
aws
certified
Solutions
Architect
Professional
DOLFINed

No SQL DataBase Options on AWS

AMAZON DYNAMODB

Amazon DynamoDB

Features

Amazon DynamoDB

DynamoDB

DynamoDB is a fully managed NoSQL database that supports both document & Key-Value store

- Is extremely fast and delivers predictable performance with seamless scalability
- Use for applications that need **consistent, single-digit millisecond latency** at any scale.
- Is a web service that uses HTTP over SSL (HTTPS) a a transport and JSON as a message serialization format

- Use cases:
  - Mobile Apps
  - Web Apps
  - Gaming Apps
  - Ad-tech Apps
  - Internet of things (IoT)

Source:aws.amazon.com

DynamoDB – Tables

- DynamoDB tables are schemaless
  - Which means that neither the attributes nor their data types need to be defined beforehand
  - Each item can have its own distinct attributes
- DynamoDB does not support:
  - Complex relations DB querying or Joins
  - Does not support complex transactions

Source:aws.amazon.com

DynamoDB – Durability & Performance

- DynamoDB automatically replicates data across three facilities (Data Centers [Not AZs])in an AWS region for H.A and data durability
  - It also partitions your DB over sufficient number of servers according to your read/write capacity
  - Performs automatic failover in case of any failure

- DynamoDB runs exclusively on SSD volumes which provides:
  - Low latency
  - Predictable performance
  - High I/Os

Source:aws.amazon.com

DynamoDB – Read Consistency

DynamoDB supports both Eventual Consistency (default) and Strong Consistency models

- **Eventually Consistent reads (Default):**
  - When you read data from a DynamoDB table, the response might not reflect the results of a recently completed write operation.
  - Best read throughput
  - Consistency across all copies is reached in 1 second

- **Strongly Consistent reads:**
  - A read returns a result that reflects all writes that received a successful response prior to the read

- Users/Applications reading from DynamoDB tables can specify in their requests if they want strong consistency, otherwise it will be eventually consistent reads (default)
  - So the application will dictate what is required , Strong, Eventual, or both

Source:aws.amazon.com

Amazon DynamoDB

DynamoDB

- DynamoDB allows low latency read/write access to items ranging from 1 byte to 400KBytes
- DynamoDB can be used to store pointers to S3 stored objects, or items of sizes larger than 400KB, too if needed
- DynamoDB stores data indexed **by a primary key**
  - You specify the primary key when you create the 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.
- The primary key is the only required attribute for items in a table
- DynamoDB supports GET/PUT operations using a user defined Primary Key

Source:aws.amazon.com

DynamoDB – Read Capacity Units

- One read capacity unit represents one strongly consistent read per second, or two eventually consistent reads per second for an item up to 4KB in size

- If you need to read an item that is larger than 4 KB, DynamoDB will need to consume additional read capacity units.
  - The total number of read capacity units required depends on the item size, and whether you want an eventually consistent or strongly consistent read.

Source:aws.amazon.com

DynamoDB – Write Capacity Units

- One write capacity unit represents one write per second for an item up to 1 KB in size.
- If you need to write an item that is larger than 1 KB, DynamoDB will need to consume additional write capacity units.
- The total number of write capacity units required depends on the item size.

Source:aws.amazon.com

Amazon DynamoDB

- Features
- Durability
- Scalability
- Performance
- Throttling

Amazon DynamoDB

Features

- Is a fast, fully-managed NoSQL database service that makes it simple and cost-effective to store and retrieve any amount of data, and serve any level of request traffic.
- Amazon DynamoDB helps offload the administrative burden of operating and scaling a highly-available distributed database cluster.
- It meets the latency and throughput requirements of highly demanding applications by providing extremely fast and predictable performance with seamless throughput and storage scalability.
- DynamoDB supports three data types: number, string, and binary, in both scalar and multi-valued sets.
- DynamoDB provides both eventually-consistent reads (by default), and strongly-consistent reads (optional), as well as implicit item-level transactions for item put, update, delete, conditional operations, and increment/decrement.

Amazon DynamoDB – Durability and Availability

- Amazon DynamoDB has built-in fault tolerance that **automatically and synchronously** replicates data across **three Availability Zones in a region** for high availability and to help protect data against individual machine, or even facility failures.

- Amazon DynamoDB is integrated with other services, such as:
  - Amazon Elastic MapReduce (Amazon EMR), Amazon Redshift, Amazon Data Pipeline, and Amazon S3, which allows for using DynamoDB data for:
    - Analytics,
    - Data warehouse,
    - Data import/export,
    - Backup and Archive.

Amazon DynamoDB – Scalability and Elasticity

- Amazon DynamoDB is both highly-scalable and elastic.
- Data is automatically partitioned and re-partitioned as needed,
- The use of SSDs provides predictable low-latency response times at any scale.
- The service is also elastic, in that you can simply “dial-up” or “dial-down” the read and write capacity of a table as your needs change.
- DynamoDB Cross Region Replication and DynamoDB streams
- You can scale the provisioned capacity of your DynamoDB table anytime you want
- There is no limit to the number of items (data) you can store in a DynamoDB table
- There is no limit on how much data your can store per DynamoDB table
- Supports auto scaling and on-demand capacity to scale or when the required WCU/RCUs are unknown.

DynamoDB - Throttling

- If your read or write requests exceed the throughput settings for a table, DynamoDB can throttle that request.
- DynamoDB can also throttle read requests exceeds for an index.
  - 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
- The AWS SDKs have built-in support for retrying throttled requests

Source:aws.amazon.com

Amazon DynamoDB - Performance

- SSDs and limiting indexing on attributes provides:
  - High throughput and low latency (single-digit milliseconds typical for average server-side response times), and
  - Drastically reduces the cost of read and write operations.

- As the datasets grow, predictable performance is required so that low-latency for the workloads can be maintained.
  - This predictable performance can be achieved by defining the provisioned throughput capacity required for a given table.
    - Behind the scenes, the service handles the provisioning of resources to achieve the requested throughput rate,
      - This takes the burden away from the customer to have to think about instances, hardware, memory, and other factors that can affect an application's throughput rate.
- Provisioned throughput capacity reservations are elastic and can be increased or decreased on demand.

Amazon DynamoDB - Interfaces

- While standard SQL isn't available for Amazon DynamoDB, you may use the Amazon DynamoDB select operation to create SQL-like queries that retrieve a set of attributes based on criteria that you provide.

Amazon DynamoDB

- Data Partitions
- Primary/Partition Key
- Sort Key

Amazon DynamoDB

DynamoDB Table Example

Customer ID	Customer First Name	Last Name	Phone number	Email	Open Orders	Paid	Address
1244567	John	Smith	12345	j.smith@a.com	True	Yes	
289147	Marcia	Herndon	57849				
345987	Ahmed	Essa	98765	a.essa@Kmail.net	True	On Delivery	1234 Happiness Ct, NYC, NY 11111
5456567	Kumar	Shariyar	12568				

Partitions and Data Distribution

- 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 service, you do not need to worry about it
- When you create a table, DynamoDB allocates enough partitions to the table so that it can handle your provisioned throughput requirements.
- DynamoDB allocates additional partitions to a table in the following situations:
  - If you increase the table's provisioned throughput settings beyond what the existing partitions can support.
  - If an existing partition fills to capacity and more storage space is required.

DynamoDB and Indexing

- Tables do not have a fixed schema, so each data item can have a different number of attributes.
- The primary key can either be a single-attribute partition/hash key or a composite (Partition/Hash key AND a Sort/Range key).
- For Composite keys, Hash key defines the partition where data is stored, and Sort key sorts or saves the data within that partition
- Local secondary indexes provide additional flexibility for querying against attributes other than the primary key.

Primary Key – Simple Key (Partition/Hash 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.

DynamoDB Table Example – Simple Key

Customer ID	Customer First Name	Last Name	Phone number	Email	Open Orders	Paid	Address
1244567	John	Smith	12345	j.smith@a.com	True	Yes	
289147	Marcia	Herndon	57849				
345987	Ahmed	Essa	98765	a.essa@Kmail.net	True	On Delivery	1234 Happiness Ct, NYC, NY 11111
5456567	Kumar	Shariyar	12568				

Primary Key – Composite (Partition/Hash and Sort/Range Keys)

- 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 in the simple Key
  - It then stores all of the items with the same partition key value physically close together, ordered by sort key value.

- To read that same item from a DynamoDB table with composite key, You need to use the Partition key and, optionally, a value for the sort key
  - DynamoDB calculates the hash value of using the Partition key, yielding the partition in which the items are stored.
  - DynamoDB then scans the sort key attribute values until it finds the sort key value indicated.
  - You can use conditions to the sort key (=, >, <, >=, <=..etc) to limit the matching results returned

Amazon DynamoDB

DynamoDB Table Example – Composite Key

Customer ID	Customer First Name	Last Name	Phone number	Email	Open Orders	Paid	Address
1244567	John	Smith	12345	j.smith@a.com	True	Yes	
289147	Marcia	Herndon	57849				
345987	Ahmed	Essa	98765	a.essa@Kmail.net	True	On Delivery	1234 Happiness Ct, NYC, NY 11111
5456567	Kumar	Shariyar	12568				

Primary Key – Partition Key and Performance Considerations

- For best performance you should ensure a good key design to allow for uniform distribution of read and writes across the available partitions
  - You should avoid creating hot partitions where data in a partition is accessed (read / write) more than other partitions hence, the values of the partition key are very important to be carefully designed
  - For example, if the partition key is based on events in a specific day, that means so many values and attributes will be written and read from the same location, hence, creating hot partitions

The Golden Rule

- DynamoDB is optimized for uniform distribution of items across a table's partitions, no matter how many partitions there may be.
- AWS recommends that you choose a partition key that can have a large number of distinct values relative to the number of items in the table.

Hash Key Design Use Cases

- You have two community colleges in a city, each has at least 1000 students enrolled. Combined the colleges have 40 courses students can enroll into. 95% of students are in-state, while the rest are either international or out of state.
  - How would you design your Hash/Partition key for best performance?
    - College name , Course ID, Student ID, In/out of state status?

- You client is about to launch a specialized products' e-commerce website, you are designing their DynamoDB database, they have a total of 40 products in 5 categories, The website is expected to attract thousands of customers,
  - How would you design the Hash/partition key for best performance?
    - Product ID? Category ID? Customer ID? Product Price?

Amazon DynamoDB

- Query and Scan Operations
- Indexing
- DynamoDB Secondary indexes

Query and Scan operations

DynamoDB provides the following operations for reading data:

- **GetItem –** Retrieves a single item from a table using its primary key.
  - This is the most efficient way to read a single item, because it provides direct access to the physical location of the item.
  - DynamoDB also provides the BatchGetItem operation, allowing you to perform up to 100 GetItem calls in a single operation.

- **Query –** Retrieves **all of the items that have a specific partition key.**
  - Within those items, you can apply a condition to the sort key and retrieve only a subset of the data.
  - You can use Query with any table that has a composite primary key (partition key and sort key).
    - You must specify an equality condition for the partition key, and you can optionally provide another condition for the sort key.
  - Query provides quick, efficient access to the partitions where the data is stored.

- **Scan –** Retrieves **all of the items in the specified table**.
  - This operation should not be used with large tables, because it can consume large amounts of system resources

Databases and Indexing

- A **database index** is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure.
  - Creating the index involves backfilling data from the table into the new index. During backfilling, the main table remains available.

- Indexes give you access to alternate query patterns, and can speed up queries.
  - Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed.

- Indexes can be created using one or more columns of a DB table. This provides the basis for both rapid random lookups and efficient access of ordered records.
  - An index is a copy of selected columns of data from a table that can be searched efficiently. It includes a low-level disk block address to the complete row of data it was copied from.

DynamoDB Secondary Indexes

Customer ID	Customer First Name	Last Name	Phone number	Email	Open Orders	Paid	Address
1244567	John	Smith	12345	j.smith@a.com	True	Yes	
289147	Marcia	Herndon	57849				
345987	Ahmed	Essa	98765	a.essa@Kmail.net	True	On Delivery	1234 Happiness Ct, NYC, NY 11111
5456567	Kumar	Shariyar	12568				

DynamoDB Secondary Indexes

Customer ID	Customer First Name	Last Name	Phone number	Email	Open Orders	Paid	Address
1244567	John	Smith	12345	j.smith@a.com	True	Yes	
289147	Marcia	Herndon	57849				
345987	Ahmed	Essa	98765	a.essa@Kmail.net	True	On Delivery	1234 Happiness Ct, NYC, NY 11111
5456567	Kumar	Shariyar	12568				

Secondary Indexes

- Indexes give you access to alternate query patterns and can speed up queries.
- In DynamoDB, When you create a secondary index, you must specify its key attributes – a partition key and a sort key.
- After you create the secondary index, you can Query it or Scan it just as you would with a table. You must specify both TableName and IndexName.
- DynamoDB supports two different kinds of indexes:
  - **Local secondary indexes** – The partition key of the index must be the same as the partition key of the DynamoDB table. However, the sort key can be any other attribute.
  - **Global secondary indexes** – The primary key (Partition and Sort keys) of the index can be any two attributes from the DynamoDB table.
- Each table is limited to 20 global secondary indexes (default limit) and 5 local secondary indexes.
- Every index belongs to a table, which is called the *base table* for the 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.

Amazon DynamoDB

- Local Secondary Indexes
- Global Secondary Indexes

DynamoDB Local Secondary Indexes

- **Local secondary indexes** – The partition key of the index must be the same as the partition key of the DynamoDB table. However, the sort key can be any other attribute.

- DynamoDB ensures that the data in a secondary index is eventually consistent with its table. You can request strongly consistent Query or Scan actions on a table or a local secondary index. Global secondary indexes only support eventual consistency.

- Local Secondary Index can be created at the base table creation time only
  - Once created it can not be edited, or deleted

- Local Secondary indexes share the provisioned throughput capacity of the base DynamoDB table

- 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.

DynamoDB Global Secondary Indexes

- **Global secondary indexes** – The primary key (Partition and Sort keys) of the index can be any two attributes from the DynamoDB table.
- You can add a global secondary index to an existing table,
- Global Secondary index has its own Throughput capacity (Reads and Writes capacity), which is separate from those of the base table.
- Global secondary indexes in DynamoDB are also composed of partitions.
  - The data in a GSI is stored separately from the data in its base table, but index partitions behave in much the same way as table partitions.
- 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.
- You should always rely on Global Secondary Indexes rather than local secondary indexes.
- However, when you need strong consistency in your query results, then you need to use local secondary indexes can provide but a global secondary index cannot (global secondary index queries only support eventual consistency).

Amazon DynamoDB

- Storing large items
- Sparse indexes

Storing large items

- If your application needs to store more data in an item than the DynamoDB size limit permits, you can:
  - Try compressing one or more large attributes, or
  - Store them as an object in Amazon Simple Storage Service (Amazon S3) and store the Amazon S3 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.

Amazon DynamoDB

Sparse Indexes

Customer ID	Customer First Name	Last Name	Phone number	Email	Open Orders	Paid	Address
1244567	John	Smith	12345	j.smith@a.com	True	Yes	
289147	Marcia	Herndon	57849				
345987	Ahmed	Essa	98765	a.essa@Kmail.net	True	On Delivery	1234 Happiness Ct, NYC, NY 11111
5456567	Kumar	Shariyar	12568				

GSI of the previous table

Customer ID	Open Orders	Customer First Name	Last Name	Phone number	Email	Paid	Address
1244567	True	John	Smith	12345	j.smith@a.com	Yes	
345987	True	Ahmed	Essa	98765	a.essa@Kmail.net	On Delivery	1234 Happiness Ct, NYC, NY 11111

Sparse Secondary Indexes

- For any item in a table, DynamoDB writes a corresponding index entry **only if the index sort key value is present in the item**.
  - If the sort key doesn't appear in every table item, the index is said to be *sparse*.
  - When you have thousands of orders of which only a small number are open, it's faster and less expensive to query that index than to scan the entire table.

- Sparse indexes are useful for queries over a small subsection of a table, resulting better performance

- Global Secondary Indexes are Sparse by default
  - Only items in the parent table that contain the attributes, which the GSI uses as Partition and Sort keys, appear in the index.

Sparse Secondary Indexes

- Good examples:
  - IVR calls tracked in a DynamoDB table and you want to track active calls at any given time. Add an attribute for active calls, and create a GSI based on that attribute as Partition or Sort key
  - Tracking open orders, create an attribute that shows order open date, and include it only for open orders. Create a GSI based on Open orders attribute as partition or sort key
  - Listing highest scores and winners in a competition, include an award/winner attribute for each player per game, and create a GSI based on award/winner attribute as the partition or sort key.

Amazon DynamoDB

- DynamoDB Local
- Backup
- Point in Time Recovery
- Time To Live (TTL)
- DynamoDB Accelerator (DAX)
- DynamoDB Streams
- DynamoDB Transactions

DynamoDB Local

- You can download a version of DynamoDB that is self contained on a local computer
- This enables you to write and/or test application without having to access the AWS DynamoDB service
- With minor code changes, this can then be deployed to the DynamoDB web service when ready

On-Demand Backup and Restore for DynamoDB

- Amazon DynamoDB provides on-demand backup capability.
  - This allows you to create full backups of your tables for long-term retention and archival for regulatory compliance needs.
  - Backup and restore actions execute with zero impact on table performance or availability.
- Backup can be completed in seconds regardless of the table size.
- All backups persist until manually deleted
- The backup and restore functionality works in the same Region as the source table.
- They do not charge any extra cost beyond the storage costs
- All backups in DynamoDB work without consuming any provisioned throughput on the table
- Global and Local secondary indexes, DynamoDB Streams, and Provisioned Read/Write capacity are included

DynamoDB Point In Time Recovery (PITR)

- On-demand backups and point in time recovery can be enabled for your Amazon DynamoDB tables
- PITR provides continuous backups of your DynamoDB table data.
  - This will help protect against accidental table or item deletions.
  - With this feature enabled, there is no need to worry about creating, maintaining, and scheduling on-demand backups
- It is disabled by default, you need to enable it
- Global and Local Secondary indexes, Provisioned Read/Write capacity and Encryption settings are also restored to the new table
  - Recovery happens to a new table, not the same base table.

DynamoDB TTL

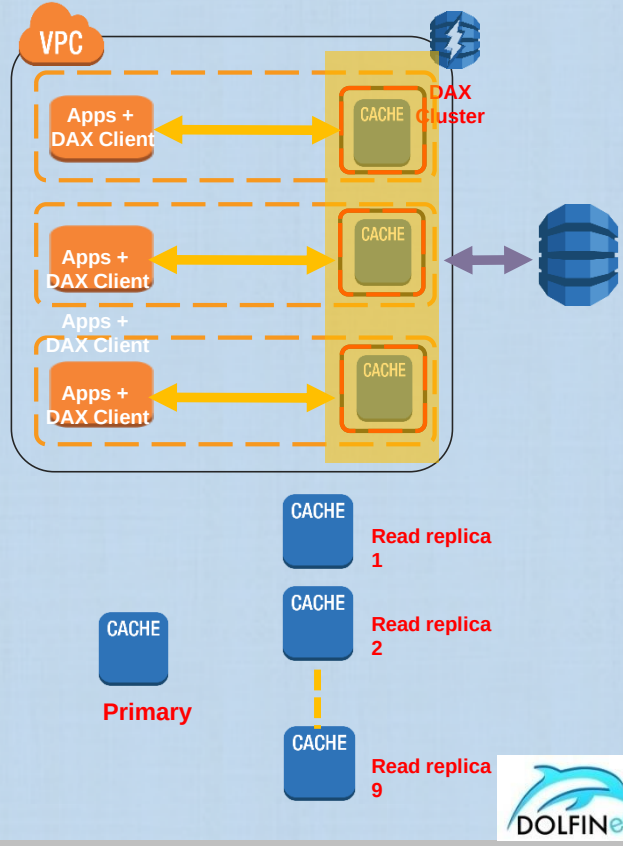
- Amazon DynamoDB Time to Live (TTL) defines when items in a table expire so that they can be automatically deleted from the database.
  - This is particularly helpful in reducing storage usage by deleting the irrelevant data without consuming from the provision throughput
- TTL does not cost extra.
- With TTL enabled on a table, you need to create an attribute in each item (expiry column) which defines a timestamp for deletion on a per item basis.
- Examples of data that can benefit from TTL, session state 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.

In-Memory Acceleration with DynamoDB Accelerator (DAX)

- DAX is a DynamoDB-compatible caching service that enables you to benefit from fast in-memory performance for demanding applications.
  - It is designed for DynamoDB only and NOT for other AWS services.
- DynamoDB is designed to provide a consistent milliseconds response time
- Using DynamoDB DAX provides access to microsecond latency data reads from DynamoDB tables.
  - DAX scales where a Multi-AZ DAX cluster can serve millions of requests per second.
- DAX is for eventually **consistent reads only, not for strongly consistent reads**
- DAX is deployed as a cluster in a VPC (default VPC by default)

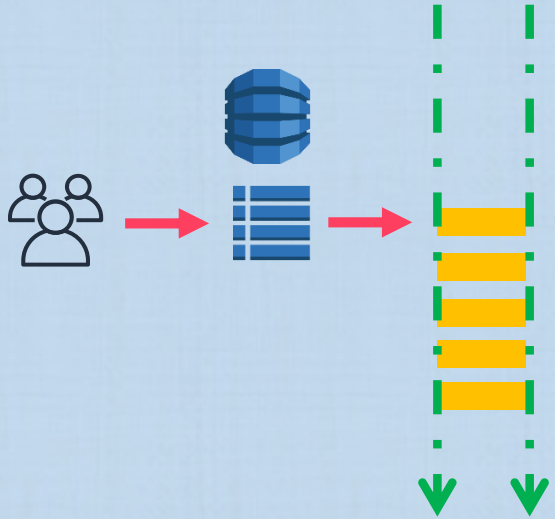
DynamoDB Accelerator (DAX) – Clusters and Fault tolerance

- Applications that use DAX will be deployed on Amazon EC2 instances, in your VPC, with the **DAX Client**.
- DAX cluster consists of one or more nodes (up to 10).
- Each node runs its own instance of the DAX caching software.
- One of the nodes serves as the primary node for the cluster. Additional nodes (if present) serve as read replicas
- For a fault tolerant cluster, AWS recommends deploying a minimum of 3 nodes in 3 different AZs
- DAX cluster in an AWS Region can only interact with DynamoDB tables that are in the same Region.
  - It is also designed for use with DynamoDB only, not other AWS services



DynamoDB Streams

- Having the ability to capture changes to the items in a DynamoDB table, at the time when the change occurs, can be useful in many applications.
- A DynamoDB stream is an ordered flow (carries sequence numbers) of data/information about changes to items in a DynamoDB table.
- DynamoDB Streams captures a time-ordered (with sequence numbers and time stamps) of item-level modifications in any DynamoDB table
  - This data is stored for 24 hours, after which it gets removed from the stream automatically.
- Applications can access this log and view the data items as they appeared before and after they were modified, in near-real time.
  - DynamoDB streams operate asynchronously with no performance hit on the table.
- Encryption at rest encrypts the data in DynamoDB streams.



DynamoDB Transactions

- DynamoDB **transactional read and write** APIs can be used to manage complex business workflows that require adding, updating, or deleting **multiple items as a single, all-or-nothing operation**.
- Amazon DynamoDB transactions simplify the development side of making **coordinated, all-or-nothing changes to multiple items both within and across tables.**
- For example, a video game developer can ensure that players' profiles are updated correctly when they exchange items in a game or make in-game purchases.
- Transactions provide atomicity, consistency, isolation, and durability (ACID) in DynamoDB, helping to maintain applications data correctness.
- With the transaction write API, it is possible to group multiple Put, Update, Delete, and Condition Check actions.

DynamoDB Transactions

- With the transaction write API, it is possible to group multiple Put, Update, Delete, and Condition Check actions.
- The actions can then be submitted as a single **TransactWriteItems operation** that either succeeds or fails as a unit.
- The same is true for multiple Get actions, which can be grouped and submitted as a single **TransactGetItems operation**.
- For every item involved in a DynamoDB transaction, DynamoDB performs two underlying reads (2 RCUs) or two writes (2 WCUs):
  - One to prepare the transaction and one to commit the transaction.
  - These two-underlying read/write operations are visible in your Amazon CloudWatch metrics.
- There is no additional cost to enable transactions for DynamoDB tables.
  - Charges are only for the reads or writes (WCUs and RCUs) that are part of the transaction.

Amazon DynamoDB

Pricing

Amazon DynamoDB – Cost Model

- With Amazon DynamoDB, you pay only for what you use and there is no minimum fee.

- Amazon DynamoDB has three pricing components:
  - Provisioned throughput capacity (Reads and Writes monthly, pro rated),
  - Indexed data storage (per GB per month),
  - Data transfer in or out (per GB per month).

DynamoDB - Pricing

As a rule of thumb, reads are cheaper than writes when using DynamoDb

You pay for :

- Each table's provisioned read/wrote throughput (Hourly rates)
  - You are charged for the provisioned throughput regardless whether you use it or not
- Indexed Data Storage
- Internet Data Transfer (if crosses a region)
- Free tier per account (across all tables) of 25 read capacity units, and 25 write capacity units pe month

Source:aws.amazon.com