DynamoDB

If you are studying for AWS Developer Associate Exam, this guide will help you with quick revision before the exam. it can use as study notes for your preparation.

Dashboard Other Certification Notes

DynamoDB

- NoSQL serverless database
- Fully managed, highly available with replication across 3 AZs
- Scales to massive workloads, distributed
- Millions records per seconds, trillions of rows, 100TB of storage
- Fast and consistent in performance (low latency retrieval)
- Integrates with IAM for security and administration
- Enables event driven programming
- Low cost

Traditional databases	NoSQL
Traditional applications leverage RDBMS databases	Non-relational, **distributed** databases
SQL query language	Many different query languages, SQL can be one
Strong requirements about how the data should be modelled	All the data should be present in one row
Ability to do joins, aggregations and computations	Do not support join, can't perform aggregations such as "SUM"
Vertical scaling	**Horizontal scaling**

DynamoDB Basics

- Made of **tables**
- Each tables has a **primary key**
- Each table can have inf. number of rows(items)
- Each item can have **attributes** (columns, but the can be nested)
- Max size of an item is 400KB
- Data types supported:
  - Scalar: String, Number, Binary, Boolean, Null
  - Document Types: List, Map
  - Set Types: String Set, Number Set, Binary Set

DynamoDB Primary Keys

- Option I: partition key only (HASH), should be uniq
  - Partition key must be unique for each item
  - Partition key must be "diverse" so data will be distributed
- Option II: partition key + sort key, combination should be uniq
  - Data is grouped by partition key
  - Sort key = range key

DynamoDB Provisioned Throughput

- Table must have a provisioned read and write capacity
- **Read Capacity Units (RCU)**: throughput for reads
- **Write Capacity Units (WCU)**: throughput for writes
- Option to setup auto-scaling of throughput to meet demand
- Throughput can be exceeded temporarily using "burst credits"
- If no burst credits are available, we get a **ProvisionedThroughputException**
- We should use exponential back-off for retries in case get the exception above

- One write capacity unit represents one write per second for an item up to 1KB in size
- If the items are larger than 1KB, more WCU is consumed

- Examples:
  - 10 objects per second each of 2KB

    ```
    2 * 10 = 20 WCU
    ```

  - 6 objects per second, 4.5 each

    ```
    6 * 5 = 30 WCU (4.5 should be rounded to 5)
    ```

- **Eventually consistent reads**: if we read after a while it is possible to get unexpected data, there is a possibility that the data did not replicate to every node
- **Strongly consistent reads**: if we read data just after the write we get correct data
- By default: DynamoDB uses eventually consistent reads, but *GetItem*, *Query* and *Scan* can provide a **ConsistentRead** parameter to achieve strong consistency
- 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 the items are larger than 4KB, more RCU is consumed

- Examples:
  - 10 strongly consistent reads per seconds of 4KB each

    ```
    10 * 4KB / 4KB = 10 RCU
    ```

  - 16 eventual consistent reads per second of 12KB records

    ```
    (16 / 2) * (12 / 4) = 24 RCU
    ```

  - 10 strong consistent reads, 6KB each

    ```
    10 * 8KB / 4  = 20 RCU (we have to round up 6KB to 8 KB)
    ```

DynamoDB Internal Partitions

- Data is divided in partitions
- Partition keys are hashed in order to know in which partition will the data go
- To compute the number of partitions:
  - Capacity: `(TOTAL RCU / 3000) + (TOTAL WCU / 1000)`
  - Size: `(TOTAL SIZE / 10GB)`
  - Total number of partitions: `CEILING(MAX(Capacity, Size))`
- WCU and RCU are spread evenly between partitions!

DynamoDB Throttling

- If we exceed our RCU or WCU we get **ProvisionedThroughputExceededException**
- Reasons:
  - Hot keys
  - Hot partition
  - Very large items
- Solutions:
  - Exponential back-off
  - Distribute partition keys as much as possible
  - If RCU issue, use DynamoDB Accelerator (DAX)

DynamoDB - API

Writing Data

- **PutItem** - write data to DynamoDB (create data or full replace data)
  - Consumes WCU
- **UpdateItem** - update data in DynamoDB (partial update of attributes)
  - Possibility to use Atomic Counters and increase them
- **Conditional Writes**:
  - Accept a write/update only if some conditions are met
  - Helps with concurrent access
  - No additional cost

Deleting Data

- **DeleteItem**
  - Delete an individual row
  - Ability to perform conditional deletes
- **DeleteTable**

    - Deletes the whole table
    - Much quicker deletion than *DeleteItem*

Batch Writes

- **BatchWriteItem**
    - Up to 25 *PutItem* and/or *DeleteItem* in one call
    - Up to 16MB of data written
    - Up to 400KB or data per item
- Batching allows to save latency by reducing the number of API calls
- Operations are done in parallel by DynamoDB
- It is possible for a batch to fail, we can retry just the failed items

Reading Data

- **GetItem**
    - Read based on the primary key
    - Primary key = HASH or HASH-RANGE (partition key + sort key)
    - Eventually consistent by default, but has an option to have strong consistency
    - **ProjectionExpression** can be specified to include only certain attributes
- **BatchGetItem**
    - Up to 100 items
    - Up to 16MB data
    - Items are retrieved in parallel
    - Can be combined with projection expressions
- **Query**
    - PartitionKey value (*must be equals ("=") operator*)
    - SortKey value (operators: =, <, <= >, >=, Between, Begin) - optional
    - FilterExpression to further filter the data (this will happen on the client side)
    - Returns up to 1MB of data or number of items specified by the **Limit**
    - Able to do pagination
    - We can query a table, secondary index or global secondary index
    - Efficient way to query DynamoDB!
- **Scan**
    - Scans the entire table and then filter data
    - Returns up to 1MB ot data - we can use pagination to keep reading
    - Consumes lot of RCU
    - Limit impact using **Limit** / reduce the size of the result in order to reduce costs
    - For faster performance we can use parallel scan
        - Way more RCU!
    - Can we use a combination of *ProjectionExpression + FilterExpression* (no charge to RCU)
    - Inefficient way to read data from DynamoDB!

DynamoDB Indexes

Local Secondary Index (LSI)

- Alternate range key for the table, *local to the hash key*
- Up to 5 LSI/table
- The sort key consist of exactly one scalar attribute
- The attribute can be String, Number, Binary
- *LSI must be defined at table creation time!*

Global Secondary Index (GSI)

- Used to speed up queries on non-key attributes
- GSI = partition key + optional sort key
- This index can be seen as a new "table"
- We can project attributes on the new "table"
    - The partition key and the sort key of the original table are always projected (KEYS_ONLY)
    - We can specify extra attributes to be projected (INCLUDE)
    - We can use all attributes from main table (ALL)
- We must define a RCU/WCU for the index
- *GSI can be created and modified after origin table creation!*

Throttling

- DynamoDB indexes can cause throttling
    - GSI: **If writes are throttled in case of a GSI, the main table is throttled as well!** This can happen even if the WCU on the main table is just fine
    - LSI: uses the same WCU and RCU on the main table, can not throttle the main table

DynamoDB Concurrency Model

- Conditional update/delete: ensures the item hasn't change before altering it

- This feature makes DynamoDB an **optimistic locking / concurrent** database

DynamoDB Accelerator (DAX)

- It is a seamless caching mechanism for DynamoDB
- It can be activated without doing any code change on the application which uses the database
- Writes go through DAX to DynamoDB
- Micro second latency for cache reads & queries
- Solves one big problem: **Hot Key problem** (too many requests for the same key)
- By default: 5 min TTL for every item in the cache
- Up to 10 nodes in the cluster
- Multi AZ (3 nodes minimum recommended)
- Security (encryption at rest with KMS, IAM, CloudTrail)

DynamoDB Streams

- Changes in a DynamoDB table (create, update, delete) are pushed into a stream
- This stream can be read by Lambda or by EC2 instances
- They can react to this changelog by:
  - Doing analytics
  - Create derivate tables/view
  - Insert data in ElasticSearch
  - etc.
- Could implement cross region replication using Streams (nowadays it is a provided feature by DynamoDB)
- Stream have 24 hours of retention (we can not change this)
- We can choose what type of information ends up in stream:
  - KEYS_ONLY - only the key attributes of the modified items are pushed to the stream
  - NEW_IMAGE - the entire item is pushed to the stream after it was modified
  - OLD_IMAGE - the entire item is pushed to the stream before it was modified
  - NEW_AND_OLD_IMAGES - the entire item is pushed to the stream
- DynamoDB streams are made of shards, we don't have to provision them
- Records are not retroactively pushed to the stream after enabling it!

DynamoDB Streams and Lambda

- We need to define an event source mapping to read from the stream
- We need to ensure the Lambda has the appropriate methods
- The Lambda is invoked synchronously

DynamoDB TTL (Time to Live)

- TTL = item is automatically deleted after an expiry date/time
- TTL is provided at no extra cost / no additional WCU/RCU used
- TTL is background task operated by DynamoDB
- Helps reduce storage and manage the size of the table over time
- Helps adhere to regulatory norms
- TTL is enabled per row (we define a TTL columns and add a date there)
  - Can be named however we want
  - Should be a number (we should use time to epoch conversion for deletion timestamp)
- DynamoDB deletes expired items within 48 hours (it wont happen right after the item expired)
- Deleted items are deleted from the indexes as well (GSI/LSI)
- Deleted items could be recovered if DynamoDB Streams were enabled

DynamoDB CLI

- –projection-expression: attributes to retrieve
- –filter-expression: filter results
- General CLI pagination options:
  - Optimization:
    - –page-size: full dataset is still received but each API call will request less data (helps avoid timeouts)
  - Pagination:
    - –max-number: max number of results returned by the CLI. Returns *NextToken*
    - –starting-token: specify the last received *NextToken* to keep reading data from the next page
- CLI Examples:
  - Scan:

```
aws dynamodb scan --table-name blog-posts --projection-expression "post_id, content" --region
```


```
aws dynamodb scan --table-name blog-posts --projection-expression "post_id, content" --region
```

```
aws dynamodb scan --table-name blog-posts --projection-expression "post_id, content" --region
```

```
aws dynamodb scan --table-name blog-posts --projection-expression "post_id, content" --region
```

DynamoDB Transactions

- Ability to create/update/delete multiple rows in different tables at the same time
- “All or nothing” type operation
- Write modes: Standard, Transactional
- Read modes: Eventual Consistent, Strong Consistent, Transactional
- For Transactional it consumes 2x the WCU/RCU

DynamoDB for Store Session State

- It is common for DynamoDB to be used to store session state of a web application
- Same functionality can be achieved with ElastiCache as well, differences:
  - ElastiCache is in-memory, DynamoDB is serverless (automatic scaling)
  - Both are key/value store
- Same functionality can also be achieved with EFS disk storage, differences:
  - EFS must be attached to an EC2 instance, DynamoDB can be accessed via REST calls
  - EFS can not be used with Lambda
- ... EBS & Instance Store disk storage:
  - EBS & Instance store can only be used as local caching, they are not shared
- ... S3:
  - S3 has higher latency, not a great tool for session state storage

DynamoDB Write Sharding

- In case the partition key is not distributed enough, or we have limited number of partition keys, we can add a random suffix (or computed suffix) to the key to achieve higher distribution
- Example: voting system if we have 2 candidates

DynamoDB Write Types

- Concurrent writes:
  - Two users try to update the same record in the same time
  - The last update wins
- Conditional writes
  - A write can only happen if a specific condition is met
  - First write will succeed, second write will fail because the condition wont be met
  - Important: **data is not overridden**
- Atomic writes:
  - INCREASE BY or DECREASE BY writes
- Batch writes:
  - Write an update containing many items at the same time

DynamoDB - Large Object Patterns

- Max object size is 400KB
- Pattern:
  - Large object is uploaded to S3
  - Metadata is stored in DynamoDB
  - Reader clients reads the metadata which contains the object path and reads the object from the path
- Index objects in S3:
  - Write into S3 bucket -> trigger notification starts a Lambda function -> Lambda updates DynamoDB table

DynamoDB Operations

- Table cleanup:
  - Option 1: Scan + Delete => very slow, consumes RCU and WCU
  - Option 2: Drop table => fast, cheap, efficient
- Copy table:
  - Option 1: Use AWS DataPipeline (use EMR): takes the table, puts it into S3, puts it back into DynamoDB table (outdated)
  - Option 2: create a back-up, restore it into a new table
  - Option 3: Scan + Write

DynamoDB Security & Other Features

- Security:

    - VPC Endpoints to access DynamoDB without internet
    - Access fully controlled by IAM
    - Encryption at rest using KMS
    - Encryption in transit using SSL/TLS
- Backup/Restore:
    - Point in time restore la RDS
    - No performance impact
- Global Table:
    - Multi region, fully replicated, high performance
- Amazon DMS:
    - Used to migrate data from Mongo, Oracle, MySQL, S3 to DynamoDB
- Local DynamoDB for development