Assignment 1: Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer name and email address for customers in a specific city.

```
SELECT * FROM customers;
```

```
| customer_id | customer_name | email_address       | city        | country   |
| 1           | John Doe      | john@example.com    | New York    | USA       |
| 2           | Alice Smith   | alice@example.com   | Los Angeles| USA       |
| 3           | Bob Johnson   | bob@example.com     | Chicago     | USA       |
| 4           | Emily Brown   | emily@example.com   | London      | UK        |
| 5           | Michael Lee   | michael@example.com | Sydney      | Australia |
```

```
SELECT John Doe,  john@example.com    FROM customers WHERE city = ‘New York’   ;
```

Assignment 2: 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.

```
CREATE TABLE orders (
   order_id INT,
   customer_id INT,
   order_date DATE,
   total_amount DECIMAL(10, 2)
);

INSERT INTO orders (order_id, customer_id, order_date, total_amount)
VALUES
```

```
   (1, 1, '2024-05-01', 100.00),
   (2, 3, '2024-05-02', 75.50),
   (3, 2, '2024-05-03', 120.00),
   (4, 1, '2024-05-04', 45.25),
   (5, 5, '2024-05-05', 200.00);

SELECT orders.order_id, customers.customer_name, customers.city,
orders.total_amount
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id
WHERE customers.country = 'USA';

SELECT customers.customer_id, customers.customer_name, orders.order_id,
orders.total_amount
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

Assignment 3: 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.

```
SELECT customer_id, customer_name
FROM customers
WHERE customer_id IN (
   SELECT customer_id
   FROM orders
   GROUP BY customer_id
```

```
    HAVING AVG(total_amount) > (
        SELECT AVG(total_amount)
        FROM orders
    )
);

SELECT customer_id, customer_name, city, country
FROM customers
WHERE country = 'USA'
UNION
SELECT customer_id, customer_name, city, country
FROM customers
WHERE country = 'UK';
```

Assignment 4: 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.

```
BEGIN TRANSACTION;

INSERT INTO orders (customer_id, order_date, total_amount)
VALUES (1, '2024-05-29', 150.00);

COMMIT;
UPDATE products
SET price = price * 1.1
WHERE category = 'Electronics';
```

```
ROLLBACK;
```

Assignment 5: Begin a transaction, perform a series of INSERTs into 'orders', setting a SAVEPOINT after each, rollback to the second SAVEPOINT, and COMMIT the overall transaction.

```
BEGIN TRANSACTION;

INSERT INTO orders (customer_id, order_date, total_amount)
VALUES (1, '2024-05-29', 150.00);
SAVEPOINT savepoint1;

INSERT INTO orders (customer_id, order_date, total_amount)
VALUES (2, '2024-05-30', 200.00);
SAVEPOINT savepoint2;

ROLLBACK TO SAVEPOINT savepoint2;

COMMIT;
```

Assignment 6: Draft a brief report on the use of transaction logs for data recovery and create a hypothetical scenario where a transaction log is instrumental in data recovery after an unexpected shutdown.

Transaction logs play a critical role in ensuring data integrity and facilitating data recovery in database systems. They record every change made to the database, providing a detailed history of transactions. This historical record allows database administrators to recover data to a consistent state in the event of system failures, crashes, or other unexpected events.

1. Recording Changes: Transaction logs capture all modifications to the database, including INSERTs, UPDATEs, and DELETEs, along with relevant metadata such as timestamps and transaction IDs.

2. Maintaining Durability: By persistently storing transaction logs on disk, databases ensure that committed transactions are durable and not lost even in the event of power failures or system crashes.

3. Supporting Rollback and Recovery: Transaction logs enable rollback operations to reverse the effects of incomplete or aborted transactions. They also facilitate point-in-time recovery, allowing administrators to restore the database to a specific moment before a failure occurred.

Hypothetical Scenario:

Imagine a scenario where a large e-commerce platform experiences an unexpected shutdown due to a hardware failure during peak shopping hours. As a result, the database becomes inaccessible, leading to potential data loss and service disruption.

However, thanks to the robust transaction logging system in place, the database administrators can initiate recovery procedures to restore the system to a consistent state. Here's how transaction logs are instrumental in this process:

1. Identification of Last Committed Transactions: Upon restarting the database, administrators examine the transaction logs to identify the last committed transactions before the shutdown occurred.

2. Redo and Undo Operations: Using the information stored in the transaction logs, the administrators perform redo and undo operations to reapply committed transactions and roll back incomplete or uncommitted ones. This ensures that the database reflects a consistent state as of the moment before the unexpected shutdown.

3. Point-in-Time Recovery: Transaction logs allow administrators to perform point-in-time recovery, enabling them to restore the database to a specific timestamp before the failure. This capability minimizes data loss and ensures data consistency across the system.

4. Verification and Validation: After completing the recovery process, administrators verify the integrity of the restored data by comparing it with the information stored in the transaction logs. Any discrepancies are addressed promptly to ensure data accuracy.