Introduction to SQL and Advanced Functions | Assignment

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

Answer :

DDL (Data Definition Language)

- Used to define, modify, or delete database structures
- Affects the schema, not the data
- Changes are usually permanent

Example:

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

DML (Data Manipulation Language)

- Used to insert, update, or delete data in tables
- Works on existing records

Example

```
INSERT INTO Students VALUES (1, 'Alice');
```

DQL (Data Query Language)

- Used to **retrieve data** from the database
- Does not modify data

Example

```
SELECT * FROM Students;
```

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 ensure data accuracy, integrity, and reliability by enforcing rules on table columns.

Common Types of Constraints

1. PRIMARY KEY

- Uniquely identifies each record
- Cannot be NULL

Scenario: Each customer must have a unique ID.

```
CustomerID INT PRIMARY KEY
```

2. NOT NULL

- Prevents NULL values

Scenario: Customer name must always be provided.

```
CustomerName VARCHAR(100) NOT NULL
```

3. UNIQUE

- Ensures all values are different

Scenario: Email address must be unique for each customer.

```
Email VARCHAR(100) UNIQUE
```

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

Answer :

1. **LIMIT:** Specifies the maximum number of records to return.
2. **OFFSET:** Specifies how many records to skip before starting to return rows.

To retrieve the third page of results with 10 records per page, you must skip the first 20 records (Pages 1 and 2):

```
SELECT * FROM table_name
LIMIT 10 OFFSET 20;
```

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 CTE is a temporary named result set defined using the WITH keyword.

Benefits

- Improves readability
- Simplifies complex queries
- Enables reuse of logic
- Supports recursion

```
WITH ExpensiveProducts AS (
   SELECT ProductName, Price
   FROM Products
   WHERE Price > 100
)
SELECT * FROM ExpensiveProducts;
```

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

Answer :

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

1. **1NF (First Normal Form):** Data must be atomic (one value per cell) and each record must be unique.

- Atomic values (no multiple values in one column)
- No repeating groups

2. **2NF (Second Normal Form):** Must be in 1NF, and all non-key attributes must be fully dependent on the primary key.

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

3. **3NF (Third Normal Form):** Must be in 2NF, and no non-key attribute should depend on another non-key attribute (no transitive dependencies).

- Must be in 2NF

- No transitive dependency

Question 6 : Create a database named ECommerceDBand 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**
    - **Categories**

CategoryID	Category Name
1	Electronics
2	Books
3	Home Goods
4	Apparel

    - **Products**

ProductID	ProductName	CategoryID	Price	StockQuantity
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

CustomerID	CustomerName	Email	Joining Date
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**

OrderID	CustomerID	OrderDate	TotalAmount

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**

Answer :

1. Create Database

```
CREATE DATABASE ECommerceDB;
USE ECommerceDB;
```

2. 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)
);
```

3. Insert Data

Categories

```
INSERT INTO Categories VALUES
(1, 'Electronics'),
(2, 'Books'),
(3, 'Home Goods'),
(4, 'Apparel');
```

Products

```
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

```
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);
```

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);
```

Question 7 : Generate a report showing CustomerName, Email, and the TotalNumberofOrdersfor each customer. Include customers who have not placed any orders, in which case their TotalNumberofOrdersshould 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;
```

Question 8 : Retrieve Product Information with Category: Write a SQL query to display the ProductName, Price, StockQuantity, and CategoryNamefor all products. Order the results by CategoryNameand 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, p.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 Pricefor 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 rn
    FROM Products p
    JOIN Categories c ON p.CategoryID = c.CategoryID
)
SELECT CategoryName, ProductName, Price
FROM RankedProducts
WHERE rn <= 2;
```

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.
2. Which **3 movie categories** have the **highest rental counts**? Display the category name and number of times movies from that category were rented.
3. Calculate how many films are available at each store and how many of those have

never been rented.

4. Show the **total revenue per month** for the year 2023 to analyze business seasonality.
5. Identify customers who have rented **more than 10 times** in the last 6 months.

Answer:

1. Top 5 Customers by Total 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;
```

2. Top 3 Movie 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;
```

3. Films Available and Never Rented per Store

```
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;
```

4. 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;
```

5. Customers Renting More Than 10 Times 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 RentalCount > 10;
```