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 WHERE city = ‘Delhi’;
SELECT customer_name, email FROM customers WHERE city = ‘Bangalore’;
```

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.

```
SELECT c.customer_name, c.email, o.order_id, o.order_date
FROM customers c
INNER JOIN orders o ON c.customer_id = o.customer_id
WHERE c.region = ‘Delhi’;

SELECT c.customer_name, c.email, o.order_id, o.order_date
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.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
FROM orders
GROUP BY customer_id
HAVING AVG(order_value) > (SELECT AVG(order_value) FROM orders);
```

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;

INSERT INTO orders (customer_id, product_id, quantity)
VALUES (1, 101, 5);

COMMIT;

BEGIN;

UPDATE products SET stock = stock - 5 WHERE product_id = 101;

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;

INSERT INTO orders (customer_id, product_id, quantity)
VALUES (1, 101, 5);
SAVEPOINT savepoint1;

INSERT INTO orders (customer_id, product_id, quantity)
VALUES (2, 102, 3);
SAVEPOINT savepoint2;

INSERT INTO orders (customer_id, product_id, quantity)
VALUES (3, 103, 4);

ROLLBACK TO 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 maintain a sequential record of all transactions, including INSERTs, UPDATEs, DELETEs, and other database operations. They provide a detailed history of changes made to the database, enabling DBMS to restore data to a consistent state in case of failures or errors. Key benefits of transaction logs for data recovery include:

1. **Point-in-Time Recovery:**
   Transaction logs allow DBMS to restore databases to a specific point in time before the occurrence of an issue. This capability ensures data consistency and minimizes data loss during recovery processes.

2. **Rollback and Rollforward Operations:**
   Transaction logs facilitate rollback operations to undo uncommitted transactions and rollforward operations to apply committed transactions during recovery. This ensures that database changes are reverted or applied accurately to restore data integrity.

3. **Reduction of Downtime:**
   By leveraging transaction logs, organizations can minimize downtime associated with data recovery efforts. DBMS can quickly replay logged transactions to restore databases, thereby reducing the impact on business operations.

Hypothetical Scenario:
Consider a retail company that maintains a comprehensive inventory database to track product stock levels, sales transactions, and customer orders. One day, the company experiences an unexpected power outage, resulting in the abrupt shutdown of its database server. As a result of the shutdown, the inventory database becomes corrupted, leading to potential data loss and inconsistencies.

In this scenario, the transaction log associated with the inventory database becomes instrumental in data recovery efforts. Here's how transaction logs aid in restoring the database:

1. **Identification of Unfinished Transactions:**
   Upon restarting the database server, the DBMS analyzes the transaction log to identify any incomplete transactions that were active at the time of the shutdown. Unfinished transactions are rolled back to maintain data consistency.

2. **Recovery to a Consistent State:**
   The DBMS utilizes the transaction log to roll forward committed transactions, ensuring that all changes made to the database before the shutdown are applied. This process restores the inventory database to a consistent state without data loss.

3. **Verification and Integrity Checks:**
   Before resuming normal operations, the DBMS performs verification and integrity checks using the transaction log. This ensures that the recovered database is free from errors and discrepancies, guaranteeing data reliability.