Day-10 _Assignment:

Manjula Nannuri

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

Sol:

```
> SELECT *
    -> FROM customers;
+-------------+---------------+--------------------------+-------------+
| customer_id | customer_name | email_address            | city        |
+-------------+---------------+--------------------------+-------------+
|           1 | John Doe      | john.doe@example.com      | New York    |
|           2 | Jane Smith    | jane.smith@example.com    | Los Angeles |
|           3 | Alice Johnson | alice.johnson@example.com | New York    |
|           4 | Bob Brown     | bob.brown@example.com     | Chicago     |
|           5 | Emily Davis   | emily.davis@example.com   | New York    |
+-------------+---------------+--------------------------+-------------+
5 rows in set (0.00 sec)

mysql> SELECT customer_name, email_address
    -> FROM customers
    -> WHERE city = 'New York';
+---------------+---------------------------+
| customer_name | email_address             |
+---------------+---------------------------+
| John Doe      | john.doe@example.com      |
| Alice Johnson | alice.johnson@example.com |
| Emily Davis   | emily.davis@example.com   |
+---------------+---------------------------+
3 rows in set (0.00 sec)
```

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.

Sol:

```
mysql> SELECT *

    -> FROM orders

    -> INNER JOIN customers ON orders.customer_id = customers.customer_id

    -> WHERE customers.region = 'East';

+----------+-------------+------------+--------------+-------------+---------------+--------+

| order_id | customer_id | order_date | total_amount | customer_id | customer_name | region |

+----------+-------------+------------+--------------+-------------+---------------+--------+

|      101 |           1 | 2024-05-20 |       100.50 |           1 | John Doe      | East   |

|      103 |           1 | 2024-05-22 |       200.75 |           1 | John Doe      | East   |

|      104 |           3 | 2024-05-23 |       150.00 |           3 | Alice Johnson | East   |

+----------+-------------+------------+--------------+-------------+---------------+--------+

3 rows in set (0.00 sec)


mysql>

mysql> SELECT *

    -> FROM customers

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

+-------------+---------------+--------+----------+-------------+------------+--------------+

| customer_id | customer_name | region | order_id | customer_id | order_date | total_amount |

+-------------+---------------+--------+----------+-------------+------------+--------------+

|           1 | John Doe      | East   |      101 |           1 | 2024-05-20 |       100.50 |

|           1 | John Doe      | East   |      103 |           1 | 2024-05-22 |       200.75 |

|           2 | Jane Smith    | West   |      102 |           2 | 2024-05-21 |        75.25 |

|           3 | Alice Johnson | East   |      104 |           3 | 2024-05-23 |       150.00 |
|           4 | Bob Brown     | West   |     NULL |        NULL | NULL       |         NULL |
|           5 | Emily Davis   | East   |     NULL |        NULL | NULL       |         NULL |
+-------------+---------------+--------+----------+-------------+------------+--------------+
6 rows in set (0.00 sec)

mysql>
mysql> SELECT *
    -> FROM customers
    -> LEFT JOIN orders ON customers.customer_id = orders.customer_id
    -> WHERE customers.region = 'East';
+-------------+---------------+--------+----------+-------------+------------+--------------+
| customer_id | customer_name | region | order_id | customer_id | order_date | total_amount |
+-------------+---------------+--------+----------+-------------+------------+--------------+
|           1 | John Doe      | East   |      101 |           1 | 2024-05-20 |       100.50 |
|           1 | John Doe      | East   |      103 |           1 | 2024-05-22 |       200.75 |
|           3 | Alice Johnson | East   |      104 |           3 | 2024-05-23 |       150.00 |
|           5 | Emily Davis   | East   |     NULL |        NULL | NULL       |         NULL |
+-------------+---------------+--------+----------+-------------+------------+--------------+
4 rows in set (0.00 sec)
```

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.

Sol:

```
mysql> SELECT *
    -> 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
    ->     )
    -> );
+-------------+---------------+--------+
| customer_id | customer_name | region |
+-------------+---------------+--------+
|           1 | John Doe      | East   |
|           3 | Alice Johnson | East   |
+-------------+---------------+--------+
2 rows in set (0.01 sec)

mysql> SELECT customer_name
    -> FROM customers
    -> WHERE region = 'East'
    -> UNION
    -> SELECT customer_name
    -> FROM customers
    -> WHERE region = 'West';
+---------------+
| customer_name |
+---------------+
| John Doe      |
| Alice Johnson |
| Emily Davis   |
| Jane Smith    |
| Bob Brown     |
+---------------+
5 rows in set (0.00 sec)
```

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.

Sol:

```
mysql> CREATE TABLE products (
    ->     product_id INT PRIMARY KEY,
    ->     product_name VARCHAR(100),
    ->     price DECIMAL(10,2)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO products (product_id, product_name, price) VALUES
    -> (1, 'Product A', 50.00),
    -> (2, 'Product B', 75.00),
    -> (3, 'Product C', 100.00);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from products;
+------------+--------------+--------+
| product_id | product_name | price  |
+------------+--------------+--------+
|          1 | Product A    |  50.00 |
|          2 | Product B    |  75.00 |
|          3 | Product C    | 100.00 |
+------------+--------------+--------+
3 rows in set (0.00 sec)

mysql> BEGIN;
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE products
    -> SET price = price * 1.10;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> select * from products;
+------------+--------------+--------+
| product_id | product_name | price  |
+------------+--------------+--------+
|          1 | Product A    |  55.00 |
|          2 | Product B    |  82.50 |
|          3 | Product C    | 110.00 |
+------------+--------------+--------+
3 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from products;
+------------+--------------+--------+
| product_id | product_name | price  |
+------------+--------------+--------+
|          1 | Product A    |  55.00 |
|          2 | Product B    |  82.50 |
|          3 | Product C    | 110.00 |
+------------+--------------+--------+
3 rows in set (0.00 sec)
```

```
mysql> rollback;

Query OK, 0 rows affected (0.00 sec)


mysql> select * from products;

+------------+--------------+--------+

| product_id | product_name | price  |

+------------+--------------+--------+

|          1 | Product A    |  55.00 |

|          2 | Product B    |  82.50 |

|          3 | Product C    | 110.00 |

+------------+--------------+--------+

3 rows in set (0.00 sec)
```

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

Sol:

```
mysql> CREATE TABLE orders (

    ->     order_id INT PRIMARY KEY,

    ->     customer_id INT,

    ->     order_date DATE,

    ->     total_amount DECIMAL(10,2)

    -> );

Query OK, 0 rows affected (0.07 sec)


mysql> BEGIN;

Query OK, 0 rows affected (0.00 sec)


mysql>

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

```
    -> (201, 1, '2024-06-01', 200.00);
Query OK, 1 row affected (0.01 sec)

mysql> SAVEPOINT sp1;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> INSERT INTO orders (order_id, customer_id, order_date, total_amount)
VALUES
    -> (202, 2, '2024-06-02', 300.00);
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT sp2;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> INSERT INTO orders (order_id, customer_id, order_date, total_amount)
VALUES
    -> (203, 3, '2024-06-03', 400.00);
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT sp3;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> INSERT INTO orders (order_id, customer_id, order_date, total_amount)
VALUES
    -> (204, 4, '2024-06-04', 500.00);
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT sp4;
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> select * from orders;
+----------+-------------+------------+--------------+
| order_id | customer_id | order_date | total_amount |
+----------+-------------+------------+--------------+
|      201 |           1 | 2024-06-01 |       200.00 |
|      202 |           2 | 2024-06-02 |       300.00 |
|      203 |           3 | 2024-06-03 |       400.00 |
|      204 |           4 | 2024-06-04 |       500.00 |
+----------+-------------+------------+--------------+
4 rows in set (0.00 sec)

mysql> ROLLBACK TO sp2;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from orders;
+----------+-------------+------------+--------------+
| order_id | customer_id | order_date | total_amount |
+----------+-------------+------------+--------------+
|      201 |           1 | 2024-06-01 |       200.00 |
|      202 |           2 | 2024-06-02 |       300.00 |
+----------+-------------+------------+--------------+
2 rows in set (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

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

Sol:

Report on the Use of Transaction Logs for Data Recovery

Introduction:

Transaction logs are essential components of database management systems, serving as a reliable method for ensuring data integrity and facilitating recovery in the event of system failures or unexpected shutdowns. This report explores the significance of transaction logs in data recovery and presents a hypothetical scenario illustrating their crucial role.

Importance of Transaction Logs:

Transaction logs are critical for data recovery due to the following reasons:

Data Integrity: Transaction logs record all changes made to a database, providing a detailed history of transactions. In case of system failures, these logs enable administrators to restore the database to a consistent state, minimizing data loss or corruption.

Point-in-Time Recovery: Transaction logs allow for point-in-time recovery, enabling administrators to restore the database to a specific moment before the failure occurred. This capability is crucial for recovering from errors or disasters while maintaining data consistency.

Minimization of Downtime: By utilizing transaction logs, organizations can minimize downtime associated with data recovery efforts. Rather than restoring an entire database from backups, administrators can replay transactions from the logs to bring the database up to date quickly.

Auditing and Compliance: Transaction logs provide an audit trail of database activities, aiding in compliance with regulatory requirements and internal auditing processes.

Hypothetical Scenario:

Imagine a scenario where a large e-commerce company experiences an unexpected shutdown of its database server due to a power outage. The database stores critical customer information, including orders, payment details, and inventory data. The company relies heavily on this database for its day-to-day operations.

Upon rebooting the server, the database administrators discover that the database is inaccessible, and there are signs of data corruption. However, the company has implemented robust data recovery measures, including transaction logs.

The administrators begin the recovery process by examining the transaction logs to identify the last checkpoint before the shutdown. They analyze the sequence of transactions recorded in the logs to determine the point at which the database became inconsistent.

Using this information, the administrators carefully replay the transactions from the logs, applying each change to the database to bring it back to a consistent state. This process involves rolling forward committed transactions and, if necessary, rolling back uncommitted transactions to maintain data integrity.

After completing the recovery process, the database is restored to its state just before the unexpected shutdown. All customer information, orders, and inventory data are intact and consistent, ensuring minimal disruption to the company's operations.

Conclusion:

Transaction logs play a crucial role in data recovery efforts, providing a reliable mechanism for restoring databases to a consistent state after unexpected failures. By leveraging transaction logs effectively, organizations can minimize downtime, preserve data integrity, and ensure compliance with regulatory requirements. As demonstrated in the hypothetical scenario, a well-planned data recovery strategy that incorporates transaction logs is essential for mitigating the impact of system failures and maintaining business continuity.