## Introduction

Over the years, we have seen a shift from the concept of traditional IT, where a relational database was used for everything, to the public cloud. As the rate of generation and the demand for data continues to grow at an exponential rate, many businesses now face the challenge of not knowing how to cope with the rapid changes while having so much data at their disposal. Users that once dealt with Gigabytes (GBs) of data find themselves now having to learn how to work with Petabytes (PBs). This goes to show that in order for businesses to continue to excel, they need to remain proactive and evolve along with the ever-changing world of technology.

In today's world, it is common to use a broad range of databases to cater to each aspect of a business.

## Amazon Database Types

AWS consists of multiple fully managed database services, each built for its unique purpose (see the table below). As data continues to become an organization's most valuable asset, the amount of data also increases exponentially, creating newer and bigger challenges for businesses today. This means that we can no longer use a one-size-fits-all solution.

New data demands from organizations:

- Ability to store PBs of data
- Sub-millisecond latency requirements for global users
- Management of millions of requests per second
- Online leaderboards in real-time driving analytics

| AWS Service | Use Case | Database Type |
|---|---|---|
| Amazon Aurora, Amazon RDS, Amazon Redshift | Traditional applications, ERP, CRM, e-commerce | Relational |
| Amazon DynamoDB | High-traffic web apps, e-commerce systems, gaming applications | Key-Value |
| Amazon Elasticache for Memcached, Amazon Elasticache for Redis | Caching, session management, gaming leaderboards, geospatial applications | In-Memory |
| Amazon DocumentDB | Content management, catalogs, user profiles | Document |
| Amazon Neptune | Fraud detection, social networking, recommendation engines | Graph |
| Amazon QLDB | Systems of record, supply chain, registrations, banking transactions | Ledger |

## How to Choose the Right Database

Customers often ask me, "What database is right for my business?" This is a tricky question to answer on the spot, because the reality is that there is no one database in the cloud that does it all. We usually see a mix of databases across an organization.

When choosing a database, we have to evaluate several aspects, such as users, security, and data. Here are some starter questions to kick-start the evaluation process:

- What kind of data are we storing? Structured, semi-structured, or unstructured?
- What is the amount of data to store? What is the object size?
- What is the duration of storage?
- Is it read/write heavy or balanced?
- What are the latency requirements?
- What type of schema? Strong or flexible?
- What is the RTO/RPO for DR?
- How many concurrent users?

## Data

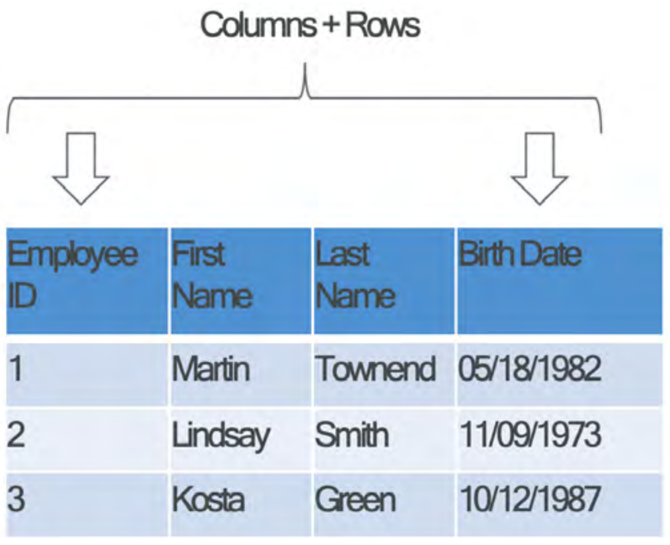
Data formats exist in many different structures – these can range from text data to application data, database data, and many more. Given all these different forms of data, it is impossible for an organization to organize all its data into one place.

There are usually multiple different teams across any organization, and each of these teams has its own data utilization needs. For instance, there are Data Scientists that want to run and build algorithms on the data, BI that want to run reporting, and Analysts that want to do ad-hoc analysis. Apart from this, you also have applications that are powered based on insights drawn from this data, making it crucial for the data to be available in real time.

As the volume of data and the number of users continue to grow, it becomes imperative to choose a solution that is reliable, secure, and supports scalability so that your organization can support a wide variety of uses cases across a wide variety of formats.

Let's take a closer look at the different data types.

### Structured Data


Columns + Rows

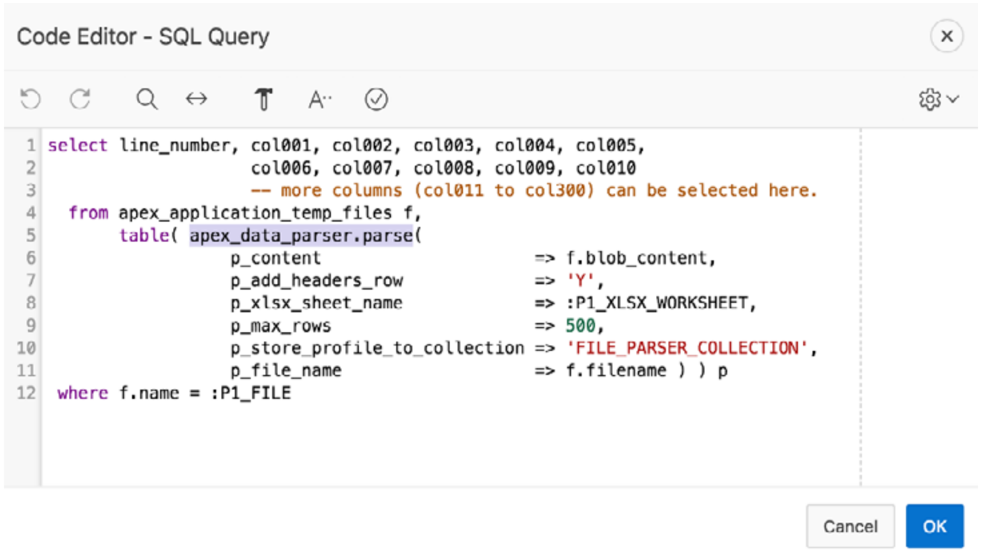
| Employee ID | First Name | Last Name | Birth Date |
|---|---|---|---|
| 1 | Martin | Townend | 05/18/1982 |
| 2 | Lindsay | Smith | 11/09/1973 |
| 3 | Kosta | Green | 10/12/1987 |

Structured Data is data created using a defined (fixed) schema that is typically organized in a Tabular format. The schema is what interprets the data of attribute names and data types.

A good example of Structured Data is a relational database, where tables are linked using unique IDs and a query language. For example, SQL is used to query and interact with the data.

Structured Data is data that is stored in tables, rows, and columns. It is commonly suited for complex queries and analytics.

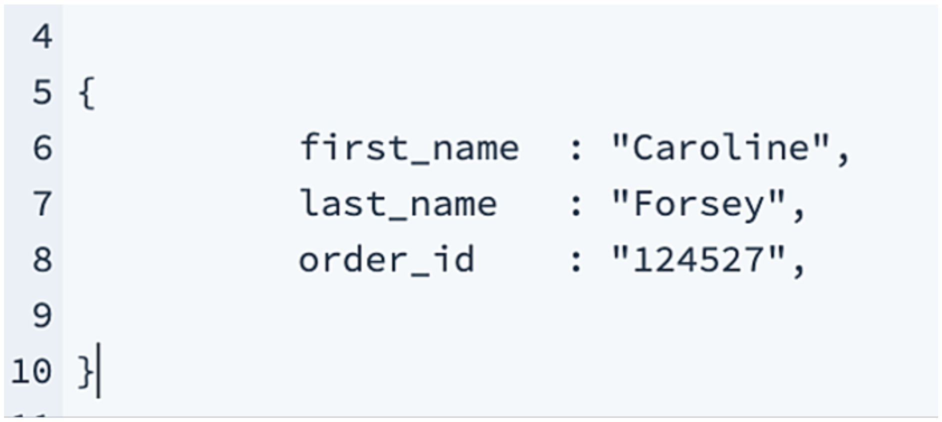
## Unstructured Data



```
select line_number, col001, col002, col003, col004, col005,
                    col006, col007, col008, col009, col010
                    -- more columns (col011 to col300) can be selected here.
  from apex_application_temp_files f,
       table( apex_data_parser.parse(
                  p_content                     => f.blob_content,
                  p_add_headers_row             => 'Y',
                  p_xlsx_sheet_name             => :P1_XLSX_WORKSHEET,
                  p_max_rows                    => 500,
                  p_store_profile_to_collection => 'FILE_PARSER_COLLECTION',
                  p_file_name                   => f.filename ) ) p
 where f.name = :P1_FILE
```

Unlike Structured Data, Unstructured Data has no rows, columns, or tables, and does not incorporate into a spreadsheet. It has no defined schema and is characterized by little to no rigid, formal structural properties. This is unorganized data, and typically of non-relational databases, file systems, data stores, or data lakes like S3. A typical example of Unstructured Data is a PDF data format.

## Semi-Structured Data



```

{
            first_name   : "Caroline",
            last_name    : "Forsey",
            order_id     : "124527",

}
```

Semi-Structured Data is a cross between Structured and Unstructured Data. While it is partially structured, the data is still too unstructured to be relational data. With Semi-Structured Data, the data is organized, but it is not constrained by a fixed schema, typically of a non-relational database.

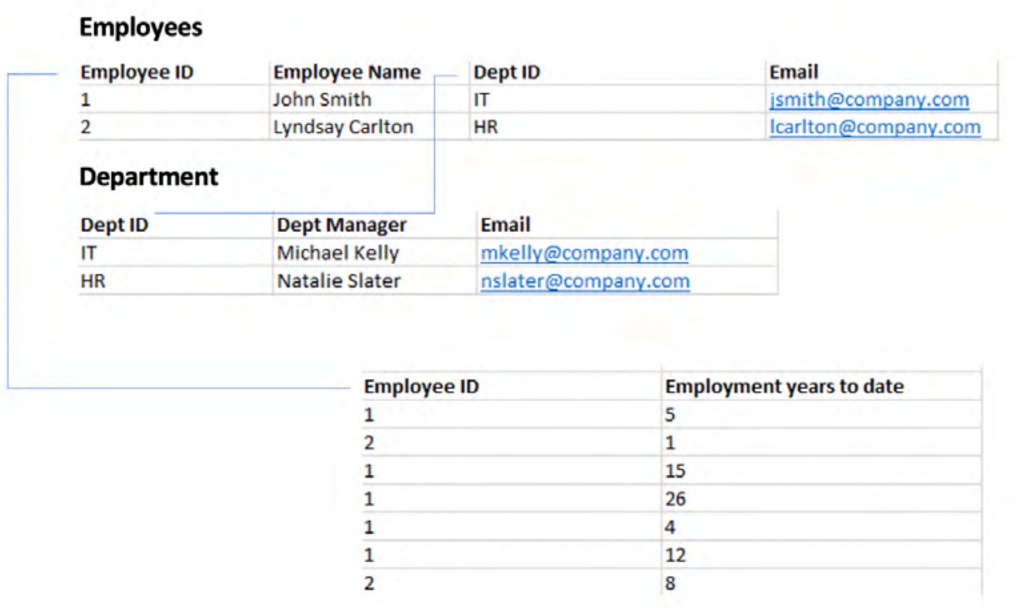
Semi-Structured Data is a type of data that has some consistent and definite characteristics. It does not confine into a rigid structure such as that needed for relational databases. Organizational properties like metadata or semantics tags are used with Semi-Structured Data to make it more manageable, but it still contains some variability and inconsistency. This data is non-relational data, NoSQL data, and data that is in CSV, XML, or JSON format. It is also well-suited for big data and low-latency applications.

| Element | Structured Data | Unstructured Data | Semi-Structured Data |
|---|---|---|---|
| **Foundation** | Schema, based on Relational Database tables | Based on simple character and binary data | Based on XML/RDF |
| **Transaction Management** | Management and concurrency of data is present and therefore mostly preferred in multitasking process | No transaction management and concurrency are present | (Resource Description Framework) |
| **Versioning** | It supports Relational Databases, so versioning is done over tuples, rows, and table | Only as a whole, data has no support of database at all | Transaction is not default; it becomes adopted from DBMS but data concurrency is not present. |
| **Flexibility** | It is schema-dependent and less flexible than Unstructured and Semi-Structured, and less scalable | As there is no dependency on any database, Unstructured Data is more flexible and scalable as compared to Structured and Semi-Structured Data | Only over tuples or graph is possible |
| **Performance** | With Structured Data, structured query allows complex joining that has a significant performance increase effect | Only textual query is possible, so performance is lower than both Structured and Semi-Structured Data | More flexible than Structured Data but less flexible than Unstructured Data |

## Relational Databases

Relational Databases consist of a predefined schema. Essentially, they are made for Structured Data, and with using this kind of data, some typical use cases are OLTP (Transactional) and OLAP (Analytical) applications. Some examples are MySQL, Microsoft SQL Server, PostgreSQL, MariaDB, and Oracle. Relational Databases are characterized by multiple tables interconnected by foreign key relationships and use indexes.

The "Employees" table below includes a Foreign key for "Dept ID," which points to the Primary key in the "Department" table of "Dep ID." This is how Relational tables are interconnected and linked to each other.



**Employees**

| Employee ID | Employee Name | Dept ID | Email |
|---|---|---|---|
| 1 | John Smith | IT | jsmith@company.com |
| 2 | Lyndsay Carlton | HR | lcarlton@company.com |

**Department**

| Dept ID | Dept Manager | Email |
|---|---|---|
| IT | Michael Kelly | mkelly@company.com |
| HR | Natalie Slater | nslater@company.com |

| Employee ID | Employment years to date |
|---|---|
| 1 | 5 |
| 2 | 1 |
| 1 | 15 |
| 1 | 26 |
| 1 | 4 |
| 1 | 12 |
| 2 | 8 |

The AWS services that support Relational Data are:

- Amazon RDS and Aurora for OLTP (Transactional)
- Amazon Redshift for OLAP (Analytical)

## ACID Compliant

Relational Databases are also ACID compliant—but what is ACID?

**ACID:**

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

| ACID | |
|---|---|
| **Atomicity** | Transactions; an item must be in your cart to be able to pay for it |
| **Consistency** | Must follow data validation rules, schema consistency |
| **Isolation** | Refers to the ability to concurrently process multiple transactions in a way that one does not affect another |
| **Durability** | In databases that possess durability, data is saved once a transaction is completed, even if a power outage or system failure occurs |

**Note: ACID compliance only refers to the compliance enforced by the database. However, ACID behavior can also be enforced by your application if the database does not enforce ACID.**

## Non-Relational Databases

Non-Relational Databases are suitable for Semi-Structured and Unstructured Data. These databases are called NoSQL.

Non-Relational Data is stored in a denormalized form, so there may be duplicate or redundant data stored in a single table, unlike Relational tables, where the data is interconnected with a foreign key relationship.

Non-Relational Databases are essentially used for big data applications and are best suited for low-latency applications given that they use very simple data formats like XML, for example.

**Note: Non-Relational Databases do not inherently comply with ACID, and are not suited for OLAP (Analytical Applications/Workloads).**

### BASE Compliance

Non-Relational Databases do support BASE compliance—but what is BASE?

**BASE:**

- **B**asically **A**vailable
- **S**oft-state
- **E**ventually-consistent

| BASE | |
|---|---|
| **Basically Available** | Basic R/W operations are always available |
| **Soft-state** | No consistency guarantees, data state may possibly change |
| **Eventually consistent** | Data will eventually become consistent over time |

## Relational and Non-Relational Comparison

Here we will look at the comparisons between both Relational and Non-Relational Databases in the table below.

| Relational (SQL) | Non-Relational (NoSQL) |
|---|---|
| Data stored in multiple tables interconnected through foreign key relationships | Data stored as collections or key-value pairs |
| ACID Compliance | BASE Compliance |
| Fixed Schema (Structured Data) | Flexible Schema (Semi-Structured + Unstructured Data) |
| Vertical Scaling | Horizontal Scaling |
| Uses SQL | Uses Object-based APIs |
| Best suited for OLTP and OLAP | Best suited for OLTP (web/mobile apps) |

## Amazon RDS

Amazon's RDS is a Relational Database Service (server) that allows for the creation of managed database instances using SQL (Structured Query Language).

RDS supports the following Engines: PostgreSQL, MySQL, MariaDB, Oracle, Microsoft SQL Server, and Aurora.

| Engine | Max Storage | IOPS |
|---|---|---|
| PostgreSQL | 100 GiB - 64 TiB | 1,000 - 80,000 |
| MySQL | 100 GiB - 64 TiB | 1,000 - 80,000 |
| MariaDB | 100 GiB  - 64 TiB | 1,000 - 80,000 |
| Oracle | 100 GiB  - 64 TiB | 1,000 - 80,000 |
| Microsoft SQL Server (Enterprise and Standard) | 20 GiB -16 TiB | 1000 – 32,000 or 64,000 for Nitro-based m5 instance types |
| Microsoft SQL Server (Web and Express) | 100 GiB – 16 TiB | 1000 – 32,000 or 64,000 for Nitro-based m5 instance types |

RDS is a managed database service in which AWS takes care of the underlying hardware of the server and the operating system (OS), as well as patching the OS and the database, which frees up more time for database admins. AWS also manages backups, automatic failure detection, and recovery.

There can be a maximum of 40 Amazon RDS instances.

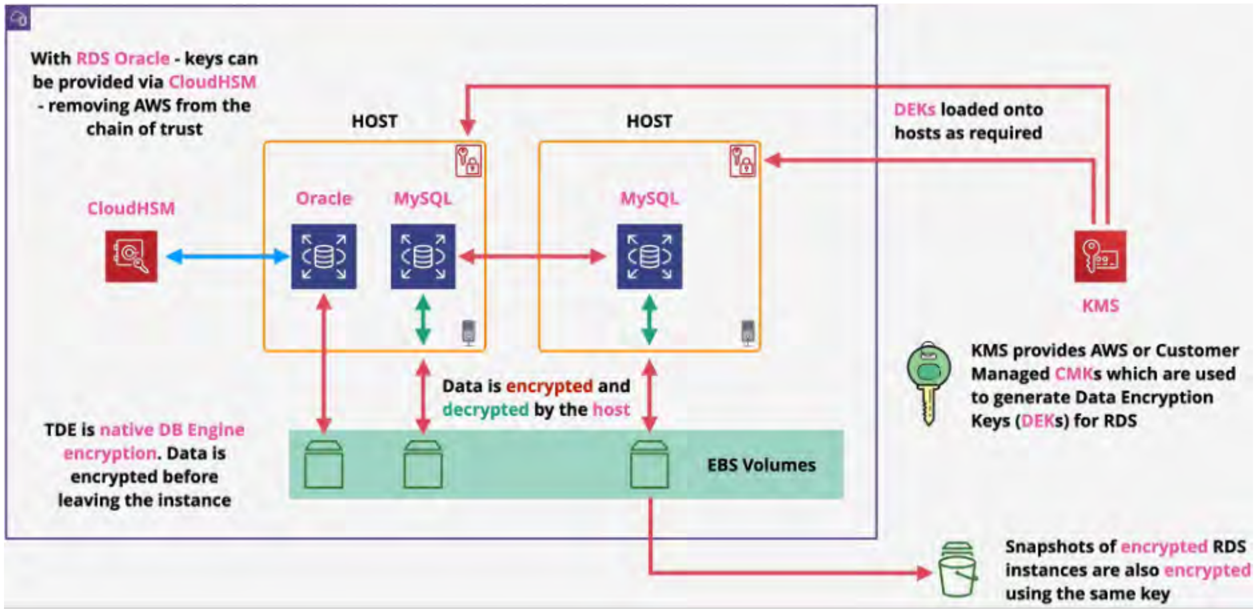
### RDS Security

When we talk about security in the cloud in AWS, we always have to refer to the Shared Responsibility Model:

- **Security of the Cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides the customer with services that they can use securely. Third-party auditors regularly test and verify the effectiveness of AWS security as part of the AWS compliance programs.
- **Security in the Cloud** – The customer's responsibility is determined by the AWS service that they use. The customer is also responsible for other factors, including the sensitivity of their data, their organization's requirements, and applicable laws and regulations.

**The security best practices for RDS are:**

- SSL/TLS (in Transit) is available for RDS, Supports EBS Volume Encryption –KMS.
- RDS MSSQL and Oracle support TDE – Transparent Data Encryption which is handled by the DB engine.
- Run the database instance in a virtual private cloud (VPC), usually a private subnet, control network access using security groups. These can control access at DB, EC2, and VPC level.
- Use Identity and Access Management (IAM) to secure access to the RDS database resources by assigning permissions that determine which users are allowed to do different tasks in the database.
- Use Amazon RDS encryption to secure data at-rest and in-transit.


With RDS Oracle - keys can be provided via CloudHSM - removing AWS from the chain of trust
HOST
HOST
DEKs loaded onto hosts as required
CloudHSM
Oracle
MySQL
MySQL
KMS
KMS provides AWS or Customer Managed CMKs which are used to generate Data Encryption Keys (DEKs) for RDS
Data is encrypted and decrypted by the host
TDE is native DB Engine encryption. Data is encrypted before leaving the instance
EBS Volumes
Snapshots of encrypted RDS instances are also encrypted using the same key

**Note: RDS IAM Authentication Authorization is controlled by the database engine. Permissions are assigned to the local DB user. IAM is not used to authorize, only for authentication.**

**RDS DB instance in a VPC best practices:**

- A VPC must have minimum of two subnets. These subnets should be in two different Availability zones in the region where you want to deploy the database instance.
- A database subnet group must be created in the VPC
- A VPC security must be created that allows access to the database instance.

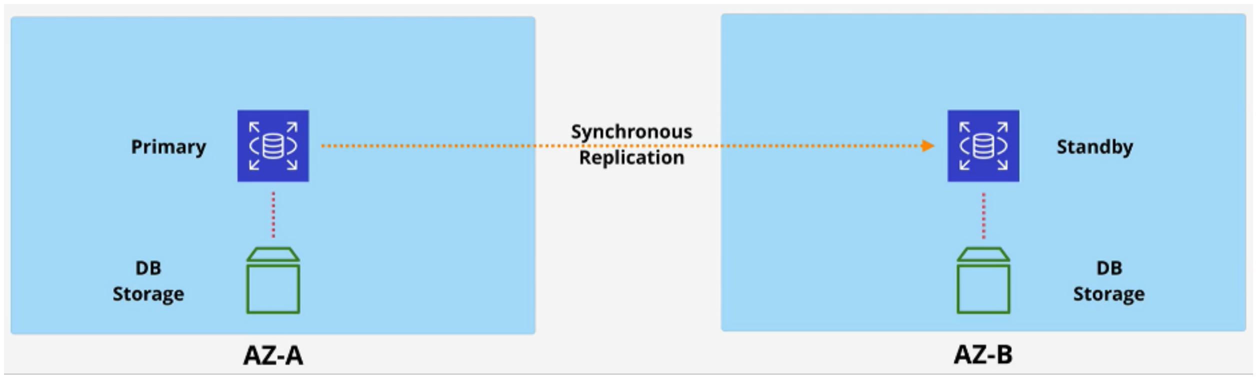
## Multi AZ – High Availability

Amazon RDS uses Always-on Availability Groups for Multi-AZ configurations in AWS regions. However, it is possible to configure a database instance in a Single-AZ deployment into a Multi-AZ deployment.

When Multi-AZ is deployed, there will be a Primary instance (always one), a Standby instance in another AZ, and synchronous replication.

The Primary database instance switches over automatically to the Standby replica if any of the following conditions apply:

- An availability zone outage, primary DB instance failure, the DB instance server type is modified, the DB instance OS ID is being patched, or a manual failover of the DB instance is initiated with Reboot with failover.
- A multi-AZ configuration works by a user accessing the RDS endpoint via a CNAME that always accesses only the primary instance. Once a user writes or modifies any data, then the database will write the data to the disk on the Primary instance while replicating the new or modified data to the Standby instance in tandem.
- If there is an event that occurs on the Primary so it may no longer be functioning, then RDS will modify the CNAME to point to the Standby instance; however, this could take up to 60-120 seconds for failover, so there may be a slight disruption.



**Note: Multi-AZ must be in the same region!**

## Read Replica

Read Replicas, as the name states, are read-only replicas of an RDS instance, and unlike Multi-AZ, they can only perform read operations and perform two main benefits.

- Performance benefits
- Availability benefits

Read Replicas use asynchronous replication to keep in sync, which means that once data is written to the DB Primary instance and stored on disk, the data is then synced to the Read Replica.

Read Replicas can be created in either the same region as the Primary instance or a different region, which is known as a Cross-Region Read Replica. AWS creates the networking and encryption for this automatically.

**Read Replica benefits:**

- Max 5x Read Replicas per instance, each performing an additional instance of read performance
- Global performance / Scale out reads
- Snapshots + Backups improve RPO
- Promote to Primary for near-zero RTO

## RDS Backup and Restore

There are two types of snapshots with RDS, and both utilize AWS managed S3 buckets:

- **Automatic Backups** – A retention period can be set from 0 to 35 days.    In addition to Automated Backups, every 5 minutes Transaction Logs (stored data that changes inside a database) are written to S3. With that being said, a Point-in-Time (PIT) restore can take place with an RPO of 5 minutes.
- **Manual Snapshots** – Function the same as EBS snapshots, where the first copy is a full snapshot, and from then on they are incremental. The manual Snapshots are not automatically deleted, which uses more storage, driving up costs. Therefore, we must delete them. Even if we delete an RDS instance, the snapshots will still be there.

RDS Restores:

- When a Restore is pushed, then RDS will create a new RDS instance, which will therefore create a new Endpoint address.
- With Snapshots, it uses a single-point-in-time restore.
- Automated Backups uses a 5-minute-point-in-time recovery using the transaction log.
- Snapshots can take a long time to restore.
- Snapshots are the only way to restore from data corruption (not replicating).

## RDS Parameter Groups

A Parameter Group consists of configuration values that are specific to the Database (DB) engine. If a DB instance is created without specifying a DB Parameter Group, the DB instance uses a default DB Parameter Group. Each default DB Parameter Group contains database engine defaults and Amazon RDS system defaults based on the engine, compute class, and allocated storage of the instance.

You can't modify the parameter settings of a default Parameter Group. Instead, you create your own Parameter Group where you choose your own Parameter settings. Not all DB engine parameters can be changed in a Parameter Group that you create.

**Note: The default Parameter Group cannot be modified.**

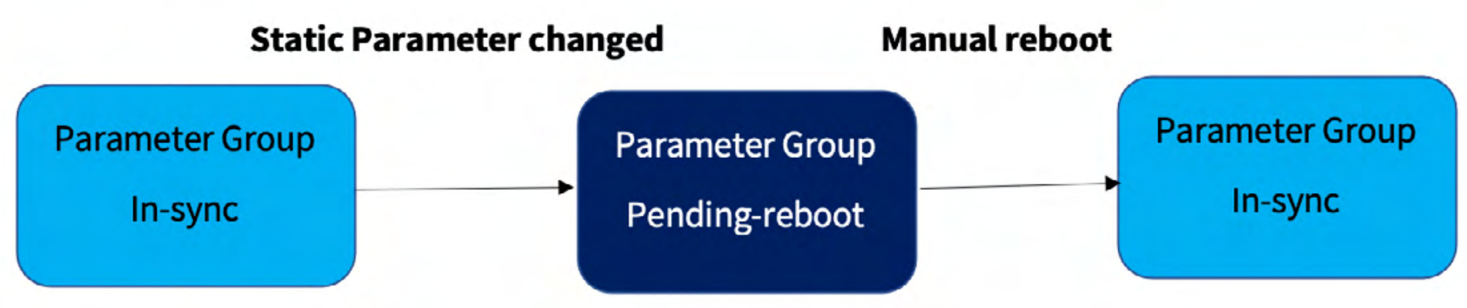
To make changes to a Parameter Group, a new Parameter Group must be created, and it will inherit settings from the default Parameter Group.

Examples of parameters are:

- Force_ssl
- Max_connections
- Time_zone

Changes to dynamic parameters always are applied immediately.

Changes to static parameters require a manual reboot.



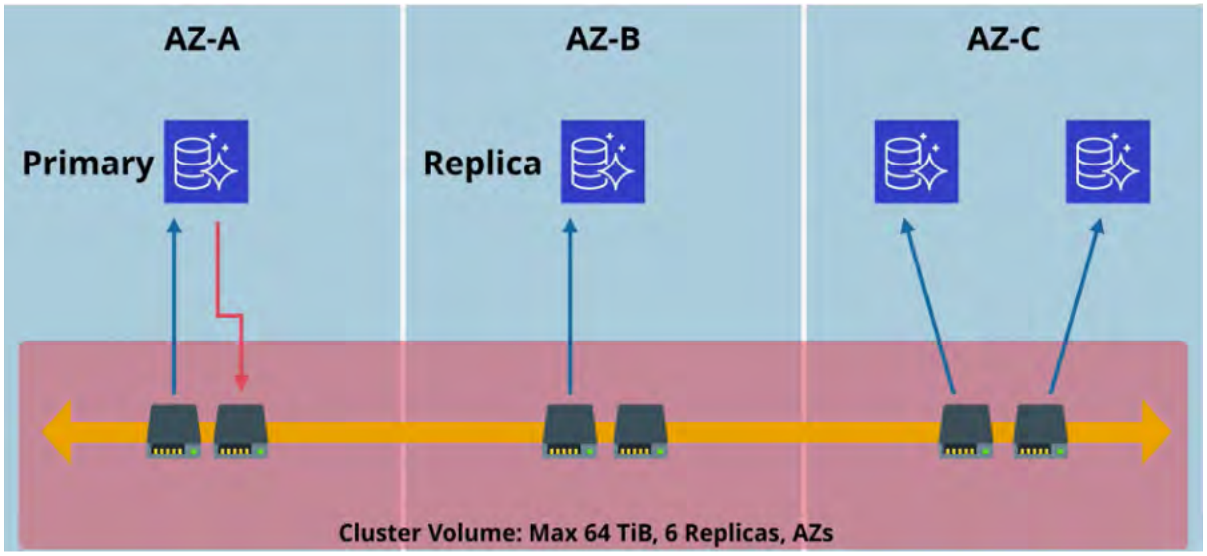
## Amazon Aurora

Aurora is Amazons RDS Relational Database service offering. Aurora is MySQL and PostgreSQL-compatible.

Aurora is:

- Five times faster than standard MySQL Databases
- Three times faster than PostgreSQL Databases
- Able to create 15 Read Replicas (Multi-AZ, Autoscaling Read-Replicas)
- Aurora Global DB – Supports Multi-region read replication
- Maintains six copies across three AZs

Aurora uses a cluster base, a single Primary instance with zero or more replicas that uses a shared cluster volume for storage (no local storage).



Aurora storage architecture is all SSD-based, which allows for low latency and high IOPs.

The Aurora Endpoints work differently than that of RDS. While there is still a CNAME that is used, there is usually a Cluster Endpoint and a Reader Endpoint. The Cluster Endpoint always points at the Primary instance, and the Reader will point at the Replicas.

**Aurora Clone and Backtrack**

Backups in Aurora work in the same way as RDS with Automated Backup and manual Snapshots. Unlike RDS, it is not possible to disable automatic backups and they have a minimum of one-day retention.

Backtrack, which is only supported with MySQL, can also be used. This allows in-place rewinds to a previous point in time to perform an in-place restore. Backtrack allows up to 72 hours of PITR or rewind option.

Fast Clones make a new database much faster than copying all the data (copy-on-write protocol).

Cloning in Aurora lets you create a copy of a DB for Read/Write, so Cloning is very different from a Read Replica. Clones will use the same storage layer as the source cluster. It is possible to clone from a clone and it also supports cross-account cloning.

Cloning is possible only within the same region. However, it can be in a different VPC, but must be in the same AZs.

Cloning is a great use case for DEV/TEST using a copy of production.

When Aurora restores, it will create a new Cluster.

**Note: It is not possible to Backtrack a clone to a time prior to the clone being created.**

## Aurora Serverless

Aurora Serverless is a relational database-as-a-service that offers MySQL and PostgreSQL interfaces.

With Aurora Serverless you don't need to provision resources in the same way as Aurora.

You still create a cluster; however, Aurora Serverless uses the concept of Aurora Capacity Units (ACU). ACU represents a certain amount of Compute and corresponding amount of memory.

Aurora Serverless is a fully managed on-demand, auto-scaling Aurora configuration. Aurora Serverless can automatically shut down when there is no load on the system. This is called "Automatic Pause." There is no charge when in this state. Additionally, another great feature is that it can automatically scale up and down based on the workload/application.

With Aurora Serverless, the same resilience is assumed as Aurora, which is six copies across AZs.

In the case of cluster or AZ-failure, Aurora Serverless creates the DB instance in a different AZ, which is an automatic multi-AZ failover. Some use cases are:

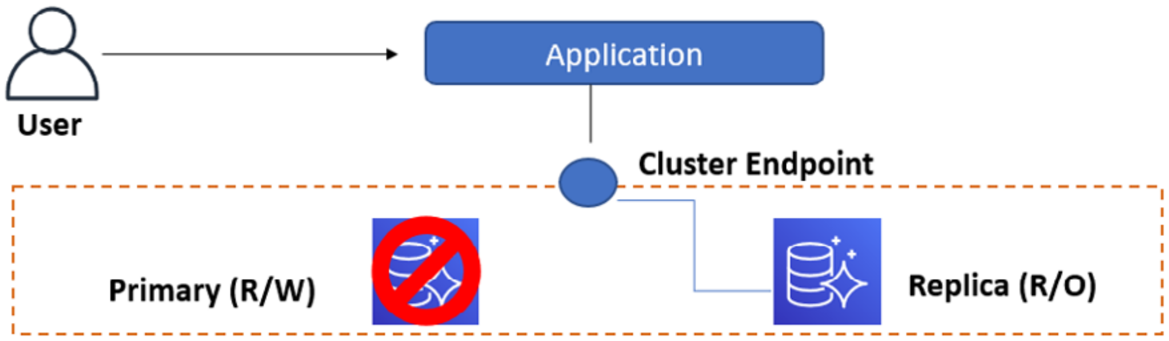
- Development and testing
- Low-volume web apps
- Infrequently used apps

### Aurora Multi-Master

So far, we have only talked about a Single-Master, which is the default Aurora mode. It equates to one Read-Write, which is a single database instance, and in addition can have zero or more Read-Only replicas.

Using Multi-Master mode, all database instances are Read and Write, so in a case of a failover, the process is quick.

**Aurora with Single Master Writer**



**Aurora Multi-Master**

In the case of Multi-Master, if the Application is configured correctly and the Writer node fails, then 1000% of the traffic will automatically be redirected to the alternative Writer node, with little to no disruption. This can create a fault-tolerant Application.

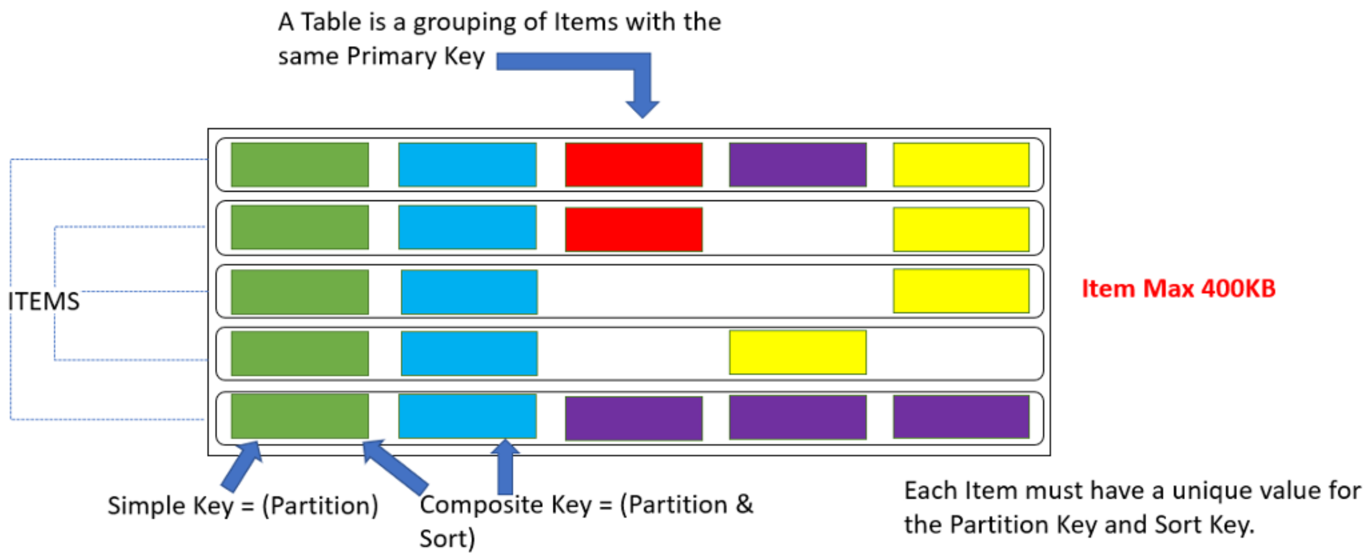
## DynamoDB

DynamoDB is a Non-Relational Database and key-value store, with ultra-low latency and high performance with scale. DynamoDB is an AWS fully managed, Serverless, NoSQL database.

DynamoDB's base entity are Tables. A Table is a grouping of Items with the same Primary Key.

A Primary Key can be of two kinds:

- A Simple Key – Partition Key (PK)
- Composite Primary Key – Partition Key (PK) or Sort Key (SK)


A Table is a grouping of Items with the same Primary Key
ITEMS
Item Max 400KB
Simple Key = (Partition)
Composite Key = (Partition & Sort)
Each Item must have a unique value for the Partition Key and Sort Key.

DynamoDB tables can be configured with provisioned capacity or on-demand capacity (speed/performance).

| Provisioned Capacity Mode | On-Demand Capacity Mode |
|---|---|
| Typically used in production | DEV/TEST or for small apps |
| Used when predictable traffic | Used when variable, unpredicted traffic |
| Can result in throttling if no auto scale | Throttling can occur if exceed 2x previous peak within 30 minutes |
| Consider reserved capacity for steady and predictable traffic (cost savings) | Instantly accommodates up to 2x the previous peak traffic on the table |
| More cost effective | Space traffic growth over previous 30 minutes |

Capacity = Speed is set on a table:

- (Write Capacity Unit) 1 WCU = 1KB PS
- (Read Capacity Unit) 1 RCU = 4KB PS

## DynamoDB Consistency

Consistency refers to the process of how when new data is written to the database, and then immediately read, that read data is immediately the same as the recent update or it is only eventually the same data.

In DynamoDB, there are two consistency models:

- Write Consistency – Standard and Transactional
- Read Consistency – Strong, Eventual, and Transactional

| Consistency | Details |
|---|---|
| Strong | The most current data (must be requested explicitly), more expensive, and scales well |
| Eventual | Default consistency for all read operations. 50% less cost than strong consistency; may or may not reflect the current data |
| Transactional Read + Writes | 2x the cost of standard writes, 2x the cost of strongly consistent reads, and ACID support across on or more tables |

It is very important to know how to calculate the WCU and RCU for the setting on the Table:

### WCU Calculation

As an example: In the event we need to Store 10 Items per second:

- 2.5K average size per item
- Calculate WCU per Item……Round up (Item size/1KB) = 3
- Multiply by average number per second = 30
- Total = 30 WCU required

**Note: WCU are 1KB in size.**

### RCU Calculation

As an example: In the event we need to Retrieve 10 Items per second:

- 2.5K average size per item
- Calculate RCU per Item… Round up (Item size/4KB) =1
- Multiply by average read ops per second = 10
- Strongly Consistent RCU required = 10
- (50% of strongly consistent) Total = RCU required 5

**Note: RCU are 4KB in size.**

## DynamoDB Indexes

Indexes in DynamoDB are a way to improve the efficiency of data retrieval operations.

DynamoDB has two types of indexes:

- LSI = Local Secondary Index – Which allows an alternative view using a different sort key.
- GSI = Global Secondary Index – Which Allows an alternative view with a different partition key, and sort key.

**Local Secondary Index (LSI)**

- Local Secondary Indexes must be created with a base table; they are not allowed after the base table has been created. There are five LSIs allowed per base table. The LSI always has the same partition key as that as the table. The LSI does have a different sort/range key than the primary index table. The indexed items have a limit of 10GB per partition. Local secondary index is "local" because every partition of a local secondary index is scoped to a table partition that has the same partition key. LSIs can only query a single partition (specified by the hash key).
- Supports Eventual, Strong, and Transactional consistency.

**GSI Global Secondary Index (GSI)**

Global Secondary Indexes can be created at any time, even after the table has been created. There is, however, a default limit of 20 per base table. GSIs let you use an alternative partition key and sort key. GSIs also have their own RCU and WCU allocations. With GSIs, there is no restriction on size of partitions.

- GSIs can query across partitions (over the entire table)
- Supports only Eventual consistency
- GSIs have their own provisioned throughput

**When to choose LSI or GSI**

This is a very popular question that arises in discussions with many customers.

**Note: AWS recommends using GSIs as default and only using LSIs when a strong consistency is required.**

**LSIs**

- When the application needs the same partition key as the table
- When you need to avoid additional costs
- When the application needs strongly consistent index reads

**GSIs**

- When the application needs different or same partition key as the table
- When the application needs finer throughput control
- When the application only needs eventually consistent index reads

| Characteristics | 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 | No Restrictions | Must be 10GB or less |
| Deletion | Can be deleted at any time | Cannot be deleted independently |
| Read Consistency | Supports Eventual Consistency | Supports either Strong or Eventual consistency |
| Throughput | Needs separate provisioned Throughput | Consumes Tables provisioned Throughput |
| Projected Attributes | Returns only projects attributes configured during creation | Can request attributes that are not projected into the index |
| Search | Search across entire table and all partitions | Search limited to a single partition |

## DynamoDB Backup and Restore

In DynamoDB, Backups are automatically encrypted, catalogued, and easily discoverable.

DynamoDB offers on-demand Backups for the database or enabling continuous Backups with Point-in-Time recovery.

- On-demand Backups – Configured from console or CLI or API operation.
- Continuous Backups with PITR – Restore is available from the last 35 days with an RPO of approximately five minutes.

DynamoDB can now be integrated with AWS Backup Service as another option.

What settings are not automatically restored?

- Tags
- CloudWatch metrics and Alarms
- Streams and TTL configs
- Auto-Scaling policies, IAM policies

**Note: To retain the original table name, delete the existing table before running the restore.**

## DynamoDB Streams

Streams are essentially a time-ordered list of Item-changes in a DynamoDB table. Every time an Item is changed in the table, a change is recorded chronologically in a DynamoDB stream.

- 24-hour rolling window of time-ordered log of all table-write activity
- Can also be read with EC2, Lambda, ES, and Kinesis
- Enabled on a per-table basis
- Records Updates, Deletes, and Inserts

There are four different view types that the stream can be configured:

- Keys_Only – Only captures key attributes of changed Item
- New_Image – Captures entire Item after changes
- Old_Image – Captures entire Item before changes
- New_and_Old_Images – Captures entire Item before and after changes

DynamoDB streams are organized into Shards. Triggers can be enabled. For example, a Lambda function can be invoked when a stream event occurs.

## DynamoDB Accelerator (DAX)

DAX is an in-memory cache that is integrated into DynamoDB to increase performance.

DAX takes the complexity out of usual caching technologies, as it has software installed into the application (SDK) that makes a single call for the data, which is  returned to DAX. DAX then either returns the data from its cache or retrieves it from the database and then caches it.

DAX is deployed with a VPC and is recommended to be deployed across multiple AZs.