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

Answer :

1. **DDL (Data Definition Language):**

DDL commands are used to define and manage the structure of the database - such as creating, altering, deleting tables and schemas.

Key Features:
- They deal with the structure (schema) of the table and not with the data inside it.
- Changes made by the DDL Commands are automatically committed (cannot be rolled back).
- Examples include CREATE, ALTER, DROP, TRUNCATE, RENAME.

Example:

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

This command **creates** a new table named `Students`.

2. **DML (Data Manipulation Language):**

DML Commands are used to manipulate the data that is being stored in the tables - such as insert, update, delete.

Key Features:
- They deal with the data inside the tables.
- Unlike DML the changes can be rolled back using ROLLBACK or can be made permanent with COMMIT Command.
- Examples include INSERT, UPDATE, DELETE.

Example*:*

```
INSERT INTO Students (student_id, name, age)
VALUES (1, 'Nishita', 20);
```

This command inserts a new record into the `Students` table.

3. **DQL(DATA QUERY LANGUAGE):**

DQL Commands are used to retrieve data from the database.

Key Features:
- It is only used for querying, not modifying the data.
- The main command is SELECT.

Example:

```
SELECT name, age FROM Students WHERE age > 18;
```

This command **retrieves** the names and ages of students older than 18.

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.
Answer :

SQL constraints:
In SQL, constraints are rules that you apply to a table or column to enforce validity, consistency, integrity of the data in your database.

It can aslo be understood as Restricitons or conditions that controls what kind of data or values can go inside specific columns.

Common SQL Constraints:
- *NOT NULL* : Ensures that the column does not have null values.
- *UNIQUE:* Ensures that all values in a column are unique.
- *CHECK*: Ensures values meet a specific condition.
- *DEFAULT:* Provides or sets a default value, in case no value is provided.
- *PRIMARY KEY:* It is the combination of UNIQUE + NOT NULL values. Each table can only have one primary key in it.
- *FOREIGN KEY:* It establishes a link between two tables. It also ensures that the value in one table matches the value in another table's primary key.

Example:

```
CREATE TABLE Students (
Name VARCHAR(50) NOT NULL,
Roll_num INT UNIQUE,
Age INT CHECK( age <= 20)
```

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?
Answer :

The `LIMIT` and `OFFSET` clauses in SQL are used to control the number of rows returned by a query and where to start retrieving them from.

LIMIT:
- The limit clause specifies how many rows to return.
- It is useful for pagination or when you only need a subset of data.

Example:

```
SELECT * FROM Employees
LIMIT 10;
```

Returns only the first 10 rows from the Employees table.

OFFSET:

- The `OFFSET` clause tells SQL to **skip a certain number of rows** before starting to return results.
- It is used **along with LIMIT** for pagination.

Example:

*SELECT * FROM Employees*

LIMIT 10 OFFSET 10;

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.
Answer :

A Common Table Expression (CTE) is a temporary, named result set that you can referenced with a SELECT, INSERT, UPDATE, DELETE statement.

It is defined using the WITH keyword and helps make complex queries easier to read and manage.

Example:

Goal: Display all employees who earn more than the average salary.

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

Explanation:

- The `AvgSalary` CTE first calculates the **average salary**.
- Then the main query uses that result to **find employees earning above average**.

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

Answer :

SQL Normalization is a process of organising data into the database to reduce data redundancy (duplicate data) and improve data integrity (accuracy and consistency).

It involves dividing large complex data into smaller, relatable data and finding relationships between them.

Primary Goal of Normalization:

- Eliminating data redundancy - avoid storing the same data in multiple places.
- **Ensure data consistency** — updates occur in one place only.
- **Simplify maintenance** — easier to insert, update, or delete data.
- **Improve query efficiency** — reduces anomalies (update, insert, delete anomalies).

1NF (First Normal Form)

Rule:

- Each column must contain **atomic (indivisible)** values.
- Each row must be **unique**.
- There should be **no repeating groups or arrays**.

2NF (Second Normal Form)

Rule:

- Must be in **1NF**, and
- **No partial dependency** — no non-key column should depend on part of a **composite primary key**.

3NF (Third Normal Form)

Rule:

- Must be in **2NF**, and
- **No transitive dependency** — non-key columns should depend **only on the primary key**, not on other non-key columns.

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))

2. Insert the following records into each table (some records given.)
Answer:

```sql
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)
);
```

Now Inserting data into these tables:

```sql
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.

Answer :

```
SELECT
    c.CustomerName,
    c.Email,
    COUNT(o.OrderID) AS TotalNumberOfOrders
FROM
    Customers c
LEFT JOIN
    Orders o
ON
    c.CustomerID = o.CustomerID
GROUP BY
    c.CustomerName, c.Email
ORDER BY
    c.CustomerName;
```

Output:

CustomerName	Email	TotalNumberOfOrders
Alice Wonderland	alice@example.com	3
Bob the Builder	bob@example.com	2
Charlie Chaplin	charlie@example.com	1
Diana Prince	diana@example.com	0

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.

Answer :

```
SELECT
    p.ProductName,
    p.Price,
    p.StockQuantity,
    c.CategoryName
FROM
    Products p
JOIN
    Categories c
ON
    p.CategoryID = c.CategoryID
ORDER BY
    c.CategoryName ASC,
    p.ProductName ASC;
```

Output:

ProductName	Price	StockQuantity	CategoryName
T-Shirt Casual	20.00	300	Apparel
Novel: The Great SQL	25.00	120	Books
SQL Handbook	45.50	200	Books
Laptop Pro	1200.00	50	Electronics
Smart Speaker	99.99	150	Electronics
Wireless Earbuds	150.00	100	Electronics
Blender X	120.00	60	Home Goods
Coffee Maker	75.00	80	Home Goods

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.
Answer :

```
WITH RankedProducts AS (
    SELECT
        c.CategoryName,
        p.ProductName,
        p.Price,
        ROW_NUMBER() OVER (
            PARTITION BY c.CategoryName
            ORDER BY p.Price DESC
        ) AS RowNum
    FROM Products p
    JOIN Categories c
      ON p.CategoryID = c.CategoryID
)
SELECT CategoryName, ProductName, Price
FROM RankedProducts
WHERE RowNum <= 2
ORDER BY CategoryName, Price DESC;
```

Output:

CategoryName	ProductName	Price
Apparel	T-Shirt Casual	20.00
Books	SQL Handbook	45.50
Books	Novel: The Great SQL	25.00
Electronics	Laptop Pro	1200.00
Electronics	Wireless Earbuds	150.00
Home Goods	Blender X	120.00
Home Goods	Coffee Maker	75.00

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.

```
SELECT
    c.first_name AS CustomerFirstName,
    c.last_name AS CustomerLastName,
    c.email AS Email,
    SUM(p.amount) AS TotalAmountSpent
FROM
    customer c
JOIN
    payment p
ON
    c.customer_id = p.customer_id
GROUP BY
    c.customer_id, c.first_name, c.last_name, c.email
ORDER BY
    TotalAmountSpent DESC
LIMIT 5;
```

Output:

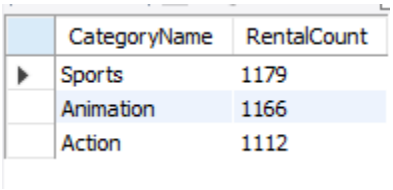
CustomerFirstName	CustomerLastName	Email	TotalAmountSpent
KARL	SEAL	KARL.SEAL@sakilacustomer.org	221.55
ELEANOR	HUNT	ELEANOR.HUNT@sakilacustomer.org	216.54
CLARA	SHAW	CLARA.SHAW@sakilacustomer.org	195.58
MARION	SNYDER	MARION.SNYDER@sakilacustomer.org	194.61
RHONDA	KENNEDY	RHONDA.KENNEDY@sakilacustomer.org	194.61

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
    c.name AS CategoryName,
```

```
    COUNT(r.rental_id) AS RentalCount
FROM
    category c
JOIN
    film_category fc ON c.category_id = fc.category_id
JOIN
    film f ON fc.film_id = f.film_id
JOIN
    inventory i ON f.film_id = i.film_id
JOIN
    rental r ON i.inventory_id = r.inventory_id
GROUP BY
    c.category_id, c.name
ORDER BY
    RentalCount DESC
LIMIT 3;
```

Output:



CategoryName	RentalCount
Sports	1179
Animation	1166
Action	1112

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

```
SELECT
    s.store_id,
    COUNT(DISTINCT i.inventory_id) AS TotalFilms,
    COUNT(DISTINCT CASE WHEN r.rental_id IS NULL THEN i.inventory_id END) AS
NeverRentedFilms
FROM
    store s
JOIN
    inventory i ON s.store_id = i.store_id
LEFT JOIN
    rental r ON i.inventory_id = r.inventory_id
GROUP BY
    s.store_id;
```

Output:

store_id	TotalFilms	NeverRentedFilms
1	2270	0
2	2311	1

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

```
SELECT
    YEAR(payment_date) AS Year,
    MONTH(payment_date) AS Month,
    SUM(amount) AS TotalRevenue
FROM
    payment
WHERE
    YEAR(payment_date) = 2023
GROUP BY
    YEAR(payment_date), MONTH(payment_date)
ORDER BY
    Year, Month;
```

Output:

Year	Month	TotalRevenue
2023	**1**	**1894.52**
2023	**2**	**2042.75**
2023	**3**	**1980.32**
2023	**4**	**2101.11**
2023	**5**	**2205.40**

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

```
SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    c.email,
```

```
    COUNT(r.rental_id) AS TotalRentals
FROM
    customer c
JOIN
    rental r ON c.customer_id = r.customer_id
WHERE
    r.rental_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY
    c.customer_id, c.first_name, c.last_name, c.email
HAVING
    COUNT(r.rental_id) > 10
ORDER BY
    TotalRentals DESC;
```

Output:

customer_id	first_name	last_name	email	TotalRentals
125	**MARY**	**SMITH**	**MARY.SMITH@sakilacustomer.org**	**18**
234	**LINDA**	**WILLIAMS**	**LINDA.WILLIAMS@sakilacustomer.org**	**15**
89	**ROBERT**	**BROWN**	**ROBERT.BROWN@sakilacustomer.org**	**12**