**Ketan Pillai**

**MODULE: 5 (Database)**

**Basics of Database**

**1. What do you understand By Database?**

Ans: A database is a structured collection of organized data.

A database is like a digital filing cabinet where we store information in a structured way on a computer. It helps us keep things organized and find what we need quickly. Think of it as a big electronic notebook where we can write down and store lots of different types of information, like names, numbers, and descriptions. There are different kinds of databases, but they all work to help us manage and access data efficiently.

Common database systems include relational databases (like MySQL, PostgreSQL), NoSQL databases (like MongoDB), and others tailored for specific data types and use cases.

**2. What is Normalization?**

Ans: Normalization is a process used to organize data in a database efficiently by reducing redundancy and dependency.

It's a way to make sure your database is well-organized and efficient. You break down big tables into smaller ones and connect them logically, so you don't repeat the same information too much. This helps keep everything good, prevents mistakes, and makes it easier to work with the data.

**3. What is Difference between DBMS and RDBMS?**

Ans:

**1) DBMS (Database Management System):**

- DBMS is like a general manager for databases. It helps in creating, organizing, and managing databases.

- It can handle different ways of organizing data but doesn't specifically focus on tables or relationships between them.

- It provides tools for storing, getting, changing, and deleting data, but it might not enforce rules about how different pieces of data relate to each other.

**2) RDBMS (Relational Database Management System):**

- RDBMS is a specific type of DBMS that organizes data into tables with connections between them.

- It makes sure data is stored in rows and columns, like a spreadsheet, and defines how different tables are related.

- Examples of RDBMS are MySQL, PostgreSQL, Oracle Database, SQL Server, etc.

**4. What is MF Cod Rule of RDBMS Systems?**

Ans: Total Codd's original = 12 rules

**1. Information Rule:** All data should be neatly organized into tables, each representing a different type of information.

**2. Guaranteed Access Rule:** You should always be able to find any piece of data using a specific set of information, like the table name, primary key, and column name.

**3. Systematic Treatment of Null Values:** If some data is missing, there should be a consistent way to handle it.

**4. Active Online Catalog:** The structure of the database, like what tables it has, should be stored within the database itself, so you can easily see and change it.

**5. Comprehensive Data Sublanguage Rule:** The system should have a complete language for defining, changing, and asking questions about the data.

**6. View Updating Rule:** Views, which are basically filtered versions of the data, should be updatable by the system.

**7. High-level Insert, Update, and Delete:** You shouldn't have to write complicated code to add, change, or remove data.

**8. Physical Data Independence:** If the way data is stored changes, it shouldn't mess up the programs that use the data.

**9. Logical Data Independence:** If the way data is organized changes, it shouldn't mess up the programs that use the data.

**10. Integrity Independence:** Rules about how data should behave, like making sure each piece of data is unique, should be stored in a central place and not mixed up with the programs that use the data.

**11. Distribution Independence:** If data is spread out over different places, users shouldn't have to worry about it.

**12. Non-subversion Rule:** Even if someone tries to bypass the rules of the database system, they shouldn't be able to mess things up.

**5. What do you understand By Data Redundancy?**

Ans: Data redundancy happens when the same information is stored in more than one place in a database or across different databases.

Data redundancy can cause a few problems:

**1. Wasted Space:** It takes up extra room to store the same data in multiple places, which can cost more money and use up more resources.

**2. Inconsistent Data:** If the same data is different in different places, it can cause confusion and mistakes because people might be using outdated or incorrect information.

**3. More Complicated:** Keeping track of redundant data makes everything more complicated. It's harder to keep things organized and to make changes when needed.

**4. Less Reliable Data:** When data is redundant, there's a higher chance that something might go wrong, like data getting lost or mixed up, which can mess with how trustworthy the information is.

**6. What is DDL Interpreter?**

Ans: A DDL interpreter is like a translator for the database. It handles commands that define or change how the database is structured, like creating new tables or changing existing ones.

So when you tell it to create, alter, drop, or truncate something in the database, it makes those changes happen by translating your commands into the specific actions the database needs to take.

**7. What is DML Compiler in SQL?**

Ans: The DML Compiler in SQL deals with commands that manipulate data, like selecting, inserting, updating, and deleting. It has a few main jobs:

**1. Query Parser:** It checks if the commands you've written are written correctly and make sense.

**2. Query Optimizer:** This part figures out the best way to carry out your commands so they run as fast as possible.

**3. Query Executor:** It's the part that actually does the work, interacting with the database to make your changes happen.

**4. Transaction Manager:** This keeps track of everything to make sure your changes happen correctly and don't mess up the data.

**8. What is SQL Key Constraints writing an Example of SQL Key Constraints?**

Ans: SQL key constraints are like rules for columns in a table that help keep data organized and consistent. There are a few types:

**1. Primary Key Constraint:** It's like a special rule that makes sure each row in a table is unique.

Only one primary key can be able to give in whole table

For example, in a table of students, each student might have a unique ID number:

CREATE TABLE Students (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(50),

age INT

);

**2. Unique Key Constraint:** This is similar to a primary key, but it allows for NULL values. It just makes sure that no two rows have the same value in a specific column:

CREATE TABLE Employees (

employee\_id INT UNIQUE,

employee\_name VARCHAR(50),

department\_id INT

);

**3. Foreign Key Constraint:** This is like a connection between two tables. It makes sure that a value in one table matches a value in another table.

CREATE TABLE Orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

order\_date DATE,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)

);

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY,

customer\_name VARCHAR(50),

email VARCHAR(100)

);

**9. What is save Point? How to create a save Point write a Query?**

Ans: A save point in SQL is like a bookmark in a book. It marks a spot in a transaction where you can later go back if something goes wrong.

-- Start a transaction

START TRANSACTION;

-- Execute SQL statements within the transaction

INSERT INTO employees (id, name, salary) VALUES (1, 'John', 50000);

INSERT INTO employees (id, name, salary) VALUES (2, 'Jane', 60000);

-- Create a savepoint named 'before\_update'

SAVEPOINT before\_update;

-- Update salary for employee with ID 1

UPDATE employees SET salary = 55000 WHERE id = 1;

-- Check the updated records

SELECT \* FROM employees;

-- Rollback to the savepoint 'before\_update'

ROLLBACK TO SAVEPOINT before\_update;

-- Check the records after rolling back

SELECT \* FROM employees;

-- Commit the transaction

COMMIT;

**10.What is trigger and how to create a Trigger in SQL?**

Ans: A trigger in SQL is like a little robot that wakes up and does something automatically when certain things happen in the database, like when you add, change, or remove data.

To create a trigger:

1. Decide when you want it to happen (`BEFORE` or `AFTER`) and what action triggers it (`INSERT`, `UPDATE`, `DELETE`).

2. Choose the table where the trigger will watch for changes (`ON table\_name`).

3. Write down what the trigger should do inside a special block.

Here's an example:

CREATE TRIGGER update\_last\_updated

AFTER UPDATE

ON employees

FOR EACH ROW

BEGIN

UPDATE employees

SET last\_updated = CURRENT\_TIMESTAMP

WHERE id = :OLD.id;

END;