# AWS-RDS

Angel Xing

# What is Amazon RDS (Relational Database Service)?

- **Managed relational database service on AWS**
  - Amazon Aurora
  - MySQL
  - PostgreSQL
  - MariaDB
  - Oracle
  - SQL Server
- **Designed for OLTP workloads**
  - Transactions (ACID)
  - High consistency
  - Low-latency reads and writes
- **Not designed for big data analytics**
  - Not for large-scale scans
  - Not for heavy aggregations

# ACID

RDS databases provide **full ACID compliance**, ensuring reliable transaction processing.

- **Atomicity**
  - A transaction is all-or-nothing
- **Consistency**
  - A transaction brings the database from one valid state to another
- **Isolation**
  - Concurrent transactions do not interfere with each other
  - Common isolation levels:
    - Read Uncommitted (dirty reads)1000->500 not submit, 500, 1000,
    - Read Committed (no dirty reads)T1: 1000->500 commit, T2: check balance 500, T3: A->C 200 commit, T4: 300
    - Repeatable Read (stable row reads), balance: 1000, A->B 500 commit, T2: check balance 1000
    - Serializable (strongest, full isolation)
      1. shared locks (s-lock), Exclusive lock(X lock), range lock,  predicate lock？？
- **Durability**

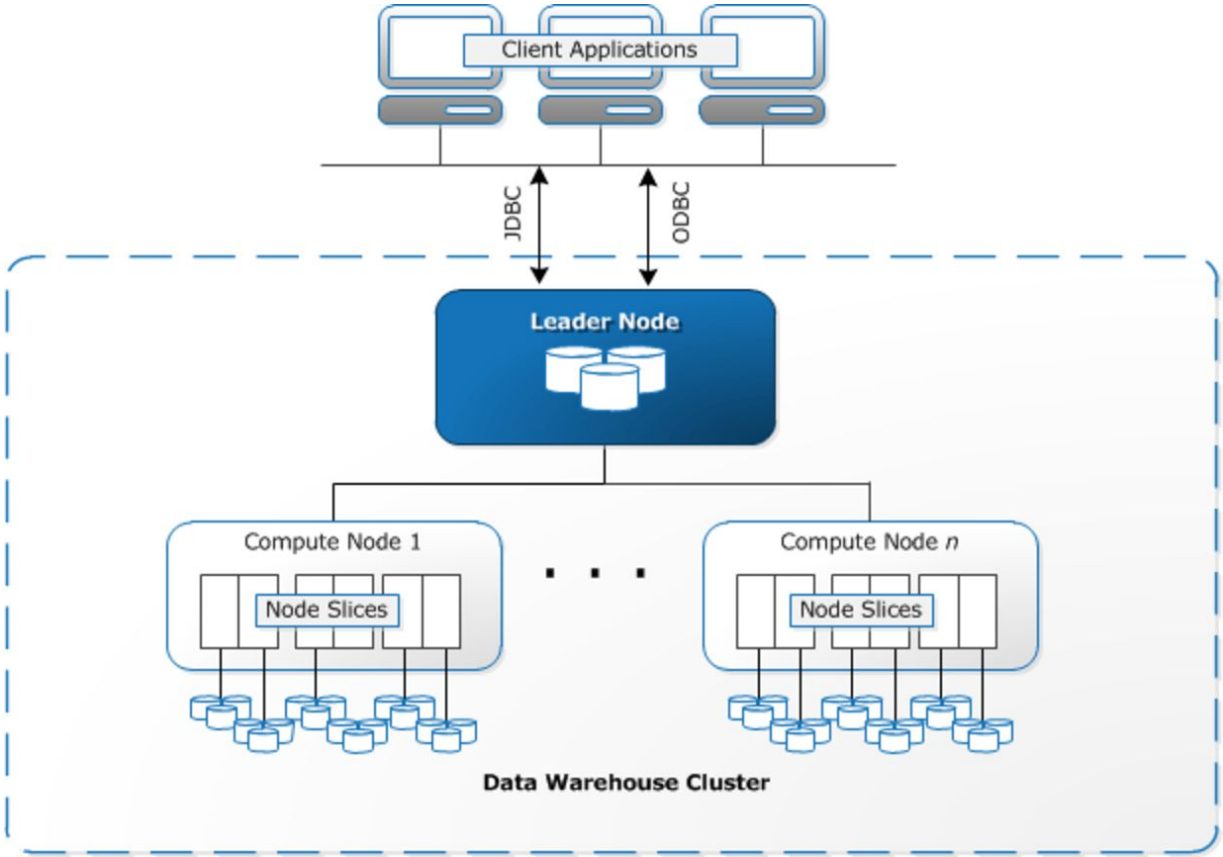
# AWS-Redshift

# Redshift

Amazon Redshift is a fully managed cloud-based data warehousing service provided by Amazon Web Services (AWS). It is designed for processing and analyzing large volumes of data in a highly performant and cost-effective manner. Redshift is based on a columnar storage architecture and is optimized for online analytical processing (OLAP) workloads.

- Accelerate analytics workloads
- Unified data warehouses and data lake
- Data warehouse modernization
- Analyze global sales data
- Store historical stock trade data
- Analyze ad impression and clicks
- Aggregate gaming data
- Analyze social trends

# MPP architecture




Columnar

Select country, sum(price)

From orders

Group by country

# Redshift spectrum

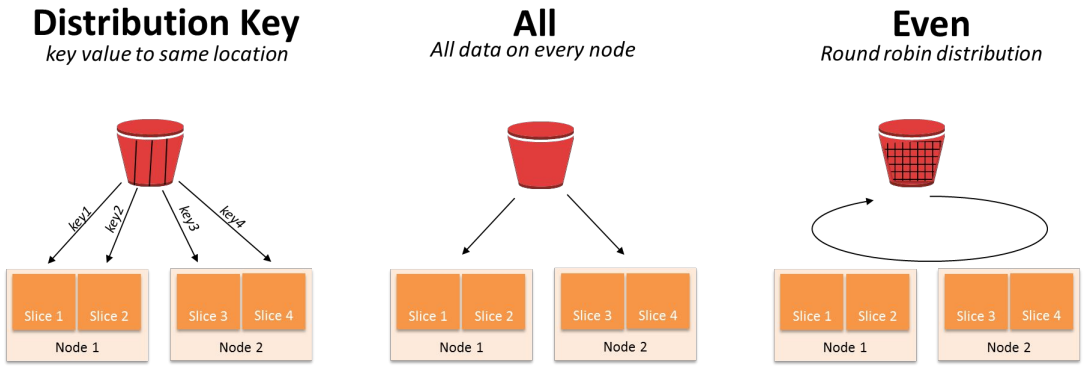
- Query exabytes of unstructrued data in s3 without loading
- Limitless concurrency
- Horizontal scaling
- Separate storage and compute resources
- Wide variety of data formats
- Support of Gzip and snappy compression

# Durability and scaling

- Replications within cluster
- Backup to s3
  - Asynchronously replicated to another region
- Auto snapshots for DR
- Vertical and horizontal scaling on demand
  - CNAME is flipped to new cluster when scaling

# Data distribution

- Auto
  - Redshift figures it out based on size of data
- Even
  - Rows distributed across slices in round-robin
- Key
  - Rows distributed based on one column
- All
  - Entire table is copied to every node: _dim



# SortKey

Compound Sort Key:

- A compound sort key consists of multiple columns defined in a specific order.
- Redshift sorts data based on the first column of the sort key and then, within each value of the first column, sorts the data based on the second column, and so on.
- Compound sort keys are useful when queries frequently filter, join, or aggregate data on multiple columns sortkey(customer_id,country_id), group by custmer_id, country_id

Interleaved Sort Key:

- An interleaved sort key includes multiple columns, similar to a compound sort key, but all columns are considered equally important in the sort order
- Redshift interleaved the data based on all columns of the sort key, ensuring a more balanced distribution of data within each data block.
- Interleaved sort keys are beneficial when queries involve a wide range of filter conditions, as they can potentially improve performance for multiple types of queries.

# Importing and exporting data

**COPY Command**

- Parallelized and highly efficient bulk loading
- Load data from:/
  - Amazon S3
  - Amazon EMR
  - Amazon DynamoDB
  - Remote hosts
- For S3:
  - Requires IAM Role
  - Manifest file recommended for large-scale loads

**INSERT INTO … SELECT**

- Load data from existing tables or queries
- Less efficient than COPY for large datasets

**UNLOAD Command**

- Export data from Redshift tables to files in Amazon S3
- Supports:
  - Parallel unload
  - Compression
  - Encryption

**Auto-Copy from Amazon S3**

- Automatically load new files arriving in S3
- Commonly used with event-driven ingestion pipelines

**Amazon Aurora Zero-ETL Integration**

- Seamless data replication without custom ETL
- Data flow example:
  - Aurora → S3 → Glue Crawler → Glue Data Catalog → Athena
- Optional:
  - Auto replication from Aurora to Redshift

**Redshift Streaming Ingestion**

- Ingest data in near real-time
- Supported sources:
  - Amazon Kinesis Data Streams
  - Amazon MSK (Kafka)

# DBLink

## Connect Redshift to PostgreSQL

- A practical way to copy and synchronize data between PostgreSQL and Amazon Redshift
- Commonly used for:
    - Incremental data sync
    - Small-to-medium reference tables
    - Cross-database querying

**Enable Required Extensions**

```
CREATE EXTENSION postgres_fdw;

CREATE EXTENSION dblink;
```

**Create Foreign Server**

```
CREATE SERVER foreign_server

FOREIGN DATA WRAPPER postgres_fdw

OPTIONS (host '<postgres_host>',

  port '<port>',

  dbname '<database_name>',

  sslmode 'require');
```

**Create User Mapping**

```
CREATE USER MAPPING FOR <redshift_user>

SERVER foreign_server

OPTIONS (user '<postgres_username>',

  password '<password>');
```

# VACUUM

- Recovers disk space from deleted or updated rows
- Reorganizes data storage to improve query performance

**VACUUM Types**

- **VACUUM FULL**
- **VACUUM DELETE ONLY**
- **VACUUM SORT ONLY**
- **VACUUM REINDEX**

# "NO" in redshift

- **Small datasets**
  - Use **Amazon RDS** instead
- **OLTP workloads**
  - Use **Amazon RDS** or **Amazon DynamoDB** instead
- **Unstructured data**
  - Perform **ETL first** using **EMR / Spark / Airflow**
- **BLOB data**
  - Do **NOT** store large binary objects in Redshift
  - Store files in **Amazon S3**, and keep only **references (paths / IDs)** in Redshift

# Resource scaling in redshift serverless

- You pay for **capacity**, measured in **Redshift Processing Units (RPUs)**
- Billing is based on **RPU-hours**, charged **per second**, plus storage

**Base RPUs**

- You can configure a **base capacity**
- Base capacity can be adjusted from **32 to 512 RPUs**
- Higher base RPUs provide **better query performance and concurrency**

# AWS-DynamoDB

# Amazon DynamoDB – NoSQL Database

Serverless

Non-relational database

Fully distributed

Low cost with automatic scaling

Standard and Infrequent Access (IA) table classes

# DynamoDB Data Model

**Database → Table (No Schema Layer)**

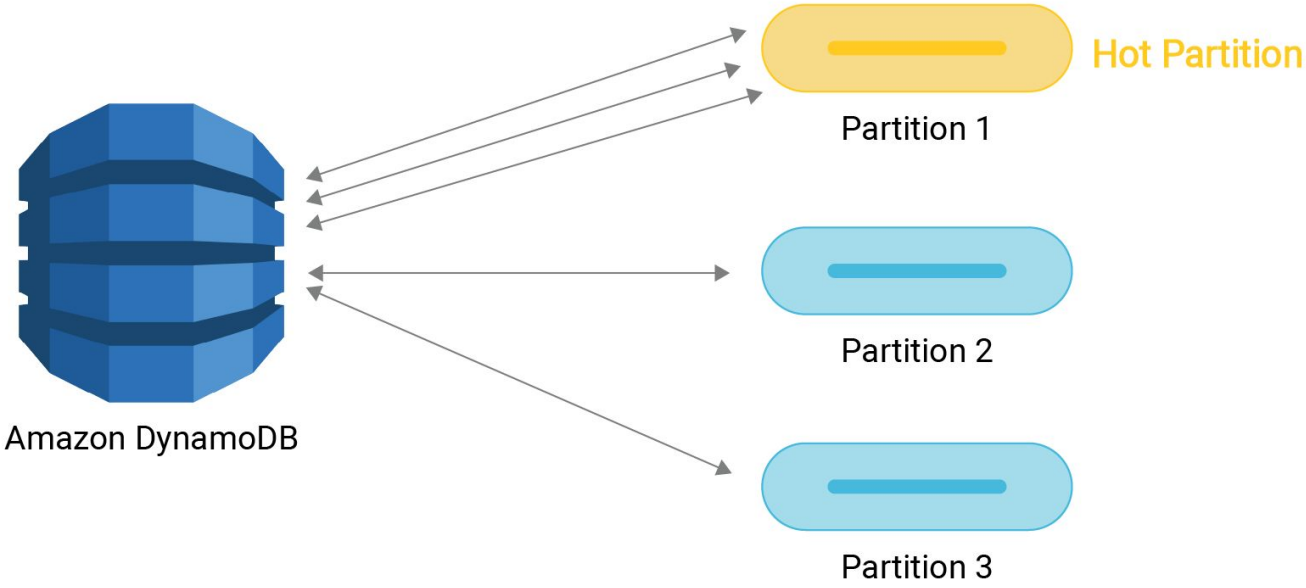
- DynamoDB is made of **tables**
- Each table has a **primary key**
- Each table can have an **unlimited number of items (rows)**
- Each item consists of **attributes(column)**
- Maximum size of a single item is **400 KB**
- **Data types**
    - Scalar types
    - Document types
    - Set types

⚠️ DynamoDB does **NOT** have database/schema/table hierarchy like RDS

# Primary Key

- **Partition Key (HASH key)**
    - A simple primary key consisting of a single attribute
    - If a table has only a partition key, **each item must have a unique partition key value**
- **Composite Primary Key (Partition Key + Sort Key)**
    - A primary key composed of a partition key and a sort key
    - Multiple items can share the same partition key
    - Items with the same partition key **must have different sort key values**

# DynamoDB partition



# “NO” in DynamoDB

- Prewritten applications tied to traditional relational database : use RDS
- Join or complex transactions
- BLOB data store in s3 + meta data in DynamoDB
- Large data with low 1/0 rate : use s3

# RCU and WCU

## RCU and WCU

- **WCU (Write Capacity Unit)**
  One write per second for an item **up to 1 KB** in size
- **RCU (Read Capacity Unit)**
  Represents:
  - **1 strongly consistent read per second**, **or**
  - **2 eventually consistent reads per second**
    for an item **up to 4 KB**
- Read consistency:
  - Strongly consistent read
  - Eventually consistent read (**default**)

Write **10 items per second**, item size **2 KB**

Write **6 items per second**, item size **4.5 KB ???**