Sql-Assignment 02 (day10)

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.

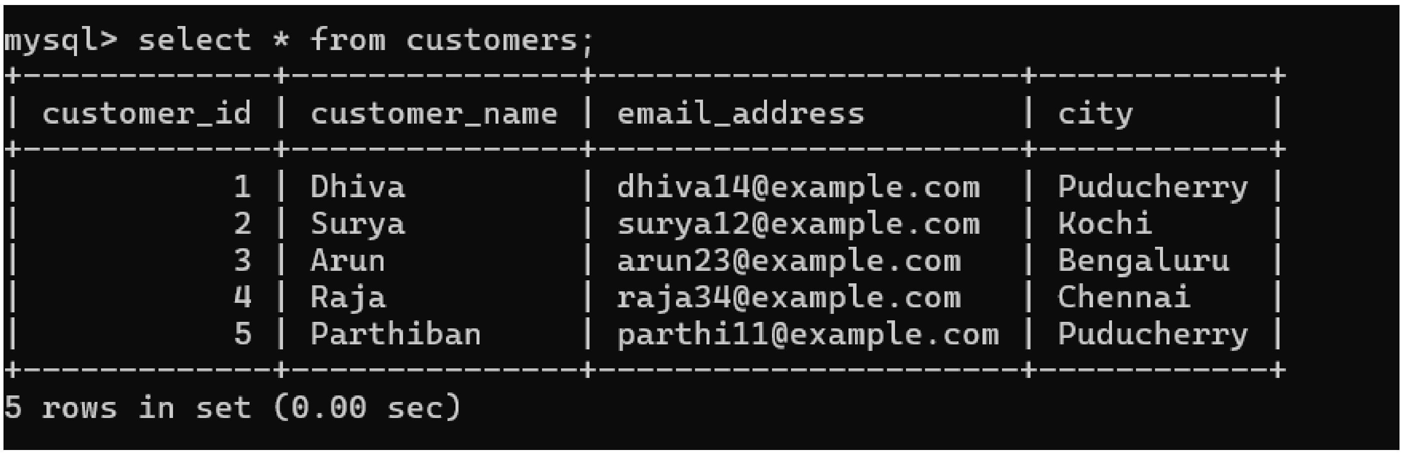
Answer:

Let's say we have a customers table with the following columns: customer_id, customer_name, email_address, city, and phone_number.

Customer_id	Customer_name	Email_address	City
1	Dhiva	Dhiva12@example.com	Puducherry
2	Surya	surya@example.com	Kochi
3	Raja	raja@example.com	Puducherry
4	Arun	arun@example.com	Bengaluru
5	Parthiban	Parthi11@example.com	Puducherry

- **Query to retrieve all columns from the customers table:**
  SELECT * FROM customers; //* will select all column from table

Output:

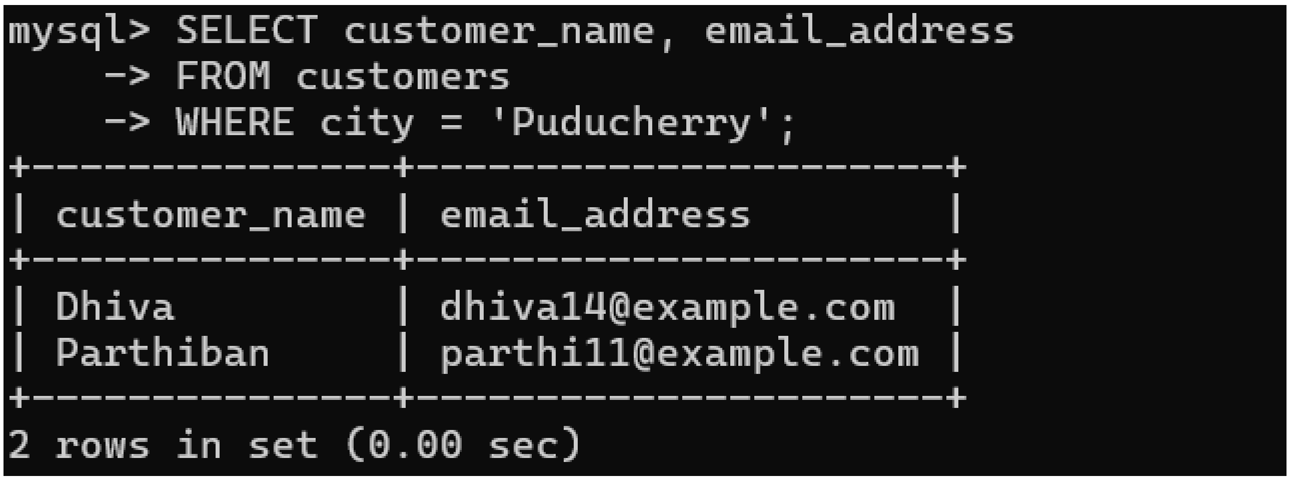


```
mysql> select * from customers;
+-------------+---------------+-----------------------+------------+
| customer_id | customer_name | email_address         | city       |
+-------------+---------------+-----------------------+------------+
|           1 | Dhiva         | dhiva14@example.com   | Puducherry |
|           2 | Surya         | surya12@example.com   | Kochi      |
|           3 | Arun          | arun23@example.com    | Bengaluru  |
|           4 | Raja          | raja34@example.com    | Chennai    |
|           5 | Parthiban     | parthi11@example.com  | Puducherry |
+-------------+---------------+-----------------------+------------+
5 rows in set (0.00 sec)
```

- **Query to retrieve only the customer name and email address for customers in 'New York':**

  SELECT customer_name, email_address
  FROM customers
  WHERE city = 'Puducherry';

Output:

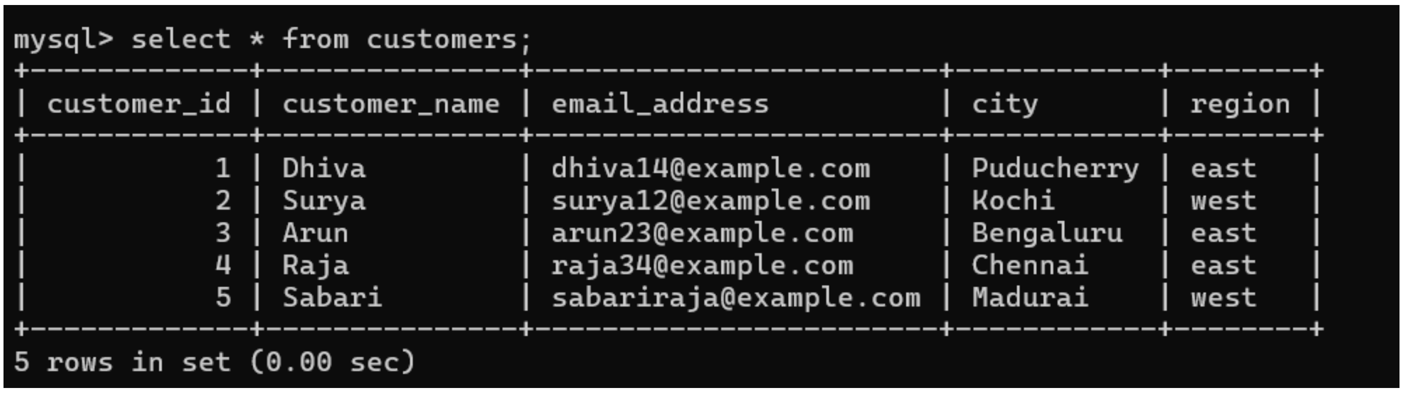


```
mysql> SELECT customer_name, email_address
    -> FROM customers
    -> WHERE city = 'Puducherry';
+---------------+----------------------+
| customer_name | email_address        |
+---------------+----------------------+
| Dhiva         | dhiva14@example.com  |
| Parthiban     | parthi11@example.com |
+---------------+----------------------+
2 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.

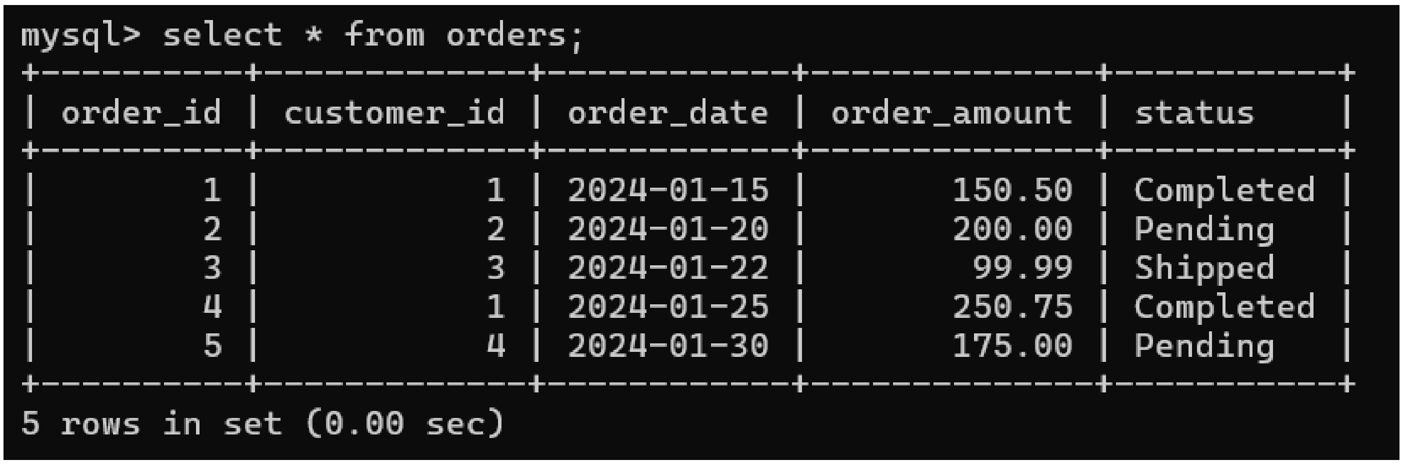
Answer:

Customer Table:



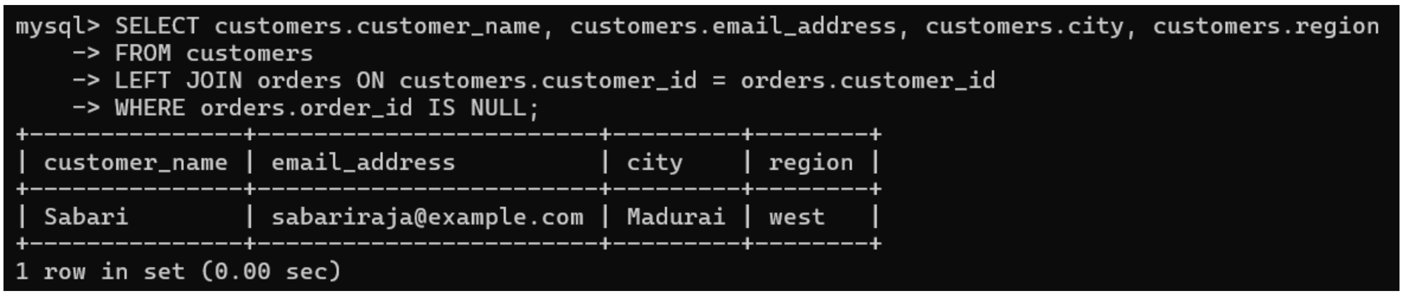
```
mysql> select * from customers;
+-------------+---------------+------------------------+------------+--------+
| customer_id | customer_name | email_address          | city       | region |
+-------------+---------------+------------------------+------------+--------+
|           1 | Dhiva         | dhiva14@example.com    | Puducherry | east   |
|           2 | Surya         | surya12@example.com    | Kochi      | west   |
|           3 | Arun          | arun23@example.com     | Bengaluru  | east   |
|           4 | Raja          | raja34@example.com     | Chennai    | east   |
|           5 | Sabari        | sabariraja@example.com | Madurai    | west   |
+-------------+---------------+------------------------+------------+--------+
5 rows in set (0.00 sec)
```

Order Table:



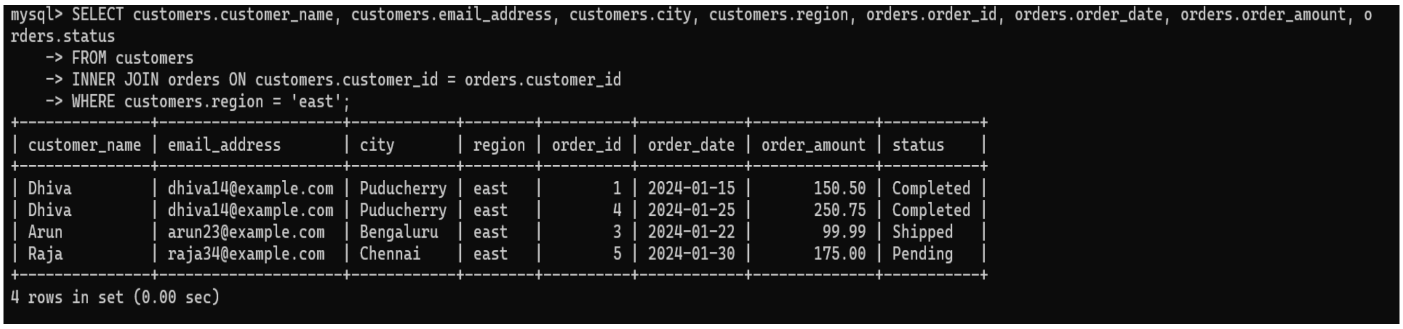
```
mysql> select * from orders;
+----------+-------------+------------+--------------+-----------+
| order_id | customer_id | order_date | order_amount | status    |
+----------+-------------+------------+--------------+-----------+
|        1 |           1 | 2024-01-15 |       150.50 | Completed |
|        2 |           2 | 2024-01-20 |       200.00 | Pending   |
|        3 |           3 | 2024-01-22 |        99.99 | Shipped   |
|        4 |           1 | 2024-01-25 |       250.75 | Completed |
|        5 |           4 | 2024-01-30 |       175.00 | Pending   |
+----------+-------------+------------+--------------+-----------+
5 rows in set (0.00 sec)
```

Display all the customers including those not in orders:



```
mysql> SELECT customers.customer_name, customers.email_address, customers.city, customers.region
    -> FROM customers
    -> LEFT JOIN orders ON customers.customer_id = orders.customer_id
    -> WHERE orders.order_id IS NULL;
+---------------+------------------------+---------+--------+
| customer_name | email_address          | city    | region |
+---------------+------------------------+---------+--------+
| Sabari        | sabariraja@example.com | Madurai | west   |
+---------------+------------------------+---------+--------+
1 row in set (0.00 sec)
```

Query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region:

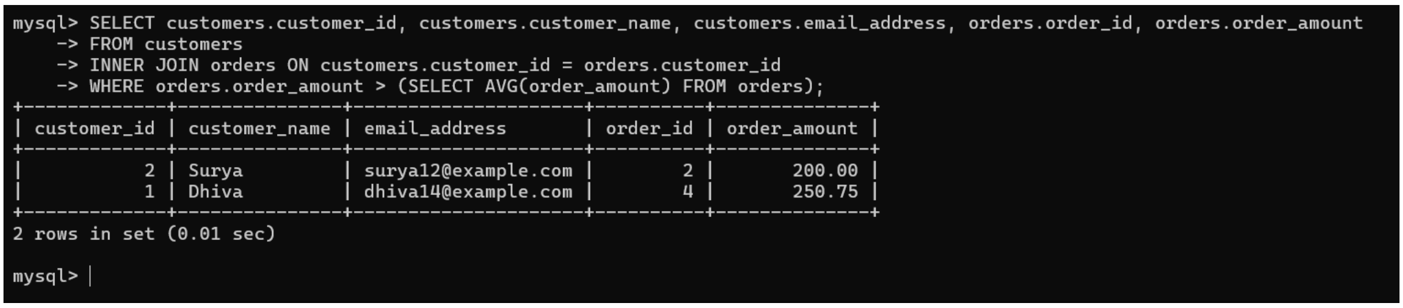


```
mysql> SELECT customers.customer_name, customers.email_address, customers.city, customers.region, orders.order_id, orders.order_date, orders.order_amount, o
rders.status
    -> FROM customers
    -> INNER JOIN orders ON customers.customer_id = orders.customer_id
    -> WHERE customers.region = 'east';
+---------------+--------------------+------------+--------+----------+------------+--------------+-----------+
| customer_name | email_address      | city       | region | order_id | order_date | order_amount | status    |
+---------------+--------------------+------------+--------+----------+------------+--------------+-----------+
| Dhiva         | dhiva14@example.com | Puducherry | east   |        1 | 2024-01-15 |       150.50 | Completed |
| Dhiva         | dhiva14@example.com | Puducherry | east   |        4 | 2024-01-25 |       250.75 | Completed |
| Arun          | arun23@example.com | Bengaluru  | east   |        3 | 2024-01-22 |        99.99 | Shipped   |
| Raja          | raja34@example.com | Chennai    | east   |        5 | 2024-01-30 |       175.00 | Pending   |
+---------------+--------------------+------------+--------+----------+------------+--------------+-----------+
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.

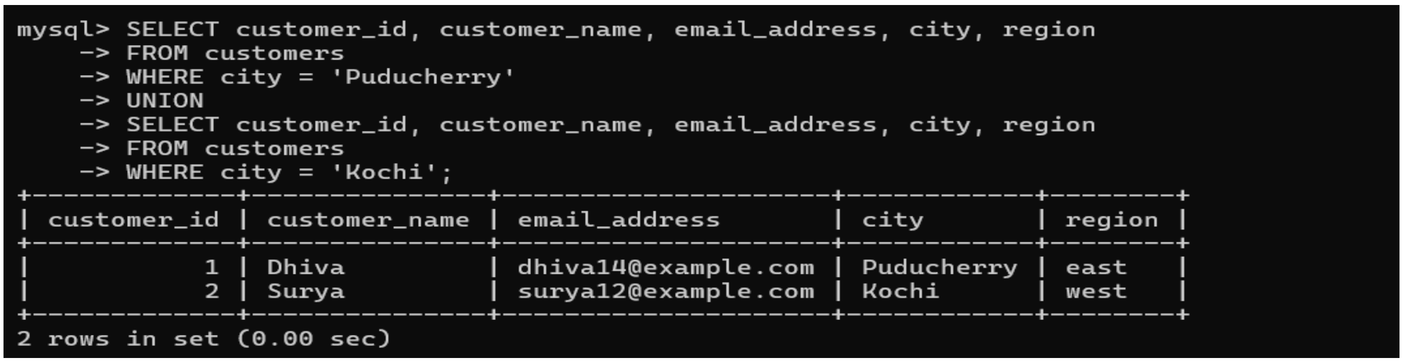
Answer:

1: Subquery to Find Customers Who Have Placed Orders Above the Average Order Value



```
mysql> SELECT customers.customer_id, customers.customer_name, customers.email_address, orders.order_id, orders.order_amount
    -> FROM customers
    -> INNER JOIN orders ON customers.customer_id = orders.customer_id
    -> WHERE orders.order_amount > (SELECT AVG(order_amount) FROM orders);
+-------------+---------------+---------------------+----------+--------------+
| customer_id | customer_name | email_address       | order_id | order_amount |
+-------------+---------------+---------------------+----------+--------------+
|           2 | Surya         | surya12@example.com |        2 |       200.00 |
|           1 | Dhiva         | dhiva14@example.com |        4 |       250.75 |
+-------------+---------------+---------------------+----------+--------------+
2 rows in set (0.01 sec)

mysql> |
```

2: UNION Query to Combine Two SELECT Statements To combine two SELECT statements using UNION, we ensure both SELECT statements have the same number of columns and compatible data types. In this example, we combine customers from 'Puducherry' and 'Kochi'.

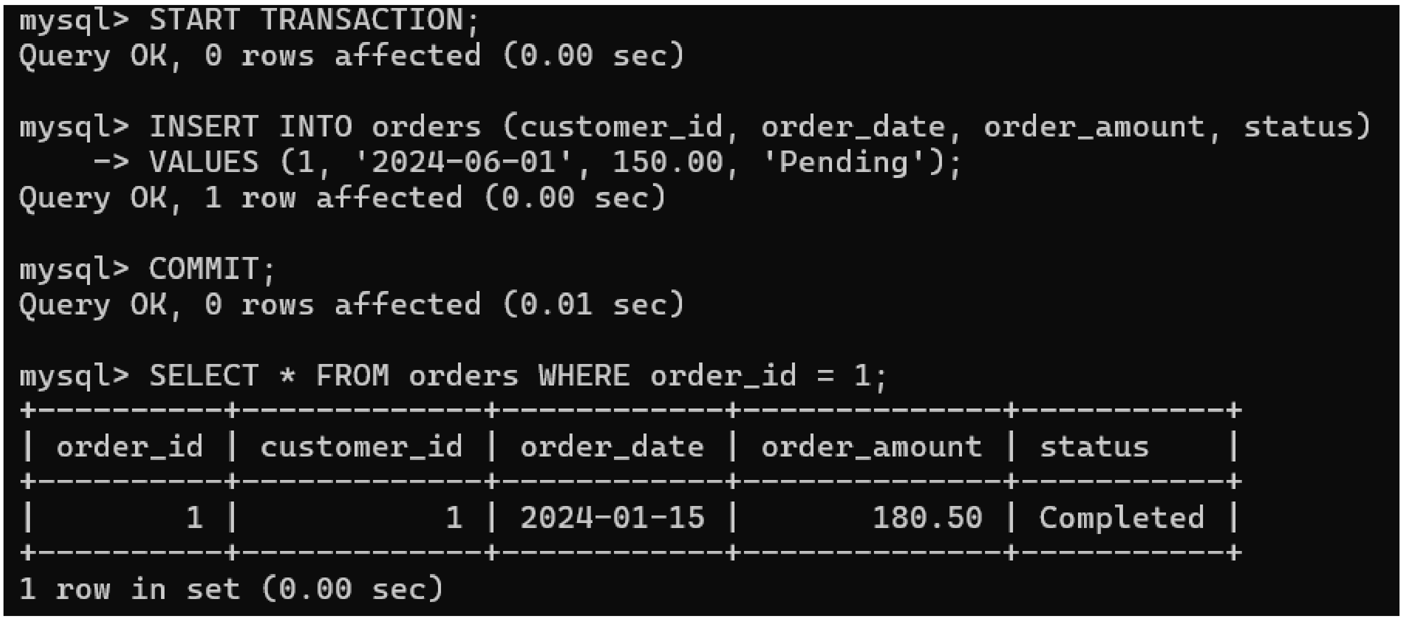


```
mysql> SELECT customer_id, customer_name, email_address, city, region
    -> FROM customers
    -> WHERE city = 'Puducherry'
    -> UNION
    -> SELECT customer_id, customer_name, email_address, city, region
    -> FROM customers
    -> WHERE city = 'Kochi';
+-------------+---------------+---------------------+------------+--------+
| customer_id | customer_name | email_address       | city       | region |
+-------------+---------------+---------------------+------------+--------+
|           1 | Dhiva         | dhiva14@example.com | Puducherry | east   |
|           2 | Surya         | surya12@example.com | Kochi      | west   |
+-------------+---------------+---------------------+------------+--------+
2 rows in set (0.00 sec)
```

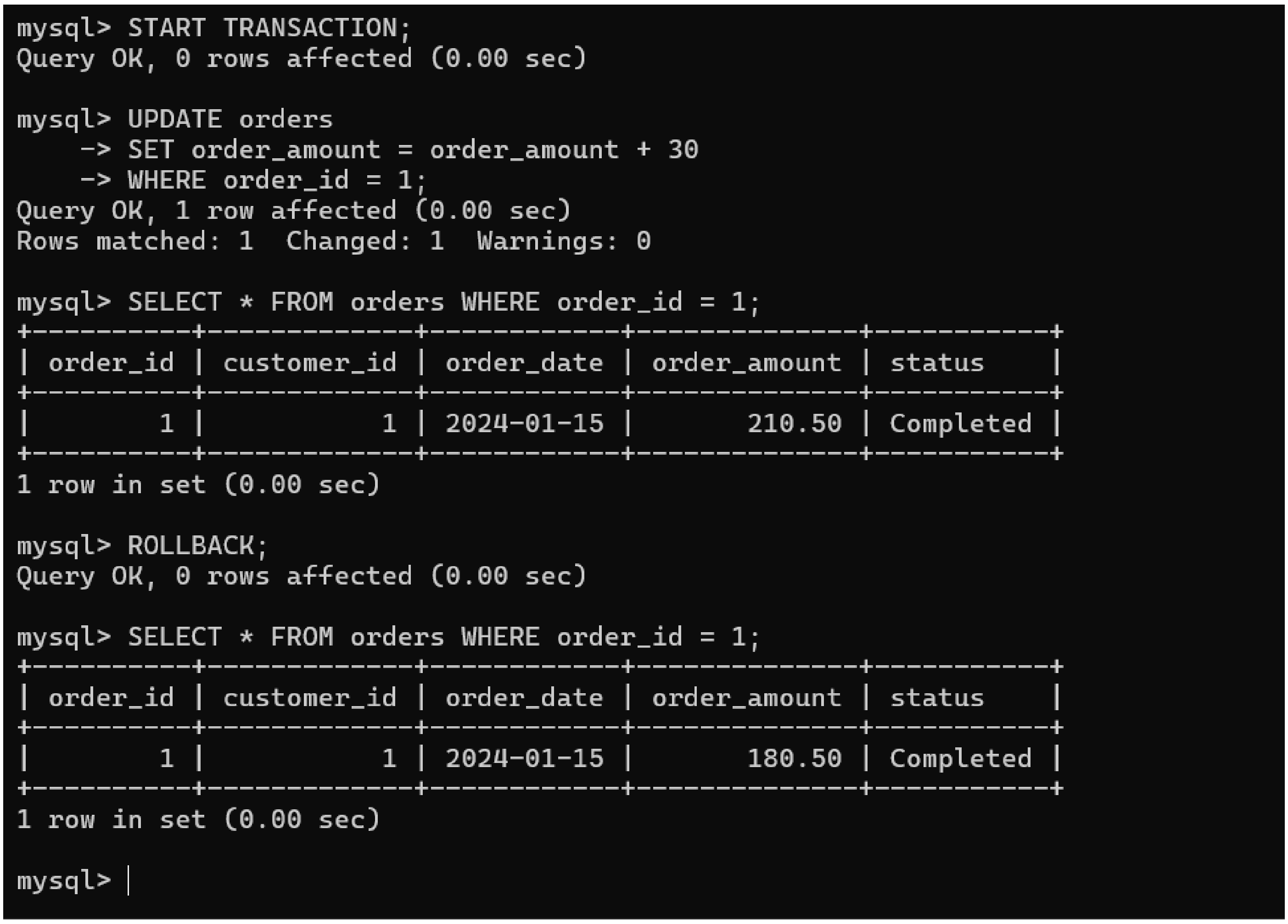
Assignment 4: Compose SQL statements to BEGIN a transaction, INSERT a new record into the 'orders' table, COMMIT the transaction, then UPDATE the 'products' table, and ROLLBACK the transaction.

Answer:

- **Starting the new Transaction and insert a new row into the “orders” table**
- **Commit transaction saves the changes made during the transaction and the next step update the “total_amount” for “order_id” 1 by adding 30 to it.**
- **The select operation displays the updated row with “total_amount” now with 210.00.**
- **The rollback transaction undoes the changes made after the last commit. The update was committed before so the rollback doesn’t revert it.**
- **The SELECT statements displays the same row as the update was committed and rollback has no effect and**
- **The update resets“total_amount” for “order_id” 1 to 180 and restoring the initial value**



```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO orders (customer_id, order_date, order_amount, status)
    -> VALUES (1, '2024-06-01', 150.00, 'Pending');
Query OK, 1 row affected (0.00 sec)

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

mysql> SELECT * FROM orders WHERE order_id = 1;
+----------+-------------+------------+--------------+-----------+
| order_id | customer_id | order_date | order_amount | status    |
+----------+-------------+------------+--------------+-----------+
|        1 |           1 | 2024-01-15 |       180.50 | Completed |
+----------+-------------+------------+--------------+-----------+
1 row in set (0.00 sec)
```

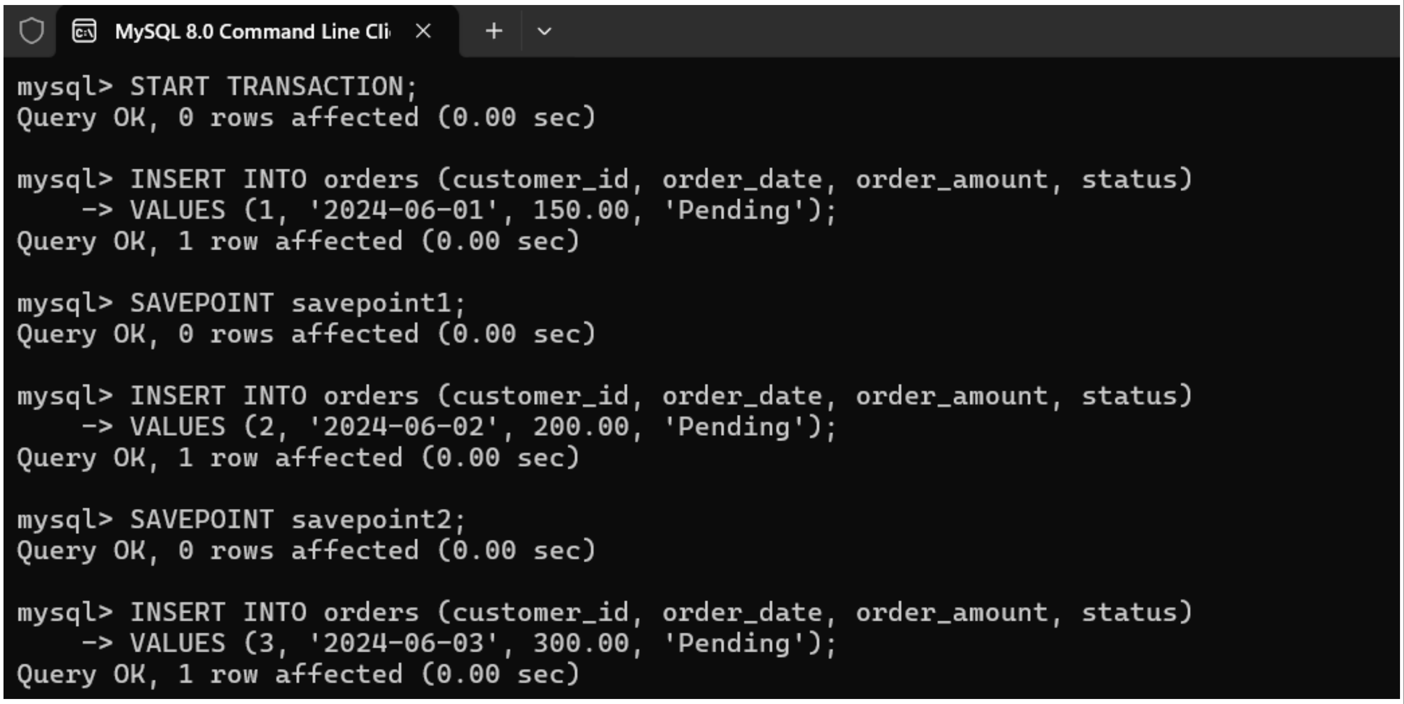


```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE orders
    -> SET order_amount = order_amount + 30
    -> WHERE order_id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM orders WHERE order_id = 1;
+----------+-------------+------------+--------------+-----------+
| order_id | customer_id | order_date | order_amount | status    |
+----------+-------------+------------+--------------+-----------+
|        1 |           1 | 2024-01-15 |       210.50 | Completed |
+----------+-------------+------------+--------------+-----------+
1 row in set (0.00 sec)

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

mysql> SELECT * FROM orders WHERE order_id = 1;
+----------+-------------+------------+--------------+-----------+
| order_id | customer_id | order_date | order_amount | status    |
+----------+-------------+------------+--------------+-----------+
|        1 |           1 | 2024-01-15 |       180.50 | Completed |
+----------+-------------+------------+--------------+-----------+
1 row in set (0.00 sec)

mysql> |
```

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.

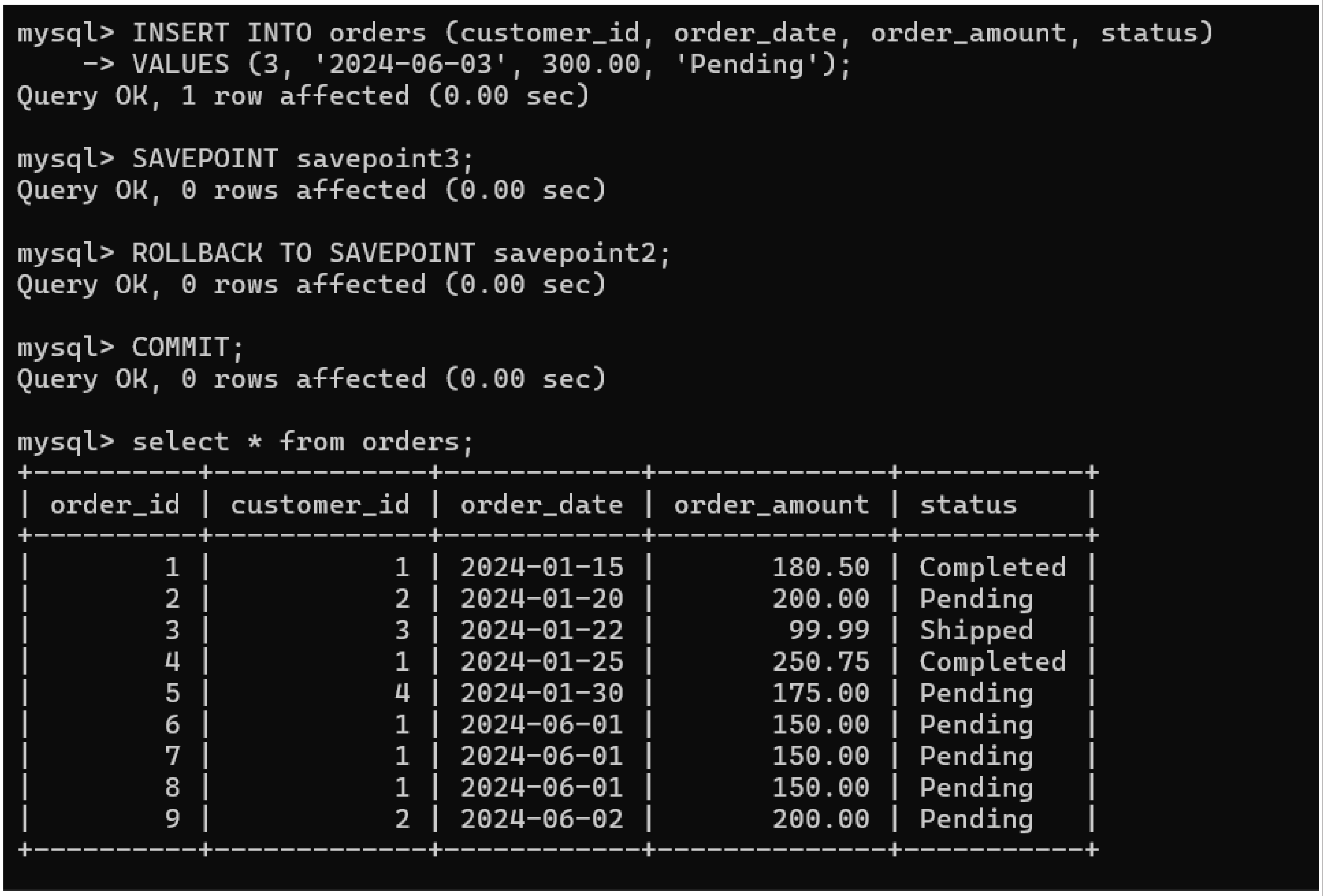
Answer:

- **Starting the new Transaction and insert a new row into the “orders” table**
- **Creating SAVEPOINT savepoint1 and inserting an new row into the table**
- **Creating SAVEPOINT savepoint2 and same repeating the last step as inserting a new row.**
- **ROLLBACK transaction TO SAVEPOINT savepoint in the table and using SELECT and using * to show data on the orders table and commit the table.**
- **Below attaching the executing commands.**



MySQL 8.0 Command Line Cli

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO orders (customer_id, order_date, order_amount, status)
    -> VALUES (1, '2024-06-01', 150.00, 'Pending');
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT savepoint1;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO orders (customer_id, order_date, order_amount, status)
    -> VALUES (2, '2024-06-02', 200.00, 'Pending');
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT savepoint2;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO orders (customer_id, order_date, order_amount, status)
    -> VALUES (3, '2024-06-03', 300.00, 'Pending');
Query OK, 1 row affected (0.00 sec)
```



```
mysql> INSERT INTO orders (customer_id, order_date, order_amount, status)
    -> VALUES (3, '2024-06-03', 300.00, 'Pending');
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT savepoint3;
Query OK, 0 rows affected (0.00 sec)

mysql> ROLLBACK TO SAVEPOINT savepoint2;
Query OK, 0 rows affected (0.00 sec)

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

mysql> select * from orders;
+----------+-------------+------------+--------------+-----------+
| order_id | customer_id | order_date | order_amount | status    |
+----------+-------------+------------+--------------+-----------+
|        1 |           1 | 2024-01-15 |       180.50 | Completed |
|        2 |           2 | 2024-01-20 |       200.00 | Pending   |
|        3 |           3 | 2024-01-22 |        99.99 | Shipped   |
|        4 |           1 | 2024-01-25 |       250.75 | Completed |
|        5 |           4 | 2024-01-30 |       175.00 | Pending   |
|        6 |           1 | 2024-06-01 |       150.00 | Pending   |
|        7 |           1 | 2024-06-01 |       150.00 | Pending   |
|        8 |           1 | 2024-06-01 |       150.00 | Pending   |
|        9 |           2 | 2024-06-02 |       200.00 | Pending   |
+----------+-------------+------------+--------------+-----------+
```

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.

Answer:

SIGNIFICANCE OF TRANSACTION LOGS:

Transaction logs are vital for maintaining data integrity and enabling recovery in database management systems. They keep a detailed record of all changes made to the database, which is essential for restoring the database to a consistent state after system failures or data corruption

CORE FUNCTION OF TRANSACTION LOGS:

1.Logging Transcations: Every SQL Server database has a transaction log that logs all transactions and modifications.

2.Maintaining Consistency: In the event of system failures, the transaction log is crucial for bringing the database back to a consistent state.

OPERATIONS ENAVLED BY THE TRANSACTION LOG:

Recovery of Individual Transactions: If a ROLLBACK statement is issued or an error occurs (e.g., communication loss with a client), the log records are used to undo incomplete transaction modifications.

Recovery During SQL Server Startup: After a server failure, SQL Server performs recovery for each database during startup. Changes recorded in the log but not yet written to data files are rolled forward. Incomplete transactions are rolled back to maintain database integrity.

Restoring to the Point of Failure: After a hardware or disk failure, the database is restored to the most recent state. This process involves restoring the last full database backup, the latest differential backup, and subsequent transaction log backups. The Database Engine reapplies changes from the log to roll forward transactions to the point of failure.

Enhancing high Availability and Disaster Recovery: Transaction logs are critical for high availability solutions such as Always On availability groups, database mirroring, and log shipping.

Example Scenario : Library Management System Consider a library system where users borrow and return books. An unexpected shutdown occurs during a busy day:

Prior to Shutdown: Users borrow books, and these transactions are recorded in the database. The transaction log captures these changes.

During ShutDown: The server crashes due to a power outage. Uncommitted changes remain in memory, and some transactions are incomplete.

Recovery Steps: Upon startup, SQL Server uses the transaction log to roll forward committed changes. It identifies and rolls back incomplete transactions. The database is restored to a consistent state.

After Recovery: Users can continue borrowing and returning books without any data loss.