ASSIGNMENT 2

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 specific city.

```
-- Retrieve all columns from the 'customers' table

SELECT * FROM customers;


-- Modify to return only the customer name and email address for customers in a specific city (let's say the city is 'Mumbai')

SELECT customer_name, email_address

FROM customers

WHERE city = 'Mumbai';
```

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.

```
-- Query using INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region (let's say the region is 'Paris')

SELECT customers.customer_name, orders.order_id, orders.order_date

FROM customers

INNER JOIN orders ON customers.customer_id = orders.customer_id

WHERE customers.region = 'Paris';
```

-- Query using LEFT JOIN to display all customers including those without orders

SELECT customers.customer_name, orders.order_id, orders.order_date

FROM customers

LEFT JOIN orders ON customers.customer_id = orders.customer_id;

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.

Using a subquery to find customers who have placed orders above the average order value:

SELECT customer_id, order_id, order_value

FROM orders

WHERE order_value > (SELECT AVG(order_value) FROM orders);

Writing a UNION query to combine two SELECT statements with the same number of columns:

-- Query 1: Customers who have placed orders above the average order value

SELECT customer_id, order_id, order_value

FROM orders

WHERE order_value > (SELECT AVG(order_value) FROM orders)

```
UNION

-- Query 2: Customers who have placed orders below or equal to the average order value
SELECT customer_id, order_id, order_value
FROM orders
WHERE order_value <= (SELECT AVG(order_value) FROM orders);
```

4] Compose SQL statements to BEGIN a transaction, INSERT a new record into the ‘orders’ table, COMMIT the transaction, then UPDATE the ‘product’ table, and ROLLBACK the transaction.

```
-- BEGIN a transaction
BEGIN TRANSACTION;

-- INSERT a new record into the 'orders' table
INSERT INTO orders (order_id, customer_id, order_date, total_amount)
VALUES (23, 1234, '2024-06-01', 100.00);

-- COMMIT the transaction
COMMIT;

-- UPDATE the 'product' table
```

```
UPDATE product
SET stock_quantity = stock_quantity - 1
WHERE product_id = 1001;

-- ROLLBACK the transaction
ROLLBACK;
```

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 a transaction
BEGIN TRANSACTION;

-- INSERT the first record into the 'orders' table
INSERT INTO orders (order_id, customer_id, order_date, total_amount)
VALUES (23, 1234, '2024-06-01', 50.00);

-- Set a SAVEPOINT after the first INSERT
SAVEPOINT savepoint1;

-- INSERT the second record into the 'orders' table
```

```
INSERT INTO orders (order_id, customer_id, order_date, total_amount)
VALUES (DEFAULT, 5678, '2024-06-02', 75.00);

-- Set a SAVEPOINT after the second INSERT
SAVEPOINT savepoint2;

-- INSERT the third record into the 'orders' table
INSERT INTO orders (order_id, customer_id, order_date, total_amount)
VALUES (DEFAULT, 9012, '2024-06-03', 100.00);

-- Rollback to the second SAVEPOINT
ROLLBACK TO SAVEPOINT savepoint2;

-- Commit the overall transaction
COMMIT;
```

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.

Report: The Use of Transaction Logs for Data Recovery

Introduction: Transaction logs are crucial components of database management systems that record all transactions made to a database. These logs serve as a

sequential record of changes, providing a detailed history of database modifications. One of the primary purposes of transaction logs is to facilitate data recovery in the event of system failures, crashes, or unexpected shutdowns.

Importance of Transaction Logs for Data Recovery:

---Transaction logs play a vital role in ensuring data integrity and durability.

---They allow databases to maintain ACID (Atomicity, Consistency, Isolation, Durability) properties, particularly the durability aspect, by providing a mechanism for recovering data to a consistent state after failures.

How Transaction Logs Facilitate Data Recovery:

---Transaction logs record every change made to the database, including INSERTs, UPDATEs, and DELETEs, along with corresponding metadata such as timestamps and transaction identifiers.

---In the event of a system failure, the database management system can use these logs to replay transactions and restore the database to a consistent state just before the failure occurred.

Hypothetical Scenario: Consider a scenario where a retail company operates an online store with a database to manage product inventory. During a routine server maintenance procedure, an unexpected power outage causes the database server to shut down abruptly. As a result, the database becomes corrupted, and the inventory records are potentially compromised.

In this scenario, the transaction logs prove instrumental in recovering the database to a consistent state. The logs contain a sequential record of all transactions, including inventory updates, sales orders, and product additions. When the database server restarts after the unexpected shutdown, the database

management system analyses the transaction logs to identify the last committed transactions before the failure occurred.

Using this information, the database management system replays the transactions from the logs, restoring the inventory records to their state just before the system failure. This process ensures data consistency and integrity, preventing loss of critical information and minimizing downtime for the online store.

Conclusion: Transaction logs serve as invaluable tools for data recovery in database management systems. They enable organizations to restore databases to consistent states following unexpected failures or crashes, ensuring data integrity and minimizing the impact of downtime on business operations. Implementing robust backup and recovery strategies that leverage transaction logs is essential for safeguarding against data loss and ensuring continuous availability of critical information.