RELATIONAL DATABASE SERVICES (RDS)

What is DATA?

In simple words, data can be facts related to any object. For e.g: your age, job, house no, contact no., name, places are some data related to you.

What is DATABASE?

Database is a systematic collection of data. Databases supports storage and manipulation of data.

e.g : facebook, telecom companies, amazon.com

What is DBMS?

DBMS is a collection of programs which enable its users to access database, manipulate data, reporting/ representation of data.

Types of DBMS

1. Hierarchical
2. Network
3. Relational
4. Object

oriented 5.

Relational Database:

- A relational database is a data structure that allows you to link information from different tables of different types of data bucket.
- Tables are related to each other.
- All fields must be filled.
- Best suited for OLTP (online transaction processing)
- Relational DB: MySQL, Oracle, DBMS, IBM DB2
- A row of a table is also called records. It contains the specific information of each individual entry in the table.
- Each table has its own primary key.
- A schema (design of database) is used to strictly define tables, columns, indexes and relation between tables.
- Relational DB are usually used in enterprises application/scenario. Exception in MySQL which is used for web application.
- Common application for MySQL include php and java based web applications that requires a database storage backend. E.g: JOOMLA
- Cannot scale out horizontally.
- Virtually all relational DB uses SQL.

Non-Relational DB/NO-SQL DB:

- Non-relational databases store data without a structured mechanism to link data from different tables to one another.
- Required low cost hardware.
- Much faster performance (read/write) compared to relational DBs.
- Horizontal scaling is possible.
- Never provide tables with flat fixed column records. It means schema-free.
- Best suited for online analytical processing (OLAP).
- E.g: of NoSQL databases: MongoDB, Casssandra, DynamoDB, Postegre, Raven, Redis.

Types of No-SQL Databases:

1. Columnar Database (cassandra, HBase)
2. Document Databse (MongoDB, CouchDB, RavenDB)
3. Key Value Database (Redis, Riak, DynamoDB, Tokyo Cabinet)
4. Graph Based Database (Neo4J, FlockDB)

1. Columnar Database:

- A columnar database is a DBMS that stores data in columns instead of Rows.
- In a columnar DB all the column-1 values are physically together followed by all the column-2 values.

- In a row oriented DBMS the data would be stored like this: (1, bob, 30, 8000: 2, arun, 26, 4000: 3, vian, 39, 2000 ;)

- In a column based DBMS the database would be stored like this: (1, 2, 3: bob, arun, vian; 30, 26, 39; 8000, 4000, 2000 ;)

- Benefit is that because a column based DBMS is self-indexing, it uses less disk space that a RDBMS containing the same data. It easily perform operation like min, max, average.

2. Document Database:

- Document DB make it easier for developer to store and querying data in a DB by using the same document model format they use in their application code.
- Document DBs are efficient for storing catalogue.
- Store semi-structure data as document typically in JSON or XML format. (example)
- A document database is a great choice for contain management application such as blogs and video platform.

3. **Key-Value Database:**

   - A key-value DB is a simple DB that uses an associative array (like dictionary) as a fundamental model where each key is associated with one and only one value in a collection.
   - It allows horizontal scaling.
   - Used cases: shopping cart, and session store in app like fb and twitter.
   - They improve application performance by storing critical pieces of data in memory for low latency access.
   - Amazon elasticache as an in-memory key-value stores.

4. **Graph Based Database:**

   - A graph DB is basically a collection of nodes and edges.
   - Each node represent an entity and each edges represent a connection or relationship between two nodes.

   - In an AWS fully managed relational DB engines service where AWS is responsible for:
     - Security and patching.
     - Automated backup.
     - Software updates for DB engine.
     - If selected multi AZ with synchronous replication between the active and stand by DB instances.
     - Automatic failover if multi AZ option was selected.
     - By default, every DB has weekly maintenance window. (max 35 days.)

     Settings managed by the users:

     - Managing DB settings.
     - Creating relational database schema.
     - Database performance tuning.

Relational Database Engine Options:

1. MS SQL Sever
2. My SQL: supports 64TB of DB
3. Oracle
4. AWS Aurora: high throughput

5. Postgre SQL: highly reliable and stable
6. Maria DB: MySQL compatible, 64TB DB

There are two Licensing Options:

1. BYOL (Bring Your Own License)
2. License from AWS on hourly basis

RDS Limits:

- Up to 40DB instances per account.
- 10 of this 40 can be Oracle or MS-SQL server under license included

model. Or

- Under BYOL model, all 40 can be any DB engine you need.

RDS Instance Storage:

- Amazon RDS use EBS volumes (not instance store) for DB and logs storage.

1. General Purpose: use for DB workloads with moderate I/O requirement. Limits: min: 20 GB
   Max: 16384 GB

2. Provisional IOPS RDS Storage: use for high performance OLTP workloads. Limits: min: 100GB
   Max: 16384GB

Templates available in RDS:

a. Production
b. Dev/Test
c. Free-Tier

DB Instance Size:

a. Standard class
b. Memory-Optimized class
c. Burstable class

What is Multi-AZ in RDS:

- You can select multi AZ option during RDS DB instance launch.
- RDS service creates a standby instances in a different AZ in the same region and configure "synchronous replication" between the primary and standby.
- You cannot read/write to the standby RDS DB instances.
- You cannot select which AZ in the region will be chosen to create the standby DB instance.
- You can however view, which AZ is selected after the standby is created.
- Depending on the instance class it may take 1 to few minutes to failover to the standby instance.
- AWS recommends the use of provisioned IOPS instances for multi-AZ RDS instance.

When Multi-AZ RDS Failover Triggers:

- In case of failure of primary DB instance failure.
- In case of AZ failure.
- Loss of network connectivity to primary DB.
- Loss of primary EC2 instance failure.
- EBS failure of primary DB instance.
- The primary DB instance is changed.
- Patching the O.S of the DB instance.
- Manual failover. (in case of rebooting.)

Multi-AZ RDS Failover Consequences:

- During failover the CNAME of the RDS DB instance is updated to map to the standby IP address.
- It is recommended to use the end point to reference your DB instances and not its IP address.
- The CNAME doesn't change because the RDS endpoint doesn't change.
- RDS end point doesn't change by selecting multi-AZ option, however the primary and standby instances will have different IP addresses, as they are in different AZ.
- It is always recommended that you do not use the IP address to point RDS instances, always use endpoint. By using endpoint there will be no change whenever a failover happens.

When we do manual failover?

- In case of rebooting.
- This is by selecting the "reboot with failover" reboot options on the primary RDS DB instances.

- A DB instance reboot is required for changes to take effect when you change the DB parameter group on when you change a static DB parameter.

- Whenever failover occurs AWS RDS sends SNS notification.
- You can use API calls to find out the RDS events occurred in the last 14 days.
- Even you can use CLI to view last 14 days events.
- Using AWS console you can only last one day events
- In case of OS patching, system upgrades and DB scaling, these things happens on standby first then on primary to avoid outage.
- In multi-AZ, snapshots and automated backups are done on standby instance to avoid I/O suspension on primary.

RDS Multi-AZ Deployment Maintenance:

- Firstly perform maintenance of standby.
- Now convert standby into primary so that maintenance can be done on primary.(currently)
- You can manually upgrade a DB instance to a supported DB engine version from AWS console as follows:- RDS->DB instance->modify DB->set DB engine version.
- By default change will take effect during the next maintenance window.
- Or you can force a immediate upgrade if you want.
- In multi-AZ version upgrade will be conducted by both primary and standby at the same time which will cause an outage.
- Do it during maintenance window.

There are two methods to backup and restore your RDS DB instances:

1. AWS RDS automated backup
2. User initiated manual backup

- Either you can take backup of entire DB instance or just the DB.
- You can create a restore volume snapshots of your entire DB instances.
- Automated backups by AWS, backup your DB data to multiple AZ to provide for data durability.
- Select-automated backup in AWS console.
- Stored in Amazon S3.
- Multi-AZ automated backups will be taken from the standby instance.
- The DB instance must be in “ACTIVE” state for automated backup.
- RDS automatically backups the DB instances daily by creating a storage volume snapshots of your DB instance (fully daily snapshots) including the DB transaction logs.
- You can decide when you would like to take backup (window)
- No additional charge for RDS backing up your DB instance.
- For multi-AZ deployment, backups are taken from the standby DB instance (true for Maria DB, MySQL, Oracle, Postgre SQL).
- Automated backups are deleted when you delete your RDS DB instance.

- An outage occurs if you change the backup retention period from zero to non-zero value or the other way around.
- Retention period of automate backup is 7 days (by default) via AWS console.
- AWS Aurora is an exception. Its default is 1 day.
- Via CLI or API 1 day by default.
- You can increase it up to 35 days.
- If you don't want backup, put zero "0" in the retention period.
- In case of manual snapshot, point-in-time recovery is not possible.
- Manual snapshot is also stored in S3.
- They are not deleted automatically, if you delete RDS instance.
- Take a final snapshot before deleting your RDS DB instance.
- You can share manual snapshot directly with other AWS Account.
- When you restore a DB instance only the default DB parameters and security groups are associated with the restored instance.
- You cannot restore a DB snapshot into an existing DB instance rather it has to create a new DB instance it has new endpoint.
- Restoring from the backup or a DB snapshot changes the RDS instance endpoint.
- At the time of restoring, you can change the storage type (general purpose or provisioned.)
- You cannot encrypt an existing unencrypted DB instance.
- To do that you need to: create a new encrypted instance and migrate your data to it (from unencrypted to encrypted) or you can restore from a backup/snapshot into anew encrypted RDS instance.
- RDS supports encryption-at-rest for all DB engines using KMS.
- What actually encrypted when data-at-rest:
    a. All its snapshots.
    b. Backups of DB (S3 storage.)
    c. Data on EBS volume.
    d. Read replica created from the snapshots.

Some points related to RDS Billings:

- No upfront cost.
- You have to pay only for:
    a. DB instance hours (partial hour charged as full hours)
    b. Storage GB/month.
    c. Internet data transfer.
    d. Backup storage (i.e S3)
    This is increases by increasing DB backup's retention period. Also charged for:
        a. Multi-AZ DB hours.
        b. Provisioned stage (multi-AZ)
        c. Double write I/O
        d. You are nor charged for DB data transfer during replication from primary to standby.

DYNAMO DB

Database Types:

Generally there are three types of data are available such as:

1. Unstructured Data
2. Semi-structured Data
3. Structured Data

1. Unstructured Data:

- It is the information that either doesn't have a predefined data model or it is not organized in a predefined manner.
- Unstructured information is text-heavy but may contains data such as dates, numbers and facts as well as examples include email messages, word processing documents, videos, photos, audio files, presentations webpages.

2. Semi-structured Data:

- Semi-structured data is information that doesn't reside in a relational database but that does have some organizational properties that make it easier to analyze.
- E.g: XML, JSON

3. Structured Data:

- It refers to information with a high degree of organization, such that inclusion in a relational database is seamless and readily searchable by simple, straight forward search engine algorithms or other search operation.
- All data which can be stored in a database SQL in table with rows and columns. They have relational key and can be easily mapped into pre-defined fields.

DynamoDB Table:

- A table is a collection of data items.
- Like all other DB, DynamoDB stores data in tables.

Items:

- Each table contains multiple items.
- An item is a group of attributes that is uniquely identifiable among all of the other items.
- Am item consists of a primary or composite key and a flexible number of attributes.

- Items in DynamoDB are similar into rows, records in other DB.

Attributes:

- Each item is composed one or more attributes.
- An attribute consists of the attribute name and a value or a set of values.
- An attribute is a fundamental data element, something that does not need to be broken down any further.

Note: aggregate size of an item cannot exceed 400kb including key and all attributes.

- DynamoDB allows low latency read/write access to items ranging from 1 byte to 400kb.
- 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 can specify the primary key when you create the table.
- Each item in the table has a unique identifier. Or primary key, that distinguished the item from all of the others in the table.
- The primary key in the only required attributes for items in a table.
- DynamoDB tables are schema less, which means that neither the attributes not their data types need to be defined before head.
- Each item can have its own distinct attributes.

DynamoDB-Read Capacity Unit:

- One read capacity unit represents one strongly consistent read per second, or two eventually consistent reads per seconds for an item up to 4kb in size.
- It you need to read an item that is larger than 4kb, 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.

DynamoDB-Write Capacity Unit:

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

DynamoDB- Pricing:

Reads are cheaper than writes when using DynamoDB. We are only pay for:

- Each table's provisioned read/write throughput (hourly rates).
- You are charged for provisioned throughput regardless whether you use it or not.
- Indexed data storage.
- Internet data transfer (if crosses a region).
- Free tier per account (access all tables) of 25 read capacity unit and 25 write capacity units per month.
- DynamoDB can do 10,000 writes capacity units or 10,000 read capacity units per second per table.

DynamoDB Limits:

- 256 tables per account per region.
- No limits on the size of any table.

DynamoDB Streams and Triggers

DynamoDB stream is a time ordered list of changes to items in a DynamoDB table. A stream is a 24-hour rolling window of the changes. It uses Kinesis streams on the backend.

This is enabled on a per table basis. This record

- Inserts
- Updates
- Deletes

Different view types influence what is in the stream.

There are four view types that it can be configured with:

- KEYS_ONLY : only shows the item that was modified
- NEW_IMAGE : shows the final state for that item
- OLD_IMAGE : shows the initial state before the change
- NEW_AND_OLD_IMAGES : shows both before and after the change

Pre or post change state might be empty if you use **insert** or **delete**

Trigger Concepts

Allow for actions to take place in the event of a change in data

Item change generates an event that contains the data which was changed. The specifics depend on the view type. The action is taken using that data. This will combine the capabilities of stream and lambda. Lambda will complete some compute based on this trigger.

This is great for reporting and analytics in the event of changes such as stock levels or data aggregation. Good for data aggregation for stock or voting apps. This can provide messages or notifications and eliminates the need to poll databases.

DynamoDB Local (LSI) and Global (GSI) Secondary Indexes

- Index is used for improving data retrieval in DynamoDB.
- Query can only work on 1 PK(Partition key) value at a time and optionally a single or range of SK(Sort Key) values.
- Indexes are a way to provide an alternative view on table data.
- You can choose which attributes are projected to the table.

Local Secondary Indexes (LSI)

- Choose alternative sort key with the same partition key on base table data.
    - If item does not have sort key it will not show on the table.
- **These must be created with a base table in the beginning**.
    - This cannot be added later.
- Maximum of 5 LSIs per base table.
- Uses the same partition key, but different sort key.
- Shares the RCU and WCU with the table.
- It makes a smaller table and makes **scan** operates easier.
- In regard to Attributes, you can use:
    - ALL
    - KEYS_ONLY
    - INCLUDE

Global Secondary Index (GSI)

- Can be created at any time and much more flexible.
- There is a default limit of 20 GSIs for each table.
- Allows for alternative PK and SK.
- GSI will have their own RCU and WCU allocations.
- You can then choose which attributes are included in this table.
- GSIs are **always** eventually consistent. Replication between base and GSI is Async.

DynamoDB Global Tables

- Global tables provide multi-master cross-region replication.
    - All tables are the same.
- Tables are created in multiple AWS regions. In one of the tables, you configure the links between all the tables.
- DynamoDB will enable replication between all of the tables.
    - Tables become table replicas.
- Between the tables, **last writer wins** in conflict resolution.
    - DynamoDB will pick the most recent write and replicate that.
- Reads and Writes can occur to any region and are replicated within a second.
- **Strongly Consistent Reads** only in the same region as writes.
    - Application should allow for eventual consistency where data may be stale.
    - Replication is generally sub-second and depends on the region load.

- Provides Global HA and disaster recovery or business continuity easily.

DynamoDB Accelerator (DAX)

This is an in-memory cache for Dynamo.

Traditional Cache: The application needs to access some data and checks the cache. If the cache doesn't have the data, this is known as a cache miss. The application then loads directly from the database. It then updates the cache with the new data. Subsequent queries will load data from the cache as a cache hit and it will be faster

DAX: The application instance has DAX SDK added on. DAX and dynamoDB are one in the same. Application uses DAX SDK and makes a single call for the data which is returned by DAX. If DAX has the data, then the data is returned directly. If not it will talk to Dynamo and get the data. It will then cache it for future use. The benefit of this system is there is only one set of API calls using one SKD. It is tightly integrated and much less admin overhead.

DAX Architecture

This runs from within a VPC and is designed to be deployed to multiple AZs in that VPC. Must be deployed across AZs to ensure it is highly available.

DAX is a cluster service where nodes are placed into different AZs. There is a **primary node** which is the read and write note. This replicates out to other nodes which are **replica nodes** and function as read replicas. With this architecture, we have an EC2 instance running an application and the DAX SDK. This will communicate with the cluster. On the other side, the cluster communicates with DynamoDB.

DAX maintains two different caches. First is the **item cache** and this caches individual items which are retrieved via the **GetItem** or **BatchGetItem** operation. These operate on single items and must specify the items partition or sort key.

There is a **query cache** which holds data and the parameters used for the original query or scan. Whole query or scan operations can be rerun and return the same cached data.

Every DAX cluster has an endpoint which will load balance across the cluster. If data is retrieved from DAX directly, then it's called a cache hit and the results can be returned in microseconds.

Any cache misses, so when DAX has to consult DynamoDB, these are generally returned in single digit milliseconds. Now in writing data to DynamoDB, DAX can use write-through caching, so that data is written into DAX at the same time as being written into the database.

If a cache miss occurs while reading, the data is also written to the primary node of the cluster and the data is retrieved. And then it's replicated from the primary node to the replica nodes.

When writing data to DAX, it can use write-through. Data is written to the database, then written to DAX.

DAX Considerations

- **Primary node which writes and Replicas which support read** operations.
- Nodes are HA, if the primary node fails there will be an election and secondary nodes will be made primary.
- **In-memory cache allows for much faster read operations** and significantly reduced costs. If you are performing the same set of read operations on the same set of data repeatedly, you can achieve performance improvements by implementing DAX and caching those results.
- With DAX you can **scale up or scale out.**
- DAX **supports write-through**. If you write data to DynamoDB, you can use the DAX SDK. DAX will handle that data being committed to DynamoDB and also storing that data inside the cache.
- DAX is not a public service and is **deployed within a VPC**. Anything that uses that data many times will benefit from DAX.
- Any questions which talk about caching with DynamoDB, assume it is DAX.

Amazon Athena

- You can take data stored in S3 and perform Ad-hoc queries on data. Pay only for the data consumed.
- Start off with structured, semi-structured and even unstructured data that is stored in its raw form on S3.
- Athena uses **schema-on-read**, the original data is never changed and remains on S3 in its original form.
- The schema which you define in advance, modifies data in flight when its read.
- Normally with databases, you need to make a table and then load the data in.
- With Athena you create a schema and load data on this schema on the fly in a relational style way without changing the data.
- The output of a query can be sent to other services and can be performed in an event driven fully serverless way.

Athena Explained

The source data is stored on S3 and Athena can read from this data. In Athena you are defining a way to get the original data and defining how it should show up for what you want to see.

Tables are defined in advance in a data catalogue and data is projected through when read. It allows SQL-like queries on data without transforming the data itself.

This can be saved in the console or fed to other visualization tools.

You can optimize the original data set to reduce the amount of space uses for the data and reduce the costs for querying that data.