**DynamoDB**

Fully managed NoSQL database service.

Encryption at rest

High availability and durability

**Create table**

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

**Write item to a table**

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

**Read item from a table**

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

**Delete item from a table**

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

**Query a table**

aws dynamodb query --table-name Music

--key-condition-expression "ArtistName=:Artist and SongName=:Songtitle"

**Delete a table**

aws dynamodb delete-table --table-name Music

**List table names**

aws dynamodb list-tables

**Service quota basics**

Read and write units

Read capacity unit (RCU) – One strongly consistent read per second, or two eventually consistent reads per second, for items up to 4 KB in size.

Write capacity unit (WCU) – One write per second, for items up to 1 KB in size.

**Table limits**

Table size – There is no practical limit on table size. Tables are unconstrained in terms of the number of items or the number of bytes.

Number of tables – For any AWS account, there is an initial quota of 2,500 tables per AWS Region.

Page size limit for query and scan – There is a limit of 1 MB per page, per query or scan. If your query parameters or scan operation on a table result in more than 1 MB of data, DynamoDB returns the initial matching items. It also returns a LastEvaluatedKey property that you can use in a new request to read the next page.

**Indexes**

Local secondary indexes (LSIs) – You can define a maximum of five local secondary indexes. LSIs are primarily useful when an index must have strong consistency with the base table.

Global secondary indexes (GSIs) – There is a default quota of 20 global secondary indexes per table.

Projected secondary index attributes per table – You can project a total of up to 100 attributes into all of a table's local and global secondary indexes. This only applies to user-specified projected attributes.

**Partition keys**

The minimum length of a partition key value is 1 byte. The maximum length is 2048 bytes.

There is no practical limit on the number of distinct partition key values, for tables or for secondary indexes.

The minimum length of a sort key value is 1 byte. The maximum length is 1024 bytes.

In general, there is no practical limit on the number of distinct sort key values per partition key value. The exception is for tables with secondary indexes.

For more information on secondary indexes, partition key design, and sort key design, see Best practices.

**Limits for commonly used data types**

String – The length of a string is constrained by the maximum item size of 400 KB. Strings are Unicode with UTF-8 binary encoding.

Number – A number can have up to 38 digits of precision, and can be positive, negative, or zero.

Binary – The length of a binary is constrained by the maximum item size of 400 KB. Applications that work with binary attributes must encode the data in base64 encoding before sending it to DynamoDB.

For a full list of supported data types, see Data types. For more information, also see Service quotas.

**Items, attributes, and expression parameters**

The maximum item size in DynamoDB is 400 KB, which includes both attribute name binary length (UTF-8 length) and attribute value binary lengths (UTF-8 length). The attribute name counts towards the size limit.

There is no limit on the number of values in a list, map, or set, as long as the item that contains the values fits within the 400-KB item size limit.

For expression parameters, the maximum length of any expression string is 4 KB.

For more information about item size, attributes, and expression parameters, see Service quotas.

**Tables -** A collection of data. For example,

**Items** – Each table contains zero or more items. An item is a group of attributes that is uniquely identifiable among all the other items. No limit to the number of items you can store in a table.

**Attributes** – Each item is composed of one or more attributes. An attribute is a fundamental data element, something that does not need to be broken down any further

**Primary key**

Partition key – a simple primary key. One attribute only. No two items can have the same partition key value

Partition key and sort key – Composite primary key. Composed of two attributes. One partition key and one sort key. Partition key for hashing and sort key for ordering. it's possible for multiple items to have the same partition key value. However, those items must have different sort key values.

Partition key also known as hash attribute. Sort key also known as range attribute

Each primary key attribute must be a scalar (meaning that it can hold only a single value). The only data types allowed for primary key attributes are string, number, or binary

**Secondary indexes**

let’s you query the data in the table using an alternate key, (other than the primary key)

**Global secondary index** – An index with a partition key and sort key that can be different from those on the table.

**Local secondary index** – An index that has the same partition key as the table, but a different sort key.

Each table in DynamoDB has a quota of 20 global secondary indexes (default quota) and 5 local secondary indexes.

**DynamoDB Streams** is an optional feature that captures data modification events in DynamoDB tables. The data about these events appear in the stream in near-real time, and in the order that the events occurred.

**DynamoDB API**

**Control Plane** – Create and manage dynamodb tables.

CreateTable /DescribeTable, ListTables, UpdateTable, DeleteTable

**Data Plane** – CRUD on data in the table.

PartiQL - A SQL-compatible query language for DynamoDB

Creating data – PutItem, BatchWriteItem(max 25 items to a table)

Reading data – GetItem, BatchGetItem, Query, Scan

Updating data – UpdateItem

Deleting data – DeleteItem, BatchWriteItem(max 25 items from one ore more tables)

**DynamoDB Streams –** enable/disable stream on a table

ListStreams, DescribeStream, GetShardIterator, GetRecords

**Transactions** – ACID. Can use PartiQL

Classic APIS – TransactWriteItems, TransactGetItems

Data types

Scalar Types –string, number, Boolean, null

Document types – list and map (JSON)

Set Types -string set, number set, binary set

**Table classes**

Standard – default. For majority workloads

Standard-Infrequent Access(IA)

**Partitions and data distribution**

Partition – allocation of storage for a table. SSDs. Replicated across AZs. Partition management completely by AWS.

Creating table  
{

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

}

}

**DynamoDB throughput capacity**

On-demand mode

Provisioned mode

Burst and Adaptive capacity

**global tables** are a fully managed, multi-Region, and multi-active database option that delivers fast and localized read and write performance for massively scaled global applications.

In-memory acceleration with **DynamoDB Accelerator (DAX)**

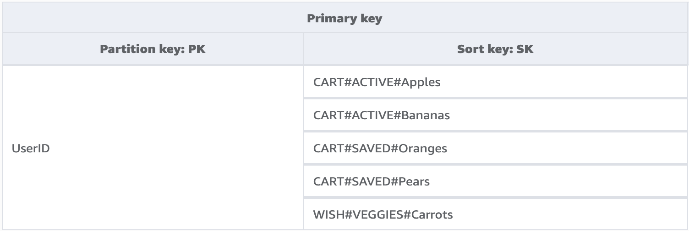
As an in-memory cache, DAX reduces the response times of eventually consistent read workloads by an order of magnitude from single-digit milliseconds to microseconds

Data modelling foundations

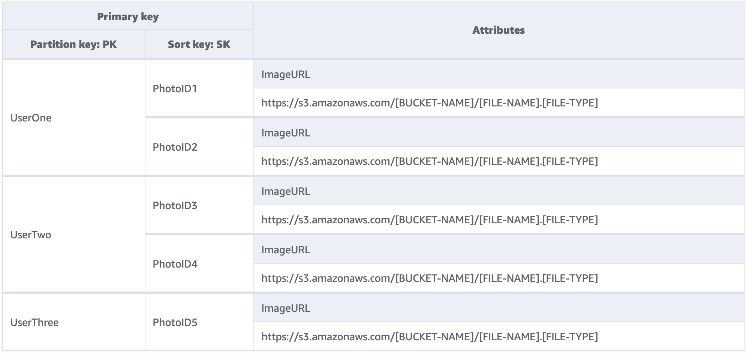
* Single table design foundation: Store multiple types of data in single table.
* Multiple table design foundation: single type of data in each DynamoDB table

Data modelling building blocks

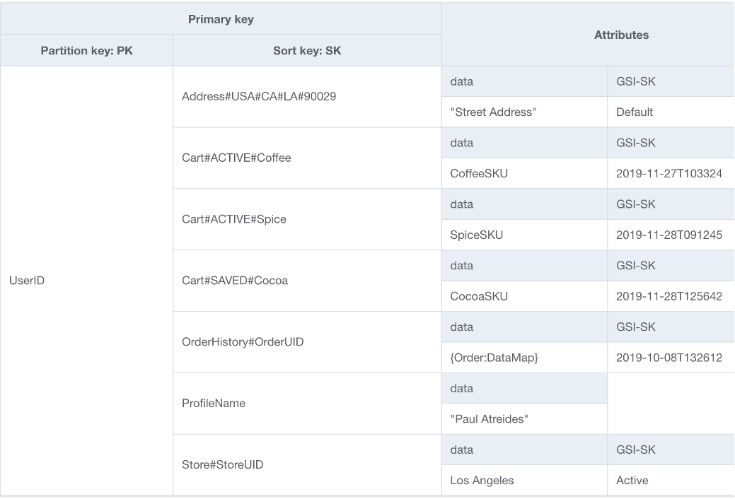
* Composite sort key building block: No relationships in DynamoDB schema. Use sort key to create logical hierarchy of data. (parent layer > child layer > grandchild layer)



* Multi-tenancy building block



* Sparse index building block: looking for items that match a rare item or an item that receives a status. Rather than regularly query across the entire dataset for these items, we can leverage the fact that global secondary indexes (GSI) are sparsely loaded with data. This means that only items in the base table that have the attributes defined in the index will be replicated to the index.
* Time to live building block (TTL)
* Time to live for archival building block
* Vertical partitioning building block



* Write sharding building block

DynamoDB limitation – restriction of how much throughput a single physical partition can maintain per second.

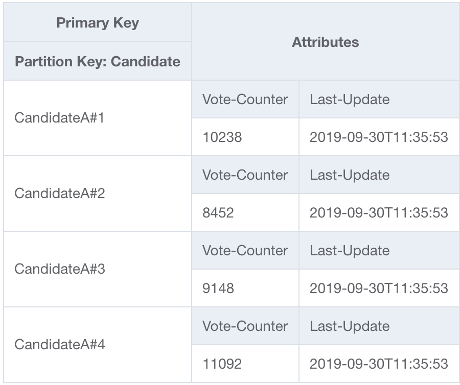
These limits are presently:

1000 WCU (or 1000 <=1KB items written per second) and 3000 RCU (or 3000 <=4KB reads per second) strongly consistent or

6000 <=4KB reads per second eventually consistent

Use cases that require read operations beyond that limit will mostly be served best by placing a read cache in front of DynamoDB, but write operations require a schema level design known as write sharding.





To solve this problem, we'll append a random integer onto the end of the partition key for each contestant in the application's UpdateItem code.

Migrate to DynamoDB

Reasons

* Scalability (to handle massive workloads)
* Performance (low latency data access)
* Fully managed
* Serverless architecture
* NoSQL flexibility
* High availability and durability
* Security and compliance (AWS IAM)
* Integration with AWS ecosystem

Considerations when migrating

* Querying the database
* Designing the database

Best Practices

1. NoSQL design

RDBMS, data can be queried flexibly, but queries are relatively expensive and don't scale well in high-traffic situations.

NoSQL - data can be queried efficiently in a limited number of ways, outside of which queries can be expensive and slow.

These differences make database design different between the two systems:

RDBMS- design for flexibility without worrying about implementation details or performance. Query optimization generally doesn't affect schema design, but normalization is important.

DynamoDB - design your schema specifically to make the most common and important queries as fast and as inexpensive as possible. Your data structures are tailored to the specific requirements of your business use cases.

1. Deletion protection can keep your table from being accidentally deleted
2. DynamoDB Well-Architected Lens to optimize your DynamoDB workload

* Cost optimisation
* Evaludate costs at the table level : apply table tags in cost explorer
* Evaluate table capacity mode (On-demand capacity mode and provisioned capacity mode)
* Evaluate tables auto scaling settings (high utilisation target / low utilisation target)
* Evaluate tables class selection (Standard table class / Standard-IA table class)
* Identifying unused resources (ConsumedReadCapacityUnits/ConsumedWriteCapacityUnits)
* Evaluate table usage patterns
* Perform fewer strongly-consistent read operations
* Perform fewer transactions for read operations
* Perform fewer scans
* Shorten attribute names
* Enable Time to Live (TTL)
* Replace global tables with cross-region backups
* Evaluate streams usage
* Evaluate provisioned capacity for right-sized provisioning(RCU and WCU)

1. Partition key design - By default, every partition in the table deliver the full capacity of 3,000 RCU and 1,000 WCU

* Distributing workloads
* Write sharding (using random/calculated suffixes)
* Upload data efficiently

1. Sort key design

* Using sort keys for version control

1. Secondary indexes

* Global secondary index (GSI) – an index with a partition key and a sort key that can be different from those on base table. Global – because the queries on the index can span all of the data in the base table (across all partitions)
* Local secondary index (LSI) – an index that has the same partition key as the base table, but different sort key. Local – because every partition of a local secondary index is scoped to a base table partition.

Each table can have maximum 20 GSI and 5 LSI

* Take advantage of **Sparse index -** If the sort key doesn't appear in every table item, or if the index partition key is not present in the item, the index is said to be sparse. Sparse indexes are useful for queries over a small subsection of a table. Eg: To track open orders, insert an attribute named isOpen in order items that have not already shipped. Then when the order ships, you can delete the attribute. If you then create an index on CustomerId (partition key) and isOpen (sort key), only those orders with isOpen defined appear in it. When you have thousands of orders of which only a small number are open, it's faster and less expensive to query that index for open orders than to scan the entire table.
* Aggregation (can use streams+lambda)
* GSI overloading
* GSI sharding
* Creating replica

- Set different provisioned read capacity for different readers

- Eliminate reads from a table entirely

1. Best practices for storing large items and attributes

Each item 400KB size limit.

* Compress large attribute values
* Vertical partitioning
* Storing large attribute values in S3

1. Time seriers data:

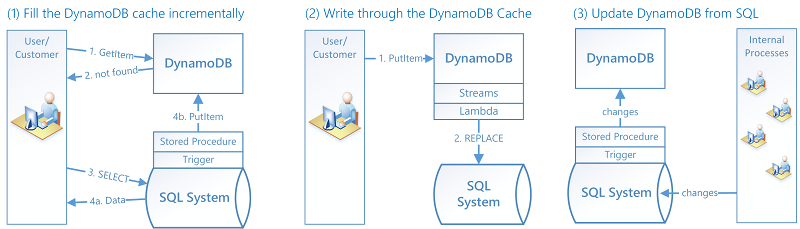
Design patterns for time series data:

* Create one table per period, provisioned with the required read and write capacity and the required indexes.
* Before the end of each period, prebuild the table for the next period. Just as the current period ends, direct event traffic to the new table. You can assign names to these tables that specify the periods they have recorded.
* As soon as a table is no longer being written to, reduce its provisioned write capacity to a lower value (for example, 1 WCU), and provision whatever read capacity is appropriate. Reduce the provisioned read capacity of earlier tables as they age. You might choose to archive or delete the tables whose contents are rarely or never needed.

1. Many-many relationships

* Adjacency list design pattern
* Materialized graph pattern

1. Hybrid DynamoDB-RDMS



1. Relational modelling
2. Querying and scanning

* Avoiding sudden spikes in read activity
  + Reduce page size
  + Isolate scan operations
* Taking advantage of parallel scans
  + Multiple worker threads in a background “sweeper” process could scan a table at a low priority without affecting production traffic
* Choosing TotalSegments

1. Table design
2. Global table design

Global table – data replicates automatically across regions. Employ an active-active replication model.

* Write to any region mode (no primary)
* Write to one region (single primary)
* Write to your region (Mixed primary)

1. Control plane

* Manage DynamoDB tables as well as objects that are dependent on tables such as indexes

1. Biiling and usage reports
2. Switching capacity modes
3. Migrating a DynamoDB table from one account to another
4. DAX prescriptive guidance

DynamoDB-compatible caching service that provides fast in-memory performance

Integration with other services sample

* integrate with Amazon Cognito
* Loading data from DynamoDB into Amazon Redshift
* Processing DynamoDB data with Apache Hive on Amazon EMR
* Integrating with Amazon S3
* DynamoDB zero-ETL integration with Amazon OpenSearch Service

Integration best practices

* Create snapshot in DynamoDB (to s3 or another table)
* Capturing data change
* Zero-ETL integration with OpenSearch service
* Observability
* Scaling