Assignment Code: DA-AG-014

Introduction to SQL and Advanced Functions | Assignment

Instructions:
Carefully read each question before attempting. Use Google Docs, Microsoft Word, or a similar tool to type out each theoretical question along with its answer. For practical questions, use **SQL Workbench** (or your designated SQL tool) to complete the required tasks. Once you have finished, save both the Word/Docs file and SQL File as PDF documents. Please ensure that you do not zip or archive the files before uploading. Submit the PDF files directly to the LMS or as instructed by your teacher. Each question carries 20 marks.

Total Marks: 200

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 commands are used to **define or modify the structure of database objects** like tables, databases, indexes, etc.

```
CREATE TABLE Students (
    ID INT,
    Name VARCHAR(50),
    Age INT
);
```

2 :- DML commands are used to **insert, update, and delete data inside tables**.

```
INSERT INTO Students VALUES (1, 'Akshit', 20);
```

3 :- DQL commands are used to **retrieve data from the database**.

```
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 are used to enforce rules on table data to maintain data integrity and consistency.
Common constraints include PRIMARY KEY (ensures unique identification), NOT NULL (prevents null values), and UNIQUE (ensures all values are different). These constraints help avoid invalid and duplicate data in the database

---

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 : LIMIT specifies the maximum number of rows returned by a query, while OFFSET specifies how many rows to skip before starting to return rows. To retrieve the third page with 10 records per page, we use 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 Common Table Expression (CTE) is a temporary named result set defined using the WITH clause that can be referenced within a SQL query. It improves query readability, simplifies complex queries, supports recursive queries, and allows reuse of query logic.

Get all students with age greater than 18 using a CTE.

```
WITH AdultStudents AS (
    SELECT ID, Name, Age
    FROM Students
    WHERE Age > 18
)
SELECT * FROM AdultStudents;
```

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

The primary goals are to eliminate duplicate data and anomalies.

1NF requires atomic values and no repeating groups.
2NF requires no partial dependency on a composite key.
3NF requires no transitive dependency among non-key attributes.

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**
    - **Categories**

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

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 Products

- 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 : 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 VALUES
(1, 'Electronics'),
(2, 'Books'),
(3, 'Home Goods'),
(4, 'Apparel');

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);

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');

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);

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 Customers.CustomerName,
Customers.Email,
COUNT(Orders.OrderID) AS TotalNumberofOrders
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
GROUP BY 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.

Answer :

```
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`.

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.

1) Top 5 customers by total amount spent

```
SELECT
    c.customer_id,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    c.email,
    SUM(p.amount) AS total_spent
FROM customer c
JOIN payment p ON c.customer_id = p.customer_id
GROUP BY c.customer_id, customer_name, c.email
ORDER BY total_spent DESC
LIMIT 5;
```

2) Top 3 movie categories by rental count

```
SELECT
    cat.name AS category_name,
    COUNT(r.rental_id) AS rental_count
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 rental_count DESC
LIMIT 3;
```

3) Films available at each store + never rented films

(A) Total films available per store

```
SELECT
    s.store_id,
    COUNT(i.inventory_id) AS total_films
FROM store s
JOIN inventory i ON s.store_id = i.store_id
GROUP BY s.store_id;
```

(B) Films never rented per store

```
SELECT
    s.store_id,
    COUNT(i.inventory_id) AS never_rented_films
FROM store s
JOIN inventory i ON s.store_id = i.store_id
LEFT JOIN rental r ON i.inventory_id = r.inventory_id
WHERE r.rental_id IS NULL
GROUP BY s.store_id;
```

4) Total revenue per month for the year 2023

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

5) Customers who rented more than 10 times in the last 6 months

```
SELECT

    c.customer_id,
```

```
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    COUNT(r.rental_id) AS rental_count
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, customer_name
HAVING rental_count > 10
ORDER BY rental_count DESC;
```