Week 2 SQL Weekend Task: Building a Simple E-Commerce Database

Objective:
In this weekend task, you will apply the concepts learned in the SQL course to build a simple e-commerce database. This task will help you consolidate your understanding of SQL syntax, data types, functions, and transaction control. You will create a dynamic database where users can add, view, update, and delete information about products, orders, and customers. This task covers essential SQL topics, including string functions, numeric functions, date functions, aggregate functions, and transaction management.

Task Overview:

You will be required to:

1. Set Up the Database Environment:

- Install MySQL Server and MySQL Workbench or any SQL client you prefer.
- Ensure that the MySQL environment is correctly installed and configured.

2. Create the E-Commerce Database Structure:

- Create a new database named `e_commerce`.
- Create the following tables:
  - **Customers**: For storing customer information with columns `customer_id`, `first_name`, `last_name`, `email`.
  - **Products**: For storing product information with columns `product_id`, `product_name`, `price`, `stock_quantity`.
  - **Orders**: For storing order details with columns `order_id`, `customer_id`, `order_date`, `total_amount`.
  - **Order_Details**: For storing order item details with columns `order_detail_id`, `order_id`, `product_id`, `quantity`, `unit_price`.

3. Insert Initial Data into the Database:

- Insert at least 5 records into each of the following tables:
  - **Customers**: Ensure you use a variety of first and last names with different capitalization.
  - **Products**: Add product names, prices, and stock levels.
  - **Orders**: Add sample orders placed by different customers.
  - **Order_Details**: Link products to orders, specifying the quantity of each product ordered.

4. String Manipulation with MySQL Functions:

- Use the `concat()` function to concatenate customers' first and last names in the format "First Last".
- Use the `trim()` function to clean up any extra spaces in the customer names.
- Write a query that combines the `concat()` and `trim()` functions to return a list of customers with their properly formatted names.

Expected Output:

```
+-------------------+
| full_name         |
+-------------------+
| John Doe          |
| Jane Smith        |
| Alice Johnson     |
+-------------------+
```

5. Numeric Functions for Product Pricing:

- Calculate a 10% discount on all products using the `abs()` and `mod()` functions.
- Write a query to calculate the discounted price of each product.
- Use the `truncate()` function to ensure that the discounted price is rounded to two decimal places.

Expected Output:

```
+--------------------+-------------------+--------------------+
| product_name       | original_price    | discounted_price   |
+--------------------+-------------------+--------------------+
| Laptop             | 1000.00           | 900.00             |
| Smartphone         | 500.00            | 450.00             |
+--------------------+-------------------+--------------------+
```

6. Date Functions for Order Management:

- Use the `curdate()` and `now()` functions to generate a report of all orders placed in the current month.
- Write a query to calculate the difference in days between `order_date` and the current date, showing how long ago each order was placed.

Expected Output:

```
+-----------+------------+--------------------+
| order_id  | order_date | days_since_order   |
+-----------+------------+--------------------+
| 1         | 2024-09-01 | 7                  |
| 2         | 2024-09-03 | 5                  |
+-----------+------------+--------------------+
```

7. Aggregate Functions for Order Statistics:

- Use the `sum()` and `avg()` functions to calculate the total and average order amounts.

- Write a query to calculate the total number of products ordered for each order using the `count()` function.

Expected Output:

Week 2 SQL Weekend Task: Building a Simple E-Commerce Database

Objective:
In this weekend task, you will apply the concepts learned in the SQL course to build a simple e-commerce database. This task will help you consolidate your understanding of SQL syntax, data types, functions, and transaction control. You will create a dynamic database where users can add, view, update, and delete information about products, orders, and customers. This task covers essential SQL topics, including string functions, numeric functions, date functions, aggregate functions, and transaction management.

Task Overview:

You will be required to:

1. Set Up the Database Environment:

- Install MySQL Server and MySQL Workbench or any SQL client you prefer.
- Ensure that the MySQL environment is correctly installed and configured.

2. Create the E-Commerce Database Structure:

- Create a new database named `e_commerce`.
- Create the following tables:
  - **Customers**: For storing customer information with columns `customer_id`, `first_name`, `last_name`, `email`.
  - **Products**: For storing product information with columns `product_id`, `product_name`, `price`, `stock_quantity`.
  - **Orders**: For storing order details with columns `order_id`, `customer_id`, `order_date`, `total_amount`.
  - **Order_Details**: For storing order item details with columns `order_detail_id`, `order_id`, `product_id`, `quantity`, `unit_price`.

3. Insert Initial Data into the Database:

- Insert at least 5 records into each of the following tables:
  - **Customers**: Ensure you use a variety of first and last names with different capitalization.
  - **Products**: Add product names, prices, and stock levels.
  - **Orders**: Add sample orders placed by different customers.
  - **Order_Details**: Link products to orders, specifying the quantity of each product ordered.

4. String Manipulation with MySQL Functions:

- Use the `concat()` function to concatenate customers' first and last names in the format "First Last".
- Use the `trim()` function to clean up any extra spaces in the customer names.
- Write a query that combines the `concat()` and `trim()` functions to return a list of customers with their properly formatted names.

Expected Output:

```
diff
Copy code
+-------------------+
| full_name         |
+-------------------+
| John Doe          |
| Jane Smith        |
| Alice Johnson     |
+-------------------+
```

5. Numeric Functions for Product Pricing:

- Calculate a 10% discount on all products using the `abs()` and `mod()` functions.
- Write a query to calculate the discounted price of each product.
- Use the `truncate()` function to ensure that the discounted price is rounded to two decimal places.

Expected Output:

```
diff
Copy code
+--------------------+-------------------+--------------------+
| product_name       | original_price    | discounted_price   |
+--------------------+-------------------+--------------------+
| Laptop             | 1000.00           | 900.00             |
| Smartphone         | 500.00            | 450.00             |
+--------------------+-------------------+--------------------+
```

6. Date Functions for Order Management:

- Use the `curdate()` and `now()` functions to generate a report of all orders placed in the current month.
- Write a query to calculate the difference in days between `order_date` and the current date, showing how long ago each order was placed.

Expected Output:

```
diff
Copy code
+-----------+------------+--------------------+
| order_id  | order_date | days_since_order   |
+-----------+------------+--------------------+
| 1         | 2024-09-01 | 7                  |
| 2         | 2024-09-03 | 5                  |
+-----------+------------+--------------------+
```

7. Aggregate Functions for Order Statistics:

- Use the `sum()` and `avg()` functions to calculate the total and average order amounts.
- Write a query to calculate the total number of products ordered for each order using the `count()` function.

Expected Output:

```
diff
Copy code
+-----------+----------------+
| order_id  | total_products  |
+-----------+----------------+
| 1         | 5              |
| 2         | 3              |
+-----------+----------------+
```

8. Transaction Management with COMMIT and ROLLBACK:

- Write a set of queries to simulate placing a new order. First, insert a new order record into the `Orders` table, followed by inserting related records into the `Order_Details` table. After each insertion, use the `SAVEPOINT` keyword.
- Simulate an error by rolling back to a previous `SAVEPOINT` if the quantity of any product ordered exceeds the stock available. Otherwise, commit the transaction.

Expected Steps:

1. Begin Transaction.
2. Insert order into `Orders` table.
3. Insert items into `Order_Details`.
4. If stock is sufficient, commit the transaction. If not, roll back to the `SAVEPOINT`.

Submission:

- Submit the SQL scripts used to create and populate the database tables.
- Include a README file with instructions on how to set up and run the queries.
- Ensure that all functionalities work as expected and handle edge cases, such as insufficient stock in transactions.

```
+-----------+----------------+
| order_id  | total_products  |
+-----------+----------------+
| 1         | 5              |
| 2         | 3              |
+-----------+----------------+
```

8. Transaction Management with COMMIT and ROLLBACK:

- Write a set of queries to simulate placing a new order. First, insert a new order record into the `Orders` table, followed by inserting related records into the `Order_Details` table. After each insertion, use the `SAVEPOINT` keyword.
- Simulate an error by rolling back to a previous `SAVEPOINT` if the quantity of any product ordered exceeds the stock available. Otherwise, commit the transaction.

Expected Steps:

1. Begin Transaction.
2. Insert order into `Orders` table.
3. Insert items into `Order_Details`.
4. If stock is sufficient, commit the transaction. If not, roll back to the `SAVEPOINT`.

Submission:

- Submit the SQL scripts used to create and populate the database tables.
- Include a README file with instructions on how to set up and run the queries.
- Ensure that all functionalities work as expected and handle edge cases, such as insufficient stock in transactions.