Quiz

Task 1: Employee Management Database Operations

Imagine you are tasked with managing an employee database for a company. Below are several tasks involving SQL commands to manipulate the database.

1. **Table Creation with Constraints**: Create a table named **EmployeeRecords** with the following columns:
    - **EmployeeID** (INTEGER, PRIMARY KEY)
    - **FirstName** (VARCHAR(50))
    - **LastName** (VARCHAR(50))
    - **Department** (VARCHAR(100))
    - **Salary** (DECIMAL(10,2), CHECK (Salary >= 0))
2. **Data Insertion with Transactions**: Insert the following data into the **EmployeeRecords** table within a single transaction:
    - EmployeeID: 101, FirstName: "John", LastName: "Doe", Department: "Finance", Salary: 50000.00
    - EmployeeID: 102, FirstName: "Jane", LastName: "Smith", Department: "HR", Salary: 60000.00
    - EmployeeID: 103, FirstName: "Alice", LastName: "Johnson", Department: "IT", Salary: 70000.00
    - Your own data: EmployeeID: X, FirstName: "YourFirstName", LastName: "YourLastName", Department: "YourDepartment", Salary: YourSalary
3. **Adding and Deleting Columns with Constraints**: Perform the following operations:
    - Add a new column named **HireDate** of type DATE to the **EmployeeRecords** table.
    - Add a NOT NULL constraint to the **HireDate** column.
    - Delete the newly added **HireDate** column from the table.
4. **Employees with Longest Last Names**: Write a query to retrieve the names of employees with the longest last names in the database.

5. **Average Salary by Department**: Calculate the average salary for each department and display the results along with the department names.
6. **Salary Increment**: Update the salary of all employees in the IT department by 10%.
7. **Employee Count by Department**: Retrieve the count of employees in each department and display the results along with the department names.

Task 2: Online Retail Database Operations

Suppose you are managing a database for an online retail company. Below are several tasks involving SQL commands to manipulate the database.

Tables:

1. **Customers**:
    - **CustomerID** (INT, PRIMARY KEY)
    - **FirstName** (VARCHAR(50))
    - **LastName** (VARCHAR(50))
    - **Email** (VARCHAR(100))
2. **Orders**:
    - **OrderID** (INT, PRIMARY KEY)
    - **CustomerID** (INT, FOREIGN KEY referencing Customers(CustomerID))
    - **OrderDate** (DATE)
    - **TotalAmount** (DECIMAL(10,2))
3. **OrderItems**:
    - **OrderItemID** (INT, PRIMARY KEY)
    - **OrderID** (INT, FOREIGN KEY referencing Orders(OrderID))
    - **ProductID** (INT)
    - **Quantity** (INT)
    - **UnitPrice** (DECIMAL(10,2))
4. **Products**:

- **ProductID** (INT, PRIMARY KEY)
- **ProductName** (VARCHAR(100))
- **Category** (VARCHAR(50))
- **UnitPrice** (DECIMAL(10,2))

Sample Data:

1. **Customers**:

```
INSERT INTO Customers (CustomerID, FirstName, LastName, Email) VALUES (1, 'Ali', 'Khan', 'ali.khan@example.com'), (2, 'Fatima', 'Ahmed', 'fatima.ahmed@example.com'), (3, 'Sana', 'Hussain', 'sana.hussain@example.com'), (4, 'Imran', 'Malik', 'imran.malik@example.com');
```

2. **Orders**:

```
INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount) VALUES (101, 1, '2024-03-15', 150.00), (102, 2, '2024-03-16', 200.00), (103, 3, '2024-03-17', 300.00), (104, 4, '2024-03-18', 250.00);
```

3. **OrderItems**:

```
INSERT INTO OrderItems (OrderItemID, OrderID, ProductID, Quantity, UnitPrice) VALUES (1001, 101, 1, 2, 25.00), (1002, 102, 2, 3, 30.00), (1003, 103, 3, 1, 50.00), (1004, 104, 1, 2, 25.00);
```

4. **Products**:

```
INSERT INTO Products (ProductID, ProductName, Category, UnitPrice) VALUES (1, 'Product A', 'Category 1', 25.00), (2, 'Product B', 'Category 2', 30.00), (3, 'Product C', 'Category 1', 50.00), (4, 'Product D', 'Category 3', 20.00);
```

Tasks:

1. **Retrieve Customer Orders**: Write a query to retrieve the first and last names of customers along with the order IDs and total amounts for each order they placed.
    - Using Join
2. **Most Expensive Order**: Find the order ID, total amount, and order date for the most expensive order placed.
    - Using Subquery
3. **Product Categories**: List all distinct product categories available in the database.
    - No Join or Subquery Required

4. **Average Order Value**: Calculate the average order value (total amount) for all orders placed.
    - No Join or Subquery Required
5. **Order Details**: Retrieve the order date, product name, quantity, and unit price for each order item.
    - Using Join
6. **Customer Order Count**: Count the number of orders placed by each customer and display the results along with the customer's first and last names.
    - Using Join
7. **Total Revenue by Product Category**: Calculate the total revenue generated for each product category.
8. **Recent Orders**: Retrieve the order IDs, order dates, and total amounts for the 5 most recent orders placed.