MODULE 5 (DATABASE)

1.) What do you understand By Database

A database is an organized collection of information that is stored and managed on a computer. It allows you to easily add, retrieve, update, and delete data. Think of it as a digital filing cabinet where you can store various types of data (like names, addresses, phone numbers) and quickly find or modify them when needed.

Databases are often used by websites, apps, and businesses to manage large amounts of information in a structured way. Common examples include customer records for an online store or user profiles for a social media platform.

2.) What is Normalization?

Normalization in a database is the process of organizing data to reduce redundancy (duplicate data) and ensure data integrity. It involves structuring a database into smaller, related tables, so that each piece of information is stored only once. This makes the database more efficient and easier to maintain.

In simple terms, it's like cleaning up and organizing a messy closet by grouping similar items together, so you don't have the same thing in multiple places, and it's easier to find and manage later.

3.) What is Difference between DBMS and RDBMS?

The main difference between **DBMS (Database Management System)** and **RDBMS (Relational Database Management System)** is how they organize and manage data:

1. **DBMS**:
    - A general system to store and manage data in a database.
    - It doesn't enforce relationships between data.
    - Suitable for small databases with simpler data structures.
    - Example: File systems, XML, etc.
2. **RDBMS**:
    - A specialized type of DBMS where data is stored in **tables** (rows and columns) and relationships between the data are defined using **keys** (like primary keys and foreign keys).
    - It maintains relationships between tables and ensures data integrity.

- Suitable for complex databases with large amounts of related data.
- Example: MySQL, PostgreSQL, Oracle.

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

E.F. Codd's rules for RDBMS define what makes a system truly "relational." Here’s a super short summary:

1. All data is in tables (rows and columns).
2. Data is accessible by table names, row keys, and column names.
3. Null values (missing data) are handled correctly.
4. Metadata (info about the database) is stored in the database itself.
5. SQL-like language is supported for all tasks (queries, updates, etc.).
6. Views (virtual tables) can be updated like real tables.
7. Bulk operations (insert, update, delete) are supported.
8. Changes in physical storage don’t affect how you access data.
9. Database changes (like adding columns) shouldn’t break applications.
10. Data rules (like accuracy checks) are stored in the database.
11. Works even if distributed across different locations.
12. No workarounds to bypass relational rules.

These rules ensure an RDBMS properly manages relational data and maintains data integrity.

5.) What do you understand By Data Redundancy?

Data redundancy means storing the same piece of information in multiple places in a database or system. This can lead to problems like:

1. Wasting storage space : Since the same data is repeated unnecessarily.
2. Data inconsistency : If one copy of the data is updated but the others aren’t, leading to conflicting information.
3. Maintenance issues : It's harder to manage and update the data because changes need to be made in multiple places.

In simple terms, data redundancy is like having several copies of the same document scattered around your computer. It can cause confusion and take up unnecessary space.

6.) What is DDL Interpreter?

A DDL (Data Definition Language) Interpreter is a part of a database management system (DBMS) that processes and interprets DDL commands . DDL commands are used to define and manage the structure of a database, such as creating, altering, or deleting tables, indexes, and other database objects.

For example, when you run a command like `CREATE TABLE` or `ALTER TABLE`, the DDL interpreter translates that command and ensures that the database structure is updated according to the request.

In simple terms, it's like the part of the database that understands instructions for building or changing the "blueprint" of how the data is organized.

7.) What is DML Compiler in SQL?

A DML (Data Manipulation Language) Compiler in SQL is a part of the database system that processes DML commands, which are used to work with the data inside a database. DML commands include actions like **inserting, updating, deleting, and retrieving data from tables.

When you run SQL commands like `INSERT`, `UPDATE`, `DELETE`, or `SELECT`, the DML compiler translates these commands into low-level instructions that the database can understand and execute efficiently.

In simple terms, it's like a tool that converts your requests to add, change, or get data from the database into something the system can perform.

8.) What is SQL Key Constraints writing an Example of SQL Key Constraints

SQL Key Constraints are rules applied to columns in a database to ensure data accuracy and consistency. They help maintain relationships between tables and ensure that the data in the database is valid.

Here are some common SQL key constraints:

1. **Primary Key :** Ensures each row in a table is unique and cannot be null.
   -Example:
    sql
```
 CREATE TABLE Students (
     StudentID INT PRIMARY KEY,
     Name VARCHAR(50)
 );
```

   -This makes `StudentID` unique for each student.

2. **Foreign Key :** Links two tables and ensures that the value in one table must exist in another (maintaining relationships).
   - Example:
     sql
```
  CREATE TABLE Enrollments (
     EnrollmentID INT PRIMARY KEY,
     StudentID INT,
     CourseID INT,
     FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
 );
```

   - This makes sure that `StudentID` in `Enrollments` exists in the `Students` table.

3. **Unique Key :** Ensures all values in a column are unique, but allows null values.
   - Example:
     sql
```
 CREATE TABLE Employees (
     EmployeeID INT PRIMARY KEY,
     Email VARCHAR(100) UNIQUE
 );
```

   - This ensures `Email` is unique for each employee.

4. **Not Null :** Ensures that a column cannot have null values.
   - Example:
     sql
```
 CREATE TABLE Products (
     ProductID INT PRIMARY KEY,
     ProductName VARCHAR(50) NOT NULL
 );
```
   - This ensures `ProductName` cannot be left empty.

These constraints help maintain data integrity and establish meaningful relationships between tables.

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

A **SAVEPOINT** in SQL is a marker you can set within a transaction that allows you to roll back to that point if needed. This way, you can undo specific parts of a transaction without affecting the entire process.

```
BEGIN TRANSACTION;

INSERT INTO Employees (EmployeeID, Name) VALUES (1, 'Alice');
SAVEPOINT MySavePoint;

INSERT INTO Employees (EmployeeID, Name) VALUES (2, 'Bob');
-- Suppose there's an error or you want to undo this insertion.

ROLLBACK TO MySavePoint;  -- This undoes the last insert.

COMMIT;  -- Finalizes the transaction.
```

In this example:

- A savepoint named `MySavePoint` is created after inserting Alice.
- If there's an issue with inserting Bob, you can roll back to the savepoint, keeping Alice's data intact.

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

A **trigger** in SQL is a piece of code that runs automatically in response to certain actions in a database, such as **INSERT**, **UPDATE**, or **DELETE** operations on a table. Triggers are often used to enforce business rules, maintain data integrity, or perform automatic tasks like logging changes.

Here’s a short example that creates a trigger to log when a new record is inserted into the `Orders` table:

```
CREATE TRIGGER LogNewOrder
AFTER INSERT ON Orders
FOR EACH ROW
BEGIN
   INSERT INTO OrderLogs (OrderID, ActionDate)
   VALUES (NEW.OrderID, NOW());
END;
```

Trigger Name: `LogNewOrder`
Event: Fires **after an insert** on the `Orders` table.
Action: Inserts data into the `OrderLogs` table, capturing the `OrderID` and the timestamp.