NAME-VISHAKHA KOLHE

COURSE- PW DATA SCIENCE WITH GEN AI

Question 1 : Explain the fundamental differences between DDL, DML, and DQL commands in SQL. Provide one example for each type of command.

Sol-

1. DDL (Data Definition Language)

DDL commands are used to create, modify, or delete database structure like tables.

These commands change the schema (design) of the database.

Changes are permanent (Auto-commit).

Example: CREATE TABLE

2 . DML commands -are used to **manipulate or modify the data stored inside tables**.

These commands do not change the structure of the table, only the data inside it.

🔹 Key Characteristics:

- Used to insert, update, and delete data.
- Changes can be rolled back (using ROLLBACK).
- Affects rows (records) inside tables.

🔹 Common DML Commands:

- `INSERT`
- `UPDATE`
- `DELETE`

3️⃣ DQL (Data Query Language)

🔹 Definition:

DQL commands are used to **retrieve or fetch data from the database**.

It does not change data or structure — it only displays data.

🔹 Key Characteristics:

- Used to retrieve information.
- Does not modify table structure.
- Does not modify stored data.

🔹 Main DQL Command:

- `SELECT`

Question 2 : What is the purpose of SQL constraints? Name and describe three common types of constraints, providing a simple scenario where each would be useful.

Sol:

SQL Constraints are **rules applied to table columns** to ensure that the data stored in the database is **accurate, valid, and reliable**.

Constraints help maintain:

- ✔ Data Integrity

- ✔ Data Accuracy
- ✔ Data Consistency
- ✔ Prevention of Invalid Data Entry

Three Common Types of Constraints

1. PRIMARY KEY
2. NOT NULL
3. UNIQUE

1️⃣ PRIMARY KEY Constraint

🔹 **Definition:**

A PRIMARY KEY constraint uniquely identifies each record in a table.

- It does NOT allow duplicate values.
- It does NOT allow NULL values.
- Each table can have only ONE primary key.

🔹 **Practical Example:**

```
CREATE TABLE Students (
  student_id INT PRIMARY KEY,
  name VARCHAR(50),
  age INT
);
```

🔹 **Explanation:**

- `student_id` must be unique.
- It cannot be NULL.
- No two students can have the same ID.

NOT NULL Constraint

- 🔹 **Definition:**

The NOT NULL constraint ensures that a column cannot have a NULL (empty) value.

It forces the user to enter data in that column.

- 🔹 **Practical Example:**

```
CREATE TABLE Employees (
  emp_id INT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  salary INT
);
```

- 🔹 **Explanation:**

- The `name` column cannot be empty.
- If someone tries to insert a record without a name, MySQL will show an error.

3️⃣ UNIQUE Constraint

- 🔹 **Definition:**

The UNIQUE constraint ensures that all values in a column are different.

It prevents duplicate values but allows one NULL value (in MySQL).

- 🔹 **Practical Example:**

```
CREATE TABLE Users (
  user_id INT PRIMARY KEY,
  email VARCHAR(100) UNIQUE,
  password VARCHAR(50)
);
```

- 🔹 **Explanation:**

- No two users can have the same email.
- If duplicate email is inserted, MySQL will give an error.

Question 3 : Explain the difference between LIMIT and OFFSET clauses in SQL. How would you use them together to retrieve the third page of results, assuming each page has 10 records?

Sol-

1. LIMIT Clause

Definition

The LIMIT clause is used to restrict the number of rows returned by a SELECT query.

Example

```
SELECT * FROM Students

LIMIT 10;
```

Explanation

This query retrieves only the first 10 records from the Students table.

2. OFFSET Clause

Definition

The OFFSET clause is used to skip a specified number of rows before starting to return rows.

It tells the database:
“How many records should I skip before displaying results?”

Example

```
SELECT * FROM Students

LIMIT 10 OFFSET 10;
```

Explanation

This query:

- Skips the first 10 records
- Then displays the next 10 records

Using LIMIT and OFFSET Together for Pagination

Question: Retrieve the third page of results, assuming each page has 10 records.

Final Query for Third Page

SELECT * FROM Students

LIMIT 10 OFFSET 20;

Explanation

- LIMIT 10 → Show 10 records
- OFFSET 20 → Skip first 20 records
- Result → Records 21 to 30 (Third page)

Question 4 : What is a Common Table Expression (CTE) in SQL, and what are its main benefits? Provide a simple SQL example demonstrating its usage.

Sol-

A Common Table Expression (CTE) is a temporary result set in SQL that you can define within a query using the WITH keyword. It exists only during the execution of that query.

A CTE allows you to write a subquery once, give it a name, and then reference it like a table in the main query.

Main Benefits of CTE

1. Improves readability by breaking complex queries into simpler logical parts.
2. Makes queries easier to maintain and understand.
3. Allows reuse of the same result set multiple times within a query.
4. Helps organize multi-step calculations in a structured way.
5. Reduces the need for deeply nested subqueries.

Scenario:

Find employees whose salary is greater than the average salary.

SQL Query Using CTE

```
WITH AvgSalary AS (
  SELECT AVG(salary) AS average_salary
  FROM Employees
)
SELECT name, salary
FROM Employees, AvgSalary
WHERE Employees.salary > AvgSalary.average_salary;
```

Question 5 : Describe the concept of SQL Normalization and its primary goals. Briefly explain the first three normal forms (1NF, 2NF, 3NF).

Sol-

SQL Normalization is a process used to organize data in a database to reduce duplication and improve accuracy. Its main goal is to store data in a structured way so that each piece of information is stored only once, which helps avoid errors during inserting, updating, or deleting data. Normalization also makes the database easier to manage and maintain by dividing large tables into smaller related tables.

The First Normal Form (1NF) ensures that each column contains only one value and there are no repeating groups or multiple values in a single field. Each row should be uniquely identified by a primary key. The Second Normal Form (2NF) builds on 1NF

and removes partial dependency, meaning all non-key columns must depend on the entire primary key, not just part of it. The Third Normal Form (3NF) removes transitive dependency, meaning non-key columns should depend only on the primary key and not on other non-key columns. These normal forms help create a clean, efficient, and reliable database structure.

Question 6 : Create a database named ECommerceDB and perform the following tasks:

1. Create the following tables with appropriate data types and constraints:

● Categories ○ CategoryID (INT, PRIMARY KEY) ○ CategoryName (VARCHAR(50), NOT NULL, UNIQUE)

● Products ○ ProductID (INT, PRIMARY KEY) ○ ProductName (VARCHAR(100), NOT NULL, UNIQUE) ○ CategoryID (INT, FOREIGN KEY → Categories) ○ Price (DECIMAL(10,2), NOT NULL) ○ StockQuantity (INT)

● Customers ○ CustomerID (INT, PRIMARY KEY) ○ CustomerName (VARCHAR(100), NOT NULL) ○ Email (VARCHAR(100), UNIQUE) ○ JoinDate (DATE)

● Orders ○ OrderID (INT, PRIMARY KEY) ○ CustomerID (INT, FOREIGN KEY → Customers) ○ OrderDate (DATE, NOT NULL) ○ TotalAmount (DECIMAL(10,2))

Sol-

```
CREATE DATABASE ECommerceDB;

USE ECommerceDB;


CREATE TABLE Categories (

    CategoryID INT PRIMARY KEY,

    CategoryName VARCHAR(50) NOT NULL UNIQUE

);
```

```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100) NOT NULL UNIQUE,
    CategoryID INT,
    Price DECIMAL(10,2) NOT NULL,
    StockQuantity INT,
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    JoinDate DATE
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE NOT NULL,
    TotalAmount DECIMAL(10,2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

INSERT INTO Categories (CategoryID, CategoryName) VALUES
(1, 'Electronics'),
```

(2, 'Books'),

(3, 'Home Goods'),

(4, 'Apparel');

INSERT INTO Products (ProductID, ProductName, CategoryID, Price, StockQuantity) VALUES

(101, 'Laptop Pro', 1, 1200.00, 50),

(102, 'SQL Handbook', 2, 45.50, 200),

(103, 'Smart Speaker', 1, 99.99, 150),

(104, 'Coffee Maker', 3, 75.00, 80),

(105, 'Novel: The Great SQL', 2, 25.00, 120),

(106, 'Wireless Earbuds', 1, 150.00, 100),

(107, 'Blender X', 3, 120.00, 60),

(108, 'T-Shirt Casual', 4, 20.00, 300);

INSERT INTO Customers (CustomerID, CustomerName, Email, JoinDate) VALUES

(1, 'Alice Wonderland', 'alice@example.com', '2023-01-10'),

(2, 'Bob the Builder', 'bob@example.com', '2022-11-25'),

(3, 'Charlie Chaplin', 'charlie@example.com', '2023-03-01'),

(4, 'Diana Prince', 'diana@example.com', '2021-04-26');

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount) VALUES

```
(1001, 1, '2023-04-26', 1245.50),

(1002, 2, '2023-10-12', 99.99),

(1003, 1, '2023-07-01', 145.00),

(1004, 3, '2023-01-14', 150.00),

(1005, 2, '2023-09-24', 120.00),

(1006, 1, '2023-06-19', 20.00);
```

```
SELECT * FROM Categories;

SELECT * FROM Products;

SELECT * FROM Customers;

SELECT * FROM Orders;
```

Question 7 : Generate a report showing CustomerName, Email, and the TotalNumberofOrders for each customer. Include customers who have not placed any orders, in which case their TotalNumberofOrders should be 0. Order the results by CustomerName

Sol-

```
SELECT

    Customers.CustomerName,

    Customers.Email,

    COUNT(Orders.OrderID) AS TotalNumberOfOrders

FROM Customers
```

```
LEFT JOIN Orders

ON Customers.CustomerID = Orders.CustomerID

GROUP BY Customers.CustomerID, Customers.CustomerName, Customers.Email

ORDER BY Customers.CustomerName;
```

Question 8 : Retrieve Product Information with Category: Write a SQL query to display the ProductName, Price, StockQuantity, and CategoryName for all products. Order the results by CategoryName and then ProductName alphabetically.

```
SELECT

    Products.ProductName,

    Products.Price,

    Products.StockQuantity,

    Categories.CategoryName

FROM Products

INNER JOIN Categories

ON Products.CategoryID = Categories.CategoryID

ORDER BY Categories.CategoryName, Products.ProductName;
```

Question 9 : Write a SQL query that uses a Common Table Expression (CTE) and a Window Function (specifically ROW_NUMBER() or RANK()) to display the CategoryName, ProductName, and Price for the top 2 most expensive products in each CategoryName

Sol-

```
WITH RankedProducts AS (
    SELECT
        Categories.CategoryName,
        Products.ProductName,
        Products.Price,
        ROW_NUMBER() OVER (
            PARTITION BY Categories.CategoryID
            ORDER BY Products.Price DESC
        ) AS PriceRank
    FROM Products
    INNER JOIN Categories
    ON Products.CategoryID = Categories.CategoryID
)

SELECT
    CategoryName,
    ProductName,
    Price
FROM RankedProducts
```

```
WHERE PriceRank <= 2

ORDER BY CategoryName, Price DESC;
```

Question 10 : You are hired as a data analyst by Sakila Video Rentals, a global movie rental company. The management team is looking to improve decision-making by analyzing existing customer, rental, and inventory data. Using the Sakila database, answer the following business questions to support key strategic initiatives. Tasks & Questions:

1. Identify the top 5 customers based on the total amount they've spent. Include customer name, email, and total amount spent.

```
CREATE DATABASE SakilaLite;

USE SakilaLite;

CREATE TABLE Customers (

    customer_id INT PRIMARY KEY,

    first_name VARCHAR(50),

    last_name VARCHAR(50),

    email VARCHAR(100)

);


CREATE TABLE Stores (

    store_id INT PRIMARY KEY,

    store_name VARCHAR(50)

);

CREATE TABLE Films (

    film_id INT PRIMARY KEY,

    title VARCHAR(100),

    category VARCHAR(50)
```

```
);
CREATE TABLE Inventory (
    inventory_id INT PRIMARY KEY,
    film_id INT,
    store_id INT,
    FOREIGN KEY (film_id) REFERENCES Films(film_id),
    FOREIGN KEY (store_id) REFERENCES Stores(store_id)
);
CREATE TABLE Rentals (
    rental_id INT PRIMARY KEY,
    inventory_id INT,
    customer_id INT,
    rental_date DATE,
    return_date DATE,
    amount DECIMAL(8,2),
    FOREIGN KEY (inventory_id) REFERENCES Inventory(inventory_id),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
INSERT INTO Customers VALUES
(1,'Alice','Sharma','alice@mail.com'),
(2,'Bob','Patel','bob@mail.com'),
(3,'Charlie','Mehta','charlie@mail.com'),
(4,'Diana','Kapoor','diana@mail.com'),
(5,'Ethan','Rao','ethan@mail.com');
INSERT INTO Stores VALUES
```

```
(1,'Store A'),
(2,'Store B');
INSERT INTO Films VALUES
(101,'Avengers','Action'),
(102,'Inception','Sci-Fi'),
(103,'Frozen','Animation'),
(104,'Joker','Drama'),
(105,'Toy Story','Animation'),
(106,'Batman','Action');
INSERT INTO Inventory VALUES
(1,101,1),
(2,102,1),
(3,103,1),
(4,104,2),
(5,105,2),
(6,106,2);
INSERT INTO Rentals VALUES
(1,1,1,'2023-01-10','2023-01-12',5.99),
(2,2,2,'2023-02-05','2023-02-07',4.99),
(3,3,1,'2023-03-01','2023-03-03',3.99),
(4,4,3,'2023-04-15','2023-04-18',6.99),
(5,5,1,'2023-05-10','2023-05-12',4.99),
(6,6,2,'2023-06-20','2023-06-22',5.99),
(7,1,1,'2023-07-11','2023-07-13',5.99),
(8,2,1,'2023-08-01','2023-08-03',4.99),
```

```
(9,3,4,'2023-09-09','2023-09-10',3.99),

(10,4,1,'2023-10-02','2023-10-05',6.99);

SELECT

    CONCAT(first_name,' ',last_name) AS CustomerName,

    email,

    SUM(amount) AS TotalSpent

FROM Customers

JOIN Rentals USING(customer_id)

GROUP BY customer_id

ORDER BY TotalSpent DESC

LIMIT 5;
```

2. Which 3 movie categories have the highest rental counts? Display the category name and number of times movies from that category were rented.

```
SELECT

    Films.category,

    COUNT(*) AS RentalCount

FROM Rentals

JOIN Inventory USING(inventory_id)

JOIN Films USING(film_id)

GROUP BY Films.category

ORDER BY RentalCount DESC

LIMIT 3;
```

3. Calculate how many films are available at each store and how many of those have never been rented.

```
SELECT
    store_id,
    COUNT(*) AS TotalFilms,
        SUM(CASE WHEN Rentals.rental_id IS NULL THEN 1 ELSE 0 END) AS NeverRented
FROM Inventory
LEFT JOIN Rentals USING(inventory_id)
GROUP BY store_id;
```

4. Show the total revenue per month for the year 2023 to analyze business seasonality.

```
SELECT
    MONTH(rental_date) AS Month,
    SUM(amount) AS MonthlyRevenue
FROM Rentals
WHERE YEAR(rental_date) = 2023
GROUP BY MONTH(rental_date)
ORDER BY Month;
```

5. Identify customers who have rented more than 10 times in the last 6 months.

Sol-

```
SELECT
    customer_id,
    COUNT(*) AS RentalCount
```

```
FROM Rentals

WHERE rental_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)

GROUP BY customer_id

HAVING COUNT(*) > 10;
```