> 本節中的範例假設您會遵循 教學課程：使用 Amazon DynamoDB 和 Apache Hive 中的步驟，並且在 DynamoDB 中有一個名為 ddb_features 的外部資料表。

Example從 DynamoDB 到原生 Hive 資料表

您可以建立一個原生 Hive 資料表並使用 ddb_features 的資料填入內容，如下所示：

```
CREATE TABLE features_snapshot AS
SELECT * FROM ddb_features;
```

您可以隨時重新整理資料：

```
INSERT OVERWRITE TABLE features_snapshot
SELECT * FROM ddb_features;
```

在這些範例中，子查詢 `SELECT * FROM ddb_features` 將從 ddb_features 擷取所有資料。如果您只想複製資料的子集，則可以在子查詢中使用 `WHERE` 子句。

下列範例會建立原生 Hive 資料表，其中只包含一些湖泊和山峰的屬性：

```
CREATE TABLE lakes_and_summits AS
SELECT feature_name, feature_class, state_alpha
FROM ddb_features
WHERE feature_class IN ('Lake','Summit');
```

Example從原生 Hive 資料表到 DynamoDB

使用下列 HiveQL 陳述式，將資料從本機 Hive 資料表複製到 ddb_features：

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM features_snapshot;
```

在 DynamoDB 與 Amazon S3 之間複製資料

如果 DynamoDB 資料表中有資料，則可以使用 Hive 將資料複製到 Amazon S3 儲存貯體。

如果您想要在 DynamoDB 資料表中建立資料的封存，則可執行此操作。例如，假設您的測試環境需要在 DynamoDB 中使用一組基準測試資料。您可以將基準資料複製到 Amazon S3 儲存貯體，然後執行測試。之後，您可以將 Amazon S3 儲存貯體中的基準資料還原到 DynamoDB，以此重設測試環境。

如果透過 教學課程：使用 Amazon DynamoDB 和 Apache Hive 作業，則表示您已經擁有一個包含 Amazon EMR 日誌的 Amazon S3 儲存貯體。如果您知道儲存貯體的根路徑，則可以將此儲存貯體用於本節中的範例：

1. 在 https：//https://console.aws.amazon.com/emr 開啟 Amazon EMR 主控台。
2. 對於 Name (名稱)，選擇您的叢集。
3. URI 會列在 Configuration Details (組態詳細資訊) 下方的 Log URI (記錄 URI)。
4. 記下儲存貯體的根路徑。命名慣例是：

   ```
   s3://aws-logs-accountID-region
   ```

   其中 *accountID* 是 AWS 您的帳戶 ID，而區域是儲存貯 AWS 體的區域。

> **Note**
>
> 對於這些範例，我們將使用儲存貯體內的子路徑，如下列範例所示：
> `s3://aws-logs-123456789012-us-west-2/hive-test`

下列程序假設您會遵循教學課程中的步驟，並且在 DynamoDB 中有一個名為 ddb_features 的外部資料表。

主題

- 使用 Hive 預設格式複製資料

- 以使用者指定格式複製資料
- 在無資料欄映射的情況下複製資料
- 檢視 Amazon S3 中的資料

使用 Hive 預設格式複製資料

Example從 DynamoDB 到 Amazon S3

使用 INSERT OVERWRITE 陳述式直接寫入至 Amazon S3。

```
INSERT OVERWRITE DIRECTORY 's3://aws-logs-123456789012-us-west-2/hive-test'
SELECT * FROM ddb_features;
```

Amazon S3 中的資料檔案如下所示：

```
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
```

每個欄位都以 SOH 字元分隔 (標題的開頭，0x01)。在檔案中，SOH 會顯示為 **^A**。

Example從 Amazon S3 到 DynamoDB

1. 建立一個指向 Amazon S3 中未格式化資料的外部資料表。

   ```
   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. 將資料複製到 DynamoDB。

   ```
   INSERT OVERWRITE TABLE ddb_features
   SELECT * FROM s3_features_unformatted;
   ```

以使用者指定格式複製資料

如果您想要指定自己的欄位分隔符號字元，則可建立映射至 Amazon S3 儲存貯體的外部資料表。您可以使用此技術建立具有逗號分隔值 (CSV) 的資料檔案。

Example從 DynamoDB 到 Amazon S3

1. 建立映射至 Amazon S3 的 Hive 外部資料表。執行此操作時，請確定資料類型與 DynamoDB 外部資料表的資料類型一致。

```
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. 從 DynamoDB 複製資料。

```
INSERT OVERWRITE TABLE s3_features_csv
SELECT * FROM ddb_features;
```

Amazon S3 中的資料檔案如下所示：

```
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從 Amazon S3 到 DynamoDB

使用單一 HiveQL 陳述式，您可以使用來自 Amazon S3 的資料填入 DynamoDB 資料表：

```
INSERT OVERWRITE TABLE ddb_features
```

```
SELECT * FROM s3_features_csv;
```

在無資料欄映射的情況下複製資料

您可以使用原始格式從 DynamoDB 複製資料，並將其寫入至 Amazon S3，而無需指定任何資料類型或資料欄映射。您可以使用此方法建立 DynamoDB 資料的封存，並將其存放到 Amazon S3 中。

Example從 DynamoDB 到 Amazon S3

1. 建立與 DynamoDB 資料表相關聯的外部資料表。(此 HiveQL 陳述式中沒有 `dynamodb.column.mapping`。)

   ```
   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. 建立另一個與 Amazon S3 儲存貯體相關聯的外部資料表。

   ```
   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. 將資料從 DynamoDB 複製到 Amazon S3。

   ```
   INSERT OVERWRITE TABLE s3_features_no_mapping
   SELECT * FROM ddb_features_no_mapping;
   ```

Amazon S3 中的資料檔案如下所示：

```
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
```

每個欄位都以 STX 字元 (文字的開頭，0x02) 開頭，並以 ETX 字元 (文字結尾，0x03) 結尾。在此檔案中，STX 會顯示為 **^B**，ETX 會顯示為 **^C**。

Example從 Amazon S3 到 DynamoDB

使用單一 HiveQL 陳述式，您可以使用來自 Amazon S3 的資料填入 DynamoDB 資料表：

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM s3_features_no_mapping;
```

檢視 Amazon S3 中的資料

如果您使用 SSH 連線到領導節點，則可以使用 AWS Command Line Interface (AWS CLI) 來存取 Hive 寫入至 Amazon S3 的資料。

以下步驟的撰寫是假設您已使用本節其中一個程序，將資料從 DynamoDB 複製到 Amazon S3。

1. 如果您目前處於 Hive 命令提示字元，請離開至 Linux 命令提示字元。

   ```
   hive> exit;
   ```

2. 列出 Amazon S3 儲存貯體中的 Hive 測試目錄內容。(這是 Hive 從 DynamoDB 複製資料的位置。)

   ```
   aws s3 ls s3://aws-logs-123456789012-us-west-2/hive-test/
   ```

   回應外觀會與下列類似：

   ```
   2016-11-01 23:19:54 81983 000000_0
   ```

   檔案名稱 (000000_0) 是由系統產生的。

3. (選用) 您可以將資料檔案從 Amazon S3 複製到領導節點上的本機檔案系統。執行此操作之後，您可以使用標準 Linux 命令列公用程式來使用檔案中的資料。

   ```
   aws s3 cp s3://aws-logs-123456789012-us-west-2/hive-test/000000_0 .
   ```

回應外觀會與下列類似：

```
download: s3://aws-logs-123456789012-us-west-2/hive-test/000000_0
to ./000000_0
```

> **Note**
>
> 領導節點上的本機檔案系統容量有限。請勿將此命令與大於本機檔案系統中可用空間的檔案搭配使用。

在 DynamoDB 與 HDFS 之間複製資料

如果 DynamoDB 資料表中有資料，則可以使用 Hive 將資料複製到 Hadoop 分散式檔案系統 (HDFS)。

如果正在執行需要來自 DynamoDB 之資料的 MapReduce 任務，您可能會執行這項操作。如果您將資料從 DynamoDB 複製到 HDFS，Hadoop 可以處理此資訊，同時會使用 Amazon EMR 叢集中的所有可用節點。完成 MapReduce 任務後，您就能將結果從 HDFS 寫入至 DDB。

在下列範例中，Hive 將在下列 HDFS 目錄讀取及寫入內容：`/user/hadoop/hive-test`

> **Note**
>
> 本節中的範例假設您會遵循 教學課程：使用 Amazon DynamoDB 和 Apache Hive 中的步驟，並且在 DynamoDB 中有一個名為 ddb_features 的外部資料表。

主題

- 使用 Hive 預設格式複製資料
- 以使用者指定格式複製資料
- 在無資料欄映射的情況下複製資料
- 存取 HDFS 中的資料

使用 Hive 預設格式複製資料

Example從 DynamoDB 到 HDFS

使用 `INSERT OVERWRITE` 陳述式直接寫入至 HDFS。

```
INSERT OVERWRITE DIRECTORY 'hdfs:///user/hadoop/hive-test'
SELECT * FROM ddb_features;
```

HDFS 中的資料檔案外觀如下：

```
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
```

每個欄位都以 SOH 字元分隔 (標題的開頭，0x01)。在檔案中，SOH 會顯示為 **^A**。

Example從 HDFS 到 DynamoDB

1. 建立映射至 HDFS 中未格式化資料的外部資料表。

   ```
   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. 將資料複製到 DynamoDB。

   ```
   INSERT OVERWRITE TABLE ddb_features
   SELECT * FROM hdfs_features_unformatted;
   ```

以使用者指定格式複製資料

如果要使用不同的欄位分隔符號字元，則可以建立映射至 HDFS 目錄的外部資料表。您可以使用此技術建立具有逗號分隔值 (CSV) 的資料檔案。

Example從 DynamoDB 到 HDFS

1. 建立映射至 HDFS 的 Hive 外部資料表。執行此操作時，請確定資料類型與 DynamoDB 外部資料表的資料類型一致。

```
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. 從 DynamoDB 複製資料。

```
INSERT OVERWRITE TABLE hdfs_features_csv
SELECT * FROM ddb_features;
```

HDFS 中的資料檔案外觀如下：

```
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從 HDFS 到 DynamoDB

使用單一 HiveQL 陳述式，您可以使用來自 HDFS 的資料填入 DynamoDB 資料表：

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_csv;
```

在無資料欄映射的情況下複製資料

您可以使用原始格式從 DynamoDB 複製資料，並將其寫入至 HDFS，而無需指定任何資料類型或資料欄映射。您可以使用此方法建立 DynamoDB 資料的封存，並將其存放到 HDFS 中。

> **Note**
>
> 如果 DynamoDB 資料表包含 Map、List、Boolean 或 Null 類型的屬性，這便是您可以使用 Hive 將資料從 DynamoDB 複製到 HDFS 的唯一方法。

Example從 DynamoDB 到 HDFS

1. 建立與 DynamoDB 資料表相關聯的外部資料表。(此 HiveQL 陳述式中沒有 `dynamodb.column.mapping`。)

```
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. 建立另一個與 HDFS 目錄相關聯的外部資料表。

```
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. 將資料從 DynamoDB 複製到 HDFS。

```
INSERT OVERWRITE TABLE hdfs_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

HDFS 中的資料檔案外觀如下：

```
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
```

每個欄位都以 STX 字元 (文字的開頭，0x02) 開頭，並以 ETX 字元 (文字結尾，0x03) 結尾。在此檔案中，STX 會顯示為 **^B**，ETX 會顯示為 **^C**。

Example從 HDFS 到 DynamoDB

使用單一 HiveQL 陳述式，您可以使用來自 HDFS 的資料填入 DynamoDB 資料表：

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM hdfs_features_no_mapping;
```

存取 HDFS 中的資料

HDFS 是一個分散式檔案系統，可供 Amazon EMR 叢集中的所有節點存取。如果您使用 SSH 連線到領導節點，則可以使用命令列工具來存取 Hive 寫入至 HDFS 的資料。

HDFS 與領導節點上的本機檔案系統不盡相同。您無法使用標準 Linux 命令 (例如 `cat`、`cp`、`mv` 或 `rm`) 處理 HDFS 中的檔案及目錄。這些任務要使用 `hadoop fs` 命令來執行。

以下步驟的撰寫是假設您已使用本節其中一個程序，將資料從 DynamoDB 複製到 HDFS。

1. 如果您目前處於 Hive 命令提示字元，請離開至 Linux 命令提示字元。

   ```
   hive> exit;
   ```

2. 列出 HDFS 中 /user/hadoop/hive-test 目錄的內容。(這是 Hive 從 DynamoDB 複製資料的位置。)

   ```
   hadoop fs -ls /user/hadoop/hive-test
   ```

   回應外觀會與下列類似：

   ```
   Found 1 items
   -rw-r--r-- 1 hadoop hadoop 29504 2016-06-08 23:40 /user/hadoop/hive-test/000000_0
   ```

   檔案名稱 (000000_0) 是由系統產生的。

3. 檢視檔案的內容：

   ```
   hadoop fs -cat /user/hadoop/hive-test/000000_0
   ```

> **Note**
>
> 在此範例中，檔案相對較小 (大約 29 KB)。請審慎對非常大或包含不可列印字元的檔案使用此命令。

4. (選用) 您可以將資料檔案從 HDFS 複製到領導節點上的本機檔案系統。執行此操作之後，您可以使用標準 Linux 命令列公用程式來使用檔案中的資料。

```
hadoop fs -get /user/hadoop/hive-test/000000_0
```

此命令不會覆寫檔案。

> **Note**
>
> 領導節點上的本機檔案系統容量有限。請勿將此命令與大於本機檔案系統中可用空間的檔案搭配使用。

使用資料壓縮

在使用 Hive 在不同的資料來源之間複製資料時，您可以要求即時資料壓縮。Hive 提供了幾個壓縮解碼器。您可以在 Hive 工作階段期間選擇一個。在執行此操作時，會以指定格式來壓縮資料。

以下範例會使用 Lempel-Ziv-Oberhumer (LZO) 演算法壓縮資料。

```
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;
```

Amazon S3 中產生的檔案會具有系統產生的名稱，且以 `.lzo` 結尾 (例如，`8d436957-57ba-4af7-840c-96c2fc7bb6f5-000000.lzo`)。

可用的壓縮轉碼器為：

- 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

讀取不可列印的 UTF-8 字元資料

您可以在建立 Hive 資料表時使用 STORED AS SEQUENCEFILE 子句，以此讀取和寫入不可列印的 UTF-8 字元資料。SequenceFile 是 Hadoop 二進位檔案格式。您需要使用 Hadoop 來讀取此檔案。下列範例顯示如何將資料從 DynamoDB 匯出到 Amazon S3。您可以使用此功能處理不可列印的 UTF-8 編碼字元。

```
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;
```

效能調校

在建立映射至 DynamoDB 資料表的 Hive 外部資料表時，您不會從 DynamoDB 消耗任何讀取或寫入容量。不過，Hive 資料表上的讀取和寫入活動 (例如 INSERT 或 SELECT) 會直接轉譯為基礎 DynamoDB 資料表上的讀取和寫入作業。

Amazon EMR 上的 Apache Hive 會實作自己的邏輯來平衡 DynamoDB 資料表上的輸入/輸出負載，並盡量減少超出資料表佈建輸送量的可能性。每個 Hive 查詢結束時，Amazon EMR 都會傳回執行時期指標，包括超出佈建輸送量的次數。您可以將此資訊與 DynamoDB 資料表上的 CloudWatch 指標一起使用，改善後續要求的效能。

Amazon EMR 主控台可為叢集提供基本監控工具。如需詳細資訊，請參閱《Amazon EMR 管理指南》中的檢視及監控叢集。

您也可以使用以 Web 類型工具 (例如 Hue、Ganglia 和 Hadoop Web 介面) 來監控叢集及 Hadoop 任務。如需詳細資訊，請參閱《Amazon EMR 管理指南》中的檢視 Amazon EMR 叢集上託管的 Web 介面。

本節說明可在外部 DynamoDB 資料表上調校 Hive 作業效能所採取的步驟。

主題

- DynamoDB 佈建輸送量
- 調整映射器
- 其他主題

DynamoDB 佈建輸送量

在針對外部 DynamoDB 資料表發出 HiveQL 陳述式時，`DynamoDBStorageHandler` 類別會產生適當的低階 DynamoDB API 請求 (這些請求會消耗佈建的輸送量)。如果 DynamoDB 資料表上的讀取或寫入容量不足，則會對請求進行限流，繼而導致 HiveQL 效能緩慢。因此，您應該確保資料表擁有足夠的輸送容量。

例如，假設您已為 DynamoDB 資料表佈建 100 單位的讀取容量。這將讓您每秒讀取 409,600 個位元組 (100 × 4 KB 讀取容量單位大小)。現在假設資料表包含 20 GB 的資料 (21,474,836,480 個位元組)，而且您想要使用 `SELECT` 陳述式來選擇使用 HiveQL 的所有資料。您可以估計查詢執行需要多長時間，如下所示：

「21,474,836,480 / 409,600 = 52,429 秒 = 14.56 小時」

在此案例中，DynamoDB 資料夾是瓶頸。這將無助於新增更多 Amazon EMR 節點，因為 Hive 輸送量限制為每秒只有 409,600 個位元組。減少 `SELECT` 陳述式所需時間的唯一方法，是增加 DynamoDB 資料表佈建的讀取容量。

您可以執行類似的計算，估計將大量載入資料帶入映射至 DynamoDB 資料表之 Hive 外部資料表所需的時間。決定每個項目所需的寫入容量單位總數 (小於 1KB = 1、1-2KB = 2，以此類推)，並將其乘以要載入的項目數。這將為您提供所需的寫入容量單位數目。將該數目除以每秒配置的寫入容量單位數目。這將產生載入資料表所需的秒數。

您應該定期監控資料表的 CloudWatch 指標。如需在 DynamoDB 主控台取得快速概觀，請選擇您的資料表，然後選擇 Metrics (指標) 索引標籤。您可以從此處檢視已耗用的讀取和寫入容量單位，以及已限流的讀取和寫入請求。

讀取容量

Amazon EMR 會根據資料表佈建輸送量設定，管理您的 DynamoDB 資料表的請求負載。不過，如果在任務輸出中發現大量的 ProvisionedThroughputExceeded 訊息，您可以調整預設讀取率。若要執行此操作，您可以修改 dynamodb.throughput.read.percent 組態變數。您可以使用 SET 命令在 Hive 命令提示字元中設定此變數：

```
SET dynamodb.throughput.read.percent=1.0;
```

此變數僅持續用於目前的 Hive 工作階段。如果您退出 Hive 並於之後返回查看，dynamodb.throughput.read.percent 會傳回其預設值。

dynamodb.throughput.read.percent 的值可能介於 0.1 及 1.5 (含) 之間。0.5 代表預設讀取率，表示 Hive 會嘗試使用資料表一半的讀取容量。如果將值增加到 0.5 以上，Hive 會增加請求率；如果將值減少到 0.5 以下，則會降低讀取請求率。(根據不同因素，例如在 DynamoDB 資料表中是否有統一金鑰分佈等，實際讀取率可能會有所不同。)

如果您注意到 Hive 經常耗盡資料表的佈建讀取容量，或者如果讀取請求限流過多，請嘗試將 dynamodb.throughput.read.percent 減少到 0.5 以下。如果資料表有足夠的讀取容量，並且您想要提升 HiveQL 作業的回應能力，則可以將值設定在 0.5 以上。

寫入容量

Amazon EMR 會根據資料表佈建輸送量設定，管理您的 DynamoDB 資料表的請求負載。不過，如果在任務輸出中發現大量的 ProvisionedThroughputExceeded 訊息，您可以調整預設寫入率。若要執行此操作，您可以修改 dynamodb.throughput.write.percent 組態變數。您可以使用 SET 命令在 Hive 命令提示字元中設定此變數：

```
SET dynamodb.throughput.write.percent=1.0;
```

此變數僅持續用於目前的 Hive 工作階段。如果您退出 Hive 並於之後返回查看，dynamodb.throughput.write.percent 會傳回其預設值。

dynamodb.throughput.write.percent 的值可能介於 0.1 及 1.5 (含) 之間。0.5 代表預設寫入率，表示 Hive 會嘗試使用資料表一半的寫入容量。如果將值增加到 0.5 以上，Hive 會增加請求率；如果將值減少到 0.5 以下，則會降低寫入請求率。(根據不同因素，例如在 DynamoDB 資料表中是否有統一金鑰分佈等，實際寫入率可能會有所不同。)

如果您注意到 Hive 經常耗盡資料表的佈建寫入容量，或者如果寫入請求限流過多，請嘗試將 `dynamodb.throughput.write.percent` 減少到 `0.5` 以下。如果資料表有足夠的容量，並且您想要提升 HiveQL 作業的回應能力，則可以將值設定在 `0.5` 以上。

在使用 Hive 將資料寫入 DynamoDB 時，請確認寫入容量單位數大於叢集中的映射器數量。例如，考慮使用由 10 個 m1.xlarge 節點的 Amazon EMR 叢集。此 m1.xlarge 節點類型提供 8 個映射器任務，因此叢集總共有 80 個映射器 (10 × 8)。如果 DynamoDB 資料表的寫入容量單位少於 80 個，則 Hive 寫入作業可能會耗用該資料表的所有寫入輸送量。

若要判斷 Amazon EMR 節點類型的映射器數目，請參閱《Amazon EMR 開發人員指南》中的任務組態。

如需映射器的詳細資訊，請參閱 調整映射器。

調整映射器

Hive 啟動 Hadoop 任務時，此任務會由一或多個映射器任務處理。假設 DynamoDB 資料表擁有足夠的輸送容量，您可以修改叢集中的映射器數目，潛在地改善效能。

> **Note**
>
> 在 Hadoop 任務中使用的映射器任務數目會受到輸入分割影響，其中 Hadoop 會將資料細分成邏輯區塊。如果 Hadoop 未執行足夠的輸入分割，則您的寫入作業可能會無法使用 DynamoDB 資料表中所有可用的寫入輸送量。

增加映射器的數目

Amazon EMR 中的每個映射器都有每秒 1 MiB 的最大讀取率。叢集中的映射器數目視叢集中的節點大小而定。(如需節點大小和每個節點映射器數目的相關資訊，請參閱《Amazon EMR 開發人員指南》中的任務組態。)

如果您的 DynamoDB 資料表擁有足夠的讀取輸送容量，您可以執行下列其中一項操作來嘗試增加映射器的數目：

- 增加叢集中節點的大小。例如，如果您的叢集正在使用 m1.large 節點 (每個節點有三個映射器)，則可以嘗試升級至 m1.xlarge 節點 (每個節點有八個映射器)。
- 增加叢集中節點的數目。例如，如果您有 m1.xlarge 節點的三節點叢集，則可用的映射器總共有 24 個。如果將叢集 (擁有相同類型的節點) 大小加倍，則會有 48 個映射器。

您可以使用 AWS 管理主控台 來管理叢集中的節點大小或數量。(您可能需要重新啟動叢集，才能讓這些變更生效。)

另一種增加映射器數目的方式，便是修改 mapred.tasktracker.map.tasks.maximum Hadoop 組態參數。(這是 Hadoop 參數，而非 Hive 參數。您無法從命令提示字元以互動方式加以修改。) 若想增加 mapred.tasktracker.map.tasks.maximum 的值，您只需增加映射器的數目，無需增加節點大小或數量。不過，如果您設定的值太高，叢集節點可能會耗盡記憶體。

在第一次啟動 Amazon EMR 叢集時，您可以將 mapred.tasktracker.map.tasks.maximum 設定為自舉動作。如需詳細資訊，請參閱《Amazon EMR 管理指南》中的 (選用) 建立引導操作來安裝其他軟體。

減少映射器的數目

如果使用 SELECT 陳述式從映射至 DynamoDB 的外部 Hive 資料表中選取資料，Hadoop 任務可視需要使用任意數量的任務，最多可達叢集中的映射器數目上限。在此案例中，長時間執行的 Hive 查詢可能會消耗 DynamoDB 資料表的所有佈建讀取容量，對其他使用者造成負面影響。

您可以使用 dynamodb.max.map.tasks 參數來設定映射任務的上限：

```
SET dynamodb.max.map.tasks=1
```

此數值必須等於或大於 1。在 Hive 處理查詢時，從 DynamoDB 資料表讀取時，產生的 Hadoop 作業使用不會超過 dynamodb.max.map.tasks。

其他主題

下列是使用 Hive 存取 DynamoDB 來調整應用程式的一些更多方法。

Retry duration (重試持續時間)

依預設，如果 Hive 未在兩分鐘內從 DynamoDB 傳回任何結果，Hive 將會重新執行 Hadoop 任務。您可以藉由修改 dynamodb.retry.duration 參數來調整此間隔：

```
SET dynamodb.retry.duration=2;
```

此值必須是非零的整數，代表重試間隔中的分鐘數。dynamodb.retry.duration 預設值為 2 (分鐘)。

平行資料請求

針對單一資料表的多個資料請求，無論是來自一名以上的使用者還是一個以上的應用程式，都會耗盡讀取佈建輸送量，降低效能。

處理持續時間

DynamoDB 中的資料一致性取決於每個節點讀取和寫入操作的順序。當 Hive 查詢正在進行中時，另一個應用程式可以將新資料載入 DynamoDB 資料表，或修改或刪除現有的資料。在此案例中，Hive 查詢的結果有可能無法反映執行查詢的過程中所做的資料變更。

請求時間

當對 DynamoDB 資料表的需求較低時，排程存取 DynamoDB 資料表的 Hive 查詢可改善效能。例如，若您大多數的應用程式使用者住在舊金山，您可以選擇在大多數的使用者仍在睡眠中的太平洋標準時間 (PST) 早上 4 點匯出每日資料，而不更新您 DynamoDB 資料庫中的紀錄。

將 DynamoDB 與 Amazon S3 整合

Amazon DynamoDB 匯入與匯出功能提供簡易且有效率的方式，可在 Amazon S3 和 DynamoDB 資料表之間移動資料，無需撰寫任何程式碼。

DynamoDB 匯入與匯出功能可協助您移動、轉換和複製 DynamoDB 資料表帳戶。您可以從 S3 來源匯入，然後將 DynamoDB 資料表資料匯出到 Amazon S3，並使用 AWS 服務 (例如 Athena、Amazon SageMaker AI 和 AWS Lake Formation) 來分析您的資料，以擷取可行見解。您也可以將資料直接匯入新的 DynamoDB 資料表，以便建置具有個位數毫秒的規模效能之新應用程式、促進資料表和帳戶之間的資料共用，並簡化災難復原和業務持續性計畫。

主題

- 從 Amazon S3 匯入 DynamoDB 資料：運作方式
- DynamoDB 資料匯出至 Amazon S3：運作方式

從 Amazon S3 匯入 DynamoDB 資料：運作方式

若要將資料匯入 DynamoDB，您的資料必須位於 Amazon S3 儲存貯體中，且為 CSV、DynamoDB JSON 或 Amazon Ion 格式。資料可壓縮為 ZSTD 或 GZIP 格式，也可以直接匯入為未壓縮格式。來源資料可以是單一 Amazon S3 物件，或使用相同字首的多個 Amazon S3 物件。

您的資料將匯入到新的 DynamoDB 資料表中，該資料表將在您發起匯入請求時建立。您可以使用次要索引建立此資料表，然後在匯入完成後立即查詢並更新所有主要索引和次要索引的資料。您也可以在匯入完成後新增全域資料表複本。

> **Note**
>
> 在 Amazon S3 匯入流程期間，DynamoDB 會建立一個新的目標資料表，並將其匯入。此功能目前不支援匯入至現有資料表。

從 Amazon S3 匯入不會耗用新資料表的寫入容量，因此您不需要佈建任何額外容量即可將資料匯入 DynamoDB。資料匯入定價取決於 Amazon S3 中來源資料的未壓縮大小，該資料會在匯入時進行處理。由於格式化或來源資料的不一致，導致無法載入資料表中的項目，也會在匯入流程中計費。如需詳細資訊，請參閱 Amazon DynamoDB 定價。

您可以從不同帳戶所擁有的 Amazon S3 儲存貯體匯入資料，前提是您有讀取該儲存貯體的正確許可。新資料表也可能位於與來源 Amazon S3 儲存貯體不同的區域。如需詳細資訊，請參閱 Amazon Simple Storage Service 設定和許可。

匯入時間與 Amazon S3 中的資料特性直接相關。這包括資料大小、資料格式、壓縮方法、資料分佈的一致性、Amazon S3 物件數量以及其他相關變數。具體而言，索引鍵均勻分佈的資料集，匯入速度會比偏斜的資料集更快。例如，若您的次要索引鍵是以月份分割，而您的資料全部來自十二月，那麼匯入資料就會需要更久時間。

與索引鍵相關聯的屬性在基礎資料表中應為唯一。如有任何索引鍵並非唯一，則匯入程序會覆寫相關聯的項目，直到僅留下最後一次覆寫。例如，若主索引鍵是月份，而有多個項目設定為 9 月份，則每個新項目都會覆寫先前寫入的項目，並僅保留一個「月份」主索引鍵設定為 9 月的項目。此情況下，匯入資料表描述中處理的物件數，與目標資料表中的物件數量不相符。

AWS CloudTrail 會記錄資料表匯入的所有主控台和 API 動作。如需詳細資訊，請參閱使用 記錄 DynamoDB 操作 AWS CloudTrail。

以下影片介紹如何直接從 Amazon S3 匯入到 DynamoDB。

從 Amazon S3 匯入

主題

- 請求在 DynamoDB 中匯入資料表
- DynamoDB 的 Amazon S3 匯入格式
- 匯入格式配額與驗證

- 將 Amazon S3 匯入到 DynamoDB 的最佳實務

請求在 DynamoDB 中匯入資料表

DynamoDB 匯入功能可讓您將 Amazon S3 儲存貯體中的資料匯入至新的 DynamoDB 資料表。您可以使用 DynamoDB 主控台、CLI、CloudFormation 或 DynamoDB API 請求資料表匯入。

如果您想要使用 AWS CLI，您必須先進行設定。如需詳細資訊，請參閱存取 DynamoDB。

> **Note**
>
> - 匯入資料表功能會與多個不同的 AWS 服務互動，例如 Amazon S3 和 CloudWatch。在開始匯入之前，請確定調用匯入 API 的使用者或角色具有該功能所依賴之所有服務和資源權限。
> - 請不要在匯入進行時修改 Amazon S3 物件，因為這可能會導致操作失敗或遭到取消。
>
> 如需如何處理錯誤和疑難排解的詳細資訊，請參閱 匯入格式配額與驗證。

主題

- 設定 IAM 許可
- 使用 請求匯入 AWS 管理主控台
- 取得 中過去匯入的詳細資訊 AWS 管理主控台
- 使用 請求匯入 AWS CLI
- 取得 中過去匯入的詳細資訊 AWS CLI

設定 IAM 許可

您可以從您擁有讀取許可的任何 Amazon S3 儲存貯體匯入資料。來源儲存貯體無需與來源資料表位於相同區域或擁有相同的擁有者。Your AWS Identity and Access Management (IAM) 必須包含來源 Amazon S3 儲存貯體的相關動作，以及提供偵錯資訊所需的 CloudWatch 許可。以下所示為政策範例。

JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Sid": "AllowDynamoDBImportAction",
      "Effect": "Allow",
      "Action": [
        "dynamodb:ImportTable",
        "dynamodb:DescribeImport"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table*"
    },
    {
      "Sid": "AllowS3Access",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::your-bucket/*",
        "arn:aws:s3:::your-bucket"
      ]
    },
    {
      "Sid": "AllowCloudwatchAccess",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:PutRetentionPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group/aws-dynamodb/*"
    },
    {
      "Sid": "AllowDynamoDBListImports",
      "Effect": "Allow",
      "Action": "dynamodb:ListImports",
      "Resource": "*"
    }
  ]
}
```

Amazon S3 許可

從另一個帳戶擁有的 Amazon S3 儲存貯體來源上開始匯入時，請確保該角色或使用者可以存取 Amazon S3 物件。您可以執行 Amazon S3 `GetObject` 命令並使用憑證以確認這點。使用 API 時，Amazon S3 儲存貯體擁有者參數會預設為目前使用者的帳戶 ID。對於跨帳戶匯入，請確定此參數已正確填入儲存貯體擁有者的帳戶 ID。下列程式碼為來源帳戶中 Amazon S3 儲存貯體政策的範例。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {"Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

AWS Key Management Service

建立新資料表進行匯入時，如果您選取非 DynamoDB 擁有的靜態加密金鑰，則必須提供操作使用客戶受管金鑰加密的 DynamoDB 資料表所需的 AWS KMS 許可。如需詳細資訊，請參閱授權使用您的 AWS KMS 金鑰。如果 Amazon S3 物件使用伺服器端加密 KMS (SSE-KMS) 加密，請確保啟動匯入的角色或使用者具有使用 AWS KMS 金鑰解密的存取權。此功能不支援客戶提供加密金鑰 (SSE-C) 加密的 Amazon S3 物件。

CloudWatch 許可

發起匯入的角色或使用者，將需要與匯入相關聯的日誌群組和日誌串流之建立和管理員權限。

使用 請求匯入 AWS 管理主控台

以下範例示範如何使用 DynamoDB 主控台將現有資料匯入名為 `MusicCollection` 的新的資料表。

請求資料表匯入

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Import from S3 (從 S3 匯入)。
3. 在出現的頁面上，選取 Import from S3 (從 S3 匯入)。
4. 選擇 Import from S3 (從 S3 匯入)。
5. 在來源 S3 URL 中，輸入 Amazon S3 來源 URL。

   如果您擁有來源儲存貯體，請選擇瀏覽 S3 進行搜尋。或者，以下列格式輸入儲存貯體的 URL：`s3://bucket/prefix`。`prefix` 是 Amazon S3 金鑰字首。它可以是您要匯入的 Amazon S3 物件名稱，或是包含您想要匯入之所有 Amazon S3 物件共用的金鑰字首。

   > **Note**
   >
   > 您不能使用與 DynamoDB 匯出請求相同的字首。匯出功能會為所有匯出建立資料夾結構和資訊清單檔案。如果您使用相同的 Amazon S3 路徑，則會導致錯誤。
   > 反之，將匯入指向資料夾，其中包含來自該特定匯出的資料。在此情況下，正確路徑的格式將為 `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/data/`，其中 `XXXXXXXX-XXXXXX` 是匯出 ID。您可以在匯出 ARN 中找到匯出 ID，其格式如下：`arn:aws:dynamodb:<Region>:<AccountID>:table/<TableName>/export/<XXXXXXXX-XXXXXX>`。例如 `arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/export/01234567890123-a1b2c3d4`。

6. 指定您是否為 S3 bucket owner (S3 儲存貯體擁有者)。如果來源儲存貯體由不同的帳戶擁有，請選取不同的 AWS 帳戶。然後輸入儲存貯體擁有者的帳戶 ID。
7. 在 Import file compression (匯入檔案壓縮)，請選取 No compression (無壓縮)、GZIP (GZIP) 或者 ZSTD (ZSTD)，視情況而定。
8. 選取適當的匯入檔案格式。選項包括 DynamoDB JSON (DynamoDB JSON)、Amazon Ion (Amazon Ion) 或者CSV (CSV)。如果您選擇 CSV (CSV)，您將有兩個額外的選項：CSV header (CSV 標頭) 和 CSV delimiter character (CSV 分隔符號)。

   針對 CSV header (CSV 標頭)，請選擇是否要從檔案的第一行取得標頭或自訂。如果您選擇Customize your headers (自訂標頭)，您可以指定您要匯入的標頭值。用此方法指定的 CSV 標頭區分大小寫，且應包含目標資料表的索引鍵。

針對 CSV delimiter character (CSV 分隔符號)，您可以設置用以分隔項目的字元。根據預設，會選取逗號。如果您選擇 Custom delimiter character (自訂分隔符號字元)，分隔符號必須符合正則表達式模式：`[,;:|\t ]`。

9. 選擇 Next (下一步) 按鈕，然後選擇建立以儲存資料的新資料表選項。

> **Note**
>
> 「主索引鍵」和「排序索引鍵」必須符合檔案中的屬性，否則匯入將會失敗。屬性區分大小寫。

10. 選擇 Next (下一步) 以再次檢視您的匯入選項，然後按一下 Import (匯入)，以啟動匯入任務。首先，您會看到「資料表」中列出的新資料表狀態為「建立中」。目前無法存取資料表。
11. 匯入完成後，狀態將顯示為「啟用」，您可以開始使用資料表。

取得 中過去匯入的詳細資訊 AWS 管理主控台

您可以按一下導覽側列中的 Import from S3 (從 S3 匯入)，然後選取 Imports (匯入) 索引標籤，找到您過去曾執行的匯入任務資訊。匯入面板包含您在過去 90 天內建立的所有匯入清單。選取 Imports (匯入) 索引標籤中所列出任務的 ARN，即可擷取該匯入的相關資訊，包括您選擇的任何進階組態設定。

使用 請求匯入 AWS CLI

下列範例會從名為字首儲存貯體的 S3 儲存貯體，將 CSV 格式的資料匯入到名為 target-table 的新資料表。

```
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**
>
> 如果您選擇使用受 AWS Key Management Service (AWS KMS) 保護的金鑰加密匯入，則金鑰必須與目的地 Amazon S3 儲存貯體位於相同的區域。

取得 中過去匯入的詳細資訊 AWS CLI

您可以使用 `list-imports` 命令尋找過去曾執行的匯入任務資訊。此命令會傳回您在過去 90 天內建立的所有匯入清單。請注意，雖然匯入任務中繼資料會在 90 天後過期，也無法在此清單中找到早於該時間的任務，但 DynamoDB 不會刪除 Amazon S3 儲存貯體或匯入期間建立的任何物件。

```
aws dynamodb list-imports
```

若要擷取特定匯入任務的詳細資訊 (包括任何進階組態設定)，請使用 `describe-import` 命令。

```
aws dynamodb describe-import \
    --import-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/exp
```

DynamoDB 的 Amazon S3 匯入格式

DynamoDB 可以匯入三種格式的資料：CSV、DynamoDB JSON 和 Amazon Ion。

主題

- CSV
- DynamoDB JSON
- Amazon Ion

CSV

CSV 格式的檔案由以換行符分隔的多個項目組成。根據預設，DynamoDB 會將匯入檔案的第一行解讀為標頭，並預期欄會以逗號分隔。只要標頭符合檔案中的欄數，您也可以定義欲套用的標頭。如果標頭定義明確，檔案的第一行將會匯入為值。

> **Note**
>
> 從 CSV 檔案匯入時，除了基礎資料表的雜湊範圍與索引鍵和次要索引外，所有欄都會匯入為 DynamoDB 字串。

轉義雙逸出

CSV 檔案中存在的任何雙引號字元都必須逸出。如果未經逸出 (如下列範例)，則匯入將失敗：

```
id,value
"123",Women's Full "Length" Dress
```

如果使用兩組雙引號逸出，則相同的匯入將成功：

```
id,value
"""123""","Women's Full ""Length"" Dress"
```

當文字經過正確逸出和匯入之後，就會如原始 CSV 檔案中一樣顯示：

```
id,value
"123",Women's Full "Length" Dress
```

匯入異質項目類型

您可以使用單一 CSV 檔案，將不同的項目類型匯入一個資料表。定義標頭列，其中包含所有項目類型的屬性，並為不適用於指定項目的屬性將資料欄保留空白。從匯入的項目省略空資料欄，而不是儲存為空字串。

```
PK,SK,EntityType,Name,Email,OrderDate,Amount,ProductName,Quantity
USER#1,PROFILE,User,Alice,alice@example.com,,,,
USER#1,ORDER#2024-01-15,Order,,,2024-01-15,99.99,,
USER#1,ORDER#2024-02-10,Order,,,2024-02-10,149.50,,
PRODUCT#101,METADATA,Product,,,,,Laptop,50
PRODUCT#102,METADATA,Product,,,,,Mouse,200
USER#2,PROFILE,User,Bob,bob@example.com,,,,
USER#2,ORDER#2024-01-20,Order,,,2024-01-20,75.00,,
PRODUCT#103,METADATA,Product,,,,,Keyboard,150
USER#3,PROFILE,User,Charlie,charlie@example.com,,,,
PRODUCT#104,METADATA,Product,,,,,Monitor,30
```

在此範例中，使用者設定檔、訂單和產品共用相同的資料表。每個項目類型只會使用與其相關的資料欄。

DynamoDB JSON

DynamoDB JSON 格式的檔案可能由多個項目物件組成。每個個別物件均採用 DynamoDB 的標準封送處理 JSON 格式，而新行則用作項目分隔符號。新增功能中，預設支援從時間點的匯出可作為匯入來源。

> **Note**
>
> 新行用作 DynamoDB JSON 格式檔案的項目分隔符號，不應用於項目物件。

```
{"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 是輸入豐富、自行描述、階層資料序列化格式，旨在處理快速開發、解耦及日常遇到的效率挑戰等問題，同時設計大規模的服務導向架構。

將資料匯入為 Ion 格式時，DynamoDB 資料表中使用的 DynamoDB 資料類型會映射至 Ion 資料類型。

序號	Ion 至 DynamoDB 資料類型轉換	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	Set (SS, NS, BS)

序號	Ion 至 DynamoDB 資料類型轉換	B
	_SS, $dynamodb_NS, or $dynamodb_BS)	
7	list	List
8	struct	Map

Ion 檔案中的項目以新行分隔。每行都以 Ion 版本標記開頭，後面接著 Ion 格式的項目。

> **Note**
>
> 下列範例中，為了便於閱讀，Ion 格式檔案的項目採用多行格式。

```
$ion_1_0
[
  {
    Item:{
      Authors:$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"
    }
  }
]
```

匯入格式配額與驗證

匯入配額

Amazon S3 的 DynamoDB 匯入可支援多達 50 個並行匯入任務，在 us-east-1、us-west-2 和 eu-west-1 區域的匯入來源物件大小總計達 15TB。在所有其他區域中，最多支援 50 個並行匯入任務，大小總計為 1TB。每個匯入任務在所有區域中最多可以使用 50,000 個 Amazon S3 物件。這些預設配額會套用至每個帳戶。如果您認為需要修改這些配額，請聯絡您的客戶團隊，我們會根據個別情況進行考慮。如需 DynamoDB 限制的詳細資訊，請參閱 Service Quotas。

驗證錯誤

在匯入流程期間，DynamoDB 可能會在剖析資料時遇到錯誤。DynamoDB 會針對每個錯誤發出一份 CloudWatch 日誌，並保留遇到的錯誤總數計數。如果 Amazon S3 物件本身格式錯誤或其內容無法形成 DynamoDB 項目，我們可能會略過處理物件的剩餘部分。

> **Note**
>
> 如果 Amazon S3 資料來源有多個共用相同索引鍵的項目，這些項目將會覆寫，直到僅留下一個項目為止。這看起來像是匯入了 1 個項目而忽略了其他項目。重複的項目將以隨機順序覆寫，不會計為錯誤，也不會發出到 CloudWatch 日誌中。

匯入完成後，您可以查看匯入的項目總數、錯誤總數，以及處理的項目總數。如需進一步的疑難排解，您也可以檢查所匯入項目的總大小與所處理資料的總大小。

匯入錯誤分為三類：API 驗證錯誤、資料驗證錯誤和組態錯誤。

API 驗證錯誤

API 驗證錯誤是來自同步 API 的項目層級錯誤。常見原因是許可問題、缺少必要參數以及參數驗證失敗。`ImportTable` 請求所擲回的例外狀況中，包含 API 呼叫失敗原因的詳細資訊。

驗證錯誤

資料驗證錯誤可能發生在項目層級或檔案層級。匯入期間，會先根據 DynamoDB 規則驗證項目，然後再匯入目標資料表。當項目驗證失敗且未匯入時，匯入任務會略過該項目，並繼續下一個項目。工作結束時，匯入狀態會設定為「失敗」，伴隨 FailureCode、ItemValidationError 以及 FailureMessage：「部分項目驗證檢查失敗且未匯入，如需詳細資訊，請檢查 CloudWatch 錯誤日誌。」

造成資料驗證錯誤的常見原因包括物件無法剖析、物件格式不正確 (輸入指定 DYNAMODB_JSON，但物件不在 DYNAMODB_JSON 中)，以及結構描述與指定來源資料表的金鑰不相符。

組態錯誤

組態錯誤通常是因為權限驗證而導致的工作流程錯誤。匯入工作流程會在接受請求之後檢查某些權限。如果在呼叫 Amazon S3 或 CloudWatch 等任何必要的相依性時發生問題，則流程會將匯入狀態標示為「失敗」。所以 `failureCode` 和 `failureMessage` 指向失敗原因。在適用的情況下，失敗訊息還包含請求 ID，您可以用來調查 CloudTrail 中失敗的原因。

常見的組態錯誤包括 Amazon S3 儲存貯體的 URL 錯誤，而且沒有存取 Amazon S3 儲存貯體、CloudWatch Logs 和用於解密 Amazon S3 物件的 AWS KMS 金鑰的許可。如需詳細資訊，請參閱使用與資料金鑰。

驗證 Amazon S3 物件

若要驗證來源 S3 物件，請執行下列步驟。

1. 驗證資料格式與壓縮類型

   - 請確認指定字首下的所有相符 Amazon S3 物件都具有相同的格式 (DYNAMODB_JSON、DYNAMODB_ION、CSV)

- 請確認指定字首下的所有相符 Amazon S3 物件都以相同的方式壓縮 (GZIP、ZSTD、無)

> **Note**
>
> Amazon S3 物件不需要具有對應的副檔名 (.csv / .json / .ion / .gz / .zstd 等)，因為應以 ImportTable 呼叫中指定的輸入格式優先。

2. 驗證匯入資料是否符合所需的資料表結構描述

- 請確定來源資料中的每個項目都有主索引鍵。匯入的排序索引鍵為選用。
- 請確定與主索引鍵和排序索引鍵相關聯的屬性類別符合 Table 和 GSI 結構描述中的屬性類別，如資料表建立參數所指定

疑難排解

CloudWatch 日誌

對於匯入失敗的工作，詳細錯誤訊息會張貼到 CloudWatch 日誌。若要存取這些日誌，請先從輸出擷取 ImportArn，然後使用此命令進行描述匯入：

```
aws dynamodb describe-import --import-arn arn:aws:dynamodb:us-east-1:ACCOUNT:table/
target-table/import/01658528578619-c4d4e311
}
```

輸出範例：

```
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."
    }
}
```

從上述回應擷取日誌群組並匯入 ID，並用其擷取錯誤日誌。匯入 ID 是 ImportArn 欄位的最後一個路徑元素。日誌群組的名稱為 /aws-dynamodb/imports。錯誤日誌串流的名稱為 import-id/error。此例子中，為 01658528578619-c4d4e311/error。

缺少項目中的金鑰 pk

如果來源 S3 物件不包含作為參數的主索引鍵，匯入將會失敗。例如，當您將匯入的主索引鍵定義為欄名稱「pk」時。

```
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"}'
```

來源物件 import-test.csv 中缺少欄「pk」，其中包含下列內容：

```
title,artist,year_of_release
The Dark Side of the Moon,Pink Floyd,1973
```

由於資料來源中缺少主索引鍵，因此這項匯入會因為項目驗證錯誤而失敗。

CloudWatch 錯誤日誌範例：

```
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"
}
```

此錯誤日誌表示「一個或多個參數值無效：缺少項目中的金鑰 pk」。由於此匯入工作失敗，現在存在資料表「target-table」但其為空白，因為沒有匯入任何項目。已處理第一個項目，物件的項目驗證失敗。

若要修正此問題，請先將「target-table」刪除 (若不再需要)。然後使用來源物件中存在的主索引鍵欄名稱，或將來源資料更新為：

```
pk,title,artist,year_of_release
Albums::Rock::Classic::1973::AlbumId::ALB25,The Dark Side of the Moon,Pink Floyd,1973
```

目標資料表存在

當您啟動匯入工作並收到回應時，如下所示：

```
An error occurred (ResourceInUseException) when calling the ImportTable operation:
 Table already exists: target-table
```

若要修正此錯誤，您必須選擇不存在的資料表名稱，然後再一次匯入。

指定的儲存貯體不存在

如果來源儲存貯體不存在，匯入將失敗，並會在 CloudWatch 中記錄錯誤訊息詳細資料。

描述匯入範例：

```
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)"
}
}
```


此 `FailureCode` 為 `S3NoSuchBucket`，其中 `FailureMessage` 包含請求 ID 等詳細資訊以及擲出錯誤服務。由於在將資料匯入資料表之前就已發現錯誤，因此不會建立新的 DynamoDB 資料表。部分情況下，若資料開始匯入後才遇到這些錯誤，就會保留含有部分匯入資料的資料表。

若要修正此錯誤，請確定來源 Amazon S3 儲存貯體存在，然後重新啟動匯入程序。

將 Amazon S3 匯入到 DynamoDB 的最佳實務

以下為將資料從 Amazon S3 匯入到 DynamoDB 的最佳實務。

保持在 50,000 個 S3 物件的限制以下

每個匯入工作最多支援 50,000 個 S3 物件。如果您的資料集包含超過 50,000 個物件，請考慮將它們合併為較大的物件。

避免過大的 S3 物件

S3 物件會平行匯入。擁有眾多中型 S3 物件允許平行執行，而不會產生過大負荷。對於 1 KB 以下的項目，請考慮在每個 S3 物件中放置 400 萬個項目。如果您的平均項目大小較大，請在每個 S3 物件中按比例放置較少的項目。

隨機化已排序的資料

如果 S3 物件以排序順序保存資料，它會建立滾動的經常性分割區。這種情況是由其中一個分割區接收所有活動，接著是下一個分割區，依此類推。按排序順序的資料的定義是 S3 物件中的循序項目，將在匯入期間寫入相同目標分割區。資料以排序順序排列的一個常見情況是 CSV 檔案，其中項目會依分割區索引鍵排序，以便讓重複的項目共用相同的分割區索引鍵。

若要避免滾動的經常性分割區，我們建議您在這些情況下隨機化順序。這可以透過散佈寫入操作來改善效能。如需詳細資訊，請參閱在將資料上傳至 DynamoDB 時有效分配寫入活動。

壓縮資料，將 S3 物件總大小保持在區域限制以下

在從 S3 匯入程序中，要匯入之 S3 物件資料的總大小有限制。us-east-1、us-west-2 和 eu-west-1 區域中，限制為 15 TB，在所有其他區域中為 1 TB。此限制是以原始 S3 物件大小為基礎。

壓縮可讓更多原始資料符合限制。如果單獨壓縮不足以讓匯入符合限制範圍，您也可以聯絡 AWS Premium Support 請求提高配額。

注意項目大小如何影響效能

如果您的平均項目大小非常小 (低於 200 位元組)，匯入程序可能會比較大的項目大小更久。

請勿在作用中匯入期間修改 S3 物件

確保來源 S3 物件在匯入操作進行時保持不變。如果在匯入期間修改 S3 物件，操作將會失敗，並顯示錯誤碼`ObjectModifiedInS3DuringImport`和訊息「無法匯入 S3 物件，因為它被覆寫。」

如果您遇到此錯誤，請使用 S3 物件的穩定版本重新啟動匯入操作。若要避免此問題，請等待目前的匯入完成，然後再變更來源檔案。

考慮在沒有全域次要索引的情形下匯入

匯入任務的持續時間可能取決於是否存在一或多個全域次要索引 (GSI)。如果您計劃使用低基數的分割區索引鍵來建立索引，那麼如果將索引建立延遲到匯入任務完成之後 (而不是包含在匯入工作中)，匯入速度可能會更快。

> **Note**
>
> 建立 GSI 不會產生寫入費用，無論是在匯入期間或之後建立。

DynamoDB 資料匯出至 Amazon S3：運作方式

DynamoDB 匯出至 S3 是一種全受管解決方案，可將 DynamoDB 資料大規模匯出至 Amazon S3 儲存貯體。您可以使用 DynamoDB 匯出至 S3，將資料從時間點復原 (PITR) 時段內任何時間點的 Amazon DynamoDB 資料表，匯出至 Amazon S3 儲存貯體。您需在資料表上啟用 PITR，才能使用匯出功能。此功能可讓您使用 Athena、Amazon SageMaker AI AWS Glue、Amazon EMR 和 等 AWS 其他服務，對資料執行分析和複雜查詢 AWS Lake Formation。

DynamoDB 匯出至 S3 可讓您從 DynamoDB 資料表匯出完整資料和增量資料。匯出是非同步的，而且不會使用任何讀取容量單位 (RCU)，也不會影響資料表效能和可用性。支援的匯出檔案格式為 DynamoDB JSON 和 Amazon Ion 格式。您也可以將資料匯出至另一個 AWS 帳戶擁有的 S3 儲存貯體，以及匯出至不同的 AWS 區域。您的資料一律會進行端對端加密。

DynamoDB 完整匯出的費用是根據匯出完成之時間點的 DynamoDB 資料表大小 (資料表資料和本機次要索引) 來計費。DynamoDB 增量匯出的費用是根據匯出期間內從連續備份處理的資料大小來計費。增量匯出的最低費用為 10 MB。其他費用包括將匯出的資料儲存在 Amazon S3 中，以及對 Amazon S3 儲存貯體發出的 PUT 請求。如需這些費用的詳細資訊，請參閱 Amazon DynamoDB 定價和 Amazon S3 定價。

如需 Service Quotas 的詳細資訊，請參閱 資料表匯出至 Amazon S3。

主題

- 請求在 DynamoDB 中匯出資料表
- DynamoDB 資料表匯出輸出格式

請求在 DynamoDB 中匯出資料表

DynamoDB 資料表匯出可讓您將資料表資料匯出至 Amazon S3 儲存貯體，讓您使用 Athena、 AWS Glue、Amazon SageMaker AI、Amazon EMR 和 等 AWS 其他服務，對資料執行分析和複雜的查詢 AWS Lake Formation。您可以使用 AWS 管理主控台、 AWS CLI或 DynamoDB API 請求資料表匯出。

> **Note**
>
> 不支援申請者支付 Amazon S3 儲存貯體。

DynamoDB 同時支援完整匯出和增量匯出：

- 使用完整匯出時，您可以從時間點復原 (PITR) 時段內任何時間點，將資料表的完整快照匯出至 Amazon S3 儲存貯體。
- 使用增量匯出時，您可以將 PITR 時段中指定期間內變更、更新或刪除的 DynamoDB 資料表中的資料匯出至 Amazon S3 儲存貯體。

主題

- 先決條件
- 使用 請求匯出 AWS 管理主控台
- 取得 中過去匯出的詳細資訊 AWS 管理主控台
- 使用 AWS CLI 和 AWS SDKs 請求匯出
- 使用 AWS CLI 和 AWS SDKs 取得過去匯出的詳細資訊

先決條件

啟用 PITR

若要使用匯出至 S3 功能，您將需要在資料表上啟用 PITR。如需如何啟用 PITR 的詳細資訊，請參閱時間點復原。如果您請求匯出未啟用 PITR 的資料表，您的請求將會失敗，並顯示例外狀況訊息：「呼叫 `ExportTableToPointInTime` 操作時發生錯誤 (PointInTimeRecoveryUnavailableException)：資料表 'my-dynamodb-table' 未啟用時間點復原」。您只能從設定 PITR `RecoveryPeriodInDays` 內的時間點請求和匯出。

設定 S3 許可

您可以將資料表資料匯出至您擁有寫入許可的任何 Amazon S3 儲存貯體。目的地儲存貯體不需要與來源資料表擁有者位於相同 AWS 區域或擁有相同的擁有者。您的 AWS Identity and Access Management (IAM) 政策需要允許您執行 S3 動作 (`s3:AbortMultipartUpload`、 `s3:PutObject`和 `s3:PutObjectAcl`) 和 DynamoDB 匯出動作 (`dynamodb:ExportTableToPointInTime`)。以下範例政策將授予您的使用者執行匯出至 S3 儲存貯體的許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "AllowDynamoDBExportAction",
            "Effect": "Allow",
            "Action": "dynamodb:ExportTableToPointInTime",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table"
        },
        {
            "Sid": "AllowS3BucketWrites",
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

如果您需要寫入另一個帳戶中的 S3 儲存貯體，或者您沒有寫入其中的許可，則 S3 儲存貯體擁有者將需要新增儲存貯體政策，以允許您從 DynamoDB 匯出至該儲存貯體。以下是目標 S3 儲存貯體的範例政策。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
```

```
    ]
}
```

在匯出過程中撤銷這些許可會產生部分檔案。

> **Note**
>
> 如果您要匯出的目標資料表或儲存貯體使用客戶自管金鑰加密，該 KMS 金鑰的政策必須允許 DynamoDB 使用該金鑰。此許可是透過觸發匯出任務的 IAM 使用者/角色授予。如需有關加密及其最佳實務的詳細資訊，請參閱 DynamoDB 如何使用 AWS KMS 以及使用自訂 KMS 金鑰。

使用 請求匯出 AWS 管理主控台

以下範例示範如何使用 DynamoDB 主控台匯出名為 `MusicCollection` 的現有資料表。

> **Note**
>
> 此程序假設您已經啟用時間點復原。若要為 `MusicCollection` 資料表啟用該功能，進入資料表 Overview (概觀) 索引標籤的 Table details (資料表詳細資訊) 部分，然後為 Point-in-time recovery (時間點復原) 選擇 Enable (啟用)。

請求資料表匯出

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Exports to S3 (匯出至 S3)。
3. 選取匯出至 S3 按鈕。
4. 選擇來源資料表和目的地 S3 儲存貯體。如果您的帳戶擁有該目的地儲存貯體，您可以使用 Browse S3 (瀏覽 S3) 按鈕尋找它。如果不是，請使用 `s3://bucketname/prefix format.` 輸入儲存貯體的 URL；**`prefix`** 是有助於您整理目的地儲存貯體的選用資料夾。
5. 選擇完整匯出或增量匯出。完整匯出會輸出資料表在您所指定時間點的完整資料表快照。增量匯出會輸出在指定匯出期間對資料表所做的變更。您的輸出會經過壓縮，因此只會包含匯出期間內項目的最終狀態。即使項目在同一匯出期間內有多個更新，該項目也只會在匯出中出現一次。

Full export

1. 選取您要匯出完整資料表快照的來源時間點。這個時間點可以是 PITR 時段內的任何時間點。或者，您可以選取目前時間以匯出最新快照。
2. 對於匯出的檔案格式，請選擇 DynamoDB JSON 和 Amazon Ion 兩者之一。根據預設，您的資料表會以 DynamoDB JSON 格式在時間點復原時段中的最近可還原時間匯出，並使用 Amazon S3 金鑰 (SSE-S3) 進行加密。您可以視需要變更這些匯出設定。

> **Note**
>
> 如果您選擇使用受 AWS Key Management Service (AWS KMS) 保護的金鑰來加密匯出，則金鑰必須位於與目的地 S3 儲存貯體相同的區域。

Incremental export

1. 選取您要匯出增量資料的匯出期間。選擇在 PITR 時段內的開始時間。匯出期間必須至少為 15 分鐘，且不超過 24 小時。匯出期間的開始時間包含在內，結束時間則不包含在內。
2. 選擇絕對模式或相對模式。
   a. 絕對模式會依您指定的期間匯出增量資料。
   b. 相對模式會在相對於匯出任務提交時間的匯出期間匯出增量資料。
3. 對於匯出的檔案格式，請選擇 DynamoDB JSON 和 Amazon Ion 兩者之一。根據預設，您的資料表會以 DynamoDB JSON 格式在時間點復原時段中的最近可還原時間匯出，並使用 Amazon S3 金鑰 (SSE-S3) 進行加密。您可以視需要變更這些匯出設定。

> **Note**
>
> 如果您選擇使用受 AWS Key Management Service (AWS KMS) 保護的金鑰來加密匯出，則金鑰必須位於與目的地 S3 儲存貯體相同的區域。

4. 對於匯出檢視類型，請選取新舊映像或僅限新映像。新映像提供項目的最新狀態。舊映像提供項目在指定的「開始日期和時間」之前的狀態。預設設定為新舊映像。如需新映像和舊映像的詳細資訊，請參閱 增量匯出輸出。

6. 選擇匯出以開始。

匯出的資料在交易上不一致。您的交易操作可能在兩次匯出輸出之間未完成。您可能看到經交易操作修改的項目子集反映在匯出中，而同一交易中的另一個修改內容子集卻未反映在相同的匯出請求中。不過，匯出最後會是一致的。如果交易在匯出期間未完成，交易的其餘部分保證會在下一次連續匯出中出現，而且不會重複。用於匯出的期間是以內部系統時鐘為基礎，且可能與您的應用程式本機時鐘相差一分鐘。

取得 中過去匯出的詳細資訊 AWS 管理主控台

按一下導覽側邊欄中的匯出至 S3 區段，即可找到您過去曾執行的匯出任務的相關資訊。此區段包含您在過去 90 天內建立的所有匯出清單。選取匯出索引標籤中所列出任務的 ARN 以擷取該匯出的相關資訊，包括您選擇的任何進階組態設定。請注意，雖然匯出任務中繼資料會在 90 天後過期，也無法在此清單中找到早於該時間的任務，但只要儲存貯體政策允許，S3 儲存貯體中的物件就會保留。DynamoDB 絕不會刪除在匯出時其在您的 S3 儲存貯體中建立的任何物件。

使用 AWS CLI 和 AWS SDKs 請求匯出

下列範例示範如何將現有資料表匯出至 S3 儲存貯體。

> **Note**
>
> 此程序假設您已經啟用時間點復原。若要針對 `MusicCollection` 資料表啟用程序，請執行下列命令。
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name MusicCollection \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

完整匯出

AWS CLI

> **Note**
>
> 如果要求跨帳戶資料表匯出，請務必包含 `--s3-bucket-owner` 選項。

```
aws dynamodb export-table-to-point-in-time \
```

```
  --table-arn arn:aws:dynamodb:us-west-2:111122223333:table/MusicCollection \
  --s3-bucket ddb-export-musiccollection-9012345678 \
  --s3-prefix 2020-Nov \
  --export-format DYNAMODB_JSON \
  --export-time 1604632434 \
  --s3-bucket-owner 9012345678 \
  --s3-sse-algorithm AES256
```

Python

```
import boto3
from datetime import datetime

client = boto3.client('dynamodb')

client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:111122223333:table/TABLE',
    ExportTime=datetime(2023, 9, 20, 12, 0, 0),
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

Java

```
DynamoDbClient client = DynamoDbClient.create();

client.exportTableToPointInTime(b -> b
    .tableArn("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE")
    .exportTime(Instant.parse("2023-09-20T12:00:00Z"))
    .s3Bucket("bucket")
    .s3Prefix("prefix")
    .s3SseAlgorithm(S3SseAlgorithm.AES256)
    .exportFormat(ExportFormat.DYNAMODB_JSON));
```

.NET

```
var client = new AmazonDynamoDBClient();

await client.ExportTableToPointInTimeAsync(new ExportTableToPointInTimeRequest
{
```

```
    TableArn = "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE",
    ExportTime = new DateTime(2023, 9, 20, 12, 0, 0, DateTimeKind.Utc),
    S3Bucket = "bucket",
    S3Prefix = "prefix",
    S3SseAlgorithm = S3SseAlgorithm.AES256,
    ExportFormat = ExportFormat.DYNAMODB_JSON
});
```

JavaScript

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

const client = new DynamoDBClient();

await client.send(new ExportTableToPointInTimeCommand({
    TableArn: "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE",
    ExportTime: new Date("2023-09-20T12:00:00Z"),
    S3Bucket: "bucket",
    S3Prefix: "prefix",
    S3SseAlgorithm: "AES256",
    ExportFormat: "DYNAMODB_JSON"
}));
```

Go

```
cfg, _ := config.LoadDefaultConfig(context.TODO())
client := dynamodb.NewFromConfig(cfg)

exportTime := time.Date(2023, 9, 20, 12, 0, 0, 0, time.UTC)
client.ExportTableToPointInTime(context.TODO(),
 &dynamodb.ExportTableToPointInTimeInput{
    TableArn:       aws.String("arn:aws:dynamodb:us-east-1:111122223333:table/
TABLE"),
    ExportTime:     &exportTime,
    S3Bucket:       aws.String("bucket"),
    S3Prefix:       aws.String("prefix"),
    S3SseAlgorithm: types.S3SseAlgorithmAes256,
    ExportFormat:   types.ExportFormatDynamodbJson,
})
```

增量匯出

AWS CLI

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:REGION:ACCOUNT:table/TABLENAME \
  --s3-bucket BUCKET --s3-prefix PREFIX \
  --incremental-export-specification
 ExportFromTime=1693569600,ExportToTime=1693656000,ExportViewType=NEW_AND_OLD_IMAGES
 \
  --export-type INCREMENTAL_EXPORT
```

Python

```
import boto3
from datetime import datetime

client = boto3.client('dynamodb')

client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:111122223333:table/TABLE',
    IncrementalExportSpecification={
      'ExportFromTime': datetime(2023, 9, 20, 12, 0, 0),
      'ExportToTime': datetime(2023, 9, 20, 13, 0, 0),
      'ExportViewType': 'NEW_AND_OLD_IMAGES'
    },
    ExportType='INCREMENTAL_EXPORT',
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

Java

```
DynamoDbClient client = DynamoDbClient.create();

client.exportTableToPointInTime(b -> b
    .tableArn("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE")
    .exportType(ExportType.INCREMENTAL_EXPORT)
    .incrementalExportSpecification(i -> i
        .exportFromTime(Instant.parse("2023-09-20T12:00:00Z"))
        .exportToTime(Instant.parse("2023-09-20T13:00:00Z"))
        .exportViewType(ExportViewType.NEW_AND_OLD_IMAGES))
    .s3Bucket("bucket")
```

```
    .s3Prefix("prefix")
    .s3SseAlgorithm(S3SseAlgorithm.AES256)
    .exportFormat(ExportFormat.DYNAMODB_JSON));
```

.NET

```
var client = new AmazonDynamoDBClient();

await client.ExportTableToPointInTimeAsync(new ExportTableToPointInTimeRequest
{
    TableArn = "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE",
    ExportType = ExportType.INCREMENTAL_EXPORT,
    IncrementalExportSpecification = new IncrementalExportSpecification
    {
        ExportFromTime = new DateTime(2023, 9, 20, 12, 0, 0, DateTimeKind.Utc),
        ExportToTime = new DateTime(2023, 9, 20, 13, 0, 0, DateTimeKind.Utc),
        ExportViewType = ExportViewType.NEW_AND_OLD_IMAGES
    },
    S3Bucket = "bucket",
    S3Prefix = "prefix",
    S3SseAlgorithm = S3SseAlgorithm.AES256,
    ExportFormat = ExportFormat.DYNAMODB_JSON
});
```

JavaScript

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

const client = new DynamoDBClient();

await client.send(new ExportTableToPointInTimeCommand({
    TableArn: "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE",
    ExportType: "INCREMENTAL_EXPORT",
    IncrementalExportSpecification: {
        ExportFromTime: new Date("2023-09-20T12:00:00Z"),
        ExportToTime: new Date("2023-09-20T13:00:00Z"),
        ExportViewType: "NEW_AND_OLD_IMAGES"
    },
    S3Bucket: "bucket",
    S3Prefix: "prefix",
    S3SseAlgorithm: "AES256",
    ExportFormat: "DYNAMODB_JSON"
```

```
}));
```

Go

```
cfg, _ := config.LoadDefaultConfig(context.TODO())
client := dynamodb.NewFromConfig(cfg)

fromTime := time.Date(2023, 9, 20, 12, 0, 0, 0, time.UTC)
toTime := time.Date(2023, 9, 20, 13, 0, 0, 0, time.UTC)
client.ExportTableToPointInTime(context.TODO(),
 &dynamodb.ExportTableToPointInTimeInput{
    TableArn:   aws.String("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE"),
    ExportType: types.ExportTypeIncrementalExport,
    IncrementalExportSpecification: &types.IncrementalExportSpecification{
        ExportFromTime: &fromTime,
        ExportToTime:   &toTime,
        ExportViewType: types.ExportViewTypeNewAndOldImages,
    },
    S3Bucket:       aws.String("bucket"),
    S3Prefix:       aws.String("prefix"),
    S3SseAlgorithm: types.S3SseAlgorithmAes256,
    ExportFormat:   types.ExportFormatDynamodbJson,
})
```

> **Note**
>
> 如果您選擇使用受 AWS Key Management Service (AWS KMS) 保護的金鑰來加密匯出，則金鑰必須位於與目的地 S3 儲存貯體相同的區域。

使用 AWS CLI 和 AWS SDKs 取得過去匯出的詳細資訊

使用 `list-exports` 命令即可找到您過去曾執行之匯出請求的相關資訊。此命令會傳回您在過去 90 天內建立的所有匯出清單。請注意，雖然匯出任務中繼資料會在 90 天後過期，也無法使用 `list-exports` 命令傳回早於該時間的任務，但只要儲存貯體政策允許，S3 儲存貯體中的物件就會保留。DynamoDB 絕不會刪除在匯出時其在您的 S3 儲存貯體中建立的任何物件。

在匯出成功或失敗之前，其狀態會是 `PENDING`。如果成功，狀態將變更為 `COMPLETED`。如果失敗，狀態將變更為 `FAILED` 並帶有 `failure_message` 和 `failure_reason`。

列出匯出

AWS CLI

```
aws dynamodb list-exports \
    --table-arn arn:aws:dynamodb:us-east-1:111122223333:table/ProductCatalog
```

Python

```
import boto3

client = boto3.client('dynamodb')

print(
  client.list_exports(
     TableArn='arn:aws:dynamodb:us-east-1:111122223333:table/TABLE',
  )
)
```

Java

```
DynamoDbClient client = DynamoDbClient.create();

ListExportsResponse response = client.listExports(b -> b
    .tableArn("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE"));

response.exportSummaries().forEach(System.out::println);
```

.NET

```
var client = new AmazonDynamoDBClient();

var response = await client.ListExportsAsync(new ListExportsRequest
{
    TableArn = "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE"
});

response.ExportSummaries.ForEach(Console.WriteLine);
```

JavaScript

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

```
const client = new DynamoDBClient();

const response = await client.send(new ListExportsCommand({
    TableArn: "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE"
}));

console.log(response.ExportSummaries);
```

Go

```
cfg, _ := config.LoadDefaultConfig(context.TODO())
client := dynamodb.NewFromConfig(cfg)

response, _ := client.ListExports(context.TODO(), &dynamodb.ListExportsInput{
    TableArn: aws.String("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE"),
})

fmt.Println(response.ExportSummaries)
```

描述匯出

AWS CLI

```
aws dynamodb describe-export \
    --export-arn arn:aws:dynamodb:us-east-1:111122223333:table/ProductCatalog/
export/01695353076000-a1b2c3d4
```

Python

```
import boto3

client = boto3.client('dynamodb')

print(
  client.describe_export(
     ExportArn='arn:aws:dynamodb:us-east-1:111122223333:table/TABLE/
export/01695353076000-06e2188f',
  )['ExportDescription']
)
```

Java

```
DynamoDbClient client = DynamoDbClient.create();

DescribeExportResponse response = client.describeExport(b -> b
    .exportArn("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE/
export/01695353076000-06e2188f"));

System.out.println(response.exportDescription());
```

.NET

```
var client = new AmazonDynamoDBClient();

var response = await client.DescribeExportAsync(new DescribeExportRequest
{
    ExportArn = "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE/
export/01695353076000-06e2188f"
});

Console.WriteLine(response.ExportDescription);
```

JavaScript

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

const client = new DynamoDBClient();

const response = await client.send(new DescribeExportCommand({
    ExportArn: "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE/
export/01695353076000-06e2188f"
}));

console.log(response.ExportDescription);
```

Go

```
cfg, _ := config.LoadDefaultConfig(context.TODO())
client := dynamodb.NewFromConfig(cfg)

response, _ := client.DescribeExport(context.TODO(), &dynamodb.DescribeExportInput{
```

```
    ExportArn: aws.String("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE/
export/01695353076000-06e2188f"),
})

fmt.Println(response.ExportDescription)
```

DynamoDB 資料表匯出輸出格式

DynamoDB 資料表匯出內容除了包含資料表資料的檔案之外，還包括資訊清單檔案。這些檔案全都儲存於您在匯出請求中指定的 Amazon S3 儲存貯體。以下各節說明每個輸出物件的格式和內容。

完整匯出輸出

資訊清單檔案

DynamoDB 會針對每次匯出請求，在指定的 S3 儲存貯體中建立資訊清單檔案及其檢查總和檔案。

```
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
```

在請求資料表匯出時，您可以選擇 **`export-prefix`**。其有助於您在目的地 S3 儲存貯體中整理檔案。**`ExportId`** 是由服務產生的唯一權杖，可確保匯出至相同 S3 儲存貯體和 `export-prefix` 的多筆資料不會相互覆寫。

此匯出功能會為每個分割區建立至少 1 個檔案。對於空白的分割區，您的匯出請求將會建立空白檔案。每個檔案的所有項目都來自該特定分割區的雜湊金鑰空間。

> **Note**
>
> DynamoDB 也會在資訊清單檔案所在的相同目錄中建立名為 `_started` 的空白檔案。此檔案驗證目的地儲存貯體可寫入並已開始進行匯出。您可以安全地刪除此檔案。

摘要資訊清單

`manifest-summary.json` 檔案包含匯出任務的摘要資訊。這可讓您知道共用資料資料夾中的哪些資料檔案與此匯出相關聯。其格式如下：

```
{
   "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"
}
```

檔案資訊清單

`manifest-files.json` 檔案包含內含匯出資料表資料的檔案資訊。檔案採用 JSON Lines 格式，因此新行將作為項目分隔符號使用。在下列範例中，為了便於閱讀，檔案資訊清單的資料檔案詳細資訊會採用多行格式。

```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/01693685827463-2d8752fd/data/asdl123dasas.json.gz"
}
```

資料檔案

DynamoDB 會以兩種格式匯出資料表資料：DynamoDB JSON 和 Amazon Ion。無論選擇哪種格式，您的資料都會寫入多個以金鑰命名的壓縮檔案。這些檔案也會在 `manifest-files.json` 檔案中列出。

完整匯出後，Amazon S3 儲存貯體的目錄結構將包含匯出 ID 資料夾下的所有資訊清單檔案和資料檔案。

```
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

以 DynamoDB JSON 格式匯出的資料表由多個 `Item` 物件組成。每個個別物件均採用 DynamoDB 的標準封送處理 JSON 格式。

建立 DynamoDB JSON 匯出資料的自訂剖析器時，格式為 JSON Lines。這表示將新行用作項目分隔符號。Athena 和 等許多 AWS 服務 AWS Glue會自動剖析此格式。

在下列範例中，為了便於閱讀，DynamoDB JSON 匯出資料的單一項目採用多行格式。

```
{
    "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 是輸入豐富、自行描述、階層資料序列化格式，旨在處理快速開發、解耦及日常遇到的效率挑戰等問題，同時設計大規模的服務導向架構。DynamoDB 支援匯出 Ion 文字格式的資料表資料，亦即 JSON 的超集。

將資料表匯出成 Ion 格式時，資料表中使用的 DynamoDB 資料類型會映射至 Ion 資料類型。DynamoDB 設定使用 Ion 類型註釋，以釐清在來源資料表中使用的資料類型。

下表列出 DynamoDB 資料類型與離子資料類型的映射：

DynamoDB 資料類型	Ion 表示法
String (S)	string
Boolean (BOOL)	bool
Number (N)	decimal
Binary (B)	blob
Set (SS、NS、BS)	list (包含類型註解 $dynamodb_SS、$dynamodb_NS 或 $dynamodb_BS)
清單	list
Map	struct

Ion 匯出中的項目以新行分隔。每行都以 Ion 版本標記開頭，後面接著 Ion 格式的項目。在下列範例中，為了便於閱讀，Ion 匯出的項目採用多行格式。

```
$ion_1_0 {
    Item:{
        Authors:$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"
    }
}
```

增量匯出輸出

資訊清單檔案

DynamoDB 會針對每次匯出請求，在指定的 S3 儲存貯體中建立資訊清單檔案及其檢查總和檔案。

```
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
```

在請求資料表匯出時，您可以選擇 **`export-prefix`**。其有助於您在目的地 S3 儲存貯體中整理檔案。**`ExportId`** 是由服務產生的唯一權杖，可確保匯出至相同 S3 儲存貯體和 `export-prefix` 的多筆資料不會相互覆寫。

此匯出功能會為每個分割區建立至少 1 個檔案。對於空白的分割區，您的匯出請求將會建立空白檔案。每個檔案的所有項目都來自該特定分割區的雜湊金鑰空間。

> **Note**
>
> DynamoDB 也會在資訊清單檔案所在的相同目錄中建立名為 `_started` 的空白檔案。此檔案驗證目的地儲存貯體可寫入並已開始進行匯出。您可以安全地刪除此檔案。

摘要資訊清單

manifest-summary.json 檔案包含匯出任務的摘要資訊。這可讓您知道共用資料資料夾中的哪些資料檔案與此匯出相關聯。其格式如下：

```
{
 "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"
}
```

檔案資訊清單

manifest-files.json 檔案包含內含匯出資料表資料的檔案資訊。檔案採用 JSON Lines 格式，因此新行將作為項目分隔符號使用。在下列範例中，為了便於閱讀，檔案資訊清單的資料檔案詳細資訊會採用多行格式。

```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/data/sgad6417s6vss4p7owp0471bcq.json.gz"
}
```

資料檔案

DynamoDB 會以兩種格式匯出資料表資料：DynamoDB JSON 和 Amazon Ion。無論選擇哪種格式，您的資料都會寫入多個以金鑰命名的壓縮檔案。這些檔案也會在 manifest-files.json 檔案中列出。

增量匯出的資料檔案全部包含在 S3 儲存貯體的常用資料資料夾中。您的資訊清單檔案會在匯出 ID 資料夾下。

```
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
    #   ...
```

在匯出檔案中，每個項目的輸出都包含時間戳記，代表該項目在您的資料表中更新的時間，還有指出該項目是 insert、update 或 delete 操作的資料結構。時間戳記是以內部系統時鐘為基礎，且可能因應用程式時脈而有所不同。對於增量匯出，您可以為輸出結構選擇兩種匯出檢視類型：新舊映像或僅限新映像。

- 新映像提供項目的最新狀態
- 舊映像提供項目在指定的開始日期和時間之前的狀態

如果您想要查看項目在匯出期間內的變更情形，檢視類型會很有幫助。它對於有效率地更新下游系統來說也很實用，特別是在這些下游系統的分割區索引鍵與 DynamoDB 分割區索引鍵不同的情況下。

您可以藉由查看輸出的結構，推斷增量匯出輸出中的項目是 insert、update 或 delete。下表針對兩種匯出檢視類型摘要說明增量匯出結構及其對應操作。

作業	僅限新映像	新舊映像
Insert	按鍵 + 新映像	按鍵 + 新映像
更新	按鍵 + 新映像	鍵 + 新映像 + 舊映像
刪除	金鑰	按鍵 + 舊映像
Insert + delete	無輸出	無輸出

DynamoDB JSON

DynamoDB JSON 格式的資料表匯出包含中繼資料時間戳記，該時間戳記指出項目的寫入時間，後面接著項目的索引鍵和值。以下顯示使用新舊映像作為匯出檢視類型的 DynamoDB JSON 輸出範例。

```
// 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 是輸入豐富、自行描述、階層資料序列化格式，旨在處理快速開發、解耦及日常遇到的效率挑戰等問題，同時設計大規模的服務導向架構。DynamoDB 支援匯出 Ion 文字格式的資料表資料，亦即 JSON 的超集。

將資料表匯出成 Ion 格式時，資料表中使用的 DynamoDB 資料類型會映射至 Ion 資料類型。DynamoDB 設定使用 Ion 類型註釋，以釐清在來源資料表中使用的資料類型。

下表列出 DynamoDB 資料類型與離子資料類型的映射：

DynamoDB 資料類型	Ion 表示法
String (S)	string

DynamoDB 資料類型	Ion 表示法
Boolean (BOOL)	bool
Number (N)	decimal
Binary (B)	blob
Set (SS、NS、BS)	list (包含類型註解 $dynamodb_SS、$dynamodb_NS 或 $dynamodb_BS)
清單	list
Map	struct

Ion 匯出中的項目以新行分隔。每行都以 Ion 版本標記開頭，後面接著 Ion 格式的項目。在下列範例中，為了便於閱讀，Ion 匯出的項目採用多行格式。

```
$ion_1_0 {
    Record:{
        Keys:{
             ISBN:"333-3333333333"
         },
        Metadata:{
            WriteTimestampMicros:1684374845117899.
        },
        OldImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
            ISBN:"333-3333333333",
            Id:103.,
            InPublication:false,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        },
        NewImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
            Dimensions:"8.5 x 11.0 x 1.5",
            ISBN:"333-3333333333",
            Id:103.,
            InPublication:true,
            PageCount:6d2,
```

```
            Price:2d3,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        }
    }
}
```

DynamoDB 與 Amazon SageMaker 資料湖倉的零 ETL 整合

DynamoDB 與 Amazon SageMaker 資料湖倉的零 ETL 整合可自動將 DynamoDB 資料複寫至 Amazon SageMaker 資料湖倉，無需建置自訂資料移動管道。此無程式碼整合可協助客戶使用 Amazon SageMaker 資料湖倉在其 DynamoDB 資料上執行分析工作負載，而不會消耗任何 DynamoDB 資料表容量。整合會自動從您的資料表匯出資料，並保持目標為最新狀態，通常在 15 到 30 分鐘內。

主題

- DynamoDB 與 Amazon SageMaker 資料湖倉的零 ETL 整合

DynamoDB 與 Amazon SageMaker 資料湖倉的零 ETL 整合

設定 DynamoDB 資料表與 Amazon SageMaker Lakehouse 之間的整合需要先決條件，例如設定 IAM 角色，其 AWS Glue 使用 從來源存取資料並寫入目標，以及使用 KMS 金鑰加密中繼或目標位置的資料。

主題

- 建立 DynamoDB 與 Amazon SageMaker 資料湖倉的零 ETL 整合之前的先決條件
- 建立 DynamoDB 與 Amazon SageMaker 資料湖倉的零 ETL 整合
- 檢視 CloudWatch 指標以進行整合

建立 DynamoDB 與 Amazon SageMaker 資料湖倉的零 ETL 整合之前的先決條件

若要設定與 DynamoDB 來源的零 ETL 整合，您需要設定資源型存取 (RBAC) 政策，AWS Glue 允許 從 DynamoDB 資料表存取和匯出資料。政策應包含特定許可，例如 `ExportTableToPointInTime`、`DescribeTable` 和 `DescribeExport`，其條件限制對特定 AWS 帳戶 和區域的存取。如需詳細資訊，請參閱設定 Amazon DynamoDB 來源。

資料表必須啟用Point-in-time復原 (PITR)，而且您可以使用 AWS CLI 命令套用政策。透過指定更嚴格的存取控制的完全整合 ARN，可進一步精簡政策。如需詳細資訊，請參閱設定零 ETL 整合的先決條件。

建立 DynamoDB 與 Amazon SageMaker 資料湖倉的零 ETL 整合

完成整合先決條件後，您可以依照下列指引建立、修改或刪除零 ETL 整合：

建立整合

若要建立整合

1. 登入 AWS 管理主控台，並在 開啟 Amazon DynamoDB 主控台https://console.aws.amazon.com/dynamodbv2。
2. 在導覽窗格中選擇整合。
3. 選取建立與 Amazon SageMaker 資料湖倉的零 ETL 整合，然後選擇下一步。
4. 若要建立整合，請參閱建立整合。
5. 若要修改整合，請參閱修改整合。
6. 若要刪除整合，請參閱刪除整合。
7. 若要設定跨帳戶整合，請參閱設定跨帳戶整合。

在目標 Amazon S3 資料表上啟用壓縮

您可以啟用壓縮來改善 Amazon Athena 中的查詢效能。

首先，完成壓縮資源的先決條件設定，包括設定必要的 IAM 角色。如需詳細的 IAM 角色組態步驟，請參閱 Lake Formation 文件。請參閱最佳化資料表以進行壓縮。

若要在整合期間建立的 AWS Glue 資料表上啟用壓縮，請遵循 Lake Formation 壓縮啟用程序。這有助於最佳化資料表的效能和查詢效率。

檢視 CloudWatch 指標以進行整合

整合完成後，您可以看到這些 CloudWatch 指標和 EventBridge 通知在您的帳戶中為每個 AWS Glue 任務產生。如需詳細資訊，請參閱監控整合。

DynamoDB 與 Amazon OpenSearch Service 的零 ETL 整合

Amazon DynamoDB 透過適用於 OpenSearch Ingestion 的 DynamoDB 外掛程式提供與 Amazon OpenSearch Service 的零 ETL 整合。Amazon OpenSearch Ingestion 提供全受管、無程式碼的體驗，可將資料擷取至 Amazon OpenSearch Service。

使用適用於 OpenSearch Ingestion 的 DynamoDB 外掛程式，您可以使用一或多個 DynamoDB 資料表做為擷取至一或多個 OpenSearch Service 索引的來源。您可以在 AWS 管理主控台中瀏覽和設定 OpenSearch Ingestion 管道，並將 DynamoDB 做為來自 OpenSearch Ingestion 或 DynamoDB Integrations 的來源。

- 遵循 OpenSearch Ingestion 入門指南，開始使用 OpenSearch Ingestion。
- 在適用於 OpenSearch Ingestion 文件的 DynamoDB 外掛程式了解適用於 DynamoDB 外掛程式的先決條件和所有組態選項。

運作方式

外掛程式使用 DynamoDB 匯出至 Amazon S3 來建立初始快照以載入 OpenSearch。載入快照後，外掛程式會使用 DynamoDB Streams 近乎即時地複寫任何進一步的變更。每個項目都會在 OpenSearch Ingestion 中以事件處理，並可使用處理器外掛程式修改。您可以捨棄屬性或建立複合屬性，並透過路由將其傳送至不同的索引。

您必須啟用時間點復原 (PITR)，才能使用匯出至 Amazon S3。您還必須啟用 DynamoDB Streams (選取新舊映像選項)，才能使用該功能。您可以透過排除匯出設定，在不拍攝快照的情況下建立管道。

您也可以透過排除串流設定，建立僅具有快照而沒有更新的管道。外掛程式不會在您的資料表上使用讀取或寫入輸送量，因此可以安全使用，不會影響您的生產流量。在建立此整合或其他整合之前，您應該考量的串流上平行取用者數量有所限制。如需其他考量，請參閱 the section called “整合最佳實務”。

對於簡單的管道，單一 OpenSearch 運算單位 (OCU) 每秒可以處理約 1 MB 的寫入。這相當於大約 1000 個寫入請求單位 (WCU)。視管道的複雜性和其他因素而定，您可能會達到高於或低於此目標。

OpenSearch Ingestion 支援無效字母佇列 (DLQ)，適用於會導致無法復原錯誤的事件。此外，即使 DynamoDB、管道或 Amazon OpenSearch Service 的服務中斷，管道也可以從中斷的地方繼續，而無需使用者介入。

如果中斷持續超過 24 小時，可能會導致更新遺失。不過，管道會繼續處理可用性恢復時仍然可用的更新。除非事件在無效字母佇列中，否則您將需要執行新的索引建置，以修正因捨棄事件而造成的任何異常。

如需外掛程式的所有設定和詳細資訊，請參閱 OpenSearch Ingestion DynamoDB 外掛程式文件。

透過主控台整合建立體驗

DynamoDB 和 OpenSearch Service 在 中具有整合的體驗 AWS 管理主控台，可簡化入門程序。當您完成各項步驟時，服務會自動選取 DynamoDB 藍圖，並為您新增適當的 DynamoDB 資訊。

若要建立整合，請遵循 OpenSearch Ingestion 入門指南。當您前往步驟 3：建立管道時，請以下列步驟取代步驟 1 和 2：

1. 導覽至 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇整合。
3. 選取您要複寫至 OpenSearch 的 DynamoDB 資料表。
4. 選擇 Create (建立)。

從這裡，您可以繼續教學課程的其餘部分。

後續步驟

如需進一步了解 DynamoDB 如何與 OpenSearch Service 整合，請參閱以下內容：

- Amazon OpenSearch Ingestion 入門
- DynamoDB 外掛程式組態和需求

處理索引的中斷變更

OpenSearch 可以動態地將新屬性新增至您的索引。不過，為指定索引鍵設定映射範本之後，您將需要採取其他動作來變更它。此外，如果您的變更需要重新處理 DynamoDB 資料表中的所有資料，則需要採取步驟來啟動新的匯出。

> **Note**
>
> 在所有這些選項中，如果您的 DynamoDB 資料表類型與您指定的映射範本發生衝突，您仍可能會遇到問題。確定您已啟用無效字母佇列 (DLQ) (即使在開發過程中也是如此)。這可讓您更輕鬆了解在 OpenSearch 上為記錄編製索引時，造成衝突的記錄可能發生的問題。

主題

- 運作方式
- 刪除您的索引並重設管道 (以管道為中心的選項)
- 重新建立索引並重設管道 (以索引為中心的選項)
- 建立新的索引和接收器 (線上選項)
- 避免和偵錯類型衝突的最佳實務

運作方式

以下是處理索引中斷變更時所採取動作的快速概觀。請參閱以下各節中的逐步引導程序。

- 停止和啟動管道：此選項會重設管道的狀態，而管道將以新的完整匯出重新啟動。它不具破壞性，因此不會刪除您的索引或 DynamoDB 中的任何資料。如果您在執行此操作之前未建立新的索引，您可能會因為版本衝突而看到大量錯誤，因為匯出會嘗試插入比索引中目前 `_version` 更舊的文件。可以放心忽略這些錯誤。管道停止時，不會向您收取管道費用。
- 更新管道：此選項會使用藍/綠方法更新管道中的組態，而不會遺失任何狀態。如果您對管道進行重大變更 (例如將新路由、索引或索引鍵新增至現有索引)，您可能需要對管道進行完整重設並重新建立索引。此選項不會執行完整匯出。
- 刪除並重新建立索引：此選項會移除索引上的資料和映射設定。您應該在對映射進行任何重大變更之前執行此操作。它會中斷任何依賴索引的應用程式，直到索引重新建立和同步為止。刪除索引不會啟動新的匯出。請務必在更新管道之後，才可以刪除索引。否則，您的索引可能會在您更新設定之前重新建立。

刪除您的索引並重設管道 (以管道為中心的選項)

如果您仍在開發中，此方法通常是最快的選項。您將刪除 OpenSearch Service 中的索引，然後停止並啟動管道，以啟動所有資料的新匯出。這可確保映射範本不會與現有索引衝突，也不會遺失未完整處理之資料表的資料。

1. 透過 AWS 管理主控台或搭配 或 AWS CLI SDK 使用 StopPipeline API 操作來停止管道。
2. 使用新的變更更新管道組態。
3. 透過 REST API 呼叫或 OpenSearch Dashboard 在 OpenSearch Service 中刪除索引。
4. 透過 主控台或搭配 或 SDK 使用 `StartPipeline` API 操作 AWS CLI 來啟動管道。

> **Note**
>
> 這樣會啟動新的完整匯出，將產生額外費用。

5. 監控是否有任何非預期的問題，因為會產生新的匯出以建立新索引。
6. 確認索引符合您在 OpenSearch Service 中的期望。

匯出完成後，會繼續從串流讀取，您的 DynamoDB 資料表資料現在將可在索引中使用。

重新建立索引並重設管道 (以索引為中心的選項)

如果您需要在 OpenSearch Service 中對索引設計執行大量迭代，再從 DynamoDB 繼續管道，則此方法很有效。當您想要在搜尋模式中快速迭代，並希望避免在每次迭代之間等待新匯出完成時，這對於開發很有用。

1. 透過 停止管道 AWS 管理主控台，或使用 AWS CLI 或 SDK 呼叫 StopPipeline API 操作。
2. 使用您要使用的映射範本，在 OpenSearch 中刪除並重新建立索引。您可以手動插入一些範例資料，以確認您的搜尋如預期般運作。如果您的範例資料可能與 DynamoDB 中的任何資料衝突，請務必先將其刪除，再繼續進行下一個步驟。
3. 如果您的管道中有索引範本，請將其移除，或將其取代為您已在 OpenSearch Service 中建立的範本。請確定您的索引名稱與管道中的名稱相符。
4. 透過 主控台，或使用 AWS CLI 或 SDK 呼叫 `StartPipeline` API 操作來啟動管道。

> **Note**
>
> 這樣將啟動新的完整匯出，會產生額外費用。

5. 監控是否有任何非預期的問題，因為會產生新的匯出以建立新索引。

匯出完成後，會繼續從串流讀取，您的 DynamoDB 資料表資料現在可在索引中使用。

建立新的索引和接收器 (線上選項)

如果您需要更新映射範本，但目前在生產環境中使用您的索引，則此方法很有效。這會建立全新的索引，您需要在該索引同步和驗證後將應用程式移至該處。

> **Note**
>
> 這將在串流上建立另一個取用者。如果您還有其他消費者，例如 AWS Lambda 或 全域資料表，這可能是問題。您可能需要暫停現有管道的更新，以建立能載入新索引的容量。

1. 使用新的設定和不同的索引名稱建立新的管道。
2. 監控新索引是否有任何非預期的問題。
3. 將應用程式切換到新索引。
4. 驗證一切正常運作後，停止並刪除舊管道。

避免和偵錯類型衝突的最佳實務

- 一律使用無效字母佇列 (DLQ)，以便在發生類型衝突時更輕鬆偵錯。
- 一律使用索引範本搭配映射並設定 `include_keys`。雖然 OpenSearch Service 會動態映射新的索引鍵，這可能會導致非預期行為的問題 (例如預期某個項目是 `GeoPoint`，但被建立為 `string` 或 `object`) 或錯誤 (例如具有混合 `long` 和 `float` 值的 `number`)。
- 如果您需要讓現有的索引在生產環境中運作，您也可以在管道組態檔案中重新命名索引，以取代任何先前的刪除索引步驟。這會建立新的索引。接著，您的應用程式將需要更新，以便在完成之後指向新的索引。
- 如果您有使用處理器修正的類型轉換問題，您可以使用 `UpdatePipeline` 測試此問題。若要這麼做，您需要停止並啟動或處理無效字母佇列，以修正任何先前略過但發生錯誤的文件。

使用 DynamoDB 零 ETL 整合和 OpenSearch Service 的最佳實務

DynamoDB 具有與 Amazon OpenSearch Service 的 DynamoDB 零 ETL 整合。如需詳細資訊，請參閱適用於 OpenSearch Ingestion 的 DynamoDB 外掛程式和適用於 Amazon OpenSearch Service 的特定最佳實務。

Configuration

- 只針對您要執行搜尋的資料編製索引。一律使用映射範本 (`template_type: index_template` 和 `template_content`) 和 `include_keys` 來實作此操作。

- 監控日誌是否有與類型衝突相關的錯誤。OpenSearch Service 預期指定索引鍵的所有值都具有相同的類型。如果不相符，則會產生例外狀況。如果遇到這類錯誤，您可以新增處理器，以捕捉指定的索引鍵是否一律具有相同的值。
- document_id 值通常使用 primary_key 中繼資料值。在 OpenSearch Service 中，文件 ID 等同於 DynamoDB 中的主索引鍵。使用主索引鍵可讓您輕鬆尋找文件，並確保更新一致複寫至其中，不會發生衝突。

  您可以使用協助程式函數 getMetadata 取得您的主索引鍵 (例如 document_id: "${getMetadata('primary_key')}")。如果您使用的是複合主鍵，協助程式函數會為您將它們串連在一起。
- 一般而言，action 設定請使用 opensearch_action 中繼資料值。這將確保複寫更新時，OpenSearch Service 中的資料符合 DynamoDB 的最新狀態。

  您可以使用協助程式函數 getMetadata 取得您的主索引鍵 (例如 action: "${getMetadata('opensearch_action')}")。針對篩選等使用案例，您也可以透過 dynamodb_event_name 取得串流事件類型。不過，您通常不應該將其用於 action 設定。

可觀測性

- 一律在 OpenSearch 接收器上使用無效字母佇列 (DLQ) 來處理捨棄的事件。DynamoDB 通常不如 OpenSearch Service 結構化，而且總是可能發生非預期的情況。透過無效字母佇列，您可以復原個別事件，甚至自動化復原程序。這可協助您避免需要重建整個索引。
- 一律設定複寫延遲不超過預期數值的提醒。假設一分鐘提醒不會太吵，通常是安全的。根據寫入流量的尖峰程度，以及管道上的 OpenSearch Compute Unit (OCU) 設定，可能有所不同。

  如果您的複寫延遲超過 24 小時，串流會開始捨棄事件，除非從頭開始完整重建索引，否則會有準確性問題。

擴展

- 針對管道使用自動擴展，以協助擴展或縮減 OCU，使其最適合工作負載。
- 對於沒有自動擴展功能的已佈建輸送量資料表，建議您根據寫入容量單位 (WCU) 除以 1000 來設定 OCU。將最小值設定為低於該數量 1 個 OCU (但至少 1 個)，並將最大值設定為高於該數量至少 1 個 OCU。
  - 公式：

```
OCU_minimum = GREATEST((table_WCU / 1000) - 1, 1)
OCU_maximum = (table_WCU / 1000) + 1
```

  - 範例：您的資料表已佈建 25000 WCU。管道的 OCU 應設定為最小值 24 (25000/1000 - 1)，最大值至少 26 (25000/1000 + 1)。
- 對於具有自動擴展功能的已佈建輸送量資料表，建議根據您的最小和最大 WCU 除以 1000 來設定 OCU。將最小值設定為低於 DynamoDB 最小值 1 個 OCU，並將最大值設定為至少高於 DynamoDB 最大值 1 個 OCU。
  - 公式：

```
OCU_minimum = GREATEST((table_minimum_WCU / 1000) - 1, 1)
OCU_maximum = (table_maximum_WCU / 1000) + 1
```

  - 範例：您的資料表具有自動擴展政策，最小值為 8000，最大值為 14000。您管道的 OCU 應設定為最小值 7 (8000/1000 - 1)，最大值 15 (14000/1000 + 1)。
- 對於隨需輸送量資料表，建議您根據每秒寫入請求單位的典型尖峰和低谷值來設定 OCU。您可能需要針對較長的時段進行平均，具體取決於您可用的彙總。將最小值設定為低於 DynamoDB 最小值 1 個 OCU，並將最大值設定為至少高於 DynamoDB 最大值 1 個 OCU。
  - 公式：

```
# 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
```

  - 範例：您的資料表的平均谷值為每秒 300 次寫入請求單位，平均峰值為 4300。管道的 OCU 應設定為最小值 1 (300/1000 - 1，但至少為 1)，最大值 5 (4300/1000 + 1)。
- 遵循擴展目的地 OpenSearch Service 索引的最佳實務。如果您的索引規模不足，則會減慢從 DynamoDB 擷取的速度，並可能導致延遲。

> **Note**
>
> GREATEST 是 SQL 函數，給定一組引數後，會傳回具有最大值的引數。

整合 DynamoDB 與 Amazon EventBridge

Amazon DynamoDB 提供 DynamoDB Streams 用於擷取變更資料，可擷取 DynamoDB 資料表中的項目層級變更。DynamoDB Streams 可以調用 Lambda 函式來處理這些變更，能夠與其他服務和應用程式以事件驅動的方式進行整合。DynamoDB Streams 也支援篩選，實現高效及鎖定目標的事件處理。

DynamoDB Streams 支援每碎片最多兩個同時取用者，並支援透過 Lambda 事件篩選進行篩選，因此只會處理符合特定條件的項目。有些客戶可能需要支援兩個以上取用者。其他客戶則可能需要先擴充變更事件再加以處理，或使用更進階的篩選和路由。

整合 DynamoDB 與 EventBridge 可支援前述需求。

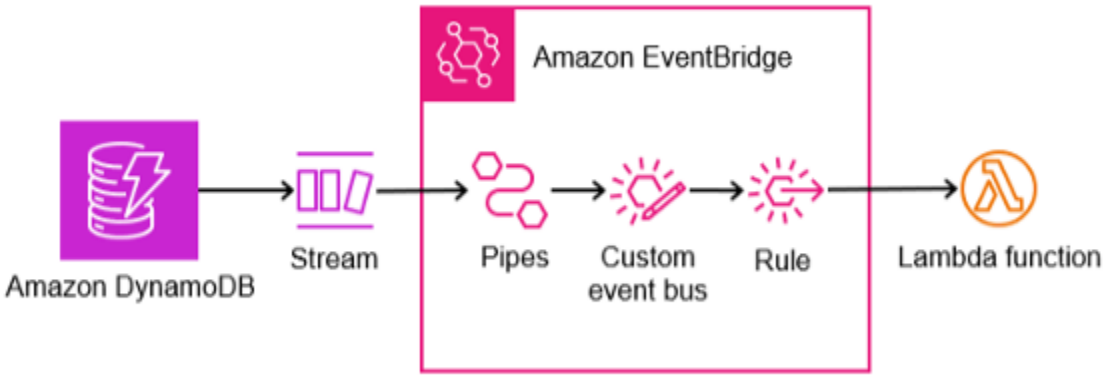
Amazon EventBridge 是一種無伺服器服務，該服務使用事件將應用程式元件連接在一起，讓您更輕鬆地建置可擴展的事件驅動型應用程式。EventBridge 透過 EventBridge Pipes 與 Amazon DynamoDB 進行原生整合，實現從 DynamoDB 到 EventBridge 匯流排的無縫資料流程。然後該匯流排可以透過一組規則和目標展開傳送到多個應用程式和服務。

主題

- 運作方式
- 透過主控台建立整合
- 後續步驟

運作方式

DynamoDB 與 EventBridge 管道之間的整合，會使用 DynamoDB Streams 擷取 DynamoDB 資料表中項目層級變更的時間順序序列。以這種方式擷取的每個記錄都包含資料表中修改的資料。



EventBridge 管道會使用來自 DynamoDB Streams 的事件，並將其路由至 EventBridge 匯流排等目標 (事件匯流排是接收事件並將其交付至目的地的路由器，也稱為目標)。交付是以符合事件內容的規則為依據。管道還可選用篩選特定事件的功能，並可先擴充事件資料然後再將其傳送至目標。

雖然 EventBridge 支援多種目標類型，但實作展開傳送設計的常見選擇是使用 Lambda 函式作為目標。下列範例會示範與 Lambda 函式的整合。

透過主控台建立整合

請依照下列步驟透過 AWS 管理主控台建立整合。

1. 遵循《DynamoDB 開發人員指南》啟用串流 章節中的步驟，在來源資料表上啟用 DynamoDB Streams。如果來源資料表已啟用 DynamoDB Streams，請確認目前取用者少於兩個。取用者可以是 Lambda 函式、DynamoDB 全域資料表、Amazon DynamoDB 與 Amazon OpenSearch Service 的零 ETL 整合，或是直接從串流讀取的應用程式，例如透過 DynamoDB Streams Kinesis 轉接器讀取的應用程式。
2. 按照《EventBridge 使用者指南》中建立 Amazon EventBridge 事件匯流排一節中的步驟建立 EventBridge 事件匯流排。
   a. 建立事件匯流排時，請啟用結構描述探索。
3. 按照《EventBridge 使用者指南》中建立 Amazon EventBridge 管道一節中的步驟建立 EventBridge 管道。
   a. 設定來源時，在來源欄位中選取 DynamoDB，然後在 DynamoDB Streams 欄位中選取來源資料表串流的名稱。
   b. 設定目標時，請在目標服務欄位中選取 EventBridge 事件匯流排，然後在事件匯流排作為目標欄位選取在步驟 2 建立的事件匯流排。
4. 將範例項目寫入來源 DynamoDB 資料表以觸發事件。這將讓 EventBridge 從範例項目推斷結構描述。此結構描述可用來建立路由事件的規則。例如若您要實作涉及超載屬性的設計模式，您可能需要根據排序索引鍵的值觸發不同規則。有關如何將項目寫入 DynamoDB 的詳細資訊，請參閱《DynamoDB 開發人員指南》中的使用項目和屬性一節。
5. 依照《Lambda 開發人員指南》中使用 Python 建置 Lambda 函式一節中的步驟，建立範例 Python Lambda 函式作為目標。建立函式時，您可以使用下列範例程式碼來示範整合。調用時，函式會列印隨事件接收的 `NewImage` 及 `OldImage`，可在 CloudWatch Logs 中檢視。

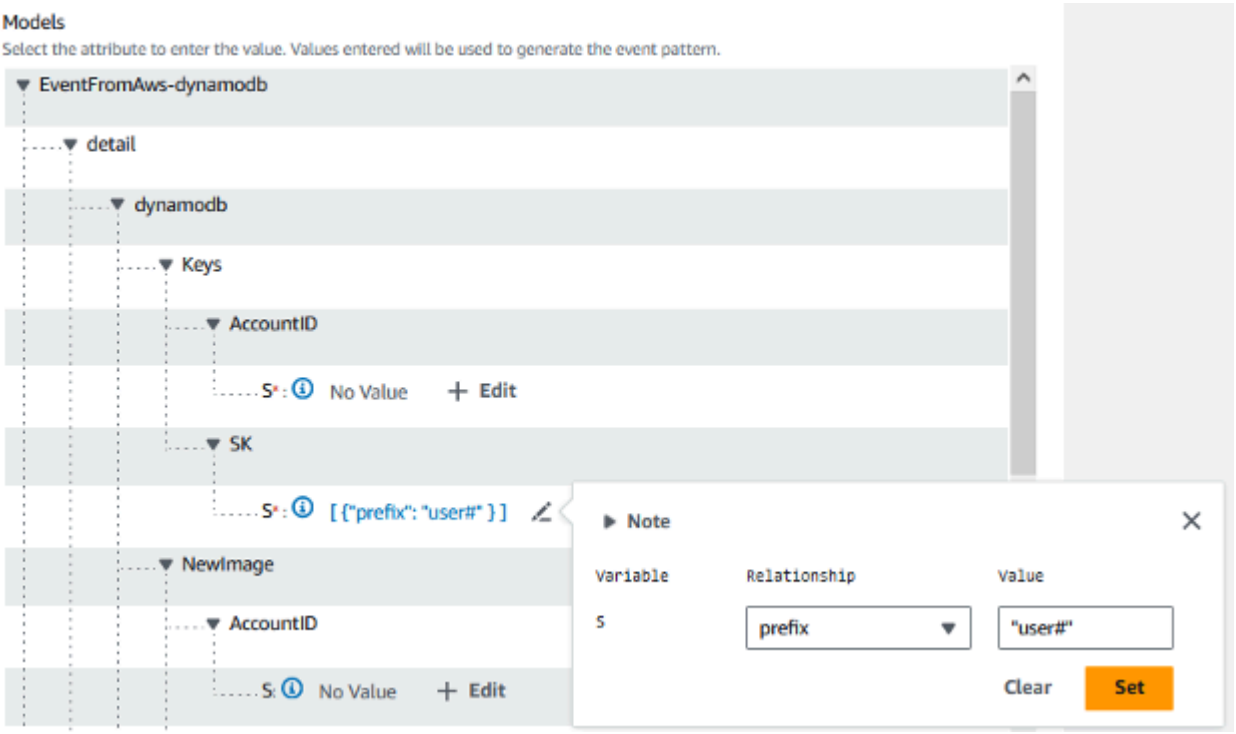
```
import json

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. 依照《EventBridge 使用者指南》中回應事件的建立規則一節所述步驟，建立 EventBridge 規則以將事件路由至新的 Lambda 函式。

   a. 定義規則詳細資訊時，請選取您在步驟 2 中建立為事件匯流排的事件匯流排名稱。

   b. 建構事件模式時，請遵循現有結構描述指南。您可在此為事件選取探索到的結構描述登錄檔和探索到的結構描述。這可讓您設定專屬於使用案例的事件模式，僅會路由符合特定屬性的訊息。例如若您只想比對 SK 開頭為 “user#” 的 DynamoDB 項目，就會使用這樣的設定。



   c. 依據您的結構描述完成設計模式後，請按一下以 JSON 格式產生事件模式。如果您想要比對 DynamoDB Streams 上顯示的所有事件，請針對事件模式使用下列 JSON。

```
{
  "source": ["aws.dynamodb"]
}
```

d. 選取目標時，請遵循 AWS 服務指南。在選取目標欄位選擇 Lambda 函式。在函式欄位選取您在步驟 5 建立的 Lambda 函式。

7. 您現在可以依照《EventBridge 使用者指南》中在事件匯流排啟動或停止結構描述探索一節中的步驟，停止事件匯流排上的結構描述探索。
8. 將第二個範例項目寫入來源 DynamoDB 資料表以觸發事件。驗證事件是否在每個步驟成功處理。
   a. 遵循《EventBridge 使用者指南》中監控 Amazon EventBridge 一節所述內容，檢視事件匯流排的 CloudWatch 指標 PutEventsApproximateSuccessCount。
   b. 遵循《Lambda 開發人員指南》中監控和疑難排解 Lambda 函式一節所述內容，檢視 Lambda 函式的函式日誌。如果您的 Lambda 函式使用提供的範例程式碼，您應該會在 CloudWatch Logs 日誌群組中看到列印的 DynamoDB Streams `NewImage` 和 `OldImage`。
   c. 遵循《Lambda 開發人員指南》中監控和疑難排解 Lambda 函式一節所述內容，檢視 Lambda 函式的錯誤計數和成功率 (%) 指標。

後續步驟

此範例提供與單一 Lambda 函式作為目標的基本整合。如需進一步了解更複雜的設定，例如建立多項規則、建立多個目標、與其他服務整合，以及擴充事件，請參閱完整的 EventBridge 使用者指南：EventBridge 入門。

> **Note**
>
> 請注意任何可能與應用程式相關的 EventBridge 配額。雖然 DynamoDB Streams 容量會隨資料表擴展，但 EventBridge 配額是另外計算的。在大型應用程式中要注意的常見配額為每秒交易的調用限流限制，以及每秒交易的 PutEvents 限流限制。這些配額會指定可傳送至目標的調用數量，以及每秒可放入匯流排的事件數量。

將 DynamoDB 與 Amazon Managed Streaming for Apache Kafka 整合

Amazon Managed Streaming for Apache Kafka (Amazon MSK) 可透過全受管、高可用性的 Apache Kafka 服務，輕鬆即時擷取和處理串流資料。

Apache Kafka 是一種分散式資料存放區，針對即時擷取和處理串流資料進行最佳化。Kafka 可以處理記錄串流、以記錄產生的順序有效地存放記錄串流，以及發布和訂閱記錄串流。

由於這些功能，Apache Kafka 通常用於建置即時串流資料管道。資料管道能夠可靠地處理資料，並將資料從一個系統移動到另一個系統，當透過使用多個支援不同使用案例的資料庫以實現專用資料庫策略時，這可能會是重要的功能。

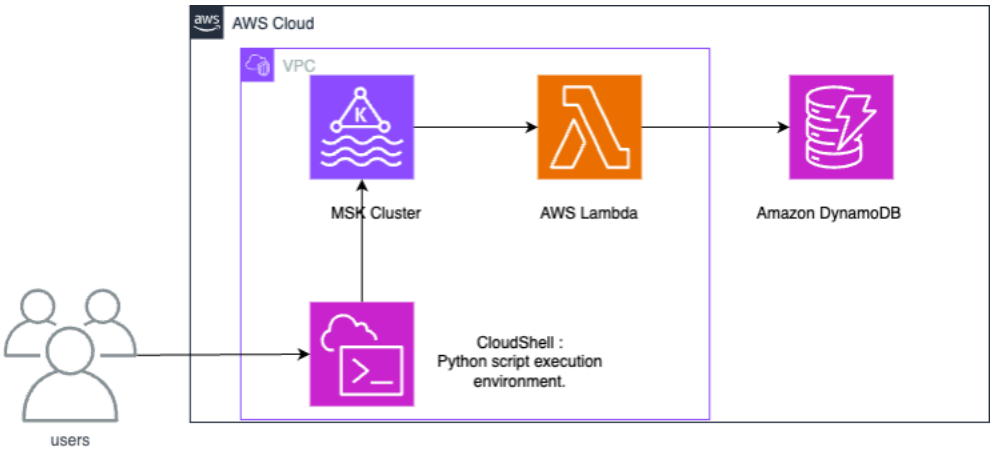
Amazon DynamoDB 是這些資料管道中的常見目標，可支援使用鍵值或文件資料模型的應用程式，並希望具有幾毫秒一致效能的無限可擴展性。

主題

- 運作方式
- 設定 Amazon MSK 與 DynamoDB 之間的整合
- 後續步驟

運作方式

Amazon MSK 和 DynamoDB 之間的整合使用 Lambda 函式來取用來自 Amazon MSK 的記錄，並將其寫入 DynamoDB。



Lambda 會在內部輪詢 Amazon MSK 中的新訊息，然後同步調用目標 Lambda 函式。Lambda 函式的事件承載包含來自 Amazon MSK 的訊息批次。對於 Amazon MSK 和 DynamoDB 之間的整合，Lambda 函式會將這些訊息寫入 DynamoDB。

設定 Amazon MSK 與 DynamoDB 之間的整合

> **Note**
>
> 您可以在下列 GitHub 儲存庫下載此範例中使用的資源。

下列步驟說明如何設定 Amazon MSK 和 Amazon DynamoDB 之間的範例整合。此範例代表由物聯網 (IoT) 裝置產生並擷取至 Amazon MSK 的資料。當資料導入 Amazon MSK 時，它可以與分析服務或與 Apache Kafka 相容的第三方工具整合，從而實現各種分析使用案例。整合 DynamoDB 也提供個別裝置記錄的索引鍵值查詢。

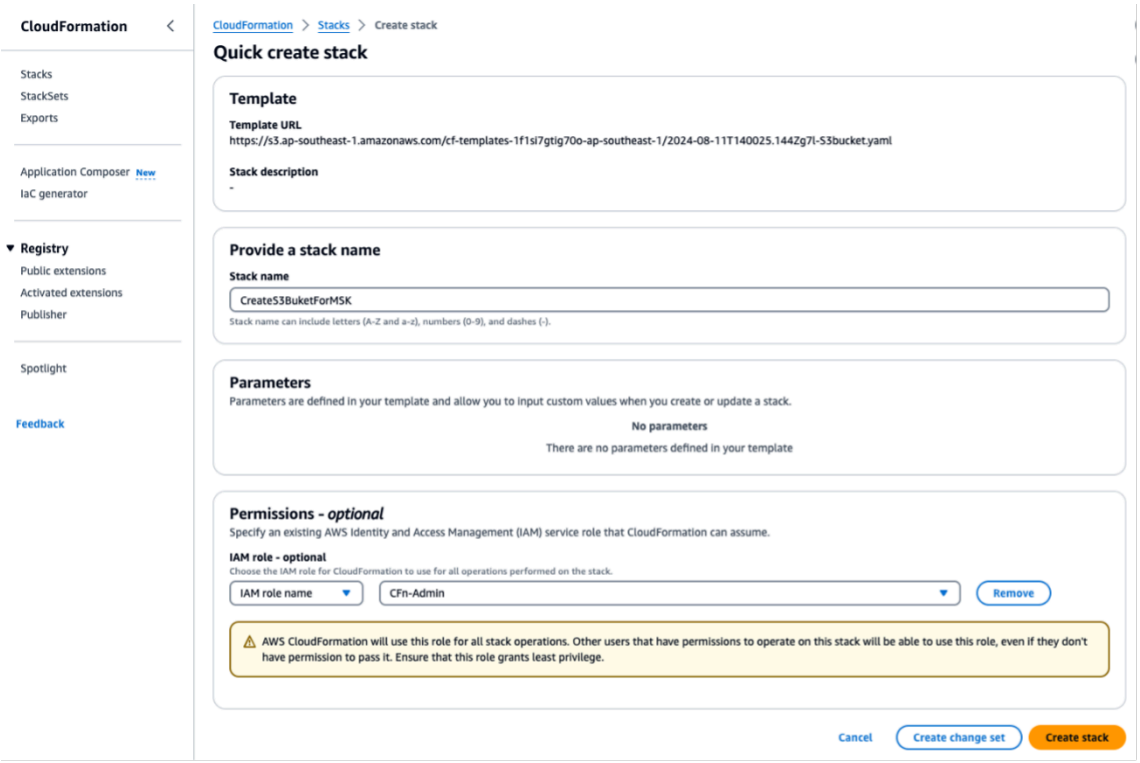
此範例將示範 Python 指令碼如何將 IoT 感應器資料寫入 Amazon MSK。然後，Lambda 函式會將具有分割區索引鍵「`deviceid`」的項目寫入 DynamoDB。

提供的 CloudFormation 範本將建立下列資源：Amazon S3 儲存貯體、Amazon VPC、Amazon MSK 叢集，以及 AWS CloudShell 用於測試資料操作的 。

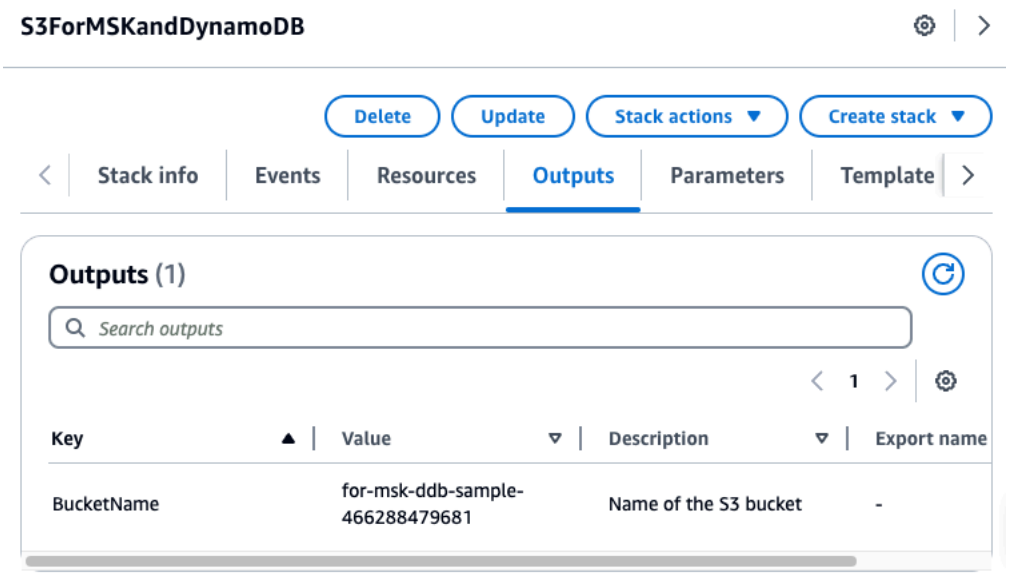
若要產生測試資料，請建立 Amazon MSK 主題，然後建立 DynamoDB 資料表。您可以從管理主控台使用 Session Manager 登入 CloudShell 作業系統並執行 Python 指令碼。

執行 CloudFormation 範本後，您可以執行下列操作來完成建置此架構。

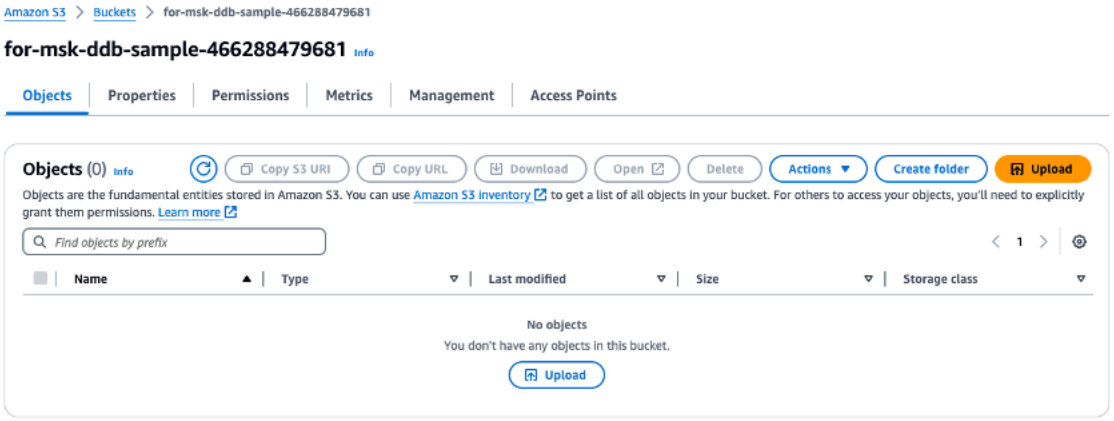
1. 執行 CloudFormation 範本 `S3bucket.yaml` 以建立 S3 儲存貯體。對於任何後續指令碼或操作，請在相同區域中執行它們。輸入 `ForMSKTestS3` 做為 CloudFormation 堆疊名稱。



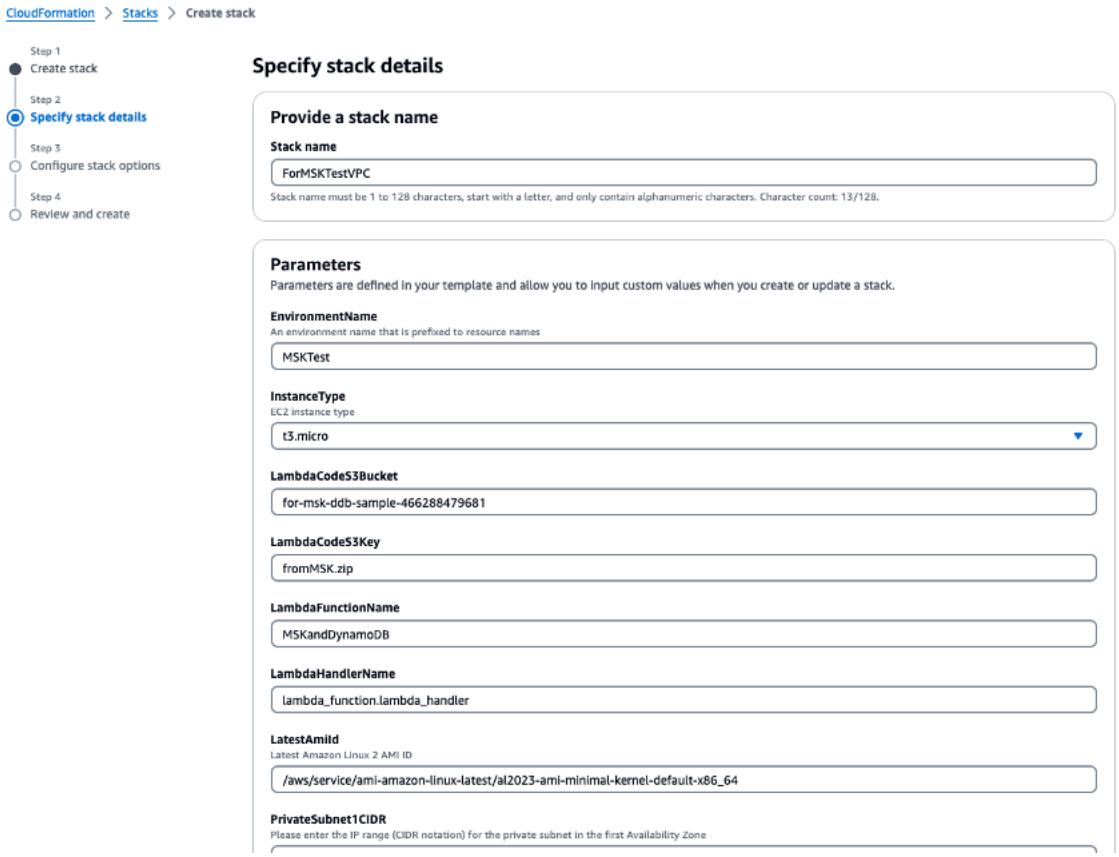
完成後，記下輸出下方的 S3 儲存貯體名稱輸出。您在步驟 3 中將需要此名稱。



2. 將下載的 ZIP 檔案 `fromMSK.zip` 上傳至您剛建立的 S3 儲存貯體。

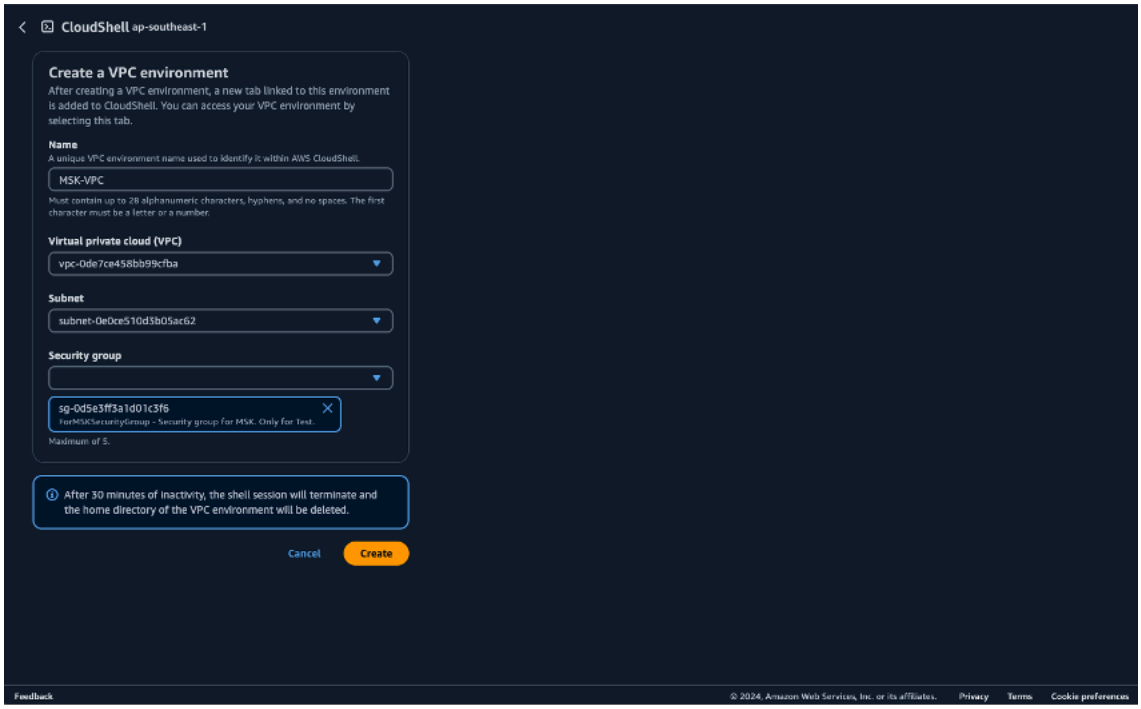


3. 執行 CloudFormation 範本 `VPC.yaml` 來建立 VPC、Amazon MSK 叢集和 Lambda 函式。在參數輸入畫面上，輸入您在步驟 1 中被要求提供 S3 儲存貯體時建立的 S3 儲存貯體名稱。將 CloudFormation 堆疊名稱設定為 `ForMSKTestVPC`。



4. 準備在 CloudShell 中執行 Python 指令碼的環境。您可以在 AWS 管理主控台上使用 CloudShell。如需使用 CloudShell 的詳細資訊，請參閱 AWS CloudShell入門。啟動 CloudShell 後，建立屬於您剛建立的 VPC 的 CloudShell，以連線至 Amazon MSK 叢集。在私有子網路中建立 CloudShell。填寫下列欄位：

   1. 名稱 - 可以設定為任何名稱。例如 MSK-VPC
   2. VPC - 選取 MSKTest
   3. 子網路 - 選取 MSKTest 私有子網路 (AZ1)
   4. SecurityGroup - 選取 ForMSKSecurityGroup



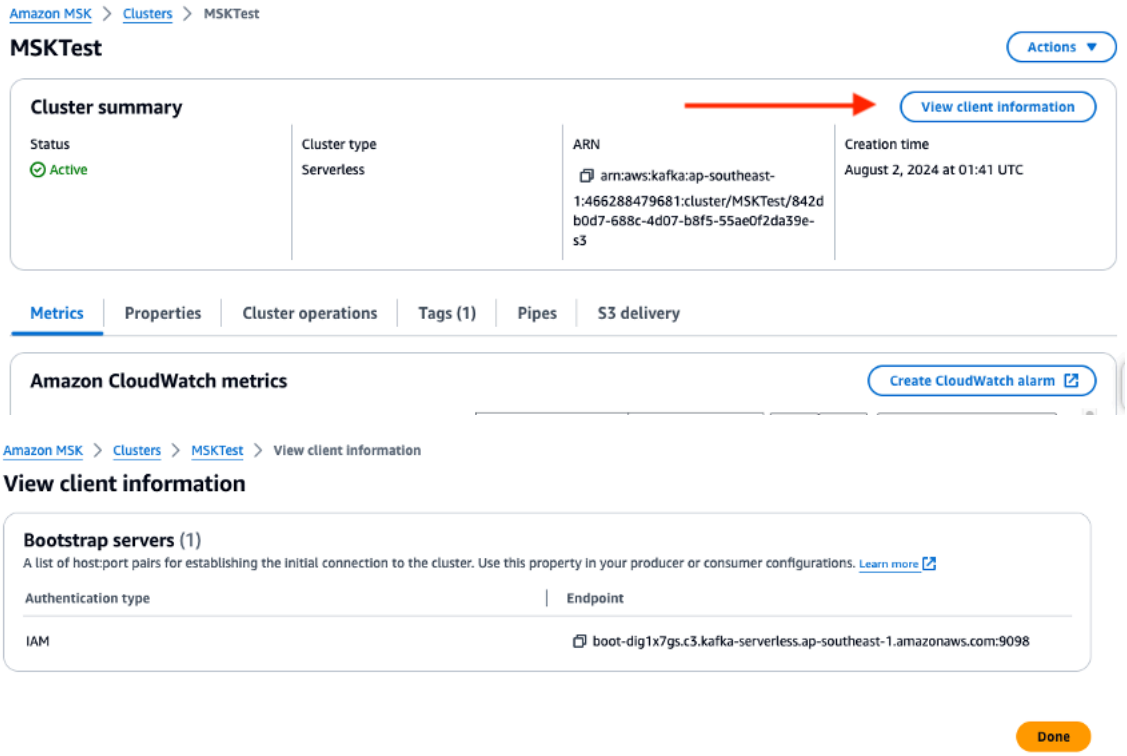
屬於私有子網路的 CloudShell 啟動後，請執行下列命令：

```
pip install boto3 kafka-python aws-msk-iam-sasl-signer-python
```

5. 從 S3 儲存貯體下載 Python 指令碼。

```
aws s3 cp s3://[YOUR-BUCKET-NAME]/pythonScripts.zip ./
unzip pythonScripts.zip
```

6. 檢查管理主控台，並在 Python 指令碼中設定代理程式 URL 和區域值的環境變數。在管理主控台中檢查 Amazon MSK 叢集代理程式端點。



7. 在 CloudShell 上設定環境變數。如果您使用的是美國西部 (奧勒岡)：

```
export AWS_REGION="us-west-2"
export MSK_BROKER="boot-YOURMSKCLUSTER.c3.kafka-serverless.ap-
southeast-1.amazonaws.com:9098"
```

8. 執行下列 Python 指令碼。

   建立 Amazon MSK 主題：

```
python ./createTopic.py
```

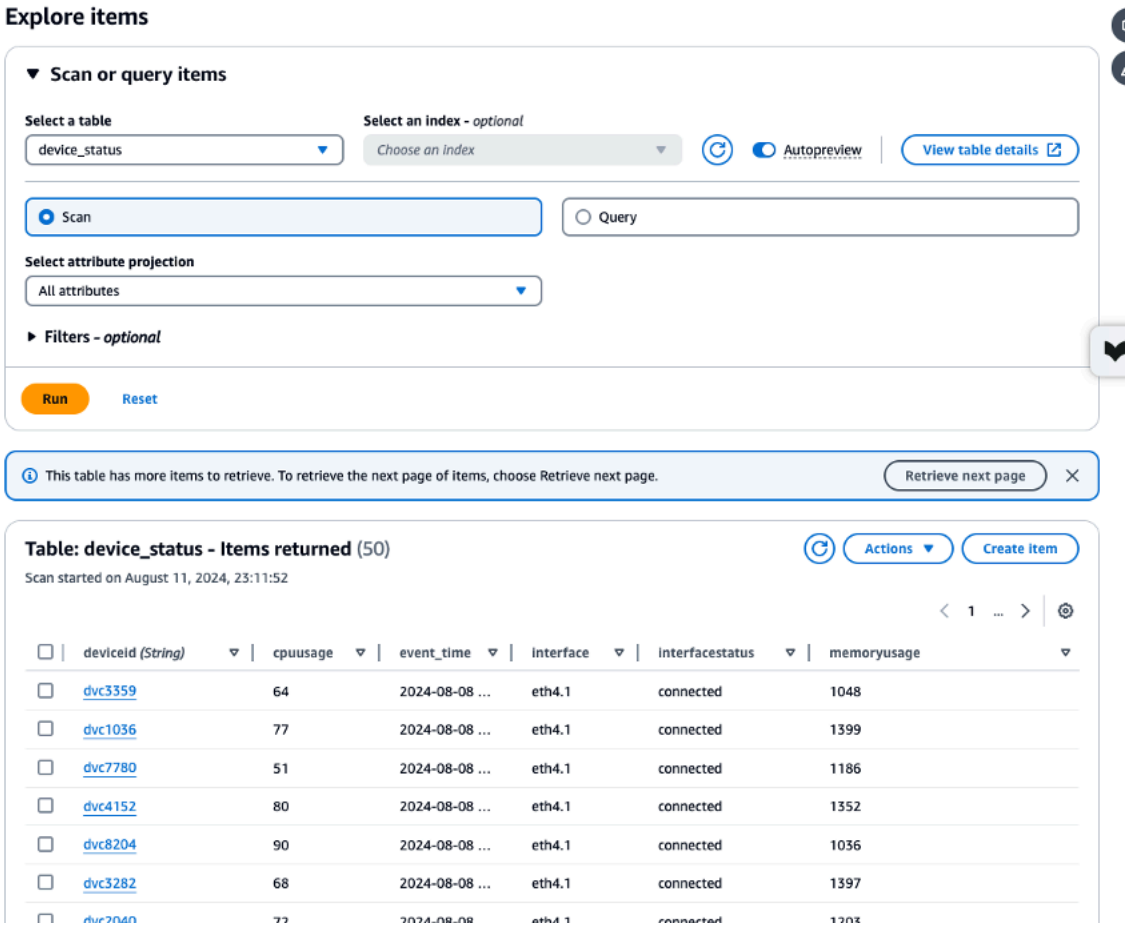
   建立 DynamoDB 資料表：

```
python ./createTable.py
```

將測試資料寫入 Amazon MSK 主題：

```
python ./kafkaDataGen.py
```

9. 檢查所建立 Amazon MSK、Lambda 和 DynamoDB 資源的 CloudWatch 指標，並使用 DynamoDB Data Explorer 驗證存放在 `device_status` 資料表中的資料，以確保所有程序都正確執行。如果每個程序執行均無錯誤，您可以檢查從 CloudShell 寫入 Amazon MSK 的測試資料是否也會寫入 DynamoDB。



10. 當您完成此範例時，請刪除本教學課程中建立的資源。刪除兩個 CloudFormation 堆疊：`ForMSKTestS3` 和 `ForMSKTestVPC`。如果堆疊刪除成功完成，則會刪除所有資源。

後續步驟

> **Note**
>
> 如果您在遵循此範例時建立了資源，請記得將其刪除，以避免任何非預期的費用。

整合已識別連結 Amazon MSK 和 DynamoDB 的架構，以啟用串流資料以支援 OLTP 工作負載。從這裡，可以將 DynamoDB 與 OpenSearch Service 連結來實現更複雜的搜尋。請考慮與 EventBridge 整合以因應更複雜的事件驅動需求，以及與 Amazon Managed Service for Apache Flink 等擴充功能整合以因應更高輸送量和更低延遲的需求。

與 DynamoDB 整合的最佳實務

整合 DynamoDB 與其他服務時，您應該始終遵循使用每個個別服務的最佳實務。不過，您應該考量一些特定於整合的最佳實務。

主題

- 在 DynamoDB 中建立快照
- 在 DynamoDB 中擷取資料變更

在 DynamoDB 中建立快照

- 一般而言，我們建議使用匯出至 Amazon S3 來建立快照，以進行初始複寫。這種作法既符合成本效益，也不會與應用程式的流量競爭輸送量。您也可以考慮備份與還原至新資料表，然後再進行掃描操作。如此可避免與您的應用程式競爭輸送量，但通常成本效益會比匯出更低。
- 執行匯出時，請務必設定 `StartTime`。這可讓您輕鬆地判斷要從何處開始變更資料擷取 (CDC)。
- 使用匯出至 S3 時，請在 S3 儲存貯體上設定生命週期動作。一般而言，將過期動作設定為 7 天是安全的，但您仍應遵循公司可能設立的任何準則。即使您在擷取後明確刪除項目，此動作也有助於發現問題，如此有助於降低不必要的成本並防止違反政策。

在 DynamoDB 中擷取資料變更

- 如果您需要近乎即時的 CDC，請使用 DynamoDB Streams 或 Amazon Kinesis Data Streams (KDS)。當您在決定要使用何者時，通常會考慮哪個最容易搭配下游服務使用。如果您需要在分割區索引鍵層級提供依序事件處理，或您有異常大型的項目，請使用 DynamoDB Streams。
- 如果您不需要近乎即時的 CDC，可以使用以增量匯出匯出到 Amazon S3，僅匯出在兩個時間點之間發生的變更。

  如果您使用匯出到 S3 來產生快照，此功能特別有用，因為您可以使用類似的程式碼來處理增量匯出。匯出到 S3 的成本通常略低於先前的串流選項，但成本通常不是選擇該使用哪個選項的主要因素。

- 您通常只能同時有兩個 DynamoDB 串流的取用者。在規劃整合策略時，請將這一點納入考量。
- 請勿使用掃描來偵測變更。這可能適用於小規模作業，但很快就會變得相當不切實際。

搭配 DynamoDB 使用代理式 AI

Amazon DynamoDB 是無伺服器、全受管的分散式 NoSQL 資料庫，具有任何規模的單一位數毫秒效能。DynamoDB 已針對高輸送量工作負載進行最佳化，您可以整合生成式 AI 模型來擴展其功能。使用生成式 AI 模型，您可以即時處理存放在 DynamoDB 資料表中的資料，並建置內容感知及高度個人化的應用程式。您也可以徹底運用業務、使用者和應用程式資料，自訂生成式 AI 解決方案，以強化最終使用者體驗。

如需有關 gen AI 和 解決方案 AWS 提供以建置 gen AI 應用程式的詳細資訊，請參閱使用生成式 AI 轉換您的業務。

主題

- DynamoDB 的生成式 AI 使用案例
- DynamoDB 的生成式 AI 部落格
- 利用 DynamoDB 與 OpenSearch Service 的零 ETL 整合
- 使用 DynamoDB 做為 LangGraph 代理程式的檢查點存放區

DynamoDB 的生成式 AI 使用案例

DynamoDB 廣泛用於 AI 支援的對話式應用程式中，例如使用基礎模型 (FM) 建置的聊天機器人和客服中心。您可以透過 Amazon Bedrock、Amazon SageMaker AI 或其他模型供應商存取 FM。這類應用程式通常會使用 DynamoDB 來提升個人化效果，並強化應用程式資料、業務資料及使用者資料這三種資料模式的使用者體驗。以下是這些資料模式的範例：

- 透過整合 LangChain、LlamaIndex 或自訂程式碼，來儲存應用程式資料，例如聊天訊息歷程記錄。此內容可讓模型和使用者互相交談以強化使用者體驗。
- 利用庫存、定價和文件等業務資料，建立自訂使用者體驗。
- 套用使用者資料，例如 Web 歷史記錄、過去的訂單和使用者偏好設定，以提供個人化的答案。

例如，保險公司可以使用 DynamoDB 建置聊天機器人，供檢索增強生成 (RAG) 型 AI 模型近即時存取資料。此類資料的範例包括即時貸款利率、產品定價、合規/標準合約副本、使用者 Web 歷史記錄和使用者偏好設定。結合 DynamoDB 與 RAG，可新增保險產品和使用者資料的深入及更新資訊。如此可讓提示和答案更豐富，為最終使用者提供準確、個人化且近乎即時的體驗。

同樣地，金融服務產業客戶可使用 DynamoDB、Amazon Bedrock 知識庫和 Amazon Bedrock 代理程式，來建置 RAG 型生成式 AI 應用程式。這些應用程式可以使用開源營利報告和通話文字記錄。亦可運用使用者特定的產品組合和交易歷史記錄，來產生產品組合隨選摘要，包括未來的前景。

DynamoDB 的生成式 AI 部落格

下列文章提供詳細的使用案例、最佳實務和逐步指南，協助您利用 DynamoDB 功能來建置進階的 AI 驅動應用程式。

- 適用於生成式 AI 聊天機器人的 Amazon DynamoDB 資料模型
- 使用 Amazon DynamoDB、Amazon Bedrock 和 LangChain 建置可擴展的內容感知聊天機器人
- 使用 LangGraph 和 Amazon DynamoDB 建置耐用的 AI 代理器

利用 DynamoDB 與 OpenSearch Service 的零 ETL 整合

您可以搭配使用 Amazon Bedrock 與 DynamoDB，提供對基礎模型 (FM) 的無伺服器存取，例如 Amazon Titan 和其他第三方模型。您可以運用 Amazon OpenSearch Service 零 ETL 整合，在建置生成式 AI 應用程式時啟用向量搜索功能。採用零 ETL 整合 DynamoDB、OpenSearch 和 Amazon Bedrock 的生成式 AI 工作坊提供實作體驗，讓您設定 DynamoDB 與 OpenSearch 的零 ETL 整合。此工作坊會執行下列任務：

- 建立從 DynamoDB 資料表到 OpenSearch 的管道。
- 在 OpenSearch 中建立 Amazon Bedrock 連接器。
- 利用 OpenSearch 作為向量存放區，查詢 Amazon Bedrock。
- 使用 Amazon Bedrock 中的 Claude FM，以簡單的英文建立文字回應，說明 OpenSearch 傳回的搜尋結果。

此工作坊可讓您整合 DynamoDB 與 OpenSearch，以建置生成式 AI 應用程式。它還示範了跨資料庫引擎的彈性查詢功能，有助您針對傳統使用案例整合 DynamoDB 和 OpenSearch。此工作坊是 Amazon DynamoDB Immersion Day 的七個單元之一。您可以在任何 中執行此研討會 AWS 帳戶。

您也可以參閱下列部落格文章，了解如何在 DynamoDB 和 OpenSearch Service 之間設定零 ETL 整合。此部落格文章也說明如何在 OpenSearch Service 中設定模型連接器，以使用 Amazon Bedrock 自動產生傳入資料的嵌入。Amazon DynamoDB 與 Amazon OpenSearch Service 的零 ETL 向量搜索。

使用 DynamoDB 做為 LangGraph 代理程式的檢查點存放區

LangGraph 是使用大型語言模型 (LLMs) 建置具狀態、多動作 AI 應用程式的架構。LangGraph 代理程式需要持久性儲存來維持對話狀態、啟用human-in-the-loop工作流程、支援容錯能力，並提供時間歷程偵錯功能。DynamoDB 的無伺服器架構、單一位數毫秒延遲和自動擴展，使其成為 上生產 LangGraph 部署的理想檢查點存放區 AWS。

`langgraph-checkpoint-aws` 套件提供實作 LangGraph 檢查點介面的`DynamoDBSaver`類別，可讓您在 DynamoDB 中保留代理程式狀態，並為大型檢查點提供選用的 Amazon Simple Storage Service 卸載。

主要功能

狀態持久性

在每個步驟後自動儲存代理程式狀態，讓代理程式從中斷中恢復並從失敗中復原。

即時清理的時間

使用 DynamoDB 存留時間自動使舊檢查點過期，以管理儲存成本。

壓縮

選擇性地使用 gzip 壓縮檢查點資料，以降低儲存成本並改善輸送量。

Amazon S3 卸載

自動將大型檢查點（大於 350 KB) 卸載至 Amazon Simple Storage Service，以在 DynamoDB 項目大小限制內運作。

同步和非同步支援

同步和非同步 APIs皆可在不同應用程式架構中提供彈性。

先決條件

- Python 3.10 或更新版本
- AWS 帳戶 具有建立 DynamoDB 資料表（以及選用 Amazon S3 儲存貯體）許可的
- AWS 已設定的登入資料（請參閱登入資料設定選項 AWS 的文件）

> **Important**
>
> 本指南會建立可能會產生費用 AWS 的資源。DynamoDB 預設使用pay-per-request，如果您啟用大型檢查點卸載，則會收取 Amazon S3 費用。按照清除區段，在完成後刪除資源。

安裝

從 PyPI 安裝檢查點套件：

```
pip install langgraph-checkpoint-aws
```

基本使用

下列範例示範如何將 DynamoDB 設定為 LangGraph 代理程式的檢查點存放區：

```
from langgraph.graph import StateGraph
from langgraph_checkpoint_aws import DynamoDBSaver
from typing import TypedDict

# Define your state schema
class State(TypedDict):
    input: str
    result: str

# Initialize the DynamoDB checkpoint saver
checkpointer = DynamoDBSaver(
    table_name="langgraph-checkpoints",
    region_name="us-east-1"
)

# Build your LangGraph workflow
builder = StateGraph(State)
builder.add_node("process", lambda state: {"result": "processed"})
builder.set_entry_point("process")
builder.set_finish_point("process")

# Compile the graph with the DynamoDB checkpointer
graph = builder.compile(checkpointer=checkpointer)

# Invoke the graph with a thread ID to enable state persistence
```

```
config = {"configurable": {"thread_id": "session-123"}}
result = graph.invoke({"input": "data"}, config)
```

組態thread_id中的 充當 DynamoDB 中的分割區索引鍵，可讓您維護個別的對話執行緒，並擷取任何執行緒的歷史狀態。

生產組態

對於生產部署，您可以啟用存留時間、壓縮和 Amazon S3 卸載。您也可以使用 endpoint_url 參數指向本機 DynamoDB 執行個體進行測試：

```
import boto3
from botocore.config import Config
from langgraph_checkpoint_aws import DynamoDBSaver

# Production configuration
session = boto3.Session(
    profile_name="production",
    region_name="us-east-1"
)

checkpointer = DynamoDBSaver(
    table_name="langgraph-checkpoints",
    session=session,
    ttl_seconds=86400 * 7,           # Expire checkpoints after 7 days
    enable_checkpoint_compression=True,  # Enable gzip compression
    boto_config=Config(
        retries={"mode": "adaptive", "max_attempts": 6},
        max_pool_connections=50
    ),
    s3_offload_config={
        "bucket_name": "my-checkpoint-bucket"
    }
)

# Local testing with DynamoDB Local
local_checkpointer = DynamoDBSaver(
    table_name="langgraph-checkpoints",
    region_name="us-east-1",
    endpoint_url="http://localhost:8000"
)
```

DynamoDB 資料表組態

檢查點儲存器需要具有複合主索引鍵的 DynamoDB 資料表。您可以使用下列 AWS CloudFormation 範本建立資料表：

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'DynamoDB table for LangGraph checkpoint storage'

Parameters:
  TableName:
    Type: String
    Default: langgraph-checkpoints

Resources:
  CheckpointTable:
    Type: AWS::DynamoDB::Table
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
    Properties:
      TableName: !Ref TableName
      BillingMode: PAY_PER_REQUEST
      AttributeDefinitions:
        - AttributeName: PK
          AttributeType: S
        - AttributeName: SK
          AttributeType: S
      KeySchema:
        - AttributeName: PK
          KeyType: HASH
        - AttributeName: SK
          KeyType: RANGE
      TimeToLiveSpecification:
        AttributeName: ttl
        Enabled: true
      PointInTimeRecoverySpecification:
        PointInTimeRecoveryEnabled: true
      SSESpecification:
        SSEEnabled: true
```

使用 AWS CLI 部署範本：

```
aws cloudformation deploy \
  --template-file template.yaml \
```

```
  --stack-name langgraph-checkpoint \
  --parameter-overrides TableName=langgraph-checkpoints
```

所需的 IAM 許可

下列 IAM 政策提供 DynamoDB 檢查點儲存器所需的最低許可。將 *111122223333* 取代為您的 AWS 帳戶 ID，並更新 區域以符合您的環境。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query",
        "dynamodb:BatchGetItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/langgraph-checkpoints"
    }
  ]
}
```

如果您啟用 Amazon S3 卸載，請將下列陳述式新增至政策：

```
{
  "Effect": "Allow",
  "Action": [
    "s3:PutObject",
    "s3:GetObject",
    "s3:DeleteObject",
    "s3:PutObjectTagging"
  ],
  "Resource": "arn:aws:s3:::my-checkpoint-bucket/*"
},
{
  "Effect": "Allow",
  "Action": [
    "s3:GetBucketLifecycleConfiguration",
    "s3:PutBucketLifecycleConfiguration"
```

```
  ],
  "Resource": "arn:aws:s3:::my-checkpoint-bucket"
}
```

非同步用量

對於非同步應用程式，請使用檢查點儲存器提供的非同步方法：

```
import asyncio
from langgraph.graph import StateGraph
from langgraph_checkpoint_aws import DynamoDBSaver
from typing import TypedDict

class State(TypedDict):
    input: str
    result: str

async def main():
    checkpointer = DynamoDBSaver(
        table_name="langgraph-checkpoints",
        region_name="us-east-1"
    )
    builder = StateGraph(State)
    builder.add_node("process", lambda state: {"result": "processed"})
    builder.set_entry_point("process")
    builder.set_finish_point("process")
    graph = builder.compile(checkpointer=checkpointer)

    config = {"configurable": {"thread_id": "async-session-123"}}
    result = await graph.ainvoke({"input": "data"}, config)
    return result

asyncio.run(main())
```

清除

若要避免持續收費，請刪除您建立的資源：

```
# Delete the DynamoDB table
aws dynamodb delete-table --table-name langgraph-checkpoints

# Delete the CloudFormation stack (if you used the template above)
```

```
aws cloudformation delete-stack --stack-name langgraph-checkpoint

# If you created an S3 bucket for large checkpoint offloading, empty and delete it
aws s3 rm s3://my-checkpoint-bucket --recursive
aws s3 rb s3://my-checkpoint-bucket
```

錯誤處理

常見錯誤案例：

- 找不到資料表：驗證 `table_name`並`region_name`符合您的 DynamoDB 資料表。
- 調節：如果您看到 `ProvisionedThroughputExceededException`，請考慮切換至隨需計費模式或增加佈建容量。
- 超過項目大小：如果檢查點超過 350 KB，請啟用 Amazon S3 卸載（請參閱 生產組態)。
- 登入資料錯誤：確認您的 AWS 登入資料有效且具有必要的許可。

其他資源

- PyPI 上的 langgraph-checkpoint-aws
- GitHub 上的 langgraph-checkpoint-aws
- LangGraph 文件
- DynamoDB 最佳實務
- 使用 LangGraph 和 Amazon DynamoDB 建置耐用的 AI 代理器

Amazon DynamoDB 的配額和限制條件

本節說明 Amazon DynamoDB 中的目前配額 (過去稱為限制)。本主題也說明如何執行配額管理任務，例如檢視目前的配額並請求增加配額。

主題

- 在 DynamoDB 中執行配額管理任務
- 在 DynamoDB 中請求增加配額
- Amazon DynamoDB 中的配額
- Amazon DynamoDB 的限制

在 DynamoDB 中執行配額管理任務

Amazon DynamoDB 有數個服務元件，例如資料表、串流、索引等。當您建立 AWS 帳戶 時，這些元件會設定預設配額 (先前稱為限制)。除非另有說明，否則每個配額都是區域特定規定。您可以請求提高一部分配額的限制。在達到資源的配額後，建立該資源的額外呼叫便會失敗並發生例外狀況。

存取 DynamoDB 配額

您可透過以下方式來使用 DynamoDB Service Quotas 服務：

- AWS 管理主控台

  Service Quotas 主控台是一種以瀏覽器為基礎的介面，可用來管理您的 Service Quotas。您可以從任何 AWS 管理主控台 頁面頂端的導覽列上選擇，或在 AWS 管理主控台 中搜尋 Service Quotas，以存取 Service Quotas。

- AWS Command Line Interface 工具

  使用 AWS Command Line Interface 工具時，您可以在系統的命令列發出命令，以執行 Service Quotas 任務。若您想要建構執行 AWS 任務的指令碼，命令列工具非常實用。

- AWS SDK

  您可以使用適用於各種程式設計語言和平台的 AWS SDK (例如 Java、Python、Ruby、.NET、iOS 和 Android 等)，以執行 Service Quotas 任務。

如果 Service Quotas 主控台中無法使用可調整配額，請使用 AWS Support Center Console 建立 Service Quotas 增加案例。

在主控台中檢視目前的配額

使用 Service Quotas 主控台檢視目前的 DynamoDB 配額

1. 開啟位於 https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/ 的 Service Quotas 主控台。
2. 從導覽列 (位於畫面頂部) 中，選取一個區域。
3. 主控台會在帳戶層級或資源層級顯示有關 DynamoDB 配額名稱、已套用帳戶層級配額值、AWS 預設配額值、使用率和配額調整能力的詳細資訊。

   如果套用的配額值或使用率不可用，主控台會顯示無法使用。您可以透過支援中心主控台請求套用的配額值。
4. 選擇特定配額名稱以檢視詳細資訊頁面，其中會顯示該配額的描述、配額代碼、配額 ARN、使用率、已套用的帳戶層級配額值、調整能力和AWS 預設配額值。

   如果適用，詳細資訊頁面也會顯示任何監控選項、警示、請求歷史記錄和任何配額的標籤。

使用 AWS CLI 檢視目前的配額

若要檢視 DynamoDB 配額的預設值：

- 使用 DynamoDB 服務代碼 (dynamodb) 呼叫 `ListDefaultServiceQuotas` 操作，以擷取 Amazon DynamoDB 服務配額的預設值。

```
$ 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
        }......
    ]
}
```

若要檢視套用的配額值：

- 使用 DynamoDB 服務程式碼 (dynamodb) 呼叫 [ListServiceQuotas](#) 操作，透過分別傳遞 ACCOUNT、RESOURCE 或 ALL 作為參數 QuotaAppliedAtLevel 的值，在帳戶層級、資源層級或所有層級擷取所有套用的配額值。下列 CLI 範例會擷取在帳戶層級套用的配額值。

```
$ 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"
        }.....

        }
    ]
}
```

在 DynamoDB 中請求增加配額

您可以使用 Service Quotas 主控台、AWS CLI 或支援案例，請求提高每個區域的配額。如果 Service Quotas 主控台中沒有可調整的配額，請使用 AWS Support Center Console 來建立服務配額增加案例。

支援 可以核准、拒絕或部分核准您的配額增加請求。增加的配額不會立即授予，可能需要幾天的時間才會生效。

使用 Service Quotas 主控台請求提高

1. 開啟 Service Quotas 主控台，網址為 https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/
2. 從導覽列 (位於畫面頂部) 中，選取一個區域。
3. 依資源名稱篩選清單。例如，輸入 On-Demand (隨需) 以找出隨需執行個體的配額。
4. 如果配額可調整，請選取配額，然後選擇請求提高配額。
5. 對於變更配額值，輸入新配額值。
6. 選擇請求。
7. 若要檢視主控台中任何擱置或最近解決的請求，請從導覽窗格中選擇儀表板。對於擱置的請求，請選擇請求狀態以開啟請求回條。請求的初始狀態為 Pending (待定)。狀態變更為已請求配額後，您會看到帶有 支援 的案例編號。選擇案例編號，為請求開啟票證。

如需詳細資訊，包含如何使用 AWS CLI 或 SDK 請求增加配額，請參閱 Service Quotas 使用者指南中的請求增加配額。

Amazon DynamoDB 中的配額

本節說明 Amazon DynamoDB 中的當前配額 (過去稱為限制)。各項配額除非另有說明，否則都是區域特定規定。

> **Note**
>
> DynamoDB 中的所有大小測量都會使用二進位單位。DynamoDB 表示 1 KB = 1024 位元組，1 MB = 1024 KB，1 GB = 1024 MB，1 TB = 1024 GB。

主題

- 讀取/寫入輸送量
- 預留容量
- 表格
- 全域資料表
- 次要索引
- 每份資料表的預估次要索引屬性
- DynamoDB Streams
- 從 Amazon S3 匯入
- 資料表匯出至 Amazon S3
- 備份與還原
- Contributor Insights

讀取/寫入輸送量

輸送量預設配額

AWS 會將一些預設配額放在您的帳戶可在區域內佈建和使用的輸送量上。

帳戶層級的讀取輸送量和帳戶層級的寫入輸送量配額適用於帳戶層級。這些帳戶層級配額適用於給定區域中您帳戶的所有資料表和全域次要索引的佈建輸送容量總和。帳戶的所有可用輸送量可佈建至單一資料表或多份資料表。這些配額僅適用於使用佈建容量模式的資料表。

資料表層級的讀取輸送量和資料表層級的寫入輸送量配額，會以不同方式套用至使用佈建容量模式的資料表以及使用隨需容量模式的資料表。

對於佈建容量模式資料表和 GSI，配額是可佈建給區域中任何資料表或其任何 GSI 之讀取和寫入容量單位的最大數量。任何個別資料表及其所有 GSI 的總計也必須低於帳戶層級的讀取和寫入輸送量配額。這是對所有佈建資料表及其 GSI 的總計必須維持在帳戶層級讀取和寫入輸送量配額之下要求的額外補充。

對於隨需容量模式資料表和 GSI，資料表層級配額是可用於任何資料表或該表中任何個別 GSI 的最大讀取和寫入容量單位。在隨需模式下，不會將帳戶層級的讀取和寫入輸送量配額套用至資料表。

以下是預設情況下適用於您帳戶的輸送量配額。

> **Note**
>
> 所有容量單位和請求單位配額都是每秒測量。例如，40，000 個讀取容量單位的配額表示每秒 40，000 個讀取。

> **Note**
>
> 您可以透過增加服務配額，為您的 DynamoDB 資料表請求任意數量的讀取容量單位 (RCU) 或寫入容量單位 (WCU)。下表列出的值代表初始預設配額。這些不是資料表的最大限制。

輸送量配額名稱	On-Demand	佈建	可調整
每個資料表	40,000 個讀取請求單位與 40,000 個寫入請求單位	40,000 個讀取容量單位和 40,000 個寫入容量單位	是
每個帳戶	不適用	80,000 個讀取容量單位和 80,000 個寫入容量單位	是
任何資料表或全域次要索引的輸送量下限	不適用	1 個讀取容量單位和 1 個寫入容量單位	是

提高或降低輸送量 (已佈建的資料表)

提高佈建輸送量

您可以視需要經常提高 ReadCapacityUnits 或 WriteCapacityUnits，使用 AWS 管理主控台 或 UpdateTable 操作。在單一呼叫中，您可以提高資料表、該資料表中任何全域次要索引，或這些項目任意組合的佈建輸送量。新的設定值要在 UpdateTable 操作完成後才會生效。

當您新增佈建容量時，不能超過您的每個帳戶配額，而且 DynamoDB 不允許您非常快速地提高佈建容量。在這些限制之外，您可以將您資料表的佈建容量提高至您需要的數量。如需每個帳戶配額的詳細資訊，請參閱前一節 輸送量預設配額。

降低佈建輸送量

您可以在 UpdateTable 操作中降低每個資料表和全域次要索引的 ReadCapacityUnits 或 WriteCapacityUnits (或兩個都降低)。新的設定值要在 UpdateTable 操作完成後才會生效。

您每天可以在 DynamoDB 資料表上執行的佈建容量減少的次數存在預設配額。一天是根據國際標準時間 (UTC) 來定義。您從每天的 4 個可用減少開始。每小時可再減少 1 次，最多 4 次。在整天的 24 小時期間，這可讓您減少最多 27 次 （第一個小時 4 次，其餘 23 小時各 1 次）。

> **Important**
>
> 資料表和全域次要索引調降限制是分開的，所以特定資料表的任何全域次要索引皆有各自的調降限制。但如果單一請求降低了資料表和全域次要索引的輸送量，只要其中之一超過目前的限制就會遭到拒絕。請求未得到部分處理。

Example

在一天的第一個 4 小時中，具有全域次要索引的資料表可依下列方式修改：

- 調降資料表的 WriteCapacityUnits 或 ReadCapacityUnits (或兩者) 四次。
- 調降全域次要索引的 WriteCapacityUnits 或 ReadCapacityUnits (或兩者) 四次。

在同一天結束時，資料表和全域次要索引的輸送量有可能各調降 27 次。

預留容量

AWS 會對您的帳戶可購買的作用中預留容量數量設定預設配額。配額限制是寫入容量單位 (WCU) 和讀取容量單位 (RCU) 的預留容量總和。

預留容量配額	作用中的預留容量	可調整
每個帳戶	1,000,000 個佈建容量單位 (WCU _ RCU)	是

如果您嘗試單次購買超過 1,000,000 個佈建容量單位，將會收到此服務配額限制錯誤訊息。如果您擁有作用中的預留容量，並嘗試購買額外的預留容量，導致作用中的佈建容量單位總數超過 1,000,000 個，也會收到此服務配額限制錯誤訊息。

表格

資料表大小

資料表大小沒有任何實際限制。就項目數或位元組數而言，資料表是沒有限制的。

每個帳戶每個區域的資料表數目上限

對於任何 AWS 帳戶，每個 AWS 區域的初始配額為 2，500 個資料表。

如果單一帳戶需要超過 2,500 個資料表，請聯絡您的 AWS 客戶團隊，了解將資料表數量增加到最多 10,000 個的可能性。如需超過 10,000 個資料表，建議的最佳實務是設定多個帳戶，每個帳戶最多可以提供 10,000 個資料表。

全域資料表

使用全域資料表時，適用下列預設配額。

預設全域資料表配額	On-Demand	佈建
MRSC 全域資料表的數量 (請參閱 the section called “一致性模式”)	在任何容量模式中共 400 份 MRSC 全域資料表	在任何容量模式中共 400 份 MRSC 全域資料表
針對多區域最終一致性 (MREC) 設定的每資料表輸送量	40,000 個讀取請求單位與 40,000 個寫入請求單位	40,000 個讀取容量單位和 40,000 個寫入容量單位

預設全域資料表配額	On-Demand	佈建
針對多區域高度一致性 (MRSC) 設定的每資料表輸送量	40,000 個讀取請求單位與 40,000 個寫入請求單位	40,000 個讀取容量單位和 40,000 個寫入容量單位
每個帳戶、每個區域、每天新複本的回填資料	10 TB	10 TB

> **Note**
>
> 在某些情況下，您可能需要請求提高配額限制 AWS 支援。如果您適用下列任一項，請參閱 https://aws.amazon.com/support：
>
> - 全域資料表輸送量配額必須等於或大於每個資料表輸送量配額，才能成功建立複本。MREC 和 MRSC 全域資料表有不同的全域資料表輸送量配額。
> - 如果您要將一個複本或多個複本在 24 小時內新增到一個目標區域且合計大於 10TB，則必須針對您的新增複本資料回填配額請求新增服務配額。
> - 如果您遇到類似下列的錯誤：
>   - 無法在 'example_region_A' 區域中建立資料表 'example_table' 的複本，因為超過 'example_region_B' 區域中目前帳戶的限制。

次要索引

每份資料表最多可以定義 5 個本機次要索引。

每個資料表有 20 個全域次要索引的預設配額。

每份資料表的預估次要索引屬性

您最多可以投影 100 個屬性到資料表所有的區域和全域次要索引。此配額僅適用於使用者指定的投影屬性。

在 `CreateTable` 操作中，如果您指定 `INCLUDE` 的 `ProjectionType`，則 `NonKeyAttributes` 中指定的屬性總數 (對所有次要索引求總) 絕對不能超過 100。如果您將相同的屬性名稱投影到兩個不同的索引，在判斷總數時，會計算為兩個不同的屬性。

這項限制不適用於含 KEYS_ONLY 或 ALL 的 ProjectionType 的次要索引。

DynamoDB Streams

DynamoDB Streams 中的碎片同時讀取

對於不是全域資料表的單一區域資料表，您最多可以設計兩個同步程序，以便同時讀取同一個 DynamoDB Streams 碎片。超過此限制會導致請求限流。對於全域資料表，我們建議您將同時讀取的數量限制為一個，以避免請求限流。

啟用 DynamoDB Streams 之資料表的寫入容量上限

AWS 會在啟用 DynamoDB Streams 的 DynamoDB 資料表寫入容量上放置一些預設配額。這些預設配額僅適用於佈建讀/寫容量模式下的資料表。

- 每份資料表：40,000 個寫入容量單位

從 Amazon S3 匯入

Amazon S3 的 DynamoDB 匯入可支援多達 50 個並行匯入任務，在 us-east-1、us-west-2 和 eu-west-1 區域的匯入來源物件大小總計達 15TB。在所有其他區域中，最多支援 50 個並行匯入任務，大小總計為 1TB。每個匯入任務在所有區域中最多可以使用 50,000 個 Amazon S3 物件。如需匯入和驗證的詳細資訊，請參閱匯入格式配額與驗證。

資料表匯出至 Amazon S3

完整匯出：所有執行中的資料表可以匯出最多 300 個並行匯出任務，或匯出最多 100TB。匯出排入佇列之前，會檢查這兩項限制。

增量匯出：DynamoDB 增量匯出至 Amazon S3 最多可支援 300 個並行匯出任務，或來自所有傳輸中資料表匯出的總計 100 TB。匯出期間時段限制為最短 15 分鐘，最長 24 小時。

備份與還原

DynamoDB 透過 DynamoDB 隨需或連續備份支援最多 50 個並行還原，總計 50 TB。 AWS Backup 支援最多 50 個並行還原，總計 25 TB。

Contributor Insights

當您在 DynamoDB 資料表上啟用 Customer Insights 時，您仍然受限於 Contributor Insights 規則限制。如需詳細資訊，請參閱 CloudWatch 服務配額。

Amazon DynamoDB 的限制

本節說明 Amazon DynamoDB 中的當前配額 (過去稱為限制)。

> **Note**
>
> DynamoDB 中的所有大小測量都會使用二進位單位。DynamoDB 表示 1 KB = 1024 位元組，1 MB = 1024 KB，1 GB = 1024 MB，1 TB = 1024 GB。

主題

- 讀取/寫入容量模式
- 次要索引
- 分割區索引鍵和排序索引鍵
- 命名規則
- 資料類型
- 項目
- 屬性
- 表達式參數
- DynamoDB 交易
- DynamoDB Streams
- DynamoDB Accelerator (DAX)
- API 特定限制
- DynamoDB 靜態加密

讀取/寫入容量模式

您可在 24 小時滾動期間內，最多將資料表從佈建容量模式切換至隨需模式四次。您可隨時將資料表從隨需模式切換回佈建容量模式。

如需了解如何在讀取與寫入容量模式間切換，請參閱 切換 DynamoDB 容量模式時的注意事項。

容量單位大小 (已佈建的資料表)

一個讀取容量單位 = 每秒一個高度一致性讀取，或每秒兩個最終一致讀取，適用於大小上限為 4 KB 的項目。

一個寫入容量單位 = 每秒一個寫入，適用於大小上限為 1 KB 的項目。

交易讀取請求需要 2 個讀取容量單位才能執行每秒讀取一個大小上限為 4 KB 的項目。

交易讀取請求需要兩個寫入容量單位才能執行每秒寫入一個大小上限為 1 KB 的項目。

請求單位大小 (隨需資料表)

一個讀取請求單位 = 每秒一個高度一致性讀取，或每秒兩個最終一致讀取，適用於大小上限為 4 KB 的項目。

一個寫入請求單位 = 每秒一個寫入，項目大小上限為 1 KB。

交易讀取請求需要兩個讀取請求單位，才能執行每秒讀取一個大小上限為 4 KB 的項目。

交易讀取請求需要兩個寫入請求單位，才能執行每秒寫入一個大小上限為 1 KB 的項目。

次要索引

每份資料表的預估次要索引屬性

您最多可以投影 100 個屬性到資料表所有的區域和全域次要索引。這只適用於使用者指定的投影屬性。

在 `CreateTable` 操作中，如果您指定 `INCLUDE` 的 `ProjectionType`，則 `NonKeyAttributes` 中指定的屬性總數，即所有次要索引總數，絕對不能超過 100。如果您將相同的屬性名稱投影到兩個不同的索引，在判斷總數時，這會計算為兩個不同的屬性。

這項限制不適用於含 `KEYS_ONLY` 或 `ALL` 的 `ProjectionType` 的次要索引。

分割區索引鍵和排序索引鍵

分割區索引鍵長度

分割區索引鍵值的長度下限為 1 個位元組。長度上限為 2048 個位元組。

分割區索引鍵值

資料表或次要索引中，不同的分割區索引鍵值數目沒有實際限制。

排序索引鍵長度

排序索引鍵值的長度下限為 1 個位元組。長度上限為 1024 個位元組。

排序索引鍵值

一般而言，每個分割區索引鍵值的相異排序索引鍵值數目沒有實際限制。

例外狀況是有次要索引的資料表。項目集合是具有相同分割區索引鍵屬性值的一組項目。在全域次要索引中，項目集合與基礎資料表相互獨立 (而且可以具有不同的分割區索引鍵屬性)，但在本機次要索引中，索引檢視會與資料表中的項目共同存在於相同的分割區中，並共用相同的分割區索引鍵屬性。鑑於此位置資訊，當資料表擁有一或多個 LSI 時，項目集合無法分發至多個分割區。

對於具有一或多個 LSI 的資料表，項目集合的大小不得超過 10GB。這包括所有基礎資料表項目，以及具有與分割區索引鍵屬性相同值的所有投影 LSI 視圖。分割區的大小上限為 10GB。如需詳細資訊，請參閱項目集合大小限制。

命名規則

資料表名稱和次要索引名稱

資料表和次要索引名稱至少必須為 3 字元長，但不能超過 255 個字元。以下為允許的字元：

- `A-Z`
- `a-z`
- `0-9`
- `_` (底線)
- `-` (連字號)
- `.` (點號)

屬性名稱

一般而言，屬性名稱至少必須為一個字元長，但不能超過 64 KB。

以下為例外狀況。這些屬性名稱絕對不能超過 255 個字元。

- 次要索引分割區索引鍵名稱。
- 次要索引排序索引鍵名稱。

- 任何使用者指定的投影屬性名稱 (僅適用於本機次要索引)。在 `CreateTable` 操作中，如果您指定 `ProjectionType` 為 `INCLUDE`，則 `NonKeyAttributes` 參數中的屬性名稱會有長度限制。`KEYS_ONLY` 和 `ALL` 投影類型不受影響。

這些屬性名稱必須使用 UTF-8 編碼，而且各名稱大小總計 (編碼後) 不能超過 255 個位元組。

資料類型

String

字串長度受到項目大小上限 400 KB 的限制。

字串是 UTF-8 二進位編碼的 Unicode。因為 UTF-8 是一種變動寬度的編碼方式，所以 DynamoDB 會使用其 UTF-8 位元組判斷字串的長度。

Number

數字的精準度最多可達 38 位數，可為正數、負數或零。

- 正數範圍：`1E-130` 到 `9.9999999999999999999999999999999999999E+125`
- 負數範圍：`-9.9999999999999999999999999999999999999E+125` 到 `-1E-130`

DynamoDB 使用 JSON 字串表示請求與回覆中的數字資料。如需詳細資訊，請參閱 DynamoDB 低階 API。

若數字精確度很重要，則應使用從數字類型轉換而來的字串，將數字傳遞到 DynamoDB。

二進位

二進位長度受到項目大小上限 400 KB 的限制。

使用二進制屬性的應用程式必須先以 Base64 格式編碼資料，再傳送到 DynamoDB。DynamoDB 收到這些資料後，會將資料解碼為不帶正負號的位元組陣列，並用為屬性長度。

項目

項目大小

DynamoDB 的項目大小上限是 400 KB，包括屬性名稱二進位長度 (UTF-8 長度) 和屬性值長度 (又一個二進位長度)。屬性名稱算作大小限制的一部分。

例如，假設一個有兩個屬性的項目：一個屬性名為 "shirt-color"，其值為 "R"，另一個屬性名為 "shirt-size"，其值為 "M"。該項目的大小總計為 23 位元組。

具有本機次要索引之資料表的項目大小

針對資料表中每一個本機次要索引，每一個於下列總計的限制為 400 KB：

- 資料表的項目資料大小。
- 本機次要索引中對應項目 (包括其索引鍵值及投影屬性) 的大小。

屬性

每個項目的屬性名稱/值對

每個項目的屬性累積大小必須符合 DynamoDB 項目大小上限 (400 KB)。

清單、映射或集合的值數目

清單、映射或集合中值數目不限，只要含有值的項目符合 400 KB 項目大小限制。

屬性值

如果屬性未用作為資料表或索引的索引鍵屬性，則允許空字串和二進制屬性值。集合、清單和映射類型中允許空字串和二進位值。屬性值不得為空的集合 (字串集合、數字集合或二進位集合)。不過，允許空白清單與映射。

巢狀屬性深度

DynamoDB 支援巢狀屬性，最多 32 層深。

表達式參數

表達式參數包括 `ProjectionExpression`、`ConditionExpression`、`UpdateExpression` 和 `FilterExpression`。

長度

任何表達式字串的長度上限為 4 KB。例如，`ConditionExpression a=b` 的大小是 3 個位元組。

任何單一表達式屬性名稱或表達式屬性值的長度上限是 255 個位元組。例如，#name 是 5 個位元組；:val 是 4 個位元組。

表達式中所有替換變數的長度上限為 2 MB。這是所有 ExpressionAttributeNames 和 ExpressionAttributeValues 的長度加總。

運算子和運算元

UpdateExpression 中允許的運算子或函數數目上限為 300。例如，UpdateExpression SET a = :val1 + :val2 + :val3 包含兩個 "+" 運算子。

IN 比較子的運算元數目上限為 100。

保留字

DynamoDB 不會阻止您使用與保留字衝突的名稱。(如需完整清單，請參閱「DynamoDB 中的保留字」。)

但如果您在表達式參數中使用了保留字，您必須也要指定 ExpressionAttributeNames。如需詳細資訊，請參閱 DynamoDB 中的表達式屬性名稱 (別名)。

DynamoDB 交易

DynamoDB 交易 API 操作有下列限制：

- 一個交易不能包含超過 100 個不同的動作。
- 一個交易不能包含超過 4 MB 的資料。
- 交易中的任何兩個動作皆無法對相同資料表中的相同項目進行操作。例如，您不能在一個交易中，對相同項目同時進行 ConditionCheck 與 Update。
- 交易無法在多個 AWS 帳戶或區域中的資料表上操作。
- 交易操作僅在最初進行寫入的 AWS 區域中提供原子性、一致性、隔離性和耐久性 (ACID) 保證。全域資料表不支援跨區域交易。舉例來說，假設您在美國東部 (俄亥俄) 與美國西部 (奧勒岡) 區域中有具有複本的全域資料表，並且在美國東部 (維吉尼亞北部) 區域中執行 TransactWriteItems 操作。這麼做會使變更受到複寫，所以您可能會在美國西部 (奧勒岡) 區域中看到部分已完成交易。當變更已在來源區域遞交後，這些變更才會複寫至其他區域。

DynamoDB Streams

DynamoDB Streams 中的碎片同時讀取

對於不是全域資料表的單一區域資料表，您最多可以設計兩個程序，以便同時讀取同一個 DynamoDB Streams 碎片。超過此限制會導致請求限流。對於全域資料表，我們建議您將同時讀取的數量限制為一個，以避免請求限流。

DynamoDB Accelerator (DAX)

AWS 區域可用性

如需可使用 DAX AWS 的區域清單，請參閱《》中的 DynamoDB Accelerator (DAX)AWS 一般參考。

節點

DAX 叢集包含一個主節點和 0 到 10 個僅供讀取複本節點。

單一 AWS 區域中的節點總數（每個 AWS 帳戶）不得超過 50 個。

參數群組

每個區域最多可以建立 20 個 DAX 參數群組。

子網路群組

每個區域最多可以建立 50 個 DAX 子網路群組。

您可在子網路群組內定義最多 20 個子網路。

> **Important**
>
> DAX 叢集最多支援 500 個 DynamoDB 資料表。一旦超過 500 個 DynamoDB 資料表，叢集可能在可用性和效能方面遭到降級。

API 特定限制

CreateTable/UpdateTable/DeleteTable/PutResourcePolicy/DeleteResourcePolicy

一般而言，您最多可以同時以任何組合執行 500 個 CreateTable、UpdateTable、DeleteTable、PutResourcePolicy 和 DeleteResourcePolicy 請求。換言之，CREATING、UPDATING 或 DELETING 狀態的資料表總數不能超過 500。

您可以跨資料表群組，提交每秒最多 2,500 個可變 (CreateTable、DeleteTable、UpdateTable、PutResourcePolicy 和 DeleteResourcePolicy) 控制平面 API 請求。不過，PutResourcePolicy 和 DeleteResourcePolicy 請求的個別限制較低。如需詳細資訊，請參閱下列 PutResourcePolicy 和 DeleteResourcePolicy 的配額詳細資料。

包含資源型政策的 CreateTable 和 PutResourcePolicy 請求會作為兩個額外請求計入政策的每個 KB。例如，大小 5 KB 的政策的 CreateTable 或 PutResourcePolicy 請求將計為 11 個請求。CreateTable 請求 1 個，資源型政策 10 個 (2 x 5 KB)。同樣地，大小 20 KB 的政策將計為 41 個請求。CreateTable 請求 1 個，資源型政策 40 個 (2 x 20 KB)。

PutResourcePolicy

您可以在一組資料表中每秒提交最多 25 個 PutResourcePolicy API 請求。在針對個別資料表成功請求後，在接下來的 15 秒內不支援新的 PutResourcePolicy 請求。

資源型政策文件的最大支援大小為 20 KB。DynamoDB 在依此限制計算政策大小時，會將空白字元納入計算。

DeleteResourcePolicy

您可以在一組資料表中每秒提交最多 50 個 DeleteResourcePolicy API 請求。在針對個別資料表的 PutResourcePolicy 請求成功後，在接下來的 15 秒內不支援任何 DeleteResourcePolicy 請求。

BatchGetItem

單一 BatchGetItem 操作最多可擷取 100 個項目。所有擷取項目大小總計不能超過 16 MB。

BatchWriteItem

單一 BatchWriteItem 操作最多可以包含 25 個 PutItem 或 DeleteItem 請求。所有寫入項目大小總計不能超過 16 MB。

DescribeStream

您可以每秒最多呼叫 DescribeStream 10 次。

DescribeTableReplicaAutoScaling

DescribeTableReplicaAutoScaling 方式僅支援每秒 10 個請求。

DescribeLimits

DescribeLimits 應只能定期呼叫。如果每分鐘呼叫一次以上，您應該會收到限流錯誤。

DescribeContributorInsights/ListContributorInsights/UpdateContributorInsights

DescribeContributorInsights、ListContributorInsights 和 UpdateContributorInsights 應只能定期呼叫。DynamoDB 於每個 API 支援最多每秒五個請求。

DescribeTable/ListTables/GetResourcePolicy

您每秒最多能提交 2,500 個唯讀 (DescribeTable、ListTables 和 GetResourcePolicy) 控制平面 API 請求的組合。GetResourcePolicy API 的個別限制較低，為每秒 100 個請求。

DescribeTimeToLive

DescribeTimeToLive 操作會限流為每秒 10 個讀取請求單位。如果您超過此限制，DynamoDB 會傳回 ThrottlingException 錯誤。

Query

Query 的結果集受到每次呼叫為 1 MB 的限制。您可以使用查詢回應的 LastEvaluatedKey 擷取更多結果。

Scan

Scan 的結果集受到每次呼叫為 1 MB 的限制。您可以使用掃描回應的 LastEvaluatedKey 擷取更多結果。

UpdateKinesisStreamingDestination

執行 UpdateKinesisStreamingDestination 操作時，最多可以在 24 小時內 ApproximateCreationDateTimePrecision 將設為新值 3 次。

UpdateTableReplicaAutoScaling

UpdateTableReplicaAutoScaling 方式僅支援每秒 10 個請求。

UpdateTableTimeToLive

每小時針對每個指定資料表，此 UpdateTableTimeToLive 方法僅支援一個啟用或停用 Time to Live（TTL）的請求。完整處理此變更最多可能需要一個小時。在此一小時期間內，對相同資料表的額外 UpdateTimeToLive 呼叫會導致 ValidationException 狀況。

DynamoDB 靜態加密

您最多可以在 AWS 擁有的金鑰 AWS 受管金鑰、 和客戶受管金鑰之間切換四次，從建立資料表開始，每個 24 小時時段的任何時候，以每個資料表為基礎。如果過去 6 小時內沒有變化，則允許額外變更。這會有效地將一天的變更次數增加到 8 次 (第一個 6 小時內 4 次變更，以及一天中後續每 6 小時 1 次的變更)。

即使上述配額已用盡，您也可以 AWS 擁有的金鑰 視需要切換加密金鑰以使用 。

除非您請求較高的數量，否則配額如下。若要請求提高服務配額，請參閱 https://aws.amazon.com/support。

DynamoDB API 參考

《Amazon DynamoDB API 參考》包含下列支援的完整操作清單：

- DynamoDB
- DynamoDB Streams
- DynamoDB Accelerator (DAX)

疑難排解 Amazon DynamoDB

下列主題提供當您在使用 Amazon DynamoDB 時，可能遇到的錯誤和問題的疑難排解建議。如果您發現未列在此處的問題，您可以使用此頁面上的 Feedback (意見回饋) 按鈕來報告。

如需更多故障診段建議和常見支援問題的解答，請瀏覽 AWS 知識中心。

主題

- 疑難排解 Amazon DynamoDB 中的內部伺服器錯誤
- 對 Amazon DynamoDB 中的延遲問題進行疑難排解
- 在 Amazon DynamoDB 中針對限流問題進行疑難排解

疑難排解 Amazon DynamoDB 中的內部伺服器錯誤

在 DynamoDB 中，內部伺服器錯誤 (500 錯誤) 表示服務無法處理請求。這些錯誤可能由多種原因導致，例如機群中的暫時性網路問題、基礎設施異常或儲存節點相關問題等。

在 DynamoDB 資料表的生命週期中，您可能會遇到部分內部伺服器錯誤。由於服務的分散式架構，這屬於預期行為，通常無需特別關注。DynamoDB 會即時自動修復服務的任何暫時性問題，無需使用者介入。不過，如果您在對資料表的請求中觀察到持續高比例的內部伺服器錯誤 (如 the section called “SystemErrors” 指標所示)，建議進一步調查。

主題

- 調查內部伺服器錯誤
- 將內部伺服器錯誤的影響降至最低
- 提升營運覺察

調查內部伺服器錯誤

若您在 DynamoDB 資料表中發生內部伺服器錯誤，請考慮以下作法：

1. 檢查 AWS 運作狀態儀表板。

   若要識別問題，第一個步驟是檢查AWS 服務運作狀態儀表板和 AWS 您的帳戶運作狀態儀表板。這些儀表板會提供服務層級問題、受影響的資料表、持續性問題，以及問題解決後的根本原因等重要資訊。

檢閱這些儀表板中的詳細資訊，可讓您更深入了解 AWS 服務 正在使用的 的目前狀態，以及影響您帳戶的任何潛在問題。此資訊可協助您決定後續行動，以解決問題並將營運中斷降至最低。

2. 請聯絡 支援。

   如果您在請求中觀察到長時間持續的錯誤，可能代表服務出現問題。一般而言，如果您在過去 15 分鐘內看到整體故障率為 1% 或更高，則適合將問題呈報至 AWS 支援團隊。如需更多資訊，請參閱 DynamoDB 服務水準協議。

   向 AWS 支援團隊開立案例時，請提供下列詳細資訊，以協助加速疑難排解程序：

   - 受影響的 DDB：資料表或次要索引
   - 錯誤發生的觀察時間範圍
   - DynamoDB 請求 ID (例如 `4KBNVRGD25RG1KE09UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG`)，可在應用程式日誌中找到。

   在支援案例中包含這些詳細資訊，將有助於 AWS 團隊了解問題並提供更快的解決方法。即使沒有請求 ID，也應使用其他可用資訊建立支援案例。

將內部伺服器錯誤的影響降至最低

若在使用 DynamoDB 時發生內部伺服器錯誤，請採取以下最佳實務，以將對應用程式的影響降至最低：

- 使用退避與重試機制 – DynamoDB 的預設 SDK 行為設計旨在為大多數應用程式在退避與重試策略間取得最佳平衡。不過，您可根據應用程式對停機時間的容忍度與效能需求調整這些設定。進一步了解退避與重試機制，以掌握如何微調重試設定。
- 使用最終一致讀取 – 若應用程式不需高度一致性讀取，建議採用最終一致讀取。此類讀取成本較低，且較不易因內部伺服器錯誤而遭遇暫時性問題，因其可由任一可用儲存節點提供。如需詳細資訊，請參閱DynamoDB 讀取一致性。

提升營運覺察

在現今數位環境中，維持應用程式的高可用性與可靠性至關重要。其中一個關鍵作法是主動監控 DynamoDB 資料表與全域次要索引 (GSI) 中的內部伺服器錯誤 (ISE)。透過建立 CloudWatch 警示監控這些錯誤，您可提升營運覺察，並在潛在問題影響最終使用者前收到通知。此方法符合 AWS Well-

Architected Framework 的卓越營運支柱，確保您的 DynamoDB 工作負載已針對效能、安全性和可靠性進行最佳化。

建立 CloudWatch 警示

應在 DynamoDB 資料表上設定 CloudWatch 警示，以接收持續高比例內部伺服器錯誤的通知，而非手動監控指標。這與適用於任何工作負載的 Well-Architected 架構的卓越營運支柱有關 AWS。如需進一步了解如何依 Well-Architecting 原則設計您的 DynamoDB 資料表，請參閱 使用 DynamoDB Well-Architected Lens 來最佳化您的 DynamoDB 工作負載。

這些警示使用自訂指標運算，以計算 5 分鐘區間內的失敗請求比例。建議的最佳實務是將警示設定為當連續三個資料點超過 1% 閾值時進入 ALARM 狀態，代表在 15 分鐘內約有 1% 的請求失敗。

以下範例是 CloudFormation 範本，可協助您在資料表上建立 CloudWatch 警示，並在資料表上建立 GSI。

```
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'
```

對 Amazon DynamoDB 中的延遲問題進行疑難排解

若您的工作負載似乎出現高延遲，您可以分析 CloudWatch SuccessfulRequestLatency 指標，並透過百分位數指標 (p50) 檢視平均與中位延遲，以判斷問題是否與 DynamoDB 有關。所回報的 SuccessfulRequestLatency 出現部分變化屬正常現象，偶發的尖峰 (尤其是在 Maximum 統計資料與高百分位數中) 無需特別擔心。然而，若 Average 統計資料或 p50 (中位數) 持續急遽上升，請檢視 AWS 服務健康狀態儀表板及個人健康儀表板，以取得更多資訊。可能的原因包括資料表中項目的大小 (1 KB 項目與 400 KB 項目的延遲會有所差異)，或查詢的範圍 (10 個項目與 100 個項目相比)。

百分位數指標 (p99、p90 等) 可協助您更深入了解延遲分布情形。例如：

- p50 (中位數) 代表您的工作負載的典型延遲值。
- p90 表示有 90% 的請求速度快於此值。
- p99 有助於識別影響 1% 請求的最差延遲情況。

若 p99 值偏高但 p50 值正常，可能代表零星問題影響少部分請求；若 p50 值持續升高，則可能顯示效能衰退。

> **Note**
>
> 若要分析自訂百分位數值 (例如 p99.9)，可在 CloudWatch 指標統計欄位中手動輸入所需百分位數。這可讓您評估超出下拉選單預設百分位數範圍的延遲分布。

延遲指標出現變化 (特別是在高百分位數) 屬預期情形，通常是由 DynamoDB 背景作業造成，這些作業可協助維持儲存在 DynamoDB 資料表中的資料高可用性與耐用性，或反映暫時性基礎設施問題。

如有必要，請考慮使用 AWS 支援 建立支援案例，然後根據您的執行手冊，繼續評估您應用程式的任何可用回復選項 (例如，如果您有多區域架構，則撤離一個區域)。您應記錄慢速請求的請求 ID，以便在開啟支援案例時提供給 AWS 支援。

`SuccessfulRequestLatency` 指標僅衡量 DynamoDB 服務內部延遲，不包含用戶端活動與網路往返時間。若要深入了解用戶端呼叫 DynamoDB 服務的整體延遲，您可以在 AWS SDK 中啟用延遲指標記錄。

> **Note**
>
> 對於大部分單一操作 (透過完全指定主索引鍵的值來套用至單一項目的操作)，DynamoDB 提供個位數的毫秒 `Average SuccessfulRequestLatency`。此值不包括存取 DynamoDB 端點之呼叫者程式碼的傳輸負荷。對於多項目資料操作，延遲會因結果集的大小、傳回資料結構的複雜度，以及套用的任何條件表達式和篩選條件表達式等因素而有所不同。對於具有相同參數之相同資料集的重複多項操作，DynamoDB 提供高度一致的 `Average SuccessfulRequestLatency`。

請考慮下列一或多種策略來減少延遲：

- 重複使用連線：DynamoDB 請求預設透過 HTTPS 的已驗證工作階段傳送。建立連線需經多次往返且耗時，因此第一個請求的延遲通常高於後續重複使用連線的請求。經由已啟動連線的請求，可以提供 DynamoDB 一致的低延遲。為減少建立新連線的延遲負擔，若無其他 `GetItem` 請求，建議每 30 秒傳送一次請求，以實作「保持連線」機制。
- 使用最終一致讀取：如果您的應用程式不需要高度一致性讀取，請考慮使用預設的最終一致讀取。最終一致讀取的成本較低，且可從多個可用區域讀取資料，可選擇與請求端共置的可用區域，以降低延遲。如需更多詳細資訊，請參閱 DynamoDB 讀取一致性。
- 實作請求對沖：若對 p99 延遲有極低要求，建議考慮實作請求對沖機制。在使用請求對沖時，若初始請求未能及時收到回應，系統會傳送第二個等效請求，並讓兩者同時競爭，以最先回應者為準。此

方法可改善尾端延遲，但需付出額外請求的成本。您可自行設定在傳送第二個請求前的等待時間。對於讀取操作，對沖策略較容易實作。對於寫入操作，請使用以時間戳記為基礎的排序機制，以確保對沖請求視為在首次嘗試時發生，防止更新順序錯亂。此方法已在 Amazon DynamoDB 寫入對沖的時間戳記寫入一文中詳述。

- 調整請求逾時與重試行為：從用戶端到 DynamoDB 的傳輸路徑會經過多個元件，且各元件皆具備備援設計。請考量以下面向：
  - 網路韌性
  - TCP 封包逾時
  - DynamoDB 的分散式架構

  SDK 的預設行為已針對多數應用程式進行最佳化。不過，您也可採用快速檢錯策略，並視需要調整逾時設定。明顯超出一般時間的請求，最終成功機率較低。採用快速檢錯並重新嘗試的方式，您可以透過其他路徑更快完成請求。此策略類似於請求對沖，但會終止第一個請求，而非讓其持續執行。

  避免將逾時值設定得過低。過低的逾時設定可能造成用戶端引起的可用性問題。例如，若通訊端逾時設定為 50 毫秒，當網路延遲出現尖峰 (例如單一流量接近 Amazon EC2 執行個體頻寬上限) 時，可能導致連線錯誤。請仔細權衡較低逾時設定的效益與其對應用程式可用性造成的風險，並建議優先採用對沖策略，而非縮短逾時。

  如需更深入的討論，請參閱為延遲感知的 Amazon DynamoDB 應用程式調整 AWS Java SDK HTTP 請求設定。
- 減少用戶端與 DynamoDB 端點之間的距離：如果您的使用者遍布全球，請考慮使用 全域資料表：多重作用中的多區域複寫。透過全域資料表，您可將資料表複寫至指定的 AWS 區域，以確保在所需區域可供使用。可將資料複本部署於接近終端使用者的位置，以降低讀寫操作的網路延遲。如需深入了解如何有效使用 DynamoDB 全域資料表，請參閱使用 Amazon DynamoDB 全域資料表 (AWS 規範指引)。
- 使用快取：若您的應用以讀取為主，請考慮以下快取服務：
  - DynamoDB 加速器 (DAX)：一種全受管、高可用性的記憶體快取，可為 DynamoDB 帶來高達 10 倍的效能提升，將延遲從毫秒降至微秒，即使每秒處理數百萬次請求亦能維持效能。如需 DAX 的詳細資訊，請參閱 使用 DynamoDB Accelerator (DAX) 的記憶體內加速：
  - Amazon ElastiCache：全受管的記憶體快取服務，可與 DynamoDB 整合，並採用快取旁路模式以提升讀取效能。如需更多資訊，請參閱使用讀取快取整合 Amazon DynamoDB 與 Amazon ElastiCache (AWS 規範指引)。

在 Amazon DynamoDB 中針對限流問題進行疑難排解

DynamoDB 實作限流機制的兩個主要目的為：維持整體服務效能與成本控制。限流可作為主動防護措施，在使用量超出容量時防止效能下降，或於達到最大輸送量或服務配額時發揮成本控制作用。發生限流時，DynamoDB 會回傳特定例外狀況，說明請求被限流的原因及受影響的資源。每個限流原因皆對應特定的 CloudWatch 指標，能提供限流事件頻率與模式的進一步洞見。

下圖說明 DynamoDB 採用保護性限流機制的四種主要情境：

1. 金鑰範圍輸送量超出上限 (適用於兩種模式)：

   對特定分割區的存取量超出內部分割區層級輸送量限制。

2. 佈建輸送量超出限制 (於佈建模式中)：

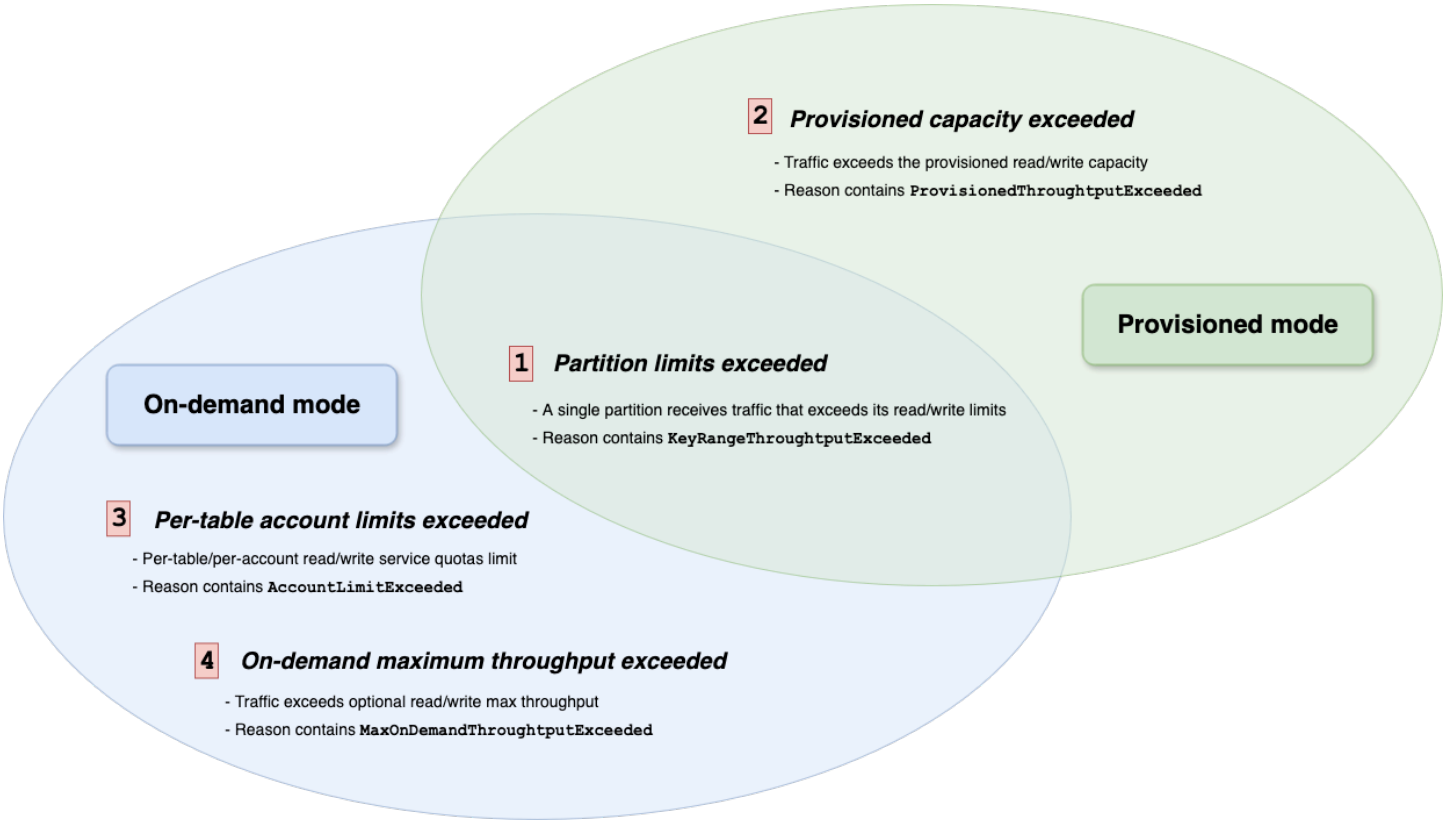
   使用量超出為資料表或全域次要索引 (GSI) 設定的佈建容量單位 (讀取或寫入)。

3. 帳戶層級服務配額超出限制 (於隨需模式中)：

   使用會導致資料表或 GSI 超過目前 AWS 區域中讀取/寫入輸送量的每個資料表帳戶層級服務配額。這些配額作為後備防護機制，並可依需求提升。

4. 超過隨需輸送量上限 （在隨需模式下）：

   使用量超出為資料表或 GSI 所設定的輸送量上限。這些限制是您為控制成本而特別設定的。



本指南旨在協助您理解並處理 DynamoDB 的限流機制。首先，我們將協助您透過診斷架構辨識影響工作負載的限流類型。

接著，解決方案指南章節針對各限流情境提供具體建議，包括用於偵測與分析的 CloudWatch 監控指標，以及最佳化步驟建議。採用此結構化方法可協助您更準確地診斷限流根因，並實施適當解決方案，以確保 DynamoDB 資料表高效運作。

開始之前，請依照 the section called “限流診斷” 了解如何判定影響工作負載的限流類型，並採用建議的解決策略。

主題

- 限流診斷
- DynamoDB 限流解決指南
- 認識 DynamoDB 中全域次要索引 (GSI) 的寫入限流與背壓現象
- CloudWatch 限流度量指標

限流診斷

當應用程式發生限流時，DynamoDB 會提供詳細例外狀況資訊與對應的 CloudWatch 指標，協助您進行診斷。

本節說明如何以系統化方式了解 DynamoDB 應用程式中的限流事件。說明如何解譯限流例外狀況，並將其與 CloudWatch 指標關聯，以獲得更深入的洞察，進而了解可減少 DynamoDB 限流的變更。

了解限流例外狀況

當 DynamoDB 對請求進行限流時，會傳回包含詳細診斷資訊的特定例外狀況。例如，在 Java 中，這些例外狀況可能包括 `ProvisionedThroughputExceededException`、`RequestLimitExceeded` 或 `ThrottlingException`。

每個例外狀況都包含 `ThrottlingReasons`，其中的 `ThrottlingReason` 集合包含兩個關鍵欄位，可協助識別並理解限流原因：

- 原因 - 依照 `<ResourceType><OperationType><LimitType>` 格式串接的欄位
- 資源 ARN - 受影響資料表或索引的 Amazon Resource Name (ARN)

原因欄位遵循一致模式，可協助精確判斷事件發生原因：

- ResourceType (受限流的資源)：`Table` 或 `Index`
- OperationType (操作類型)：`Read` 或 `Write`
- LimitType (限流發生原因)：
  - `KeyRangeThroughputExceeded`：當支援資料表或索引的特定分割區耗用的讀寫容量超過內部分割區輸送量限制時，會發生此情況。
  - `ProvisionedThroughputExceeded`：當讀取或寫入消耗率超過預先佈建的容量時，此情況會發生在佈建資料表或全域次要索引上。
  - `AccountLimitExceeded`：當讀取或寫入消耗率超過帳戶層級設定的資料表及其索引的最大允許消耗率時，此情況會發生在隨需資料表或索引上。您可以提出請求以提高此配額。
  - `MaxOnDemandThroughputExceeded`：當讀取或寫入消耗率超過使用者為資料表或索引設定的最大允許消耗率時，這種情況會發生在隨需資料表或索引上。您可以自行將此值提高至帳戶限制範圍內的任意值，或設定為 -1 以表示未設定使用者自訂上限。

資源 ARN 可精確識別正在被限流的資料表或索引：

- 資料表：arn:aws:dynamodb:[region]:[account-id]:table/[table-name]
- 索引：arn:aws:dynamodb:[region]:[account-id]:table/[table-name]/index/[index-name]

完整限流原因範例：

- TableReadProvisionedThroughputExceeded
- IndexWriteAccountLimitExceeded

這有助於準確識別被限流的資源、造成限流的操作類型，以及限流發生的原因。

例外範例

範例 1：GSI 超出佈建容量

```
{
    "ThrottlingReasons": [
        {
            "reason": "IndexWriteProvisionedThroughputExceeded",
            "resource": "arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders/
index/OrderDateIndex"
        }
    ],
    "awsErrorDetails": {
        "errorCode": "ProvisionedThroughputExceeded",
        "errorMessage": "The level of configured provisioned throughput for the index
 was exceeded",
        "serviceName": "DynamoDB",
        "sdkHttpResponse": {
            "statusText": "Bad Request",
            "statusCode": 400
        }
    }
}
```

在此範例中，應用程式會收到原因為 IndexWriteProvisionedThroughputExceeded 的 ProvisionedThroughputExceededException。對 OrderDateIndex 的寫入正在被限流，因為寫入消耗已超出 GSI 所設定的佈建寫入容量。

範例 2：超過隨需輸送量上限

```
{
    "ThrottlingReasons": [
        {
            "reason": "TableReadMaxOnDemandThroughputExceeded",
            "resource": "arn:aws:dynamodb:us-east-1:123456789012:table/UserSessions"
        }
    ],
    "awsErrorDetails": {
        "errorMessage": "Throughput exceeds the maximum OnDemandThroughput configured
 on table or index",
        "errorCode": "ThrottlingException",
        "serviceName": "DynamoDB",
        "sdkHttpResponse": {
            "statusText": "Bad Request",
            "statusCode": 400
        }
    }
}
```

在此範例中，來自 UserSessions 資料表的讀取請求被限流，因為其超出該資料表設定的隨需輸送量上限。

DynamoDB 限流診斷架構

當應用程式發生限流時，請依下列步驟進行診斷與解決。

步驟 1 - 分析 ThrottlingReason 詳細資訊

1. 檢查原因欄位，以識別限流的具體原因。該原因會詳細說明被限流的資源類型 (資料表或索引)、造成限流事件的操作類型 (讀取或寫入)，以及超出的限制類型 (分割區、佈建輸送量、帳戶限制)。
2. 檢查 resourceArn 欄位，以識別正在被限流的資源 (資料表或全域次要索引)。
3. 使用這些綜合資訊以了解限流問題的完整脈絡。

   例如，在下列情境中，您會收到 ProvisionedThroughputExceededException 例外狀況，其限流原因為 TableWriteKeyRangeThroughputExceeded。受影響的 resourceARN 為 arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders。

此組合表示您的 `CustomerOrders` 資料表寫入操作正被限流。限流發生在分割區層級 (非資料表層級，該層級會顯示為 `TableWriteProvisionedThroughputExceeded`)。根本原因在於超出特定分割區鍵值或範圍的最大輸送容量，顯示這是熱分割區問題。

了解例外狀況元素間的關聯有助於您採取正確的緩解策略——在此情境中，應處理熱分割區，而非提高資料表的整體佈建容量。

步驟 2 - 識別並分析相關的 CloudWatch 指標

1. 識別您的指標：DynamoDB 中的每個限流原因都直接對應可供監控的特定 CloudWatch 指標，用以追蹤並分析限流事件。您可以根據限流原因系統性地推導出對應的 CloudWatch 指標名稱。
2. 使用下列資料表將限流原因對應至相應的 CloudWatch 指標：

   限流原因與 CloudWatch 指標完整對照表

Category	限流原因	主要 CloudWatch 指標
佈建容量超出	TableReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents
	TableWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents
	IndexReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents (GSI)
	IndexWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents (GSI)
分割區限制超出	TableReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents

Category	限流原因	主要 CloudWatch 指標
	TableWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents
	IndexReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents (GSI)
	IndexWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents (GSI)
隨需上限超出	TableReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents
	TableWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents
	IndexReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents (GSI)
	IndexWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents (GSI)
超過帳戶限制	TableReadAccountLimitExceeded	ReadAccountLimitThrottleEvents
	TableWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents

Category	限流原因	主要 CloudWatch 指標
	IndexReadAccountLimitExceeded	ReadAccountLimitThrottleEvents (GSIs)
	IndexWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents (GSIs)

例如，若您收到 `IndexWriteProvisionedThroughputExceeded`，至少應監控 `ResourceArn` 中所識別之特定索引的 `WriteProvisionedThroughputThrottleEvents` CloudWatch 指標。

3. 在 CloudWatch 中監控這些指標，以掌握限流事件的發生頻率與時機、區分讀取與寫入限流、識別限流增加的時間模式，並追蹤容量使用趨勢。

   DynamoDB 會針對每個資料表與全域次要索引發布詳細的監控指標。指標 `ReadThrottleEvents`、`WriteThrottleEvents`、`ThrottledRequests` 會彙整資料表及其索引中的所有限流事件。

步驟 3 - 使用 CloudWatch Contributor Insights 識別被限流的金鑰與高存取率 (適用於分割區層級的限流)

若您在步驟 1 中發現分割區相關問題 (例如 `KeyRangeThroughputExceeded` 錯誤)，適用於 DynamoDB 的 CloudWatch Contributor Insights 可協助診斷哪些特定金鑰造成流量集中，並在資料表或索引中發生限流事件。

1. 根據您的 `ResourceARN`，為遭受限流的資料表或索引啟用 CloudWatch Contributor Insights。

   您可以選擇限流金鑰模式，僅專注於遭受限流次數最多的金鑰。此模式非常適合持續監控，因為僅在限流發生時才會處理事件。或者，您可以使用存取與限流金鑰模式，以協助您尋找最常被存取金鑰中的存取模式。

2. 分析報告，以辨識潛在的異常模式。找出存取或限流速率異常偏高的金鑰，並分析限流與流量之間的關聯模式。您可以結合 Contributor Insights 圖表與 DynamoDB CloudWatch 指標，建立整合式儀表板。

如需啟用與使用 CloudWatch Contributor Insights 的詳細資訊，請參閱使用 CloudWatch Contributor Insights 於 DynamoDB。

步驟 4 - 判定合適的解決方案

完成限流原因診斷後，依據實際情境實作建議的解決方案。適當的解決方案取決於多項因素，包括限流情境、資料表的容量模式、資料表與金鑰設計、存取模式與查詢效率、全域與次要索引設定，以及整體系統架構與整合節點。

如需針對特定限流情境的詳細解決方案，請參閱 the section called “解決指南” 章節。此資源提供依您的限流原因與容量模式設定量身調整的修復策略。

步驟 5 - 監控執行成效

1. 追蹤與限流情境對應的 CloudWatch 指標。
2. 透過觀察限流事件次數的下降，確認緩解策略的有效性。

DynamoDB 限流解決指南

本節針對 DynamoDB 可能傳回的各項特定限流原因，提供具體的解決指引。每個項目皆提供根據最佳實務的建議解決方法，並附上需監控的對應 CloudWatch 指標。

DynamoDB 於四大類別中定義了 16 種不同的限流原因。根據應用程式例外狀況中的限流原因，直接導向對應的指引內容。

金鑰範圍輸送量超出上限 (熱分割區)

當個別分割區超出輸送量限制時，將同時影響佈建模式與隨需模式，並觸發這些限流原因：

- TableReadKeyRangeThroughputExceeded
- TableWriteKeyRangeThroughputExceeded
- IndexReadKeyRangeThroughputExceeded
- IndexWriteKeyRangeThroughputExceeded

佈建輸送量超出上限

當使用率在佈建模式中超出設定的容量上限時，會觸發這些限流原因：

- TableReadProvisionedThroughputExceeded
- TableWriteProvisionedThroughputExceeded
- IndexReadProvisionedThroughputExceeded

- IndexWriteProvisionedThroughputExceeded

超過帳戶限制

當使用率超過您 AWS 區域中的帳戶層級輸送量配額時，會發生這些調節原因：

- TableReadAccountLimitExceeded
- TableWriteAccountLimitExceeded
- IndexReadAccountLimitExceeded
- IndexWriteAccountLimitExceeded

超過隨需輸送量上限

當使用率在隨需模式下超出設定的最大輸送量上限時，會觸發這些限流原因：

- TableReadMaxOnDemandThroughputExceeded
- TableWriteMaxOnDemandThroughputExceeded
- IndexReadMaxOnDemandThroughputExceeded
- IndexWriteMaxOnDemandThroughputExceeded

1 - 金鑰範圍輸送量超出上限 (熱分割區)

Amazon DynamoDB 會在資料表與全域次要索引 (GSI) 的分割區層級上強制執行特定的輸送量限制。每個分割區每秒可支援的最大讀取容量單位 (RCU) 與寫入容量單位 (WCU)。當分割區承受超出限制的集中流量時，會發生限流，而其他操作可能因使用率偏低而形成「熱分割區」。DynamoDB 的分割區層級限流會分別針對讀取與寫入獨立運作——分割區可能在寫入仍正常時限制讀取，或反之亦然。即使資料表或 GSI 具備充足的整體容量，仍可能出現限流情況。進一步了解以下主題：

- 如需了解 DynamoDB 的分割區限制及可有效防止熱分割區的索引鍵設計，請參閱在 DynamoDB 中有效設計與使用分割區索引鍵的最佳實務。
- 如需了解分割區的一般概念與資料分佈，請參閱 DynamoDB 中的分割區。
- 如需更多關於管理分割區索引鍵與輸送量的指引與實務案例，請參閱 the section called “其他資源”。

當個別分割區超出輸送量限制時，DynamoDB 會在限流例外狀況中傳回 `KeyRangeThroughputExceeded` 限流原因類型。此資訊可指出哪個分割區正遭遇高流量，以及是由哪種操作類型 (讀取或寫入) 造成問題。

金鑰範圍輸送量超出上限緩解措施

本節提供針對分割區層級限流情境的處理指引。使用本指南前，請先從應用程式的例外狀況處理中確認具體的限流原因，並確定受影響資源的 Amazon Resource Name (ARN)。如需了解如何擷取限流原因並識別遭限流的資源，請參閱 the section called “診斷架構”。

在深入分析特定限流情境之前，請先確認問題是否會自動解決：

- DynamoDB 通常會透過其自動 split-for-heat 機制來因應熱分割區。若限流事件在短時間內停止，表示資料表可能已自動透過分割熱分割區完成調整。當分割區被分割後，每個新分割區將處理金鑰空間中較小的範圍，有助於更平均地分散負載。多數情況下無需額外操作，因為 DynamoDB 已自動修復問題。

  如需了解 split-for-heat 機制的詳細資訊，請參閱 the section called “其他資源”。

若限流情況持續發生，請參閱下列特定限流情境，以採取相應的修正方案：

- the section called “TableReadKeyRangeThroughputExceeded”
- the section called “TableWriteKeyRangeThroughputExceeded”
- the section called “IndexReadKeyRangeThroughputExceeded”
- the section called “IndexWriteKeyRangeThroughputExceeded”

TableReadKeyRangeThroughputExceeded

發生這種情況時

DynamoDB 資料表中一個或多個分割區的消耗速率超出分割區的讀取輸送量限制。無論資料表的總佈建容量為何，仍會發生此限流，且同時影響佈建資料表及隨需資料表。可透過 the section called “常見診斷與監控” 監控 CloudWatch 指標，以分析限流事件。

修正方案

可參考以下步驟處理限流事件：

適用於佈建與隨需模式：

- 預熱容量：若限流持續，請檢查資料表是否受限於其 the section called “暖輸送量” 容量。對於預期流量增加，可預先使用暖輸送量或提升讀取佈建容量。提高暖輸送量可增強資料表在限流發生前處理突發流量尖峰的能力。長期觀察下，若實際輸送量持續接近暖輸送量層級，DynamoDB 可能會依據觀察到的使用模式分割繁忙分割區。
- 識別您的熱鍵：若資料表未自動處理，且暖輸送量已高或增加仍無效，您需識別特定熱鍵。使用 the section called “使用 CloudWatch Contributor Insights 識別熱鍵” 判斷特定分割區索引鍵值是否為熱鍵。這是有效鎖定緩解措施的第一步。請注意，識別不一定簡單，尤其是對於滾動熱分割區 (不同分割區隨時間成為熱鍵) 或當限流由掃描等操作觸發時。對於這些複雜情況，您可能需要分析應用程式的存取模式，並將其與限流事件的時間進行對應分析。
- 依據您的使用案例，請考慮採用最終一致讀取：由強一致讀取切換至最終一致讀取，可節省一半 RCUs，並立即將有效讀取容量加倍。如需了解實作最終一致讀取以降低讀取容量消耗的最佳實務，請參閱 the section called “DynamoDB 讀取一致性”。
- 改善分割區索引鍵設計：作為長期解決方案，請考慮 the section called “改善分割區索引鍵設計” 在各分割區間更均衡地分配存取量。這種方法通常透過解決根本原因，對熱分割區問題提供最完整的解決方案。不過，這需要謹慎規劃，因為涉及大量移轉挑戰。

TableWriteKeyRangeThroughputExceeded

發生這種情況時

您的 DynamoDB 資料表中一個或多個分割區的消耗率超過其寫入輸送量限制。無論資料表的總佈建容量為何，仍會發生此限流，且同時影響佈建資料表及隨需資料表。可透過 the section called “常見診斷與監控” 監控 CloudWatch 指標，以分析限流事件。

修正方案

可參考以下步驟處理限流事件：

適用於佈建與隨需模式：

- 預熱容量：若限流持續，請檢查資料表是否受限於其 認識 DynamoDB 暖輸送量 容量。可針對預期流量增加，預先使用暖輸送量或提高寫入佈建容量。提高暖輸送量可增強資料表在限流發生前處理突發流量尖峰的能力。長期觀察下，若實際輸送量持續接近暖輸送量層級，DynamoDB 可能會依據觀察到的使用模式分割繁忙分割區。
- 識別您的熱鍵：若資料表未自動處理，且暖輸送量已高或增加仍無效，您需識別特定熱鍵。使用 the section called “使用 CloudWatch Contributor Insights 識別熱鍵” 判斷特定分割區索引鍵值是否為熱鍵。這是有效鎖定緩解措施的第一步。可參考以下常見模式：
  - 若在限流資料中頻繁出現相同分割區索引鍵，表示該熱鍵集中。

- 若未見重複金鑰，但以有序方式寫入資料 (如連續時間戳或依金鑰空間順序的掃描操作)，可能存在滾動熱分割區，不同金鑰會隨時間變為熱鍵。

請注意，寫入限流也可能發生於操作，例如同時影響多個項目的 `BatchWriteItem` 或交易。當 `BatchWriteItem` 請求中的單個項目受限流影響時，DynamoDB 不會將這些限流錯誤傳遞至應用程式程式碼。反之，DynamoDB 會在回應中返回未處理項目資訊，應用程式需針對這些項目進行重試處理。對於交易，若任一項目遇到限流，整個操作會以 `TransactionCanceledException` 失敗。對於這些複雜情況，您可能需要分析應用程式的寫入模式及資料擷取工作流程，將其與限流事件的時間對應，並實施適當的重試策略。

- 改善分割區索引鍵設計：作為長期解決方案，請考慮 the section called “改善分割區索引鍵設計” 在各分割區間更均衡地分配存取量。這種方法通常透過解決根本原因，對熱分割區問題提供最完整的解決方案。不過，這需要謹慎規劃，因為涉及大量移轉挑戰。

IndexReadKeyRangeThroughputExceeded

發生這種情況時

DynamoDB GSI 中一或多個分割區的消耗率超過該分割區的讀取輸送量限制。無論您的 GSI 總佈建容量為何，仍會發生此限流，且同時影響佈建資料表及隨需資料表。可透過 the section called “常見診斷與監控” 監控 CloudWatch 指標，以分析限流事件。

修正方案

可參考以下步驟處理限流事件：

- 預熱容量：若限流持續存在，請檢查您的 GSI 是否受限於其 認識 DynamoDB 暖輸送量 容量。對於預期流量增加，可預先使用暖輸送量或提升讀取佈建容量。提高暖輸送量可增強 GSI 在限流發生前處理突發流量峰值的能力。長期觀察下，若實際輸送量持續接近暖輸送量層級，DynamoDB 可能會依據觀察到的使用模式分割繁忙分割區。
- 識別您的熱鍵：若 GSI 未自動解決，且暖輸送量高或提升後仍無效，您需識別特定熱鍵。使用 the section called “使用 CloudWatch Contributor Insights 識別熱鍵” 判斷特定分割區索引鍵值是否為熱鍵。這是有效鎖定緩解措施的第一步。請注意，對於 GSIs，分割區金鑰分佈可能與基礎資料表顯著不同，形成不同的熱鍵模式。
- 重新設計 GSI 分割區索引鍵：考慮 GSI 索引鍵設計是否可能產生人工熱點 (如狀態旗標、僅日期索引鍵或布林屬性)，導致讀取集中於少數分割區。考慮使用將低基數屬性與高基數屬性 (如「ACTIVE#customer123」而非僅「ACTIVE」) 結合的複合索引鍵，或將 the section called “寫入碎片” 技術套用至影響 GSI 分佈的基礎資料表項目，以將寫入分散到多個分割區。雖然查詢碎片資料需要額外應用程式邏輯來彙總結果，但此方法可透過更平均的存取模式分佈來避免限流。

IndexWriteKeyRangeThroughputExceeded

發生這種情況時

DynamoDB GSI 中一或多個分割區的消耗率超過該分割區的寫入輸送量限制。無論您的 GSI 總佈建容量為何，仍會發生此限流，且同時影響佈建資料表及隨需資料表。可透過 the section called “常見診斷與監控” 監控 CloudWatch 指標，以分析限流事件。

修正方案

可參考以下步驟處理限流事件：

- 重新設計 GSI 分割區索引鍵：檢閱 GSI 分割區索引鍵設計，以確認其基數 (唯一性) 足以平均分配寫入。GSI 寫入限流的常見原因是使用低基數屬性作為 GSI 分割區索引鍵 (如僅有幾個可能值的狀態旗標)。即使基礎資料表分割區索引鍵分佈良好，若 GSI 分割區索引鍵將寫入集中於少數值，GSI 仍可能出現熱分割區。例如，若 80% 項目 status="ACTIVE"，將在狀態型 GSI 中形成嚴重熱分割區。考慮使用將低基數屬性與高基數屬性 (如「ACTIVE#customer123」而非僅「ACTIVE」) 結合的複合索引鍵，或將 the section called “寫入碎片” 技術套用至影響 GSI 分佈的基礎資料表項目，以將寫入分散到多個分割區。雖然查詢碎片資料需要額外應用程式邏輯來彙總結果，但此方法可透過更平均的存取模式分佈來避免限流。
- 預熱容量：檢查您的 GSI 是否受限於其認識 DynamoDB 暖輸送量容量。可針對預期流量增加，預先使用暖輸送量或提高寫入佈建容量。提高暖輸送量可增強 GSI 在限流發生前處理突發流量峰值的能力。長期觀察下，若實際輸送量持續接近暖輸送量層級，DynamoDB 可能會依據觀察到的使用模式分割繁忙分割區。
- 最佳化 GSI 投影：套用 the section called “最佳化 GSI 投影” 技術以降低 GSI 寫入量。投影較少屬性可大幅降低每次 GSI 更新的寫入容量消耗。

常見診斷與監控

疑難排解分割區層級限流時，數個 CloudWatch 指標可協助識別熱分割區並確認根本原因。

主要 CloudWatch 指標

監控這些核心指標以診斷分割區層級限流：

- 分割區層級限流事件：`ReadKeyRangeThroughputThrottleEvents` 和 `WriteKeyRangeThroughputThrottleEvents` 追蹤個別分割區何時超過其輸送量限制。`ReadThrottleEvents` 和 `WriteThrottleEvents` 追蹤任何讀或寫請求何時超過佈建容量。

- 容量耗用情形：ConsumedReadCapacityUnits 和 ConsumedWriteCapacityUnits 顯示整體使用模式。

解決步驟

使用 CloudWatch Contributor Insights 識別熱鍵

使用此程序識別哪些分割區索引鍵引起限流。

1. 在資料表或 GSI 上啟用 CloudWatch Contributor Insights，以追蹤限流最嚴重的金鑰。考慮使用限流金鑰模式，持續啟用 CloudWatch 貢獻者洞察以進行即時限流警示。此模式僅針對限流請求，僅在發生限流時處理事件。此目標監控是以經濟有效的方式持續監控限流問題。
2. 識別造成熱分割區問題的金鑰。
3. (若啟用完整存取和限流金鑰模式) 分析一段時間的存取模式，以判斷熱鍵是否持續存在或僅在特定時段出現。

改善分割區索引鍵設計

當您能修改資料表結構以更均衡地分配分割區流量時，請使用此方法。若可行，這是解決熱分割區問題最有效的長期方案。理想情況下，應在初始資料表設計階段仔細規劃分割區索引鍵設計。

分割區索引鍵重新設計是對資料模型的根本性變更，將影響整個應用程式生態系統。在採用此方法前，請仔細評估以下重大限制：

- 資料移轉複雜性：重新設計分割區索引鍵需移轉所有現有資料，對大型資料表而言可能非常耗費資源且耗時。
- 應用程式碼變更：所有與資料表讀寫相關的程式碼都需更新，以使用新的索引鍵結構。
- 生產影響：移轉至新索引鍵設計通常需停機或採用複雜的雙寫策略。

如需分割區索引鍵設計的完整指引與原則，請參閱 the section called “分割區索引鍵設計” 和 the section called “分佈工作負載”。

最佳化 GSI 投影

檢視應用程式的查詢模式，確認查詢 GSI 時需要的屬性，並將投影限制為僅這些屬性。當您更新未投影到 GSI 的屬性時，該 GSI 不會執行寫入操作，從而減少更新期間的寫入輸送量消耗。此目標投影策略可同時最佳化效能與成本，並持續支援應用程式的查詢需求。請注意，投影較少屬性可降低寫入容量消耗，但可能需要額外讀取基礎資料表。

如需了解高效投影策略的詳細資訊，請參閱在 DynamoDB 中使用次要索引的最佳實務。

其他資源

下列部落格文章提供本指南涵蓋概念的實作範例與實務細節：

- 如需關於擴展 DynamoDB 及管理熱分割區的實作指引，請參閱第 1 部分：擴展 DynamoDB – 分割區、熱鍵及熱分割對效能的影響。
- 如需了解 DynamoDB split-for-heat 機制的運作方式、優點與實作細節，請參閱第 3 部分：摘要與最佳實務。
- 如需詳細寫入碎片策略，請參閱 the section called “寫入碎片”。

2 - 佈建輸送量超出上限

當應用程式的使用率超過為資料表或全域次要索引設定的讀取或寫入容量單位 (RCU/WCU) 時，系統會發生佈建容量限流。雖然 DynamoDB 提供高載容量以應對偶發流量尖峰，但若請求持續超出佈建限制，仍會觸發限流。當發生這種情況時，DynamoDB 會在限流例外中傳回 `ProvisionedThroughputExceeded` 限流原因類型。此原因可辨識問題屬於讀取或寫入操作，並指出影響範圍是主資料表或全域次要索引。

無論是否啟用 Auto Scaling，皆可能發生限流。Auto Scaling 會隨使用量增加自動調整，但回應並非即時，且受您設定的最大容量上限限制。因此，當流量突增或使用量超出 Auto Scaling 上限時，仍可能出現限流。

佈建輸送量超出上限的緩解措施

本節提供佈建容量限流情境的解決指引。使用本指南前，請先確認您已在應用程式的例外處理中識別出特定限流原因，並找出受影響資源的 Amazon Resource Name (ARN)。如需了解如何擷取限流原因與識別受限流的資源，請參閱 the section called “診斷架構”。

在深入特定限流情境前，請先評估該限流是否真需處理：

- 在經過良好最佳化的 DynamoDB 應用程式中，偶發限流是正常且可預期的。限流僅表示您已使用完佈建的全部容量。若您的應用程式能透過重試機制妥善處理限流，且整體效能符合需求，則無須立即因限流採取行動。
- 然而，若限流導致用戶端延遲過高、使用者體驗下降，或關鍵操作無法及時完成，請採取以下緩解措施。

當需要處理限流問題時，請先確定限流的成因：

- 暫時性流量尖峰：短時間內超出佈建容量但不持續的流量增加。這類情況所需策略與持續高流量有所不同。
- 持續高流量：長期工作負載穩定超出您佈建的容量。

對於流量尖峰，請參考使用 Amazon DynamoDB 佈建容量處理流量尖峰部落格中的策略，詳見 the section called “其他資源”。

若為持續高流量，請考慮以下容量調整方法：

- the section called “TableReadProvisionedThroughputExceeded”
- the section called “TableWriteProvisionedThroughputExceeded”
- the section called “IndexReadProvisionedThroughputExceeded”
- the section called “IndexWriteProvisionedThroughputExceeded”

TableReadProvisionedThroughputExceeded

發生這種情況時

您的應用程式讀取使用率超出為資料表設定的佈建讀取容量單位 (RCU)。可透過 the section called “常見診斷與監控” 監控 CloudWatch 指標，以分析限流事件。

解決方法

請考慮以下策略以解決讀取容量限流問題：

- 切換至隨需容量模式：若因流量尖峰而頻繁限流，請考慮將資料表切換為隨需模式。隨需模式可免除佈建管理，並會隨工作負載自動擴展。
- 若仍採用佈建模式且未啟用 Auto Scaling：
  - 請考慮提高資料表的讀取容量。
  - 在資料表上啟用 Auto Scaling 以管理讀取容量。
- 若已啟用 Auto Scaling (主控台建立的資料表預設啟用)：
  - 最佳化資料表讀取 Auto Scaling 參數。

TableWriteProvisionedThroughputExceeded

發生這種情況時

您的應用程式寫入耗用速率超過為資料表設定的佈建寫入容量單位 (WCU)。可透過 the section called "常見診斷與監控" 監控 CloudWatch 指標，以分析限流事件。

解決方法

可採用以下策略以解決寫入容量限流問題：

- 切換至隨需容量模式：若因流量尖峰而頻繁限流，請考慮將資料表切換為隨需模式。隨需模式可免除佈建管理，並會隨工作負載自動擴展。
- 若仍採用佈建模式且未啟用 Auto Scaling：
  - 請考慮提高資料表的寫入容量。
  - 在資料表上為寫入容量啟用 Auto Scaling。
- 若已啟用 Auto Scaling (主控台建立的資料表預設啟用)：
  - 最佳化資料表寫入 Auto Scaling 的參數。

IndexReadProvisionedThroughputExceeded

發生這種情況時

全域次要索引 (GSI) 的讀取耗用量超出該 GSI 的佈建讀取容量單位 (RCU)。可透過 the section called "常見診斷與監控" 監控 CloudWatch 指標，以分析限流事件。

解決方法

請考慮以下策略以解決 GSI 讀取容量限流問題：

- 切換為隨需容量模式：若因流量激增而頻繁發生限流，請考慮將基礎資料表切換為隨需容量模式。隨需模式可免除佈建管理，並會隨工作負載自動擴展。
- 若仍採用佈建模式且未啟用 Auto Scaling：
  - 請考慮提高 GSI 的讀取容量。
  - 在 GSI 上啟用讀取容量的 Auto Scaling。
- 若已啟用 Auto Scaling (主控台建立的資料表預設啟用)：
  - 最佳化 GSI 的讀取 Auto Scaling 參數設定。

IndexWriteProvisionedThroughputExceeded

發生這種情況時

基礎資料表中項目的更新會觸發對 GSI 的寫入，超出該 GSI 的佈建寫入容量。這會使基礎資料表寫入出現背壓限流。可透過 the section called “常見診斷與監控” 監控 CloudWatch 指標，以分析限流事件。

解決方法

請考慮以下策略以解決 GSI 寫入容量限流問題：

- 切換為隨需容量模式：若因流量激增而頻繁發生限流，請考慮將基礎資料表切換為隨需容量模式。隨需模式可免除佈建管理，並會隨工作負載自動擴展。
- 若仍採用佈建模式且未啟用 Auto Scaling：
  - 請考慮提高 GSI 的寫入容量。
  - 在 GSI 上啟用寫入容量的 Auto Scaling。
- 若已啟用 Auto Scaling (主控台建立的資料表預設啟用)：
  - 最佳化 GSI 的寫入 Auto Scaling 參數設定。

常見診斷與監控

在疑難排解輸送量錯誤時，數個 CloudWatch 指標可協助識別根本原因。

主要 CloudWatch 指標

監控以下關鍵指標，以診斷佈建容量限流問題：

- 限流事件：`ReadProvisionedThroughputThrottleEvents` 和 `WriteProvisionedThroughputThrottleEvents` 會追蹤因該原因被限流的請求時間；`ReadThrottleEvents` 和 `WriteThrottleEvents` 則會追蹤任何讀寫請求超出佈建容量的情況。
- 容量耗用情形：`ConsumedReadCapacityUnits` 和 `ConsumedWriteCapacityUnits` 顯示實際使用情況。
- 佈建容量：`ProvisionedReadCapacityUnits` 和 `ProvisionedWriteCapacityUnits` 顯示所設定的容量上限。

解決步驟

提高資料表輸送容量

當未啟用 Auto Scaling 且需要立即提升容量時，請依此程序操作。

1. 使用 DynamoDB 主控台或 SDK AWS CLI更新資料表的佈建容量：
   - 針對讀取容量：提高 `ReadCapacityUnits` 參數，該參數指定在 DynamoDB 進行限流前，每秒可執行的高度一致性讀取上限。
   - 針對寫入容量：提高 `WriteCapacityUnits` 參數，該參數指定在 DynamoDB 進行限流前，每秒可執行的最大寫入次數。
2. 請確認新容量設定未超出每個資料表的輸送量配額，且整體帳戶用量仍低於所在區域的每帳戶輸送量配額。若接近這些限制，建議改用隨需容量模式。

設定資料表 Auto Scaling，以調整資料表或 GSI 的讀寫容量

設定 DynamoDB Auto Scaling，依據流量模式自動調整讀寫容量。可針對資料表與 GSIs 分別設定 Auto Scaling，並獨立控制讀寫容量單位。

1. 在資料表或 GSI 上啟用讀取、寫入或雙向容量的 Auto Scaling。
2. 設定目標使用率百分比，並預留空間以應對流量高峰。

> **Note**
>
> 較低的目標使用率將提升成本並增加擴展頻率。若目標值低於 40%，可能造成過度佈建。持續監控用量與成本，以維持效能與效率的平衡。

3. 設定容量上下限：
   - 最小 RCUs/WCUs：確保在低流量期間維持足夠容量。
   - 最大 RCUs/WCUs：支援尖峰流量需求，並防止擴展失控。

如需有關設定與管理 DynamoDB Auto Scaling 的指引，請參閱使用 DynamoDB Auto Scaling 自動管理輸送容量。

> **Note**
>
> Auto Scaling 通常需要數分鐘才能對流量變化做出回應。當流量突增時，資料表的高載容量可在 Auto Scaling 調整期間提供即時保護。設定具足夠餘裕的目標使用率，以確保有時間進行擴展，並保留高載容量以應對突發需求。

最佳化資料表或索引的讀寫 Auto Scaling 設定

當 Auto Scaling 已啟用但仍發生限流時，請依此程序操作。可分別為資料表與全域次要索引 (GSIs) 調整 Auto Scaling，並獨立控制讀寫容量單位。

- 調整目標使用率：可考慮降低資料表或 GSIs 的目標使用率，以在限流發生前提前觸發擴展。進行調整後，請持續監控流量變化。如需了解容量使用與成本影響的更多資訊，請參閱 the section called "設定資料表 Auto Scaling，以調整資料表或 GSI 的讀寫容量"。
- 檢視容量上下限：確保最小與最大容量設定符合實際工作負載模式。

切換至隨需容量模式

如需了解切換容量模式的一般資訊，請參閱 the section called "切換容量模式"。請參閱 Service Quotas，以了解切換模式時的特定限制。

提高 GSI 輸送容量

當 GSI 未啟用 Auto Scaling 或需立即提升容量時，請依此程序操作。

1. 使用 DynamoDB 主控台或 SDK AWS CLI更新 GSI 的佈建容量：
   - 針對讀取容量：提高特定 GSI 的 `ReadCapacityUnits` 參數，該參數指定在 DynamoDB 進行限流前，GSI 每秒可執行的最大讀取次數。請注意，GSI 僅支援最終一致讀取。
   - 針對寫入容量：提高特定 GSI 的 `WriteCapacityUnits` 參數，該參數指定在 DynamoDB 進行限流前，GSI 每秒可執行的最大寫入次數。
2. 確保 GSI 的佈建輸送容量維持在每帳戶與每資料表的輸送量配額範圍內。

其他資源

- 如需詳細資訊，說明如何在 DynamoDB 佈建容量資料表中處理流量尖峰，包括從利用 Auto Scaling 與高載容量到策略性限流管理等多種方法，請參閱使用 Amazon DynamoDB 佈建容量處理流量尖峰。
- 如需了解如何使用 Cron 表達式排程擴展政策，請參閱透過排程 DynamoDB 佈建容量來最佳化成本。
- 如需實作指南，說明如何在佈建容量模式中監控與分析 DynamoDB 資料表的輸送量使用模式，請參閱如何在佈建模式中評估 Amazon DynamoDB 資料表的輸送量使用率。

3- 超過帳戶限制

隨需資料表無需管理佈建容量層級，但 DynamoDB 仍會強制執行帳戶層級輸送量限制，以防止執行失控並確保所有客戶的資源公平使用。這些每資料表帳戶限制作為可調整的保護機制，針對每個帳戶與區域組合進行設定。當您的讀取或寫入耗用率超出這些限制時，DynamoDB 會在限流例外狀況中回傳 `AccountLimitExceeded` 限流原因類型。若資料表未設定自訂最大輸送量，則會自動套用預設的每資料表帳戶限制。您可選擇設定最大輸送量以提升成本控制與可預測性；若應用程式需求超出預設限制，則可透過 the section called “配額” 主控台請求提高配額。

超出帳戶限制的緩解措施

本節說明帳戶限制導致限流的解析與解決指引。使用本指南前，請先從應用程式的例外狀況處理中確認具體的限流原因，並確定受影響資源的 Amazon Resource Name (ARN)。如需了解如何擷取限流原因與識別受限流的資源，請參閱 the section called “診斷架構”。

在分析特定限流情境前，請先判斷是否確實需要採取行動：

- 評估效能影響：檢查應用程式在發生限流時是否仍符合效能需求。許多應用程式即使在達到或接近帳戶限制時仍能正常運作，特別是在大量操作或資料移轉期間。
- 檢視限流模式：若限流為間歇性，且應用程式能有效處理重試，則目前限制可能已足以支撐工作負載。

若應用程式即使偶爾達到帳戶限制仍能穩定運作，您可選擇持續監控而非立即進行變更。

若您判斷限流導致效能或可靠性問題，請從下列限流原因中選擇，以查看建議的緩解方案：

- the section called “TableReadAccountLimitExceeded”
- the section called “TableWriteAccountLimitExceeded”
- the section called “IndexReadAccountLimitExceeded”
- the section called “IndexWriteAccountLimitExceeded”

TableReadAccountLimitExceeded

發生這種情況時

您的資料表讀取耗用情形已超出您所在區域的帳戶層級每資料表讀取輸送量配額。可透過 the section called “常見診斷與監控” 監控 CloudWatch 指標，以分析限流事件。

解決方法

請依下列步驟解決此限流問題：

- 請求配額增加：

  請求提高每資料表讀取輸送量限制 (Quota 程式碼 L-CF0CBE56)。如需提交請求的詳細步驟，請參閱請求每資料表配額增加。

TableWriteAccountLimitExceeded

發生這種情況時

您的資料表寫入耗用情形已超出您所在區域的帳戶層級每資料表寫入輸送量配額。可透過 the section called “常見診斷與監控” 監控 CloudWatch 指標，以分析限流事件。

解決方法

請依下列步驟解決此限流問題：

- 請求配額增加：請求提高每資料表寫入輸送量限制 (Quota 程式碼 L-AB614373)。如需提交請求的詳細步驟，請參閱請求每資料表配額增加。

IndexReadAccountLimitExceeded

發生這種情況時

針對全域次要索引 (GSI) 的讀取操作耗用的輸送量，超出您帳戶在目前 AWS 區域允許的每資料表讀取配額。帳戶層級的每資料表讀取輸送量配額會整體套用至該資料表及其所有 GSI。可透過 the section called “常見診斷與監控” 監控 CloudWatch 指標，以分析限流事件。

解決方法

請依帳戶容量分佈選擇合適的解決方法：

- 請求配額增加：請求提高每資料表讀取輸送量限制 (Quota 程式碼 L-CF0CBE56)。如需提交請求的詳細步驟，請參閱請求每資料表配額增加。
- 最佳化 GSI 耗用情形：檢視 GSI 設計與查詢模式，以減少不必要的讀取容量耗用。

IndexWriteAccountLimitExceeded

發生這種情況時

對基底資料表的寫入操作會產生對應的 GSI 更新，這些 GSI 總共超過您 AWS 區域的帳戶層級每個資料表寫入輸送量配額。每次將包含 GSI 索引屬性的項目寫入基礎資料表時，系統都會觸發對應的 GSI 寫入操作。這些合併的寫入操作會計入您的每資料表寫入輸送量配額。您可以在 the section called “常見診斷與監控” 中監控 CloudWatch 指標，以分析這些限流事件的模式與時間，並識別造成過度 GSI 寫入的操作。

解決方法

請依帳戶容量分佈選擇合適的解決方法：

- 請求配額增加：請求提高每個資料表的寫入輸送量上限 (Quota 程式碼 L-AB614373)，以因應基礎資料表操作所產生的更高 GSI 寫入流量。每個資料表的寫入輸送量配額會套用至整個資料表，包括其所有的 GSI。如需提交請求的詳細步驟，請參閱請求每資料表配額增加。
- 最佳化 GSI 投影：檢視 GSI 投影與設計，以降低 GSI 寫入量。

常見診斷與監控

疑難排解帳戶限制超限限流事件時，多項 CloudWatch 指標可協助判斷您是達到每資料表限制或整體帳戶限制，並協助分析容量分佈模式。

主要 CloudWatch 指標

監控以下關鍵指標，以診斷帳戶限制導致的限流：

- 帳戶限制限流事件：`ReadAccountLimitThrottleEvents` 與 `WriteAccountLimitThrottleEvents` 會追蹤因帳戶層級限制而遭限流的請求；`ReadThrottleEvents` 與 `WriteThrottleEvents` 則會追蹤任何讀取或寫入請求何時超出佈建容量。
- 依資源的容量耗用情形：每個資料表與 GSI 的 `ConsumedReadCapacityUnits` 和 `ConsumedWriteCapacityUnits` 指標會顯示各自的資源使用情況。
- 全帳戶耗用：使用 CloudWatch 的指標數學運算式，將所有資料表與 GSI 的耗用容量加總，以監控帳戶總體用量。

解決步驟

請求每個資料表配額增加

如果您的應用程式必須在超出目前每個資料表輸送量上限的情況下運作，請依下列程序提交配額增加申請。您 AWS 帳戶中的每個 DynamoDB 資料表 （連同其所有相關聯的 GSIs) 都受到特定區域內這些輸

送量配額的約束。這些配額代表每個資料表及其 GSI 可共同使用的最大讀取或寫入容量，並會獨立套用於各資料表，而非以整個帳戶所有資料表的總量計算。

您也可以選擇為每個資料表或每個 GSI 設定最大隨需輸送量，以限制其最高可用容量。

1. 識別需要提高的特定配額：
   - 每個資料表讀取輸送量限制 (配額代碼 L-CF0CBE56)：每個資料表的預設值為 40,000 RCUs
   - 每個資料表寫入輸送量限制 (配額代碼 L-AB614373)：每個資料表的預設值為 40,000 WCUs
2. 使用 AWS Service Quotas 主控台提出配額提高請求：
   - 在 Service Quotas 主控台中導覽至 DynamoDB 服務
   - 透過配額代碼尋找對應的配額
   - 根據預期尖峰使用量提出配額提高請求
3. 說明提高配額的理由，包括：
   - 目前的使用模式與尖峰流量需求
   - 提高容量的業務理由
   - 提高容量所需的時間規劃

> **Note**
>
> 配額提高的處理通常需要 24 至 48 小時。請相應規劃您的請求，並在等待核准期間考慮暫時的應變策略。

最佳化 GSI 投影與設計

透過最佳化全域次要索引 (GSI) 的投影與設計，降低容量消耗並提升效能。

選擇性投影策略

若查詢僅需存取少數屬性，則在基礎資料表項目變更時，只投影這些屬性能減少寫入 GSI 的資料量。如需投影類型的詳細資訊，請參閱全域次要索引投影。

1. 分析查詢模式：檢視應用程式的查詢模式，以確認哪些屬性實際透過 GSI 存取。
2. 使用選擇性投影：僅投影查詢實際需要的屬性，以減少寫入量。

3. 考慮使用 KEYS_ONLY：若查詢僅需鍵屬性，可使用 KEYS_ONLY 投影以將寫入量降至最低。
4. 平衡讀寫取捨：減少投影屬性可降低寫入容量耗用，但可能需要額外的基礎資料表讀取。

稀疏 GSI 實作

稀疏 GSI 僅包含具有索引屬性的項目，而非基礎資料表的所有項目。此設計可降低分割區密度，並在頻繁查詢特定資料子集時提升效能。

1. 設計僅包含特定屬性值項目的 GSI。
2. 僅在需建立索引的項目上設定 GSI 分割區索引鍵屬性，以實作條件式索引。
3. 在稀疏 GSI 中使用複合索引鍵 (例如 status#timestamp)，以進一步分散索引項目子集的流量。

如需實作這些策略的詳細資訊，請參閱在 DynamoDB 中使用次要索引的最佳實務。

4 - 超過隨需輸送量上限

當您設定隨需資料表或 GSI 時，可選擇在資料表或索引層級設定最大輸送量上限 (`MaxReadRequestUnits` 和 `MaxWriteRequestUnits`)，以防止成本失控，或避免下游系統因流量過大而超載。如需最大輸送量上限的詳細資訊，請參閱 the section called “DynamoDB 隨需資料表的最大輸送量”。

當讀取或寫入耗用量超出這些自訂限制時，超出部分的其他請求會收到即時限流回應。DynamoDB 會傳回含有 `MaxOnDemandThroughputExceeded` 限流原因類型的例外狀況，用以指出已達輸送量界限的資源。

超出限流的隨需輸送量上限

本節提供超出限流案例的隨需最大輸送量的解析指引。使用本指南前，請先從應用程式的例外狀況處理中確認具體的限流原因，並確定受影響資源的 Amazon Resource Name (ARN)。如需了解如何擷取限流原因並識別遭限流的資源，請參閱 the section called “診斷架構”。

在深入分析特定限流情境前，請先評估是否真的需要採取行動：

- 評估您的最大輸送量上限設定：這些限制是為了控制成本或保護下游系統而刻意設定的。如果您收到 `MaxOnDemandThroughputExceeded` 限流事件，代表您的限制正在依設計運作。評估提高這些限制是否符合原先的成本控制或系統保護目標。
- 評估應用程式影響：確認限流是否確實影響您的應用程式或使用者。若您的應用程式能有效處理重試，且在偶發限流下仍符合效能需求，維持現有限制可能是最佳選項。

- 檢視流量模式：分析限流是否屬於預期的流量行為或異常高峰。對於可預測且持續超出限制的週期性流量模式，建議調整最大輸送量上限設定。若為暫時性高峰，實作更佳的請求分散策略可能比提升限制更為合宜。

若評估後確認需調整最大輸送量上限設定，請參閱以下特定限流情境，以取得對應的修正方案：

- the section called “TableReadMaxOnDemandThroughputExceeded”
- the section called “TableWriteMaxOnDemandThroughputExceeded”
- the section called “IndexReadMaxOnDemandThroughputExceeded”
- the section called “IndexWriteMaxOnDemandThroughputExceeded”

TableReadMaxOnDemandThroughputExceeded

發生這種情況時

您的隨需模式資料表已超出其設定的最大讀取輸送容量上限。可透過 the section called “常見診斷與監控” 監控 CloudWatch 指標，以分析限流事件。

修正方案

可參考以下步驟處理限流事件：

- 提高最大輸送量上限：使用 DynamoDB 主控台、AWS CLI 或 DynamoDB `UpdateTable` API 來提高受影響資料表的 `MaxReadRequestUnits` 值，再進行監控與調整。這可讓資料表在發生限流前處理更高的讀取輸送量。
- 移除上限：將 `MaxReadRequestUnits` 設定為 `-1`，以移除上限，允許依需求擴展至帳戶層級輸送量配額。這將移除自訂限制，但仍保留 AWS的帳戶層級防護機制。不過，移除此限制後請密切監控支出，因為資料表在達到帳戶層級配額前可能耗用顯著更多的容量。

TableWriteMaxOnDemandThroughputExceeded

發生這種情況時

您的隨需模式資料表已超出其設定的最大寫入輸送容量上限。可透過 the section called “常見診斷與監控” 監控 CloudWatch 指標，以分析限流事件。

修正方案

可參考以下步驟處理限流事件：

- 提高最大輸送量上限：使用 DynamoDB 主控台、AWS CLI 或 DynamoDB `UpdateTable` API 來提高受影響資料表的 `MaxWriteRequestUnits` 值，再進行監控與調整。
- 移除上限：將 `MaxWriteRequestUnits` 設定為 `-1`，以移除上限，允許依需求擴展至帳戶層級輸送量配額。這將移除自訂限制，但仍保留 AWS的帳戶層級防護機制。不過，移除此限制後請密切監控支出，因為資料表在達到帳戶層級配額前可能耗用顯著更多的容量。

IndexReadMaxOnDemandThroughputExceeded

發生這種情況時

在隨需模式中，對 GSI 的讀取請求已超出其設定的最大讀取輸送容量上限。可透過 the section called "常見診斷與監控" 監控 CloudWatch 指標，以分析限流事件。

修正方案

可參考以下步驟處理限流事件：

- 提高 GSI 最大輸送量上限：使用 DynamoDB 主控台、AWS CLI 或 DynamoDB `UpdateTable` API 來提高受影響 GSI 的 `MaxReadRequestUnits` 值，然後進行監控與調整。
- 移除 GSI 上限：將 GSI 的 `MaxReadRequestUnits` 設定為 `-1`，以移除上限，允許依需求擴展至帳戶層級輸送量配額。這會移除您的自訂限制，但仍維持 AWS帳戶層級的保護。不過，移除此限制後請密切監控支出。

IndexWriteMaxOnDemandThroughputExceeded

發生這種情況時

當基礎資料表項目的更新在隨需模式下觸發對 GSI 的寫入，超出其設定的最大寫入輸送容量上限時，會導致背壓限流。可透過 the section called "常見診斷與監控" 監控 CloudWatch 指標，以分析限流事件。

修正方案

可參考以下步驟處理限流事件：

- 提高 GSI 最大輸送量上限：使用 DynamoDB 主控台、AWS CLI 或 DynamoDB `UpdateTable` API 來提高受影響 GSI 的 `MaxWriteRequestUnits` 值，然後進行監控與調整。

- 移除 GSI 上限：將 GSI 的 `MaxWriteRequestUnits` 設定為 `-1`，以移除上限，允許依需求擴展至帳戶層級輸送量配額。這會移除您的自訂限制，但仍維持 AWS帳戶層級的保護。不過，移除此限制後請密切監控支出。

常見診斷與監控

當對隨需輸送量超過限流事件進行疑難排解時，數個 CloudWatch 指標有助於識別根本原因和擴展模式。

主要 CloudWatch 指標

監控這些關鍵指標，以診斷超出限流的隨需最大輸送量：

- 最大輸送量限流事件：`ReadMaxOnDemandThroughputThrottleEvents` 與 `WriteMaxOnDemandThroughputThrottleEvents` 用於追蹤因超出上限而被限流的請求；`ReadThrottleEvents` 與 `WriteThrottleEvents` 則追蹤任何超出佈建容量的讀寫請求。
- 資料表或全域次要索引目前設定的最大輸送量上限：`OnDemandMaxReadRequestUnits` 與 `OnDemandMaxWriteRequestUnits` 顯示現行的最大容量限制。
- 實際容量耗用情形：`ConsumedReadCapacityUnits` 與 `ConsumedWriteCapacityUnits` 顯示實際使用模式。

分析步驟

請依照下列步驟確認隨需輸送量超過診斷上限：

1. 比較已耗用容量與最大容量上限，檢查耗用量是否持續接近或超出上限。
2. 檢視限流事件的頻率與發生時間，以辨識模式。檢查是否有與限流事件同時出現的耗用容量急增。
3. 使用 CloudWatch Contributor Insights 以識別耗用容量最高的項目或分割區索引鍵。

認識 DynamoDB 中全域次要索引 (GSI) 的寫入限流與背壓現象

GSI 背壓限流是 DynamoDB 中最複雜的限流情境之一，因其在寫入操作與限流之間形成間接關聯：應用程式雖寫入基礎資料表，卻因一或多個索引的容量受限而遭遇限流。

認識 GSI 背壓限流

當您對 DynamoDB 資料表進行寫入時，該表的任何全域次要索引 (GSI) 都會依「最終一致性模式」進行非同步更新。若 GSI 的容量不足以處理這些更新，DynamoDB 會對基礎資料表的寫入進行限流，以

維持資料一致性。這種情況稱為 GSI 背壓。如需了解 GSI 的運作方式，請參閱 DynamoDB 全域次要索引。

不同於直接資料表限流 (被存取的資源同時也是導致限流的資源)，GSI 背壓會在基礎資料表與索引間建立相依關係。即使基礎資料表容量充足，若任一關聯 GSI 無法處理更新量，寫入仍會遭限流。此關係尤其重要，因為分割區層級限制會獨立套用於基礎資料表與每個 GSI，兩者皆具有各自的分割區結構與對應的輸送量上限。

GSI 的分割設計是依其分割區索引鍵決定，通常與基礎資料表的分割區索引鍵不同。即使基礎資料表的存取在各分割區間分佈均勻，GSI 更新仍可能集中於特定分割區，導致 GSI 產生熱點。如需了解資料表與 GSI 的分割區索引鍵設計最佳實務，請參閱 DynamoDB 分割區索引鍵設計。

例如，若基礎資料表使用 `customerId` 作為分割區索引鍵 (平均分佈)，而 GSI 使用 `status` 作為分割區索引鍵 (可能值有限，如 "active"、"pending"、"closed")，則即使基礎資料表存取均衡，針對熱門狀態值項目的更新仍可能造成 GSI 熱分割區。這種情況特別具挑戰性，因為即使基礎資料表與 GSI 均具足夠整體容量，且基礎資料表存取模式分佈良好，應用程式仍可能因 GSI 熱分割區而遭遇限流。

即使限流例外狀況指向 GSI (透過 `ResourceArn`)，實際受到限流的操作仍是對基礎資料表的寫入。這可能令人困惑，因為應用程式實際寫入的是基礎資料表，卻收到與 GSI 有關的例外狀況。

GSI 限流類型

GSI 背壓限流會依具體容量限制，呈現不同類型的例外狀況：

- 超過 GSI 佈建容量：當 GSI 沒有足夠的寫入容量單位來處理基礎資料表的更新作業時發生。這會產生包含原因 `IndexWriteProvisionedThroughputExceeded` 的 `ProvisionedThroughputExceededException`，並直接 `ResourceArn` 指向發生容量限制的特定 GSI。
- 超過 GSI 隨需輸送量上限：當 GSI 寫入操作超過對隨需資料表設定的最大限制時發生。這會產生包含原因 `IndexWriteMaxOnDemandThroughputExceeded` 的 `ThrottlingException`，用以識別具設定輸送量限制的特定 GSI。
- 超過 GSI 分割區限制：即使整體 GSI 容量看似充足，若個別 GSI 分割區的輸送量超出上限 (熱分割區)，仍會發生此情況。這會產生包含原因 `IndexWriteKeyRangeThroughputExceeded` 的 `ThrottlingException`，指出於 `ResourceArn` 中識別的特定 GSI 出現熱分割區問題。這點尤其重要，因為 GSI 的分割區分佈可能與基礎資料表顯著不同，即使基礎資料表的存取均勻分佈，仍可能在 GSI 中產生熱點。
- 超過 GSI 帳戶限制：當對特定 GSI 的寫入操作超過帳戶層級設定的每資料表 （或該資料表內的任何個別 GSI) 區域輸送量界限時觸發。DynamoDB 會傳回包含原因 `IndexWriteAccountLimitExceeded` 的 `ThrottlingException`，指出用量超出帳戶限制的特

定 GSI。此限流會針對每個超出限制的 GSI 獨立發生。如需各資料表、各帳戶、區域及服務配額的詳細資訊，請參閱 the section called “配額”。

CloudWatch 限流度量指標

本頁提供 CloudWatch 度量指標的完整指南，專為協助您識別、診斷與解決 DynamoDB 資料表及索引中的限流問題而設計。

一般限流度量指標

- `ThrottledRequests`
  - 當請求中的任一事件被限流時，無論該請求包含多少事件，指標都會增加 1。例如，在具有全域次要索引 (GSI) 的資料表中更新項目時，會觸發多個事件：對基礎資料表的寫入，以及對各索引的寫入。若這些事件中的任一被限流，`ThrottledRequests` 度量指標僅會增加一次。

    此行為對理解與診斷 DynamoDB 效能問題極為重要，因其可能掩蓋實際限流程度。若需更完整的洞察，可將 `ThrottledRequests` 度量指標與事件層級指標 (如 `ReadThrottleEvents`、`WriteThrottleEvents`) 及特定目標指標 (如 `ReadKeyRangeThroughputThrottleEvents`) 進行比較。本頁提供所有依原因分類的度量指標完整清單。每個度量指標對應於限流例外中所記錄的特定限流原因。如需在限流事件期間取得與解析這些原因的說明，請參閱 the section called “限流診斷” 章節，其中提供識別並解決限流根本原因的步驟。
- `ReadThrottleEvents`
  - 注意超出資料表或 GSI 佈建 RCU 的請求。
- `WriteThrottleEvents`
  - 注意超出資料表或 GSI 佈建 WCU 的請求。

依原因分類的詳細限流度量指標

隨需輸送量限流

- `ReadMaxOnDemandThroughputThrottleEvents`
  - 因隨需模式達最大輸送量而被限流的讀取請求數。
- `WriteMaxOnDemandThroughputThrottleEvents`
  - 因隨需模式達最大輸送量而被限流的寫入請求數。

帳戶層級限流

- ReadAccountLimitThrottleEvents
  - 因帳戶限制而被限流的讀取請求數。
- WriteAccountLimitThrottleEvents
  - 因帳戶限制而被限流的寫入請求數。

分割區層級限流

- ReadKeyRangeThroughputThrottleEvents
  - 因分割區限制而被限流的讀取請求數。
- WriteKeyRangeThroughputThrottleEvents
  - 因分割區限制而被限流的寫入請求數。

容量分析度量指標

- OnlineIndexConsumedWriteCapacity
  - 當您將新 GSI 新增至現有資料表時，DynamoDB 會執行回填程序，將基礎資料表的資料複製到新索引中。此程序會耗用寫入容量單位。OnlineIndexConsumedWriteCapacity 度量指標會追蹤此特定耗用量。

    此耗用獨立於 ConsumedWriteCapacityUnits 所追蹤的常規寫入操作，並屬額外耗用。GSI 的常規 ConsumedWriteCapacityUnits 度量指標不包含初始建立索引期間所耗用的寫入容量。
- ProvisionedReadCapacityUnits 和 ProvisionedWriteCapacityUnits
  - 檢視在指定時段，針對資料表或指定的全域次要索引耗用了多少已佈建的讀取或寫入容量單位。
  - 請注意，TableName 維度預設僅針對資料表傳回 ProvisionedReadCapacityUnits。若要檢視全域次要索引的已佈建讀取或寫入容量單位，必須同時指定 TableName 與 GlobalSecondaryIndexName。
- ConsumedReadCapacityUnits 和 ConsumedWriteCapacityUnits
  - 檢視指定時間範圍內耗用的讀取與寫入容量單位。ConsumedWriteCapacityUnits 不包含初始建立索引期間的寫入容量耗用。

如需 DynamoDB CloudWatch 度量指標的詳細資訊，請參閱 DynamoDB 指標和維度。

DynamoDB 附錄

主題

- 使用 DynamoDB 對 SSL/TLS 連線建立問題進行疑難排解
- 在 DynamoDB 中使用的資料表和資料範例
- 在 DynamoDB 中建立範例資料表和上傳資料
- 使用 的 DynamoDB 範例應用程式 適用於 Python (Boto) 的 AWS SDK：Tic-tac-toe
- DynamoDB 中的保留字
- AWS 適用於 Java 的 SDK 1.x 範例
- AWS 適用於 Go 的 SDK 1.x 範例
- AWS 適用於 Node.js 的 SDK 2.x 範例

使用 DynamoDB 對 SSL/TLS 連線建立問題進行疑難排解

Amazon DynamoDB 正在移動端點，以便保護由 Amazon Trust Services (ATS) 憑證認證機構而非第三方憑證認證機構所簽署的憑證。2017 年 12 月，我們推出 EU-WEST-3 (巴黎) 區域，附有由 Amazon Trust Services 發行的安全憑證。所有於 2017 年 12 月之後推出的新區域都具有附帶 Amazon Trust Services 發行的憑證的端點。本指南說明如何對 SSL/TLS 連線建立問題進行驗證和疑難排解。

測試您的應用程式或服務

AWS SDKs和命令列界面 (CLIs) 支援 Amazon Trust Services 憑證授權單位。如果您使用的是 2013 年 10 月 29 日之前發行的適用於 Python 或 CLI 的 AWS SDK 版本，則必須升級。.NET、Java、PHP、Go、JavaScript、C++ 軟體開發套件以及 CLI 無任何憑證套件，其憑證來自基礎作業系統。自 2015 年 6 月 10 日起，Ruby 軟體開發套件已包含至少一個必要 CA。在該日期之前，Ruby 第 2 版軟體開發套件並無憑證套件。如果您使用不支援、自訂或修改版本的 AWS SDK，或者如果您使用自訂信任存放區，您可能沒有 Amazon Trust Services Certificate Authority 所需的支援。

若要驗證能否存取 DynamoDB 端點，您需要進行測試來存取 EU-WEST-3 區域中的 DynamoDB API 或 DynamoDB Streams API，並驗證 TLS 交握是否成功。您需要在這種測試中存取的特定端點是：

- DynamoDB：https://dynamodb.eu-west-3.amazonaws.com
- DynamoDB Streams：https://streams.dynamodb.eu-west-3.amazonaws.com

如果應用程式不支援 Amazon Trust Services 憑證認證機構 (CA)，則會看到下列其中一項失敗：

- SSL/TLS 溝通錯誤
- 在軟體收到錯誤指出 SSL/TLS 溝通失敗之前，會有長時間延遲。延遲時間依用戶端的重試策略和逾時組態而定。

測試您的用戶端瀏覽器

若要確認瀏覽器是否可連線到 Amazon DynamoDB，請開啟以下網址：https://dynamodb.eu-west-3.amazonaws.com。如果測試成功，則會看到如下的訊息：

```
healthy: dynamodb.eu-west-3.amazonaws.com
```

如果測試不成功，則會顯示類似以下內容的錯誤：https://untrusted-root.badssl.com/。

更新您的軟體應用程式用戶端

如果應用程式尚未支援下列任何 CA，則在存取 DynamoDB 或 DynamoDB Streams API 端點時 (無論是透過瀏覽器還是編寫程式的方式)，均需更新用戶端機器上的信任 CA 清單：

- Amazon 根 CA 1
- Starfield Services 根憑證認證機構：G2
- Starfield 類別 2 憑證認證機構

如果用戶端已經信任上述三個 CA 之中的任一個，則這些用戶端會信任 DynamoDB 使用的憑證，不需要進行任何動作。不過，如果用戶端尚未信任上述任何 CA，則與 DynamoDB 或 DynamoDB Streams API 的 HTTPS 連線將會失敗。如需詳細資訊，請參閱此部落格文章：https://aws.amazon.com/blogs/security/how-to-prepare-for-aws-move-to-its-own-certificate-authority/。

更新您的用戶端瀏覽器

只要更新瀏覽器即可更新瀏覽器中的憑證套件。常用瀏覽器的說明可以在瀏覽器網站上找到：

- Chrome：https://support.google.com/chrome/answer/95414?hl=en
- 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/help/17295/windows-internet-explorer-which-version#ie=other

手動更新您的憑證套件

如果您無法存取 DynamoDB API 或 DynamoDB Streams API，則需要更新憑證套件。為此，您需要匯入至少一個必要 CA。您可以在此尋找這些 CA：https://www.amazontrust.com/repository/。

下列作業系統和程式設計語言支援 Amazon Trust Services 憑證認證機構：

- 具備自 2005 年 1 月起之更新的 Microsoft Windows 版本，Windows Vista、Windows 7、Windows Server 2008 和更新版本。
- MacOS X 10.4 搭配 Java for MacOS X 10.4 第 5 版、MacOS X 10.5 和更新版本。
- Red Hat Enterprise Linux 5 (2007 年 3 月)、Linux 6 和 Linux 7 以及 CentOS 5、CentOS 6、CentOS 7
- Ubuntu 8.10
- Debian 5.0
- Amazon Linux (所有版本)
- Java 1.4.2_12、Java 5 更新版本 2 及所有更新版本，包括 Java 6、Java 7、Java 8

如果您仍然無法連線，請參閱您的軟體文件、作業系統廠商，或聯絡 AWS Support https：//https://aws.amazon.com/support 以取得進一步協助。

在 DynamoDB 中使用的資料表和資料範例

所以此《Amazon DynamoDB 開發人員指南》使用範例表來說明 DynamoDB 的各個層面。

資料表名稱	主索引鍵
ProductCatalog	簡單主索引鍵： • `Id` (數字)
Forum	簡單主索引鍵： • `Name` (字串)
Thread	複合主鍵： • `ForumName` (字串)

資料表名稱	主索引鍵
	• Subject (字串)
Reply	複合主鍵： • Id (字串) • ReplyDateTime (字串)

此 Reply 資料表有一個名為 PostedBy-Message-Index 的全域次要索引。此索引會加速對 Reply 資料表的兩個非索引鍵屬性的查詢。

索引名稱	主索引鍵
PostedBy-Message-Index	複合主鍵： • PostedBy (字串) • Message (字串)

如需這些資料表的詳細資訊，請參閱 步驟 1：在 DynamoDB 建立資料表 和 步驟 2：將資料寫入 DynamoDB 資料表。

範例資料檔案

主題

- ProductCatalog 範例資料
- Forum 範例資料
- Thread 範例資料
- Reply 範例資料

下列各部分會顯示用於載入 ProductCatalog、Forum、Thread 和 Reply 的資料表。

每個資料檔案會包含多個 PutRequest 元素，每個元素則會包含一個項目。借助 AWS Command Line Interface(AWS CLI)，可將這些 PutRequest 元素用作 BatchWriteItem 操作的輸入。

ProductCatalog 範例資料

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "101"
                    },
                    "Title": {
                        "S": "Book 101 Title"
                    },
                    "ISBN": {
                        "S": "111-1111111111"
                    },
                    "Authors": {
                        "L": [
                            {
                                "S": "Author1"
                            }
                        ]
                    },
                    "Price": {
                        "N": "2"
                    },
                    "Dimensions": {
                        "S": "8.5 x 11.0 x 0.5"
                    },
                    "PageCount": {
                        "N": "500"
                    },
                    "InPublication": {
                        "BOOL": true
                    },
                    "ProductCategory": {
                        "S": "Book"
                    }
                }
            }
        },
        {
            "PutRequest": {
```

```
                "Item": {
                    "Id": {
                        "N": "102"
                    },
                    "Title": {
                        "S": "Book 102 Title"
                    },
                    "ISBN": {
                        "S": "222-2222222222"
                    },
                    "Authors": {
                        "L": [
                            {
                                "S": "Author1"
                            },
                            {
                                "S": "Author2"
                            }
                        ]
                    },
                    "Price": {
                        "N": "20"
                    },
                    "Dimensions": {
                        "S": "8.5 x 11.0 x 0.8"
                    },
                    "PageCount": {
                        "N": "600"
                    },
                    "InPublication": {
                        "BOOL": true
                    },
                    "ProductCategory": {
                        "S": "Book"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "103"
                    },
```

```
                    "Title": {
                        "S": "Book 103 Title"
                    },
                    "ISBN": {
                        "S": "333-3333333333"
                    },
                    "Authors": {
                        "L": [
                            {
                                "S": "Author1"
                            },
                            {
                                "S": "Author2"
                            }
                        ]
                    },
                    "Price": {
                        "N": "2000"
                    },
                    "Dimensions": {
                        "S": "8.5 x 11.0 x 1.5"
                    },
                    "PageCount": {
                        "N": "600"
                    },
                    "InPublication": {
                        "BOOL": false
                    },
                    "ProductCategory": {
                        "S": "Book"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "201"
                    },
                    "Title": {
                        "S": "18-Bike-201"
                    },
                    "Description": {
```

```
                    "S": "201 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Mountain A"
                },
                "Price": {
                    "N": "100"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Red"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "202"
                },
                "Title": {
                    "S": "21-Bike-202"
                },
                "Description": {
                    "S": "202 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Brand-Company A"
```

```
                    },
                    "Price": {
                        "N": "200"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Green"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "203"
                    },
                    "Title": {
                        "S": "19-Bike-203"
                    },
                    "Description": {
                        "S": "203 Description"
                    },
                    "BicycleType": {
                        "S": "Road"
                    },
                    "Brand": {
                        "S": "Brand-Company B"
                    },
                    "Price": {
                        "N": "300"
                    },
                    "Color": {
                        "L": [
                            {
```

```
                                "S": "Red"
                            },
                            {
                                "S": "Green"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "204"
                    },
                    "Title": {
                        "S": "18-Bike-204"
                    },
                    "Description": {
                        "S": "204 Description"
                    },
                    "BicycleType": {
                        "S": "Mountain"
                    },
                    "Brand": {
                        "S": "Brand-Company B"
                    },
                    "Price": {
                        "N": "400"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            }
                        ]
                    },
```

```
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "205"
                    },
                    "Title": {
                        "S": "18-Bike-204"
                    },
                    "Description": {
                        "S": "205 Description"
                    },
                    "BicycleType": {
                        "S": "Hybrid"
                    },
                    "Brand": {
                        "S": "Brand-Company C"
                    },
                    "Price": {
                        "N": "500"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        }
    ]
```

```
}
```

Forum 範例資料

```
{
    "Forum": [
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon DynamoDB"},
                    "Category": {"S":"Amazon Web Services"},
                    "Threads": {"N":"2"},
                    "Messages": {"N":"4"},
                    "Views": {"N":"1000"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon S3"},
                    "Category": {"S":"Amazon Web Services"}
                }
            }
        }
    ]
}
```

Thread 範例資料

```
{
    "Thread": [
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon DynamoDB"
                    },
                    "Subject": {
                        "S": "DynamoDB Thread 1"
                    },
                    "Message": {
                        "S": "DynamoDB thread 1 message"
```

```
                    },
                    "LastPostedBy": {
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-22T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "0"
                    },
                    "Replies": {
                        "N": "0"
                    },
                    "Answered": {
                        "N": "0"
                    },
                    "Tags": {
                        "L": [
                            {
                                "S": "index"
                            },
                            {
                                "S": "primarykey"
                            },
                            {
                                "S": "table"
                            }
                        ]
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon DynamoDB"
                    },
                    "Subject": {
                        "S": "DynamoDB Thread 2"
                    },
                    "Message": {
                        "S": "DynamoDB thread 2 message"
                    },
```

```
                    "LastPostedBy": {
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-15T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "3"
                    },
                    "Replies": {
                        "N": "0"
                    },
                    "Answered": {
                        "N": "0"
                    },
                    "Tags": {
                        "L": [
                            {
                                "S": "items"
                            },
                            {
                                "S": "attributes"
                            },
                            {
                                "S": "throughput"
                            }
                        ]
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon S3"
                    },
                    "Subject": {
                        "S": "S3 Thread 1"
                    },
                    "Message": {
                        "S": "S3 thread 1 message"
                    },
                    "LastPostedBy": {
```

```
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-29T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "0"
                    },
                    "Replies": {
                        "N": "0"
                    },
                    "Answered": {
                        "N": "0"
                    },
                    "Tags": {
                        "L": [
                            {
                                "S": "largeobjects"
                            },
                            {
                                "S": "multipart upload"
                            }
                        ]
                    }
                }
            }
        }
    ]
}
```

Reply 範例資料

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

在 DynamoDB 中建立範例資料表和上傳資料

本附錄提供以程式設計方式建立資料表和新增資料的程式碼。

主題

- 使用 建立範例資料表和上傳資料 適用於 Java 的 AWS SDK
- 使用 建立範例資料表和上傳資料 適用於 .NET 的 AWS SDK

使用 建立範例資料表和上傳資料 適用於 Java 的 AWS SDK

下列 Java 程式碼範例會建立資料表，並將資料上傳至資料表。如需使用 Eclipse 執行此程式碼的逐步說明，請參閱 Java 程式碼範例。

```
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());

        }
    }

}
```

使用 建立範例資料表和上傳資料 適用於 .NET 的 AWS SDK

下列 C# 程式碼範例會建立資料表，並將資料上傳至資料表。如需在 Visual Studio 中執行此程式碼的逐步說明，請參閱 .NET 程式碼範例。

```
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);
        }
    }
}
```

使用 的 DynamoDB 範例應用程式 適用於 Python (Boto) 的 AWS SDK：Tic-tac-toe

井字遊戲是建置在 Amazon DynamoDB 上的範例 Web 應用程式。應用程式使用 適用於 Python (Boto) 的 AWS SDK 進行必要的 DynamoDB 呼叫，以將遊戲資料存放在 DynamoDB 資料表中，並使用 Python Web 架構 Flask 來說明 DynamoDB end-to-end應用程式開發，包括如何建立資料的模型。它也示範在 DynamoDB 中為資料建立模型的最佳實務，包含您為遊戲應用程式建立的資料表、您定義的主索引鍵、根據您的查詢需求所需的其他索引，以及使用串連值屬性。

您在 Web 上玩井字遊戲應用程式的程序如下所示：

1. 您要登入應用程式首頁。
2. 接著邀請另一位使用者做為您的對手玩遊戲。

   在另一位使用者接受您的邀請前，遊戲狀態都會維持在 `PENDING`。對手接受邀請後，遊戲狀態會變更為 `IN_PROGRESS`。
3. 遊戲會在對手登入並接受邀請後開始。
4. 應用程式會將所有遊戲的移動和狀態資訊存放在 DynamoDB 資料表。
5. 遊戲會以獲勝或平手結束，這會將遊戲狀態設為 `FINISHED`。

我們會依步驟說明端對端應用程式的建置練習：

- 步驟 1：在本機上部署及測試：在本節中，您會在您的本機電腦上下載、部署及測試應用程式。您會在可下載版 DynamoDB 中建立必要的資料表。
- 步驟 2：檢查資料模型和實作詳細資訊：本節會先詳細說明資料模型，包含索引和使用串連值屬性。接著會說明應用程式運作的方式。
- 步驟 3：使用 DynamoDB 服務在生產環境中部署：本節重點為生產環境中的部署考量。在此步驟中，您會使用 Amazon DynamoDB 服務建立資料表，並使用 AWS Elastic Beanstalk部署應用程式。當您在生產環境中部署該應用程式時，您還要授予適當的許可，讓應用程式可存取 DynamoDB 資料表。本節說明會帶您演練端對端生產部署。
- 步驟 4：清除資源：本節重點說明此範例未涵蓋的部分。本節也提供移除您在上述步驟中建立 AWS 之資源的步驟，讓您避免產生任何費用。

步驟 1：在本機上部署及測試

主題

- 1.1：下載並安裝必要套件
- 1.2：測試遊戲應用程式

在此步驟中您會在您的本機電腦上下載、部署及測試井字遊戲應用程式。您並不會使用 Amazon DynamoDB Web 服務，而是將 DynamoDB 下載至您的電腦，並在該位置建立必要的資料表。

1.1：下載並安裝必要套件

若要在本機上測試此應用程式，您需要下列項目：

- Python
- Flask (Python 的微框架)
- 適用於 Python (Boto) 的 AWS SDK
- 在您電腦上執行的 DynamoDB
- Git

若要取得這些工具，請執行下列作業：

1. 安裝 Python。如需逐步說明，請參閱 Download Python。

   井字遊戲應用程式已使用 Python 2.7 版測試。
2. 適用於 Python (Boto) 的 AWS SDK 使用 Python 套件安裝程式 (PIP) 安裝 Flask 和 ：
   - 安裝 PIP。

     如需說明，請參閱 Install PIP。在安裝頁面上，選擇 get-pip.py 連結，然後儲存檔案。接著以管理員身分開啟命令終端機，在命令提示中輸入如下內容。

     ```
     python.exe get-pip.py
     ```

     在 Linux 上，您不需要指定 `.exe` 副檔名。只要指定 `python get-pip.py` 就可以了。
   - 利用 PIP 使用下列程式碼安裝 Flask 和 Boto 套件。

     ```
     pip install Flask
     ```

```
pip install boto
pip install configparser
```

3. 將 DynamoDB 下載到您的電腦。如需執行方式的說明，請參閱 設定 DynamoDB 本機版 (可下載版本)。
4. 下載井字遊戲應用程式：
   a. 安裝 Git。如需說明，請參閱 git Downloads。
   b. 執行下列程式碼以下載應用程式。

```
git clone https://github.com/awslabs/dynamodb-tictactoe-example-app.git
```

1.2：測試遊戲應用程式

若要測試井字遊戲應用程式，您需要在您的電腦本機上執行 DynamoDB。

執行井字遊戲應用程式

1. 啟動 DynamoDB。
2. 啟動井字遊戲應用程式的 Web 伺服器。

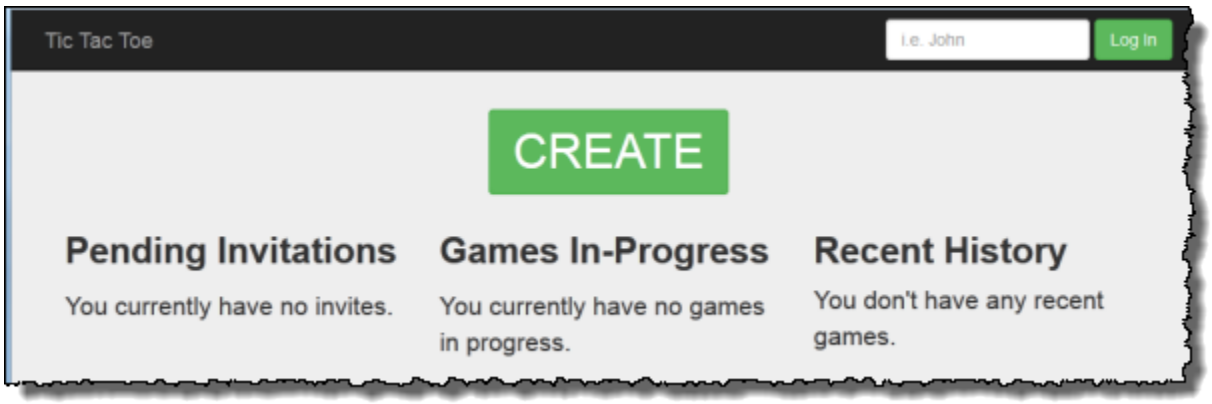
   若要執行此作業，請開啟命令終端機，導覽至您下載井字遊戲應用程式的所在資料夾，並使用下列程式碼在本機上執行應用程式。

```
python.exe application.py --mode local --serverPort 5000 --port 8000
```

   在 Linux 上，您不需要指定 `.exe` 副檔名。
3. 開啟您的 Web 瀏覽器，然後輸入如下內容。

```
http://localhost:5000/
```

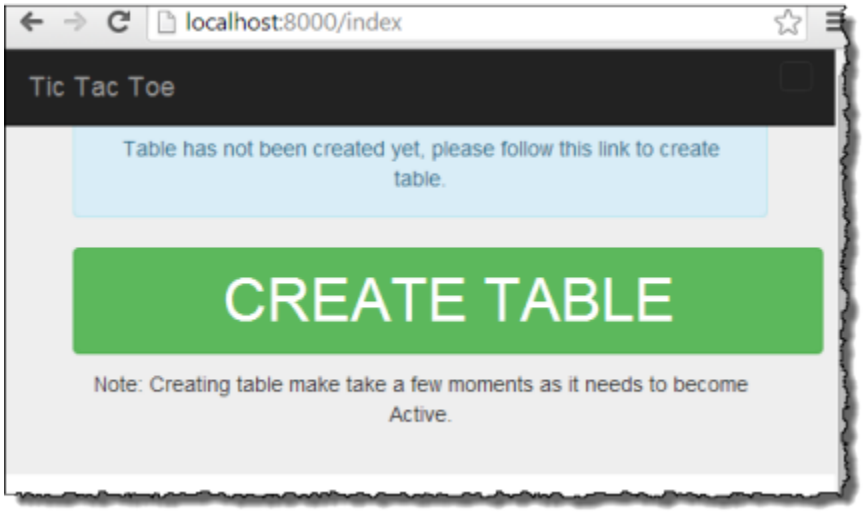
   瀏覽器會顯示出首頁。



4. 在 Log in (登入) 方塊中輸入 **`user1`**，以 user1 身分登入。

> **Note**
>
> 此範例應用程式不會執行任何使用者身分驗證。使用者 ID 僅會用於識別玩家。若兩名玩家使用同一個別名登入，應用程式仍會運作，如同您在兩個不同的瀏覽器中玩遊戲一樣。

5. 若這是您第一次玩遊戲，則會顯示一個頁面，要求您在 DynamoDB 中建立必要的資料表 (`Games`)。選擇 CREATE TABLE (建立資料表)。



6. 選擇 CREATE (建立) 建立第一個井字遊戲。
7. 在 Choose an Opponent (選擇對手) 方塊中輸入 **`user2`**，然後選擇 Create Game! (建立遊戲！)

執行此作業會在 Games 資料表中新增項目，而建立遊戲。它會將遊戲狀態設為 PENDING。

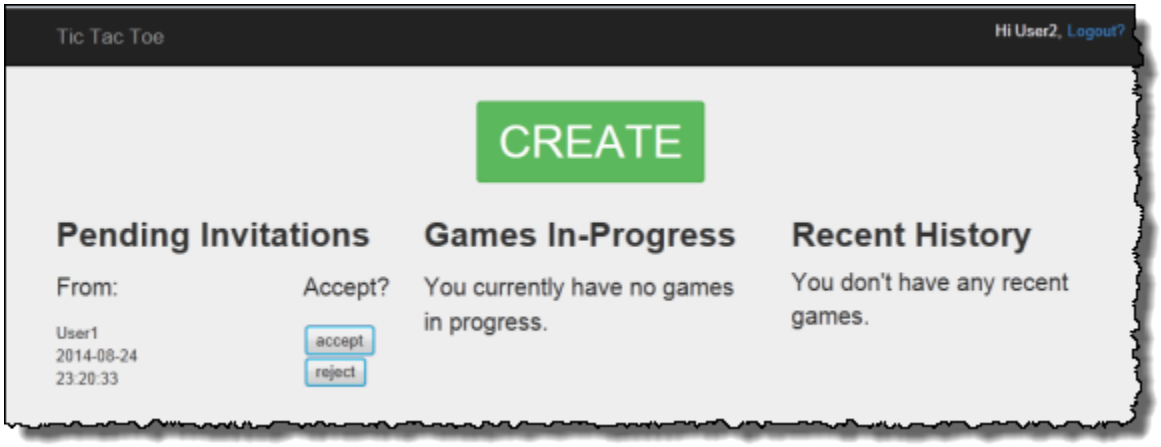
8. 開啟另一個瀏覽器視窗，並輸入如下內容。

```
http://localhost:5000/
```

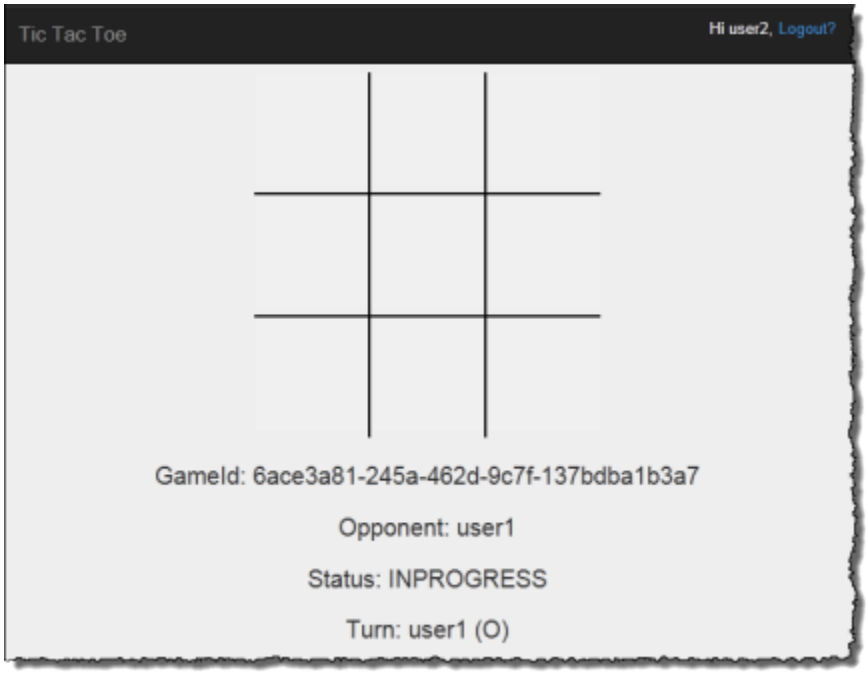
瀏覽器會透過 Cookie 傳遞資訊，因此建議您使用無痕 (incognito) 模式或隱私瀏覽，以免留下您的 Cookie。

9. 以 user2 登入。

此時將出現一個頁面，顯示 user1 發送的待定邀請。



10. 選擇 accept (接受) 接受邀請。



遊戲頁面會隨即顯示，其中有一個空白的井字遊戲方格。該頁面也會顯示相關的遊戲資訊，例如遊戲 ID、目前輪到哪一方及遊戲狀態。

11. 玩遊戲。

使用者每移動一步，Web 服務就會傳送請求到 DynamoDB，以條件式更新 Games 資料表中的遊戲項目。例如，條件會確保移動有效、方格可供使用者選擇，以及輪到該使用者移動。若移動有效，更新操作會為棋盤上的選擇新增對應屬性。更新操作也會將現有屬性的值設為可移動下一步的使用者。

在遊戲頁面上，應用程式每秒皆會發出非同步的 JavaScript 呼叫，最多達 5 分鐘，以檢查 DynamoDB 中的遊戲狀態是否已變更。若狀態已變更，應用程式會以新的資訊更新頁面。5 分鐘後，應用程式會停止發出請求，您必須重新整理頁面才能取得更新資訊。

步驟 2：檢查資料模型和實作詳細資訊

主題

- 2.1：基本資料模型
- 2.2：應用程式的實際運作 (程式碼演練)

2.1：基本資料模型

此範例應用程式的 DynamoDB 資料模型概念重點如下：

- Table (資料表)：在 DynamoDB 中，資料表為項目 (即紀錄) 的集合，而每一個項目都是名稱/值對的集合，稱為屬性。

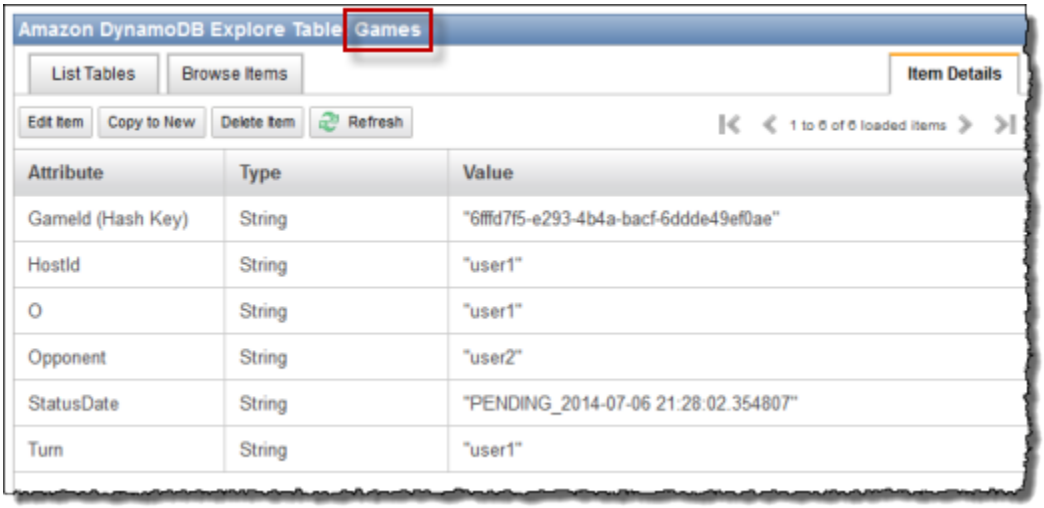
  在此井字遊戲範例中，應用程式會將所有遊戲資料存放在 Games 資料表。應用程式會為每一場遊戲在資料表中建立一個項目，並將所有遊戲資料以屬性的形式存放。一場井字遊戲最多可移動九步。因為 DynamoDB 資料表沒有結構描述，在必要屬性僅為主索引鍵的情況下，應用程式可針對每個遊戲項目存放不同數目的屬性。

  Games 資料表具有由單一屬性 GameId 組成的簡易主索引鍵，其類型為字串。該應用程式會為每一場遊戲指派唯一的 ID。如需 DynamoDB 主索引鍵的詳細資訊，請參閱 主索引鍵。

  當使用者透過邀請另一名使用者玩遊戲，來起始井字遊戲時，應用程式會在 Games 資料表中，以存放遊戲中繼資料的屬性建立新的項目，如下列範例：

  - HostId，起始遊戲的使用者。
  - Opponent，獲邀玩遊戲的使用者。
  - 輪到玩遊戲的使用者。起始遊戲的使用者先開始玩遊戲。
  - 在棋盤上使用 O 符號的使用者。起始遊戲的使用者使用 O 符號。

  此外，應用程式也會建立 StatusDate 串連屬性，將初始遊戲狀態標記為 PENDING。下列螢幕擷取畫面顯示範例項目在 DynamoDB 主控台中的樣子：

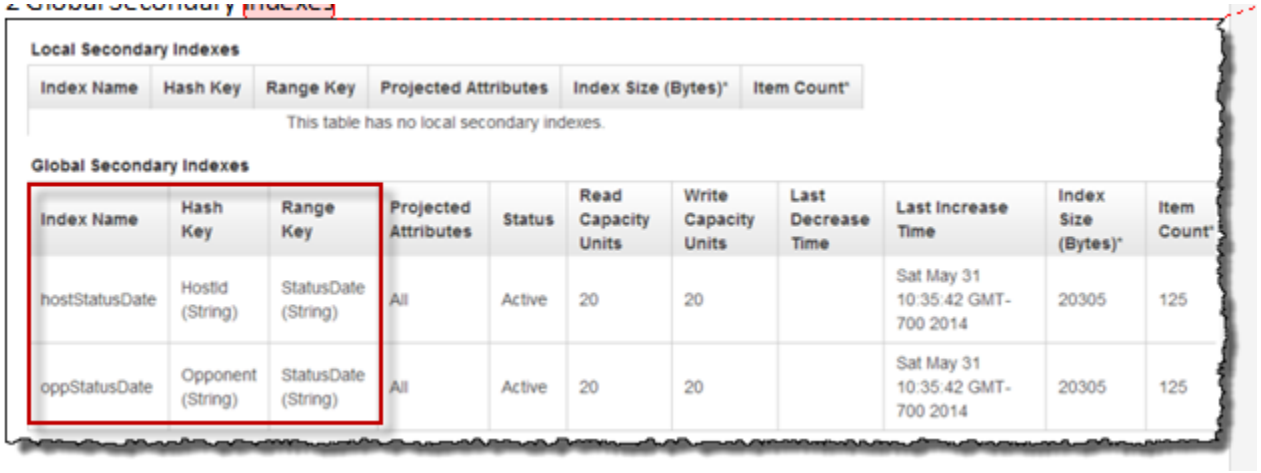


  隨著遊戲的進行，應用程式會為遊戲中的每步移動，新增一個屬性到資料表。屬性名稱為棋盤位置，例如 TopLeft 或 BottomRight。例如，移動可能是值為 TopLeft 的 O 屬性、值為 TopRight 的 O 屬性，及值為 BottomRight 的 X 屬性。屬性值不是 O 就是 X，取決於移動的使用者。例如，請參閱下列棋盤。

- Concatenated value attributes (串連值屬性)：StatusDate 屬性為串連值屬性的範例。在此方法中，並非透過建立個別屬性來存放遊戲狀態 (PENDING、IN_PROGRESS 和 FINISHED) 及日期 (最後一步的移動時間)，而是將其合併為單一屬性，例如 IN_PROGRESS_2014-04-30 10:20:32。

  然後應用程式會使用 StatusDate 屬性，將 StatusDate 指定為索引的排序索引鍵，來建立次要索引。使用 StatusDate 串連值屬性的好處會在接下來討論的索引中進一步說明。

- Global secondary indexes (全域次要索引)：您可以使用資料表的主索引鍵 (GameId) 有效率地查詢資料表，來尋找遊戲項目。若要查詢資料表上主索引鍵屬性以外的屬性，DynamoDB 支援建立次要索引。在此範例應用程式中，您會建置下列兩個次要索引：



  - HostId-StatusDate-index。此索引的分割區索引鍵為 HostId，排序索引鍵為 StatusDate。您可以使用此索引查詢 HostId，例如尋找由特定使用者發起的遊戲。
  - OpponentId-StatusDate-index。此索引的分割區索引鍵為 OpponentId，排序索引鍵為 StatusDate。您可以使用此索引查詢 Opponent，例如尋找對手為特定使用者的遊戲。

這些索引稱為全域次要索引，因為這些索引中的分割區索引鍵與資料表主索引鍵所使用的分割區索引鍵 (GameId) 不同。

請注意，這兩種索引皆指定 StatusDate 做為排序索引鍵。這可讓您進行下列作業：

- 您可以使用 BEGINS_WITH 比較運算子查詢。例如，您可以尋找由特定使用者發起且屬性為 IN_PROGRESS 的所有遊戲。在此案例中，BEGINS_WITH 運算子會檢查開頭為 StatusDate 的 IN_PROGRESS 值。
- DynamoDB 會根據排序索引鍵值，將項目依排序存放於索引中。因此若所有狀態字首都相同 (例如 IN_PROGRESS)，日期部分所使用的 ISO 格式會將項目從最舊排序到最新。此方法可讓特定查詢有效率地執行，例如下列查詢：
  - 擷取最多 10 個最近由登入使用者發起的 IN_PROGRESS 遊戲。針對此查詢，您要指定 HostId-StatusDate-index 索引。
  - 擷取最多 10 個最近登入使用者為對手的 IN_PROGRESS 遊戲。針對此查詢，您要指定 OpponentId-StatusDate-index 索引。

如需次要索引的詳細資訊，請參閱「在 DynamoDB 中使用次要索引來改善資料存取」。

2.2：應用程式的實際運作 (程式碼演練)

此應用程式有兩個主要頁面：

- Home page (首頁)：此頁面為使用者提供簡易登入、可建立新井字遊戲的 CREATE (建立) 按鈕、正在進行的遊戲清單、遊戲歷史記錄，以及任何作用中的待定遊戲邀請。

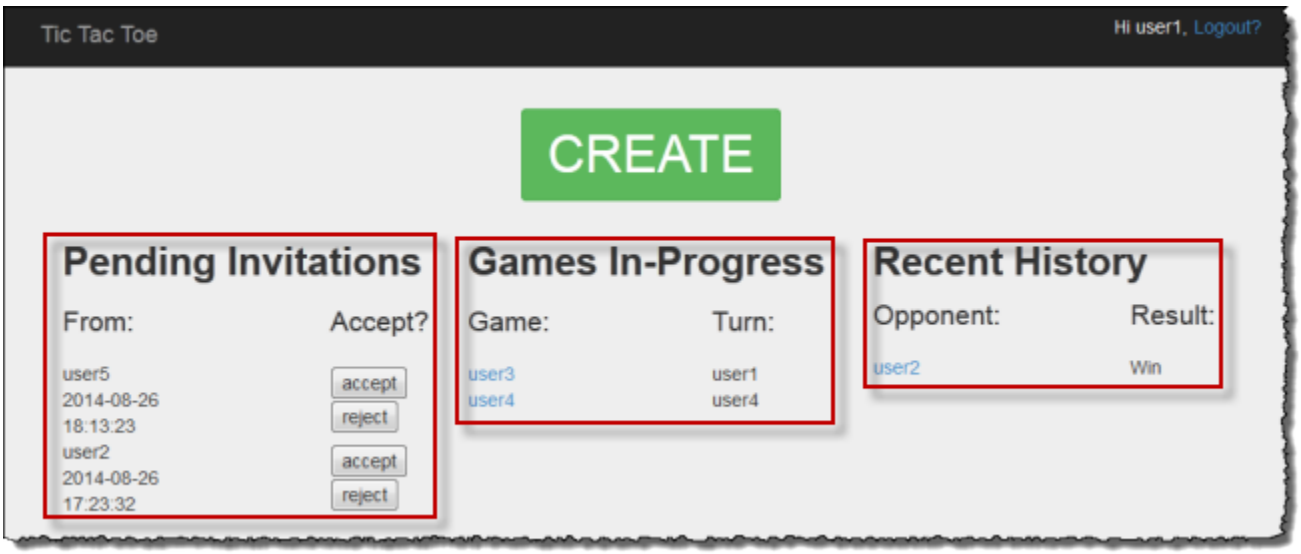
  首頁不會自動重新整理，您必須重新整理頁面才能重新整理清單。
- Game page (遊戲頁面)：此頁面會顯示使用者玩的井字遊戲方格。

  應用程式會每秒自動更新遊戲頁面一次。您瀏覽器中的 JavaScript 會每秒呼叫 Python Web 伺服器一次，查詢 Games 資料表以檢查資料表中的遊戲項目是否已變更。若已變更，JavaScript 會觸發頁面重新整理，讓使用者可看到更新的棋盤。

讓我們了解應用程式的詳細運作方式。

首頁

在使用者登入後，應用程式會顯示下列三項資訊清單。



- Invitations (邀請)：此清單會顯示最多 10 則最近其他使用者發送並正等待登入使用者接受的邀請。在先前的螢幕擷取畫面中，user1 有來自 user5 和 user2 的待定邀請。
- Games In-Progress (正在進行的遊戲) - 此清單會顯示最多 10 個最近正在進行的遊戲。這些遊戲是使用者正在玩的遊戲，其狀態為 `IN_PROGRESS`。在螢幕擷取畫面中，user1 正在與 user3 和 user4 玩井字遊戲。
- Recent History (近期歷史記錄) - 此清單會顯示最多 10 個最近使用者完成的遊戲，其狀態為 `FINISHED`。在螢幕擷取畫面中顯示的遊戲裡，user1 先前與 user2 玩過遊戲。清單會為每個完成的遊戲顯示遊戲結果。

在程式碼中，`index` 函數 (位於 `application.py` 中) 會發出下列三個呼叫，來擷取遊戲狀態資訊：

```
inviteGames     = controller.getGameInvites(session["username"])
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

這些呼叫各會從 DynamoDB 傳回 `Game` 物件所包裝的項目清單。在檢視中從這些物件擷取資料非常容易。index 函數會將這些物件清單傳遞到檢視，以轉譯 HTML。

```
return render_template("index.html",
                       user=session["username"],
                       invites=inviteGames,
                       inprogress=inProgressGames,
                       finished=finishedGames)
```

井字遊戲應用程式會定義 Game 類別，主要用以存放從 DynamoDB 擷取的遊戲資料。這些函數會傳回 Game 物件的清單，讓您可將應用程式剩餘的部分與有關 Amazon DynamoDB 項目的程式碼隔離。因此，這些函數可協助您將應用程式程式碼和資料存放層的詳細資訊分離。

此處說明的架構模式又稱為模型-檢視-控制器 (MVC) UI 模式。在此案例中，Game 物件執行個體 (代表資料) 為模型，而 HTML 頁面為檢視。控制器則分為兩個檔案。application.py 檔案具有 Flask 框架的控制器邏輯，商業邏輯則隔離在 gameController.py 檔案中。意即應用程式會將與 DynamoDB 軟體開發套件有關的所有事物存放在 dynamodb 資料夾中其自身的個別檔案裡。

讓我們檢閱三個函數，及其如何使用全域次要索引查詢 Games 資料表，來擷取相關資料。

使用 getGameInvites 擷取待定遊戲邀請的清單

getGameInvites 函數會擷取 10 個最近待定邀請的清單。這些遊戲已由使用者建立，但對手尚未接受遊戲邀請。在對手接受邀請前，這些遊戲皆會維持在 PENDING 狀態。若對手拒絕邀請，應用程式會從資料表移除對應項目。

函數指定查詢的方式如下所示：

- 其指定 OpponentId-StatusDate-index 索引，以搭配下列索引鍵值和比較運算子使用：
  - 分割區索引鍵為 OpponentId，並使用索引鍵 *user ID*。
  - 排序索引鍵為 StatusDate，並使用比較運算子和索引鍵值 beginswith="PENDING_"。

  您可以使用 OpponentId-StatusDate-index 索引擷取登入使用者獲邀的遊戲，即對手為登入使用者的遊戲。
- 查詢會將結果限制為 10 個項目。

```
gameInvitesIndex = self.cm.getGamesTable().query(
                                            Opponent__eq=user,
                                            StatusDate__beginswith="PENDING_",
                                            index="OpponentId-StatusDate-index",
                                            limit=10)
```

在索引中，針對每個 OpponentId (分割區索引鍵)，DynamoDB 都會依 StatusDate (排序索引鍵) 讓項目保持排序。因此，查詢傳回的遊戲將會是 10 個最近的遊戲。

使用 getGamesWithStatus 擷取特定狀態的遊戲清單

對手接受遊戲邀請後，遊戲狀態會變更為 IN_PROGRESS。遊戲完成後，該狀態會變更為 FINISHED。

尋找正在進行之遊戲的查詢與尋找已完成遊戲的查詢皆相同，唯一相異點在於狀態值不同。因此，應用程式會定義 getGamesWithStatus 函數，其使用狀態值做為參數。

```
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

下節會討論正在進行的遊戲，但該說明也同樣適用於已完成的遊戲。

指定使用者正在進行的遊戲清單同時包含下列項目：

- 由使用者發起之正在進行的遊戲
- 對手為該使用者之正在進行的遊戲

getGamesWithStatus 函數會執行下列兩項查詢，每次都會使用適當的次要索引。

- 函數會使用 HostId-StatusDate-index 索引查詢 Games 資料表。針對該索引，查詢會指定主索引鍵值：即分割區索引鍵 (HostId) 和排序索引鍵 (StatusDate) 兩者的值，以及比較運算子。

  ```
  hostGamesInProgress = self.cm.getGamesTable ().query(HostId__eq=user,
                                                       StatusDate__beginswith=status,
                                                       index="HostId-StatusDate-index",
                                                       limit=10)
  ```

  請注意比較運算子的 Python 語法：

  - HostId__eq=user 會指定對等比較運算子。
  - StatusDate__beginswith=status 會指定 BEGINS_WITH 比較運算子。
- 函數會使用 OpponentId-StatusDate-index 索引查詢 Games 資料表。

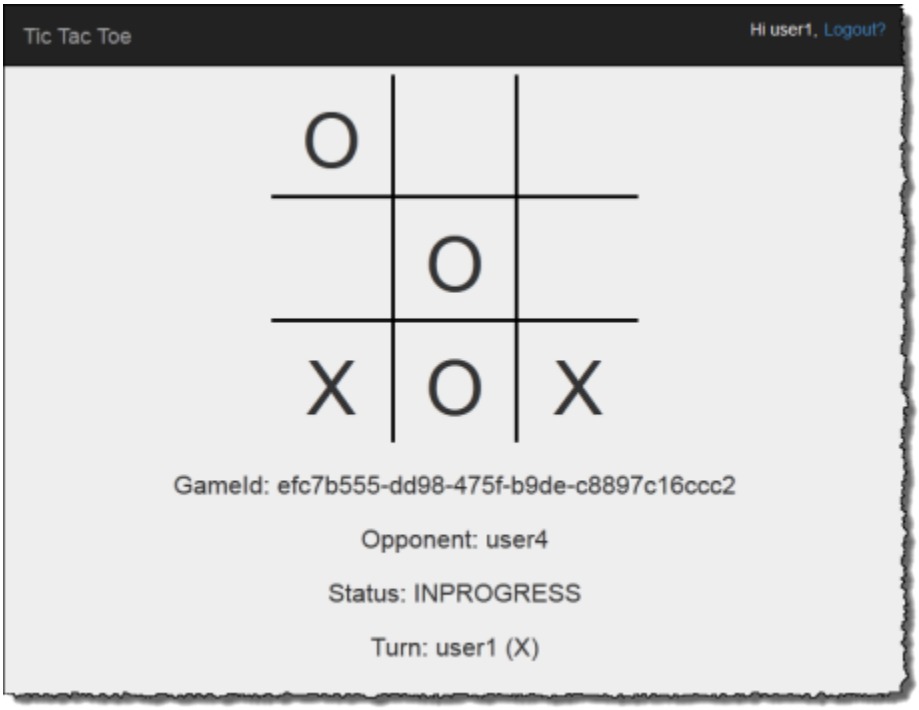
  ```
  oppGamesInProgress = self.cm.getGamesTable().query(Opponent__eq=user,
                                                     StatusDate__beginswith=status,
                                                     index="OpponentId-StatusDate-index",
                                                     limit=10)
  ```
- 函數接著會合併兩個清單、排序，然後為前 0 到 10 個項目建立 Game 物件的清單，並將清單傳回呼叫函數 (即 index)。

  ```
  games = self.mergeQueries(hostGamesInProgress,
                            oppGamesInProgress)
  ```

```
return games
```

遊戲頁面

遊戲頁面是使用者玩井字遊戲的位置。它會顯示遊戲方格和有關遊戲的資訊。下列螢幕擷取畫面顯示正在進行的範例遊戲：



應用程式會在下列情況中顯示遊戲頁面：

- 使用者建立遊戲，並邀請另一名使用者玩遊戲。

  在此案例中，頁面會顯示使用者為發起人，且遊戲狀態為 PENDING，等待對手接受。
- 使用者在首頁上接受其中一個待定邀請。

  在此案例中，頁面會顯示使用者為對手，且遊戲狀態為 IN_PROGRESS。

使用者在棋盤上的選擇會產生對應用程式的表單 POST 請求。意即，Flask 會使用 HTML 表單資料呼叫 selectSquare 函數 (位於 application.py 中)。然後此函數會呼叫 updateBoardAndTurn 函數 (位於 gameController.py 中) 來更新遊戲項目，如下所示：

- 它會新增移動專屬的屬性。
- 它會將 Turn 屬性值更新為移動下一步的使用者。

```
controller.updateBoardAndTurn(item, value, session["username"])
```

如果項目更新成功，函數會傳回 true，否則傳回 false。請注意下列關於 updateBoardAndTurn 函數的事項：

- 函數會呼叫 SDK for Python 的 update_item 函數，對現有項目進行有限次數的更新。函數會映射到 DynamoDB 中的 UpdateItem 操作。如需詳細資訊，請參閱 UpdateItem。

> **Note**
>
> UpdateItem 和 PutItem 操作之間的差異在於 PutItem 會取代整個項目。如需詳細資訊，請參閱 PutItem。

在 update_item 呼叫方面，程式碼會識別下列項目：

- Games 資料表的主索引鍵 (即 ItemId)。

```
key = { "GameId" : { "S" : gameId } }
```

- 要新增的屬性 (目前使用者移動專屬的屬性) 及其值 (例如 TopLeft="X")。

```
attributeUpdates = {
    position : {
        "Action" : "PUT",
        "Value" : { "S" : representation }
    }
}
```

- 條件必須為 true 才能更新：
  - 遊戲必須正在進行。即 StatusDate 屬性值的開頭必須為 IN_PROGRESS。
  - 目前輪次必須是 Turn 屬性指定的有效使用者輪次。
  - 方格必須可供使用者選擇。即對應到該方格的屬性不得存在。

```
expectations = {"StatusDate" : {"AttributeValueList": [{"S" : "IN_PROGRESS_"}],
    "ComparisonOperator": "BEGINS_WITH"},
    "Turn" : {"Value" : {"S" : current_player}},
    position : {"Exists" : False}}
```

函數現在會呼叫 update_item 更新項目。

```
self.cm.db.update_item("Games", key=key,
    attribute_updates=attributeUpdates,
    expected=expectations)
```

函數傳回後，selectSquare 函數會呼叫 redirect，如下列範例所示。

```
redirect("/game="+gameId)
```

此呼叫會使瀏覽器重新整理。在此重新整理中，應用程式會檢查遊戲是否以獲勝或平手結束。若遊戲已結束，則應用程式會根據結果更新遊戲項目。

步驟 3：使用 DynamoDB 服務在生產環境中部署

主題

- 3.1：建立 Amazon EC2 的 IAM 角色
- 3.2：在 Amazon DynamoDB 中建立遊戲資料表
- 3.3：組合及部署井字遊戲應用程式的程式碼
- 3.4：設定 AWS Elastic Beanstalk 環境

在先前的章節中，您已在電腦本機上使用 DynamoDB 本機版部署和測試井字遊戲應用程式。現在，您會在生產環境中部署應用程式，如下所示：

- 使用easy-to-use的服務來部署應用程式 AWS Elastic Beanstalk，以部署和擴展 Web 應用程式和服務。如需詳細資訊，請參閱將 flask 應用程式部署至 AWS Elastic Beanstalk。

  Elastic Beanstalk 會啟動一或多個 Amazon Elastic Compute Cloud (Amazon EC2) 執行個體，您可以透過 Elastic Beanstalk 加以設定，且您的井字遊戲應用程式也會在這些執行個體上執行。
- 使用 Amazon DynamoDB 服務，建立位於 AWS 上而非您電腦本機上的 Games 資料表。

此外，您也必須設定許可。您建立的任何 AWS 資源，例如 DynamoDB 中的Games資料表，預設都是私有的。只有資源擁有者 (即建立 Games 資料表的 AWS 帳戶) 才能存取此資料表。因此根據預設，您的井字遊戲應用程式無法更新 Games 資料表。

若要授予必要的許可，您可以建立 AWS Identity and Access Management (IAM) 角色，並授予此角色存取Games資料表的許可。您的 Amazon EC2 執行個體首先會擔任此角色。作為回應， AWS 會傳回

Amazon EC2 執行個體可用來代表 Tic-Tac-Toe 應用程式更新Games資料表的暫時安全登入資料。當您設定 Elastic Beanstalk 應用程式時，您可以指定 Amazon EC2 執行個體可以擔任的 IAM 角色。如需 IAM 角色的詳細資訊，請參閱 Amazon EC2 使用者指南中的 Amazon EC2 的 IAM 角色。

> **Note**
>
> 為 Tic-Tac-Toe 應用程式建立 Amazon EC2 執行個體之前，您必須先決定您希望 Elastic Beanstalk 建立執行個體 AWS 的區域。在您建立 Elastic Beanstalk 應用程式後，您要在組態檔案中提供同一個區域名稱和端點。井字遊戲應用程式會使用此檔案中的資訊，來建立 Games 資料表，並在特定的 AWS 區域中傳送後續請求。DynamoDB Games 資料表和 Elastic Beanstalk 啟動的 Amazon EC2 執行個體皆必須位於同一個區域。如需可用區域清單，請參閱 Amazon Web Services 一般參考 中的 Amazon DynamoDB。

總而言之，您會執行下列作業，以在生產環境中部署井字遊戲應用程式：

1. 使用 IAM 服務建立 IAM 角色。您會將政策連接到此角色，授予 DynamoDB 動作的許可，使其可存取 Games 資料表。
2. 將井字遊戲應用程式的程式碼和組態檔案組合，並建立 .zip 檔案。您會使用這個 .zip 檔案，將井字遊戲應用程式的程式碼提供給 Elastic Beanstalk，以放置在您的伺服器上。如需建立套件的詳細資訊，請參閱《AWS Elastic Beanstalk 開發人員指南》中的建立應用程式來源套件。

   在組態檔案 (beanstalk.config) 中，您要提供 AWS 區域和端點的資訊。井字遊戲應用程式會使用此資訊判斷要通訊的 DynamoDB 區域。
3. 設定 Elastic Beanstalk 環境。Elastic Beanstalk 會啟動一或多個 Amazon EC2 執行個體，並在上面部署您的井字遊戲應用程式套件。Elastic Beanstalk 環境就緒後，您要新增 CONFIG_FILE 環境變數，來提供組態檔案名稱。
4. 建立 DynamoDB 資料表。使用 Amazon DynamoDB 服務，您可以在 上建立Games資料表 AWS，而不是在本機電腦上建立資料表。請記得，此資料表具有由字串類型之 GameId 分割區索引鍵組成的簡易主索引鍵。
5. 在生產環境中測試遊戲。

3.1：建立 Amazon EC2 的 IAM 角色

建立 Amazon EC2 類型的 IAM 角色，可讓執行您井字遊戲應用程式的 Amazon EC2 執行個體擔任正確的角色，並發出應用程式請求，來存取 Games 資料表。當您建立角色時，請選擇 Custom Policy (自訂政策) 選項，並將下列政策複製並貼上。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Action":[
            "dynamodb:ListTables"
         ],
         "Effect":"Allow",
         "Resource":"*"
      },
      {
         "Action":[
            "dynamodb:*"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:dynamodb:us-west-2:922852403271:table/Games",
            "arn:aws:dynamodb:us-west-2:922852403271:table/Games/index/*"
         ]
      }
   ]
}
```

如需進一步說明，請參閱《IAM 使用者指南》中的為 AWS 服務 (AWS 管理主控台) 建立角色。

3.2：在 Amazon DynamoDB 中建立遊戲資料表

DynamoDB 中的 Games 資料表會存放遊戲資料。若沒有該資料表，應用程式會為您建立資料表。在此案例中，我們會讓應用程式建立 Games 資料表。

3.3：組合及部署井字遊戲應用程式的程式碼

若您遵循此範例的步驟，則您已有下載完成的井字遊戲應用程式。若沒有，請下載應用程式，並將所有檔案解壓縮到您本機電腦上的資料夾。如需指示，請參閱 步驟 1：在本機上部署及測試。

當您解壓縮所有檔案後，您會有一個 code 資料夾。若要將此資料夾傳遞給 Elastic Beanstalk，您要將此資料夾的內容組合成 .zip 檔案。首先，請在該資料夾新增組態檔案。您的應用程式會使用區域和端點的資訊在指定區域中建立 DynamoDB 資料表，並使用指定端點發出後續的資料表操作請求。

1. 切換到您下載井字遊戲應用程式的所在資料夾。
2. 在應用程式的根資料夾中，使用下列內容建立名為 `beanstalk.config` 的文字檔案。

```
[dynamodb]
region=<AWS region>
endpoint=<DynamoDB endpoint>
```

例如，您可以使用下列內容。

```
[dynamodb]
region=us-west-2
endpoint=dynamodb.us-west-2.amazonaws.com
```

如需可用區域清單，請參閱《Amazon Web Services 一般參考》中的 Amazon DynamoDB。

> **Important**
>
> 組態檔案中指定的區域為井字遊戲應用程式在 DynamoDB 中建立 `Games` 資料表的位置。您必須在同一個區域中建立下一節討論的 Elastic Beanstalk 應用程式。

> **Note**
>
> 當您建立 Elastic Beanstalk 應用程式時，您要請求啟動您可選擇類型的環境。若要測試井字遊戲範例應用程式，您可以選擇 Single Instance (單一執行個體) 環境類型，並跳過下列步驟，進行下一步。
> 但是，Load balancing, autoscaling (負載平衡、自動擴展) 環境類型可提供具高可用性且具可擴展性的環境，您應在建立和部署其他應用程式時加以考慮。若您選擇此環境類型，您還要產生 UUID，並將其新增到組態檔案，如下所示。
>
> ```
> [dynamodb]
> region=us-west-2
> endpoint=dynamodb.us-west-2.amazonaws.com
> [flask]
> secret_key= 284e784d-1a25-4a19-92bf-8eeb7a9example
> ```
>
> 在用戶端與伺服器端的通訊中，當伺服器傳送回應時，基於安全性考量，伺服器會傳送簽署過的 Cookie，讓用戶端在下次請求時傳回伺服器。當只有一個伺服器時，伺服器可在啟

動時於本機產生加密金鑰。當有多個伺服器時，他們都必須知道同一個加密金鑰，否則無法讀取對等伺服器設定的 Cookie。將 `secret_key` 新增到組態檔案，可以告知所有伺服器使用此加密金鑰。

3. 壓縮應用程式根資料夾的內容 (其中包含 `beanstalk.config` 檔案)，例如 `TicTacToe.zip`。
4. 將 `.zip` 檔案上傳到 Amazon Simple Storage Service (Amazon S3) 儲存貯體。在下一節中，您會將此 `.zip` 檔案提供給 Elastic Beanstalk，以上傳到伺服器。

   如需如何上傳檔案至 Amazon S3 儲存貯體的說明，請參閱《Amazon Simple Storage Service 使用者指南》中的建立儲存貯體及新增物件至儲存貯體。

3.4：設定 AWS Elastic Beanstalk 環境

在此步驟中，您會建立 Elastic Beanstalk 應用程式，其為包含環境的元件集合。針對此範例，您會啟動一個 Amazon EC2 執行個體，以部署並執行您的井字遊戲應用程式。

1. 輸入下列自訂 URL 設定 Elastic Beanstalk 主控台，以設定環境。

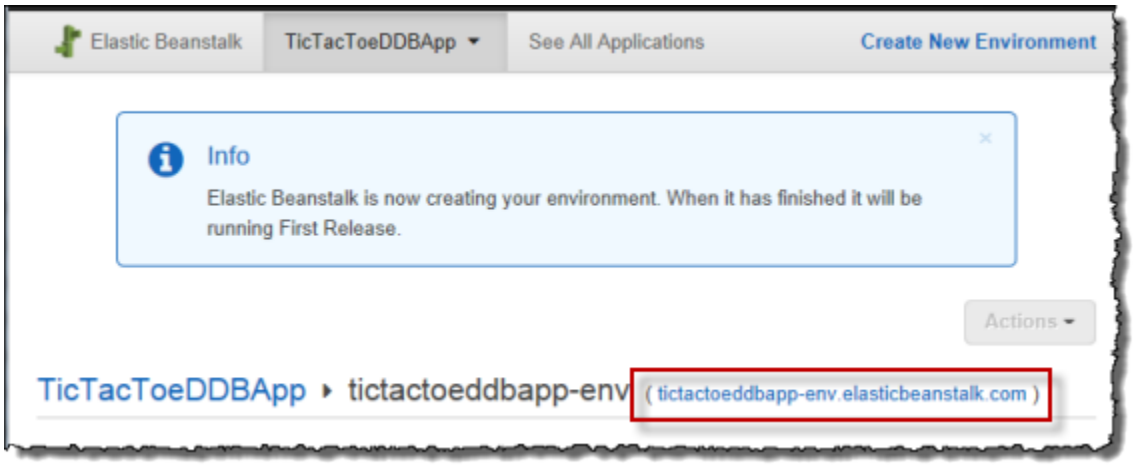
```
https://console.aws.amazon.com/elasticbeanstalk/?region=<AWS-Region>#/
newApplication
?applicationName=TicTacToeyour-name
&solutionStackName=Python
&sourceBundleUrl=https://s3.amazonaws.com/<bucket-name>/TicTacToe.zip
&environmentType=SingleInstance
&instanceType=t1.micro
```

   如需自訂 URL 的詳細資訊，請參閱《AWS Elastic Beanstalk 開發人員指南》中的建構立即啟動 URL。在 URL 方面，請注意下列事項：

   - 您必須提供 AWS 區域名稱 （與您在組態檔案中提供的名稱相同）、Amazon S3 儲存貯體名稱和物件名稱。
   - 為進行測試，URL 會請求 SingleInstance 環境類型及 `t1.micro` 執行個體類型。
   - 應用程式名稱必須是唯一的。因此，在先前的 URL 中，建議您在 `applicationName` 之前加上您的名稱。

   執行此操作會開啟 Elastic Beanstalk 主控台。在某些案例中，您可能需要登入。

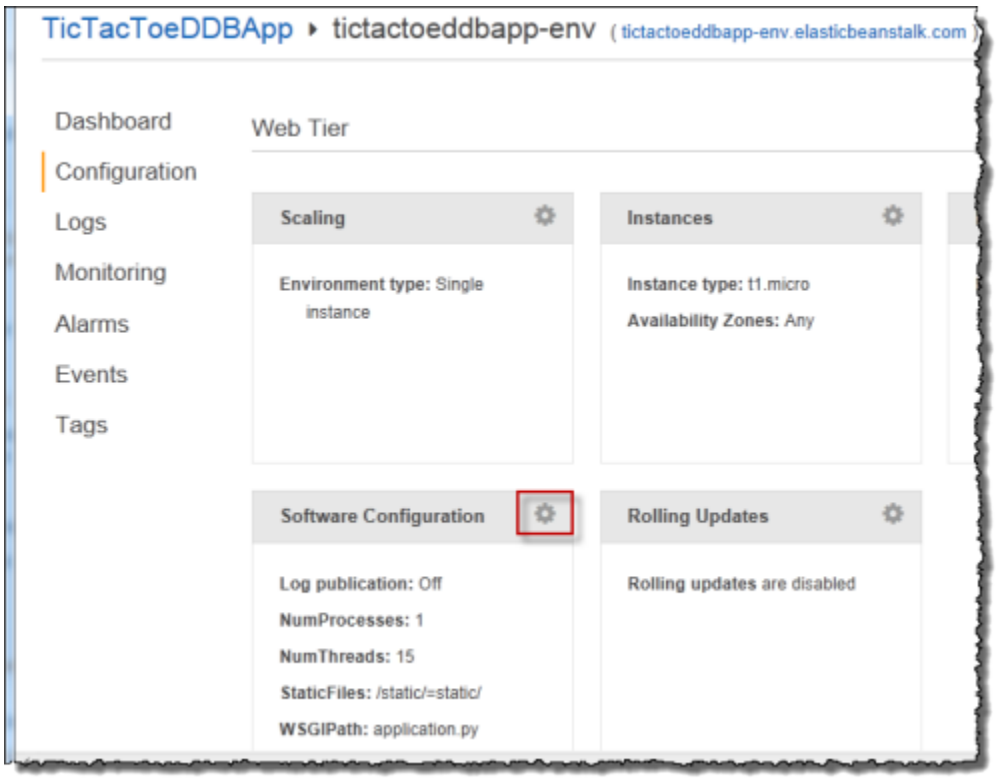
2. 在 Elastic Beanstalk 主控台中，選擇 Review and Launch (檢閱和啟動)，然後選擇 Launch (啟動)。
3. 請記下 URL，以供未來參考。此 URL 會開啟您的井字遊戲應用程式首頁。



4. 設定井字遊戲應用程式，讓其可得知組態檔案的位置。

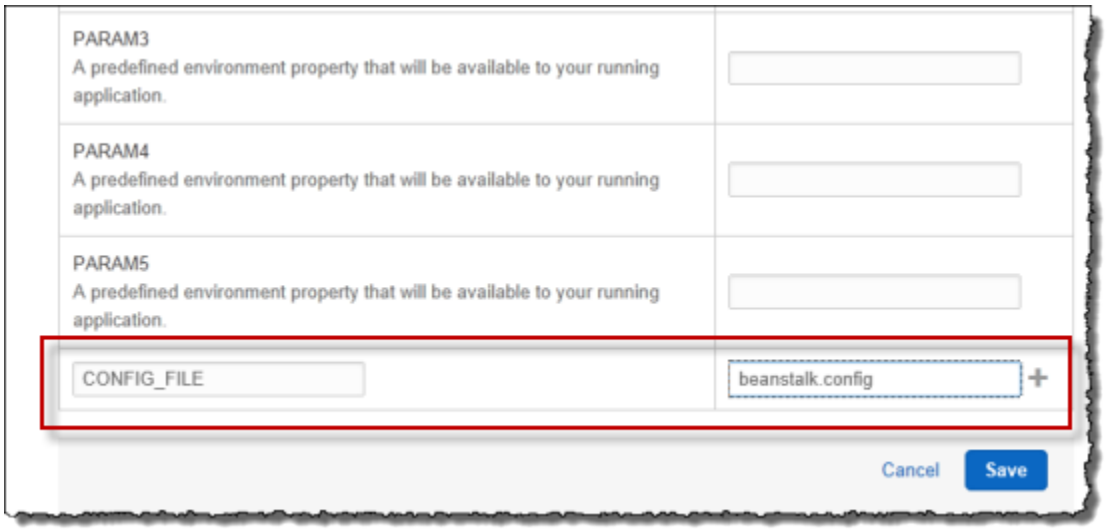
   Elastic Beanstalk 建立應用程式後，請選擇 Configuration (組態)。

   a. 選擇 Software Configuration (軟體組態) 旁邊的齒輪圖示，如下列螢幕擷取畫面所示。



b. 在 Environment Properties (環境屬性) 區段的結尾，輸入 **CONFIG_FILE** 及其值 **beanstalk.config**，然後選擇 Save (儲存)。

完成環境更新需要幾分鐘的時間。

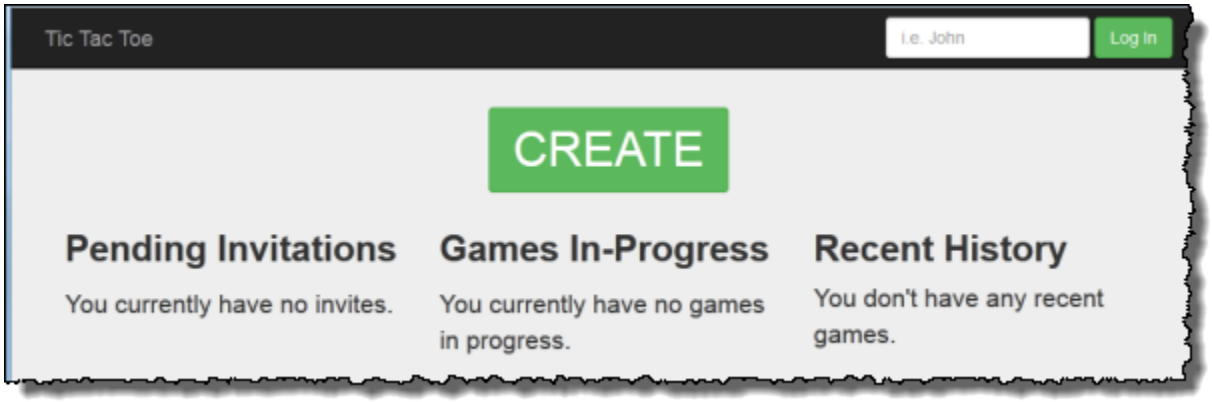


更新完成後，您便可以玩遊戲。

5. 在瀏覽器中，輸入您在先前步驟中複製的 URL，如下列範例所示。

```
http://<pen-name>.elasticbeanstalk.com
```

執行此作業會開啟應用程式首頁。



6. 以 testuser1 登入，然後選擇 CREATE (建立)，以開始新的井字遊戲。
7. 在 Choose an Opponent (選擇對手) 方塊中輸入 **testuser2**。

8. 開啟另一個瀏覽器視窗。

   確認您已清除您瀏覽器視窗中的所有 Cookie，以免使用同一個使用者登入。

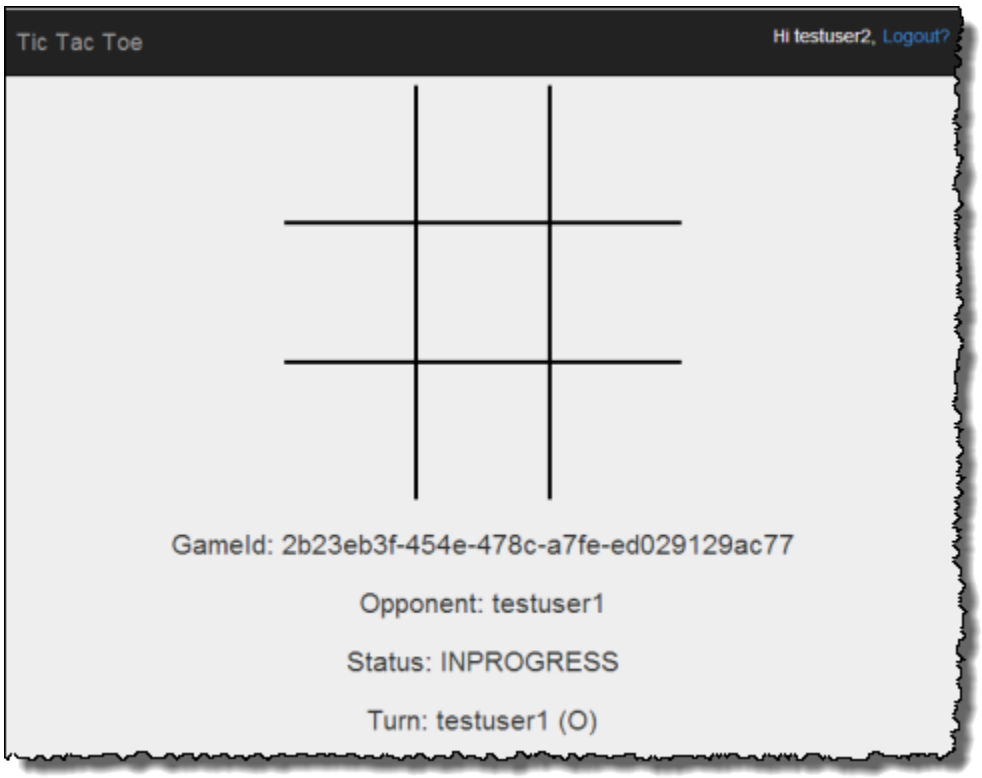
9. 輸入同一個 URL，以開啟應用程式首頁，如下列範例所示。

```
http://<env-name>.elasticbeanstalk.com
```

10. 以 testuser2 登入。

11. 針對待定邀請清單中 testuser1 發送的邀請，選擇 accept (接受)。

12. 現在遊戲頁面便會出現。



testuser1 和 testuser2 都可玩遊戲。每移動一步，應用程式就會將該移動儲存在 Games 資料表中的對應項目裡。

步驟 4：清除資源

現在您已完成井字遊戲應用程式的部署和測試。該應用程式涵蓋 Amazon DynamoDB 上的端對端 Web 應用程式開發，但不包括使用者身分驗證。應用程式在首頁使用登入資訊，只是為了在建立遊戲時新增玩家的名稱。在生產應用程式中，您要新增必要的程式碼，才能執行使用者登入和身分驗證。

若您已完成測試，可以移除您建立用來測試井字遊戲應用程式的資源，以免產生任何費用。

移除您建立的資源

1. 移除您在 DynamoDB 中建立的 Games 資料表。
2. 終止 Elastic Beanstalk 環境，以便釋放 Amazon EC2 執行個體。
3. 刪除您建立的 IAM 角色。
4. 移除您在 Amazon S3 中建立的物件。

DynamoDB 中的保留字

下列關鍵字會保留給 DynamoDB 使用。請勿在表達式中使用任何這些單字作為屬性名稱。此表不區分大小寫。

如果需要撰寫的表達式會包含與 DynamoDB 保留字衝突的屬性名稱，您可以定義要取代保留字的表達式屬性名稱。如需更多詳細資訊，請參閱 DynamoDB 中的表達式屬性名稱 (別名)。

```
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 適用於 Java 的 SDK 1.x 範例

本節包含使用適用於 Java 1.x 的開發套件的 DAX 應用程式範例程式碼。

主題

- 搭配適用於 Java 1.x 的 AWS 開發套件使用 DAX
- 修改適用於 Java 1.x 的開發套件的現有應用程式來使用 DAX
- 使用適用於 Java 1.x 的開發套件查詢全域次要索引

搭配適用於 Java 1.x 的 AWS 開發套件使用 DAX

請按照此程序操作，在 Amazon EC2 執行個體上執行 Amazon DynamoDB Accelerator (DAX) 的 Java 範例。

> **Note**
>
> 這些說明是針對使用適用於 Java 1.x 的 AWS 軟體開發套件的應用程式。如需使用適用於 Java 2.x 的 AWS 軟體開發套件的應用程式，請參閱 Java 與 DAX。

執行 DAX 的 Java 範例

1. 安裝 Java 開發套件 (JDK)。

   ```
   sudo yum install -y java-devel
   ```

2. 下載 適用於 Java 的 AWS SDK (`.zip` 檔案)，並將其解壓縮。

```
wget http://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip

unzip aws-java-sdk.zip
```

3. 下載 DAX Java 用戶端 (`.jar` 檔) 最新版本。

```
wget http://dax-sdk.s3-website-us-west-2.amazonaws.com/java/DaxJavaClient-
latest.jar
```

> **Note**
>
> DAX SDK for Java 用戶端可在 Apache Maven 上取得。如需詳細資訊，請參閱 使用用戶端做為 Apache Maven 依存項目。

4. 設定您的 CLASSPATH 變數。在此範例中，將 *sdkVersion* 取代為 適用於 Java 的 AWS SDK 的實際版本號碼 (例如 `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. 下載範例程式來源碼 (`.zip` 檔案)。

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

下載完成後，解壓縮來源檔案。

```
unzip TryDax.zip
```

6. 前往 Java 程式碼目錄並編譯程式碼，如下所示。

```
cd TryDax/java/
javac TryDax*.java
```

7. 執行程式。

```
java TryDax
```

您應該會看到類似下列的輸出。

```
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.
```

記下計時資訊：GetItem、Query 和 Scan 測試所需要的毫秒數。

8. 在先前的步驟中，您已針對 DynamoDB 端點執行程式。現在，請再次執行程式，但這一次 GetItem、Query 和 Scan 操作會由您的 DAX 叢集處理。

   若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

   - 使用 DynamoDB 主控台：選擇您的 DAX 叢集。叢集端點會在主控台上顯示，如以下範例。

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - 使用 AWS CLI：輸入下列命令。

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     如下列範例所示，叢集端點會在輸出上顯示。

     ```
     {
         "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"
     }
     ```

   現在重新執行程式，但這一次，請將叢集端點做為命令列參數指定。

   ```
   java TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
   ```

   查看輸出的剩餘部分，並記下計時資訊。使用 DAX 的 GetItem、Query 和 Scan 已耗用時間應遠低於使用 DynamoDB 的已耗用時間。

如需此程式的詳細資訊，請參閱下列各節：

- TryDax.java
- TryDaxHelper.java

- TryDaxTests.java

使用用戶端做為 Apache Maven 依存項目

遵循這些步驟，在您的應用程式中將 DAX SDK for Java 用戶端做為依存項目使用。

使用用戶端作為 Maven 依存項目

1. 下載並安裝 Apache Maven。如需詳細資訊，請參閱下載 Apache Maven 和安裝 Apache Maven。
2. 將用戶端 Maven 依存項目新增至您應用程式的專案物件模型 (POM) 檔案。在此範例中，將 `x.x.x.x` 取代為用戶端的實際版本號碼 (例如 `1.0.200704.0`)。

```
<!--Dependency:-->
<dependencies>
    <dependency>
     <groupId>com.amazonaws</groupId>
     <artifactId>amazon-dax-client</artifactId>
     <version>x.x.x.x</version>
    </dependency>
</dependencies>
```

TryDax.java

`TryDax.java` 檔案包含 `main` 方法。若您不使用任何命令列參數執行程式，它會建立 Amazon DynamoDB 用戶端，並使用該用戶端進行所有 API 操作。若您在命令列指定 DynamoDB Accelerator (DAX) 叢集端點，程式也會建立 DAX 用戶端，並用它來進行 `GetItem`、`Query` 和 `Scan` 操作。

您可以使用幾種方法修改程式：

- 使用 DAX 用戶端，而非 DynamoDB 用戶端。如需詳細資訊，請參閱 Java 與 DAX。
- 為測試資料表選擇不同的名稱。
- 變更 `helper.writeData` 參數，修改寫入的項目數目。第二個參數是分割區索引鍵的數字，第三個參數則為排序索引鍵的數字。根據預設，程式會使用 1-10 作為分割區索引鍵的值，使用 1-10 作為排序索引鍵的值，總共會將 100 個項目寫入資料表。如需詳細資訊，請參閱 TryDaxHelper.java。
- 修改 `GetItem`、`Query` 和 `Scan` 測試的數字，並修改其參數。
- 將包含 `helper.createTable` 和 `helper.deleteTable` 的行變更為註解 (若您不希望每次執行程式時都建立和刪除資料表的話)。

> **Note**
>
> 若要執行此程式，您可以設定 Maven 將適用於 Java 的 DAX 軟體開發套件的用戶端與 適用於 Java 的 AWS SDK 用作依存項目。如需詳細資訊，請參閱 使用用戶端做為 Apache Maven 依存項目。
> 或者，您必須在 classpath 中同時下載和包含 DAX Java 用戶端及 適用於 Java 的 AWS SDK。請參閱 Java 與 DAX 以取得設定您 CLASSPATH 變數的範例。

```
public class TryDax {

    public static void main(String[] args) throws Exception {

        TryDaxHelper helper = new TryDaxHelper();
        TryDaxTests tests = new TryDaxTests();

        DynamoDB ddbClient = helper.getDynamoDBClient();
        DynamoDB daxClient = null;
        if (args.length >= 1) {
            daxClient = helper.getDaxClient(args[0]);
        }

        String tableName = "TryDaxTable";

        System.out.println("Creating table...");
        helper.createTable(tableName, ddbClient);
        System.out.println("Populating table...");
        helper.writeData(tableName, ddbClient, 10, 10);

        DynamoDB testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }

        System.out.println("Running GetItem, Scan, and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");

        // GetItem
```

```
        tests.getItemTest(tableName, testClient, 1, 10, 5);

        // Query
        tests.queryTest(tableName, testClient, 5, 2, 9, 5);

        // Scan
        tests.scanTest(tableName, testClient, 5);

        helper.deleteTable(tableName, ddbClient);
    }

}
```

TryDaxHelper.java

`TryDaxHelper.java` 檔案包含公用程式方法。

`getDynamoDBClient` 和 `getDaxClient` 方法提供 Amazon DynamoDB 和 DynamoDB Accelerator (DAX) 用戶端。針對控制平面操作 (`CreateTable`、`DeleteTable`) 及寫入操作，程式會使用 DynamoDB 用戶端。若您指定 DAX 叢集端點，主程式會建立 DAX 用戶端來執行讀取操作 (`GetItem`、`Query`、`Scan`)。

其他 `TryDaxHelper` 方法 (`createTable`、`writeData`、`deleteTable`) 用於設定及卸除 DynamoDB 資料表和其資料。

您可以使用幾種方法修改程式：

- 針對資料表使用不同的佈建輸送量設定。
- 修改每個寫入項目的大小 (請參閱 `stringSize` 方法中的 `writeData` 變數)。
- 修改 `GetItem`、`Query` 和 `Scan` 測試的數字及其參數。
- 將包含 `helper.CreateTable` 和 `helper.DeleteTable` 的行變更為註解 (若您不希望每次執行程式時都建立和刪除資料表的話)。

> **Note**
>
> 若要執行此程式，您可以設定 Maven 將適用於 Java 的 DAX 軟體開發套件的用戶端與 適用於 Java 的 AWS SDK 用作依存項目。如需詳細資訊，請參閱 使用用戶端做為 Apache Maven 依存項目。

或者，您必須在 classpath 中同時下載和包含 DAX Java 用戶端及 適用於 Java 的 AWS SDK。請參閱 Java 與 DAX 以取得設定您 CLASSPATH 變數的範例。

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
import com.amazonaws.util.EC2MetadataUtils;

public class TryDaxHelper {

    private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

    DynamoDB getDynamoDBClient() {
        System.out.println("Creating a DynamoDB client");
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
                .withRegion(region)
                .build();
        return new DynamoDB(client);
    }

    DynamoDB getDaxClient(String daxEndpoint) {
        System.out.println("Creating a DAX client with cluster endpoint " +
 daxEndpoint);
        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
        AmazonDynamoDB client = daxClientBuilder.build();
        return new DynamoDB(client);
    }

    void createTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("Attempting to create table; please wait...");
```

```
            table = client.createTable(tableName,
                    Arrays.asList(
                            new KeySchemaElement("pk", KeyType.HASH), // Partition key
                            new KeySchemaElement("sk", KeyType.RANGE)), // Sort key
                    Arrays.asList(
                            new AttributeDefinition("pk", ScalarAttributeType.N),
                            new AttributeDefinition("sk", ScalarAttributeType.N)),
                    new ProvisionedThroughput(10L, 10L));
            table.waitForActive();
            System.out.println("Successfully created table.  Table status: " +
                    table.getDescription().getTableStatus());

        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    void writeData(String tableName, DynamoDB client, int pkmax, int skmax) {
        Table table = client.getTable(tableName);
        System.out.println("Writing data to the table...");

        int stringSize = 1000;
        StringBuilder sb = new StringBuilder(stringSize);
        for (int i = 0; i < stringSize; i++) {
            sb.append('X');
        }
        String someData = sb.toString();

        try {
            for (Integer ipk = 1; ipk <= pkmax; ipk++) {
                System.out.println(("Writing " + skmax + " items for partition key: " +
ipk));
                for (Integer isk = 1; isk <= skmax; isk++) {
                    table.putItem(new Item()
                            .withPrimaryKey("pk", ipk, "sk", isk)
                            .withString("someData", someData));
                }
            }
        } catch (Exception e) {
            System.err.println("Unable to write item:");
            e.printStackTrace();
        }
```

```
    }

    void deleteTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            table.delete();
            table.waitForDelete();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
            e.printStackTrace();
        }
    }

}
```

TryDaxTests.java

`TryDaxTests.java` 檔案包含對 Amazon DynamoDB 中測試資料表執行讀取操作的方法。這些方法皆未考慮其存取資料的方法 (使用 DynamoDB 用戶端或 DAX 用戶端)，因此不需修改應用程式邏輯。

您可以使用幾種方法修改程式：

- 修改 `queryTest` 方法，使其使用不同的 `KeyConditionExpression`。
- 將 `ScanFilter` 新增至 `scanTest` 方法，使其只傳回一部分的項目。

> **Note**
>
> 若要執行此程式，您可以設定 Maven 將適用於 Java 的 DAX 軟體開發套件的用戶端與 適用於 Java 的 AWS SDK 用作依存項目。如需詳細資訊，請參閱 使用用戶端做為 Apache Maven 依存項目。
> 或者，您必須在 classpath 中同時下載和包含 DAX Java 用戶端及 適用於 Java 的 AWS SDK。請參閱 Java 與 DAX 以取得設定您 CLASSPATH 變數的範例。

```
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));
    }
}
```

修改適用於 Java 1.x 的開發套件的現有應用程式來使用 DAX

如果您已有使用 Amazon DynamoDB 的 Java 應用程式，則需要進行修改，使其可存取您的 DynamoDB Accelerator (DAX) 叢集。您不需要重新撰寫整個應用程式，因為 DAX Java 用戶端與 適用於 Java 的 AWS SDK中所含的 DynamoDB 低階用戶端十分相似。

> **Note**
>
> 這些指示適用於使用適用於 Java 的 AWS SDK 1.x 的應用程式。如需使用適用於 Java 2.x 的 AWS 軟體開發套件的應用程式，請參閱 修改現有應用程式以使用 DAX。

假設您有名為 `Music` 的 DynamoDB 資料表。資料表的分割區索引鍵為 `Artist`，而排序索引鍵為 `SongTitle`。以下程式會直接讀取 `Music` 資料表中的項目。

```
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());
        }
    }
}
```

若要修改程式，請將 DynamoDB 用戶端取代為 DAX 用戶端。

```
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)
        ** ...
        */

    }
}
```

使用 DynamoDB 文件 API

適用於 Java 的 AWS SDK 提供 DynamoDB 的文件界面。文件 API 會作為低階 DynamoDB 用戶端的包裝函式。如需詳細資訊，請參閱文件介面。

文件介面也可以與低階 DAX 用戶端搭配使用，如以下範例所示。

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.GetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;

public class GetMusicItemWithDocumentApi {

    public static void main(String[] args) throws Exception {

        //Create a DAX client

        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
        AmazonDynamoDB client = daxClientBuilder.build();

        // Document client wrapper
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");

        try {
            System.out.println("Attempting to read the item...");
            GetItemOutcome outcome = table.tgetItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Scared of My Shadow");
```

```
            System.out.println(outcome.getItem());
            System.out.println("GetItem succeeded: " + outcome);
        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }

    }
}
```

DAX 非同步用戶端

AmazonDaxClient 是同步的。針對長時間執行的 DAX API 操作 (例如大型資料表 Scan)，這可能會封鎖程式執行，直到操作完成。如果程式需要在 DAX API 操作進行時執行其他工作，您可以改用 ClusterDaxAsyncClient。

下列程式示範如何使用 ClusterDaxAsyncClient 和 Java Future 來實作非封鎖解決方案。

```
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();
```

```
 }
}
```

使用適用於 Java 1.x 的開發套件查詢全域次要索引

您可以使用 Amazon DynamoDB Accelerator (DAX) 來查詢使用 DynamoDB 程式設計介面的全域次要索引。

下列範例示範如何使用 DAX 查詢在範例：使用文件 API 的全域CreateDateIndex次要索引中建立的全域次要索引。 適用於 Java 的 AWS SDK

DAXClient 類別將與 DynamoDB 程式設計介面互動所需的用戶端物件執行個體化。

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import software.amazon.dynamodb.datamodeling.DynamoDBMapper;
import software.amazon.dynamodb.document.DynamoDB;
import com.amazonaws.util.EC2MetadataUtils;
import software.amazon.dynamodb.AmazonDynamoDB;

public class DaxClient {

 private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

 DynamoDB getDaxDocClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDB(client);
 }

 DynamoDBMapper getDaxMapperClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDBMapper(client);
 }
```

```
}
```

您可以用下列方式查詢全域次要索引：

- 在下列範例中定義的 QueryIndexDax 類別使用 queryIndex 方法。getDaxDocClient 方法在 DaxClient 類別傳回的用戶端物件，由 QueryIndexDax 當做參數。
- 如果您使用物件持久性介面，請在下列範例中定義的 queryIndexMapper 類別使用 QueryIndexDax 方法。在 DaxClient 類別定義之 getDaxMapperClient 方法傳回的用戶端物件，由 queryIndexMapper 當做參數。

```
import java.util.Iterator;
import software.amazon.dynamodb.datamodeling.DynamoDBMapper;
import java.util.List;
import software.amazon.dynamodb.datamodeling.DynamoDBQueryExpression;
import software.amazon.dynamodb.model.AttributeValue;
import java.util.HashMap;
import software.amazon.dynamodb.document.Item;
import software.amazon.dynamodb.document.utils.ValueMap;
import software.amazon.dynamodb.document.spec.QuerySpec;
import software.amazon.dynamodb.document.QueryOutcome;
import software.amazon.dynamodb.document.ItemCollection;
import software.amazon.dynamodb.document.Index;
import software.amazon.dynamodb.document.Table;
import software.amazon.dynamodb.document.DynamoDB;

public class QueryIndexDax {

 //This is used to query Index using the low-level interface.
 public static void queryIndex(DynamoDB client, String tableName, String indexName) {
  Table table = client.getTable(tableName);

  System.out.println("\n*******************************************************
\n");
  System.out.print("Querying index " + indexName + "...");

  Index index = table.getIndex(indexName);

  ItemCollection<QueryOutcome> items = null;

  QuerySpec querySpec = new QuerySpec();
```

```
 if (indexName == "CreateDateIndex") {
  System.out.println("Issues filed on 2013-11-01");
  querySpec.withKeyConditionExpression("CreateDate = :v_date and
begins_with(IssueId, :v_issue)")
    .withValueMap(new ValueMap().withString(":v_date",
"2013-11-01").withString(":v_issue", "A-"));
  items = index.query(querySpec);
 } else {
  System.out.println("\nNo valid index name provided");
  return;
 }

 Iterator<Item> iterator = items.iterator();

 System.out.println("Query: printing results...");

 while (iterator.hasNext()) {
  System.out.println(iterator.next().toJSONPretty());
 }

}

//This is used to query Index using the high-level mapper interface.
public static void queryIndexMapper(DynamoDBMapper mapper, String tableName, String
indexName) {
 HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
 eav.put(":v_date", new AttributeValue().withS("2013-11-01"));
 eav.put(":v_issue", new AttributeValue().withS("A-"));
 DynamoDBQueryExpression<CreateDate> queryExpression = new
DynamoDBQueryExpression<CreateDate>()
   .withIndexName("CreateDateIndex").withConsistentRead(false)
   .withKeyConditionExpression("CreateDate = :v_date and
begins_with(IssueId, :v_issue)")
   .withExpressionAttributeValues(eav);

 List<CreateDate> items = mapper.query(CreateDate.class, queryExpression);
 Iterator<CreateDate> iterator = items.iterator();

 System.out.println("Query: printing results...");

 while (iterator.hasNext()) {
  CreateDate iterObj = iterator.next();
  System.out.println(iterObj.getCreateDate());
  System.out.println(iterObj.getIssueId());
```

```
  }
 }
}
```

以下的類別定義代表問題表，用於 queryIndexMapper 方法。

```
import software.amazon.dynamodb.datamodeling.DynamoDBTable;
import software.amazon.dynamodb.datamodeling.DynamoDBIndexHashKey;
import software.amazon.dynamodb.datamodeling.DynamoDBIndexRangeKey;
import software.amazon.dynamodb.datamodeling.DynamoDBHashKey;

@DynamoDBTable(tableName = "Issues")
public class CreateDate {
 private String createDate;
 @DynamoDBHashKey(attributeName = "IssueId")
 private String issueId;

 @DynamoDBIndexHashKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "CreateDate")
 public String getCreateDate() {
  return createDate;
 }

 public void setCreateDate(String createDate) {
  this.createDate = createDate;
 }

 @DynamoDBIndexRangeKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "IssueId")
 public String getIssueId() {
  return issueId;
 }

 public void setIssueId(String issueId) {
  this.issueId = issueId;
 }
}
```

AWS 適用於 Go 的 SDK 1.x 範例

本節包含使用 Go 1.x 的 DAX 應用程式範例程式碼。

主題

- 適用於 Go 的 DAX 軟體開發套件

適用於 Go 的 DAX 軟體開發套件

請按照此程序操作，在 Amazon EC2 執行個體上執行 Amazon DynamoDB Accelerator (DAX) 適用於 Go 的開發套件範例應用程式。

為 DAX 執行適用於 Go 的開發套件範例

1. 在您的 Amazon EC2 執行個體上設定適用於 Go 的開發套件：

   a. 安裝 Go 程式設計語言 (Golang)。

   ```
   sudo yum install -y golang
   ```

   b. 測試 Golang 是否已安裝並正常運作。

   ```
   go version
   ```

   這類的訊息應會出現。

   ```
   go version go1.15.5 linux/amd64
   ```

   其餘指示仰賴模組支援，成為 Go 版本 1.13 的預設。

2. 安裝範例 Golang 應用程式。

   ```
   go get github.com/aws-samples/aws-dax-go-sample
   ```

3. 執行下列 Golang 程式。第一個程式會建立名為 `TryDaxGoTable` 的 DynamoDB 資料表。第二個程式會將資料寫入資料表。

   ```
   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. 執行下列 Golang 程式。

```
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
```

記下計時資訊：GetItem、Query 和 Scan 測試所需要的毫秒數。

5. 在先前的步驟中，您已針對 DynamoDB 端點執行程式。現在，請再次執行程式，但這一次 GetItem、Query 和 Scan 操作會由您的 DAX 叢集處理。

若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

- 使用 DynamoDB 主控台：選擇您的 DAX 叢集。叢集端點會在主控台上顯示，如以下範例。

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- 使用 AWS CLI ：輸入下列命令。

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

如下列範例所示，叢集端點會在輸出上顯示。

```
{
    "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"
}
```

現在重新執行程式，但這一次，請將叢集端點做為命令列參數指定。

```
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
```

查看輸出的剩餘部分，並記下計時資訊。使用 DAX 的 `GetItem`、`Query` 和 `Scan` 已耗用時間應遠低於使用 DynamoDB 的已耗用時間。

6. 執行以下 Golang 程式，刪除 `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 適用於 Node.js 的 SDK 2.x 範例

本節包含使用適用於 Node.js 2.x 的 AWS SDK 之 DAX 應用程式的範例程式碼。

主題

- Node.js 和 DAX

Node.js 和 DAX

請按照此程序操作，在 Amazon EC2 執行個體上執行 Node.js 範例應用程式。

執行 DAX 的 Node.js 範例

1. 在您的 Amazon EC2 執行個體上設定 Node.js，如下所示：

   a. 安裝節點版本管理工具 (nvm)。

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
```

b. 使用 nvm 安裝 Node.js。

```
nvm install 12.16.3
```

c. 測試 Node.js 已安裝且正常運作。

```
node -e "console.log('Running Node.js ' + process.version)"
```

這應該會顯示以下訊息。

```
Running Node.js v12.16.3
```

2. 使用節點套件管理工具 (npm) 安裝 DAX Node.js 用戶端。

```
npm install amazon-dax-client
```

3. 下載範例程式來源碼 (.zip 檔案)。

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

下載完成後，解壓縮來源檔案。

```
unzip TryDax.zip
```

4. 執行下列 Node.js 程式。第一個程式會建立名為 TryDaxTable 的 Amazon DynamoDB 資料表。第二個程式會將資料寫入資料表。

```
node 01-create-table.js
node 02-write-data.js
```

5. 執行下列 Node.js 程式。

```
node 03-getitem-test.js
node 04-query-test.js
node 05-scan-test.js
```

記下計時資訊：GetItem、Query 和 Scan 測試所需要的毫秒數。

6. 在先前的步驟中，您已針對 DynamoDB 端點執行程式。請再次執行程式，但這一次 `GetItem`、`Query` 和 `Scan` 操作會由您的 DAX 叢集處理。

   若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

   - 使用 DynamoDB 主控台：選擇您的 DAX 叢集。叢集端點會在主控台上顯示，如以下範例。

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - 使用 AWS CLI- 輸入下列命令。

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     如下列範例所示，叢集端點會在輸出上顯示。

     ```
     {
         "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"
     }
     ```

   現在重新執行程式，但這一次，請將叢集端點做為命令列參數指定。

   ```
   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
   ```

   查看輸出的剩餘部分，並記下計時資訊。使用 DAX 的 `GetItem`、`Query` 和 `Scan` 已耗用時間應遠低於使用 DynamoDB 的已耗用時間。

7. 執行以下 Node.js 程式，刪除 `TryDaxTable`。

   ```
   node 06-delete-table
   ```

如需這些程式的詳細資訊，請參閱下列各節：

- 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

`01-create-table.js` 程式會建立資料表 (`TryDaxTable`)。本節剩餘的 Node.js 程式會依存此資料表。

```
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

02-write-data.js 程式會將測試資料寫入 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

`03-getitem-test.js` 程式會在 GetItem 上執行 TryDaxTable 操作。

```
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

`04-query-test.js` 程式會在 Query 上執行 TryDaxTable 操作。

```
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

05-scan-test.js 程式會在 TryDaxTable 上執行 Scan 操作。

```
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

06-delete-table.js 程式會刪除 TryDaxTable。請在完成測試之後執行此程式。

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

DynamoDB 的文件歷史記錄

下表說明 2018 年 7 月 3 日後每個《DynamoDB 開發人員指南》版本的重要變更。如需有關此文件更新的通知，您可以訂閱 RSS 摘要 (位於本頁面左上角)。

變更	描述	日期
DAX 現在 AWS PrivateLink 支援商業和中國區域	新增使用 AWS PrivateLink 搭配 DynamoDB Accelerator (DAX) 的指引，以協助使用者在相同 內建立與 DAX 控制平面端點的私有、安全連線 AWS 區域。如需詳細資訊，請參閱使用 Console-to-Code 為 Amazon DynamoDB 產生基礎結構程式碼。	2025 年 10 月 30 日
DynamoDB 現在支援網際網路通訊協定第 6 版 (IPv6)	當您連線到 DynamoDB 資料表、串流和 DAX 時，DynamoDB 現在支援 Amazon VPC 中的網際網路通訊協定第 6 版 (IPv6) 地址，包括 AWS PrivateLink 閘道和界面端點。您可以使用 IPv6 來簡化網路堆疊，並透過支援 IPv4 和 IPv6 的網路滿足合規要求。	2025 年 10 月 3 日
已新增 DynamoDB 新功能 Console-to-Code 的文件	Console-to-Code 會將 AWS 管理主控台 中的手動 DynamoDB 資料表建立步驟轉換為自動化、可重複的基礎設施程式碼，涵蓋多種格式，例如 AWS CDK 和 CloudFormation。如需詳細資訊，請參閱使用 Console-to-Code 為 Amazon	2025 年 9 月 23 日

	DynamoDB 產生基礎結構程式碼。	
已發行 DynamoDB 本機版 3.1.0 版	此更新可改善 PartiQL 查詢的效能，包括 Joda-time 相依性。	2025 年 9 月 14 日
DynamoDB 已使用新的 CloudWatch Contributor Insights 模式增強限流診斷功能，並新增限流例外狀況詳細資訊，以及目標型 CloudWatch 指標	DynamoDB 在 CloudWatch Contributor Insights 中以全新的僅限 `Throttled keys` 模式加強限流診斷功能，可特別追蹤限流最多的分割區索引鍵、更詳細的限流例外狀況 (包括結構化 `ThrottlingReason` 欄位)、提供特定原因程式碼和資源 ARN，以協助識別遭到限流的確切資源及其原因，以及 16 個對應於不同限流情境的全新目標型 CloudWatch 指標，以精確監控特定限流事件。如需詳細資訊，請參閱對 DynamoDB 中的限流問題進行疑難排解。	2025 年 8 月 14 日
放寬容量模式之間的切換限制	資料表從佈建容量模式切換到隨需容量模式的限制，從每 24 小時一次放寬到每 24 小時四次。對於每天需要多次載入大量資料的工作負載，或是需要因應意外流量增加的工作負載，此項變更可提供更多彈性。如需詳細資訊，請參閱切換 DynamoDB 容量模式時的注意事項。	2025 年 8 月 12 日

已發行 DynamoDB Local 3.0 版，從適用於 Java 的 AWS SDK V1 遷移至 V2	此更新會移除 V1 相依性、改善效能，並與 AWS SDK V2 套件結構保持一致。	2025 年 7 月 17 日
新增對 DynamoDB Streams 中最佳化碎片探索的支援	新的碎片探索機制可減少初始探索時間、提升穩定狀態的處理效能、提供可見度協助掌握處理狀態，以及在大規模操作中維持一致的 p100 延遲。如需詳細資訊，請參閱適用於 Amazon DynamoDB Streams 的AWS PrivateLink。	2025 年 7 月 17 日
新增對 DynamoDB 全域資料表的多區域強一致性 (MRSC) 支援	DynamoDB 現在對具有選用見證區域的全域資料表支援多區域強一致性 (MRSC)。MRSC 可在區域故障期間以零復原點目標 (RPO) 實現跨區域的高度一致性讀取。MRSC 全域資料表必須剛好部署在三個區域中，其中包含三個複本或兩個複本和一個見證。如需了解詳細資訊和可用區域，請參閱多區域強一致性 (MRSC)。	2025 年 6 月 30 日
Amazon DynamoDB Streams Kinesis Client Library (KCL) 2.x 轉接器	DynamoDB 現在支援與 KCL 2.x 和 AWS SDK v2 相容的升級版 DynamoDB Streams 轉接器。如需詳細資訊，請參閱從 KCL1.x 移轉至 KCL 3.x。	2025 年 6 月 11 日
新增支援在 中執行 DynamoDB 本機 AWS CloudShell	您現在可以直接在瀏覽器透過 AWS 管理主控台 執行 DynamoDB 本機版，無需下載或安裝軟體。如需詳細資訊，請參閱在您的電腦本機部署 DynamoDB 本機版。	2025 年 5 月 19 日

新增支援 DAX 中的 r7i 執行個體類型	擴展適用於 DAX 的執行個體類型選項，以提供更多效能和容量彈性。如需新型 r7i 執行個體及其功能的詳細資訊，請參閱靜態 DAX 加密。	2025 年 4 月 29 日
以縮小範圍的新版完整存取政策 AmazonDynamoDBFullAccess_v2 取代 AmazonDynamoDBFullAccess 。	自 2025 年 4 月 25 日起，DynamoDB 已棄用 AmazonDynamoDBFullAccess 政策，並改為使用名為 AmazonDynamoDBFullAccess_v2 的縮小範圍政策。如需詳細資訊，請參閱 AWS 受管政策：AmazonDynamoDBFullAccess_v2。	2025 年 4 月 28 日
DynamoDB 推出對 DynamoDB Streams 的 AWS PrivateLink 支援	AWS 新增對 DynamoDB Streams 的 AWS PrivateLink 支援，讓客戶能夠使用私有網路連線，從 Amazon Virtual Private Cloud (VPC) 內叫用 DynamoDB Streams APIs。此功能可簡化私有網路存取，無需周遊公共網際網路，支援 DynamoDB 工作負載的合規性和安全性要求。如需詳細資訊，請參閱適用於 Amazon DynamoDB Streams 的AWS PrivateLink。	2025 年 3 月 26 日
適用於 JS 3.x 和 Go 2.x 的 DAX SDK 現已推出	適用於 JS 3.x 的 DynamoDB Accelerator (DAX) SDK 現已推出，並與適用於 Java 3.x 的 AWS SDK 相容。	2025 年 3 月 17 日

推出 NoSQL Workbench 3.13.5 預設資料表設定的隨需容量模式	您使用預設設定建立資料表時，DynamoDB 建立的資料表會使用隨需容量模式而非佈建容量模式。	2025 年 2 月 24 日
設定 DAX 用戶端的全新最佳實務	發布用於設定 DAX 用戶端的全新 DAX 規範指引最佳實務主題。如需詳細資訊，請參閱設定 DAX 用戶端。	2025 年 2 月 17 日
大量資料操作的全新最佳實務	發布在 DynamoDB 中使用複雜資料模型的全新最佳實務主題。如需詳細資訊，請參閱在 DynamoDB 中使用大量資料操作的最佳實務。	2025 年 1 月 9 日
Amazon DynamoDB 現在支援可設定的時間點復原 (PITR)	DynamoDB 現在支援可設定的 PITR 復原期間。您現在可將每個資料表的 PITR 期間設定為 1 到 35 天。如需詳細資訊，請參閱 DynamoDB 的時間點備份。	2025 年 1 月 7 日
已發布將 Amazon Managed Streaming for Apache Kafka 與 Amazon DynamoDB 整合的全新主題	了解 Amazon Managed Streaming for Apache Kafka 如何透過從 Apache Kafka 主題讀取資料並將其儲存在 DynamoDB 中，與 Amazon DynamoDB 整合。如需詳細資訊，請參閱整合 DynamoDB 與 Amazon Managed Streaming for Apache Kafka。	2024 年 12 月 26 日

DynamoDB 支援與 SageMaker AI Lakehouse 進行零 ETL 整合	DynamoDB 推出零 ETL 整合，可自動從 Amazon DynamoDB 擷取資料並將其載入到客戶的資料湖。如需詳細資訊，請參閱 DynamoDB 與 SageMaker AI Lakehouse 進行零 ETL 整合。	2024 年 12 月 3 日
DynamoDB 全域資料表現在支援多區域強一致性	多區域強一致性可協助您建置高度可用的多區域應用程式，以零復原點目標 (RPO) 達到最高等級的恢復能力。如需詳細資訊，請參閱具備多區域強一致性的全域資料表。	2024 年 12 月 3 日
DynamoDB 受管政策更新	已將兩個新許可權限新增至 `AmazonDynamoDBReadOnlyAccess` 受管政策：`dynamodb:GetAbacStatus` 和 `dynamodb:UpdateAbacStatus`。這些許可可讓您檢視 ABAC 狀態，並在 AWS 帳戶 目前區域中為 啟用 ABAC。如需詳細資訊，請參閱 AWS 受管政策：AmazonDynamoDBReadOnlyAccess。	2024 年 11 月 18 日

DynamoDB 支援屬性型存取控制 (ABAC)	ABAC 是一種授權策略，可讓您根據連接到使用者、角色 AWS 和資源的標籤來定義存取許可。ABAC 會在您的 AWS Identity and Access Management (IAM) 政策或其他政策中使用標籤型條件，在 IAM 主體的標籤符合資料表的標籤時，允許或拒絕資料表或索引上的特定動作。如需詳細資訊，請參閱使用屬性型存取控制搭配 DynamoDB。	2024 年 11 月 18 日
DynamoDB 為隨需和佈建資料表引入暖輸送量	DynamoDB 現在支援暖輸送量。暖輸送量提供可見度，協助您掌握 DynamoDB 資料表可立即支援的讀取和寫入操作數量，並能預先暖機 DynamoDB 資料表。如需更多詳細資訊，請參閱 DynamoDB 暖輸送量。	2024 年 11 月 13 日
能夠透過 Apache Flink 使用 Amazon DynamoDB Streams 記錄	您現在可以透過 Apache Flink 使用 Amazon DynamoDB Streams 記錄，並利用 Amazon Managed Service for Apache Flink 快速建置和管理端對端串流處理應用程式。如需詳細資訊，請參閱 DynamoDB Streams 和 Apache Flink。	2024 年 11 月 12 日
已發布全域資料表和備份的全新計費主題	已發布兩個有關全域資料表計費和備份計費的全新主題。如需詳細資訊，請參閱了解全域資料表的 Amazon DynamoDB 計費和了解備份的 Amazon DynamoDB 計費。	2024 年 10 月 16 日

Amazon DynamoDB 與 Amazon Redshift 的零 ETL 整合	Amazon DynamoDB 與 Amazon Redshift 的零 ETL 整合提供無程式碼的全受管 ETL 管道，其中包含從 DynamoDB 複寫到 Amazon Redshift。如需詳細資訊，請參閱 Amazon DynamoDB 與 Amazon Redshift 的零 ETL 整合。	2024 年 10 月 15 日
新增使用生成式 AI 搭配 DynamoDB 的主題 (僅文件更新)	發布全新主題，提供使用生成式 AI 搭配 DynamoDB 的相關資訊，包括 DynamoDB 的生成式 AI 使用案例範例。如需詳細資訊，請參閱使用生成式 AI 搭配 DynamoDB。	2024 年 10 月 11 日
SDK 現在支援以 AWS 帳戶為基礎的端點	已新增帳戶型端點的文件，以及 SDK 用戶端的 `ACCOUNT_ID_ENDPOINT_MODE` 設定。如需詳細資訊，請參閱AWS 帳戶型端點的 SDK 支援。	2024 年 9 月 3 日
重新設計入門體驗	重新設計入門體驗以整合資訊，並協助您更快入門。如需詳細資訊，請參閱 DynamoDB 入門。	2024 年 8 月 1 日
DAX 擴展到西班牙和瑞典的全新區域	DAX 現已在西班牙和瑞典區域提供使用。如需詳細資訊，請參閱 DAX 叢集元件。	2024 年 7 月 30 日
重組及合併 DynamoDB 備份與還原文件	《DynamoDB 開發人員指南》針對備份和還原採用全新結構。如需詳細資訊，請參閱 DynamoDB 的備份與還原。	2024 年 7 月 2 日

重新編寫「什麼是 Amazon DynamoDB？」主題	已發布「什麼是 Amazon DynamoDB？」主題的修訂和更新版本。如需詳細資訊，請參閱什麼是 Amazon DynamoDB？	2024 年 6 月 21 日
將 DynamoDB Streams 與 EventBridge 整合	已發布將 DynamoDB Streams 與 EventBridge 整合的全新主題。如需詳細資訊，請參閱與 EventBridge 整合。	2024 年 6 月 21 日
DAX 規範指引	已發布新的最佳實務主題，提供您有效使用 DynamoDB Accelerator 的完整洞見。本主題涵蓋效能最佳化、成本管理和操作最佳實務。如需詳細資訊，請參閱 DAX 規範指引。	2024 年 6 月 3 日
將 DynamoDB 資料表從一個帳戶移轉至另一帳戶	新增將 DynamoDB 資料表從一個帳戶移轉到另一個帳戶的全新主題。如需詳細資訊，請參閱將 DynamoDB 資料表從一個帳戶移轉至另一帳戶。	2024 年 5 月 29 日
重組及合併 DynamoDB 監控和日誌記錄文件	DynamoDB 中有關監控和日誌記錄的全新結構包含三個簡潔章節，分別介紹指標、日誌記錄操作和 Contributor Insights。	2024 年 5 月 3 日

重組及合併 DynamoDB 容量模式文件	DynamoDB 指南現在包含新章節，其中包含有關隨需和佈建 DynamoDB 容量模式的所有資訊。在這次更新的文件中，變更讀取/寫入容量模式的注意事項主題已移至最佳實務章節。此項主題目前已更名為切換容量模式時的注意事項，並包含切換容量模式時有關最佳實務的詳細資訊。此外指南現在也提供新章節，包含有關 DynamoDB 讀取和寫入的所有資訊，以及讀取和寫入操作的容量單位消耗。如需詳細資訊，請參閱 DynamoDB 輸送容量、切換容量模式時的注意事項，以及 DynamoDB 讀取和寫入。	2024 年 5 月 1 日
隨需請求的最大數量	您現在可以指定個別資料表及/或索引可執行的隨需請求數目上限。指定最大隨需輸送量有助於保持資料表層級的用量和成本限制，並防止耗用資源意外激增。如需詳細資訊，請參閱隨需資料表的最大輸送量。	2024 年 5 月 1 日
NoSQL Workbench 操作建置器改進	NoSQL Workbench 現在包含對深色模式的原生支援。改善操作建置器中的資料表和項目操作。項目結果和操作建置器請求資訊以 JSON 格式提供。如需詳細資訊，請參閱 NoSQL Workbench 操作建置器。	2024 年 4 月 24 日

Amazon DynamoDB 資源的資源型政策	DynamoDB 現在支援資料表、索引和串流的資源型政策。資源型政策可讓您指定有權存取各項資源的人員，以及允許前述人員對各項資源執行的動作，藉此定義存取許可權限。如需詳細資訊，請參閱使用 DynamoDB 的資源型政策。	2024 年 3 月 20 日
DynamoDB 受管政策更新	將新的許可權限 `dynamodb:GetResourcePolicy` 新增到 `AmazonDynamoDBReadOnlyAccess` 受管政策。此權限可讓您讀取附加於 DynamoDB 資源的資源型政策。如需詳細資訊，請參閱 AWS 受管政策：AmazonDynamoDBReadOnlyAccess。	2024 年 3 月 20 日
AWS PrivateLink 適用於 Amazon DynamoDB	Amazon DynamoDB 現在支援 AWS PrivateLink。透過 AWS PrivateLink，您可以使用介面 VPC 端點和私有 IP 地址，簡化虛擬私有雲端 (VPCs)、DynamoDB 和內部部署資料中心之間的私有網路連線。如需詳細資訊，請參閱適用於 DynamoDB 的AWS PrivateLink。	2024 年 3 月 19 日

使用 JavaScript 編寫程式指南	Amazon DynamoDB 提供程式設計指南 適用於 JavaScript 的 AWS SDK。了解 適用於 JavaScript 的 AWS SDK、抽象層、設定連線、處理錯誤、定義重試政策、管理保持連線及其他項目。如需詳細資訊，請參閱使用 JavaScript 編寫程式。	2024 年 3 月 6 日
使用 AWS SDK for Java 2.x 指南進行程式設計	建立新的程式設計指南，深入介紹高階、低階和文件介面、HTTP 用戶端及其組態、錯誤處理，並說明使用適用於 Java 的 SDK 2.x 時應考慮的最常見組態設定。如需詳細資訊，請參閱使用程式設計 Amazon DynamoDB AWS SDK for Java 2.x。	2024 年 3 月 5 日
使用 NoSQL Workbench 複製資料表	允許開發人員使用 NoSQL Workbench 在開發環境和區域 (DynamoDB 本機版和 DynamoDB Web) 之間複製資料表。如需詳細資訊，請參閱使用 NoSQL Workbench 複製資料表。	2024 年 2 月 26 日
使用 Python 編寫程式指南	建立新的指南，深入介紹高階和低階函式庫，並說明使用 Python SDK 時應考量的最常見組態設定。如需更多資訊，請參閱使用 Python 編寫程式。	2024 年 1 月 5 日

生存時間 (TTL) 主題重寫	完全重寫本指南的 TTL 章節。新指南在過程中提供就緒使用的程式碼片段，協助您開始使用 TTL。目前提供 Python 和 Javascript 的程式碼片段。如需詳細資訊，請參閱 TTL。	2023 年 12 月 20 日
了解 AWS 帳單和用量報告的最佳實務	新增章節以釐清 DynamoDB 中的各種用量類型，以及前述用量類型的費用。如需詳細資訊，請參閱計費與用量報告。	2023 年 12 月 15 日
Amazon DynamoDB 與 Amazon OpenSearch Service 的零 ETL 整合	Amazon DynamoDB 現在支援與 Amazon OpenSearch Service 進行零 ETL 整合，您無需使用自訂程式碼或基礎結構，即可自動複寫和轉換 DynamoDB 資料，以便搜尋 DynamoDB 資料。如需詳細資訊，請參閱 DynamoDB 與 Amazon OpenSearch Service 的零 ETL 整合。	2023 年 11 月 28 日
從關聯式資料庫移轉到 DynamoDB	建立移轉指南協助使用者了解如何從關聯式資料庫移轉至 DynamoDB。	2023 年 11 月 27 日
使用 NoSQL Workbench 產生範例資料	Amazon DynamoDB 專用 NoSQL Workbench 現在支援直接從範例資料模型範本建立資料模型，以協助您為工作負載設計資料結構描述。在 DynamoDB 上建置應用程式時，您可以使用此功能熟悉 NoSQL 資料建模最佳實務。	2023 年 9 月 28 日

增量匯出至 S3	您現在可以匯出以小幅增量插入、更新或刪除的資料。透過增量匯出，您可以在 AWS 管理主控台、API 呼叫或 AWS 命令列界面中按幾下滑鼠，將變更的資料從數 MB 匯出到 TB。	2023 年 9 月 26 日
適用於 DynamoDB 的資料建模	您現在可以使用 DynamoDB 範例進一步了解資料建模，這些範例著重於特定使用案例、其存取模式，以及實現這些存取模式的逐步指引。	2023 年 7 月 14 日
疑難排解一節	您現在可以找到 DynamoDB 資料表中可能發生的延遲和限流問題的疑難排解內容。	2023 年 3 月 13 日
Amazon DynamoDB 的刪除保護	現在，所有 AWS 區域的 Amazon DynamoDB 資料表都可使用刪除保護功能。DynamoDB 現在可讓您在執行常態資料表管理操作時，保護資料表免遭意外刪除。	2023 年 3 月 8 日
CloudFormation 支援全域資料表中的 KDSD	DynamoDB 專用 Amazon Kinesis Data Streams 現在支援 CloudFormation DynamoDB 全域資料表，這表示您可以使用 CloudFormation 範本啟用串流至 DynamoDB 全域資料表上的 Amazon Kinesis Data Streams。	2023 年 2 月 15 日
DynamoDB 本機版支援每個交易 100 個動作	您現在可以在 DynamoDB 本機版的單一交易中執行多達 100 個動作。	2023 年 2 月 9 日

使用 DynamoDB Well-Architected Lens 來最佳化您的 DynamoDB 工作負載	您現在可以使用 DynamoDB Well-Architected Lens，其收集了用於設計架構良好 DynamoDB 工作負載的設計原則和指引。	2023 年 2 月 3 日
PartiQL GovGloud 可用性	PartiQL - GovCloud （美國東部） 和 GovCloud （美國西部） 現在支援 Amazon DynamoDB 的 SQL 相容查詢語言。 AWS GovCloud AWS GovCloud	2022 年 12 月 21 日
NoSQL Workbench 和 DynamoDB 本機版的單一安裝套件	DynamoDB 專用 NoSQL Workbench 現在包含引導式 DynamoDB 本機版安裝程序，可簡化 DynamoDB 本機版開發環境的設定。	2022 年 12 月 6 日
從 S3 大量匯入	Amazon DynamoDB 現在支援從 Amazon S3 大量匯入資料，讓您可以更輕鬆地將資料移轉並載入新的 DynamoDB 資料表。	2022 年 8 月 18 日
增強與 Service Quotas 的整合	Service Quotas 現在可讓您主動管理您的帳戶和資料表配額。您可以檢視目前的值、設定配額使用率超過可設定閾值時發出警示等。	2022 年 6 月 15 日
NoSQL Workbench 新增資料表和 GSI 支援	您現在可以將 NoSQL Workbench 用於資料表和全域次要索引 (GSI) 控制平面操作，例如 CreateTable、UpdateTable 和 DeleteTable。	2022 年 6 月 2 日

標準的不常存取資料表類別現已在中國推出	Amazon DynamoDB 標準：不常存取資料表類別在中國地區推出。針對儲存不常存取的資料的資料表使用此全新資料表類別，藉此降低高達 60% 的 DynamoDB 成本。	2022 年 4 月 18 日
增加預設服務配額和資料表管理操作	DynamoDB 增加了每個帳戶和區域的資料表數目預設配額，從 256 個增加到 2,500 個資料表，並將並行資料表管理操作的數量從 50 個增加到 500 個。	2022 年 3 月 9 日
適用於 DynamoDB 的 PartiQL 的項目限制 (選用)	DynamoDB 可在每個請求中，透過選用參數，針對 DynamoDB 操作限制 PartiQL 中處理的項目數。	2022 年 3 月 8 日
AWS Backup 整合適用於中國（北京和寧夏）區域	在中國 (北京與寧夏) 區域，AWS Backup 現已與 DynamoDB 整合。透過 中的增強備份功能，例如跨帳戶和跨區域備份 AWS Backup，您可以更輕鬆地滿足合規和業務連續性要求。	2022 年 1 月 26 日
透過 PartiQL API 呼叫的輸送容量資訊	DynamoDB 可以傳回 PartiQL API 呼叫所耗用的輸送容量資訊，以協助您最佳化查詢和輸送量成本。	2022 年 1 月 18 日
AWS Backup 整合	DynamoDB 現在可透過 AWS Backup 中的強化備份功能 (例如跨帳戶和跨區域備份)，協助您更容易符合合規和商業持續性要求。	2021 年 11 月 24 日

CSV 格式的 NoSQL Workbench 匯入/匯出資料集	Amazon DynamoDB 專用 NoSQL Workbench 現在可以匯入和自動填入範例資料，以協助建置和視覺化資料模型。	2021 年 10 月 11 日
使用 篩選和擷取 Amazon DynamoDB Streams 資料平面活動 AWS CloudTrail	Amazon DynamoDB 現在為您提供了更精細的稽核日誌記錄控制，方法是讓您能夠在 AWS CloudTrail中篩選 Streams 資料平面 API 活動。	2021 年 9 月 22 日
已更新主控台	現在 DynamoDB 主控台是您的預設主控台，可協助您更輕鬆地管理資料、簡化常見任務，並讓您更快地存取資源和功能。	2021 年 8 月 25 日
適用於 Java 2.x 的 DAX SDK 現已推出	適用於 Java 的 DynamoDB Accelerator (DAX) SDK 2.x 現已推出，並與適用於 Java 的 AWS SDK 2.x 相容。您可以從最新功能中受益，包括非封鎖輸入/輸出 (I/O)。	2021 年 7 月 29 日
NoSQL Workbench 功能更新，包括控制平面操作	Amazon DynamoDB 專用 NoSQL Workbench 現在可以協助您更輕鬆地執行頻繁的操作以修改和存取資料表資料。	2021 年 7 月 28 日
DynamoDB 全域資料表現已在亞太地區提供	DynamoDB 全域資料表現已在亞太地區 (大阪) 提供。跨越您選擇的 22 個 AWS 區域自動複寫 DynamoDB 資料表。	2021 年 7 月 28 日
DAX 現已在中國提供	DynamoDB Accelerator (DAX) 現已在中國 (北京) 區域提供，並由 Sinnet 營運。	2021 年 7 月 28 日

DAX 傳輸中加密	DynamoDB Accelerator (DAX) 現在支援在應用程式和 DAX 叢集之間以及 DAX 叢集內的節點之間傳輸資料時進行加密。	2021 年 7 月 24 日
CloudFormation 和 CloudTrail 整合	整合 AWS CloudFormation 以及 CloudFormation 資料平面日誌記錄的安全性強化。	2021 年 6 月 18 日
全域資料表現在支援 CloudFormation	Amazon DynamoDB 全域資料表現在支援 AWS CloudFormation，這意味著您可以使用 CloudFormation 範本建立全域資料表並管理其設定。	2021 年 5 月 14 日
適用於 Java 2.x 的 Amazon DynamoDB 本機版支援	現在您可以使用適用於 Java 2.x 的AWS SDK 搭配 Amazon DynamoDB 的可下載版本 DynamoDB 本機版。藉助 DynamoDB Local，您可以使用在本地開發環境中執行的 DynamoDB 版本來開發和測試應用程式，而不會產生任何額外成本。	2021 年 5 月 3 日
NoSQL Workbench 現在支援 AWS CloudFormation	Amazon DynamoDB 專用 NoSQL Workbench 現在支援 AWS CloudFormation，因此您可以使用 CloudFormation 範本管理和修改 DynamoDB 資料模型。此外，您現在可以在 NoSQL Workbench 中配置資料表容量設定。	2021 年 4 月 22 日
DynamoDB 和 AWS Amplify 現在的功能整合	AWS Amplify 現在可以在單個部署中協調多個 DynamoDB 全域次要索引更新。	2021 年 4 月 20 日

AWS CloudTrail 記錄 Amazon DynamoDB Streams 資料平面 APIs	您現在可以使用 AWS CloudTrail 記錄 Amazon DynamoDB Streams 資料平面 API 活動，並監視和調查 DynamoDB 資料表中的項目層級變更。	2021 年 4 月 20 日
適用於 Amazon DynamoDB 的 Amazon Kinesis Data Streams 現在支援 DynamoDB AWS CloudFormation	適用於 Amazon DynamoDB 的 Amazon Kinesis Data Streams DynamoDB現在支援 AWS CloudFormation，這表示您可以使用 CloudFormation 範本啟用串流至 DynamoDB 資料表上的 Amazon Kinesis 資料串流。藉由將 DynamoDB 資料變更串流至 Kinesis 資料串流，您可以使用 Amazon Kinesis 服務建置進階串流應用程式。	2021 年 4 月 12 日
Amazon Keyspaces 現在提供符合 FIPS 140-2 標準的端點	Amazon Keyspaces (適用於 Apache Cassandra) 現已提供符合聯邦資訊處理標準 (FIPS) 140-2 的端點，可幫助您更輕鬆地執行高度監管的工作負載。FIPS 140-2 是美國和加拿大政府標準，指定密碼模組的安全要求以保護敏感資訊。	2021 年 4 月 8 日
適用於 DAX 的 Amazon EC2 T3 執行個體	DAX 目前支援 Amazon EC2 T3 執行個體類型，此類執行個體可提供基本水準的 CPU 效能，並可在需要時大幅突破基本水準。	2021 年 2 月 15 日

Amazon DynamoDB 專用 NoSQL Workbench 支援 PartiQL	您現在可以使用 DynamoDB 專用 NoSQL Workbench 為 DynamoDB 建置 PartiQL 陳述式。	2020 年 12 月 4 日
DynamoDB 專用 PartiQL	您現在可以使用 DynamoDB 的 PartiQL，這是一種與 SQL 相容的查詢語言，與 DynamoDB 資料表互動 AWS Command Line Interface，並使用 AWS 管理主控台和 PartiQL 的 DynamoDB API 來執行臨機操作查詢。 APIs	2020 年 11 月 23 日
Amazon DynamoDB 專用 Amazon Kinesis Data Streams	您現在可以將 Amazon DynamoDB 專用 Amazon Kinesis Data Streams 與 DynamoDB 資料表搭配使用，以便擷取項目層級的變更，並將變更複寫至 Kinesis 資料串流。	2020 年 11 月 23 日
DynamoDB 資料表匯出	您現在可以將 DynamoDB 資料表匯出至 Amazon S3，讓您能夠使用 Athena AWS Glue、 和 Lake Formation 等服務對資料執行分析和複雜的查詢。	2020 年 11 月 9 日

空值支援	DynamoDB 現在支援 DynamoDB 資料表中的非索引鍵字串和二進制屬性的空值。空值支援功能讓您能夠更靈活地將屬性運用於更廣泛的使用案例，無需在傳送至 DynamoDB 之前轉換這類屬性。清單、映射及設定資料類型也支援空白字串和二進位數值。	2020 年 5 月 18 日
Amazon DynamoDB 專用 NoSQL Workbench 支援 Linux	Amazon DynamoDB 專用 NoSQL Workbench 現在也支援 Linux Ubuntu、Fedora 及 Debian。	2020 年 5 月 4 日
DynamoDB 專用 CloudWatch Contributor Insights：GA 版	DynamoDB 專用 CloudWatch Contributor Insights 現已正式推出。DynamoDB 專用 CloudWatch Contributor Insights 是一項診斷工具，可讓您快速了解 DynamoDB 資料表的流量趨勢，並協助您識別資料表中最常存取的金鑰 (也稱為熱鍵)。	2020 年 4 月 2 日
升級全域資料表	現在，您只要在 DynamoDB 主控台中按幾下滑鼠，即可將全域資料表從 2017 年 11 月 29 日版本更新至全域資料表的最新版本 (2019 年 11 月 21 日)。透過升級全域資料表的版本，您可以將現有資料表延伸至其他 AWS 區域，輕鬆提高 DynamoDB 資料表的可用性，而不需要重建資料表。	2020 年 3 月 16 日

Amazon DynamoDB 專用 NoSQL Workbench：GA 版	Amazon DynamoDB 專用 NoSQL Workbench 現已正式推出。使用 NoSQL Workbench 設計、建立、查詢及管理 DynamoDB 資料表。	2020 年 3 月 2 日
DAX 快取叢集指標	DAX 支援新的 CloudWatch 指標，可讓您進一步了解 DAX 叢集的效能。	2020 年 2 月 6 日
DynamoDB 專用 CloudWatch Contributor Insights：預覽版	DynamoDB 專用 CloudWatch Contributor Insights 是一項診斷工具，可讓您快速了解 DynamoDB 資料表的流量趨勢，並協助您識別資料表中最常存取的金鑰 (也稱為熱鍵)。	2019 年 11 月 26 日
支援不平衡工作負載的調適型容量	Amazon DynamoDB 的自適應容量功能現在可藉由自動隔離經常存取的項目，來更妥善地處理不平衡的工作負載。如果應用程式會導致一或多個項目的流量過大，DynamoDB 則會重新平衡您的分割區，讓經常存取的項目不會駐留在同一分割區上。	2019 年 11 月 26 日
支援客戶自管金鑰	DynamoDB 現在支援客戶自管金鑰，這代表您可以完全控制如何加密與管理 DynamoDB 資料的安全性。	2019 年 11 月 25 日
支援 DynamoDB 本機版 (可下載版本) 的 NoSQL Workbench	NoSQL Workbench 現在支援連接至 DynamoDB 本機版 (可下載版本)，以設計、建立、查詢和管理 DynamoDB 資料表。	2019 年 11 月 8 日

NoSQL Workbench：預覽版	這是 DynamoDB 專用 NoSQL Workbench 的初始版本。使用 NoSQL Workbench 設計、建立、查詢及管理 DynamoDB 資料表。如需詳細資訊，請參閱 Amazon DynamoDB 專用 NoSQL Workbench (預覽版)。	2019 年 9 月 16 日
DAX 新增支援使用 Python 和 NET 的交易操作	DAX 支援以 Go、Java、NET、Node.js 和 Python 編寫之應用程式的 `TransactWriteItems` 與 `TransactGetItems` API。如需詳細資訊，請參閱使用 DAX 進行記憶體內部加速。	2019 年 2 月 14 日
Amazon DynamoDB 本機版 (可下載版本) 更新	DynamoDB 本機版 (可下載版本) 現在支援交易 API、隨需讀取/寫入容量、讀取和寫入操作容量報告，以及 20 個全域次要索引。如需詳細資訊，請參閱可下載的 DynamoDB 和 DynamoDB Web 服務之間的差異。	2019 年 2 月 4 日
Amazon DynamoDB 隨需功能	DynamoDB 隨需功能是彈性的計費選項，可在沒有容量規劃的情況下，每秒處理數千筆請求。DynamoDB 隨需提供讀取和寫入請求pay-per-request定價，因此您只需按使用量付費。如需詳細資訊，請參閱 DynamoDB 輸送容量。	2018 年 11 月 28 日

Amazon DynamoDB Transactions	DynamoDB 交易會同時對資料表內和跨資料表的多個項目，進行統籌協調式的「全有或全無」變更，在 DynamoDB 中提供原子性、一致性、隔離性和持久性 (ACID)。如需詳細資訊，請參閱 Amazon DynamoDB Transactions。	2018 年 11 月 27 日
Amazon DynamoDB 會將客戶的所有靜態資料加密	每當資料儲存到持久性的媒體時，DynamoDB 的靜態加密功能，就會透過保護加密資料表中的資料 (包括其主索引鍵、本機及全域次要索引、串流、全域資料表、備份和 DAX 叢集)，來提供多一層的資料保護機制。如需詳細資訊，請參閱 Amazon DynamoDB 靜態加密。	2018 年 11 月 15 日
運用全新的 Docker 映像檔來更輕鬆地使用 Amazon DynamoDB 本機版	現在，使用 DynamoDB 本機版 (也就是可下載的 DynamoDB 版本) 變得更加容易，可讓您使用全新的 DynamoDB 本機 Docker 映像檔，來開發和測試 DynamoDB 應用程式。如需詳細資訊，請參閱 DynamoDB (可下載版本) 和 Docker。	2018 年 8 月 22 日
DynamoDB Accelerator (DAX) 新增對靜態加密的支援	DynamoDB Accelerator (DAX) 現在支援新 DAX 叢集的靜態加密功能，可協助您在注重安全的應用中，加快從 Amazon DynamoDB 資料表讀取資料的作業，這類應用必須配合嚴格的合規與法規要求。如需詳細資訊，請參閱 DAX 靜態加密。	2018 年 8 月 9 日

DynamoDB 時間點復原 (PITR) 新增對於還原已刪除資料表的支援	若您在啟用時間點復原功能的狀態下刪除資料表，系統備份會自動建立並保留 35 天 (無需額外費用)。如需詳細資訊，請參閱在您開始使用時間點復原之前。	2018 年 8 月 7 日
現在可以透過 RSS 獲得更新	您現在可以訂閱 RSS 訊息 (在此頁面的左上角)，以接收《Amazon DynamoDB 開發人員指南》的更新通知。	2018 年 7 月 3 日

舊版更新

下表說明 2018 年 7 月 3 日前《DynamoDB 開發人員指南》的重要變更。

變更	描述	變更日期
取得 DAX 支援	您現在能夠對以 Go 程式語言撰寫之應用程式中的 Amazon DynamoDB 資料表啟用微秒讀取效能，方法是使用新 DynamoDB Accelerator (DAX) SDK for Go。如需詳細資訊，請參閱 適用於 Go 的 DAX 軟體開發套件。	2018 年 6 月 26 日
DynamoDB 發行 SLA	DynamoDB 已發行公有可用性 SLA。如需詳細資訊，請參閱 Amazon DynamoDB 服務水準協議。	2018 年 6 月 19 日
DynamoDB 持續備份和時間點復原 (PITR)	時間點復原有助於保護您的 Amazon DynamoDB 資料表免遭意外寫入或刪除操作。有了時間點復原，就無需為建立、維護或排程隨需備份而煩惱。	2018 年 4 月 25 日

變更	描述	變更日期
	例如，假設測試指令碼意外寫入至生產 DynamoDB 資料表。透過時間點復原，您可以將該資料表還原到過去 35 天內的任何時間點。DynamoDB 維護您資料表的增量備份。如需詳細資訊，請參閱DynamoDB 的時間點備份。	
適用於 DynamoDB 的靜態加密	新 DynamoDB 資料表可用的 DynamoDB 靜態加密可透過使用在 AWS Key Management Service中存放之 AWS受管的加密金鑰，來協助確保 Amazon DynamoDB 資料表中應用程式資料的安全性。如需詳細資訊，請參閱DynamoDB 靜態加密。	2018 年 2 月 8 日
DynamoDB 備份與還原	隨需備份可讓您建立 DynamoDB 資料表的完整備份以進行資料封存，協助您符合公司與政府的法規要求。您可以備份資料量從幾 MB 到數百 TB 不等的資料表，而且不會影響您生產應用程式的效能與可用性。如需詳細資訊，請參閱DynamoDB 的備份與還原。	2017 年 11 月 29 日

變更	描述	變更日期
DynamoDB 全域資料表	Global Tables 建立於 DynamoDB 全域佈局的基礎之上，提供您全受管、多區域以及多個作用中的資料庫，為大幅度擴展的全域應用程式提供快速、本機、讀取與寫的效能。Global Tables 會自動跨您選擇的 AWS 區域複寫 Amazon DynamoDB 資料表。如需詳細資訊，請參閱全域資料表：多重作用中的多區域複寫。	2017 年 11 月 29 日
DAX 的 Node.js 支援	Node.js 開發人員可以使用適用於 Node.js 的 DAX 用戶端，來利用 DynamoDB Accelerator (DAX)。如需詳細資訊，請參閱 使用 DynamoDB Accelerator (DAX) 的記憶體內加速。	2017 年 10 月 5 日
DynamoDB 的 VPC 端點	DynamoDB 端點可讓您 Amazon VPC 中的 Amazon EC2 執行個體存取 DynamoDB，而不需暴露於公有網際網路中。您的 VPC 與 DynamoDB 之間的網路流量都會在 Amazon 網路的範圍內。如需詳細資訊，請參閱 使用 Amazon VPC 端點來存取 DynamoDB。	2017 年 8 月 16 日

變更	描述	變更日期
DynamoDB Auto Scaling 功能	DynamoDB Auto Scaling 功能讓您不必手動定義或調整佈建輸送量設定。相反地，DynamoDB Auto Scaling 功能會根據實際的流量模式，動態調整讀寫容量。這可讓資料表或全域次要索引增加其佈建的讀取與寫入容量，以處理突然增加的流量，而不需限流。當工作負載降低時，DynamoDB Auto Scaling 功能就會降低佈建的容量。如需詳細資訊，請參閱 使用 DynamoDB Auto Scaling 功能自動管理輸送容量。	2017 年 6 月 14 日
DynamoDB Accelerator (DAX)	DynamoDB Accelerator (DAX) 是適用於 DynamoDB 的全受管、高可用性記憶體快取，即使每秒數百萬個請求也能將時間從毫秒縮短到微秒，提供高達 10 倍的效能改進。如需詳細資訊，請參閱 使用 DynamoDB Accelerator (DAX) 的記憶體內加速。	2017 年 4 月 19 日
DynamoDB 現在支援使用存留時間 (TTL) 的項目自動過期	Amazon DynamoDB 存留時間 (TTL) 讓您可自動刪除資料表的過期項目，無額外成本。如需詳細資訊，請參閱 使用 DynamoDB 的存留時間 (TTL) 功能。	2017 年 2 月 27 日

變更	描述	變更日期
DynamoDB 現在支援成本分配標籤	您現在可以對您的 Amazon DynamoDB 資料表新增標籤，以改善用量分類及讓成本報告更精細。如需詳細資訊，請參閱 將標籤新增至 DynamoDB 中的資源。	2017 年 1 月 19 日
新 DynamoDB `DescribeLimits` API	`DescribeLimits` API 會傳回您 AWS 帳戶在區域中的目前佈建容量限制，包括整個區域，以及您在該區域中建立的任何一個 DynamoDB 資料表。它可讓您判斷目前的帳戶等級限制，以便與目前使用的佈建容量比較，讓您在達到限制前有充分的時間申請提高限制容量。如需詳細資訊，請參閱《Amazon DynamoDB API 參考》中的 Amazon DynamoDB 中的配額 和 DescribeLimits。	2016 年 3 月 1 日

變更	描述	變更日期
DynamoDB 主控台更新與新的主索引鍵屬性術語	DynamoDB 管理主控台經過重新設計，更直觀、更易於操作。在這次更新中，我們也會推出新的主索引鍵屬性術語： • Partition Key (分割區索引鍵)：也稱為雜湊屬性。 • Sort Key (排序索引鍵)：也稱為範圍屬性。 唯一變更的只有名稱，功能仍保持不變。 當您建立資料表或次要索引時，可以選擇簡易主索引鍵 (僅分割區索引鍵) 或複合主鍵 (分割區索引鍵和排序索引鍵)。DynamoDB 文件已更新以反映這些變更。	2015 年 11 月 12 日

變更	描述	變更日期
適用於 Titan 的 Amazon DynamoDB 儲存後端	適用於 Titan 的 DynamoDB 儲存後端是在 Amazon DynamoDB 上層實作的 Titan 圖形資料庫儲存後端。使用適用於 Titan 的 DynamoDB 儲存後端時，您的資料受 DynamoDB 所保護，該服務在所有 Amazon 具高可用性的資料中心上執行。該外掛程式可供 Titan 0.4.4 版 (主要針對與現有應用程式的相容性) 和 Titan 0.5.4 版 (新的應用程式建議使用) 使用。如同適用於 Titan 的其他儲存後端，此外掛程式也支援 Tinkerpop 堆疊 (2.4 和 2.5 版)，包括 Blueprints API 和 Gremlin shell。	2015 年 8 月 20 日

變更	描述	變更日期
DynamoDB Streams、跨區域複寫和使用高度一致性讀取的掃描	DynamoDB Streams 可擷取任何 DynamoDB 資料表中依時間順序排序的項目層級修改，並將此資訊存放於日誌中長達 24 小時。應用程式可以存取此日誌，並近乎即時地檢視資料項目修改前後的顯示內容。如需詳細資訊，請參閱 DynamoDB Streams 的變更資料擷取 及《DynamoDB Streams API 參考》。 DynamoDB 跨區域複寫是一種用戶端解決方案，可近乎即時地維護不同 AWS 區域的相同 DynamoDB 資料表複本。您可以使用跨區域複寫備份 DynamoDB 資料表，或在使用者分散於不同地理位置的情況下提供資料的低延遲存取。 根據預設，DynamoDB `Scan` 操作會使用最終一致讀取。您可以將 `ConsistentRead` 參數設為 true，改用高度一致性讀取。如需詳細資訊，請參閱《Amazon DynamoDB API 參考》中的 掃描的讀取一致性 及掃描。	2015 年 7 月 16 日

變更	描述	變更日期
AWS CloudTrail 支援 Amazon DynamoDB	DynamoDB 現與 CloudTrail 整合。CloudTrail 會擷取從 DynamoDB 主控台或 DynamoDB API 發出的 API 呼叫，並在日誌檔案中加以追蹤。如需詳細資訊，請參閱 使用 記錄 DynamoDB 操作 AWS CloudTrail 及《AWS CloudTrail 使用者指南》。	2015 年 5 月 28 日
改善 Query 表達式的支援	此版本對 `KeyConditionExpression` API 新增了 `Query` 參數。`Query` 會使用主索引鍵值從資料表或索引讀取項目。`KeyConditionExpression` 參數是識別主索引鍵名稱的字串，也是要套用到索引鍵值的條件；`Query` 只會擷取那些符合表達式的項目。`KeyConditionExpression` 的語法與 DynamoDB 中其他表達式參數的語法類似，可讓您在表達式中定義名稱和值的替換變數。如需詳細資訊，請參閱 在 DynamoDB 中查詢資料表。	2015 年 4 月 27 日

變更	描述	變更日期
條件式寫入的新比較函數	在 DynamoDB 中，`ConditionExpression` 參數可決定 `PutItem`、`UpdateItem` 或 `DeleteItem` 是否成功：只有當條件評估為 true 時才會寫入項目。此版本新增了兩個函數 `attribute_type` 和 `size`，配合 `ConditionExpression` 一起使用。這些函數可讓您根據資料表中屬性的資料類型或大小，執行條件式寫入。如需詳細資訊，請參閱 DynamoDB 條件表達式 CLI 範例。	2015 年 4 月 27 日
次要索引的掃描 API	在 DynamoDB 中，`Scan` 操作會讀取資料表中的所有項目，套用使用者定義的篩選條件，再將所選資料項目傳回給應用程式。這項功能現在同樣可用於次要索引。若要掃描本機次要索引或全域次要索引，您要指定索引名稱及其父資料表的名稱。根據預設，索引 `Scan` 會傳回索引中的所有資料，您可使用篩選條件表達式縮減傳回給應用程式的結果。如需詳細資訊，請參閱 掃描 DynamoDB 中的資料表。	2015 年 2 月 10 日

變更	描述	變更日期
全域次要索引的線上操作	線上索引可讓您對現有資料表新增或移除全域次要索引。使用線上索引時，您不需要在建立資料表時定義資料表的所有索引，而可隨時新增索引。同樣地，如果您認為不再需要某個索引，就可以隨時予以移除。線上索引不是阻擋式操作，在索引新增或移除時，仍可讀取和寫入資料表。如需詳細資訊，請參閱 在 DynamoDB 中管理全域次要索引。	2015 年 1 月 27 日
JSON 的文件模型支援	DynamoDB 可讓您使用完整的文件模型支援，存放及擷取文件。新的資料類型與 JSON 標準完全相容，可讓您將文件元素彼此巢狀嵌入。您可以使用文件路徑取消參考運算子讀取及寫入個別元素，而不必擷取整份文件。此版本也會推出新的表達式參數，用以在讀取或寫入資料項目時指定投影、條件和更新動作。若要進一步了解 JSON 文件模型支援，請參閱 資料類型 和 在 DynamoDB 中使用表達式。	2014 年 10 月 7 日
彈性調整規模	您可為資料表及全域次要索引增加任意數量的佈建讀取輸送容量和佈建寫入輸送容量，前提是您未超過每份資料表及每個帳戶的限制。如需詳細資訊，請參閱Amazon DynamoDB 中的配額。	2014 年 10 月 7 日

變更	描述	變更日期
更大的項目大小	DynamoDB 的項目大小上限已從 64 KB 提高至 400 KB。如需詳細資訊，請參閱Amazon DynamoDB 中的配額。	2014 年 10 月 7 日
改善條件表達式	DynamoDB 擴展了條件表達式可用的運算子，讓您能更靈活地進行條件式放置、更新和刪除。新提供的運算子可讓您檢查屬性是否存在、是否大於或小於特定值、是否介於兩值之間、是否以特定字元開頭等。 DynamoDB 也提供選用的 OR 運算子來評估多個條件。根據預設，表達式中的多項條件使用 AND 連在一起，所以只有當表達式的所有條件都為 true 時，表達式才為 true。如果您改為指定 OR，當一或多項條件為 true 時，表達式即為 true。如需詳細資訊，請參閱在 DynamoDB 使用項目和屬性。	2014 年 4 月 24 日

變更	描述	變更日期
查詢篩選條件	DynamoDB Query API 支援新的 QueryFilter 選項。根據預設，Query 會尋找符合特定分割區索引鍵值和選用排序索引鍵條件的項目。Query 篩選條件會將條件表達式套用到其他非索引鍵屬性，如果 Query 篩選條件存在，則會在 Query 結果傳回應用程式前，捨棄不符合篩選條件的項目。如需詳細資訊，請參閱 在 DynamoDB 中查詢資料表。	2014 年 4 月 24 日
使用 匯出和匯入資料 AWS 管理主控台	DynamoDB 主控台已強化，可簡化 DynamoDB 資料表的資料匯出與匯入。只要按幾下，您就可以設定 AWS Data Pipeline 統整工作流程，以及設定 Amazon Elastic MapReduce 叢集將資料從 DynamoDB 資料表複製到 Amazon S3 儲存貯體，反之亦然。您只可執行一次匯出或匯入，或者設定每日匯出任務。您甚至可以執行跨區域匯出和匯入，將 DynamoDB 資料從一個 AWS 區域中的資料表複製到另一個 AWS 區域中的資料表。	2014 年 3 月 6 日

變更	描述	變更日期
重新整理較上層 API 的文件	現已可輕易取得下列 API 的資訊： • Java：DynamoDBMappper • .NET：文件模型和物件持久性模型 這些較上層的 API 現在都記錄在這裡：適用於 DynamoDB 的更高階程式設計界面。	2014 年 1 月 20 日
全域次要索引	DynamoDB 新增了全域次要索引的支援。如同本機次要索引，您使用資料表中的其他索引鍵來定義全域次要索引，再對索引發出 Query 請求。但與本機次要索引不同的是，全域次要索引的分割區索引鍵不必和資料表的分割區索引鍵一樣，而可以是該資料表中任何純量屬性。排序索引鍵為選用，而且也可以是任何純量資料表屬性。全域次要索引也有自己的佈建輸送量設定，而且與父資料表的佈建輸送量設定無關。如需詳細資訊，請參閱 在 DynamoDB 中使用次要索引來改善資料存取 及 在 DynamoDB 中使用全域次要索引。	2013 年 12 月 12 日

變更	描述	變更日期
精細定義存取控制	DynamoDB 新增了精細定義存取控制的支援。這項功能可讓客戶指定哪些主體 (使用者、群組或角色) 可以存取 DynamoDB 資料表或次要索引中的個別項目和屬性。應用程式也可充分利用 Web 聯合身分，將使用者身分驗證的任務交給第三方身分提供者，例如 Facebook、Google 或 Login with Amazon。如此，應用程式 (包括行動應用程式) 即可處理非常大量的使用者，同時確保除非使用者獲得授權，否則均無法存取 DynamoDB 資料項目。如需詳細資訊，請參閱 使用 IAM 政策條件進行精細定義存取控制。	2013 年 10 月 29 日
4 KB 讀取容量單位大小	讀取容量單位大小已從 1 KB 提高至 4 KB。這項強化功能可以減少許多應用程式所需的佈建讀取容量單位數目。例如，在這個版本以前，讀取 10 KB 的項目會使用 10 個讀取容量單位；而現在同樣讀取 10 KB 只會使用 3 個單位 (10 KB / 4 KB，四捨五入到下一個 4 KB 內)。如需詳細資訊，請參閱DynamoDB 輸送容量。	2013 年 5 月 14 日

變更	描述	變更日期
平行掃描	DynamoDB 新增了對平行掃描操作的支援。應用程式現在已可將資料表分割成多個邏輯區段，然後同時掃描所有區段。這項功能可減少完成掃描所需的時間，並充分利用資料表的佈建讀取容量。如需詳細資訊，請參閱 掃描 DynamoDB 中的資料表。	2013 年 5 月 14 日
本機次要索引	DynamoDB 新增了對本機次要索引的支援。您可以對非索引鍵屬性定義排序索引鍵索引，然後在 Query 請求中使用這些索引。使用本機次要索引時，應用程式可跨多維度有效率地擷取資料項目。如需詳細資訊，請參閱 本機次要索引。	2013 年 4 月 18 日
新的 API 版本	DynamoDB 在此版本中推出新的 API 版本 (2012-08-10)。並仍支援舊版 API (2011-12-05)，以與現有應用程式回溯相容。新的應用程式應使用新的 API 版本 2012-08-10。建議您將現有的應用程式移轉至最新的 API 版本 (2012-08-10)，因為新的 DynamoDB 功能 (例如本機版次要索引) 不會向後移植到舊版的 API。如需 API 版本 2012-08-10 的詳細資訊，請參閱《Amazon DynamoDB API 參考》。	2013 年 4 月 18 日

變更	描述	變更日期
IAM 政策變數支援	IAM 存取政策語言現在支援變數。當政策接受評估時，任何政策變數都會取代為已驗證使用者工作階段中內容資訊提供的值。您可使用政策變數定義一般用途政策，並不需明確列出政策的所有元件。如需政策變數的詳細資訊，請參閱AWS Identity and Access Management 使用 IAM 指南中的政策變數。 如需 DynamoDB 的政策變數範例，請參閱 Amazon DynamoDB 的 Identity and Access Management。	2013 年 4 月 4 日
已更新第 2 適用於 PHP 的 AWS SDK 版的 PHP 程式碼範例	第 2 版 適用於 PHP 的 AWS SDK 現已推出。《Amazon DynamoDB 開發人員指南》中的 PHP 程式碼範例已更新，可使用這項新的開發套件。如需開發套件第 2 版的詳細資訊，請參閱 適用於 PHP 的 AWS SDK。	2013 年 1 月 23 日
新的端點	DynamoDB 擴展到 AWS GovCloud （美國西部） 區域。如需目前的服務端點和協定清單，請參閱區域與端點。	2012 年 12 月 3 日
新的端點	DynamoDB 擴展至南美洲 (聖保羅) 區域。如需目前的支援端點清單，請參閱區域與端點。	2012 年 12 月 3 日

變更	描述	變更日期
新的端點	DynamoDB 擴展至亞太地區 (雪梨)。如需目前的支援端點清單，請參閱區域與端點。	2012 年 11 月 13 日
DynamoDB 實作 CRC32 檢查總和的支援、支援強烈一致批次擷取，及移除對並行資料表更新的限制。	• DynamoDB 會計算 HTTP 承載的 CRC32 檢查總和，並在新的標頭 `x-amz-crc32` 中傳回此檢查總和。如需詳細資訊，請參閱 DynamoDB 低階 API。 • 根據預設，`BatchGetItem` API 執行的讀取操做為最終一致。`ConsistentRead` 中的新 `BatchGetItem` 參數可讓您為請求中的任何資料表改選擇強式讀取一致性。如需詳細資訊，請參閱 描述。 • 此版本會在同時更新許多資料表時，移除一些限制。一次可更新的資料表總數仍為 10，但這些資料表現在可以是 `CREATING`、`UPDATING` 或 `DELETING` 狀態的任意組合。此外，資料表再也沒有增加或減少 ReadCapacityUnits 或 WriteCapacityUnits 的數量下限。如需詳細資訊，請參閱Amazon DynamoDB 中的配額。	2012 年 11 月 2 日

變更	描述	變更日期
最佳實務文件	《Amazon DynamoDB 開發人員指南》提供使用資料表和項目的最佳實務，以及查詢和掃描操作的建議。	2012 年 9 月 28 日

變更	描述	變更日期
二進位資料類型的支援	除數字和字串類型外，DynamoDB 現在也支援二進位資料類型。 在此版本以前，若要存放二進位資料，您要將二進位資料轉換成字串格式存放在 DynamoDB 中。除了用戶端必要的轉換工作，轉換通常會增加資料項目的大小，而需要更多儲存體和可能的額外佈建輸送容量。 使用二進位類型屬性，您現在已可存放任何二進位資料，例如壓縮的資料、加密的資料和影像。如需更多資訊，請參閱資料類型。如需使用 AWS SDKs 處理二進位類型資料的工作範例，請參閱下列各節： • 範例：使用 適用於 Java 的 AWS SDK 文件 API 處理二進位類型屬性 • 範例：使用 適用於 .NET 的 AWS SDK 低階 API 處理二進位類型屬性 對於在 AWS SDKs 中新增的二進位資料類型支援，您將需要下載最新的 SDKs，而且您也可能需要更新任何現有的應用程式。如需下載 AWS 軟體開發套件的資訊，請參閱「.NET 程式碼範例」。	2012 年 8 月 21 日

變更	描述	變更日期
您可使用 DynamoDB 主控台更新及複製 DynamoDB 資料表項目	DynamoDB 使用者現在除了可新增及刪除項目之外，還可以使用 DynamoDB 主控台更新及複製資料表項目。這項新功能簡化了透過主控台變更個別項目的程序。	2012 年 8 月 14 日
DynamoDB 降低了最低資料表輸送量需求	DynamoDB 現在支援較低的最低資料表輸送量需求，具體為 1 個寫入容量單位和 1 個讀取容量單位。如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的 Amazon DynamoDB 中的配額 主題。	2012 年 8 月 9 日
Signature 第 4 版支援	DynamoDB 現在支援 Signature 第 4 版驗證請求。	2012 年 7 月 5 日
DynamoDB 主控台中的 Table Explorer 支援	DynamoDB 主控台現在支援 Table Explorer，讓您可瀏覽及查詢資料表中的資料。您也可以插入新的項目或刪除現有的項目。已針對這些功能更新 `SampleData` 和 使用主控台 章節。	2012 年 5 月 22 日
新的端點	DynamoDB 可用性因美國西部 (加利佛尼亞北部) 區域、美國西部 (奧勒岡) 區域和亞太地區 (新加坡) 中的新端點而得以擴展。 如需目前的支援端點清單，請前往區域與端點。	2012 年 4 月 24 日

變更	描述	變更日期
BatchWriteItem API 支援	DynamoDB 現已支援批次寫入 API，讓您可在單一 API 呼叫中對一或多個資料表，存放與刪除多個項目。如需 DynamoDB 批次寫入 API 的詳細資訊，請參閱「BatchWriteItem」。 如需使用項目和使用批次寫入功能使用 AWS SDKs的詳細資訊，請參閱 在 DynamoDB 使用項目和屬性和 .NET 程式碼範例。	2012 年 4 月 19 日
記錄更多錯誤碼	如需詳細資訊，請參閱 使用 DynamoDB 時發生錯誤。	2012 年 4 月 5 日
新的端點	DynamoDB 擴展至亞太地區 (東京)。如需目前的支援端點清單，請參閱區域與端點。	2012 年 2 月 29 日
新增了 ReturnedItemCount 指標	新的指標 ReturnedItemCount 會提供於 DynamoDB 之 Query 或 Scan 操作回應中傳回的項目數，並可透過 CloudWatch 監控。	2012 年 2 月 24 日
新增了增加數值的範例	DynamoDB 支援增加和減少現有的數值。這些範例會在下列主題的「更新項目」章節中示範新增至現有數值： 使用項目：Java. 處理項目：.NET.	2012 年 1 月 25 日

變更	描述	變更日期
初始產品版本	DynamoDB 作為 Beta 版新服務推出。	2012 年 1 月 18 日

DynamoDB 的舊版功能

下列主題是 DynamoDB 仍支援的舊版功能。我們不會再對這些功能進行任何積極開發。

主題

- 全域資料表 2017.11.29 版 (舊版)
- 舊版低階 DynamoDB API (2011-12-05)
- 舊版 DynamoDB 條件式參數

全域資料表 2017.11.29 版 (舊版)

> **Important**
>
> 本文件適用於全域資料表 2017.11.29 版 (舊版)，新的全域資料表應避免使用此文件。客戶應盡可能使用 2019.11.21 版全域資料表 (目前版本)，因為相較於 2017.11.29 (舊版)，其提供更大的彈性、更高的效率，且消耗的寫入容量更少。
> 若要判斷您使用的是哪個版本，請參閱 判斷全域資料表版本。若要將現有全域資料表從 2017.11.29 版 (舊版) 更新至 2019.11.21 版 (目前版本)，請參閱 DynamoDB 全域資料表版本。

主題

- 全域資料表：運作方式
- 管理全域資料表的最佳實務和要求
- 建立全域資料表 (2017.11.29 版)
- 監控全域資料表
- 對全域資料表使用 IAM

全域資料表：運作方式

> **Important**
>
> 本文件適用於全域資料表 2017.11.29 版 (舊版)，新的全域資料表應避免使用此文件。客戶應盡可能使用 2019.11.21 版全域資料表 (目前版本)，因為相較於 2017.11.29 (舊版)，其提供更大的彈性、更高的效率，且消耗的寫入容量更少。
> 若要判斷您使用的是哪個版本，請參閱 判斷全域資料表版本。若要將現有全域資料表從 2017.11.29 版 (舊版) 更新至 2019.11.21 版 (目前版本)，請參閱 DynamoDB 全域資料表版本。

以下各節可協助您了解 Amazon DynamoDB 中全域資料表的概念和行為。

2017.11.29 版 (舊版) 的全域資料表概念

全域資料表是一或多個複本資料表的集合，全部由單一 AWS 帳戶擁有。

複本列表 (簡稱複本) 是單一 DynamoDB 資料表，可作為全域資料表的一部分運作。每個複本會存放相同的資料項目集。任何特定全域資料表的每個 AWS 區域只能有一個複本列表。

以下是如何建立全域資料表的概念性概觀。

1. 在 AWS 區域中建立已啟用 DynamoDB Streams 的一般 DynamoDB 資料表。
2. 針對要複寫資料的每個其他區域重複步驟 1。
3. 根據您建立的資料表定義 DynamoDB 全域資料表。

會將這些任務 AWS 管理主控台 自動化，讓您可以更快速輕鬆地建立全域資料表。如需詳細資訊，請參閱建立全域資料表 (2017.11.29 版)。

產生的 DynamoDB 全域資料表由多個複本列表組成，DynamoDB 會將其視為單一單位 (每個區域一個)。每個複本都有相同的資料表名稱和相同的主索引鍵結構描述。當應用程式將資料寫入某個區域的複本列表時，DynamoDB 會自動將寫入傳播到另一個 AWS 區域的其他複本列表。

> **Important**
>
> 為了讓資料表資料保持同步，全域資料表會自動為每個項目建立下列屬性：
>
> - `aws:rep:deleting`

- `aws:rep:updatetime`
- `aws:rep:updateregion`

請勿修改這些屬性或建立具有相同名稱的屬性。

您可以將複本列表新增到全域資料表，以便在其他區域中使用該複本列表。(若要執行這項操作，全域資料表必須是空的。換句話說，任何複本列表中都不得包含任何資料。)

您也可以從全域資料表中移除複本列表。如果您執行這項操作，則資料表與全域資料表的關聯會完全取消。這個新的獨立資料表不再與全域資料表互動，而且資料不再傳播至全域資料表或從全域資料表傳播。

Warning

請注意，移除複本不是自動的。為了確保一致的行為和已知狀態，您可能需要考慮事先將應用程序寫入流量從要移除的複本轉移。移除後，請等候所有複本區域端點將複本顯示為已解除關聯，再將其作為自己的隔離區域表進行進一步的寫入操作。

一般任務

全域資料表的一般任務如下。

您可以刪除全域資料表的複本資料表，方式與一般資料表相同。這將停止複寫到該區域，並刪除保留在該區域中的資料表複本。您無法中斷複寫，並讓資料表的複本以獨立實體的形式存在。

Note

在來源資料表用於啟動新區域後至少 24 小時之後，您才能刪除來源資料表。若您嘗試刪除，很快便會收到錯誤。

如果應用程式大約在同一時間更新不同區域中的相同項目，則可能會發生衝突。為了協助確保最終一致性，DynamoDB 全域資料表會在並行更新間使用「最後寫入者獲勝」方法，在並行更新間使用最後寫入者。所有的複本都會同意最新的更新，並朝它們都具有相同資料的狀態收斂。

> **Note**
>
> 有幾種方式可以避免衝突，包括：
>
> - 使用 IAM 政策，只允許寫入一個區域中的資料表。
> - 使用 IAM 政策，將使用者路由到單一區域，並將另一個區域保持為閒置待命狀態，或者交替將奇數使用者路由到一個區域，將偶數使用者路由到另一個區域。
> - 避免使用非等冪更新，例如書籤 = 書籤 + 1，以支援靜態更新，例如書籤 = 25。

監控全域資料表

您可以使用 CloudWatch 觀察 `ReplicationLatency` 指標。假設某個更新項目出現在 DynamoDB 串流的複本資料表，此指標可追蹤該項目出現在全域資料表中其他複本時經過的時間。`ReplicationLatency` 會以毫秒表示，並會針對每對來源及目的地區域對發送。這是全域資料表 v2 提供的唯一 CloudWatch 指標。

您觀察到的延遲取決於所選擇區域之間的距離以及其他變數。區域的 0.5 到 2.5 秒範圍內的延遲在同一地理區域內可能很常見。

存留時間 (TTL)

您可以使用存留時間 (TTL) 來指定屬性名稱，其值表示項目的到期時間。自 Unix epoch 開始以來，此值都以秒為單位。

使用全域資料表舊版時，系統不會自動將 TTL 刪除作業複寫到其他複本。透過 TTL 規則刪除項目時，該工作執行時不會耗用寫入單位。

請注意，如果來源和目標資料表的佈建寫入容量非常低，這可能會造成限流，因為 TTL 刪除需要寫入容量。

使用全域資料表的串流和交易

每個全域資料表都會根據其所有寫入作業產生獨立的串流，無論這些寫入的起始點為何。您可以選擇在一個區域或所有區域中獨立使用此 DynamoDB 串流。

如果您想要處理本機寫入，但不要複製寫入，可以將自己的區域屬性新增至每個項目。然後，您可以使用 Lambda 事件篩選條件，只調用 Lambda 在本機區域中進行寫入。

交易操作僅在最初進行寫入的區域內提供 ACID (不可分割性、一致性、隔離性和耐久性) 保證。全域資料表不支援跨區域交易。

例如，如果您有一個全域資料表，其在美國東部 (俄亥俄) 和美國西部 (奧勒岡) 區域有複本，並且在美國東部 (俄亥俄) 區域執行 TransactWriteItems 操作，在這些變更進行複寫之後，您可能會在美國西部 (奧勒岡) 區域看到部分完成的交易。只有當變更已在來源區域遞交的情況下，這些變更才會複寫至其他區域。

> **Note**
>
> - 全域資料表透過直接更新 DynamoDB 來「寫入」DynamoDB 加速器。因此，DAX 不會意識到它持有過時的資料。只有在快取的 TTL 到期時，才會重新整理 DAX 快取。
> - 全域資料表上的標籤不會自動傳播。

讀取和寫入輸送量

全域資料表會以下列方式管理讀取和寫入輸送量。

- 跨區域的所有資料表執行個體的寫入容量必須相同。
- 使用版本 2019.11.21 (目前版本) 時，如果資料表設為支援自動擴展或處於隨需模式，則寫入容量會自動保持同步。在這些同步的自動擴展設定中，每個區域目前佈建的寫入容量可獨立增減。如果將資料表置於隨需模式，則該模式將同步至其他複本。
- 區域之間的讀取容量可能會有所不同，因為讀取可能不相等。將全域複本新增至資料表時，會傳播來源區域的容量。建立之後，您可以調整某個複本的讀取容量，而這個新設定不會傳輸到另一端。

一致性和衝突解決

對任何複本列表中任何項目所做的任何變更都會複寫到相同全域資料表中的所有其他複本。在全域資料表中，新寫入的項目通常會在幾秒鐘內傳播到所有複本列表。

使用全域資料表，每個複本列表會存放相同的資料項目集。DynamoDB 不支援僅對某些項目進行部分複寫。

應用程式可以讀取資料和將資料寫入至任何複本列表。DynamoDB 支援跨區域的最終一致讀取，但不支援跨區域的高度一致性讀取。如果您的應用程式只使用最終一致讀取，且只針對一個 AWS 區域讀取的問題，則其將正常運作，而不會進行任何修改。不過，如果您的應用程式需要強式一致讀取，則必須在相同區域中執行所有高度一致性讀取和寫入。否則，如果您對某個區域進行寫入並從另一個區域進行讀取，則讀取回應可能包含過時資料，這些資料不會反映最近在另一個區域中完成的寫入結果。

如果應用程式大約在同一時間更新不同區域中的相同項目，則可能會發生衝突。為了協助確保最終一致性，DynamoDB 全域資料表會在並行更新間使用最後寫入者獲勝核對機制，其中 DynamoDB 會在並行更新間盡最大努力判斷最後寫入者。有了這個衝突解決機制，所有的複本都會同意最新的更新，並朝它們都具有相同資料的狀態收斂。

可用性與持久性

如果單一 AWS 區域變得隔離或降級，您的應用程式可以重新導向至不同的區域，並對不同的複本資料表執行讀取和寫入。您可以套用自訂商業邏輯來決定何時將請求重新導向至其他區域。

如果區域變得孤立或降級，DynamoDB 會追蹤已執行但尚未傳播到所有複本列表的任何寫入。當區域重新回到線上的狀態時，DynamoDB 會繼續將該區域的任何擱置寫入傳播到其他區域的複本列表中。其也會繼續將寫入從其他複本列表傳播到目前已重回到線上狀態的區域。無論區域隔離多長時間，所有之前成功的寫入都將最終傳播。

管理全域資料表的最佳實務和要求

> **Important**
>
> 本文件適用於全域資料表 2017.11.29 版 (舊版)，新的全域資料表應避免使用此文件。客戶應盡可能使用 2019.11.21 版全域資料表 (目前版本)，因為相較於 2017.11.29 (舊版)，其提供更大的彈性、更高的效率，且消耗的寫入容量更少。
> 若要判斷您使用的是哪個版本，請參閱 判斷全域資料表版本。若要將現有全域資料表從 2017.11.29 版 (舊版) 更新至 2019.11.21 版 (目前版本)，請參閱 DynamoDB 全域資料表版本。

使用 Amazon DynamoDB 全域資料表，您可以跨 AWS 區域複寫資料表資料。全域資料表中的複本列表和次要索引必須具有相同的寫入容量設定，才能確保資料能正確複寫。

主題

- 全域資料表版本
- 新增複本資料表的要求
- 管理容量的最佳實務和要求

全域資料表版本

DynamoDB 全域資料表有兩種版本可用：2019.11.21 版全域資料表 (目前版本) 和 全域資料表 2017.11.29 版 (舊版)。客戶應盡可能使用 2019.11.21 版全域資料表 (目前版本)，因為相較於 2017.11.29 (舊版)，其提供更大的彈性、更高的效率，且消耗的寫入容量更少。

若要判斷您使用的是哪個版本，請參閱 判斷全域資料表版本。若要將現有全域資料表從 2017.11.29 版 (舊版) 更新至 2019.11.21 版 (目前版本)，請參閱 升級全域資料表。

新增複本資料表的要求

若想將新的複本列表新增至全域資料表，下列每一個條件都必須成立：

- 資料表必須具備與所有其他複本相同的分割區索引鍵。
- 資料表必須具備指定的寫入容量管理設定。
- 資料表必須具備與所有其他複本相同的名稱。
- 資料表必須啟用 DynamoDB Streams，並且串流同時包含項目的新映像和舊映像。
- 全域資料表中的新或現有複本列表都不能包含任何資料。

如果指定全域次要索引，則也必須符合下列條件：

- 全域次要索引必須具有相同的名稱。
- 全域次要索引必須具有同樣的分割區索引鍵和排序索引鍵 (若有)。

> **Important**
>
> 應在全域資料表的所有複本列表和相符的次要索引之間一致地設定寫入容量設定。若要更新全域資料表的寫入容量設定，強烈建議使用 DynamoDB 主控台或 `UpdateGlobalTableSettings` API 操作。`UpdateGlobalTableSettings` 會自動將寫入容量設定的變更套用至所有複本列表和全域資料表中相符的次要索引。如果使用 `UpdateTable`、`RegisterScalableTarget` 或 `PutScalingPolicy` 操作，您應該將變更套用至每個複本列表和個別相符的次要索引。如需詳細資訊，請參閱《Amazon DynamoDB API 參考》中的 UpdateGlobalTableSettings。
>
> 我們強烈建議您啟用自動擴展來管理佈建的寫入容量設定。如果您偏好手動管理寫入容量設定，則應為所有複本列表佈建相同的複寫寫入容量單位。此外，您還要為全域資料表中的相符次要索引佈建相同的複寫寫入容量單位。

您還必須擁有適當的 AWS Identity and Access Management (IAM) 許可。如需詳細資訊，請參閱對全域資料表使用 IAM。

管理容量的最佳實務和要求

在 DynamoDB 中管理複本列表的容量設定時，請考慮下列事項。

使用 DynamoDB Auto Scaling

使用 DynamoDB Auto Scaling 功能是管理輸送容量設定的建議方式，適用於使用佈建模式的複本列表。DynamoDB 自動擴展功能會根據實際應用程式工作負載，自動調整每個複本列表的讀取容量單位 (RCU) 和寫入容量單位 (WCU)。如需詳細資訊，請參閱使用 DynamoDB Auto Scaling 功能自動管理輸送容量。

如果您使用 建立複本資料表 AWS 管理主控台，則預設會為每個複本資料表啟用自動擴展，並使用預設的自動擴展設定來管理讀取容量單位和寫入容量單位。

透過 DynamoDB 主控台或使用 `UpdateGlobalTableSettings` 呼叫對複本列表或次要索引的自動擴展設定所做的變更，會自動套用至所有複本列表，以及全域資料表中的相符次要索引。這些變更會覆寫任何現有的自動擴展設定。這可確保佈建的寫入容量設定在全域資料表中的複本列表和次要索引之間保持一致。如果使用 `UpdateTable`、`RegisterScalableTarget` 或 `PutScalingPolicy` 呼叫，您應該將變更個別套用至每個複本列表，以及相符的次要索引。

> **Note**
>
> 如果自動擴展無法滿足應用程式的容量變更 (無法預測的工作負載)，或者您不想進行其設定 (最小、最大或使用率閾值的目標設定)，則可以使用隨需模式來管理全域資料表的容量。如需詳細資訊，請參閱隨需模式。
> 如果您在全域資料表上啟用隨需模式，則複寫寫入請求單元 (rWCU) 的耗用方式會與 rWCU 的佈建方式一致。例如，如果您對在另外兩個區域中複寫的本機資料表執行 10 次寫入，則會耗用 60 個寫入要求單位 (10 + 10 + 10 = 30；30 x 2 = 60)。耗用的 60 個寫入請求單位包括全域資料表版本 2017.11.29 (舊版) 所耗用的額外寫入，以更新 `aws:rep:deleting`、`aws:rep:updatetime` 和 `aws:rep:updateregion` 屬性。

手動管理容量

如果您決定不使用 DynamoDB Auto Scaling 功能，則必須在每個複本列表和次要索引上手動設定讀取容量和寫入容量設定。

每個複本列表上佈建的複寫寫入容量單位 (rWCU) 應設定為將跨所有區域的應用程式寫入所需的 rWCU 總數乘以二。這可容納在本地區域發生的應用程式寫入，以及來自其他區域的複寫應用程式寫入。例如，假設您預期在俄亥俄州的複本列表每秒 5 次寫入，在維吉尼亞北部複本列表每秒 5 次寫入。在此情況下，您應該將 20 個 rWCU 佈建至每個複本資料表 (5 + 5 = 10；10 x 2 = 20)。

若要更新全域資料表的寫入容量設定，強烈建議使用 DynamoDB 主控台或 `UpdateGlobalTableSettings` API 操作。`UpdateGlobalTableSettings` 會自動將寫入容量設定的變更套用至所有複本列表和全域資料表中相符的次要索引。如果使用 `UpdateTable`、`RegisterScalableTarget` 或 `PutScalingPolicy` 操作，您應該將變更套用至每個複本列表和個別相符的次要索引。如需詳細資訊，請參閱《Amazon DynamoDB API 參考》。

> **Note**
>
> 若要更新 DynamoDB 中全域資料表的設定 (`UpdateGlobalTableSettings`)，您必須擁有 `dynamodb:UpdateGlobalTable`、`dynamodb:DescribeLimits`、`application-autoscaling:DeleteScalingPolicy` 以及 `application-autoscaling:DeregisterScalableTarget` 許可。如需詳細資訊，請參閱對全域資料表使用 IAM。

建立全域資料表 (2017.11.29 版)

> **Important**
>
> 本文件適用於全域資料表 2017.11.29 版 (舊版)，新的全域資料表應避免使用此文件。客戶應盡可能使用 2019.11.21 版全域資料表 (目前版本)，因為相較於 2017.11.29 (舊版)，其提供更大的彈性、更高的效率，且消耗的寫入容量更少。
> 若要判斷您使用的是哪個版本，請參閱 判斷全域資料表版本。若要將現有全域資料表從 2017.11.29 版 (舊版) 更新至 2019.11.21 版 (目前版本)，請參閱 DynamoDB 全域資料表版本。

本節說明如何使用 Amazon DynamoDB 主控台或 AWS Command Line Interface () 建立全域資料表 AWS CLI。

主題

- 建立全域資料表 (主控台)
- 建立全域資料表 (AWS CLI)

建立全域資料表 (主控台)

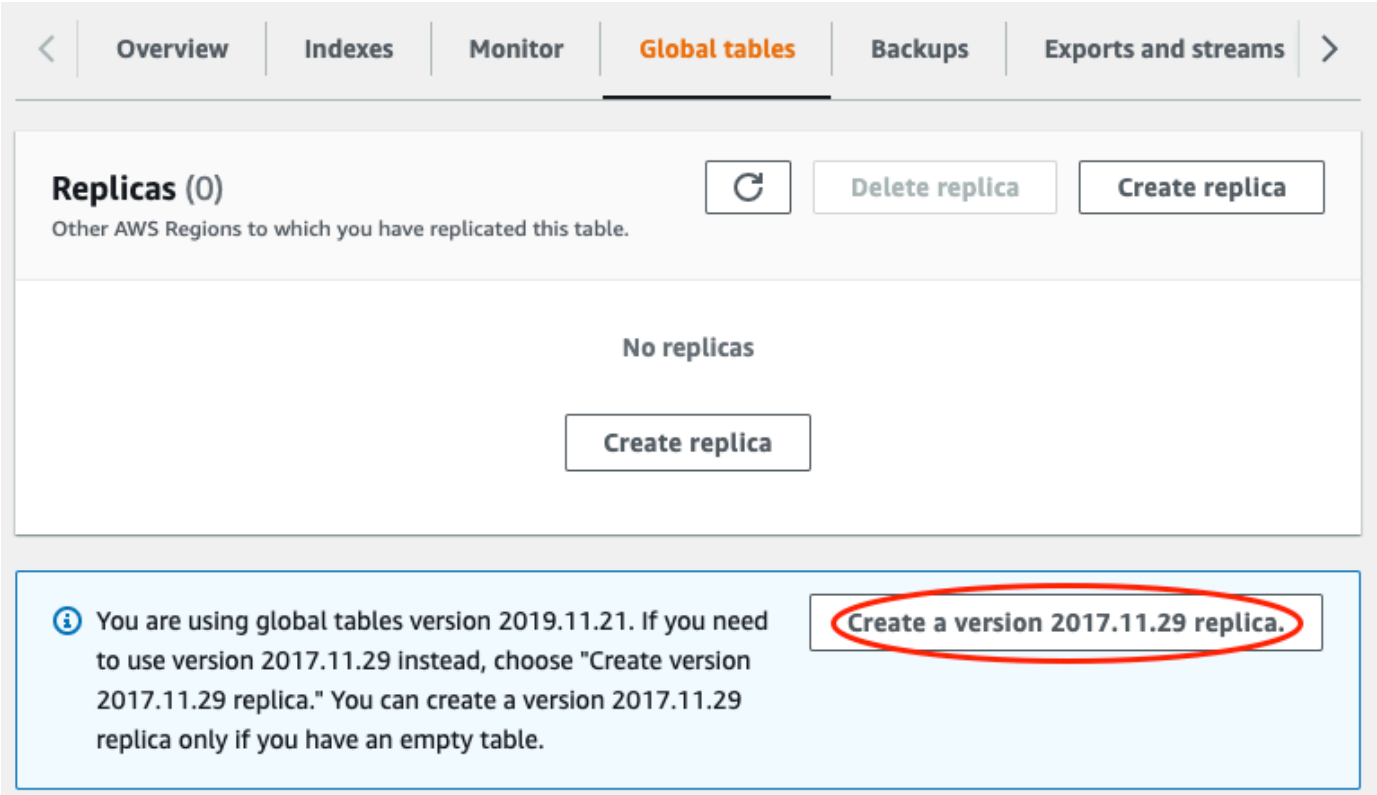
請遵循下列步驟，使用主控台建立全域資料表。下列範例使用美國及歐洲的複本列表，建立全域資料表。

1. 在 https://console.aws.amazon.com/dynamodb/home 開啟 DynamoDB 主控台。針對此範例，請選擇 us-east-2 (美國東部俄亥俄州) 區域。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇 Create Table (建立資料表)。

   對於 Table name (資料表名稱)，請輸入 **`Music`**。

   針對 Primary key (主索引鍵)，輸入 **`Artist`**。選擇 Add sort key (新增排序索引鍵)，然後輸入 **`SongTitle`**。(**`Artist`** 和 **`SongTitle`** 都必須是字串。)

   若要建立資料表，請選擇 Create (建立)。此資料表會做為新全域資料表中的第一個複本資料表。這是您稍後新增其他複本資料表的原型。
4. 選擇 Global Tables (全域資料表) 索引標籤，然後選擇 Create a version 2017.11.29 (Legacy) replica (建立版本 2017.11.29 (舊版) 複本)。



5. 從 Available replication Regions (可用的複製區域) 下拉式清單，選擇 US West (Oregon) (美國西部 (奧勒岡))。

   主控台會檢查，確認所選取的區域中沒有同名的資料表。若有同名的資料表，您必須先刪除現有的資料表，才可在該區域中建立新的複本資料表。

6. 選擇 Create replica (建立複本)。這會啟動在美國西部 (奧勒岡) 建立資料表的程序。

   所選資料表的 Global Table (全域資料表) 標籤 (及所有其他複本資料表)，會顯示該資料表已複寫到多個區域。

7. 您現在可以新增其他區域，複寫及同步美國與歐洲的全域資料表。若要執行此作業，請重複步驟 5，但這次請改為指定 Europe (Frankfurt) (歐洲 (法蘭克福))，而非 US West (Oregon) (美國西部 (奧勒岡))。

8. 您仍然應該 AWS 管理主控台 在美國東部 （俄亥俄） 區域使用 。選取左側導覽功能表中的 Items (項目)，選取 Music (音樂) 資料表，然後選擇 Create Item (建立項目)。

   a. 針對 Artist (藝人)，輸入 **`item_1`**。

   b. 針對 SongTitle (歌曲名稱)，輸入 **`Song Value 1`**。

   c. 若要寫入該項目，請選擇 Create item (建立項目)。

9. 稍待片刻之後，該項目將會複寫到您全域資料表中的所有三個區域。若要確認，請在主控台中移至右上角的區域選擇器，然後選擇 Europe (Frankfurt) (歐洲 (法蘭克福))。歐洲 (法蘭克福) 中的 `Music` 資料表此時應已包含新的項目。

10. 重複步驟 9，然後選擇 US West (Oregon) (美國西部 (奧勒岡)) 以驗證該區域中的複寫。

建立全域資料表 (AWS CLI)

請遵循下列步驟，使用 AWS CLI 建立全域資料表 `Music`。以下範例會建立全域資料表，並在美國及歐洲皆擁有複本資料表。

1. 在美國東部 (俄亥俄) 建立新的資料表 (`Music`)，並啟用 DynamoDB Streams (`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. 在美國東部 (維吉尼亞北部) 建立相同的 Music 資料表。

```
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. 建立全域資料表 (Music)，其中包含 us-east-2 和 us-east-1 區域中的複本列表。

```
aws dynamodb create-global-table \
    --global-table-name Music \
    --replication-group RegionName=us-east-2 RegionName=us-east-1 \
    --region us-east-2
```

> **Note**
>
> 全域資料表名稱 (Music) 必須與每個複本列表 (Music) 的名稱相符。如需詳細資訊，請參閱全域資料表的最佳實務。

4. 在歐洲 (愛爾蘭) 中建立另一個資料表，其設定與您在步驟 1 和步驟 2 中建立的資料表設定相同。

```
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
```

完成此步驟後，將新資料表新增至 Music 全域資料表。

```
aws dynamodb update-global-table \
    --global-table-name Music \
    --replica-updates 'Create={RegionName=eu-west-1}' \
    --region us-east-2
```

5. 若要確認複寫正常運作，請將項目新增到美國東部 (俄亥俄) 中的 Music 資料表。

```
aws dynamodb put-item \
    --table-name Music \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

6. 稍待幾秒鐘，然後檢查該項目是否已成功複寫到美國東部 (維吉尼亞北部) 與歐洲 (愛爾蘭)。

```
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
```

監控全域資料表

Important

本文件適用於全域資料表 2017.11.29 版 (舊版)，新的全域資料表應避免使用此文件。客戶應盡可能使用 2019.11.21 版全域資料表 (目前版本)，因為相較於 2017.11.29 (舊版)，其提供更大的彈性、更高的效率，且消耗的寫入容量更少。

若要判斷您使用的是哪個版本，請參閱 判斷全域資料表版本。若要將現有全域資料表從 2017.11.29 版 (舊版) 更新至 2019.11.21 版 (目前版本)，請參閱 DynamoDB 全域資料表版本。

您可以使用 Amazon CloudWatch 來監控全域資料表的行為和效能。Amazon DynamoDB 為全域資料表中的每個複本發布 ReplicationLatency 和 PendingReplicationCount 指標。

- **ReplicationLatency**：某個更新項目出現在 DynamoDB 串流的複本列表，以及當該項目出現在全域資料表中的另一個複本經過的時間。ReplicationLatency 會以毫秒表示，並會針對每對來源及目標區域配對發送。

  ReplicationLatency 在正常操作期間應該很穩定。ReplicationLatency 值上升可能表示某個複本的更新未及時散佈到其他複本資料表。一段時間後，這會造成其他複本資料表落後，因為他們不再一致地收到更新。在此情況下，您應該確認每個複本資料表的讀取容量單位 (RCU) 和寫入容量單位 (WCU) 皆相同。此外，選擇 WCU 設定時應遵循 全域資料表版本 中的建議。

  如果某個 AWS 區域遭到降級，而您在該區域有複本列表，ReplicationLatency 就會增加。這種情況下，您可以暫時將應用程式的讀取和寫入活動重新導向至不同的 AWS 區域。

- **PendingReplicationCount**：寫入複本列表，但尚未寫入全域資料表中另一個複本的項目更新數目。PendingReplicationCount 會以項目數量表示，並會針對每對來源及目標區域配對發送。

  正常的操作期間，PendingReplicationCount 的值應非常低。如果 PendingReplicationCount 大幅增加，請調查複本列表的佈建寫入容量設定是否足以滿足您目前的工作負載。

  如果某個 AWS 區域遭到降級，而您在該區域有複本列表，PendingReplicationCount 就會增加。這種情況下，您可以暫時將應用程式的讀取和寫入活動重新導向至不同的 AWS 區域。

如需更多詳細資訊，請參閱 DynamoDB 指標和維度。

對全域資料表使用 IAM

> **Important**
>
> 本文件適用於全域資料表 2017.11.29 版 (舊版)，新的全域資料表應避免使用此文件。客戶應盡可能使用 2019.11.21 版全域資料表 (目前版本)，因為相較於 2017.11.29 (舊版)，其提供更大的彈性、更高的效率，且消耗的寫入容量更少。

若要判斷您使用的是哪個版本，請參閱 判斷全域資料表版本。若要將現有全域資料表從 2017.11.29 版 (舊版) 更新至 2019.11.21 版 (目前版本)，請參閱 DynamoDB 全域資料表版本。

當您第一次建立全域資料表時，Amazon DynamoDB 會自動為您建立 AWS Identity and Access Management (IAM) 服務連結角色。此角色名為 ，可讓 DynamoDB 代您管理全域資料表的跨區域複寫。請勿刪除此服務連結角色。若您刪除，則所有全域資料表將無法再運作。

如需服務連結角色的詳細資訊，請參閱《IAM 使用者指南》中的使用服務連結角色。

若要在 DynamoDB 中建立和維護全域資料表，您必須具備 `dynamodb:CreateGlobalTable` 許可才能存取下列各項：

- 您要新增的複本列表。
- 已經屬於全域資料表的每個現有複本。
- 全域資料表本身。

若要更新 DynamoDB 中全域資料表的設定 (`UpdateGlobalTableSettings`)，您必須擁有 `dynamodb:UpdateGlobalTable`、`dynamodb:DescribeLimits`、`application-autoscaling:DeleteScalingPolicy` 以及 `application-autoscaling:DeregisterScalableTarget` 許可。

在更新現有的擴展政策時需要使用 `application-autoscaling:DeleteScalingPolicy` 和 `application-autoscaling:DeregisterScalableTarget` 許可。如此一來，全域資料表服務可以在將新政策連接至資料表或次要索引之前移除舊的擴展政策。

如果您使用 IAM 政策來管理對複本列表的存取，則應將相同的政策套用至該全域資料表中的所有其他複本。此實務可協助您在所有複本列表中維持一致的許可模型。

透過在全域資料表中的所有複本上使用相同的 IAM 政策，您也可以避免將非預期的讀取和寫入存取權授予全域資料表資料。例如，假設使用者只能存取全域資料表中的一個複本。如果該使用者可以寫入此複本，則 DynamoDB 會將寫入傳播到所有其他複本列表。實際上，使用者可以 (間接) 寫入全域資料表中的所有其他複本。在所有複本列表上使用一致的 IAM 政策，即可避免此情況。

範例：允許 CreateGlobalTable 動作

在將複本新增至全域資料表之前，您必須擁有全域資料表及其每個複本列表的 `dynamodb:CreateGlobalTable` 許可。

下列 IAM 政策會授予許可，允許在所有資料表上執行 CreateGlobalTable 動作。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateGlobalTable"],
            "Resource": "*"
        }
    ]
}
```

範例：允許 UpdateGlobalTable、DescribeLimits、application-autoscaling:DeleteScalingPolicy 和 application-autoscaling:DeregisterScalableTarget 動作

若要更新 DynamoDB 中全域資料表的設定 (UpdateGlobalTableSettings)，您必須擁有 dynamodb:UpdateGlobalTable、dynamodb:DescribeLimits、application-autoscaling:DeleteScalingPolicy 以及 application-autoscaling:DeregisterScalableTarget 許可。

下列 IAM 政策會授予許可，允許在所有資料表上執行 UpdateGlobalTableSettings 動作。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateGlobalTable",
                "dynamodb:DescribeLimits",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

範例：僅允許針對特定全域資料表名稱 (具有僅在特定區域中允許的複本) 的 CreateGlobalTable 動作

下列 IAM 政策授予許可，允許 `CreateGlobalTable` 動作來建立在兩個區域中具有複本且名為 `Customers` 的全域資料表。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:CreateGlobalTable",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Customers",
                "arn:aws:dynamodb:us-east-1:123456789012:table/Customers",
                "arn:aws:dynamodb:us-west-1:123456789012:table/Customers"
            ]
        }
    ]
}
```

舊版低階 DynamoDB API (2011-12-05)

本節記錄舊版 DynamoDB 低階 API (2011-12-05) 中可用的操作。現保留此版本的低階 API，以便與現有應用程式回溯相容。

新的應用程式應使用目前 API 版本 (2012-08-10)。如需詳細資訊，請參閱 DynamoDB API 參考。

> **Note**
>
> 建議您將應用程式移轉至最新的 API 版本 (2012-08-10)，因為新的 DynamoDB 功能不會向後移植到舊版的 API。

主題

- BatchGetItem
- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTables
- GetItem
- ListTables
- PutItem
- Query
- Scan
- UpdateItem
- UpdateTable

BatchGetItem

> **Important**
>
> ***################## API ## 2011-12-05#***
>
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

`BatchGetItem` 操作會使用多個資料表的主索引鍵從中傳回多個項目的屬性。單一操作最多可擷取的項目數為 100。此外，擷取的項目數受到 1 MB 大小限制。如果超過回應大小限制，或者因為超出資料表的佈建輸送量，或因為內部處理失敗而傳回部分結果，則 DynamoDB 會傳回 `UnprocessedKeys`

值，以便您重試從下一個要獲取的項目開始的操作。DynamoDB 會自動調整每個頁面傳回的項目數，以便強制執行此限制。例如，即使您要求擷取 100 個項目，但每一個別項目大小為 50 KB，系統也會傳回 20 個項目和適當的 UnprocessedKeys 值，以便您取得下一頁的結果。如有需要，應用程式可包含自身的邏輯，將結果頁面組合成一組。

如果因為請求中涉及的每個資料表上佈建輸送量不足而無法處理任何項目，則 DynamoDB 會傳回 ProvisionedThroughputExceededException 錯誤。

> **Note**
>
> 根據預設，BatchGetItem 會對請求中每個資料表執行最終一致讀取。如果想改為一致性讀取，可以在每份資料表將 ConsistentRead 參數設定為 true。
> BatchGetItem 會平行擷取項目，將回應延遲減至最低。
> 設計應用程式時，請注意 DynamoDB 不保證傳回的回應中的屬性順序。在 AttributesToGet 中包含請求中項目的主索引鍵值，以便協助按項目剖析回應。
> 如果請求項目不存在，則這些項目的回應中不會傳回任何內容。請求不存在項目會使用最小讀取容量單位，具體值根據讀取類型而定。如需更多詳細資訊，請參閱 DynamoDB 項目大小和格式。

請求

語法

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{"RequestItems":
    {"Table1":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
    "Table2":
        {"Keys":
```

```
            [{"HashKeyElement": {"S":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}}],
        "AttributesToGet": ["AttributeName4", "AttributeName5", "AttributeName6"]
        }
    }
}
```

名稱	描述	必要
RequestItems	要以主索引鍵取得的資料表名稱和對應項目的容器。在請求項目時，每個資料表名稱在每次操作僅能呼叫一次。 類型：字串 預設：無	是
Table	包含所要取得的項目的資料表名稱。該項目只是一個字串，用以指定不帶標籤的現有資料表。 類型：字串 預設：無	是
Table:Keys	定義指定資料表中項目的主索引鍵值。如需主索引鍵的詳細資訊，請參閱 主索引鍵。 類型：金鑰	是
Table:AttributesToGet	指定資料表中的屬性名稱陣列。如果未指定屬性名稱，則會傳回所有屬性。如果有部分屬性未找到，則這些屬性不會出現在結果中。 類型：陣列	否

名稱	描述	必要
Table:ConsistentRead	如果設定為 true，則會發送一致性讀取，反之則會使用最終一致性。 類型：布林值	否

回應

語法

```
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"]}
        }
}
```

名稱	描述
Responses	資料表名稱以及資料表個別項目屬性。 類型：映射
Table	包含項目的資料表名稱。該項目只是一個字串，用以指定不帶標籤的資料表。 類型：字串
Items	符合操作參數的屬性名稱和值的容器。 類型：屬性名稱映射，以及其資料類型和值。
ConsumedCapacityUnits	每個資料表使用的讀取容量單位數目。此值顯示套用至佈建輸送量的數字。請求不存在項目會使用最小讀取容量單位，具體值依讀取類型而定。如需更多資訊，請參閱DynamoDB 佈建容量模式。 類型：數字
UnprocessedKeys	包含可能因回應大小達到限制，而未在目前回應中處理的資料表及其各自的索引鍵的陣列。UnprocessedKeys 值形式與 RequestItems 參數相同 (因此該值可直接提

名稱	描述
	供給後續 BatchGetItem 操作)。如需詳細資訊，請參閱上述 RequestItems 參數。 類型：陣列
UnprocessedKeys ：Table: Keys	定義項目及項目相關聯屬性的主索引鍵屬性值。如需主索引鍵的詳細資訊，請參閱 主索引鍵。 類型：屬性名稱值組陣列。
UnprocessedKeys ：Table: AttributesToGet	指定資料表中的屬性名稱。如果未指定屬性名稱，則會傳回所有屬性。如果有部分屬性未找到，則這些屬性不會出現在結果中。 類型：屬性名稱陣列。
UnprocessedKeys ：Table: ConsistentRead	如果設定為 true，則指定資料表會使用一致性讀取，反之則會使用最終一致讀取。 類型：布林值。

特殊錯誤

錯誤	描述
ProvisionedThroughputExceededException	已超出允許的佈建輸送量上限。

範例

下列範例顯示使用 BatchGetItem 操作的 HTTP POST 請求和回應。如需使用 AWS 開發套件的範例，請參閱 在 DynamoDB 使用項目和屬性。

請求範例

下列範例請求兩個不同資料表的屬性。

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0
content-length: 409

{"RequestItems":
    {"comp1":
        {"Keys":
            [{"HashKeyElement":{"S":"Casey"},"RangeKeyElement":{"N":"1319509152"}},
            {"HashKeyElement":{"S":"Dave"},"RangeKeyElement":{"N":"1319509155"}},
            {"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"1319509158"}}],
        "AttributesToGet":["user","status"]},
    "comp2":
        {"Keys":
            [{"HashKeyElement":{"S":"Julie"}},{"HashKeyElement":{"S":"Mingus"}}],
        "AttributesToGet":["user","friends"]}
    }
}
```

回應範例

下列範例為回應。

```
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**
>
> ***################## API ## 2011-12-05#***
>
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

您可利用此操作，在單一呼叫中放入與刪除多個資料表的數個項目。

您可以使用 `PutItem` 上傳一個項目，並且使用 `DeleteItem` 刪除一個項目。不過，當您想上傳或刪除大量資料，例如從 Amazon EMR (Amazon EMR) 上傳大量資料，或將資料從另一個資料庫遷移至 DynamoDB，`BatchWriteItem` 提供了一種高效率的替代方法。

如果您使用 Java 之類的語言，則可以使用執行緒來平行上傳項目。這增加了應用程序處理執行緒的複雜性。其他語言不支持執行緒。例如，如果使用 PHP，則必須一次上傳或刪除一個項目。在這兩種情況下，`BatchWriteItem` 提供一種替代方法來平行處理指定的放入與刪除操作，讓您運用強大的執行緒集區方法，卻不會增加應用程式的複雜性。

請注意，`BatchWriteItem` 操作中指定的每一個放入和刪除會使用相同的容量單位。不過，由於 `BatchWriteItem` 平行執行指定操作，您會達到更低的延遲。刪除不存在項目上的操作會使用一個寫入容量單位。如需使用的容量單位的詳細資訊，請參閱 在 DynamoDB 中使用資料表和資料。

使用 `BatchWriteItem` 時，請注意以下限制：

- 單次請求中的操作上限：最多可以指定總共 25 項放入或刪除操作，不過請求大小總計不得超過 1 MB (HTTP 承載)。
- `BatchWriteItem` 操作僅可用於放入和刪除項目。您無法使用該操作來更新現有項目。
- 不是非敗即成操作：`BatchWriteItem` 當中指定的個別操作為非敗即成操作，不過 `BatchWriteItem` 整體而言是屬於最佳操作，而不是非敗即成操作。也就是說，在 `BatchWriteItem` 請求中，有些操作可能成功，有些操作可能失敗。失敗的操作會在回應中的 `UnprocessedItems` 欄位傳回。其中有些失敗可能肇因於超出為資料表所設定的佈建輸送量，或是例如網路錯誤的暫時性失敗。您可以調查並選擇性地重新傳送請求。一般而言，您會在迴圈中

呼叫 BatchWriteItem，在每次更替中檢查未處理項目，並提交一個含有這些未處理項目的新的 BatchWriteItem 請求。

- 不傳回任何項目：BatchWriteItem 是設計來有效地上傳大量資料。這不會提供 PutItem 和 DeleteItem 的一些複雜性。例如，DeleteItem 支援要求主體文中的 ReturnValues 欄位請求回應中的刪除項目。BatchWriteItem 操作不會傳回回應中的任何項目。
- 不同於 PutItem 和 DeleteItem，BatchWriteItem 不允許在操作中指定個別寫入請求的條件。
- 屬性值不可為 Null；字串和二進位類型屬性的長度必須大於零；集合類型屬性不可為空白。具有空值的請求會遭到拒絕，並出現 ValidationException。

如果符合以下任一情況，DynamoDB 會拒絕整個批次寫入操作：

- BatchWriteItem 請求中指定的一或多個資料表不存在。
- 請求中的項目上指定的主索引鍵屬性不符合對應資料表的主索引鍵結構描述。
- 嘗試在同一 BatchWriteItem 請求中的同一個項目上執行多項操作。例如，您不能在同一 BatchWriteItem 請求中放入和刪除同一個項目。
- 請求大小總計超過 1 MB 的請求大小 (HTTP 承載) 限制。
- 批次中任一個項目超過 64 KB 的項目大小限制。

請求

語法

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
  "RequestItems" :  RequestItems
}

RequestItems
{
     "TableName1" :  [ Request, Request, ... ],
     "TableName2" :  [ Request, Request, ... ],
     ...
}
```

```
Request ::=
  PutRequest | DeleteRequest

PutRequest ::=
{
  "PutRequest" : {
     "Item" : {
        "Attribute-Name1" : Attribute-Value,
        "Attribute-Name2" : Attribute-Value,
         ...
     }
  }
}

DeleteRequest ::=
{
   "DeleteRequest" : {
      "Key" : PrimaryKey-Value
   }
}


PrimaryKey-Value ::= HashTypePK | HashAndRangeTypePK

HashTypePK ::=
{
   "HashKeyElement" : Attribute-Value
}

HashAndRangeTypePK
{
   "HashKeyElement" : Attribute-Value,
   "RangeKeyElement" : Attribute-Value,
}

Attribute-Value ::= String | Numeric| Binary | StringSet | NumericSet | BinarySet

Numeric ::=
{
   "N": "Number"
}

String ::=
{
```

```
   "S": "String"
}

Binary ::=
{
    "B": "Base64 encoded binary data"
}

StringSet ::=
{
   "SS": [ "String1", "String2", ... ]
}

NumberSet ::=
{
   "NS": [ "Number1", "Number2", ... ]
}

BinarySet ::=
{
   "BS": [ "Binary1", "Binary2", ... ]
}
```

要求主體中的 RequestItems JSON 物件會描述您要執行的操作。這些操作依資料表分組。您可以使用 BatchWriteItem 來更新或刪除多個資料表的數個項目。對於每個特定寫入請求，您必須確定請求類型 (PutItem、DeleteItem)，後面附上操作詳細資訊。

- 對於 PutRequest，您要提供項目，也就是屬性及其值的清單。
- 對於 DeleteRequest，您要提供主索引鍵名稱和值。

回應

語法

以下是回應中傳回的 JSON 內文語法。

```
{
  "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.

特殊錯誤

沒有此操作特定的錯誤。

範例

以下範例顯示 BatchWriteItem 操作的 HTTP POST 請求和回應。該請求指定在 Reply (回覆) 和 Thread (主題) 資料表上執行下列操作：

- 在 Reply (回覆) 資料表中放入一個項目並刪除一個項目
- 將一個項目放入 Thread (主題) 資料表

如需使用 AWS 開發套件的範例，請參閱 在 DynamoDB 使用項目和屬性。

請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
  "RequestItems":{
    "Reply":[
      {
        "PutRequest":{
          "Item":{
            "ReplyDateTime":{
              "S":"2012-04-03T11:04:47.034Z"
            },
            "Id":{
              "S":"DynamoDB#DynamoDB Thread 5"
            }
```

```
          }
        }
      },
      {
        "DeleteRequest":{
          "Key":{
            "HashKeyElement":{
              "S":"DynamoDB#DynamoDB Thread 4"
            },
            "RangeKeyElement":{
              "S":"oops - accidental row"
            }
          }
        }
      }
    ],
    "Thread":[
      {
        "PutRequest":{
          "Item":{
            "ForumName":{
              "S":"DynamoDB"
            },
            "Subject":{
              "S":"DynamoDB Thread 5"
            }
          }
        }
      }
    ]
  }
}
```

回應範例

下列範例回應顯示 Thread (主題) 和 Reply (回覆) 資料表上的放入操作成功，以及 Reply (回覆) 資料表上的刪除操作失敗 (例如因超出資料表上的佈建輸送量而導致調節等原因)。在 JSON 回應中，請注意下列事項：

- `Responses` 物件會顯示 `Thread` 和 `Reply` 資料表兩者都成功執行放入操作而使用一個容量單位。
- `UnprocessedItems` 物件則會顯示 `Reply` 資料表失敗的刪除操作。然後，您可以發出新的 `BatchWriteItem` 呼叫來解決這些未處理的請求。

```
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**
>
> ***################## API ## 2011-12-05#***
>
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

CreateTable 操作會將新的資料表新增到帳戶。

在與發出請求的 AWS 帳戶和接收請求的 AWS 區域 (例如 dynamodb.us-west-2.amazonaws.com) 相關聯的資料表當中，資料表名稱必須是唯一的。每個 DynamoDB 端點都是完全獨立的。例如，如果有兩個名為 MyTable 的資料表，一個在 dynamodb.us-west-2.amazonaws.com 中，另一個在 dynamodb.us-west-1.amazonaws.com 中，則兩者完全獨立，不會共用任何資料。

CreateTable 操作會觸發非同步工作流程以開始建立資料表。DynamoDB 會立即傳回資料表狀態 (CREATING)，直到資料表處於 ACTIVE 狀態。一旦資料表處於 ACTIVE 狀態，您便可執行資料平面操作。

使用 DescribeTables 操作來檢查資料表的狀態。

請求

語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

名稱	描述	必要
TableName	要建立的資料表名稱。 允許的字元為 a-z、A-Z、0-9、_ (底線)、- (破折號) 和 . (點)。名稱長度可介於 3 到 255 個字元之間。	是

名稱	描述	必要
	類型：字串	
KeySchema	資料表的主索引鍵 (簡單或複合) 結構。需要 HashKeyElement 的名稱值組，且可選用 RangeKeyElement 的名稱值組 (僅適用於複合主索引鍵)。如需主索引鍵的詳細資訊，請參閱 主索引鍵。 主索引鍵元素名稱長度可以介於 1 到 255 個字元之間，且沒有字元限制。 AttributeType 的可能值為「S」(字串)、「N」(數字) 或「B」(二進位)。 類型：複合主索引鍵的 HashKeyElement 或 HashKeyElement 和 RangeKeyElement 映射。	是

名稱	描述	必要
ProvisionedThroughput	指定資料表的新輸送量，包括 ReadCapacityUnits 與 WriteCapacityUnits 的值。如需詳細資訊，請參閱 DynamoDB 佈建容量模式。 **Note** 如需目前的最大/最小值，請參閱 Amazon DynamoDB 中的配額。 類型：陣列	是
ProvisionedThroughput：ReadCapacityUnits	設定 DynamoDB 與其他操作平衡負載之前，所指定資料表每秒所需的最低一致性 ReadCapacityUnits 數目。 最終一致讀取操作比一致性讀取負擔更輕，因此每秒 50 次一致性 ReadCapacityUnits 的設定可提供每秒 100 次最終一致 ReadCapacityUnits 。 類型：數字	是

名稱	描述	必要
ProvisionedThroughput：WriteCapacityUnits	設定 DynamoDB 與其他操作平衡負載之前，所指定資料表每秒所需的最低 WriteCapacityUnits 數目。 類型：數字	是

回應

語法

```
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"
    }
}
```

名稱	描述
TableDescription	資料表屬性的容器。
CreationDateTime	建立資料表時的 UNIX epoch 格式日期。 類型：數字
KeySchema	資料表的主索引鍵 (簡單或複合) 結構。需要 HashKeyElement 的名稱值組，且可選用

名稱	描述
	RangeKeyElement 的名稱值組 (僅適用於複合主索引鍵)。如需主索引鍵的詳細資訊，請參閱 主索引鍵。 類型：複合主索引鍵的 HashKeyElement 或 HashKeyElement 和 RangeKeyElement 映射。
ProvisionedThroughput	指定資料表的輸送量，包括 ReadCapacityUnits 與 WriteCapacityUnits 的值。請參閱 DynamoDB 佈建容量模式。 類型：陣列
ProvisionedThroughput :ReadCapacityUnits	DynamoDB 與其他操作平衡負載之前，每秒所需的最低 ReadCapacityUnits 數目。 類型：數字
ProvisionedThroughput :WriteCapacityUnits	WriteCapacityUnits 與其他操作平衡負載之前，每秒所需的最低 ReadCapacityUnits 數目。 類型：數字
TableName	已建立的資料表名稱。 類型：字串
TableStatus	資料表目前的狀態 (CREATING)。一旦資料表處於 ACTIVE 狀態，您便可將資料放入其中。 使用 DescribeTables API 來檢查資料表狀態。 類型：字串

特殊錯誤

錯誤	描述
ResourceInUseException	嘗試重新建立已存在的資料表。
LimitExceededException	同步資料表請求數 (CREATING、DELETING 或 UPDATING 狀態) 超出所允許的上限。 **Note** 如需目前的最大/最小值，請參閱 Amazon DynamoDB 中的配額。 .

範例

以下範例會使用包含一個字串和數字的複合主索引鍵建立資料表。如需使用 AWS 開發套件的範例，請參閱 在 DynamoDB 中使用資料表和資料。

請求範例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0


{"TableName":"comp-table",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

回應範例

```
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"
    }
}
```

相關動作

- DescribeTables
- DeleteTable

DeleteItem

> **Important**
>
> ***################## API ## 2011-12-05#***
>
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

依主索引鍵，刪除資料表中的單一項目。您可以執行條件式刪除操作，在項目存在或者項目具有預期的屬性值時予以刪除。

> **Note**
>
> 如果指定 `DeleteItem` 而沒有屬性或值，則會刪除項目的所有屬性。

在未指定條件的情況下，DeleteItem 一律為等冪操作，在同一個項目或屬性上多次執行不會導致錯誤回應。
條件式刪除僅能用於在符合特定條件時刪除項目和屬性。如果符合條件，DynamoDB 會執行刪除。反之則不會刪除項目。
您可以在每次操作對一個屬性執行預期的條件式檢查。

請求

語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},"RangeKeyElement":
{"N":"AttributeValue2"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3"}}},
    "ReturnValues":"ALL_OLD"}
}
```

名稱	描述	必要
TableName	包含要刪除項目的資料表名稱。 類型：字串	是
Key	定義項目的主索引鍵。如需主索引鍵的詳細資訊，請參閱 主索引鍵。 類型：HashKeyElement 對其值的映射和 RangeKeyElement 對其值的映射。	是

名稱	描述	必要
Expected	指定條件式刪除的屬性。Expected 參數可讓您提供屬性名稱，以及 DynamoDB 是否應該檢查屬性值在刪除前是否具有特定值。 類型：屬性名稱映射。	否
Expected:AttributeName	條件式放置的屬性的名稱。 類型：字串	否

名稱	描述	必要

`Expected:AttributeName: ExpectedAttributeValue`	使用此參數來指定屬性名稱值組的值是否已經存在。
	
	如果該項目的 Color (顏色) 屬性不存在，則下列 JSON 符號會刪除項目：
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	下列 JSON 符號會在刪除項目之前檢查名為 Color (顏色) 的屬性是否具有現有值 Yellow (黃色)：
	
	`"Expected" : {"Color":{"Exists":true},{"Value":{"S":"Yellow"}}}`
	
	根據預設，如果使用 `Expected` 參數並提供 `Value`，DynamoDB 會假設屬性存在，且有要取代的目前值。所以您不必指定 `{"Exists":true}` ，因為這是暗含的。您可以將請求縮短為：
	
	`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}`
		否

名稱	描述	必要
	Note 如果指定 {"Exists" :true} 而無要檢查的屬性值，則 DynamoDB 會傳回錯誤。	
ReturnValues	如果要在刪除前取得屬性名稱值組，請使用此參數。可能的參數值為 NONE (預設) 或 ALL_OLD。如果指定 ALL_OLD，則會傳回舊項目的內容。如果未提供此參數，或者參數為 NONE，則不會傳回任何內容。 類型：字串	否

回應

語法

```
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
}
```

名稱	描述
Attributes	如果 ReturnValues 參數在請求中提供為 ALL_OLD，則 DynamoDB 會傳回屬性名稱值組陣列 (本質上為已刪除項目)。反之則會在回應包含空集合。 類型：屬性名稱值組陣列。
ConsumedCapacityUnits	操作所使用的寫入容量單位數目。此值顯示套用至佈建輸送量的數字。刪除不存在項目上的請求會使用 1 個寫入容量單位。如需更多資訊，請參閱DynamoDB 佈建容量模式。 類型：數字

特殊錯誤

錯誤	描述
ConditionalCheckFailedException	條件式檢查失敗。找不到預期的屬性值。

範例

請求範例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"comp-table",
    "Key":
        {"HashKeyElement":{"S":"Mingus"},"RangeKeyElement":{"N":"200"}},
    "Expected":
        {"status":{"Value":{"S":"shopping"}}},
    "ReturnValues":"ALL_OLD"
```

```
}
```

回應範例

```
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
}
```

相關動作

- PutItem

DeleteTable

> **Important**
>
> ***################## API ## 2011-12-05#***
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

DeleteTable 操作會刪除資料表及其所有項目。在發出 DeleteTable 請求後，指定的資料表會處於 DELETING 狀態，直到 DynamoDB 完成刪除為止。如果資料表處於 ACTIVE 狀態，則可予以刪除。如果資料表處於 CREATING 或 UPDATING 狀態，則 DynamoDB 會傳回 ResourceInUseException 錯誤。如果指定的資料表不存在，則 DynamoDB 會傳回 ResourceNotFoundException。如果資料表已處於 DELETING 狀態，則不會傳回任何錯誤。

> **Note**
>
> DynamoDB 可能會繼續接受 DELETING 狀態的資料表的資料平面操作請求 (例如 GetItem 和 PutItem)，直到資料表刪除完成。

在與發出請求的 AWS 帳戶和接收請求的 AWS 區域 (例如 dynamodb.us-west-1.amazonaws.com) 相關聯的資料表當中，資料表必須是唯一的。每個 DynamoDB 端點都是完全獨立的。例如，如果有兩個名為 MyTable 的資料表，一個在 dynamodb.us-west-2.amazonaws.com 中，另一個在 dynamodb.us-west-1.amazonaws.com 中，則兩者完全獨立，不會共用任何資料；刪除一個資料表並不會同時刪除另一個。

使用 DescribeTables 操作來檢查資料表的狀態。

請求

語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

名稱	描述	必要
TableName	要刪除的資料表的名稱。 類型：字串	是

回應

語法

```
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"
    }
}
```

名稱	描述
TableDescription	資料表屬性的容器。
CreationDateTime	建立資料表時的日期。 類型：數字
KeySchema	資料表的主索引鍵 (簡單或複合) 結構。需要 HashKeyElement 的名稱值組，且可選用 RangeKeyElement 的名稱值組 (僅適用於複合主索引鍵)。如需主索引鍵的詳細資訊，請參閱 主索引鍵。 類型：複合主索引鍵的 HashKeyElement 或 HashKeyElement 和 RangeKeyElement 映射。
ProvisionedThroughput	指定資料表的輸送量，包括 ReadCapacityUnits 與 WriteCapacityUnits 的值。請參閱 DynamoDB 佈建容量模式。
ProvisionedThroughput ：ReadCapacityUnits	DynamoDB 與其他操作平衡負載之前，所指定資料表每秒所需的最低 ReadCapacityUnits 數目。

名稱	描述
	類型：數字
ProvisionedThroughput：WriteCapacityUnits	DynamoDB 與其他操作平衡負載之前，所指定資料表每秒所需的最低 WriteCapacityUnits 數目。 類型：數字
TableName	已刪除的資料表的名稱。 類型：字串
TableStatus	資料表目前的狀態 (DELETING)。一旦資料表遭刪除，後續對資料表的請求會傳回 resource not found。 使用 DescribeTables 操作來檢查資料表的狀態。 類型：字串

特殊錯誤

錯誤	描述
ResourceInUseException	資料表處於狀態 CREATING 或 UPDATING，而且無法刪除。

範例

請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0
content-length: 40
```

```
{"TableName":"favorite-movies-table"}
```

回應範例

```
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"
}
```

相關動作

- CreateTable
- DescribeTables

DescribeTables

> **Important**
>
> ***################## API ## 2011-12-05#***
>
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

傳回資料表相關資訊，包括資料表目前狀態、主索引鍵結構描述以及建立資料表的時間。DescribeTable 結果為最終一致性。如果在建立資料表過程中太早使用 DescribeTable，DynamoDB 會傳回 `ResourceNotFoundException`。如果在更新資料表過程中太早使用 DescribeTable，新值可能無法立即使用。

請求

語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

名稱	描述	必要
TableName	要描述的資料表的名稱。 類型：字串	是

回應

語法

```
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"
    }
```

```
}
```

名稱	描述
Table	正在描述的資料表的容器。 類型：字串
CreationDateTime	建立資料表時的 UNIX epoch 格式日期。
ItemCount	指定資料表中的項目數。DynamoDB 大約每六個小時會更新一次此值。此值可能不會反映最近的變更。 類型：數字
KeySchema	資料表的主索引鍵 (簡單或複合) 結構。需要 HashKeyElement 的名稱值組，且可選用 RangeKeyElement 的名稱值組 (僅適用於複合主索引鍵)。雜湊索引鍵大小上限為 2048 個位元組。範圍索引鍵大小上限為 1024 個位元組。此兩項限制皆為分別執行 (意即可出現雜湊 + 範圍 2048 + 1024 位元的合併索引鍵)。如需主索引鍵的詳細資訊，請參閱 主索引鍵。
ProvisionedThroughput	指定資料表的輸送量，包含 LastIncreaseDateTime (如適用)、LastDecreaseDateTime (如適用)、ReadCapacityUnits 和 WriteCapacityUnits 的值。如果資料表輸送量從未增加或減少，則 DynamoDB 不會傳回這些元素的值。請參閱 DynamoDB 佈建容量模式。 類型：陣列
TableName	所請求資料表的名稱。 類型：字串

名稱	描述
TableSizeBytes	所指定資料表的總大小 (以位元組為單位)。DynamoDB 大約每六個小時會更新一次此值。此值可能不會反映最近的變更。 類型：數字
TableStatus	資料表目前的狀態 (CREATING、ACTIVE、DELETING 或 UPDATING)。一旦資料表處於 ACTIVE 狀態，您便可新增資料。

特殊錯誤

沒有此操作特定的錯誤。

範例

下列範例會顯示對名為 comp-table 的資料表使用 DescribeTable 操作的 HTTP POST 請求和回應。資料表具有複合主索引鍵。

請求範例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"users"}
```

回應範例

```
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"
    }
}
```

相關動作

- CreateTable
- DeleteTable
- ListTables

GetItem

> **Important**
>
> ***################## API ## 2011-12-05#***
>
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

GetItem 操作會傳回與主索引鍵相符的項目的一組 Attributes。如果沒有符合的項目，則 GetItem 不會傳回任何資料。

GetItem 操作預設會執行最終一致讀取。如果應用程式不接受最終一致讀取，則請使用 ConsistentRead。此操作可能比標準讀取需要更長的時間，但一定會傳回上次更新的值。如需更多詳細資訊，請參閱 DynamoDB 讀取一致性。

請求

語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
  "Key":
  {"HashKeyElement": {"S":"AttributeValue1"},
  "RangeKeyElement": {"N":"AttributeValue2"}
 },
 "AttributesToGet":["AttributeName3","AttributeName4"],
 "ConsistentRead":Boolean
}
```

名稱	描述	必要
TableName	包含所請求項目的資料表的名稱。 類型：字串	是
Key	定義項目的主索引鍵值。如需主索引鍵的詳細資訊，請參閱主索引鍵。 類型：HashKeyElement 對其值的映射和 RangeKeyElement 對其值的映射。	是
AttributesToGet	屬性名稱陣列。如果未指定屬性名稱，則會傳回所有屬性。如果有部分屬性未找到，則這些屬性不會出現在結果中。 類型：陣列	否

名稱	描述	必要
ConsistentRead	如果設定為 true，則會發送一致性讀取，反之則會使用最終一致性。 類型：布林值	否

回應

語法

```
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
}
```

名稱	描述
Item	包含所請求屬性。 類型：屬性名稱值組映射。
ConsumedCapacityUnits	操作所使用的讀取容量單位數目。此值顯示套用至佈建輸送量的數字。請求不存在項目會使用最小讀取容量單位，具體值依讀取類型而定。如需更多資訊，請參閱DynamoDB 佈建容量模式。 類型：數字

特殊錯誤

沒有此操作特定的錯誤。

範例

如需使用 AWS 開發套件的範例，請參閱 在 DynamoDB 使用項目和屬性。

請求範例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"comptable",
 "Key":
  {"HashKeyElement":{"S":"Julie"},
  "RangeKeyElement":{"N":"1307654345"}},
 "AttributesToGet":["status","friends"],
 "ConsistentRead":true
}
```

回應範例

請注意，ConsumedCapacityUnits 值為 1，因為選用參數 ConsistentRead 已設定為 true。如果 ConsistentRead 已設定為 false (或未指定)，則回應為最終一致性，且 ConsumedCapacityUnits 數值為 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**
>
> ***################## API ## 2011-12-05#***
>
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

傳回與目前帳戶及端點相關聯之所有資料表陣列。

每個 DynamoDB 端點都是完全獨立的。例如，如果有兩個名為 MyTable 的資料表，一個在 dynamodb.us-west-2.amazonaws.com 中，另一個在 dynamodb.us-east-1.amazonaws.com 中，則兩者完全獨立，不會共用任何資料。ListTables 操作會針對接收請求的端點，傳回與發出請求之帳戶相關聯的所有資料表名稱。

請求

語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"Table1","Limit":3}
```

根據預設，ListTables 操作會針對接收請求的端點，請求與發出請求之帳戶相關聯的所有資料表名稱。

名稱	描述	必要
`Limit`	要傳回的資料表名稱上限數。 類型：整數	否
`ExclusiveStartTabl eName`	清單之首的資料表名稱。如果您已執行 ListTables 操作並收	否

名稱	描述	必要
	到 LastEvaluatedTableName 值，請在此處使用該值繼續清單。 類型：字串	

回應

語法

```
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"}
```

名稱	描述
TableNames	在目前端點上與目前帳戶相關聯的資料表名稱。 類型：陣列
LastEvaluatedTableName	目前清單中最後一個資料表的名稱，這只在帳戶和端點的部分資料表尚未傳回時才需要。如果已傳回所有資料表名稱，則回應中不存在此值。在新的請求中使用此值作為 ExclusiveStartTableName 來繼續清單，直到傳回所有資料表名稱。 類型：字串

特殊錯誤

沒有此操作特定的錯誤。

範例

下列範例顯示使用 ListTables 操作的 HTTP POST 請求和回應。

請求範例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"comp2","Limit":3}
```

回應範例

```
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"]}
```

相關動作

- DescribeTables
- CreateTable
- DeleteTable

PutItem

> **Important**
>
> ***################## API ## 2011-12-05#***
>
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

建立新項目，或以新項目取代舊項目 (包括全部屬性)。如果具有相同主索引鍵的指定資料表中已存在某項目，則新項目會完全取代現有項目。您可以執行條件式放置 (如果具有所指定主索引鍵的項目不存在，則插入新項目)，或者在現有項目具有某些屬性值時取代項目。

屬性值不得為 Null；字串和二進位類型屬性的長度必須大於零；集合類型屬性不可為空白。具有空值的請求會遭到拒絕，並出現 `ValidationException`。

> **Note**
>
> 若要確保新項目不會取代現有項目，請使用條件式放置操作，針對主索引鍵屬性或屬性將 `Exists` 設定為 `false`。

如需有關使用 `PutItem` 的詳細資訊，請參閱 在 DynamoDB 使用項目和屬性。

請求

語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Item":{
        "AttributeName1":{"S":"AttributeValue1"},
        "AttributeName2":{"N":"AttributeValue2"},
        "AttributeName5":{"B":"dmFsdWU="}
    },
    "Expected":{"AttributeName3":{"Value": {"S":"AttributeValue"}, "Exists":Boolean}},
    "ReturnValues":"ReturnValuesConstant"}
```

名稱	描述	必要
`TableName`	要包含項目的資料表名稱。 類型：字串	是

名稱	描述	必要
Item	項目屬性映射，且必須包含定義項目的主索引鍵值。可提供其他屬性名稱值組給項目。如需主索引鍵的詳細資訊，請參閱 主索引鍵。 類型：屬性值的屬性名稱映射。	是
Expected	指定條件式放置的屬性。Expected 參數可讓您提供屬性名稱，以及 DynamoDB 是否應該檢查屬性值是否已存在；或者屬性值是否存在且在變更之前有特定值。 類型：屬性值的屬性名稱映射，以及映射是否存在。	否
Expected:AttributeName	條件式放置的屬性的名稱。 類型：字串	否

名稱	描述	必要

`Expected:AttributeName: ExpectedAttributeValue`	使用此參數來指定屬性名稱值組的值是否已經存在。
	
	如果該項目的 Color (顏色) 屬性尚未存在，則下列 JSON 符號會取代項目：
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	下列 JSON 符號會在取代項目之前檢查名為 Color (顏色) 的屬性是否具有現有值 Yellow (黃色)：
	
	`"Expected" : {"Color":{"Exists":true,{"Value":{"S":"Yellow"}}}`
	
	根據預設，如果使用 `Expected` 參數並提供 `Value`，DynamoDB 會假設屬性存在，且有要取代的目前值。所以您不必指定 `{"Exists":true}` ，因為這是暗含的。您可以將請求縮短為：
	
	`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}`
		否

名稱	描述	必要
	Note 如果指定 {"Exists":true} 而無要檢查的屬性值，則 DynamoDB 會傳回錯誤。	
ReturnValues	如果想在以 PutItem 請求更新屬性名稱值組之前取得屬性名稱值組，則請使用此參數。可能的參數值為 NONE (預設) 或 ALL_OLD。如果已指定 ALL_OLD，且 PutItem 覆寫屬性名稱值組，則會傳回舊項目的內容。如果未提供此參數，或者參數為 NONE，則不會傳回任何內容。 類型：字串	否

回應

語法

下列語法範例假設請求指定 ALL_OLD 的 ReturnValues 參數，反之則回應只有 ConsumedCapacityUnits 元素。

```
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
}
```

名稱	描述
Attributes	執行放置操作之前的屬性值，但僅限於請求中指定 ReturnValues 參數為 ALL_OLD 的情況。 類型：屬性名稱值組映射。
ConsumedCapacityUnits	操作所使用的寫入容量單位數目。此值顯示套用至佈建輸送量的數字。如需更多資訊，請參閱DynamoDB 佈建容量模式。 類型：數字

特殊錯誤

錯誤	描述
ConditionalCheckFailedException	條件式檢查失敗。找不到預期的屬性值。
ResourceNotFoundException	找不到指定的項目或屬性。

範例

如需使用 AWS 開發套件的範例，請參閱 在 DynamoDB 使用項目和屬性。

請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
```

```
 "Item":
  {"time":{"N":"300"},
  "feeling":{"S":"not surprised"},
  "user":{"S":"Riley"}
  },
 "Expected":
  {"feeling":{"Value":{"S":"surprised"},"Exists":true}}
 "ReturnValues":"ALL_OLD"
}
```

回應範例

```
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
}
```

相關動作

- UpdateItem
- DeleteItem
- GetItem
- BatchGetItem

Query

> **Important**
>
> ***################## API ## 2011-12-05#***
>
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

Query 操作以主索引鍵獲取一或多個項目的值及其屬性 (Query 僅適用於雜湊與範圍主索引鍵資料表)。您必須提供特定的 HashKeyValue，且能在主索引鍵的 RangeKeyValue 上使用比較運算子縮小查詢範圍。使用 ScanIndexForward 參數以範圍索引鍵取得正向或反向順序的結果。

未傳回結果的查詢會根據讀取類型使用最小讀取容量單位。

> **Note**
>
> 如果符合查詢參數的項目總數超過 1MB 限制，則查詢會停止，並將結果傳回給使用者，附帶 LastEvaluatedKey 用以在後續操作中繼續查詢。查詢操作不同於掃描操作，一律不會傳回空的結果集和 LastEvaluatedKey。只有在結果超過 1MB，或者您已使用 Limit 參數時，才會提供 LastEvaluatedKey。
>
> 使用 ConsistentRead 參數可設定一致性讀取的結果。

請求

語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
 "Limit":2,
 "ConsistentRead":true,
 "HashKeyValue":{"S":"AttributeValue1":},
 "RangeKeyCondition": {"AttributeValueList":
[{"N":"AttributeValue2"}],"ComparisonOperator":"GT"}
 "ScanIndexForward":true,
 "ExclusiveStartKey":{
  "HashKeyElement":{"S":"AttributeName1"},
  "RangeKeyElement":{"N":"AttributeName2"}
 },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

名稱	描述	必要
TableName	包含請求項目的資料表的名稱。 類型：字串	是
AttributesToGet	屬性名稱陣列。如果未指定屬性名稱，則會傳回所有屬性。如果有部分屬性未找到，則這些屬性不會出現在結果中。 類型：陣列	否
Limit	要傳回的項目數上限 (不一定是相符項目數)。如果 DynamoDB 在查詢資料表時處理的項目數達上限，則會停止查詢，並傳回截至該時間點的相符值，附帶 LastEvaluatedKey 以套用於後續操作中繼續查詢。此外，如果在 DynamoDB 達到此限制之前結果集大小超過 1MB，則會停止查詢並傳回相符值，附帶 LastEvaluatedKey 以套用於後續操作中繼續查詢。 類型：數字	否
ConsistentRead	如果設定為 true，則會發送一致性讀取，反之則會使用最終一致性。 類型：布林值	否
Count	如果設定為 true，則 DynamoDB 會傳回符合查詢參數的項目總數，而不是相符項	否

名稱	描述	必要
	目及其屬性的清單。您可以套用 Limit 參數至僅計數查詢。 請勿在提供 AttributesToGet 清單之時將 Count 設定為 true，否則 DynamoDB 會傳回驗證錯誤。如需詳細資訊，請參閱 計算結果中的項目。 類型：布林值	
HashKeyValue	複合主索引鍵雜湊元件的屬性值。 類型：字串、數字或二進位	是
RangeKeyCondition	屬性值容器以及用於查詢的比較運算子。查詢請求不需要 RangeKeyCondition 。如果只提供 HashKeyValue ，則 DynamoDB 會傳回具有指定雜湊索引鍵元素值的所有項目。 類型：映射	否

名稱	描述	必要
RangeKeyCondition : AttributeValueList	查詢參數要評估的屬性值。除非已指定 BETWEEN 比較，否則 AttributeValueList 一律包含一個屬性值。對於 BETWEEN 比較，AttributeValueList 包含兩個屬性值。 類型：ComparisonOperator 的 AttributeValue 映射。	否

名稱	描述	必要
RangeKeyCondition : ComparisonOperator	用於評估所提供的屬性的條件，例如等於、大於。以下是查詢操作的有效比較運算子。 **Note** 大於、等於或小於的字串值比較是根據 ASCII 字元代碼值。例如，a 大於 A，aa 大於 B。如需代碼值的清單，請參閱 http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters。針對 Binary，每當 DynamoDB 比較二進位值，例如在評估查詢表達式時，都會將二進位資料的每個位元組視為不帶正負號。 類型：字串或二進位	否

名稱	描述	必要
	EQ：等於。 對於 EQ，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不等於 {"NS":["6", "2", "1"]}。	
	LE：小於或等於。 對於 LE，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。	

名稱	描述	必要
	LT：小於。 對於 LT，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。	
	GE：大於或等於。 對於 GE，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。	

名稱	描述	必要
	GT：大於。 對於 GT，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。	
	BEGINS_WITH ：檢查字首。 對於 BEGINS_WITH ，AttributeValueList 僅可包含 String 或 Binary 類型 (非 Number 或集合) 的一個 AttributeValue 。比較的目標屬性必須是 String 或 Binary (非 Number 或集合)。	

名稱	描述	必要
	BETWEEN：大於或等於第一個數值，並且小於或等於第二個數值。 對於 BETWEEN，AttributeValueList 必須包含 String、Number 或 Binary 類型 (非集合) 的兩個 AttributeValue 元素。如果目標數值大於或等於第一個元素，並且小於或等於第二個元素，則目標屬性相符。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不與 {"N":"6"} 比較。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。	
ScanIndexForward	指定遞增或遞減的索引周遊。DynamoDB 傳回結果會反映由範圍索引鍵決定之所請求順序：如果資料類型為數字，則會依數值順序傳回結果，反之，周遊則是根據 ASCII 字元代碼值。 類型：布林值 預設為 true (遞增)。	否

名稱	描述	必要
ExclusiveStartKey	繼續先前查詢之項目的主索引鍵。如果因為結果集大小或 Limit 參數，導致查詢操作在查詢完成前中斷，較早的查詢可能提供此值為 LastEvaluatedKey 。LastEvaluatedKey 可以在新的查詢請求中傳回，以便從該時間點繼續操作。 類型：複合主索引鍵的 HashKeyElement 或 HashKeyElement 和 RangeKeyElement 。	否

回應

語法

```
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
```

```
}
```

名稱	描述
Items	符合查詢參數的項目屬性。 類型：屬性名稱映射，以及其資料類型和值。
Count	回應中的項目數。如需詳細資訊，請參閱 計算結果中的項目。 類型：數字
LastEvaluatedKey	查詢操作停止的項目的主索引鍵，包括先前的結果集。使用此值開始新操作，在新的請求中排除此值。 當整個查詢結果集完成時 (意即操作已處理「最後一頁」)，LastEvaluatedKey 為 null。 類型：複合主索引鍵的 HashKeyElement 或 HashKeyElement 和 RangeKeyElement 。
ConsumedCapacityUnits	操作所使用的讀取容量單位數目。此值顯示套用至佈建輸送量的數字。如需更多資訊，請參閱DynamoDB 佈建容量模式。 類型：數字

特殊錯誤

錯誤	描述
ResourceNotFoundException	找不到指定的資料表。

範例

如需使用 AWS 開發套件的範例，請參閱 在 DynamoDB 中查詢資料表。

請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"John"},
 "ScanIndexForward":false,
 "ExclusiveStartKey":{
  "HashKeyElement":{"S":"John"},
  "RangeKeyElement":{"S":"The Matrix"}
 }
}
```

回應範例

```
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
}
```

請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"Airplane"},
 "RangeKeyCondition":{"AttributeValueList":[{"N":"1980"}],"ComparisonOperator":"EQ"},
 "ScanIndexForward":false}
```

回應範例

```
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
}
```

相關動作

- Scan

Scan

> **Important**
>
> ***################## API ## 2011-12-05#***
>
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

Scan 操作會執行資料表完整掃描來傳回一或多個項目及其屬性。提供 ScanFilter 以取得更特定的結果。

> **Note**
>
> 如果已掃描項目總數超過 1MB 限制，則掃描會停止，並將結果傳回給使用者，附帶 LastEvaluatedKey 用以在後續操作中繼續掃描。結果還包括超出限制的項目數。掃描可能導致表內資料皆不符合篩選條件。
> 結果集為最終一致性。

請求

語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Limit": 2,
    "ScanFilter":{
        "AttributeName1":{"AttributeValueList":
[{"S":"AttributeValue"}],"ComparisonOperator":"EQ"}
    },
    "ExclusiveStartKey":{
        "HashKeyElement":{"S":"AttributeName1"},
        "RangeKeyElement":{"N":"AttributeName2"}
    },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

名稱	描述	必要
TableName	包含請求項目的資料表的名稱。	是

名稱	描述	必要
	類型：字串	
AttributesToGet	屬性名稱陣列。如果未指定屬性名稱，則會傳回所有屬性。如果有部分屬性未找到，則這些屬性不會出現在結果中。 類型：陣列	否
Limit	要評估的項目數上限 (不一定是相符的項目數)。如果 DynamoDB 在處理結果時處理的項目數達上限，則會停止工作，並傳回截至該時間點的相符值，附帶 LastEvaluatedKey 以套用於後續操作中繼續擷取項目。此外，如果在 DynamoDB 達到此限制之前已掃描的資料集大小超過 1MB，則會停止掃描，並傳回截至上限的相符值，附帶 LastEvaluatedKey 以套用於後續操作中繼續掃描。 類型：數字	否

名稱	描述	必要
Count	如果設定為 true，則 DynamoDB 會傳回掃描操作的項目總數，即使操作沒有指派篩選條件的相符項目也一樣。您可以套用限制參數至僅計數查詢。 請勿在提供 AttributesToGet 清單之時將 Count 設定為 true，否則 DynamoDB 會傳回驗證錯誤。如需詳細資訊，請參閱 計算結果中的項目。 類型：布林值	否
ScanFilter	評估掃描結果，並僅傳回所需值。多個條件會視為 AND 操作：必須符合所有條件才會納入結果中。 類型：具有比較運算子的值的屬性名稱映射。	否
ScanFilter :AttributeValueList	用來評估篩選條件掃描結果的值和條件。 類型：Condition 的 AttributeValue 映射。	否

名稱	描述	必要
ScanFilter：ComparisonOperator	用於評估所提供的屬性的條件，例如等於、大於。以下是掃描操作的有效比較運算子。 **Note** 大於、等於或小於的字串值比較是根據 ASCII 字元代碼值。例如，a 大於 A，aa 大於 B。如需代碼值的清單，請參閱 http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters。針對 Binary，每當 DynamoDB 比較二進位值，例如在評估查詢表達式時，都會將二進位資料的每個位元組視為不帶正負號。 類型：字串或二進位	否

名稱	描述	必要
	EQ：等於。 對於 EQ，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不等於 {"NS":["6", "2", "1"]}。	
	NE：不等於。 對於 NE，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不等於 {"NS":["6", "2", "1"]}。	

名稱	描述	必要
	LE：小於或等於。 對於 LE，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。	
	LT：小於。 對於 LT，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。	

名稱	描述	必要
	GE：大於或等於。 對於 GE，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。	
	GT：大於。 對於 GT，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。	
	NOT_NULL：屬性存在。	
	NULL：屬性不存在。	

名稱	描述	必要
	CONTAINS：檢查子序列，或是集合中的數值。 對於 CONTAINS，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果比較的目標屬性是字串，則操作會檢查相符子字串。如果比較的目標屬性是二進位，則操作會尋找與輸入相符的目標子序列。如果比較的目標屬性是集合 (SS、NS 或 BS)，則操作會檢查集合的成員 (不當作子字串)。	

名稱	描述	必要
	NOT_CONTAINS ：檢查子序列是否不存在，或者集合中的數值是否不存在。 對於 NOT_CONTAINS ，AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果比較的目標屬性是字串，則操作會檢查相符子字串是否不存在。如果比較的目標屬性是二進位，則操作會檢查與輸入相符的目標子序列是否不存在。如果比較的目標屬性是集合 (SS、NS 或 BS)，則操作會檢查集合的成員 (不當作子字串) 是否不存在。	
	BEGINS_WITH ：檢查字首。 對於 BEGINS_WITH ，AttributeValueList 僅可包含 String 或 Binary 類型 (非 Number 或集合) 的一個 AttributeValue 。比較的目標屬性必須是 String 或 Binary (非 Number 或集合)。	

名稱	描述	必要
	IN：檢查完全相符項目。 對於 IN，AttributeValueList 可以包含 String、Number 或 Binary 類型 (非集合) 的多個 AttributeValue 。比較的目標屬性必須為相同的類型和確切數值才相符。String 一律不與 String 集合相符。	
	BETWEEN：大於或等於第一個數值，並且小於或等於第二個數值。 對於 BETWEEN，AttributeValueList 必須包含 String、Number 或 Binary 類型 (非集合) 的兩個 AttributeValue 元素。如果目標數值大於或等於第一個元素，並且小於或等於第二個元素，則目標屬性相符。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不與 {"N":"6"} 比較。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。	

名稱	描述	必要
ExclusiveStartKey	繼續先前掃描之項目的主索引鍵。如果因為結果集大小或 Limit 參數，導致掃描操作在整個資料表掃描完之前中斷，較早的掃描可能提供此值。LastEvaluatedKey 可以在新的掃描請求中傳回，以便從該時間點繼續操作。 類型：複合主索引鍵的 HashKeyElement 或 HashKeyElement 和 RangeKeyElement 。	否

回應

語法

```
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}
```

```
}
```

名稱	描述
Items	符合操作參數的屬性的容器。 類型：屬性名稱映射，以及其資料類型和值。
Count	回應中的項目數。如需詳細資訊，請參閱 計算結果中的項目。 類型：數字
ScannedCount	套用任何篩選條件之前的已完成掃描中項目數。ScannedCount 值很高，但 Count 結果很少或沒有，表示掃描操作不足。如需詳細資訊，請參閱 計算結果中的項目。 類型：數字
LastEvaluatedKey	停止掃描操作之項目的主索引鍵。在後續掃描操作中提供此值，以便從該時間點繼續操作。 當整個掃描結果集完成時 (意即操作已處理「最後一頁」)，LastEvaluatedKey 為 null。
ConsumedCapacityUnits	操作所使用的讀取容量單位數目。此值顯示套用至佈建輸送量的數字。如需更多資訊，請參閱DynamoDB 佈建容量模式。 類型：數字

特殊錯誤

錯誤	描述
ResourceNotFoundException	找不到指定的資料表。

範例

如需使用 AWS 開發套件的範例，請參閱 掃描 DynamoDB 中的資料表。

請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable","ScanFilter":{}}
```

回應範例

```
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}
```

請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0
content-length: 125

{"TableName":"comp5",
 "ScanFilter":
  {"time":
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 }
}
```

回應範例

```
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
}
```

請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
```

```
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Limit":2,
 "ScanFilter":
  {"time":
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 },
 "ExclusiveStartKey":
  {"HashKeyElement":{"S":"Fredy"},"RangeKeyElement":{"N":"2000"}}
}
```

回應範例

```
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
}
```

相關動作

- Query
- BatchGetItem

UpdateItem

> **Important**
>
> ***################## API ## 2011-12-05#***
>
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

編輯現有項目的屬性。您可以執行條件式更新 (在新的屬性名稱值組不存在時插入新的，或者在現有名稱值組具有某些預期的屬性值時將其取代)。

> **Note**
>
> 您無法使用 UpdateItem 更新主索引鍵屬性。但您可以刪除項目並使用 PutItem 建立具有新屬性的新項目。

UpdateItem 操作包含 `Action` 參數，該參數定義如何執行更新。您可以放入、刪除或新增屬性值。

屬性值不得為 Null；字串和二進位類型屬性的長度必須大於零；集合類型屬性不可為空白。具有空值的請求會遭到拒絕，並出現 `ValidationException`。

如果現有項目具有指定的主索引鍵：

- PUT：新增指定的屬性。如果屬性存在，則會以新值取代。
- DELETE：如果未指定任何值，則會移除屬性及其值。如果有指定一組值，則所指定集合中的值將從舊集合中移除。因此，如果屬性值包含 [a、b、c]，而刪除動作包含 [a、c]，則最後屬性值為 [b]。指定值類型必須符合現有值類型。指定空集合無效。
- ADD：僅對數字使用新增動作，或者在目標屬性為集合 (包括字串集合) 時使用。如果目標屬性為單一字串值或純量二進位值，則 ADD 無法運作。指定的值會新增至數值 (現有的數值進行遞增或遞減)，或者新增為字串集合中的額外值。如果有指定一組值，則會將這些值新增至現有的集合中。例如，如果原始集合為 [1、2]，而提供的值為 [3]，則在新增操作之後，集合為 [1、2、3]，而非 [4、5]。如果已為集合屬性指定 Add 動作，且指定的屬性類型與現有集合類型不符，則會發生錯誤。

  如果對不存在的屬性使用 ADD，則會將屬性及其值新增至項目。

如果沒有項目符合所指定的主索引鍵：

- PUT：建立具有指定主索引鍵的新項目。然後新增指定屬性。
- DELETE：不進行任何動作。
- ADD：建立具有提供的主索引鍵和屬性值數字 (或一組數字) 的項目。不適用於字串或二進位類型。

> **Note**
>
> 如果對更新前不存在的項目使用 ADD 增減數值，則 DynamoDB 會以 0 為初始值。此外，如果對更新前不存在的屬性 (但項目存在) 使用 ADD 增減數值來更新項目，則 DynamoDB 會以 0 為初始值。例如，您使用 ADD 將 +3 新增至更新前不存在的屬性。DynamoDB 使用 0 為初始值，而且更新後的值為 3。

如需使用此操作的詳細資訊，請參閱 在 DynamoDB 使用項目和屬性。

請求

語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},
        "RangeKeyElement":{"N":"AttributeValue2"}},
    "AttributeUpdates":{"AttributeName3":{"Value":
{"S":"AttributeValue3_New"},"Action":"PUT"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3_Current"}}},
    "ReturnValues":"ReturnValuesConstant"
}
```

名稱	描述	必要
TableName	包含要更新項目的資料表的名稱。 類型：字串	是
Key	定義項目的主索引鍵。如需主索引鍵的詳細資訊，請參閱 主索引鍵。 類型：HashKeyElement 對其值的映射和 RangeKeyElement 對其值的映射。	是
AttributeUpdates	新值的屬性名稱映射以及更新動作。屬性名稱指定要修改的屬性，且不能包含任何主索引鍵屬性。 類型：屬性名稱映射、值，以及屬性更新動作。	
Attribute Updates :Action	指定如何執行更新。可能值：PUT (預設)、ADD 或 DELETE。語義會在 UpdateItem 描述中說明。 類型：字串 預設：PUT	否
Expected	指定條件式更新的屬性。Expected 參數可讓您提供屬性名稱，以及 DynamoDB 是否應該檢查屬性值是否已存在；或者屬性值是否存在且在變更之前有特定值。	否

名稱	描述	必要
	類型：屬性名稱映射。	
Expected:AttributeName	條件式放置的屬性的名稱。 類型：字串	否

名稱	描述	必要

`Expected:AttributeName: ExpectedAttributeValue`	使用此參數來指定屬性名稱值組的值是否已經存在。
	
	如果該項目的 Color (顏色) 屬性尚未存在，則下列 JSON 符號會更新項目：
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	下列 JSON 符號會在更新項目之前檢查名為 Color (顏色) 的屬性是否具有現有值 Yellow (黃色)：
	
	`"Expected" : {"Color":{"Exists":true},{"Value":{"S":"Yellow"}}}`
	
	根據預設，如果使用 `Expected` 參數並提供 `Value`，DynamoDB 會假設屬性存在，且有要取代的目前值。所以您不必指定 `{"Exists":true}` ，因為這是暗含的。您可以將請求縮短為：
	
	`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}`
		否

名稱	描述	必要
	Note 如果指定 {"Exists":true} 而無要檢查的屬性值，則 DynamoDB 會傳回錯誤。	
ReturnValues	如果想在以 UpdateItem 請求更新屬性名稱值組之前取得屬性名稱值組，則請使用此參數。可能的參數值為 NONE (預設) 或者 ALL_OLD、UPDATED_OLD 、ALL_NEW 或 UPDATED_NEW 。如果已指定 ALL_OLD，且 UpdateItem 覆寫屬性名稱值組，則會傳回舊項目的內容。如果未提供此參數，或者參數為 NONE，則不會傳回任何內容。如果指定 ALL_NEW，則會傳回項目新版本的所有屬性。如果指定 UPDATED_NEW ，則只會傳回已更新屬性的新版本。 類型：字串	否

回應

語法

下列語法範例假設請求指定 ALL_OLD 的 ReturnValues 參數，反之則回應只有 ConsumedCapacityUnits 元素。

```
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
}
```

名稱	描述
Attributes	屬性名稱值組映射，但僅限於 ReturnValues 參數在請求中被指定為 NONE 以外選項的情況。 類型：屬性名稱值組映射。
ConsumedCapacityUnits	操作所使用的寫入容量單位數目。此值顯示套用至佈建輸送量的數字。如需更多資訊，請參閱DynamoDB 佈建容量模式。 類型：數字

特殊錯誤

錯誤	描述
ConditionalCheckFailedException	條件式檢查失敗。屬性 ("+ name +") 值為 ("+ value +")，但預期為 ("+ expValue +")
ResourceNotFoundExceptions	找不到指定的項目或屬性。

範例

如需使用 AWS 開發套件的範例，請參閱 在 DynamoDB 使用項目和屬性。

請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
    "Key":
        {"HashKeyElement":{"S":"Julie"},"RangeKeyElement":{"N":"1307654350"}},
    "AttributeUpdates":
        {"status":{"Value":{"S":"online"},
        "Action":"PUT"}},
    "Expected":{"status":{"Value":{"S":"offline"}}},
    "ReturnValues":"ALL_NEW"
}
```

回應範例

```
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
}
```

相關動作

- PutItem
- DeleteItem

UpdateTable

> **Important**
>
> ***################## API ## 2011-12-05#***
>
> 如需目前低階 API 的文件，請參閱Amazon DynamoDB API 參考。

描述

更新特定資料表的佈建輸送量。設定資料表的輸送量可協助您管理效能，此為 DynamoDB 佈建輸送量功能的一部分。如需更多詳細資訊，請參閱 DynamoDB 佈建容量模式。

佈建輸送量數值可以根據 Amazon DynamoDB 中的配額 中所列最大值和最小值來升級或降級。

資料表必須處於 ACTIVE 狀態，這項操作才會成功。UpdateTable 為非同步操作；執行操作時，資料表處於 UPDATING 狀態。資料表處於 UPDATING 狀態時，仍具有呼叫之前的佈建輸送量。只有資料表在 UpdateTable 操作後回到 ACTIVE 狀態時，新的佈建輸送量設定才會生效。

請求

語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

名稱	描述	必要
TableName	要更新的資料表名稱。 類型：字串	是
ProvisionedThroughput	指定資料表的新輸送量，包括 ReadCapacityUnits 與	是

名稱	描述	必要
	WriteCapacityUnits 的值。請參閱 DynamoDB 佈建容量模式。 類型：陣列	
ProvisionedThroughput :ReadCapacityUnits	設定 DynamoDB 與其他操作平衡負載之前，所指定資料表每秒所需的最低一致性 ReadCapacityUnits 數目。 最終一致讀取操作比一致性讀取負擔更輕，因此每秒 50 次一致性 ReadCapacityUnits 的設定可提供每秒 100 次最終一致 ReadCapacityUnits 。 類型：數字	是
ProvisionedThroughput :WriteCapacityUnits	設定 DynamoDB 與其他操作平衡負載之前，所指定資料表每秒所需的最低 WriteCapacityUnits 數目。 類型：數字	是

回應

語法

```
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"}}
```

名稱	描述
CreationDateTime	建立資料表時的日期。 類型：數字
KeySchema	資料表的主索引鍵 (簡單或複合) 結構。需要 HashKeyElement 的名稱值組，且可選用 RangeKeyElement 的名稱值組 (僅適用於複合主索引鍵)。雜湊索引鍵大小上限為 2048 個位元組。範圍索引鍵大小上限為 1024 個位元組。此兩項限制皆為分別執行 (意即可出現雜湊 + 範圍 2048 + 1024 位元的合併索引鍵)。如需主索引鍵的詳細資訊，請參閱 主索引鍵。 類型：複合主索引鍵的 HashKeyElement 或 HashKeyElement 和 RangeKeyElement 映射。
ProvisionedThroughput	所指定資料表的目前輸送量設定，包括 LastIncreaseDateTime (如適用)、LastDecreaseDateTime (如適用) 的值。 類型：陣列

名稱	描述
TableName	已更新的資料表名稱。 類型：字串
TableStatus	資料表目前的狀態 (CREATING、ACTIVE、DELETING 或 UPDATING)，目前應該為 UPDATING。 使用 DescribeTables 操作來查看資料表狀態。 類型：字串

特殊錯誤

錯誤	描述
ResourceNotFoundException	找不到指定的資料表。
ResourceInUseException	資料表不處於 ACTIVE 狀態。

範例

請求範例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"comp1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

回應範例

```
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"}
}
```

相關動作

- CreateTable
- DescribeTables
- DeleteTable

舊版 DynamoDB 條件式參數

本文件提供 DynamoDB 中舊版條件式參數的概觀，並建議改用新的表達式參數。其涵蓋 AttributesToGet、AttributeUpdates、ConditionalOperator、Expected、KeyConditions、QueryFilter 和 ScanFilter 等參數的詳細資訊，並提供如何使用新表達式參數取代的範例。

> **Important**
>
> 我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需詳細資訊，請參閱在 DynamoDB 中使用表達式。
>
> 此外，DynamoDB 不允許在單一呼叫中混用舊式條件式參數和表達式參數。例如，使用 `AttributesToGet` 和 `ConditionExpression` 來呼叫 `Query` 操作會導致錯誤。

下表顯示仍支援這些舊版參數的 DynamoDB API 操作，以及要改用的表達式參數。若您考慮更新應用程式來改用表達式參數，此資料表會有所幫助。

如果您使用此 API 操作...	採用的是這些舊式參數...	改用此表達式參數
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

以下章節將提供舊式條件式參數的詳細資訊。

主題

- AttributesToGet (舊版)
- AttributeUpdates (舊版)
- ConditionalOperator (舊版)
- Expected (舊版)
- KeyConditions (舊版)
- QueryFilter (舊版)
- ScanFilter (舊版)

- 使用舊式參數撰寫條件

AttributesToGet (舊版)

> **Note**
>
> 我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需詳細資訊，請參閱在 DynamoDB 中使用表達式。如需取代此參數之新參數的特定資訊，請參閱 改用 ProjectionExpression。

舊版條件式參數 `AttributesToGet` 是一個陣列，包含從 DynamoDB 中擷取的一或多個屬性。如果未提供屬性名稱，則會傳回所有屬性。如果找不到請求的任一屬性，則不會在結果中顯示這些屬性。

`AttributesToGet` 可讓您擷取 List 或 Map 類型的屬性，但無法擷取 List 或 Map 中的個別元素。

請注意，`AttributesToGet` 並不會影響佈建輸送量的耗用。DynamoDB 會根據項目大小判定使用的容量單位數，而不是根據傳回給應用程式的資料量。

改用 ProjectionExpression - 範例

假設要從 Music 資料表擷取一個項目，但只想傳回部分屬性。您可以搭配 `AttributesToGet` 參數使用`GetItem`請求，如本 AWS CLI 範例所示：

```
aws dynamodb get-item \
    --table-name Music \
    --attributes-to-get '["Artist", "Genre"]' \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

您可以改用 `ProjectionExpression`。

```
aws dynamodb get-item \
    --table-name Music \
    --projection-expression "Artist, Genre" \
    --key '{
        "Artist": {"S":"No One You Know"},
```

```
        "SongTitle": {"S":"Call Me Today"}
    }'
```

AttributeUpdates (舊版)

> **Note**
>
> 我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需詳細資訊，請參閱在 DynamoDB 中使用表達式。如需取代此參數之新參數的特定資訊，請參閱 改用 UpdateExpression。。

在 UpdateItem 操作中，舊版條件式參數 AttributeUpdates 包含要修改的屬性名稱、每個屬性上要執行的動作，以及每個屬性的新值。如果您要更新的屬性是資料表上任何索引的索引鍵屬性，則其類型必須符合資料表描述的 AttributesDefinition 中所定義的索引鍵類型。您可以使用 UpdateItem 來更新任何非索引鍵屬性。

屬性值不可為 Null。String 和 Binary 類型屬性的長度必須大於零。Set 類型屬性不能為空白。具有空白數值的請求會遭到拒絕，並出現 ValidationException 例外狀況。

每個 AttributeUpdates 元素包含要修改的屬性名稱，並具有下列資訊：

- Value：此屬性的新值 (如適用)。
- Action：用以指定如何執行更新的數值。此動作僅適用於資料類型為 Number 或 Set 的現有屬性。請勿對其他資料類型使用 ADD。

  如果在資料表中找到具有指定主索引鍵的項目，則下列各數值會執行以下動作：

  - PUT：將指定屬性新增至項目。如果屬性已存在，則以新數值取代。
  - DELETE：如果未指定 DELETE 的數值，則刪除屬性及其數值。指定數值的資料類型必須符合現有數值的資料類型。

    如果指定一組數值，則會從舊的集合中減去該等數值。例如，如果屬性值是集合 [a,b,c]，而且 DELETE 動作指定 [a,c]，則最後屬性值為 [b]。指定空集合是一項錯誤。
  - ADD：如果屬性尚未存在，則將指定數值新增至項目。如果屬性已存在，則 ADD 的行為依屬性的資料類型而定：
    - 如果現有屬性是數字，而 Value 亦為數字，則 Value 會以數學方式新增至現有屬性。如果 Value 是負數，則會從現有屬性減去。

> **Note**
>
> 如果對更新前不存在的項目使用 ADD 增減數值，則 DynamoDB 會以 0 為初始數值。同樣地，如果對現有項目使用 ADD 增減更新前不存在的屬性值，則 DynamoDB 會以 0 為初始數值。例如，假設欲更新的項目不具有 itemcount 名稱的屬性，但您仍要將此屬性 ADD 數字 3。DynamoDB 會建立屬性 itemcount，將其初始數值設為 0，最後再加上 3。結果會得到數值為 3 的新 itemcount 屬性。

- 如果現有的資料類型是集合，而 Value 亦為集合，則 Value 會附加至現有集合。例如，如果屬性值是集合 [1,2]，而且 ADD 動作指定 [3]，則最後屬性值為 [1,2,3]。如果已為集合屬性指定 ADD 動作，且指定的屬性類型與現有集合類型不符，則會發生錯誤。

  兩個集合必須具有相同的基本資料類型。例如，如果現有的資料類型是一組字串，Value 也必須是一組字串。

如果未在資料表中找到具有指定索引鍵的項目，則下列各數值會執行以下動作：

- PUT：讓 DynamoDB 使用指定的主索引鍵建立新項目，然後新增屬性。
- DELETE：不進行任何動作，因為無法從不存在的項目中刪除屬性。成功完成操作，但 DynamoDB 不會建立新項目。
- ADD：讓 DynamoDB 使用為屬性值提供的主索引鍵和數字 (或一組數字) 建立項目。僅允許 Number 和 Number Set 資料類型。

如果提供屬於索引鍵一部分的任何屬性，則這些屬性的資料類型必須符合資料表屬性定義中結構描述的資料類型。

改用 UpdateExpression - 範例

假設您想修改 Music 資料表中的一個項目。您可以搭配 AttributeUpdates 參數使用 UpdateItem請求，如本 AWS CLI 範例所示：

```
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"}
        }
    }'
```

您可以改用 UpdateExpression。

```
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 (舊版)

> **Note**
>
> 我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需詳細資訊，請參閱在 DynamoDB 中使用表達式。

舊版條件式參數 ConditionalOperator 是邏輯運算子，可套用至 Expected、QueryFilter 或 ScanFilter 映射中的條件：

- AND：如果所有條件評估為 true，則整個映射評估為 true。
- OR：如果至少其中一個條件評估為 true，則整個映射評估為 true。

如果省略 ConditionalOperator，則會以 AND 為預設。

只有在整個映射評估為 true 時，操作才會成功。

> **Note**
>
> 此參數不支援 List 或 Map 類型的屬性。

Expected (舊版)

> **Note**
>
> 我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需詳細資訊，請參閱在 DynamoDB 中使用表達式。如需取代此參數之新參數的特定資訊，請參閱 改用 ConditionExpression。。

舊版條件式參數 `Expected` 是 `UpdateItem` 操作的條件式區塊。`Expected` 是屬性/條件對的映射。映射的每個元素都包含一個屬性名稱、一個比較運算子，以及一或多個數值。DynamoDB 會使用比較運算子來比較屬性與您提供的數值。對於每個 `Expected` 元素，評估結果皆為 true 或 false。

如果在 `Expected` 映射中指定一個以上元素，所有條件必須根據預設評估為 true。換句話說，條件是使用 AND 運算子的組合。(您可以改用 `ConditionalOperator` 參數將條件以 OR 連在一起。若要如此，則至少其中一個條件必須評估為 true，而不是全部都必須評估為 true。)

如果 `Expected` 映射評估為 true，則條件式操作會成功，反之則會失敗。

`Expected` 包含下列各項：

- `AttributeValueList`：針對所提供的屬性進行評估的一或多個數值。清單中值的數目依使用的 `ComparisonOperator` 而定。

  Number 類型的數值比較為數字。

  大於、等於或小於的 String 數值比較是根據 UTF-8 二進位編碼的 Unicode。例如，a 大於 A，a 大於 B。

  針對 Binary 類型，每當 DynamoDB 比較二進位值時，都會將二進位資料的每個位元組視為不帶正負號。
- `ComparisonOperator`：用於評估 `AttributeValueList` 中屬性的比較程式。執行比較時，DynamoDB 會使用高度一致性讀取。

  可以使用下列比較運算子：

  `EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

  以下是每個比較運算子的描述。

- EQ：等於。所有資料類型都支援 EQ，包括清單和映射。

  AttributeValueList 僅可包含 String、Number、Binary、String Set、Number Set 或 Binary Set 類型的一個 AttributeValue 元素。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不等於 {"NS":["6", "2", "1"]}。

- NE：不等於。所有資料類型都支援 NE，包括清單和映射。

  AttributeValueList 僅可包含 String、Number、Binary、String Set、Number Set 或 Binary Set 類型的一個 AttributeValue。如果項目內含的 AttributeValue 與所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不等於 {"NS":["6", "2", "1"]}。

- LE：小於或等於。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- LT：小於。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- GE：大於或等於。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- GT：大於。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- NOT_NULL：屬性存在。所有資料類型都支援 NOT_NULL，包括清單和映射。

> **Note**
>
> 此運算子會測試屬性是否存在，而不是其資料類型。如果屬性 a 的資料類型為 Null，並且是以 NOT_NULL 評估，則結果為布林值 true。此結果是因為屬性 a 存在；其資料類型與 NOT_NULL 比較運算子無關。

- NULL：屬性不存在。所有資料類型都支援 NULL，包括清單和映射。

> **Note**
>
> 此運算子會測試屬性是否不存在，而不是其資料類型。如果屬性 a 的資料類型為 Null，並且是以 NULL 評估，則結果為布林值 false。這是因為屬性 a 存在；其資料類型與 NULL 比較運算子無關。

- CONTAINS：檢查子序列，或是集合中的數值。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果比較的目標屬性是 String 類型，則運算子會檢查相符子字串。如果比較的目標屬性是 Binary 類型，則運算子會尋找與輸入相符的目標子序列。比較的目標屬性為集合 (SS、NS 或 BS) 時，如果找到與該集合任一成員完全相符的項目，則運算子評估為 true。

  清單支援 CONTAINS：當評估 a CONTAINS b 時，a 可以是清單，但 b 不可以是集合、映射或清單。

- NOT_CONTAINS：檢查子序列是否不存在，或者集合中的數值是否不存在。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果比較的目標屬性是 String，則運算子會檢查相符子字串是否不存在。如果比較的目標屬性是 Binary，則運算子會檢查與輸入相符的目標子序列是否不存在。比較的目標屬性為集合 (SS、NS 或 BS) 時，如果 does not 找到與該集合任一成員完全相符的項目，則運算子評估為 true。

  清單支援 NOT_CONTAINS：當評估 a NOT CONTAINS b 時，a 可以是清單，但 b 不可以是集合、映射或清單。

- BEGINS_WITH：檢查字首。

  AttributeValueList 僅可包含 String 或 Binary 類型 (非 Number 或集合類型) 的一個 AttributeValue。比較的目標屬性必須是 String 或 Binary 類型 (非 Number 或集合類型)。

- IN：檢查兩個集合內相符的元素。

  AttributeValueList 可以包含 String、Number 或 Binary 類型 (非集合類型) 的一或多個 AttributeValue 元素。這些屬性會與項目的現有集合類型屬性進行比較。如果輸入集合的任何元素存在於項目屬性中，則表達式評估為 true。

- BETWEEN：大於或等於第一個數值，並且小於或等於第二個數值。

  AttributeValueList 必須包含 String、Number 或 Binary 類型 (非集合類型) 的兩個 AttributeValue 元素。如果目標數值大於或等於第一個元素，並且小於或等於第二個元素，則目標屬性相符。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不與 {"N":"6"} 比較。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

下列參數可用來取代 AttributeValueList 和 ComparisonOperator：

- Value：DynamoDB 用來與屬性進行比較的數值。
- Exists：此為布林值，可讓 DynamoDB 在嘗試條件式操作之前評估數值：
  - 如果 Exists 為 true，則 DynamoDB 會查看該屬性值是否已存在於資料表中。如果找到該數值，則條件評估為 true，否則條件評估為 false。
  - 如果 Exists 為 false，則 DynamoDB 會假設屬性值 not 存在於資料表中。如果該數值實際上不存在，則假設有效，條件會評估為 true。如果找到該值，則即使假設該值不存在，條件仍會評估為 false。

  請注意，Exists 預設值為 true。

Value 和 Exists 參數與 AttributeValueList 和 ComparisonOperator 不相容。請注意，如果同時使用此兩組參數，DynamoDB 會傳回 ValidationException 例外狀況。

> **Note**
>
> 此參數不支援 List 或 Map 類型的屬性。

改用 ConditionExpression - 範例

假設您想只在某一條件為 true 的情況下修改 Music 資料表中的一個項目。您可以搭配 Expected 參數使用 UpdateItem請求，如本 AWS CLI 範例所示：

```
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"} ]
        }
    }'
```

您可以改用 ConditionExpression。

```
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 (舊版)

> **Note**
>
> 我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需詳細資訊，請參閱在 DynamoDB 中使用表達式。如需取代此參數之新參數的特定資訊，請參閱 改用 KeyConditionExpression。 。

舊版條件式參數 KeyConditions 包含 Query 操作的選取條件。對於資料表上的查詢，您可以僅對資料表主索引鍵屬性設定條件。您必須將分割區索引鍵名稱和值提供為 EQ 條件。您可選擇性地提供指向排序索引鍵的第二個條件。

> **Note**
>
> 如未提供排序索引鍵條件，則會擷取所有符合分割區索引鍵的項目。如果 FilterExpression 或 QueryFilter 存在，則會在擷取項目後套用。

對於索引上的查詢，您可以僅對索引鍵屬性設定條件。您必須將索引分割區索引鍵名稱和值提供為 EQ 條件。您可選擇提供指向索引排序索引鍵的第二個條件。

每個 KeyConditions 元素包含要比較的屬性名稱，並具有下列資訊：

- AttributeValueList：針對所提供的屬性進行評估的一或多個數值。清單中值的數目依使用的 ComparisonOperator 而定。

  Number 類型的數值比較為數字。

  大於、等於或小於的 String 數值比較是根據 UTF-8 二進位編碼的 Unicode。例如，a 大於 A，a 大於 B。

  針對 Binary，每當 DynamoDB 比較二進位值時，都會將二進位資料的每個位元組視為不帶正負號。
- ComparisonOperator：用於評估屬性的比較程式。例如：等於、大於和小於。

  針對 KeyConditions，只支援下列比較運算子：

  ```
  EQ | LE | LT | GE | GT | BEGINS_WITH | BETWEEN
  ```

  以下是這些比較運算子的描述。

  - EQ：等於。

    AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不等於 {"NS":["6", "2", "1"]}。
  - LE：小於或等於。

AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- LT：小於。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- GE：大於或等於。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- GT：大於。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- BEGINS_WITH：檢查字首。

  AttributeValueList 僅可包含 String 或 Binary 類型 (非 Number 或集合類型) 的一個 AttributeValue。比較的目標屬性必須是 String 或 Binary 類型 (非 Number 或集合類型)。

- BETWEEN：大於或等於第一個數值，並且小於或等於第二個數值。

  AttributeValueList 必須包含 String、Number 或 Binary 類型 (非集合類型) 的兩個 AttributeValue 元素。如果目標數值大於或等於第一個元素，並且小於或等於第二個元素，則目標屬性相符。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不與 {"N":"6"} 比較。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

改用 KeyConditionExpression - 範例

假設您想從 Music 資料表擷取數個具有相同分割區索引鍵的項目。您可以搭配 KeyConditions 參數使用Query請求，如本 AWS CLI 範例所示：

```
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"} ]
        }
    }'
```

您可以改用 KeyConditionExpression。

```
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 (舊版)

> **Note**
>
> 我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需詳細資訊，請參閱在 DynamoDB 中使用表達式。如需取代此參數之新參數的特定資訊，請參閱 改用 FilterExpression。。

在 Query 操作中，舊版條件式參數 QueryFilter 是可在項目讀取後評估查詢結果並僅傳回所需值的一個條件。

此參數不支援 List 或 Map 類型的屬性。

> **Note**
>
> QueryFilter 會在項目讀取後套用；篩選程序不會使用任何額外的讀取容量單位。

如果在 QueryFilter 映射中提供一個以上條件，所有條件必須根據預設評估為 true。換句話說，條件是使用 AND 運算子的組合。(您可以改用 ConditionalOperator (舊版) 參數將條件以 OR 連在一起。若要如此，則至少其中一個條件必須評估為 true，而不是全部都必須評估為 true。)

請注意，QueryFilter 不允許使用索引鍵屬性。您無法在分割區索引鍵或排序索引鍵上定義篩選條件。

每個 QueryFilter 元素包含要比較的屬性名稱，並具有下列資訊：

- AttributeValueList：針對所提供的屬性進行評估的一或多個數值。清單中的數值數目依 ComparisonOperator 中指定的運算子而定。

  Number 類型的數值比較為數字。

  大於、等於或小於的 String 數值比較是以 UTF-8 二進位編碼為基礎。例如，a 大於 A，a 大於 B。

  針對 Binary 類型，每當 DynamoDB 比較二進位值時，都會將二進位資料的每個位元組視為不帶正負號。

  如需在 JSON 中指定資料類型的詳細資訊，請參閱 DynamoDB 低階 API。
- ComparisonOperator：用於評估屬性的比較程式。例如：等於、大於和小於。

  可以使用下列比較運算子：

  ```
  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
  BEGINS_WITH | IN | BETWEEN
  ```

改用 FilterExpression - 範例

假設您想查詢 Music 資料表，並將條件套用至相符項目。您可以搭配 QueryFilter 參數使用 Query 請求，如本 AWS CLI 範例所示：

```
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"} ]
        }
    }'
```

您可以改用 FilterExpression。

```
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 (舊版)

> **Note**
>
> 我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需詳細資訊，請參閱在 DynamoDB 中使用表達式。如需取代此參數之新參數的特定資訊，請參閱 改用 FilterExpression。。

在 Scan 操作中，舊版條件式參數 ScanFilter 是可評估掃描結果並僅傳回所需值的一個條件。

> **Note**
>
> 此參數不支援 List 或 Map 類型的屬性。

如果在 ScanFilter 映射中指定一個以上條件，所有條件必須根據預設評估為 true。換言之，條件是以 AND 連在一起的。(您可以改用 ConditionalOperator (舊版) 參數將條件以 OR 連在一起。若要如此，則至少其中一個條件必須評估為 true，而不是全部都必須評估為 true。)

每個 ScanFilter 元素包含要比較的屬性名稱，並具有下列資訊：

- AttributeValueList：針對所提供的屬性進行評估的一或多個數值。清單中的數值數目依 ComparisonOperator 中指定的運算子而定。

  Number 類型的數值比較為數字。

  大於、等於或小於的 String 數值比較是以 UTF-8 二進位編碼為基礎。例如，a 大於 A，a 大於 B。

  針對 Binary，每當 DynamoDB 比較二進位值時，都會將二進位資料的每個位元組視為不帶正負號。

  如需在 JSON 中指定資料類型的詳細資訊，請參閱 DynamoDB 低階 API。
- ComparisonOperator：用於評估屬性的比較程式。例如：等於、大於和小於。

  可以使用下列比較運算子：

  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN

改用 FilterExpression - 範例

假設您想掃描 Music 資料表，並將條件套用至相符項目。您可以搭配 ScanFilter 參數使用 Scan 請求，如本 AWS CLI 範例所示：

```
aws dynamodb scan \
    --table-name Music \
    --scan-filter '{
        "Genre":{
            "AttributeValueList":[ {"S":"Rock"} ],
            "ComparisonOperator": "EQ"
        }
    }'
```

您可以改用 FilterExpression。

```
aws dynamodb scan \
    --table-name Music \
```

```
    --filter-expression 'Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

使用舊式參數撰寫條件

> **Note**
>
> 我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需詳細資訊，請參閱在 DynamoDB 中使用表達式。

下節說明如何撰寫搭配舊式參數 (例如 `Expected`、`QueryFilter`、`ScanFilter`) 使用的條件。

> **Note**
>
> 新的應用程式應改用表達式參數。如需詳細資訊，請參閱在 DynamoDB 中使用表達式。

簡單條件

使用屬性值，您可以撰寫條件來與資料表屬性進行比較。條件一律評估為 true 或 false，並且包含下列各項：

- `ComparisonOperator`：大於、小於、等於等。
- `AttributeValueList` (選用)：要比較的屬性值。依使用中的 `ComparisonOperator` 而定，`AttributeValueList` 可能包含一兩個或更多數值，或者根本不存在。

下列各節說明各種比較運算子，並附有這些運算子在條件中的用法範例。

不具屬性值的比較運算子

- `NOT_NULL`：如果屬性存在，則本項為 true。
- `NULL`：如果屬性不存在，則本項為 true。

使用這些運算子來檢查屬性存在與否。因為並無可比較的數值，所以不要指定 `AttributeValueList`。

範例

如果 Dimensions 屬性存在，則下列表達式評估為 true。

```
...
    "Dimensions": {
         ComparisonOperator: "NOT_NULL"
    }
...
```

具有一個屬性值的比較運算子

- EQ：如果某屬性等於某數值，則本項為 true。

  AttributeValueList 僅可包含 String、Number、Binary、String Set、Number Set 或 Binary Set 類型的一個數值。如果項目內含的數值所屬類型與請求中指定的類型不同，則數值不相符。例如，字串 "3" 不等於數字 3。此外，數字 3 不等於數字集合 [3, 2, 1]。

- NE：如果某屬性不等於某數值，則本項為 true。

  AttributeValueList 僅可包含 String、Number、Binary、String Set、Number Set 或 Binary Set 類型的一個數值。如果項目內含的數值所屬類型與請求中指定的類型不同，則數值不相符。

- LE：如果某屬性小於或等於某數值，則本項為 true。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個數值。如果項目內含的 AttributeValue 所屬類型與請求中指定的類型不同，則數值不相符。

- LT：如果某屬性小於某數值，則本項為 true。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個數值。如果項目內含的數值所屬類型與請求中指定的類型不同，則數值不相符。

- GE：如果某屬性大於或等於某數值，則本項為 true。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個數值。如果項目內含的數值所屬類型與請求中指定的類型不同，則數值不相符。

- GT：如果某屬性大於某數值，則本項為 true。

  AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個數值。如果項目內含的數值所屬類型與請求中指定的類型不同，則數值不相符。

- CONTAINS：如果某數值在集合內，或者某數值包含另一個數值，則本項為 true。

`AttributeValueList` 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個數值。如果比較的目標屬性是 String，則運算子會檢查相符子字串。如果比較的目標屬性是 Binary，則運算子會尋找與輸入相符的目標子序列。比較的目標屬性為集合時，如果找到與該集合任一成員完全相符的項目，則運算子評估為 true。

- `NOT_CONTAINS`：如果某數值不在集合內，或者某數值不包含另一個數值，則本項為 true。

  `AttributeValueList` 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個數值。如果比較的目標屬性是 String，則運算子會檢查相符子字串是否不存在。如果比較的目標屬性是 Binary，則運算子會檢查與輸入相符的目標子序列是否不存在。比較的目標屬性為集合時，如果未找到與該集合任一成員完全相符的項目，則運算子評估為 true。
- `BEGINS_WITH`：如果屬性前幾個字元與提供的數值相符，則本項為 true。請勿使用此運算子來比較數字。

  `AttributeValueList` 僅可包含 String 或 Binary 類型 (非 Number 或集合) 的一個數值。比較的目標屬性必須是 String 或 Binary (非 Number 或集合)。

使用這些運算子來比較屬性與數值。您必須指定包含單一數值的 `AttributeValueList`。大多數的運算子要求此數值必須為一個純量，不過 `EQ` 和 `NE` 運算子也支援集合。

範例

下列各表達式會評估為 true：

- 產品價格大於 100。

```
...
    "Price": {
        ComparisonOperator: "GT",
        AttributeValueList: [ {"N":"100"} ]
    }
...
```

- 產品類別以 Bo 開頭。

```
...
    "ProductCategory": {
        ComparisonOperator: "BEGINS_WITH",
        AttributeValueList: [ {"S":"Bo"} ]
    }
```

```
...
```

- 產品有紅色、綠色或黑色款式：

```
...
    "Color": {
        ComparisonOperator: "EQ",
        AttributeValueList: [
            [ {"S":"Black"}, {"S":"Red"}, {"S":"Green"} ]
        ]
    }
...
```

> **Note**
>
> 比較集合資料類型時，元素的順序無關緊要。無論在請求中指定何順序，DynamoDB 只會傳回具有相同一組數值的項目。

具有兩個屬性值的比較運算子

- BETWEEN：如果某數值介於下限和上限之間 (含端點在內)，則本項為 true。

  `AttributeValueList` 必須包含 String、Number 或 Binary 類型 (非集合) 的兩個元素。如果目標數值大於或等於第一個元素，並且小於或等於第二個元素，則目標屬性相符。如果項目內含的數值所屬類型與請求中指定的類型不同，則數值不相符。

使用此運算子判定屬性值是否在某範圍內。`AttributeValueList` 必須包含同為 String、Number 或 Binary 類型的兩個純量元素。

範例

如果產品價格介於 100 和 200 之間，則下列表達式評估為 true。

```
...
    "Price": {
        ComparisonOperator: "BETWEEN",
        AttributeValueList: [ {"N":"100"}, {"N":"200"} ]
    }
...
```

具有 N 屬性值的比較運算子

- `IN`：如果某數值等於列舉清單中任何數值，則本項為 true。清單中僅支援純量值，不支援集合。目標屬性必須為相同的類型和確切數值才相符。

  `AttributeValueList` 可以包含 String、Number 或 Binary 類型 (非集合) 的一或多個元素。這些屬性會與項目的現有非集合類型屬性進行比較。如果輸入集合的任何元素存在於項目屬性中，則表達式評估為 true。

  `AttributeValueList` 可以包含 String、Number 或 Binary 類型 (非集合) 的一或多個數值。比較的目標屬性必須為相同的類型和確切數值才相符。String 一律不與 String 集合相符。

使用此運算子判定提供的數值是否在列舉清單內。您可以在 `AttributeValueList` 中指定任意數量的純量值，但所有純量值必須屬於同一資料類型。

範例

如果 Id 的數值為 201、203 或 205，則下列表達式評估為 true。

```
...
    "Id": {
        ComparisonOperator: "IN",
        AttributeValueList: [ {"N":"201"}, {"N":"203"}, {"N":"205"} ]
    }
...
```

使用多個條件

DynamoDB 可讓您將多個條件結合成複雜的表達式。您可以提供至少兩個表達式來執行此操作，另可選用 ConditionalOperator (舊版)。

根據預設，指定一個以上條件時，所有條件必須評估為 true，以便整個表達式評估為 true。換言之，會執行隱含的 AND 操作。

範例

如果產品為一本至少有 600 頁的書，則下列表達式評估為 true。這兩個條件皆須評估為 true，因為兩者隱含地以 AND 連在一起。

```
...
```

```
    "ProductCategory": {
        ComparisonOperator: "EQ",
        AttributeValueList: [ {"S":"Book"} ]
    },
    "PageCount": {
        ComparisonOperator: "GE",
        AttributeValueList: [ {"N":600} ]
    }
...
```

可以使用 ConditionalOperator (舊版) 來清楚表示會執行一個 AND 操作。下列範例與前一範例的表現方式相同。

```
...
    "ConditionalOperator" : "AND",
    "ProductCategory": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"N":"Book"} ]
    },
    "PageCount": {
        "ComparisonOperator": "GE",
        "AttributeValueList": [ {"N":600} ]
    }
...
```

您也可以將 `ConditionalOperator` 設定為 OR，這表示其中至少一個條件必須評估為 true。

範例

如果產品為一台登山單車、屬於特定品牌，或者其價格大於 100，則下列表達式評估為 true。

```
...
    ConditionalOperator : "OR",
    "BicycleType": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Mountain" ]
    },
    "Brand": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Brand-Company A" ]
    },
    "Price": {
        "ComparisonOperator": "GT",
```

```
        "AttributeValueList": [ {"N":"100"} ]
    }
...
```

> **Note**
>
> 在複雜的表達式中，會按第一到最後一個條件的順序依序處理條件。
> 無法在單一表達式中同時使用 AND 和 OR。

其他條件式運算子

在舊版 DynamoDB 當中，`Expected` 參數在條件式寫入方面有不同的表現方式。`Expected` 映射中的每個項目顯示 DynamoDB 要檢查的屬性名稱，並具有下列資訊：

- `Value`：要與屬性比較的值。
- `Exists`：判定在嘗試操作之前是否已有此數值。

DynamoDB 持續支援 `Value` 和 `Exists` 選項；不過，這些選項僅能測試相等條件，或者是否存在某屬性。建議您改用 `ComparisonOperator` 和 `AttributeValueList`，因為這些選項可讓您建構更廣泛的條件。

Example

`DeleteItem` 可以查看書籍是否已絕版，並且只在此條件為 true 時才刪除書籍。以下為使用舊式條件的 AWS CLI 範例：

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "Exists": true,
            "Value": {"BOOL":false}
        }
    }'
```

以下範例顯示相同項目，但不使用舊式條件：

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"BOOL":false} ]
        }
    }'
```

Example

PutItem 操作可防止覆寫已有相同主索引鍵屬性的現有項目。以下為使用舊式條件的範例：

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "Exists": false }
    }'
```

以下範例顯示相同項目，但不使用舊式條件：

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "ComparisonOperator": "NULL" }
    }'
```

> **Note**
>
> 針對 `Expected` 映射中的條件，請勿搭配 `ComparisonOperator` 和 `AttributeValueList` 使用舊式的 `Value` 和 `Exists` 選項。若採取此法，條件式寫入將會失敗。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。