Amazon Dynamo DB

No Sql database

* They are non relational database and are distributed
* No database include emongo db and Dynamo DB
* They don’t support join query or limited support
* All the data that is needed for a query is present in one row
* They don’t perform aggregations such as sum avg
* They scale horizontally

Amazon dynamo db

* Fully manage highly available with replation across multiple azs
* Not a relational database
* Scales to massive workloads , distributed database
* Millions of request per seconds and trillions of rows and 100 tb of storage
* Low latency on retrieval
* Integrated with Iam for security and authorization
* Event driven programming with dynamo db streams
* Low cost

Dynamo DB basics

* Is made of tables
* Has primary key decided at cration
* Infinite rows
* Can be null and can be added over time
* Scalar types, document types, set type

Dynamo Db – Primary keys

* Option 1 : Partition key (Hash)
  + Partition key must be unique for each item
  + Diverse so that data is distributed
* Option 2 : Partition Key + Sorty Key (Hash + Range)
  + Must be unique for each item combination
  + Data is grouped by partition key

Table types

* Dynamo Db IA
* Dynamo DB standard

**Dynamo DB read and write capacity mode**

* Provisioned Mode
  + We can specity the number of reads and write per second
  + Plan capacity before hand
  + Pay for capacity units
* On demand mode
  + Automaticy scale up and down
  + No capacity planning needed
  + More expensive

You can switch between them in 24 hours

R/W Capacity Modes: Provisioned

* Must provisioned read and write capacity units
  + RCU
  + WCU
  + Through put can be exceeded temporarily using “Brust Capacity”
  + If brust capacity is exceded we get exception called
    - ProvisionedThrouhputExceededException
  + Then we should do exponential backoff retry

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

Strongly consistent read VS Eventually consistent read

* Eventually consistent read
  + If you want to read data immediately after, write it might not have been replicated to all servers so there is chance of
    - Data latency
    - Stale Data
* Strongly consistent read
  + What if you want to read data immediately and want it to be accurate
    - It will be expensive as it will consume more RCU

Dynamo DB – Read Capacity Units (RCU)

* 1 RCU represents one **Strongly consistent Read** per second

Or **Two Eventually consistent Read per second** for item up to 4 kb in size

* If the items are larger then 4kb then more RCU are consumed
* 2 eventually consistent read per second = 1 RCU
* Round up is mandatory

**Dynamo DB partitions Internal**

* Data is stored in partitions
* Partition keys go through a hashing algorithm to know which partition they go
* WCU’s and RCU’s are spread evenly across partitions evenly across.

Dynamo DB Throttling

* If we exceed provisioned RCU’s or WCU’s we get “**ProvisionedThroughputExceededException**”
* Reasons
  + Hot keys – one partition key is being read too many times
  + Hot partitions
  + Very large items
* Solutions:
  + Exponential back off when exception is encountered (already in SDK)
  + Distribute partition keys as much as possible
  + We can use DAX ( Dynamo DB accelerator)

R/W Capacity modes – On-Demand read and write

* Based on workload
* No capacity planning needed
* Expensive 2.5 times
* Use for unknown workloads

**Dynamo DB Wrting Data**

Put item

* Creates a new item or fully replace an old item (same primary key)
* Consumes WCUs

UpdateItem

* Edit an existing items attributes or adds a new item if it doesnot exist
* Can be used to implement Atomic Counters – a numeric attribute that’s unconditionally incremented

Conditional Writes

* Accept a write/update/delete only if conditions are met otherwise returns and error
* Help with concurrent access

**Dynamo DB Reading Data**

Get item

* Read based on Primary Key (hash or has +range)
* Eventually consistent or Strongly consistent

Query

* Return items based on key condition expression
  + Partition key value , Sort key value
* Filter Expression
  + Additional filtering based on query operation (before data return)
  + Non key attributes

Scan

* Used to scan entire table and filter out data , inefficient
* Returns up to 1 MB of data
* Consumes lot of RCU
* Limit impact using limit or reduce size of the result and pause
* For faster performance we need to use Parallel Scan

Delete Item

* Delete an individual item
* Ability to perform a conditional delete

Delete Table

* Delete a whole table and all its items
* Much quicker deletion than calling delete item

**Dynamo DB – Batch operations**

* Reduce the number of api calls
* They are applied in parallel
* Part of the batch can fail

Batch Write item

* Up to 25 put item or Delete item
* Up to 16 mb of data written , up to 400 kb
* Cant update items use update item

Batch get item

* Return items from multiple tables
* 100 items up to 16 mb
* Items are retrieved in parallel to minimize latency

**Dynamo DB = PartiQL**

* SQL compatible query language for Dynamo DB
* Allows you to select insert update and delete
* Run queries across multiple DynamoDb tables
* Run Querys from AWS management console

Dynamo DB **Local secondary Index**

* Alternative Sort Key for our table.
* Can be string number or binary
* Up to 5 local Secondary Index
* Must be defined at table creation time

Dynamo DB Globel secondary index

* A **GSI** allows you to **query data using different partition and sort keys** than the main table.
* It can be **created anytime** after the table is created — it’s **independent** of the base table's key schema.
* **GSI maintains its own read/write capacity** (unless on-demand) and can **scale independently** from the table.
* By default, reads on a GSI are **eventually consistent**, but you can request **strong consistency** optionally.

**🔹 DynamoDB Throttling**

* **What is Throttling?**  
  ➔ Throttling happens when your request rate **exceeds** the provisioned **read** or **write** capacity on a table or index.

**Dynamo DB Accelerator (DAX)**

* Fully managed in memory cache for Dynamo DB
* Microsecond latency for cached read and queries
* Solves Hot key problems
* Data lives for 5 minutes in DAX cluster

DAX vs elastic Chace

| **Feature** | **DAX** | **ElastiCache** |
| --- | --- | --- |
| **Scope** | Only for **DynamoDB** | For **any database or app** |
| **Use Case** | DynamoDB read acceleration | General-purpose caching |
| **Setup** | Attach directly to DynamoDB | Setup separately |
| **Consistency** | **Eventually consistent** | **Manual control** (Redis/Memcached) |
| **Best For** | DynamoDB-heavy applications | Web apps, APIs, SQL/NoSQL caching |

**Dynamo DB Streams**

* Stores List of all the modifications in the table
* They can be
  + Sent to kinesis Data Streams
  + Read AWS Lambda
  + Kenisis Client library applications

**Dynamo DB Time to LIVE**

* Automaticly deletes items after expiry date
* WE can set time to delete items automatically

**Dynamo DB – Large Objects Pattern**

* We can only store 400kb data only on dynamo db
* Instead we store large item in amazon s3 but store meta data in dynamo db
* When needed we will read meta data to find the large object image url

Dynamo DB – Index S3 object meta data

* Setting up a trigger in lambda function when ever object get pupolated in Amazon S3 and its meta data gets stored in dynamo db table.

Dynamo DB Security and other features

* VPC endpoints available to access Dynamo DB without using internet
* Fully controlled by IAM
* Encryption at rest

Amazon RDS (Relational Databse system)

* Amazon aurora, mysql, pgsql etc

Amazon Aurora

* My sql and postgre sql support
* 5x faster then mysql and 3x time fasater then pgsql
* 1/10 cheaper
* 128 tb per database, 15 read replicas
* Continuous backup to S3

**Locks Command**

* Database lock things to precent atomicity
* Tables and rows can be locked explicitly for concurrency control and data integrity
* Two types of lock
  + Shared lock:
    - Allows read , prevent writes. Can be held by multiple transactions. (FOR SHARE)
  + Exclusive lock : Prevents all read and write. Only one transaction hold exclusive lock. (FOR UPDATE)
* Syntax :
  + Lock tables employes WRITE;
  + Ulock tables;
* Redshift also use **LOCK**

Optimization in RDS

* Avoid full table scans
* Analyze table command to check health of table

Marida DB and MYSQL

* Keep tables under 16TB, ideally under 100 GB
* Have enough ram to hold indexes of actively used tables
* Try having less then 1000 tables

**AMAZON DOCUMENT DB (NO SQL Database)**

* version for mongo DB
* used to query index json data
* highly available across 3 AZ and managed fully
* grows increment 10 GB basis
* automatically scales to workloads

**AMAZON Memory DB for REDIS**

* in memory db service,
* use case: mobile apps, media streaming and online gaming
* Ultra-fast performance with 160 million requests per second

**AMAZON KEY SPACES (Apache Cassandra)**

* Apache Cassandra is an open-source NOSQL distributed database
* A manage database service for Cassandra
* Serverless, scalable, highly available, fully managed by AWS
* **Auto scale, tables replicated 3 times AZ**
* Uses Casandra Query language (CQL)

**AMAZON Neptune (Graph Database)**

* Fully managed Graph Database
* Highly available 3 AZ, 15 read replicas
* For hard and complex graph database
* Fraud Detection

**# AMAZON Neptune query languages**

* Supported : Gremlin, openCypher, SPARQL,

**Amazon Timestream**

* Time series database
* 100X faster with 1/10th cost
* Time series analytics functions