DynamoDB is a fully managed [NoSQL](https://medium.freecodecamp.org/nosql-databases-5f6639ed9574) service that works on key-value pair and other data structure documents provided by Amazon and it requires only a primary key and doesn’t require a schema to create a table. With this, we can expect a good performance even when it scales up. It can store any amount of data and serve any amount of traffic. It is a very simple and small API that follows key-value method to store, access and perform advanced data retrieval.

Dynamo is fast and flexible NoSQL database.

It delivers performance at any scale and its fully managed and it is stored in SSD storage.

It is taken care by Amazon.

It supports two kinds of Consistency:

1. Strongly Consistent read

2. Eventual Consistent read

Eventual Consistent Read :

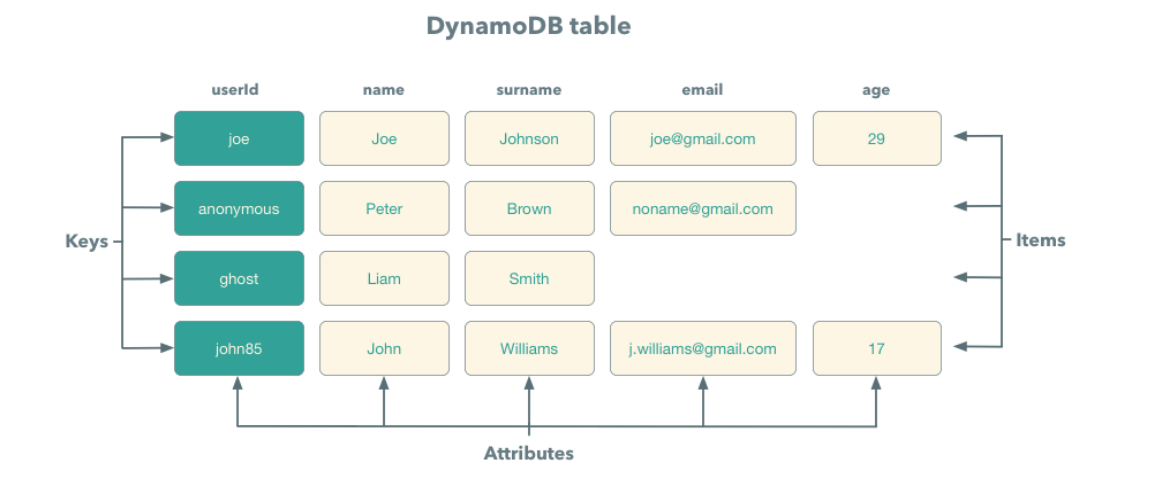
When the DynamoDB table supports eventual consistency reading, the response will not reflect the results of recently completed mutations.

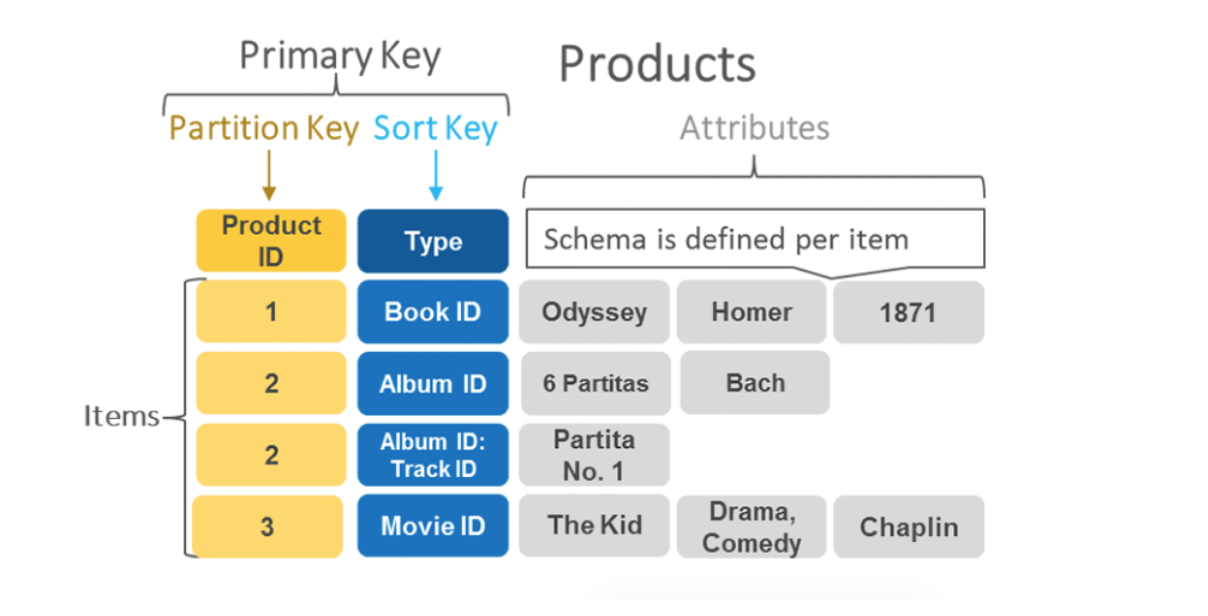
Instead, the result may include some stale data. Stale data stands for data with an older version.

Strong Consistency Read:

While the DynamoDB table supports strongly consistent reading, it results after all mutations have been fully activated.

Therefore, this model will guarantee the return of the most updated data. The process behind this behaviour is by locking down the physical nodes as they update.





DynamoDB comprises of three fundamental units known as table, attribute, and items. A table holds a set of items, an attribute is the simplest element that stores data without any further division and item holds a set of attributes.

**Indexing**

In a relational database, an index is a data structure that lets you perform fast queries on different columns in a table. You can use the CREATE INDEX SQL statement to add an index to an existing table, specifying the columns to be indexed. After the index has been created, you can query the data in the table as usual, but now the database can use the index to quickly find the specified rows in the table instead of scanning the entire table.

## What is a Partition Key?

When creating a Table and defining your DynamoDB schema, one of the first options you’ll be asked is to specify your Table’s Partition Key and Sort Key. This is an important decision that has impact on how your table’s record’s can be accessed.

DynamoDB table schemas also cannot be changed after your table is created, so its important to get it right the first time.

A DynamoDB **Partition Key** has to do with DynamoDB’s internal physical storage structure. The **partition key** is the attribute that DynamoDB will use to partition your data onto one of its many storage nodes.

This is part of the reason DynamoDB is so scalable. Because it can hash your data inputs into an arbitrary number of storage nodes, it can easily scale up to increased demands by simply adding a new partition and shifting data around.

### Partition Key Example

For example, say we have a CustomerOrders table and our partition key is CustomerId. Whenever you insert a new record into your database, the value of your record’s CustomerId will be used as an input to a hashing function.

As a general reminder, a [hashing function](https://en.wikipedia.org/wiki/Hash_function) is simply a function that receives an input and produces a mapped output.

In this case, our hash function is our CustomerId value (say CID-123), and our output is a partition that DynamoDB will internally store this record’s data on. **Which partition DynamoDB stores your record on is completely transparent to you as a user.**

However, the partitioning mechanism DynamoDB employs **does** have an impact on how data can be accessed in your table in terms of maximum capacity .

Note that if you decide to **only** specify a partition key and **not** a sort key, all records must have a unique partition key value. In other words, you will only be able to have one record with CustomerId as CID-123. You’ll see this is **not** **the case** if you opt to use sort key in combination with partition key which we’ll explore below.

## What is a Sort Key?

A **Sort Key** is a secondary key that you can optionally decide to use alongside your Partition Key. In other words, your traditional **Primary Key** can be either just a Partition Key, or a Partition Key + a Sort Key.

A Partition Key and Sort Key combination is known as a composite primary key.

With a Partition Key, you can store records with the **same**partition key value but a **different**sort key value. All records with the same partition key value are stored together on the same data storage node.

Note that a Sort Key is also known as a Range Key. The two terms are generally used interchangeably.

### Sort Key Example

Assume we have the same CustomerOrders table and this time we decide to use a **CustomerId** as the Partition Key and **OrderDate**as the Sort Key. Our table structure and sample records will look a little something like this.

|  |  |  |
| --- | --- | --- |
| **CustomerId** (Partition Key) | **OrderDate**(Sort Key) | **OrderId**(Attribute) |
| CID-123 | 2022-01-05 | 1 |
| CID-123 | 2022-01-06 | 2 |
| CID-123 | 2022-01-07 | 3 |
| CID-**456** | 2022-01-08 | 4 |

*An example table schema consisting of a partition key and sort key combination to form a composite primary*

Notice how our first three records have the same CustomerId value **CID-123** but with different OrderDates. The combination of CustomerId and OrderDate is unique and therefore DynamoDB allows us to store these records.

Note: A Partition Key is simply the key that DynamoDB uses to partition your data onto separate logical data shards. Adding a Sort Key allows us to store multiple records with the same partition key value since the partition key + sort key forms a unique pair, and is therefore our primary key.

# **Partitions and data distribution:**

Amazon DynamoDB stores data in partitions. A partition is an allocation of storage for a table, backed by solid state drives (SSDs) and automatically replicated across multiple Availability Zones within an AWS Region. Partition management is handled entirely by DynamoDB—you never have to manage partitions yourself.

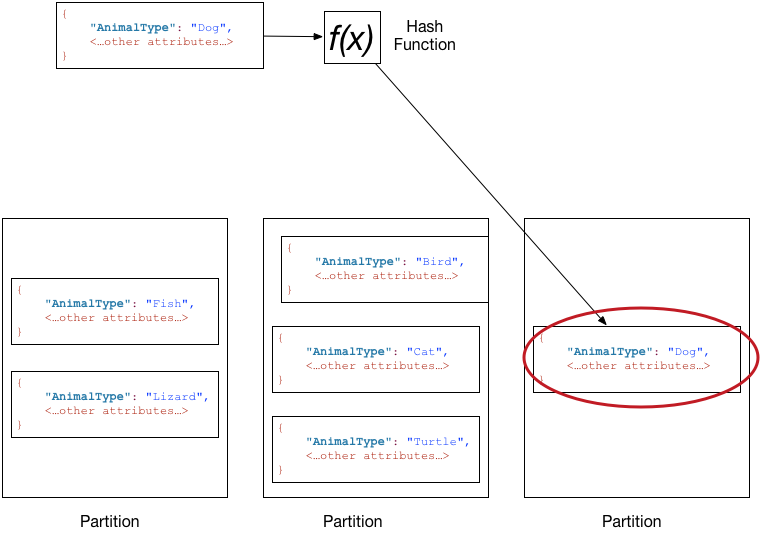
## Data distribution: Partition key

If your table has a simple primary key (partition key only), DynamoDB stores and retrieves each item based on its partition key value.

To write an item to the table, DynamoDB uses the value of the partition key as input to an internal hash function. The output value from the hash function determines the partition in which the item will be stored.

To read an item from the table, you must specify the partition key value for the item. DynamoDB uses this value as input to its hash function, yielding the partition in which the item can be found.

The following diagram shows a table named Pets, which spans multiple partitions. The table's primary key is AnimalType (only this key attribute is shown). DynamoDB uses its hash function to determine where to store a new item, in this case based on the hash value of the string Dog. Note that the items are not stored in sorted order. Each item's location is determined by the hash value of its partition key.



## Data distribution: Partition key and sort key

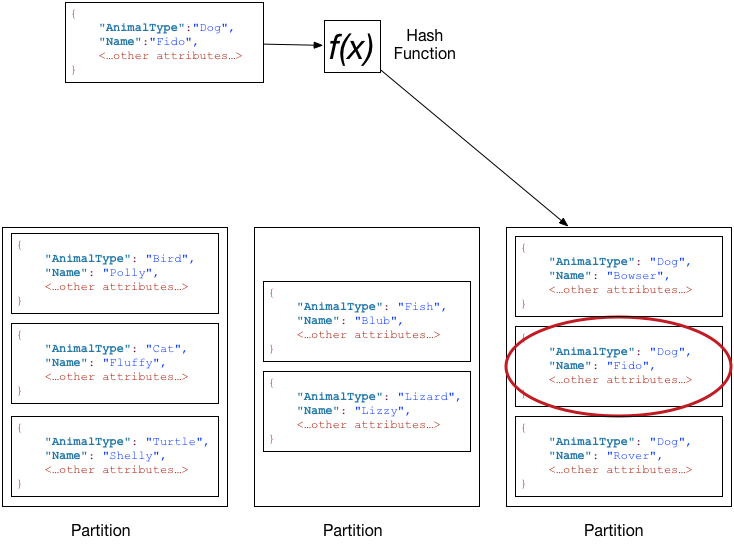
If the table has a composite primary key (partition key and sort key), DynamoDB calculates the hash value of the partition key in the same way as described in [Data distribution: Partition key](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.Partitions.html#HowItWorks.Partitions.SimpleKey). However, it stores all the items with the same partition key value physically close together, ordered by sort key value.

To write an item to the table, DynamoDB calculates the hash value of the partition key to determine which partition should contain the item. In that partition, several items could have the same partition key value. So DynamoDB stores the item among the others with the same partition key, in ascending order by sort key.

To read an item from the table, you must specify its partition key value and sort key value. DynamoDB calculates the partition key's hash value, yielding the partition in which the item can be found.

You can read multiple items from the table in a single operation (Query) if the items you want have the same partition key value. DynamoDB returns all of the items with that partition key value. Optionally, you can apply a condition to the sort key so that it returns only the items within a certain range of values.

Suppose that the Pets table has a composite primary key consisting of AnimalType (partition key) and Name (sort key). The following diagram shows DynamoDB writing an item with a partition key value of Dog and a sort key value of Fido.



To read that same item from the Pets table, DynamoDB calculates the hash value of Dog, yielding the partition in which these items are stored. DynamoDB then scans the sort key attribute values until it finds Fido.

To read all of the items with an AnimalType of Dog, you can issue a Query operation without specifying a sort key condition. By default, the items are returned in the order that they are stored (that is, in ascending order by sort key). Optionally, you can request descending order instead.

To query only some of the Dog items, you can apply a condition to the sort key (for example, only the Dog items where Name begins with a letter that is within the range A through K).

# **Core components of Amazon DynamoDB:**

## Tables, items, and attributes

The following are the basic DynamoDB components:

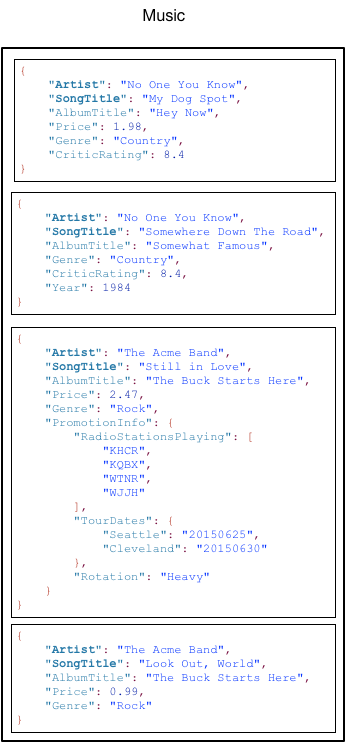
* **Tables** – Similar to other database systems, DynamoDB stores data in tables. A table is a collection of data. For example, see the example table called People that you could use to store personal contact information about friends, family, or anyone else of interest. You could also have a Cars table to store information about vehicles that people drive.
* **Items** – Each table contains zero or more items. An item is a group of attributes that is uniquely identifiable among all of the other items. In a People table, each item represents a person. For a Cars table, each item represents one vehicle. Items in DynamoDB are similar in many ways to rows, records, or tuples in other database systems. In DynamoDB, there is 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. For example, an item in a People table contains attributes called PersonID, LastName, FirstName, and so on. For a Department table, an item might have attributes such as DepartmentID, Name, Manager, and so on. Attributes in DynamoDB are similar in many ways to fields or columns in other database systems.



Note the following about the People table:

* Each item in the table has a unique identifier, or primary key, that distinguishes the item from all of the others in the table. In the People table, the primary key consists of one attribute (PersonID).
* Other than the primary key, the People table is schemaless, which means that neither the attributes nor their data types need to be defined beforehand. Each item can have its own distinct attributes.
* Most of the attributes are scalar, which means that they can have only one value. Strings and numbers are common examples of scalars.
* Some of the items have a nested attribute (Address). DynamoDB supports nested attributes up to 32 levels deep.

The following is another example table named Music that you could use to keep track of your music collection.



Note the following about the Music table:

* The primary key for Music consists of two attributes (Artist and SongTitle). Each item in the table must have these two attributes. The combination of Artist and SongTitle distinguishes each item in the table from all of the others.
* Other than the primary key, the Music table is schemaless, which means that neither the attributes nor their data types need to be defined beforehand. Each item can have its own distinct attributes.
* One of the items has a nested attribute (PromotionInfo), which contains other nested attributes. DynamoDB supports nested attributes up to 32 levels deep.

**Primary key**

When you create a table, in addition to the table name, you must specify the primary key of the table. The primary key uniquely identifies each item in the table, so that no two items can have the same key.

DynamoDB supports two different kinds of primary keys:

* **Partition key** – A simple primary key, composed of one attribute known as the *partition key*.

DynamoDB uses the partition key's value as input to an internal hash function. The output from the hash function determines the partition (physical storage internal to DynamoDB) in which the item will be stored.

In a table that has only a partition key, no two items can have the same partition key value.

The *People* table described in [Tables, items, and attributes](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html#HowItWorks.CoreComponents.TablesItemsAttributes) is an example of a table with a simple primary key (*PersonID*). You can access any item in the *People* table directly by providing the *PersonId* value for that item.

* **Partition key and sort key** – Referred to as a *composite primary key*, this type of key is composed of two attributes. The first attribute is the *partition key*, and the second attribute is the *sort key*.

DynamoDB uses the partition key value as input to an internal hash function. The output from the hash function determines the partition (physical storage internal to DynamoDB) in which the item will be stored. All items with the same partition key value are stored together, in sorted order by sort key value.

In a table that has a partition key and a sort key, it's possible for multiple items to have the same partition key value. However, those items must have different sort key values.

The *Music* table described in [Tables, items, and attributes](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html#HowItWorks.CoreComponents.TablesItemsAttributes) is an example of a table with a composite primary key (*Artist* and *SongTitle*). You can access any item in the *Music* table directly, if you provide the *Artist* and *SongTitle* values for that item.

A composite primary key gives you additional flexibility when querying data. For example, if you provide only the value for *Artist*, DynamoDB retrieves all of the songs by that artist. To retrieve only a subset of songs by a particular artist, you can provide a value for *Artist* along with a range of values for *SongTitle*.

**Note**

The partition key of an item is also known as its *hash attribute*. The term *hash attribute* derives from the use of an internal hash function in DynamoDB that evenly distributes data items across partitions, based on their partition key values.

The sort key of an item is also known as its *range attribute*. The term *range attribute* derives from the way DynamoDB stores items with the same partition key physically close together, in sorted order by the sort key value.

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. There are no such restrictions for other, non-key attributes.

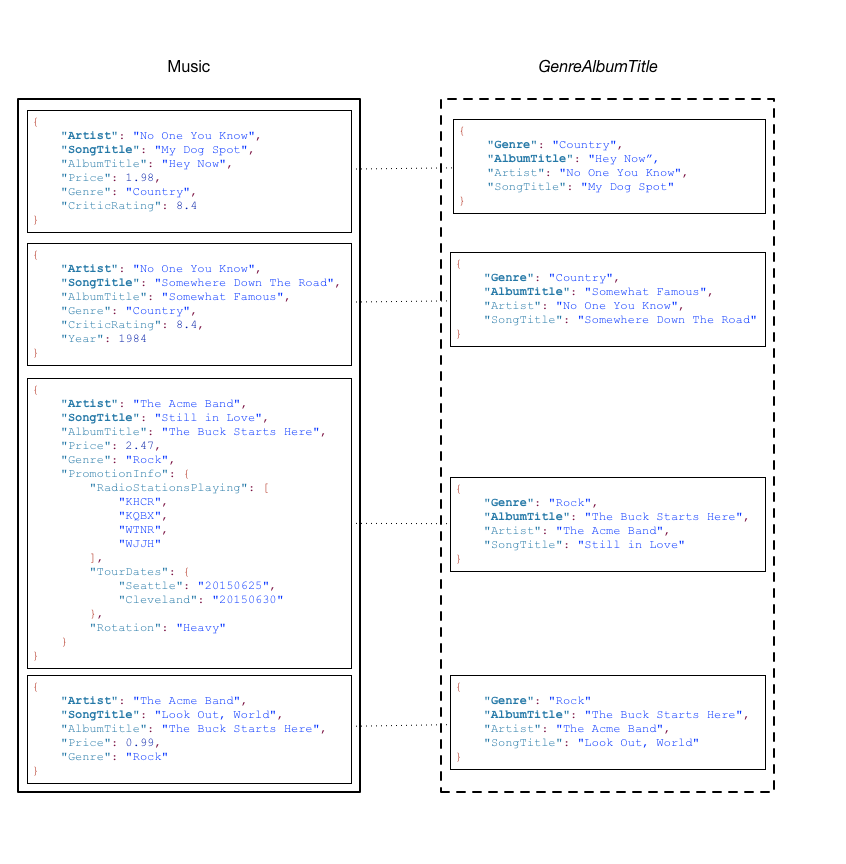
DynamoDB supports two kinds of indexes:

* 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.

In the example Music table shown previously, you can query data items by Artist (partition key) or by Artist and SongTitle (partition key and sort key). What if you also wanted to query the data by Genre and AlbumTitle? To do this, you could create an index on Genre and AlbumTitle, and then query the index in much the same way as you'd query the Music table.

The following diagram shows the example Music table, with a new index called GenreAlbumTitle. In the index, Genre is the partition key and AlbumTitle is the sort key.



Note the following about the GenreAlbumTitle index:

* Every index belongs to a table, which is called the base table for the index. In the preceding example, Music is the base table for the GenreAlbumTitle index.
* DynamoDB maintains indexes automatically. When you add, update, or delete an item in the base table, DynamoDB adds, updates, or deletes the corresponding item in any indexes that belong to that table.
* When you create an index, you specify which attributes will be copied, or projected, from the base table to the index. At a minimum, DynamoDB projects the key attributes from the base table into the index. This is the case with GenreAlbumTitle, where only the key attributes from the Music table are projected into the index.

You can query the GenreAlbumTitle index to find all albums of a particular genre (for example, all Rock albums). You can also query the index to find all albums within a particular genre that have certain album titles (for example, all Country albums with titles that start with the letter H).

# **DynamoDB API**

To work with Amazon DynamoDB, your application must use a few simple API operations. The following is a summary of these operations, organized by category.

**Topics**

* [Control plane](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.API.html#HowItWorks.API.ControlPlane)
* [Data plane](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.API.html#HowItWorks.API.DataPlane)
* [DynamoDB Streams](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.API.html#HowItWorks.API.Streams)
* [Transactions](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.API.html#HowItWorks.API.Transactions)

## Control plane

Control plane operations let you create and manage DynamoDB tables. They also let you work with indexes, streams, and other objects that are dependent on tables.

* CreateTable – Creates a new table. Optionally, you can create one or more secondary indexes, and enable DynamoDB Streams for the table.
* DescribeTable– Returns information about a table, such as its primary key schema, throughput settings, and index information.
* ListTables – Returns the names of all of your tables in a list.
* UpdateTable – Modifies the settings of a table or its indexes, creates or removes new indexes on a table, or modifies DynamoDB Streams settings for a table.
* DeleteTable – Removes a table and all of its dependent objects from DynamoDB.

## Data plane

Data plane operations let you perform create, read, update, and delete (also called CRUD) actions on data in a table. Some of the data plane operations also let you read data from a secondary index.

You can use [PartiQL - a SQL-compatible query language for Amazon DynamoDB](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-reference.html), to perform these CRUD operations or you can use DynamoDB’s classic CRUD APIs that separates each operation into a distinct API call.

### PartiQL - A SQL-compatible query language

* ExecuteStatement – Reads multiple items from a table. You can also write or update a single item from a table. When writing or updating a single item, you must specify the primary key attributes.
* BatchExecuteStatement – Writes, updates or reads multiple items from a table. This is more efficient than ExecuteStatement because your application only needs a single network round trip to write or read the items.

### Classic APIs

#### Creating data

* PutItem – Writes a single item to a table. You must specify the primary key attributes, but you don't have to specify other attributes.
* BatchWriteItem – Writes up to 25 items to a table. This is more efficient than calling PutItem multiple times because your application only needs a single network round trip to write the items. You can also use BatchWriteItem for deleting multiple items from one or more tables.

#### Reading data

* GetItem – Retrieves a single item from a table. You must specify the primary key for the item that you want. You can retrieve the entire item, or just a subset of its attributes.
* BatchGetItem – Retrieves up to 100 items from one or more tables. This is more efficient than calling GetItem multiple times because your application only needs a single network round trip to read the items.
* Query – Retrieves all items that have a specific partition key. You must specify the partition key value. You can retrieve entire items, or just a subset of their attributes. Optionally, you can apply a condition to the sort key values so that you only retrieve a subset of the data that has the same partition key. You can use this operation on a table, provided that the table has both a partition key and a sort key. You can also use this operation on an index, provided that the index has both a partition key and a sort key.
* Scan – Retrieves all items in the specified table or index. You can retrieve entire items, or just a subset of their attributes. Optionally, you can apply a filtering condition to return only the values that you are interested in and discard the rest.

#### Updating data

* UpdateItem – Modifies one or more attributes in an item. You must specify the primary key for the item that you want to modify. You can add new attributes and modify or remove existing attributes. You can also perform conditional updates, so that the update is only successful when a user-defined condition is met. Optionally, you can implement an atomic counter, which increments or decrements a numeric attribute without interfering with other write requests.

#### Deleting data

* DeleteItem – Deletes a single item from a table. You must specify the primary key for the item that you want to delete.
* BatchWriteItem – Deletes up to 25 items from one or more tables. This is more efficient than calling DeleteItem multiple times because your application only needs a single network round trip to delete the items. You can also use BatchWriteItem for adding multiple items to one or more tables.