**Dynamo DB**

A table is a collection of items (row), and each item is a collection of attributes. (column)

No limit to the number of items in table. Items can have a nested attribute up to **32 levels deep**.

**Keys**

DynamoDB supports two different kinds of primary keys:

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

**2. Partition key and sort key –** Referred as composite primary key, & is composed of two attributes: partition key and sort key. All items with the same partition key value are stored together, in sorted order by sort key value.

Partition key of an item is also known as its hash attribute & sort key as range attribute.

**Indexes**

DynamoDB supports two kinds of indexes:

**1. Global secondary index** – An index with a partition key and sort key that can be different from those on the table. It has a quota of 20 global secondary indexes (default quota). Eventually Consistent

**2. Local secondary index** – An index that has the same partition key as the table, but a different sort key. There can be 5 local secondary indexes. You can only add local secondary indexes on tables with composite primary keys. You cannot add a local secondary index to an existing table. Can be both strong as well as eventual consistent.

You can project a total of up to 100 attributes into all of a table's local and global secondary indexes.

**Projected Attributes**

When provisioning a secondary index, you specify which attributes you want to project into the index. This states which attributes will be available from the index directly without needing to make an additional call to the underlying table to retrieve attributes.

**KEYS\_ONLY**: Your index will include only the keys for the index and the table's underlying partition and sort key values, but no other attributes.

**ALL:** The full Item is available in the secondary index with all attributes.

**INCLUDE:** You may choose to name certain attributes that are projected into the secondary index.

**Streams**

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.

Each event is represented by a stream record. If you enable a stream on a table, DynamoDB Streams writes a stream record whenever one of the following events occurs:

A new item is **added** to the table: The stream captures an image of the entire item, including all of its attributes.

An item is **updated**: The stream captures the "before" and "after" image of any attributes that were modified in the item.

An item is **deleted** from the table: The stream captures an image of the entire item before it was deleted.

Each stream record also contains the name of the table, the event timestamp, and other metadata. Stream records have a lifetime of 24 hours; after that, they are automatically removed from the stream.

**READ CONSISTENCY**

**Eventually Consistent Reads :** When you read data from a DynamoDB table, the response might not reflect the results of a recently completed write operation. The response might include some stale data. If you repeat your read request after a short time, the response should return the latest data.

**Strongly Consistent Reads :** When you request a strongly consistent read, DynamoDB returns a response with the most up-to-date data, reflecting the updates from all prior write operations that were successful.

Disadvantage

Might not be available if there is a network delay or outage

May have higher latency not supported on global secondary indexes.

**Data types**

DynamoDB supports many different data types for attributes within a table. They can be categorized as follows:

**Scalar Types** – A scalar type can represent exactly one value. The scalar types are number, string, binary, Boolean, and null.

**Document Types** – A document type can represent a complex structure with nested attributes (up to 32 levels deep.), such as you would find in a JSON document. The document types are list and map.

**Set Types –** A set type can represent multiple scalar values. The set types are string set, number set, and binary set.

Numbers can be positive, negative, or zero. Numbers can have up to 38 digits of precision. Exceeding this results in an exception.

**Positive range:** 1E-130 to 9.9999999999999999999999999999999999999E+125

**Negative range:** -9.9999999999999999999999999999999999999E+125 to -1E-130

All numbers are sent across the network to DynamoDB as strings, to maximize compatibility across languages and libraries. However, DynamoDB treats them as number type attributes for mathematical operations.

The minimum length of a string can be zero, if the attribute is not used as a key for an index or table, and is constrained by the maximum DynamoDB item size limit of 400 KB.

For a simple primary key, the maximum length of the first attribute value (the partition key) is 2048 bytes.

For a composite primary key, the maximum length of the second attribute value (the sort key) is 1024 bytes.

**Binary**

The length of a Binary is constrained by the maximum item size of 400 KB.

**Document types**

The document types are list and map.

**ADAPTIVE CAPACITY**

By default, every partition in the table will strive to deliver the full capacity of 3,000 RCU and 1,000 WCU.

DynamoDB provides some flexibility for your throughput provisioning **with burst capacity**. Whenever you're not fully using your available throughput, DynamoDB reserves a portion of that unused capacity for later bursts of throughput to handle usage spikes.

DynamoDB currently retains up to 5 minutes (300 seconds) of unused read and write capacity.

**Adaptive capacity** is a feature that enables DynamoDB to run imbalanced workloads indefinitely. It minimizes throttling due to throughput exceptions. It also helps you reduce costs by enabling you to provision only the throughput capacity that you need.

**LIMITS AND QUOTAS**

**Capacity unit sizes (for provisioned tables)**

One read capacity unit = one strongly consistent read per second, or two eventually consistent reads per second, for items up to 4 KB in size.

One write capacity unit = one write per second, for items up to 1 KB in size.

Transactional read requests require two read capacity units to perform one read per second for items up to 4 KB.

Transactional write requests require two write capacity units to perform one write per second for items up to 1 KB.

**Request unit sizes (for on-demand tables)**

One read request unit = one strongly consistent read, or two eventually consistent reads, for items up to 4 KB in size.

One write request unit = one write, for items up to 1 KB in size.

Transactional read requests require two read request units to perform one read for items up to 4 KB.

Transactional write requests require two write request units to perform one write for items up to 1 KB.

|  | **On-Demand** | **Provisioned** | **Adjustable** |
| --- | --- | --- | --- |
| Per table | 40,000 read request units and 40,000 write request units | 40,000 read capacity units and 40,000 write capacity units | Yes |
| Per account | Not applicable | 80,000 read capacity units and 80,000 write capacity units | Yes |
| Minimum throughput for any table or global secondary index | Not applicable | 1 read capacity unit and 1 write capacity unit | Yes |

## Tables

### Table size

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

### Maximum number of tables per account per region

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

## Global tables

AWS places some default quotas on the throughput you can provision or utilize when using global tables.

|  | **On-Demand** | **Provisioned** |
| --- | --- | --- |
| Per table | 40,000 read request units and 40,000 write request units | 40,000 read capacity units and 40,000 write capacity units |
| Per table, per destination Region, per day | 10 TB for all source tables to which a replica was added for this destination Region | 10 TB for all source tables to which a replica was added for this destination Region |

## **Partition keys and sort keys**

### Partition key length: Minimum length of a partition key value is 1 byte. The maximum length is 2048 bytes.

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

### Sort key length: Minimum length of a sort key value is 1 byte. The maximum length is 1024 bytes.

**Naming Conventions: Table names and Secondary Index names**

Table names and index names must be between 3 and 255 characters long, and can contain only the following characters: a-z, A-Z, 0-9, \_ (underscore), - (dash), . (dot)

### Attribute names: In general, an attribute name must be at least one character long, but no greater than 64 KB long.

### Item size: 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.

### Item size for tables with Local Secondary Indexes

For each local secondary index on a table, there is a 400 KB limit on the total of the following:

* The size of an item's data in the table.
* The size of corresponding entries (including key values and projected attributes) in all local secondary indexes.

## **Attributes**

### Attribute name-value pairs per item

The cumulative size of attributes per item must fit within the maximum DynamoDB item size (400 KB).

### Number of values in list, map, or set

There is no limit on the number of values in a List, a Map, or a Set, as long as the item containing the values fits within the 400 KB item size limit.

### Attribute values

Empty String and Binary attribute values are allowed, if the attribute is not used as a key attribute for a table or index. Empty String and Binary values are allowed inside Set, List, and Map types. An attribute value cannot be an an empty Set (String Set, Number Set, or Binary Set). However, empty Lists and Maps are allowed.

## **DynamoDB transactions**

**TransactWriteItems**is a synchronous and idempotent write operation that groups up to 100 write actions in a single all-or-nothing operation. These actions can target up to 100 distinct items in one or more DynamoDB tables within the same AWS account and in the same Region. The aggregate size of the items in the transaction cannot exceed 4 MB. The actions are completed atomically so that either all of them succeed or none of them succeeds. Operation supported : PUT, UPDATE, DELETE, CONDITIONAL CHECK.

**TransactGetItems** is a synchronous read operation that groups up to 100 Get actions together. These actions can target up to 100 distinct items in one or more DynamoDB tables within the same AWS account and Region. The aggregate size of the items in the transaction can't exceed 4 MB.

DynamoDB transactional API operations have the following constraints:

* A transaction cannot contain more than 100 unique items.
* A transaction cannot contain more than 4 MB of data.
* No two actions in a transaction can work against the same item in the same table. For example, you cannot both ConditionCheck and Update the same item in one transaction.
* A transaction cannot operate on tables in more than one AWS account or Region.
* Transactional operations provide atomicity, consistency, isolation, and durability (ACID) guarantees only within the AWS Region where the write is made originally.

## **API-specific limits**

**CreateTable/UpdateTable/DeleteTable**

In general, you can have up to 500 CreateTable, UpdateTable, and DeleteTable requests running simultaneously in any combination. As a result, the total number of tables in the CREATING, UPDATING, or DELETING state cannot exceed 500.

**BatchGetItem**

A single BatchGetItem operation can retrieve a maximum of 100 items. The total size of all the items retrieved cannot exceed 16 MB.

**BatchWriteItem**

A single BatchWriteItem operation can contain up to 25 PutItem or DeleteItem requests. The total size of all the items written cannot exceed 16 MB.

**DescribeTableReplicaAutoScaling**

DescribeTableReplicaAutoScaling method supports only 10 requests per second.

**DescribeLimits**

DescribeLimits should be called only periodically. You can expect throttling errors if you call it more than once in a minute.

**DescribeContributorInsights/ListContributorInsights/UpdateContributorInsights**

DescribeContributorInsights, ListContributorInsights and UpdateContributorInsights should be called only periodically. DynamoDB supports up to five requests per second for each of these APIs.

**Query**

The result set from a Query is limited to 1 MB per call. You can use the LastEvaluatedKey from the query response to retrieve more results.

**Scan**

The result set from a Scan is limited to 1 MB per call. You can use the LastEvaluatedKey from the scan response to retrieve more results.

**UpdateTableReplicaAutoScaling**

UpdateTableReplicaAutoScaling method supports only ten requests per second.

**UpdateTableTimeToLive**

The UpdateTableTimeToLive method supports supports only one request to enable or disable Time to Live (TTL) per specified table per hour. This change can take up to one hour to fully process. Any additional UpdateTimeToLive calls for the same table during this one hour duration result in a ValidationException

## CRUD Operations

**putItem(params = {}, callback) ⇒ AWS.Request**

Creates a new item, or replaces an old item with a new item. You can perform a conditional put operation (add a new item if one with the specified primary key doesn't exist), or replace an existing item if it has certain attribute values. You can return the item's attribute values in the same operation, using the ReturnValues parameter.

When you add an item, the primary key attributes are the only required attributes. Attribute values cannot be null.

**updateItem(params = {}, callback) ⇒ AWS.Request**

It just updates the item if primary key is already present. If not present, it creates the item.

**getItem(params = {}, callback) ⇒ AWS.Request**

The GetItem returns set of attributes for the item with given primary key. If no matching item, GetItem does not return any data.

GetItem provides an eventually consistent read by default. If your application requires a strongly consistent read, set ConsistentRead to true.

**deleteItem(params = {}, callback) ⇒ AWS.Request**

Deletes a single item in a table by primary key. You can also return the item's attribute values in the same operation, using the ReturnValues parameter.

Unless you specify conditions, the DeleteItem is an idempotent operation; running it multiple times on the same item or attribute does not result in an error response.

**BATCH OPERATIONS**

**batchGetItem(params = {}, callback) ⇒ AWS.Request**

The BatchGetItem operation returns the attributes of one or more items from one or more tables. You identify requested items by primary key.

A single operation can retrieve up to 16 MB of data,or max 100 items. BatchGetItem returns a partial result if the response size limit is exceeded, the table's provisioned throughput is exceeded, or an internal processing failure occurs. If a partial result is returned, the operation returns a value for UnprocessedKeys. You can use this value to retry the operation starting with the next item to get.

If DynamoDB returns any unprocessed items, you should retry the batch operation on those items. However, we strongly recommend that you use an exponential backoff algorithm. If you retry the batch operation immediately, the underlying read or write requests can still fail due to throttling on the individual tables. If you delay the batch operation using exponential backoff, the individual requests in the batch are much more likely to succeed.

**batchWriteItem(params = {}, callback) ⇒ AWS.Request**

The BatchWriteItem operation puts or deletes multiple items in one or more tables. A single call to BatchWriteItem can transmit up to 16MB of data over the network, consisting of up to 25 item put or delete operations. While individual items can be up to 400 KB once stored, it's important to note that an item's representation might be greater than 400KB while being sent in DynamoDB's JSON format for the API call. For more details on this distinction, see Naming Rules and Data Types.

**Note**: BatchWriteItem cannot update items. To update items, use the UpdateItem action.

The individual PutItem and DeleteItem operations specified in BatchWriteItem are atomic; however BatchWriteItem as a whole is not. If any requested operations fail because the table's provisioned throughput is exceeded or an internal processing failure occurs, the failed operations are returned in the UnprocessedItems response parameter. You can investigate and optionally resend the requests. Typically, you would call BatchWriteItem in a loop. Each iteration would check for unprocessed items and submit a new BatchWriteItem request with those unprocessed items until all items have been processed.

**QUERING AND SCANNING**

Querying finds items in a table or a secondary index using only primary key attribute values. You must provide a partition key name and a value for which to search. You can also provide a sort key name and value, and use a comparison operator to refine the search results. Scanning finds items by checking every item in the specified table

**QUERING**

**query(params = {}, callback) ⇒ AWS.Request**

You must provide the name of the partition key attribute and a single value for that attribute. Query returns all items with that partition key value. Optionally, you can provide a sort key attribute and use a comparison operator to refine the search results.

A Query operation always returns a result set. If no matching items are found, the result set will be empty. Queries that do not return results consume the minimum number of read capacity units for that type of read operation.

Query results are always sorted by the sort key value. If the data type of the sort key is Number, the results are returned in numeric order; otherwise, the results are returned in order of UTF-8 bytes. By default, the sort order is ascending. To reverse the order, set the ScanIndexForward parameter to false.

A single Query operation will read up to the maximum number of items set (if using the Limit parameter) or a maximum of 1 MB of data and then apply any filtering to the results using FilterExpression. If LastEvaluatedKey is present in the response, you will need to paginate the result set. For more information, see Paginating the Results in the Amazon DynamoDB Developer Guide.

**SCANNING**

**scan(params = {}, callback) ⇒ AWS.Request**

The Scan operation returns one or more items and item attributes by accessing every item in a table or a secondary index. To have DynamoDB return fewer items, you can provide a FilterExpression operation.

If the total number of scanned items exceeds the maximum dataset size limit of 1 MB, the scan stops and results are returned to the user as a LastEvaluatedKey value to continue the scan in a subsequent operation. The results also include the number of items exceeding the limit. A scan can result in no table data meeting the filter criteria.

A single Scan operation reads up to the maximum number of items set (if using the Limit parameter) or a maximum of 1 MB of data and then apply any filtering to the results using FilterExpression. If LastEvaluatedKey is present in the response, you need to paginate the result set.

**Expression**

**Condition expressions** are used when manipulating individual items to only change an item when certain conditions are true.

* attribute\_exists(): Check for existence of an attribute;
* attribute\_not\_exists(): Check for non-existence of an attribute;
* attribute\_type(): Check if an attribute is of a certain type;
* begins\_with(): Check if an attribute begins with a particular substring;
* contains(): Check if a String attribute contains a particular substring or a Set attribute contains a particular element; and
* size(): Returns a number indicating the size of an attribute.

**Projection expressions** are used to specify a subset of attributes you want to receive when reading Items. We used these in our GetItem calls in the previous lesson.

**Update expressions** are used to update a particular attribute in an existing Item.

**Key condition expressions** are used when querying a table with a composite primary key to limit the items selected.

**Filter expressions** allow you to filter the results of queries and scans to allow for more efficient responses.