DynamoDB

# Index

Table of Contents

[Index 2](#_Toc28725080)

[DynamoDB 3](#_Toc28725081)

[Definition 3](#_Toc28725082)

[Use Cases 3](#_Toc28725083)

[Key Components 3](#_Toc28725084)

[Performance 4](#_Toc28725085)

[Throughput 5](#_Toc28725086)

[DynamoDB Streams 6](#_Toc28725087)

[DynamoDB Basic APIs 6](#_Toc28725088)

[Cost Model 8](#_Toc28725089)

[Miscellaneous Features 8](#_Toc28725090)

# DynamoDB

## Definition

* Non-relational distributed database service that is fast and flexible for any scale with single-digit millisecond latency.
* It is a fully-managed, highly available service replicated across three AZs.
* Enables customers to offload the administrative burdens of operating and scaling distributed databases to AWS so that they don’t have to worry about hardware provisioning, setup and configuration, throughput capacity planning, replication, software patching, or cluster scaling.

## Use Cases

* It supports both document and key value data models.
* It's flexible data model and reliable performance make it great a great fit for mobile web gaming, ad-tech, IOT and many other applications.

## Key Components

* Table
  + Collection of items/data with no pre-defined schema.
* Item
  + Group of attributes that are uniquely identifiable among all other items in a table.
  + Can be thought of as similar to rows in relational databases.
  + The maximum item size in DynamoDB is 400 KB, which includes both attribute name binary length (UTF-8 length) and attribute value lengths (again binary length). The attribute name counts towards the size limit.
* Attributes
  + Fundamental data element.
  + Nested attributes can be 32 layers deep.
  + Attribute Data Types:
    - Scalar: String, Binary, Number, Boolean, Null
    - Document Types: List, Map
    - Set: Number Sets, String Sets, Binary Sets
* Primary Key
  + Attribute that can uniquely identify each item in a table.
  + Primary key attributes must be scalar & non-Null.
  + It can be of two types:
    - Partition key: A simple primary key, composed of one attribute.
    - Partition key and sort key: Composed of two attributes that together form a unique combination.
  + DynamoDB uses partition key value as an input to an internal hash function, the output of which determines the partition where the item will be stored.
    - A partition key should distribute the data well-enough.
    - All items with the same partition key are stored together, in sorted order of the sort key value.
    - If no sort key is used, no two items can have the same partition key.
* Secondary indexes
  + Allows one to query the data using an alternate key, in addition to queries against the primary key.
  + One can create more than one secondary indexes on a table.
  + Two kinds of indexes:
    - Global secondary index (GSI)
      * An index with a partition key and sort key that can be different from those on the table.
      * It creates a whole new table with the newly defined primary key.
        + The new table must consist of the original table’s partition and sort key.
        + Additional attributes can be included from the original table.
        + The new table must have its throughput configured again in the form of Read/Write Capacity Units.
      * It need not be defined at table creation time.
        + It can be added and modified on-the fly, unlike LSI.
    - Local secondary index (LSI)
      * An index that has the same partition key as the table, but a different sort key.
      * It must be defined at the table creation time.
      * The sort key must consistent of exactly one scalar attribute.
        + The attribute must be a scalar String, Number or Binary.
      * One can have up to 5 LSI per table.

## Performance

* It’s backed by SSD-storage for performance improvement and is available across three distinct geographical data centers.
* Its consistency model consists of:
  + Eventually consistent reads
    - This option maximizes read throughput.
    - However, an eventually consistent read might not reflect the results of a recently completed write.
    - All copies of data usually reach consistency within a second. Repeating a read after a short time should return the updated data.
  + Strongly consistent reads
    - DynamoDB also gives you the flexibility and control to request a strongly consistent read if your application, or an element of your application, requires it.
    - A strongly consistent read returns a result that reflects all writes that received a successful response before the read.
  + DynamoDB supports ‘optimistic locking’ consistency model.
    - One can specify conditions for updates or deletes to ensure an item hasn’t changed before altering it.
    - E.g. Update an item only if it’s version number 1, etc.
* Amazon DynamoDB Accelerator (DAX) is a managed in-memory cache service for DynamoDB.
  + One can assign up to 10 nodes to a DAX cluster.
  + DAX supports multi-AZ.
  + The default TTL is 5 minutes.
  + It is launched inside a VPC subnet with appropriate IAM roles and security groups configured.
* DynamoDB auto scaling uses the AWS Application Auto Scaling service to dynamically adjust provisioned throughput capacity on one’s behalf, in response to actual traffic patterns.
  + This enables a table or a global secondary index to increase its provisioned read and write capacity to handle sudden increases in traffic, without throttling.
  + When the workload decreases, Application Auto Scaling decreases the throughput so that one doesn't pay for unused provisioned capacity.

## Throughput

* A DynamoDB table must have read and write capacity units.
* Read Capacity Units (RCU)
  + Throughput for reads.
  + One RCU represents one strongly consistent read per second , or two eventually consistent reads per second, for an item up to 4 KB in size
    - If items are larger than 4 KB, more RCU are consumed.
  + Items are rounded off to nearest upper 4KB increment.
  + Example: 10 eventually consistent reads per second of objects of size 6 KB requires (10/2) \* (8/4) = 10 RCU.
    - One can have 2 eventually consistent reads per second for an object.
    - 6 KB is rounded off to 8 KB.
* Write Capacity Units (WCU)
  + Throughput for writes.
  + One WCU represents one write per second for an item up to 1 KB in size.
  + If items are larger than 1 KB, more WCU are consumed.
    - Items around rounded off to nearest upper 1 KB.
  + Example: Writing 120 objects per minute of 1.5 KB each requires (120/60) \* 2 = 4 WCU.
    - WCU is calculated at second level, not minute. Hence divided by 60.
    - Objects are rounded off to nearest upper KB.
* Throughput can exceed temporarily to meet demand using ‘burst credit’.
  + If ‘burst credit’ is empty, one gets a ‘*ProvisionedThroughputException*’ error.
  + This can happen if:
    - One deals with very large items (more RCU/WCU required).
    - Hot partitions i.e. one partition key is being read too many times (WCU/RCU is spread across partitions).
  + Exponential back-off is advised for such cases.
  + If it’s an RCU issue, DAX can be enabled.
* One can also get ‘*ThrottlingException*’ error when trying to delete, create or update tables quickly.
* Autoscaling can be enabled to scale as per demand.

## DynamoDB Streams

* An optional feature that captures data modifications events in DynamoDB tables.
  + An event is represented by a stream record, a group of which are called shards.
  + A stream record contains the name of the table, the event timestamp and other metadata.
* It can capture events like the image of a new item being added to a table, before-and-after image of the attributes of an updated item, or an image of an entire deleted item.
* It can be paired with AWS Lambda to trigger events based on changes to a table.
* Stream records have a lifetime of 24 hours before they are deleted.

## DynamoDB Basic APIs

For writing data, we have:

* *PutItem*
  + Write data to DynamoDB (create or fully replace).
  + Consumes WCU.
* *UpdateItem*
  + Update data in DynamoDB (partial update of attributes).
  + Possibility to use Atomic Counters and increase them.
* *BatchWriteItem*
  + One can put up to 25 PutItem and/or DeleteItem in one call.
  + There is a limit of 16 MB of data written/ or 400 KB per item.
  + Allows one to save in latency by reducing the number of API calls done against DynamoDB.
  + Operations are done in parallel for better efficiency.
  + Exponential back-off can be used if part of a batch fails due to some reasons (capacity exceed).

For deleting data, we have:

* *DeleteItem*
  + Delete an individual row.
  + One can perform a conditional delete.
* *DeleteTable*
  + Delete a whole table and all its items.
  + Much faster than calling DeleteItem on all items.

For reading data, we have:

* *GetItem*
  + Read based on primary key (HASH or HASH-RANGE).
  + By default, it performs eventually consistent reads.
  + One can opt for strongly consistent reads (which will take twice as much RCU).
  + *‘ProjectionExpression’* can be specified to include only certain attributes & save network bandwidth.
* *BatchGetItem*
  + It returns items based on primary key.
  + Fetch up 100 items/ 16 MB of data.
  + Items are retrieved in parallel to minimize latency.
  + Each item is retrieved separately and hence rounding is done at individual item level, unlike using ‘Query’ where the combined output is rounded.

For querying data, we have:

* *Query*
  + It returns items based on:
    - Partition Key value (must be an ‘=’ operator).
    - Sort Key value (must be ‘= , <, <=, >, >=, BETWEEN, BEGIN) – optional.
    - *‘FilterExpression’* to further filter data. (client-side filtering)
  + Returns up to 1 MB of data or number of items specified in ‘*Limit’*.
  + One can do pagination of results.
  + One can query a table from a global/local secondary index.
* *Scan*
  + It returns the entire table and then filters the data, making it highly inefficient.
  + It returns up to 1 MB of data and uses pagination to keep on reading.
  + Consumes a lot of RCU (Can be reduced by using ‘*Limit*’).
  + One can use parallel scans for faster performance.
    - Multiple instances scan multiple partitions at the same time.
    - Increases throughput and RCU consumed.
  + One can use ‘*ProjectExpression*’ and ‘*FilterExpression*’ (no change to RCU).

## Cost Model

* Core Pricing Components
  + Provisioned Throughput for Writes
  + Provisioned Throughput for Reads
  + Indexed Data Storage (per GB stored)
* Other Pricing Factors
  + Reserved Capacity: Commit to pay minimum WCU/RCU for reduced cost
  + Global Tables: Replicated WCUs or rWCUs.
  + DynamoDB Streams: Per 100k Reads
  + DAX: Per instance used per hour
  + On-demand backup & restore

## Miscellaneous Features

* **Amazon DynamoDB Global Tables:** provides a fully managed solution for deploying a multi-region, multi-master database, without having to build and maintain a replication solution, making it ideal for massively scaled applications, with globally dispersed users.
  + One needs to enable DynamoDB streams to use global tables, among other things.
* **Atomic Counters:** Essentially, an attribute that counts updates to a particular item. DynamoDB writes are applied in the order they were received so they can be used to increment existing values for the purpose of things like a record of website visitors.
  + If operations fail, they can be retried at the risk of under or over counting.
* **Condition Writes:** Accept a write/update only if conditions are respected, otherwise reject.
  + Helps with concurrent access to items.
  + There is no performance impact.
* Throttling can be enabled on DynamoDB for a table or an index.
* One can specify a Time-To-Live (TTL) for items in a table so that they can be deleted automatically from the database.
* One can do point-in-time restores in DynamoDB like RDS with no impact in performance.
* Data types supported include:
  + Scalar Types: String, Number, Binary, Boolean, NULL
  + Document Types: List, Map
  + Set Types: String set, Number set, Binary set