Module.5 Database

- **What do you understand by Database?**

  A database is a structured collection of data that is organized in a way that allows for efficient storage, retrieval, and manipulation of data. Databases are used to store and manage various types of information, such as customer records, product catalogs, financial data, and more. They provide a structured and organized way to store and access data, making it easier to perform tasks like querying, updating, and analyzing information.

- **What is Normalization?**

  Normalization is a database design technique used to eliminate data redundancy and improve data integrity in relational databases. It involves breaking down large tables into smaller, related tables and establishing relationships between them through keys (usually primary and foreign keys). The primary goal of normalization is to minimize data redundancy and ensure that each piece of data is stored in only one place to reduce the risk of data anomalies and improve database efficiency.

- **What is the difference between DBMS and RDBMS?**
  - **DBMS (Database Management System):** A DBMS is software that manages databases. It provides tools and functions for creating, retrieving, updating, and managing data. DBMS systems may or may not follow the principles of the relational model. Examples include Microsoft Access and SQLite.
  - **RDBMS (Relational Database Management System):** RDBMS is a specific type of DBMS that follows the relational model of data storage. It organizes data into tables with rows and columns, enforces data integrity through constraints, and uses SQL (Structured Query Language) for data manipulation. Examples include MySQL, PostgreSQL, and Oracle.

- **What is MF Cod Rule of RDBMS Systems?**

  There is no commonly recognized term "MF Cod Rule" in the context of RDBMS or databases. It might be a typo or a specific rule or concept with limited use. If you have more context or details about what you mean by "MF Cod Rule," please provide additional information, and I'll try to clarify.

- **What do you understand by Data Redundancy?**

  Data redundancy refers to the situation where the same piece of data is stored in multiple places within a database or across databases. It occurs when data is duplicated unnecessarily. Data redundancy can lead to various issues, including increased storage requirements, data inconsistency, and a higher risk of data anomalies.

- **What is DDL Interpreter?**

DDL (Data Definition Language) Interpreter is a component of a database management system (DBMS) responsible for processing and executing Data Definition Language statements. DDL statements are used to define and manage the structure of a database, including creating tables, defining constraints, and altering database schema. The DDL Interpreter parses and executes these statements to create, modify, or delete database objects.

- **What is DML Compiler in SQL?**

There isn't a specific concept known as a "DML Compiler" in SQL. However, DML (Data Manipulation Language) in SQL refers to the subset of SQL statements used to manipulate data in a database. This includes statements like SELECT, INSERT, UPDATE, and DELETE. DML statements are processed by the SQL engine of the DBMS, not a separate compiler.

- **What is SQL Key Constraints, and can you provide an example?** SQL Key Constraints are rules applied to columns in a relational database table to enforce data integrity. There are several types of key constraints:

Primary Key: Ensures that each value in a column is unique and not NULL. It identifies a unique record in the table. Example:

- CREATE TABLE Students (
  student_id INT PRIMARY KEY,
  student_name VARCHAR(255)
);

Unique Key: Ensures that values in a column are unique, but it allows NULL values. Example:

CREATE TABLE Employees (
  employee_id INT UNIQUE,
  employee_name VARCHAR(255));

Foreign Key: Enforces referential integrity by linking a column to a primary key in another table. Example:

CREATE TABLE Orders (
  order_id INT PRIMARY KEY,
  customer_id INT,
  FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);

- **What is a Save Point, and how do you create one in SQL?** A savepoint is a point within a transaction to which you can later roll back if needed. It allows you to create intermediate points during a transaction, so you can undo changes made after that savepoint without affecting the entire transaction. To create a savepoint in SQL, you can use the **`SAVEPOINT`** statement:

SAVEPOINT my_savepoint;

- **What is a Trigger, and how do you create one in SQL?**

A trigger is a database object in SQL that is associated with a table and automatically executes a specified action when certain events (such as INSERT, UPDATE, DELETE) occur on that table. Triggers are used to enforce data integrity, perform auditing, or automate tasks based on database events.

Here's an example of creating a simple trigger in SQL that automatically inserts a record into an audit table when a new record is inserted into the main table:

```
CREATE TABLE MainTable (
  id INT PRIMARY KEY,
  data VARCHAR(255)
);

CREATE TABLE AuditTable (
  id INT AUTO_INCREMENT PRIMARY KEY,
  action VARCHAR(255),
  timestamp TIMESTAMP
);

DELIMITER //
CREATE TRIGGER InsertTrigger AFTER INSERT ON MainTable
FOR EACH ROW
BEGIN
  INSERT INTO AuditTable (action, timestamp) VALUES ('INSERT', NOW());
END;
//
DELIMITER ;
```

- **Create Table Name : Student and Exam:-**

```
-- Create a Student table
CREATE TABLE Student (
  student_id INT PRIMARY KEY,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  date_of_birth DATE,
  email VARCHAR(100)
);

-- Create an Exam table
CREATE TABLE Exam (
  exam_id INT PRIMARY KEY,
  exam_name VARCHAR(100),
```

```
    exam_date DATE,
    student_id INT,
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
);
```

- **Create table given below: Employee and IncentiveTable**

SQL:-

```
-- Create an Employee table
    CREATE TABLE Employee (
      employee_id INT PRIMARY KEY,
      first_name VARCHAR(50),
      last_name VARCHAR(50),
      hire_date DATE,
      department VARCHAR(100),
      salary DECIMAL(10, 2)
    );

-- Create an IncentiveTable table
CREATE TABLE IncentiveTable (
  incentive_id INT PRIMARY KEY,
  employee_id INT,
  incentive_amount DECIMAL(8, 2),
  incentive_date DATE,
  FOREIGN KEY (employee_id) REFERENCES Employee(employee_id)
);
```

a) **Get First_Name from the employee table using Tom as "Employee Name":-**

```
SELECT first_name AS "Employee Name" FROM Employee WHERE first_name =
'Tom';
```

b) **Get FIRST_NAME, Joining Date, and Salary from the employee table:-**

```
SELECT first_name, hire_date AS "Joining Date", salary FROM Employee;
```

c) **Get all employee details from the employee table ordered by First_Name (ascending) and Salary (descending):-**

```
SELECT * FROM Employee ORDER BY first_name ASC, salary DESC;
```

d) **Get employee details from the employee table whose first name contains 'J':**

```
SELECT * FROM Employee WHERE first_name LIKE '%J%';
```

e)**Get department-wise maximum salary from the employee table ordered by department:-**

```
SELECT department, MAX(salary) AS "Maximum Salary"
FROM Employee
GROUP BY department
ORDER BY "Maximum Salary" ASC;
```

f) **Select first_name, incentive amount from employee and incentives table forthose employees who have incentives and incentive amount greater than 3000:-**

(P.T.O)

```
SELECT e.first_name, i.incentive_amount
FROM Employee e
JOIN IncentiveTable i ON e.employee_id = i.employee_id
WHERE i.incentive_amount > 3000;
```

g) **Create After Insert trigger on Employee table which insert records in viewtable:-**

```
-- Create the ViewTable (if not already created with the same structure)
CREATE TABLE IF NOT EXISTS ViewTable (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    hire_date DATE,
    department VARCHAR(100),
    salary DECIMAL(10, 2)
);

-- Create the AFTER INSERT trigger
DELIMITER //
CREATE TRIGGER AfterInsertEmployee
AFTER INSERT ON Employee
```

```
FOR EACH ROW
BEGIN
  INSERT INTO ViewTable (employee_id, first_name, last_name, hire_date, department, salary)
  VALUES (NEW.employee_id, NEW.first_name, NEW.last_name, NEW.hire_date, NEW.department, NEW.salary);
END;
```

- **Retrieve the below data from above table**

a) **All orders for more than $1000:**

```
SELECT *
FROM Orders
WHERE order_amount > 1000;
```

b) **Names and cities of all salespeople in London with commission above 0.12:**

```
SELECT salesperson_name, city
FROM Salespeople
WHERE city = 'London' AND commission > 0.12;
```

c) **All salespeople either in Barcelona or in London:**

```
SELECT *
FROM Salespeople
WHERE city IN ('Barcelona', 'London');
```

d) **All salespeople with commission between 0.10 and 0.12 (boundary values excluded):**

```
SELECT *
FROM Salespeople
WHERE commission > 0.10 AND commission < 0.12;
```

e) **All customers excluding those with rating <= 100 unless they are located in Rome:**

```
SELECT *
FROM Customers
WHERE (rating > 100) OR (rating <= 100 AND city = 'Rome');
```

E N D