Introduction to SQL and Advanced Functions

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

SQL commands are divided into different categories based on their purpose.

a. DDL (Data Definition Language):

DDL commands are used to define or modify the structure of database objects like tables, schemas, or indexes.They affect the database structure andChanges are auto-committed.
Example:

```
CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   Name VARCHAR(50)
);
```

b. DML (Data Manipulation Language):

DML commands are used to manipulate data inside tables.Used for inserting, updating, deleting records and Changes can be rolled back.
Example:

```
INSERT INTO Students (StudentID, Name)
VALUES (1, 'Dimple');
```

c. DQL (Data Query Language):

DQL is used to retrieve data from the database mainly uses SELECT statement.
Example:

```
SELECT * FROM Students;
```

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.

SQL constraints are rules applied to table columns to ensure data accuracy and integrity.They:

- Prevent invalid data entry
- Maintain consistency

- Enforce business rules

Three Common Constraints:

1. PRIMARY KEY

- Uniquely identifies each record.
- Cannot be NULL.

Example: CustomerID in the Customers table ensures each customer is unique.

2. FOREIGN KEY

- Links one table to another.
- Maintains referential integrity.

Example: CategoryID in Products table references Categories table.

3. UNIQUE

- Ensures all values in a column are different.

  Example: Email in Customers table should be unique.

---

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?

- LIMIT specifies the number of records to return.
- OFFSET specifies how many records to skip.

Example: To get the third page with 10 records per page:

OFFSET = (PageNumber - 1) × RecordsPerPage

For page 3: OFFSET = (3 - 1) × 10 = 20

```
SELECT * FROM Products

LIMIT 10 OFFSET 20;
```

---

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

A Common Table Expression (CTE) is a temporary result set defined using the WITH keyword that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement.

Benefits:

- Improves readability
- Breaks complex queries into smaller parts
- Can be recursive

Example:

```
WITH ExpensiveProducts AS (

   SELECT ProductName, Price

   FROM Products

   WHERE Price > 100

)

SELECT * FROM ExpensiveProducts
```

—---

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

Normalization is the process of organizing data to reduce redundancy and improve data integrity.

Goals:

- Eliminate duplicate data
- Ensure logical data storage
- Maintain data consistency

1. First Normal Form (1NF)

- No repeating groups
- Each column contains atomic values

2. Second Normal Form (2NF)

- Must be in 1NF
- No partial dependency on composite primary key

3. Third Normal Form (3NF)

- Must be in 2NF
- No transitive dependency

SQL Implementation

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

```
CREATE DATABASE ECommerceDB;

USE ECommerceDB;
```

Create Tables

```
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 Data

Categories

```
INSERT INTO Categories VALUES
```

```sql
(1, 'Electronics'),
(2, 'Books'),
(3, 'Home Goods'),
(4, 'Apparel');
```

Products

```sql
INSERT INTO Products 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);
```

Customers

```sql
INSERT INTO Customers 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');
```

Orders

```
INSERT INTO Orders 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);
```

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.

```
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.CustomerID, c.CustomerName, c.Email

ORDER BY c.CustomerName;
```

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

---

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

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

SELECT CategoryName, ProductName, Price

FROM RankedProducts

WHERE rn <= 2;
```

Question10: 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:

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

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

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

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

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

a. Top 5 Customers by Spending

```
SELECT

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

   c.email,

   SUM(p.amount) AS TotalSpent

FROM customer c

JOIN payment p ON c.customer_id = p.customer_id

GROUP BY c.customer_id
```

```
ORDER BY TotalSpent DESC

LIMIT 5;
```

---

b. Top 3 Categories by Rental Count

```
SELECT

   cat.name AS CategoryName,

   COUNT(r.rental_id) AS RentalCount

FROM category cat

JOIN film_category fc ON cat.category_id = fc.category_id

JOIN inventory i ON fc.film_id = i.film_id

JOIN rental r ON i.inventory_id = r.inventory_id

GROUP BY cat.name

ORDER BY RentalCount DESC

LIMIT 3;
```

---

c. Films Available & Never Rented

```
SELECT

   s.store_id,

   COUNT(i.inventory_id) AS TotalFilms,

   SUM(CASE WHEN r.rental_id IS NULL THEN 1 ELSE 0 END) AS NeverRented

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;
```

---

d. Monthly Revenue for 2023

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

---

e. Customers with More Than 10 Rentals in Last 6 Months

```
SELECT
    c.customer_id,
    CONCAT(c.first_name, ' ', c.last_name) AS CustomerName,
    COUNT(r.rental_id) AS RentalCount
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
HAVING COUNT(r.rental_id) > 10;
```