Amazon DynamoDB

Characteristic	Relational database management system (RDBMS)	Amazon DynamoDB
Tools for Accessing the Database	Most relational databases provide a command line interface (CLI) so that you can enter ad hoc SQL statements and see the results immediately.	In most cases, you write application code. You can also use the AWS Management Console, the AWS Command Line Interface (AWS CLI), or NoSQL Workbench to send ad hoc requests to DynamoDB and view the results. PartiQL, a SQL-compatible query language, lets you select, insert, update, and delete data in DynamoDB.
Connecting to the Database	An application program establishes and maintains a network connection with the database. When the application is finished, it terminates the connection.	DynamoDB is a web service, and interactions with it are stateless. Applications do not need to maintain persistent network connections. Instead, interaction with DynamoDB occurs using HTTP(S) requests and responses.
Authentication	An application cannot connect to the database until it is authenticated. The RDBMS can perform the authentication itself, or it can offload this task to the host operating system or a directory service.	Every request to DynamoDB must be accompanied by a cryptographic signature, which authenticates that particular request. The AWS SDKs provide all of the logic necessary for creating signatures and signing requests. For more information, see Signing AWS API requests in the AWS General Reference.
Authorization	Applications can perform only the actions for which they have been authorized. Database administrators or application owners can use the SQL GRANT and REVOKE statements to control access to database objects (such as tables), data (such as rows within a table), or the ability to issue certain SQL statements.	In DynamoDB, authorization is handled by AWS Identity and Access Management (IAM). You can write an IAM policy to grant permissions on a DynamoDB resource (such as a table), and then allow users and roles to use that policy. IAM also features fine-grained access control for individual data items in DynamoDB tables. For more information, see Identity and Access Management for Amazon DynamoDB.
Sending a Request	The application issues a SQL statement for every database operation that it wants to perform. Upon receipt of the SQL statement, the RDBMS checks its syntax, creates a plan for performing the operation, and then runs the plan.	The application sends HTTP(S) requests to DynamoDB. The requests contain the name of the DynamoDB operation to perform, along with parameters. DynamoDB runs the request immediately.
Receiving a Response	The RDBMS returns the results from the SQL statement. If there is an error, the RDBMS returns an error status and message.	DynamoDB returns an HTTP(S) response containing the results of the operation. If there is an error, DynamoDB returns an HTTP error status and messages.

- Amazon DynamoDB is a fully managed NoSQL database service and is schemeless.
- other than the primary key attributes, you don't have to define any attributes or data types when you create tables.
- Scales to massive workloads, distributed database. (Scales horizontally)
- In DynamoDB, tables, items, and attributes are the core components.
- A table is a collection of items, and each item is a collection of attributes.
- DynamoDB uses primary keys to uniquely identify each item in a table and secondary indexes to provide more querying flexibility.
- DynamoDB is made of Tables
- Each table has a Primary Key (must be decided at creation time)
- Each table can have an infinite number of items (= rows)
- Each item has attributes (can be added over time – can be null)
- Maximum size of an item is 400KB

Supported data types :

- Scalar types : String, Number, Binary, Boolean, Null
- Document type : List, Map
- Set types : String Set, Number Set, Binary Set

DynamoDB – Primary Keys

- Two types of primary keys :
  Partition Key (HASH attribute) :
  - composed of one attribute
  - must be unique for each item
  - 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.

Partition Key + Sort Key (HASH attribute + RANGE attribute)

- composed of two attributes (partition key and sort key)
- the combination must be unique for each item.
- 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.

DynamoDB – Secondary indexes :

- Secondary indexes are alternate keys in addition to queries against primary keys.
- Not mandatory, but give applications more flexibility when querying your data.
- 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.
- Two kinds of indexes :

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

  - "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.
  - total size of indexed items for any one partition key value can't exceed 10 GB
  - shares provisioned throughput settings for read and write activity with the table it is indexing.
  - can have up to 5 local secondary indexes

  Global secondary index – An index with a partition key and 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.
  - no size limitations, separate provisioned throughput settings for read and write activity from base table.
  - can have up to 20 global secondary indexes (default quota)
  - Global secondary indexes are sparse by default.

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.	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. A global secondary index associated with global tables consumes write capacity units.	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, and these updates consume write capacity units from the base table. A local secondary index associated with global tables consumes replicated write capacity units.
Projected Attributes	With global secondary index queries or scans, you can only request the attributes that are projected into the index. DynamoDB does 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 automatically fetches those attributes from the table.

DynamoDB Time to Live (TTL) :

- Time To Live (TTL) for DynamoDB is a cost-effective method for deleting items that are no longer relevant.
- Automatically delete items after an expiry timestamp.
- Once deleted, items go into DynamoDB Streams as service deletions instead of user deletes.
- TTL deletions can be identified in DynamoDB Streams, but only in the Region where the deletion occurred.
- Doesn’t consume any WCUs (i.e., no extra cost)
- The TTL attribute must be a “Number” data type with “Unix Epoch timestamp” value. (expiration time must be in epoch format, in seconds).
- If you use any other format, the TTL processes ignore the item.
- The item will be expired after the specified time.
- Items that are deleted by the Time to Live process after expiration have the following fields:

  Records[<index>].userIdentity.type "Service"

  Records[<index>].userIdentity.principalId "dynamodb.amazonaws.com"

- Expired items deleted within 48 hours of expiration
- Expired items, that haven’t been deleted, appears in reads/queries/scans (if you don’t want them, filter them out).These items still count towards storage and read costs until they are deleted.
- Expired items are deleted from both LSIs and GSIs.
- A delete operation for each expired item enters the DynamoDB Streams (can help recover expire items).
- TTL can be enabled in the Amazon DynamoDB Console, the AWS Command Line Interface (AWS CLI), or using the Amazon DynamoDB API Reference with any of the supposed AWS SDKs.
- It takes approximately one hour for TTL to be enabled across all partitions.
- Use cases: reduce stored data by keeping only current items, adhere to regulatory obligations etc.

DynamoDB Streams :

- Dynamo DB captures time-ordered stream of item-level modifications (create/update/delete) in a table. (optional feature - we can enable/disable this feature).

- DynamoDB Streams operates asynchronously, so there is no performance impact on a table if you enable a stream.
- Each event is represented by a stream record.
- A stream record contains : item-level modifications (create/update/delete) in a table, along with table name, event timestamp, other metadata.
- Each stream record appears exactly once in the stream.
- For each item that is modified in a DynamoDB table, the stream records appear in the same sequence as the actual modifications to the item.
- Stream records have a lifetime of 24 hours; after that, they are automatically removed from the stream.
- DynamoDB Streams writes stream records in near-real time so that you can build applications that consume these streams and take action based on the contents.
- AWS maintains separate endpoints for DynamoDB and DynamoDB Streams.
- To read and process DynamoDB Streams records, your application must access a DynamoDB Streams endpoint in the same Region.
- The naming convention for DynamoDB Streams endpoints is streams.DynamoDB..amazonaws.com. e.g. dynamodb.us-west-2.amazonaws.com
- The AWS SDKs provide separate clients for DynamoDB and DynamoDB Streams.
- We can enable/disable DynamoDB streams any time using AWS management console, AWS CLI or one of the AWS SDKs.
- Following information need to choose that will be written to the streams :

  KEYS_ONLY — Only the key attributes of the modified item.

  NEW_IMAGE — The entire item, as it appears after it was modified.

  OLD_IMAGE — The entire item, as it appeared before it was modified.

  NEW_AND_OLD_IMAGES — Both the new and the old images of the item.

- If we try to enable a stream on a table that already has a stream we get ResourceInUseException.
- If we try to disable a stream on a table that doesn't have a stream we get ValidationException.
- DynamoDB Streams are made of shards, just like Kinesis Data Streams
- You don’t provision shards, this is automated by AWS. Shards are ephemeral: They are created and deleted automatically, as needed.
- Any shard can also split into multiple new shards; this also occurs automatically.
- No more than two processes at most should be reading from the same stream's shard at the same time. Having more than two readers per shard can result in throttling.
- Use cases:
  - react to changes in real-time (welcome email to users)
  - Analytics
  - Insert into derivative tables
  - Insert into OpenSearch Service
  - Implement cross-region replication

DynamoDB API

- To work with Amazon DynamoDB, your application must use a few simple API operations.

- Control plane :
  - 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 :
  - 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.
  - 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, 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
  - Scan – Retrieves all items in the specified table or index. You can retrieve entire items, or just a subset of their attributes. Can apply a filtering condition
  - UpdateItem – Modifies one or more attributes in an item. You must specify the primary key for the item that you want to modify.
  - 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, more efficient than calling DeleteItem multiple times because application only needs a single network round trip to delete the items.
- Important attributes :
  - ProjectionExpression : You can add a ProjectionExpression parameter to return only some of the attributes.
  - KeyConditionExpression : The KeyConditionExpression parameter specifies the key values that you want to query.
  - FilterExpression : optional FilterExpression to remove certain items from the results before they are returned to you.
  - UpdateExpression :
  - ConditionExpression : performs operation(read, update, delete, write) only if a specific ConditionExpression evaluates to true. 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.
  - BatchExecuteStatement – Writes, updates or reads multiple items from a table, more efficient than ExecuteStatement because application only needs a single network round trip to write/read items.
- DynamoDB Stream :

- ListStreams – Returns a list of all your streams, or just the stream for a specific table.
- DescribeStream – Returns information about a stream, such as its Amazon Resource Name (ARN) and where your application can begin reading the first few stream records.
- GetShardIterator – Returns a shard iterator, which is a data structure that your application uses to retrieve the records from the stream.
- GetRecords – Retrieves one or more stream records, using a given shard iterator.

- Transactions :
  - ExecuteTransaction – A batch operation that allows CRUD operations to multiple items both within and across tables with a guaranteed all-or-nothing result. (PartiQL)
  - TransactWriteItems – A batch operation that allows Put, Update, and Delete operations to multiple items both within and across tables with a guaranteed all-or-nothing result.
  - TransactGetItems – A batch operation that allows Get operations to retrieve multiple items from one or more tables.

Read consistency

- Amazon DynamoDB reads data from tables, local secondary indexes (LSIs), global secondary indexes (GSIs), and streams.
- There are two read consistency options:
  - eventually consistent (default)
    - supported on tables, LSIs, GSIs and streams
    - may not reflect the results of a recently completed write operation
    - If you repeat your read request after a short time, the response should eventually return the more recent item.
    - Eventually consistent reads are half the cost of strongly consistent reads.
  - Strongly consistent reads
    - only supported on tables and local secondary indexes
    - If we read just after a write, we will get the correct data
    - Set “ConsistentRead” parameter to True in API calls (GetItem, BatchGetItem, Query, Scan)
    - Consumes twice the RCU
  - Global tables read consistency
    - A global table is composed of multiple replica tables in different AWS Regions.
    - Any change made to any item in any replica table is replicated to all the other replicas within the same global table, typically within a second, and are eventually consistent.

Read/write capacity mode

- The read/write capacity mode controls how you are charged for read and write throughput and how you manage capacity.
- Secondary indexes inherit the read/write capacity mode from the base table.
- There are two modes :
  - Provisioned (default, free-tier eligible)
    - you specify the number of reads and writes per second
    - You need to plan capacity beforehand
    - Pay for provisioned read & write capacity units

- Good option when :
    - You have predictable application traffic.
    - traffic is consistent or ramps gradually.
    - You can forecast capacity requirements to control costs.

- On Demand
    - Read/writes automatically scale up/down with your workloads.
    - automatically adapt to your application's traffic volume.
    - accommodates double than previous traffic peak on table,
    - e.g. 50,000 reads per second in the previous traffic peak, on-demand capacity mode instantly accommodates traffic of up to 100,000 reads per second.
    - However, throttling can occur if you exceed double your previous peak within 30 minutes.
    - No capacity planning needed (WCU / RCU)
    - Unlimited WCU & RCU, no throttle, more expensive
    - You're charged for reads/writes that you use in terms of RRU and WRU
    - Read Request Units (RRU) – throughput for reads (same as RCU)
    - Write Request Units (WRU) – throughput for writes (same as WCU)
    - 2.5x more expensive than provisioned capacity (use with care)
    - Use cases: unknown workloads, unpredictable application traffic

DynamoDB read requests can be either strongly consistent, eventually consistent, or transactional.

- A *strongly consistent* read request of an item up to 4 KB requires one read request unit.
- An *eventually consistent* read request of an item up to 4 KB requires one-half read request unit.
- A *transactional* read request of an item up to 4 KB requires two read request units.

If you need to read an item that is larger than 4 KB, DynamoDB needs additional read request units. The total number of read request units required depends on the item size, and whether you want an eventually consistent or strongly consistent read. For example, if your item size is 8 KB, you require 2 read request units to sustain one strongly consistent read, 1 read request unit if you choose eventually consistent reads, or 4 read request units for a transactional read request.

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. Transactional write requests require 2 write request units to perform one write for items up to 1 KB. The total number of write request units required depends on the item size. For example, if your item size is 2 KB, you require 2 write request units to sustain one write request or 4 write request units for a transactional write request.

DynamoDB – Write Capacity Units (WCU)

- One *Write Capacity Unit (WCU)* represents one write per second for an item up to 1 KB in size
- If the items are larger than 1 KB, more WCUs are consumed

- **Example 1:** we write 10 items per second, with item size 2 KB
  - We need $10 * (\frac{2\ KB}{1\ KB}) = 20\ WCUs$
- **Example 2:** we write 6 items per second, with item size 4.5 KB
  - We need $6 * (\frac{5\ KB}{1\ KB}) = 30\ WCUs$ (4.5 gets rounded to the upper KB)
- **Example 3:** we write 120 items per minute, with item size 2 KB
  - We need $\left(\frac{120}{60}\right) * (\frac{2\ KB}{1\ KB}) = 4\ WCUs$

DynamoDB – Throttling

- If we exceed provisioned RCUs or WCUs, we get HTTP 400 code (Bad Request) and "ProvisionedThroughputExceededException".
- Reasons:
  - Hot Keys – one partition key is being read too many times (e.g., popular item)
  - Hot Partitions
  - Very large items, remember RCU and WCU depends on size of items
- Solutions:
  - Exponential backoff when exception is encountered (already in SDK)
  - Distribute partition keys as much as possible
  - If RCU issue, we can use DynamoDB Accelerator (DAX)

Partitions and data distribution

- Amazon DynamoDB stores data in partitions.
- Partition management occurs automatically in the background- you never have to manage partitions yourself.
- While creating table (status = CREATING), DynamoDB allocates partitions based on provisioned throughput (RCUs and WCUs).
- We can begin reading and writing data when table is created. (status = ACTIVE)
- Amazon DynamoDB allocates additional partitions to a table :
  - If you increase the table's provisioned throughput

    - If an existing partition fills to capacity and more storage space is required.
- Global secondary indexes in DynamoDB are also composed of partitions.
- The data in a global secondary index is stored separately from the data in its base table, but index partitions behave in much the same way as table partitions.
- Partition Keys go through a hashing algorithm to know to which partition they go to.
- WCUs and RCUs are spread evenly across partitions

To compute the number of partitions:

- $\#\ of\ partitions_{by\ capacity} = \left(\frac{RCUs_{Total}}{3000}\right) + \left(\frac{WCUs_{Total}}{1000}\right)$
- $\#\ of\ partitions_{by\ size} = \frac{Total\ Size}{10\ GB}$
- $\#\ of\ partitions = \text{ceil}(\max(\#\ of\ partitions_{by\ capacity}, \#\ of\ partitions_{by\ size}))$

On-Demand backup and restore for DynamoDB :

- All on-demand backups are catalogued, discoverable, and retained until they are explicitly deleted.
- The on-demand backup and restore process scales without degrading the performance or availability of your applications.
- uses a new and unique distributed technology to complete backups in seconds regardless of table size
- backups that are consistent within seconds across thousands of partitions
- There are 2 options for creating and managing DynamoDB on-demand backups :
    - Amazon Backup service
        - you can copy your on-demand backups across AWS accounts and Regions
        - add allocation tags
        - transition on-demand backups to cold storage for lower costs
        - Backups are stored in an encrypted vault with a key that is managed by the AWS Backup service.
        - The backup files have an AWS Backup ARN
        - backups can only be deleted from the AWS Backup vault, cannot delete from dynamodb console
    - DynamoDB
        - no additional cost beyond the normal pricing that's associated with backup storage size
        - backups cannot be copied to a different account or Region.

Point-in-time recovery for DynamoDB :

- Amazon DynamoDB point-in-time recovery (PITR) provides automatic backups of your DynamoDB table data.

- You can use backups or PITR
- you can enable PITR using AWS management console, AWS CLI or DynamoDB API.
- Once enabled, point-in-time recovery provides continuous backups until you explicitly turn it off.
- The point-in-time recovery process always restores to a new table.
- you can restore to any point in time within EarliestRestorableDateTime and LatestRestorableDateTime.
- LatestRestorableDateTime : 5 minutes before current time
- EarliestRestorableDateTime : last 35 days - fixed 35 days (5 calendar weeks) and can't be modified.
- If you disable point-in-time recovery and later re-enable it on a table, you reset the start time
- restore table has same RCU and WCU as source table.

In-memory acceleration with DynamoDB Accelerator (DAX)

- DAX is a DynamoDB-compatible caching service
- fast in-memory performance, fast response times for accessing eventually consistent data.
- responds in microseconds
- DAX addresses three core scenarios:
    1. DAX reduces the response times of eventually consistent read workloads from single-digit milliseconds to microseconds.
    2. DAX reduces operational and application complexity by providing a managed service that is API-compatible, requires minimal functional changes
    3. For read-heavy or busty workloads, DAX provides increased throughput and potential operational cost savings
- DAX supports server-side encryption, encryption at rest, encryption in transit.

- DAX runs within VPC dedicated to your AWS account and is isolated from other VPCs. A security group acts as a virtual firewall for your VPC,
- Unless you specify otherwise, your DAX cluster runs within your default VPC.
- To run your application, you launch an Amazon EC2 instance into your Amazon VPC, add an ingress rule to your security group --> protocol (TCP) and port number (8111)
- You then deploy your application (with the DAX client) on the EC2 instance.
- A DAX cluster consists of one or more nodes.
- Each of the individual nodes in a DAX cluster has its own hostname and port number.
- Each node runs its own instance of the DAX caching software and maintains a single replica of the cached data.
- You can scale DAX cluster either by adding more nodes or using larger node type.
- A DAX cluster can support up to 11 nodes per cluster (the primary node plus a maximum of 10 read replicas).
- Scaling : if large number of requests to access DAX, add more number of read-replicas.
- High Availability : if primary node fails, DAX automatically fails over to a read replica and designates it as the new primary.
- If a replica node fails, other nodes in the DAX cluster can still serve requests until the failed node can be recovered.

- For maximum fault tolerance, you should deploy read replicas in separate Availability Zones.
- One of the nodes serves as the primary node for the cluster.
- Additional nodes (if present) serve as read replicas. DAX automatically keeps the replicas in sync with the primary node.
- Your app can access DAX cluster using endpoint for the DAX cluster.
- Every DAX cluster has two distinct caches—an item cache and a query cache.
- DAX supports negative cache entries in both the item cache and the query cache.
- Negative caching : when DAX can't find requested items in an underlying DynamoDB table, Instead of generating an error, DAX caches an empty result and returns that result to the user.
- Negative cache entry remains until TTL expired, LRU invoked or item is modified using PutItem, UpdateItem, or DeleteItem.
- Read operations : GetItem, BatchGetItem, Query, Scan
- If DAX has the item available (a cache hit), DAX returns the item to the application without accessing DynamoDB.
- If DAX does not have the item available (a cache miss), DAX passes requests to DynamoDB, also writes results to the cache on the primary node.
- If the request specifies strongly consistent reads, DAX passes requests to DynamoDB, results from DynamoDB are not cached in DAX.
- DAX handles TransactGetItems requests the same way it handles strongly consistent reads. DAX passes requests to DynamoDB, results from DynamoDB are not cached in DAX.
- DAX also maintains a query cache to store the results from Query and Scan operations.
- DAX also maintains an LRU list for the query cache.
- If the query cache becomes full, DAX evicts older result sets (even if they have not expired yet) to make room for new result sets.
- If you specify zero as the query cache TTL setting, the query response will not be cached.
- Write Operations : PutItem, BatchWiteItem, UpdateItem, DeleteItem : writes are always first on table then to DAX.
- DAX maintains an item cache to store the results from GetItem and BatchGetItem operations.
- If a write to DynamoDB fails for any reason, including throttling, the item is not cached in DAX.
- The item cache has a Time to Live (TTL) setting, which is 5 minutes by default.
- DAX assigns a timestamp to every item that it writes to the item cache.
- DAX also maintains a least recently used (LRU) list for the item cache.
- If the item cache becomes full, DAX evicts older items (even if they haven't expired yet) to make room for new items
- The LRU algorithm is always enabled for the item cache and is not user-configurable.
- If you specify zero as the item cache TTL setting, items in the item cache will only be refreshed due to an LRU eviction or a "write-through" operation.
- DAX does not recognize any DynamoDB operations for managing tables (such as CreateTable, UpdateTable, and so on).
- At runtime, the DAX client directs all of your application's DynamoDB API requests to the DAX cluster.
- DAX not ideal for strong consistent reads, write-intensive apps, apps with own client-side logic for working with that caching solution.
- Request rate limiting :

1. If the number of requests sent to DAX exceeds the capacity of a node, DAX limits the rate at which it accepts additional requests by returning a ThrottlingException.
2. DAX continuously evaluates CPU utilization to maintain healthy cluster.
3. sends the ThrottledRequestCount metric that DAX publishes to Amazon CloudWatch.

- A DAX cluster in an AWS Region can only interact with DynamoDB tables that are in the same Region.
- For production usage, we strongly recommend using DAX with at least three nodes, where each node is placed in different Availability Zones.
- Three nodes are recommended in production for a DAX cluster to be fault-tolerant. (configured in Multi-AZ).
- Monitoring DAX :
  1. CloudWatch alarms, logs, events,
  2. CloudTrail logs
  3. Use AWS CloudTrail to monitor AWS managed KMS key usage
  4. Monitor DynamoDB operations using CloudTrail - monitor CRUD operations
  5. Use DynamoDB Streams to monitor data plane operations
     - Immediately after an item in the table is modified, a new record appears in the table's stream.
     - AWS Lambda polls the stream and invokes your Lambda function synchronously when it detects new stream records.
     - The Lambda function can perform any actions that you specify, such as sending a notification or initiating a workflow.
  6. Monitor DynamoDB configuration with AWS Config - continuously monitor and record configuration changes of your AWS resources.

Security :

- For users, applications, and other AWS services to access DynamoDB, they must include valid AWS credentials in their AWS API requests.
- Attach permissions policies to IAM identities (that is, users, groups, and roles) and thereby grant permissions to perform operations on DynamoDB resources.
- If you only require access to DynamoDB from within a VPC, you should use a VPC endpoint to limit access from only the required VPC.
- Recourse Based policies : let you define access permissions by specifying who has access to each resource, and the actions they are allowed to perform on each resource.
- All user data stored in Amazon DynamoDB is fully encrypted at rest (AES-256) using encryption keys stored in AWS Key Management Service (AWS KMS).
- You can switch between these key types at any time, (AWS Owned, AWS managed, Customer owned)
- Data in transit: All your data in DynamoDB is encrypted in transit.(HTTPS or SSL/TLS)
- Data can be protected before sending it to DynamoDB using Client Side Encryption
- DynamoDB backups are encrypted, and the table that is restored from a backup also has encryption enabled.
- IAM policy conditions for fine-grained access control :
  - Grant permissions to allow users read-only access to certain items and attributes

    - Grant permissions to allow users write-only access to certain attributes in a table, based upon the identity of that user.
- You can use web identity federation for authentication and authorization -removes the need for creating individual users instead use AWS STS for temp credentials.

Best Practices :

- Understand key differences before migrating relational DB to NoSQL like DynamoDB
- Consider Data size, Data Shape, Data Velocity before approaching NoSQL DB
- For all active production tables, turn on Deletion protection
- Using burst capacity effectively - Whenever available throughput is not fully utilized , 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.
- Using write Sharding to distribute workloads evenly --> add a random number to the end of the partition key values.
- Keep the number of indexes to a minimum. Don’t create secondary indexes on attributes that you don't query often.
- Because secondary indexes consume storage and provisioned throughput, you should keep the size of the index as small as possible.
- Take advantage of sparse indexes. useful for queries over a small subsection of a table.
- DynamoDB default item size is400KB, if want to store larger item, try compressing or breaking the item into multiple items or store it as a object in Amazon S3 & S3 object identifier in DynamoDB.
- utilize the ReturnConsumedCapacity parameter when writing items to monitor and alert on items sizes that approach the 400 KB.
- keep the number of tables you use to a minimum.
- However, for time series data, you can often best handle it by using one table per application per period.
    - create one table for current period, allocate required read and write capacity and the required indexes.
    - Just before current period ends, prebuild new table and direct events to new table
    - reduce provisioned write capacity to a lower value and Reduce the provisioned read capacity of earlier tables as they age.
    - choose to archive or delete the tables whose contents are rarely or never needed.
- Manage many-to-many relationships using Adjacency lists design pattern.
    - When different entities of an application have a many-to-many relationship between them, the relationship can be modelled as an adjacency list.
    - all top-level entities are represented using the partition key.
    - Any relationships with other entities are represented as an item within the partition by setting the value of the sort key to the target entity ID
    - A real-world example where this pattern has been useful is an invoicing system where invoices contain multiple bills. One bill can belong in multiple invoices. The partition key in this example is

either an InvoiceID or a BillID. BillID partitions have all attributes specific to bills. InvoiceID partitions have an item storing invoice-specific attributes, and an item for each BillID that rolls up to the invoice.

- Use hybrid DynamoDB-RDBMS, when you want to rely primarily on DynamoDB, but you also want to maintain a small relational system for one-time queries, or for operations that need special security or that are not time-critical.
- avoid using a Scan operation on a large table or index with a filter.
- Scan operations are less efficient than Query.
- Scan - always scans entire table/index, and then filters values, adding the extra step of removing data from the result set.
- Scan sometimes can also cause ProvisionedThroughputExceeded exception because it performs eventually consistent reads by default and can return up to 1 MB data = 128 read operations if you request Strong consistent reads, it would use twice as much provisioned throughput—256 read operations causes sudden spikes in read activity. 20.Can minimize Scan impact by
  - Reducing page size - set the page size for your request using a Limit parameter
  - Isolate scan operation - create tables for distinct purposes,
  - use parallel Scan if table size >= 20GB, RCUs are not fully being used or sequential scan are too slow. Multiple workers scan multiple data segments at the same time.
  - Choosing TotalSegments - set value for TotalSegments based on specific data, provisioned capacity and performance requirement
- keep the number of tables you use to a minimum. Recommended using a single table.
- The per account limit cannot be increased above 10,000 tables per account.
- Consider control plane limits for concurrent control plane operations
- Work with AWS solution architects to validate your design patterns for multi-tenant designs.