//Akash Chandran

//Database Assignment

Database notes

Database Concepts refer to the fundamental ideas behind organizing, managing, and interacting with data in a structured way. **database** is a structured collection of data that is stored and managed electronically. The goal is to store data efficiently and allow easy retrieval, updating, and management.

RDBMS (Relational Database Management System)

An RDBMS is a type of database management system (DBMS) that stores data in the form of tables and uses a relational model to define and organize data. RDBMSs are designed to handle large amounts of structured data while maintaining relationships between tables using keys.

Key concepts of RDBMS:

- **Tables:** Data is stored in tables, which consist of rows (records) and columns (attributes). Each row represents a record, and each column represents a field or attribute of the record.
- **Schema**: The schema defines the structure of the database, including the tables, columns, data types, and relationships. It acts like a blueprint for how data is organized and validated.
- **Tables**: As mentioned, data is stored in tables. A table consists of:
  Rows (Records): Each row represents a single entity or instance of data.
  Columns (Fields/Attributes): Each column defines a specific attribute of the entity.
- **Primary Key**: A column or set of columns that uniquely identifies each row in the table.
- **Foreign Key**: A column that establishes a link between two tables. It's a reference to a primary key in another table.
- **SQL Operations**: RDBMSs allow for a wide range of SQL operations to interact with data: **CRUD**

Benefits of RDBMS:

- **Structured Data Organization:** Data is organized in a clear and well-defined manner
  easier to understand, manage, and query.
- **Data Integrity**: ACID properties ensure data consistency and accuracy.
  Atomicity: Ensures that all operations within a transaction are completed successfully or none.
  Consistency: The database must always transition from one valid state to another after a transaction.
  Isolation: Transactions are isolated from one another to prevent interference.
  Durability: Once a transaction is committed, its effects are permanent, even in the event of a system failure.
- **Data Relationships:** Ability to define relationships between tables simplifies complex data retrieval and manipulation.
- **SQL Support:**
  Standardized Query Language: SQL (Structured Query Language) is a widely used and standardized language for querying and managing relational databases.
    - **DML (Data Manipulation Language):** Allows manipulation of data in the database.
      INSERT: Add new records.

UPDATE: Modify existing records.
DELETE: Remove records.

- **DDL (Data Definition Language):** Used to define or modify the database structure.
  CREATE: Create tables, indexes, etc.
  DROP: Delete tables, indexes, etc.
  TRUNCATE: Remove all data from a table but keep the structure.
- **DCL (Data Control Language):** Allows for permissions and access control.
  GRANT: Assign permissions to users.
  REVOKE: Remove permissions.
- **DQL (Data Query Language):** Allows users to query data.
  SELECT: Retrieve data from one or more tables.
- **TCL (Transaction Control Language):** Manages transaction behaviour.
  COMMIT: Save changes.
  ROLLBACK: Undo changes.
  SAVEPOINT: Set a point to roll back to.

- **Scalability:** RDBMS can scale to accommodate large datasets and high volumes of transactions. But cannot scale like no-SQL

Phantom Reads

A phantom read occurs when a transaction reads a set of rows that match a certain condition, but another transaction concurrently inserts, deletes, or modifies rows, causing the original transaction to see a different set of rows when it re-queries the database. This issue can occur in multi-user or multi-transaction environments when different transactions are executing concurrently.

Serializable Vs Repeatable read

Repeatable Read:

The Repeatable Read isolation level ensures that once a transaction reads a row, it will always see the same value for that row throughout the transaction. In other words, it prevents non-repeatable reads but still allows phantom reads

Non-repeatable Read: Occurs when a transaction reads a value from the database, and another transaction modifies or deletes that value before the first transaction reads it again, causing inconsistent data. **Repeatable Read** prevents this scenario.

Phantom Read: Occurs when a transaction reads a set of rows that match a condition, but another transaction inserts, deletes, or updates rows, causing the first transaction to see a different set of rows when it queries the same condition again. **Repeatable Read** does **not** protect against phantom reads.

Serializable:

The **Serializable** isolation level is the highest level of isolation. It ensures that transactions are executed in such a way that the results are the same as if they were executed **serially** (one after the other), even if they run concurrently. **Serializable** prevents **dirty reads**, **non-repeatable reads**, and **phantom reads**.

Eventual consistency

Eventual consistency means that, over time, all nodes in the system will eventually agree on the state of the data, but not necessarily right away. During this time, different nodes might return different results when queried, but eventually, the system will settle on a consistent state.

Key Characteristics:

- **High Availability**: eventual consistency can tolerate network partitions and node failures, ensuring high availability even in challenging environments. but cannot guarantee consistency
- **Scalability:** Well-suited for distributed systems that need to handle large volumes of data and high write throughput.
- **Limited Consistency Guarantees:** Does not guarantee immediate consistency. May experience temporary inconsistencies during replication. Reads may return stale data for a brief period.

CAPS THEOREM

The CAP Theorem (also known as Brewer's Theorem) is a principle that applies to distributed databases. It states that a distributed database system can achieve at most two of the following three goals simultaneously:

1. Consistency (C): Every read operation will return the most recent write. All nodes in the system have the same data at the same time.

2. Availability (A): Every request (read or write) will receive a response, even if some of the database nodes are down. The system is always available for operations.

3. Partition Tolerance (P): The system can continue to operate correctly even if network partitions occur, meaning some parts of the system can't communicate with others

Sharding in RDBMS

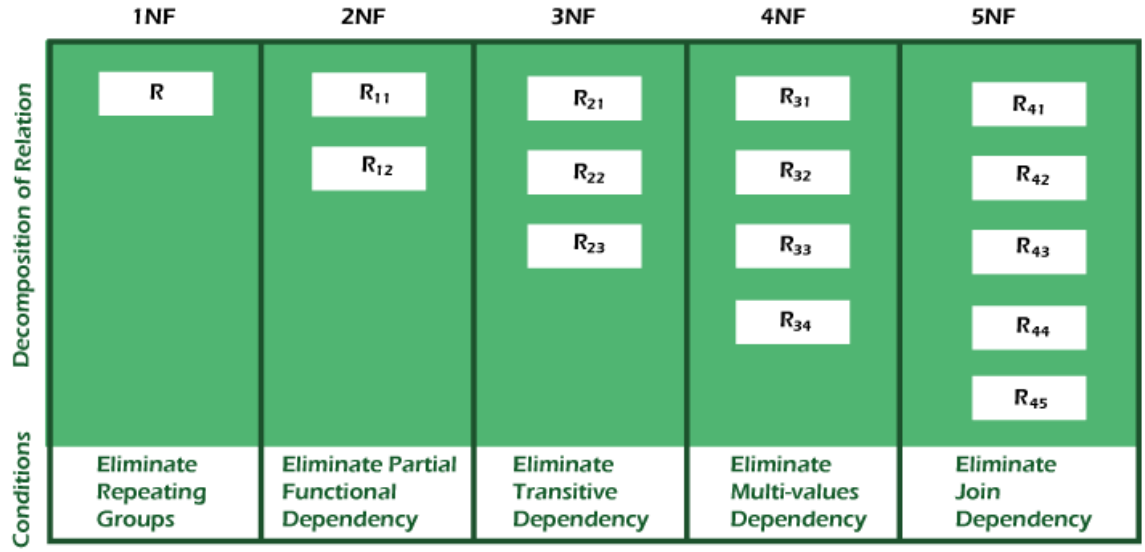
Sharding is a method of horizontal scaling where data is distributed across multiple servers (called shards) to improve performance and manage large datasets. Two Types of Sharding:

Horizontal Sharding (Data Sharding): Data is split across multiple databases or servers based on some criteria

Vertical Sharding (Database Sharding): Different tables or types of data are stored on different servers. Each shard holds a subset of the database schema.

Normalization

Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies. The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.



Basic SQL Commands:

Here's a glimpse into some fundamental SQL commands to get you started:

SELECT: used to retrieve data from one or more tables. can specify columns (attributes) filter the results using WHERE clauses.

INSERT: insert new rows of data into a table. specify table name values for each column in the new row.

UPDATE: modify existing data in a table. can update specific columns based on a WHERE clause to target the rows you want to modify.

DELETE: This command removes rows from a table. use a WHERE clause to delete specific rows based on criteria.

CREATE TABLE: create new tables within the database defining the structure with column names and data types.

Indexes in Databases

An **index** in a database is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional space and increased maintenance during data modifications (INSERT, UPDATE, DELETE). Indexes are primarily used to speed up the **search** (querying) operations but can also impact the performance of **write operations**.

```
CREATE INDEX idx_customerid ON Customer (CustomerID);
```

//Create database

```
CREATE DATABASE database_name;
```

//use database

```
Show DATABASE;
```

//create table

```
CREATE TABLE people (
        id BIGINT PRIMARY KEY,
        name varchar(500),
        email varchar(500),
        first_name varchar(500),
        last_name varchar(500),
        state varchar(200),
        birthday datetime
      );
```

//insert into table

```
INSERT INTO people (id, name, email, first_name, last_name, state,
birthday)
VALUES (1,'John Doe', 'john.doe@example.com', 'John', 'Doe',
'California', '1990-01-01');
```

//view table

```
Select * from people;
```

//DQL – Data Query Language

```
SELECT column1, column2, ... FROM table_name WHERE condition;
```

//Data Definition Language (DDL)

```
CREATE DATABASE database_name;

CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    ...
);

ALTER TABLE table_name ADD column_name datatype;

DROP DATABASE database_name;

DROP TABLE table_name;
```

//Data Manipulation Language (DML)

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);

UPDATE table_name
SET column1 = value1, column2 = value2, ...
```

```
WHERE condition;


DELETE FROM table_name WHERE condition;
```

//Data Control Language (DCL)

```
GRANT SELECT, INSERT ON Employees TO user1;

REVOKE SELECT ON Employees FROM user1;
```

//Transaction Control Language (TCL)

```
COMMIT;

ROLLBACK;

SAVEPOINT savepoint_name;

SET TRANSACTION ISOLATION LEVEL level;
```

//Backup

```
mysqldump -u root -p database_name > backup.sql
```

//Restore

```
mysql -u root -p database_name < backup.sql
```