# DynamoDB

**Overview:**

* Scalable NoSQL DB
* On-demand backup
* Also supports point in time recovery (during last **35** days)
* Stored using SSD and automatically replicated across AZs
* Supports nested attributed level up to 32

**Primary Key (hash attribute)**

* Two types
  + Partition key – hash attribute
  + Partition key and Sort key (composite primary key) – range attribute
* Must be scalar. Can be string, number and binary
* **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.

**Secondary indexes**

Two types

* Global secondary (GSI)
  + Different partition key and sort key than table
  + 20 indexes maximum
* Local secondary (LSI)
  + Same partition key as like table and different sort key
  + 5 indexes maximum
  + **Should be created while creation of table**

**Streams**

* 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.
* 24 hours life time

**DynamoDB API**

* Control plane
  + CreateTable
  + DescribeTable(No RCU Consumption)
  + ListTables
  + UpdateTable(No RCU Consumption)
  + DeleteTable
* Data plane – CRUD operation
  + Create
    - PutItem
    - BatchWriteItem
      * **Writes 25 items to table**
  + Read
    - GetItem – Single item
    - BatchGetItem – 100 items (max 16 MB)
    - Query –
      * **Retrieve all items that have a partition key**
      * We can use this if table/index has both partition and sort key
    - Scan
      * Retrieve all items from table or index
  + Update
    - UpdateItem
      * **Must specify primary key**
  + Delete
    - DeleteItem
    - BatchWriteItem
      * Writes 25 items to table. (Max 16 Mb)
* Streams
  + ListStreams – to list available streams
  + DescribeStream – describing details about stream to read
  + GetShradIterator – handle to retrieve records from stream
  + GetRecords – retrieves one or more stream records
* Transactions
  + TransactWriteItems – a batch to allow Put, Update & Delete within transactions scope. Either all or nothing
  + TransactGetItems – Get operation

**Data types in DynamoDB**

* Scalar
  + Number, string, binary, Boolean & null
  + If you define a primary key attribute as a binary type attribute, the following additional constraints apply:
* 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.
* Document Types
  + JSON
  + **Nested documents up to 32 levels**
  + Maximum size **400 kb**
* Set types
  + All the set elements should be same type
  + No limit for the set, but item should not exceed 400 kb
  + Does not support empty set

**Read consistency**

* Supports Eventually and Strong consistent reads
* Eventually read
  + Often stale data
  + Recent write will not be available on immediate read
* Strong read
  + Recent data immediately available
  + GSI will not support strong read

**Read/Write capacity mode**

The **default read consistency in DynamoDB i**s **"eventually consistent read**" which means that we need to multiply the number of items by '2

* On-demand mode

**RRU & WRU**

* One *read request unit* represents one strongly consistent read request, or two eventually consistent read requests, for an item up to 4 KB in size.
* One *write request unit* represents one write for an item up to 1 KB in size. If you need to write an item that is larger than 1 KB, DynamoDB needs to consume additional write request units.
* Provisioned mode
  + RCU & WCU
* One *read capacity unit* represents one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4 KB in size.  For example, if your item size is 8 KB, you require 2 read capacity units to sustain one strongly consistent read per second, 1 read capacity unit if you choose eventually consistent reads
* One *write capacity unit* represents one write per second for an item up to 1 KB in size. For example, if your item size is 2 KB, you require 2 write capacity units to sustain one write request per second
  + Provisioned throughput is the maximum amount of capacity that an application can consume from a table or index. If your application exceeds your provisioned throughput capacity on a table or index, it is subject to request throttling.
  + Throttling prevents your application from consuming too many capacity units. When a request is **throttled**, it fails with an HTTP 400 code (Bad Request) and a **ProvisionedThroughputExceededException**.

**Partition and 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.
* 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)—but it stores all of 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, there could be several items with 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.

**CLI**

ResourceInUseException – if duplicate table creation

**Create**-table

aws dynamodb create-table \

--table-name Customers \

--attribute-definitions \

AttributeName=ID,AttributeType=N \

AttributeName=Name,AttributeType=S \

**--key-schema \**

AttributeName=ID,KeyType=HASH \

AttributeName=Name,KeyType=RANGE \

--provisioned-throughput \

ReadCapacityUnits=10,WriteCapacityUnits=5

**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

To view details about a table, use the DescribeTable operation.

aws dynamodb describe-table --table-name Music

aws dynamodb update-table --table-name Music \

--provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10

aws dynamodb delete-table --table-name Music

To view provisioned throughput limits,

aws dynamodb describe-limits

We recommend that you choose shorter attribute names rather than long ones. This helps you reduce the amount of storage required for your data.

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"} }'

aws dynamodb get-item --consistent-read \

--table-name Music \

--key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}'

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

aws dynamodb query \

--table-name Music \

--key-condition-expression "Artist = :name" \

--expression-attribute-values '{":name":{"S":"Acme Band"}}'

aws dynamodb update-table \

--table-name Music \

--attribute-definitions AttributeName=AlbumTitle,AttributeType=S \

--global-secondary-index-updates \

"[{\"Create\":{\"IndexName\": \"AlbumTitle-index\",\"KeySchema\":[{\"AttributeName\":\"AlbumTitle\",\"KeyType\":\"HASH\"}], \

\"ProvisionedThroughput\": {\"ReadCapacityUnits\": 10, \"WriteCapacityUnits\": 5 },\"Projection\":{\"ProjectionType\":\"ALL\"}}}]"

Check for attributes in an Item

aws dynamodb delete-item \

--table-name ProductCatalog \

--key '{"Id": {"N": "456"}}' \

--condition-expression "**attribute\_not\_exists(Price)"**

**Conditional Writes**

* Conditional writes are idempotent

If a ConditionExpression evaluates to false during a conditional write, DynamoDB will still consume write capacity from the table:

* If the item does not currently exist in the table, DynamoDB will consume one write capacity unit.
* If the item does exist, then the number of write capacity units consumed depends on the size of the item. For example, a failed conditional write of a 1 KB item would consume one write capacity unit. If the item were twice that size, the failed conditional write would consume two write capacity units.
* A failed conditional write will return a ***ConditionalCheckFailedException***. When this occurs, you will not receive any information in the response about the write capacity that was consumed. However, you can view the **ConsumedWriteCapacityUnits** metric for the table in Amazon CloudWatch.
* ReturnConsumedCapacity
  + TOTAL
  + INDEXES
  + NONE
* Unlike a global secondary index, a local secondary index shares its provisioned throughput capacity with its table. **Read and write activity on a local secondary index consumes provisioned throughput capacity from the table.**

**Scans**

* Can return result set always.
* Max 1 Mb
* Can be filtered by filter expression
* For example, suppose you Scan a table, with a **Limit value of 6,** and without a filter expression. The Scan result will contain the first six items from the table that match the key condition expression from the request.
* Filter will be applied after Limit scan
* Pagination
  + **LastEvaluatedKey**
  + **ExclusiveStartKey**
  + The absence of LastEvaluatedKey is the only way to know that you have reached the end of the result set.
* Result count
  + **Scanned count** – number of items before filter
  + **Count** – number of items after filter
* Scan read consistency
  + always eventually consistent
  + set the **ConsistentRead** parameter to true in the Scan request. For strong consistency

|  |  |  |
| --- | --- | --- |
| **Characteristic** | **Global Secondary Index** | **Local Secondary Index** |
| **Key Schema** | The primary key of a global secondary index can be either simple (partition key) or composite (partition key and sort key). | The primary key of a local secondary index must be composite (partition key and sort key). |
| **Key Attributes** | The index partition key and sort key (if present) can be any base table attributes of type string, number, or binary. | The partition key of the index is the same attribute as the partition key of the base table. The sort key can be any base table attribute of type string, number, or binary. |
| **Size Restrictions Per Partition Key Value** | There are no size restrictions for global secondary indexes. | For each partition key value, the total size of all indexed items **must be 10 GB or less.** |
| **Online Index Operations** | Global secondary indexes can be created at the same time that you create a table. You can also add a new global secondary index to an existing table, or delete an existing global secondary index. For more information, see [Managing Global Secondary Indexes](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.OnlineOps.html). | **Local secondary indexes are created at the same time that you create a table. You cannot add a local secondary index to an existing table**, nor can you delete any local secondary indexes that currently exist. |
| **Queries and Partitions** | A global secondary index lets you **query over the entire table, across all partitions.** | A local secondary index lets you **query over a single partition, as specified by the partition key value in the query.** |
| **Read Consistency** | Queries **on global secondary indexes support** **eventual consistency only.** | When you query a local secondary index, you can choose either eventual consistency or strong consistency. |
| **Provisioned Throughput Consumption** | Every global secondary index has its own provisioned throughput settings for read and write activity. Queries or scans on a **global secondary index consume capacity units from the index, not from the base table.** The same holds true for global secondary index updates due to table writes. | Queries or scans on a local **secondary index consume read capacity units from the base table**. When you write to a table, its local secondary indexes are also updated; these updates consume write capacity units from the base table. |
| **Projected Attributes** | With global secondary index queries or scans, **you can only request the attributes that are projected into the index.** DynamoDB **will not fetch any attributes from the table**. | If you query or scan a local secondary index, **you can request attributes that are not projected in to the index. DynamoDB will automatically fetch tho**se attributes from the table. |

**Indexes**

GSI

* Base table's primary key attributes are always projected into an index
* If you set the **ScanIndexForward** parameter to false, the results are returned in descending order, so the highest score is returned first.
* In a DynamoDB table, each key value must be unique. However, the key values in a global secondary index do not need to be unique.
* A global secondary index only keeps track of data items where its key attribute(s) actually exist. If any new item added (does not contain Index attributes), then only existent attribute in the base table will be propagated to Index.
* Projections
  + Keys-only
  + Include
  + All
  + **If you need to access just a few attributes with the lowest possible latency, consider projecting only those attributes into a global secondary index**. The smaller the index, the less that it will cost to store it, and the less your write costs will be.
  + **If your application will frequently access some non-key attributes, you should consider projecting those attributes into a global secondary index.** The additional storage costs for the global secondary index will offset the cost of performing frequent table scans.
  + **If you need to access most of the non-key attributes on a frequent basis, you can project these attributes—or even the entire base table— into a global secondary index.** This will give you maximum flexibility; however, your **storage cost would increase, or even double.**
  + **If your application needs to query a table infrequently, but must perform many writes or updates against the data in the table, consider projecting *KEYS\_ONLY*.** The global secondary index would be of minimal size, but would still be available when needed for query activity.

LSI

* Conditions
  + Partition key is same as of its base table
  + The sort key consists of exactly one scalar attribute
  + The sort key of the base table projected to index as non-key attribute

**Best practices for DynamoDB**

* **For DynamoDB, by contrast, you shouldn't start designing your schema until you know the questions it will need to answer. Understanding the business problems and the application use cases up front is essential.**
* **You should maintain as few tables as possible in a DynamoDB application. Most well-designed applications require only one table.**
* Partition
  + Traffic against **partition key should not exceed 3000 RCU/1000 WCU**
* Burst capacity
  + DB reserves **300 seconds of unused WCU/RCU for burst**.
  + Well designed sort keys
* Indexes
  + **Global secondary index**—An index with a partition key and a sort key that can be different from those on the base table. A global secondary index is considered "global" because queries on the index can span all of the data in the base table, across all partitions. A global secondary index **has no size limitations** and has its own provisioned throughput settings for read and write activity that are separate from those of the table.
  + **Local secondary index**—An index that has the same partition key as the base table, but a different sort key. A local secondary index is "local" in the sense that every partition of a local secondary index is scoped to a base table partition that has the same partition key value. As a result, the total size of indexed items for any one partition key value **can't exceed 10 GB**. Also, a local secondary index shares provisioned throughput settings for read and write activity with the table it is indexing.
  + Should create number of indexes as minimum as possible
  + Project only attributes required
  + Sparse Index
    - Not all the items have index sort key
* Querying and Scanning
  + Scan/Query
    - Scans full table and then filter
    - Scans Should degrade performance as the table/index grows
    - Use query instead
    - Scan operation eventually consistent by default
    - Create multiple number of smaller scans
  + Use parallel scan

*Also, as a table or index grows, the Scan operation slows.*

The Scan operation examines every item for the requested values and can use up the provisioned throughput for a large table or index in a single operation.

For faster response times, design your tables and indexes so that your applications can use **Query instead of Scan**. (For tables, you can also consider using the **GetItem** and **BatchGetItem** APIs.)

# Error Handling

If the request is unsuccessful, DynamoDB returns an error. Each error has three components:

* An HTTP status code (such as 400).
* An exception name (such as ResourceNotFoundException).
* An error message (such as Requested resource not found: Table: *tablename* not found).

An HTTP 400 status code indicates a problem **with your request, such as authentication failure, missing required parameters, or exceeding a table's provisioned throughput**. You have to fix the issue in your application before submitting the request again.

* **ItemCollectionSizeLimitExceededException**
* **LimitExceededException**
* **MissingAuthenticationTokenException**
* **ProvisionedThroughputExceededException**
* **RequestLimitExceeded**
* **ResourceInUseException (**You tried to re-create an existing table, or delete a table currently in the CREATING state**)**
* **ThrottlingException**
* **UnrecognizedClientException (Due to invalid access keys)**

# DynamoDB Limits

* 1 RCU – 4 kb
  + 1 consistent read, 2 eventually consistent read
* 256 tables per region
* 5 LSI
* 20 GSI
* Partition keys and Sort keys
  + Partition key length – Min 1 and Max 2048 bytes
  + Sort key length – Min 1 and Max 1024 bytes
* Table item size – 400 KB
* Table/Index names – Min 3 characters and Max 255 long
* Data Types – String, Number, Binary
* Attributes
  + Name max 400 KB size
  + List, Map or Set – max 400 KB size
  + Nested attribute levels – 32 levels
* Expression – **ProjectionExpression/ConditionExpression/UpdateExpression**
  + Max length of 4 KB
* API Specific Limits
  + Max of 50 CreateTable,UpdateTable & DeleteTable
  + BatchGetItem – Max of 100 items
  + BatchWriteItem – Can contain 25 **PutItem/DeleteItem**. Size can’t exceed 16 MB
  + Query – Limited to 1 MB
  + Scan – Limited to 1 MB

# Storing large item in DynamoDB

As mentioned above, you can also take advantage of Amazon Simple Storage Service (Amazon S3) **to store large attribute values that cannot fit in a DynamoDB** item. You can store them as an **object in Amazon S3 and then store the object identifier in your DynamoDB item.**

You can also use the **object metadata support in Amazon S3 to provide a link back to the parent item in DynamoDB.** **Store the primary key value of the item as Amazon S3 metadata of the object in Amazon S3**. Doing this often helps with maintenance of the Amazon S3 objects.

# Time to Live

* Time to Live (TTL) for DynamoDB allows you to define when items in a table expire so that they can be automatically deleted from the database.
* With TTL enabled on a table, you can set a timestamp for deletion on a per-item basis, allowing you to limit storage usage to only those records that are relevant.
* session data, event logs, usage patterns, and other temporary data. If you have sensitive data that must be retained only for a certain amount of time according to contractual or regulatory obligations, TTL helps you ensure that it is removed promptly and as scheduled.

# Amazon DynamoDB provides easy, scalable session storage

Amazon DynamoDB is the ideal candidate for a **session storage solution in a share-nothing, distributed architecture.**

* **Easy** As a fully-hosted, fully-managed solution, there is nothing to install or setup.
* **Fast** The service is fundamentally designed for low latency (including the usage of SSDs).
* **Smart** You dont need to worry about scalability, replication, redundancy or sharding. Everything happens automatically and seamlessly.

Best practices for working with DynamoDB Streams  
Keep in mind the following best practices when you are designing solutions that use DynamoDB Streams:

* DynamoDB Streams enables you to build solutions **using near real-time synchronization of data**. It doesn’t enforce consistency or transactional capability across many tables. This must be handled at the application level. Also, be aware of the latency involved (sub second) in the processing of stream data as data is propagated into the stream. This helps you define the SLA regarding data availability for your downstream applications and end users.
* All item-level changes will be in the stream, including deletes. Your application should be able to handle deletes, updates, and creations.
* Design your stream-processing layer to handle different types of failures. Make sure that you store the stream data **in a dead letter queue such as SQS or S3, for later processing in the event of a failure.**
* Failures can occur in the application that reads the events from the stream. You can design the application to minimize the risk and blast radius. To that end, try not to update too many tables with the same code. In addition, you can design your tables so that you update multiple attributes of a single item (instead of five different items, for example). You can also define your **processing to be idempotent, which can allow you to retry** safely. You should also catch different exceptions in your code and decide if you want to retry or **ignore these records and put them in a DLQ for further analysis.**
* Be aware of the following constraints while you are designing consumer applications:
* Data **retention in streams is 24 hours**.
* No more than **two processes should be reading from a stream shard at the same time.**
* **We recommend that you consider Lambda for stream processing whenever possible because it is serverless and therefore easier to manage**. First, evaluate if Lambda can be used. If it can’t be, then use the **Kinesis Client Library (KCL).** The following comparison table can help you decide.

|  |  |  |
| --- | --- | --- |
| **Parameters** | **AWS Lambda** | **Kinesis Client Library** |
| **Deployment** | Lambda polls the DynamoDB stream and invokes your function/code as soon as it detects the new record. | You write your custom application using KCL with DynamoDB Streams Kinesis Adapter and host it in an EC2 instance. |
| **Manageability** | Lambda automatically scales based on the throughput. | You must manage the shards, monitoring, scaling, and checkpointing process in line with KCL best practices. (For details, see this [design blog post](https://aws.amazon.com/blogs/big-data/processing-amazon-dynamodb-streams-using-the-amazon-kinesis-client-library/) and this [monitoring blog post](https://aws.amazon.com/blogs/big-data/monitor-your-application-for-processing-dynamodb-streams/).) |
| **How it works** | For every DynamoDB partition, there is a corresponding shard and a Lambda function poll for events in the stream (shard). Based on the batch size you specify, it fetches the records, processes it, and then fetches the next batch. | * Enumerates the shards within the stream. * Coordinates shard associations with other workers (if any). * Instantiates a record processor for every shard it manages. * Pulls records from the stream. * Pushes the records to the corresponding record processor. * Checkpoints processed records. |
| **Execution time** | Lambda Maximum execution duration per request is 300 seconds. | There are no restrictions. |
| **Availability** | Lambda is a managed service and is fully available. There are no maintenance windows or scheduled downtimes required. | The application must be hosted in an EC2 Auto Scaling group for High Availability. |
| **Other considerations** | Note the impact of the [Lambda payload limit](http://docs.aws.amazon.com/lambda/latest/dg/limits.html) on using Lambda to consume stream messages. Any of the \_IMAGE types can exceed it, especially for larger items. Therefore you should specify the batch size accordingly. |  |

Best Practises in scan

* Parallel Scan
  + Segment
  + If table size is > 20 GB
  + If Sequential operation is too slow
  + Total Segments
    - **2 GB per segment**
* Estimating Page Size
  + Limit
* Rounding of 4 KB
  + Limit
* Recoding Process
  + LastEvaluatedKey

Errors

* AccessDeniedException
  + Client didn’t sign-in. Goto Sig v4 process
* ConditionalCheckFailedException
  + Conditional update failed due to value mismatch
* IncompleSignatureException
* ItemCollectionSizeLimitExceedException
  + Group of items with same partition key has exceeded max size 10 GB
  + Retry possible – Yes
* LimitExceededException ( Too many operations for given subscriber)
  + Retry possible – Yes
  + Simultaneously creating, updating, deleting tables/index can’t exceed 50
  + If you attempt to concurrently create more than one table with a secondary index, DynamoDB will return a **LimitExceededException**.
* ProvisionedThroughputExceededException
  + Retry – yes
  + Provisioned throughput exceeded for table/GSI
  + **Retries with exponential backoff – solution**
* RequestLimitExceeded
  + Retry yes
  + Requests rate increased
* ResourceInUseException
  + Table is in use or in creating state
* ResourceNotFoundException
* ThrottlingException - Rate of requests exceeds the allowed throughput.
  + Retry yes
  + Too many requests
* ValidationException
  + Data type mismatch

If **you want to create more than one table with secondary indexes**, you must do so sequentially. For example, you would create the first table and wait for it to become ACTIVE, create the next table and wait for it to become ACTIVE, and so on. If you attempt to concurrently create more than one table with a secondary index, DynamoDB will return a **LimitExceededException**.

Working with Queries

Key Condition Expression

To **specify search criteria, use this string. It determines the items to be read from the table or index.** **Partition key name and value as equality condition**.

Optionally provide one more condition with sort key.

Ex,

aws dynamodb query \

--table-name Thread \

--key-condition-expression "**ForumName** = :name" \

--expression-attribute-values ‘{":name":{"S":"Amazon DynamoDB"}}'

aws dynamodb query \

--table-name Thread \

--key-condition-expression "ForumName = :name and Subject = :sub" \

--expression-attribute-values file://values.json

The arguments for --expression-attribute-values are stored in the file values.json:

{

":name":{"S":"Amazon DynamoDB"},

":sub":{"S":"DynamoDB Thread 1"}

}

With the sort key

aws dynamodb query \

--table-name Reply \

--key-condition-expression "Id = :id and begins\_with(ReplyDateTime, :dt)" \

--expression-attribute-values file://values.json

By default, the sort order is ascending. To reverse the order, set the ScanIndexForward parameter to false.

Filter Expression

* Used to further filter items with in the results.
* Query will consume same amount of RCU irrespective of filter exp.
* Max. of **1 MB** of data.
* Can’t contain partition key/sort key attributes.
* Ex,

aws dynamodb query \

--table-name Thread \

--key-condition-expression "ForumName = :fn" \

--filter-expression "#v >= :num" \

--expression-attribute-names '{"#v": "Views"}' \

--expression-attribute-values <file://values.json>

Limit the Number of items in the Result Set

* Pass **Limit value parameter**
* For example, suppose you Query a table, with a **Limit value of 6,** and without a filter expression. The Query result will contain the first six items from the table that match the key condition expression from the request.
* If filter expression exists, it will only filter those 6 items

Pagination

* LastEvaluatedKey - to determine if there are results pending for next paging request. If not exist this mean no more results.
* ExclusiveStartKey  - start value to be assigned from LastEvaluatedKey value for next page.

Count

* ScannedCount – number of matching items as per key condition before filter expression
* Count – number of remaining items after filter applied.

CU

* If query a
  + Table – table’s RCU
  + GSI – index’s RCU
  + LSI – base table’s RCU
* ReturnConsumedCapacity  - to get the details of consumed RCU
  + NONE
  + TABLE
  + INDEXES

Read Consistency for Query

* By default, all Query operations are eventually consistent read.
* Set **ConsistenRead=true** for strong consistent read

# DynamoDB Encryption at Rest

DynamoDB encryption at rest provides an additional layer of data protection by securing your data in the encrypted table, including its primary key, local and global secondary indexes, streams, global tables, backups, and DynamoDB Accelerator (DAX) clusters whenever the data is stored in durable media.

When creating a new table, you can choose one of the following customer master keys (CMK) to encrypt your table:

* AWS owned CMK – **Default encryption type.** The key is owned by DynamoDB (no additional charge).
* AWS managed CMK – The key is stored in your account and is managed by AWS KMS (AWS KMS charges apply).

DynamoDB uses the CMK to generate and encrypt a **unique data key for the table, known as the table key**.

With DynamoDB, AWS Owned, or AWS Managed CMK can be used to generate &

encrypt keys. AWS Owned CMK is free of charge while AWS Managed CMK is chargeable.

Customer managed CMK’s are not supported with encryption at rest. "With encryption at rest,

Security features such as multi-factor authentication (MFA), checks for compromised

credentials, account takeover protection, and phone and email veri􀃝cation.

Customized work􀃞ows and user migration through AWS Lambda triggers.

https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identitypools.

html

DynamoDB transparently encrypts all customer data in a DynamoDB table, including its primary

key and local and global secondary indexes, whenever the table is persisted to disk

Amazon DynamoDB offers fully managed encryption at rest. DynamoDB encryption at rest

provides enhanced security by encrypting your data at rest using an AWS Key Management

Service (AWS KMS) managed encryption key for DynamoDB. This functionality eliminates the

operational burden and complexity involved in protecting sensitive data.

For more information on DynamoDB Encryption at rest, please refer to the below Link:

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/EncryptionAtRest.html