Assignment 8.1

Given: Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer's name and email address for customers in a specific city.

1. **A SELECT query to retrieve all columns from a 'customers' table:**

   ```
   SELECT * FROM Customers;
   ```

2. **To modify it to return only the Customer name and email address for customers in a specific city (let's say the city is 'New York'), you can use a WHERE clause to filter the results based on the city:**

   ```
   SELECT CusName, CusEmail FROM Customers
   WHERE City = 'Mexico';
   ```

Assignment 8.2

Given: Craft a query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region, and a LEFT JOIN to display all customers including those without orders.

Here's the SQL query combining the desired aspects:

```
SELECT c.customer_id, c.customer_name, c.region, o.order_id, o.order_date (-- Add other desired order columns here)

FROM customers c

LEFT JOIN orders o ON c.customer_id = o.customer_id

WHERE c.region = 'your_desired_region';
```

Explanation:

1. **SELECT**: This clause specifies the columns you want to retrieve from the tables.
    - c.customer_id: Customer ID from the customers table.
    - c.customer_name: Customer name from the customers table.
    - c.region: Customer region from the customers table.
    - o.order_id: Order ID from the orders table (optional, add other desired order columns).
    - o.order_date: Order date from the orders table (optional, add other desired order columns).
2. **FROM**: This clause specifies the tables involved in the query.
    - customers c: The customers table is aliased as c for readability.
3. **LEFT JOIN**: This clause joins the orders table to the customers table.
    - o.customer_id = c.customer_id: This is the join condition, ensuring rows are matched where the customer ID in both tables is the same.
    - **LEFT JOIN**: This type of join ensures all rows from the customers table are included, even if there isn't a matching order in the orders table.
4. **WHERE**: This clause filters the results based on a specific condition.
    - c.region = 'your_desired_region': This filters the results to include only customers from the specified region (replace 'your_desired_region' with the actual region value).

Assignment 8.3

Given: Utilize a subquery to find customers who have placed orders above the average order value, and write a UNION query to combine two SELECT statements with the same number of columns.

To find customers who have placed orders above the average order value using a subquery and then write a UNION query to combine two SELECT statements with the same number of columns, you can do the following:

1. **Subquery to find customers with orders above the average order value:**

```
SELECT CustomerID
FROM orders
GROUP BY CustomerID
HAVING AVG(TotalAmount) > (SELECT AVG(TotalAmount) FROM orders);
```

2. **UNION query to combine two SELECT statements with the same number of columns:**

   I want to retrieve the names of customers who have placed orders above the average order value. Then

```
-- First SELECT statement: Customers with orders above the average order
value
SELECT Name
FROM customers
WHERE CustomerID IN (
   SELECT CustomerID
   FROM orders
   GROUP BY CustomerID
   HAVING AVG(TotalAmount) > (SELECT AVG(TotalAmount) FROM orders)
)
-- Second SELECT statement: All customers
UNION
SELECT Name
FROM customers;
```

Assignment 8.4

Given: Compose SQL statements to BEGIN a transaction, INSERT a new record into the 'orders' table, COMMIT the transaction, then UPDATE the 'products' table, and ROLLBACK the transaction.

Here's an example of SQL statements to perform the actions you described:

```
-- BEGIN the transaction BEGIN;

-- INSERT a new record into the 'orders' table

   INSERT INTO orders (customer_id, order_date, total_amount)

   VALUES (123, '2024-05-28', 100.00);

-- COMMIT the transaction COMMIT;

-- UPDATE the 'products' table

   UPDATE products

   SET stock_quantity = stock_quantity - 1

   WHERE product_id = 456;

-- ROLLBACK the transaction ROLLBACK;
```

In this:

1. The transaction begins with the BEGIN statement.

2. An INSERT statement adds a new record into the 'orders' table.

3. The transaction is then committed using the COMMIT statement, making the changes permanent in the database.

4. An UPDATE statement modifies the 'products' table by decrementing the stock quantity of a specific product.

5. Finally, the transaction is rolled back using the ROLLBACK statement, undoing any changes made after the transaction began.