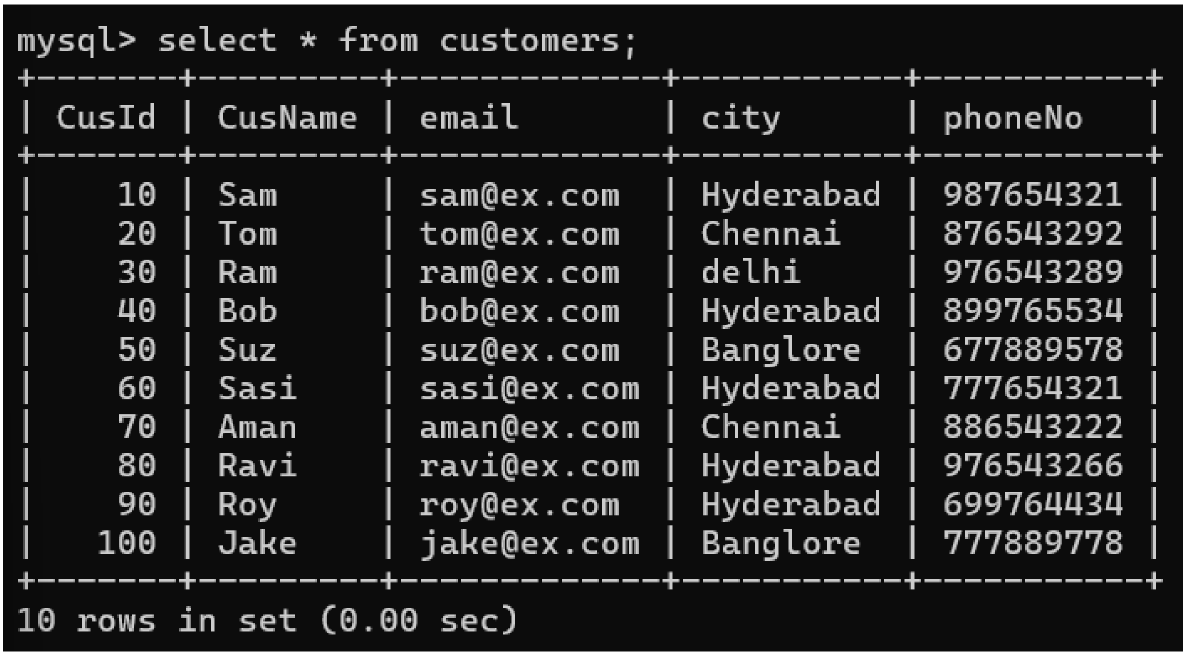
My SQL - Assignment 2

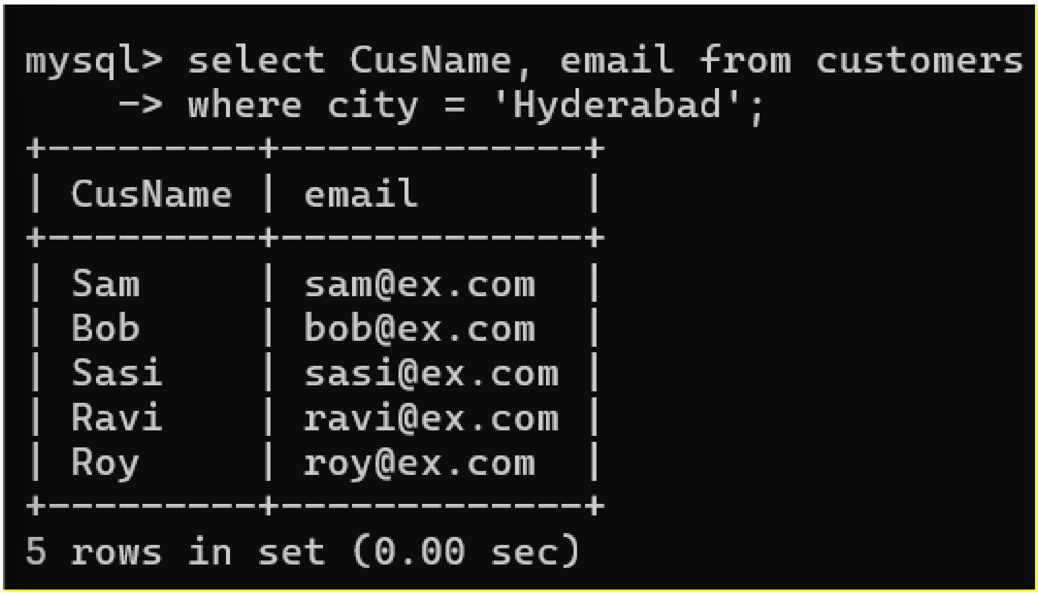
Assignment 1: Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer name and email address for customers in a specific city.

Task 1: Retrieve All Columns from the 'customers' Table



```
mysql> select * from customers;
+-------+---------+-------------+-----------+-----------+
| CusId | CusName | email       | city      | phoneNo   |
+-------+---------+-------------+-----------+-----------+
|    10 | Sam     | sam@ex.com  | Hyderabad | 987654321 |
|    20 | Tom     | tom@ex.com  | Chennai   | 876543292 |
|    30 | Ram     | ram@ex.com  | delhi     | 976543289 |
|    40 | Bob     | bob@ex.com  | Hyderabad | 899765534 |
|    50 | Suz     | suz@ex.com  | Banglore  | 677889578 |
|    60 | Sasi    | sasi@ex.com | Hyderabad | 777654321 |
|    70 | Aman    | aman@ex.com | Chennai   | 886543222 |
|    80 | Ravi    | ravi@ex.com | Hyderabad | 976543266 |
|    90 | Roy     | roy@ex.com  | Hyderabad | 699764434 |
|   100 | Jake    | jake@ex.com | Banglore  | 777889778 |
+-------+---------+-------------+-----------+-----------+
10 rows in set (0.00 sec)
```

Task 2: Retrieve Customer Name and Email Address for Customers in a Specific City

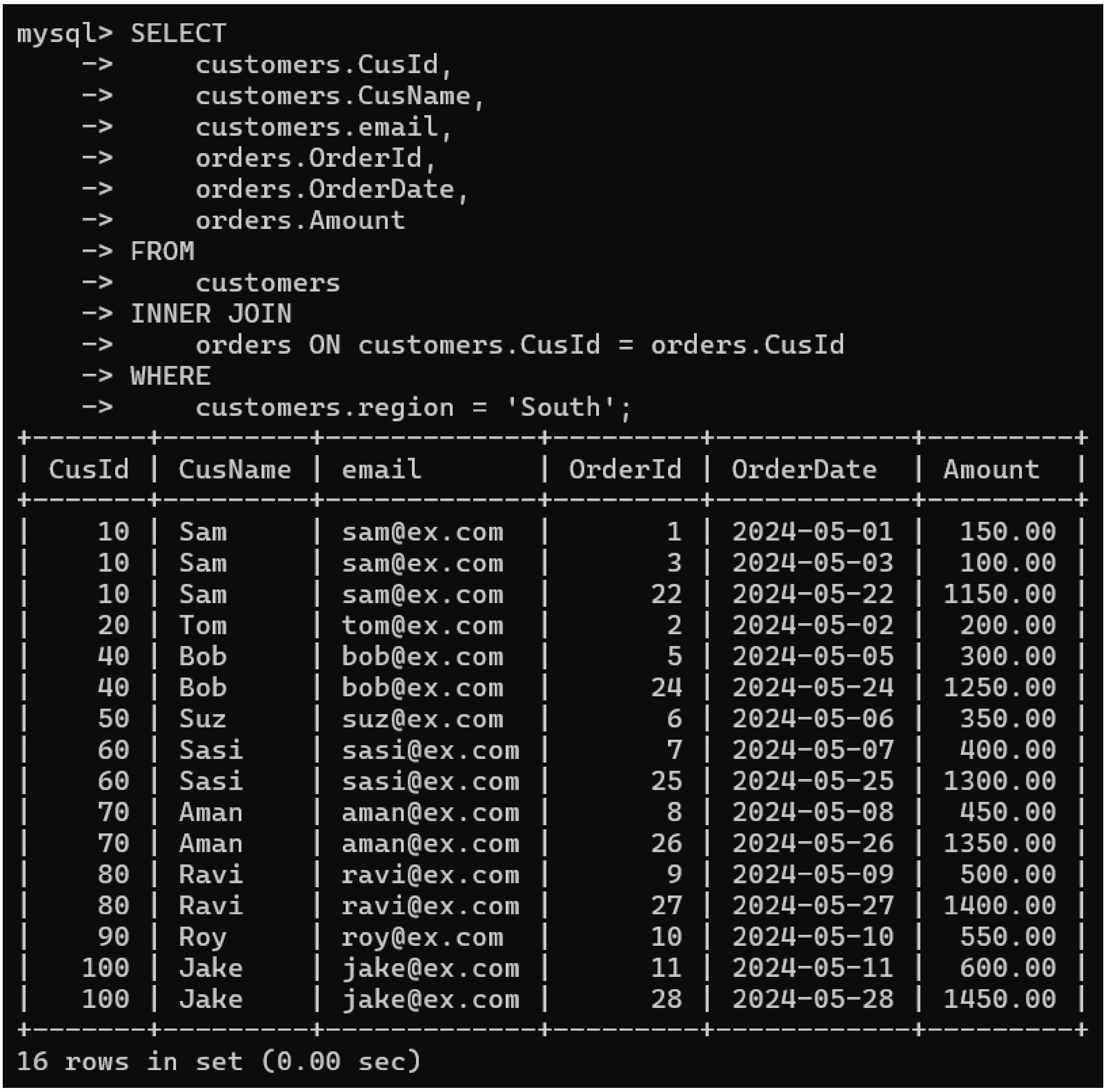


```
mysql> select CusName, email from customers
    -> where city = 'Hyderabad';
+---------+-------------+
| CusName | email       |
+---------+-------------+
| Sam     | sam@ex.com  |
| Bob     | bob@ex.com  |
| Sasi    | sasi@ex.com |
| Ravi    | ravi@ex.com |
| Roy     | roy@ex.com  |
+---------+-------------+
5 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.

Query 1:

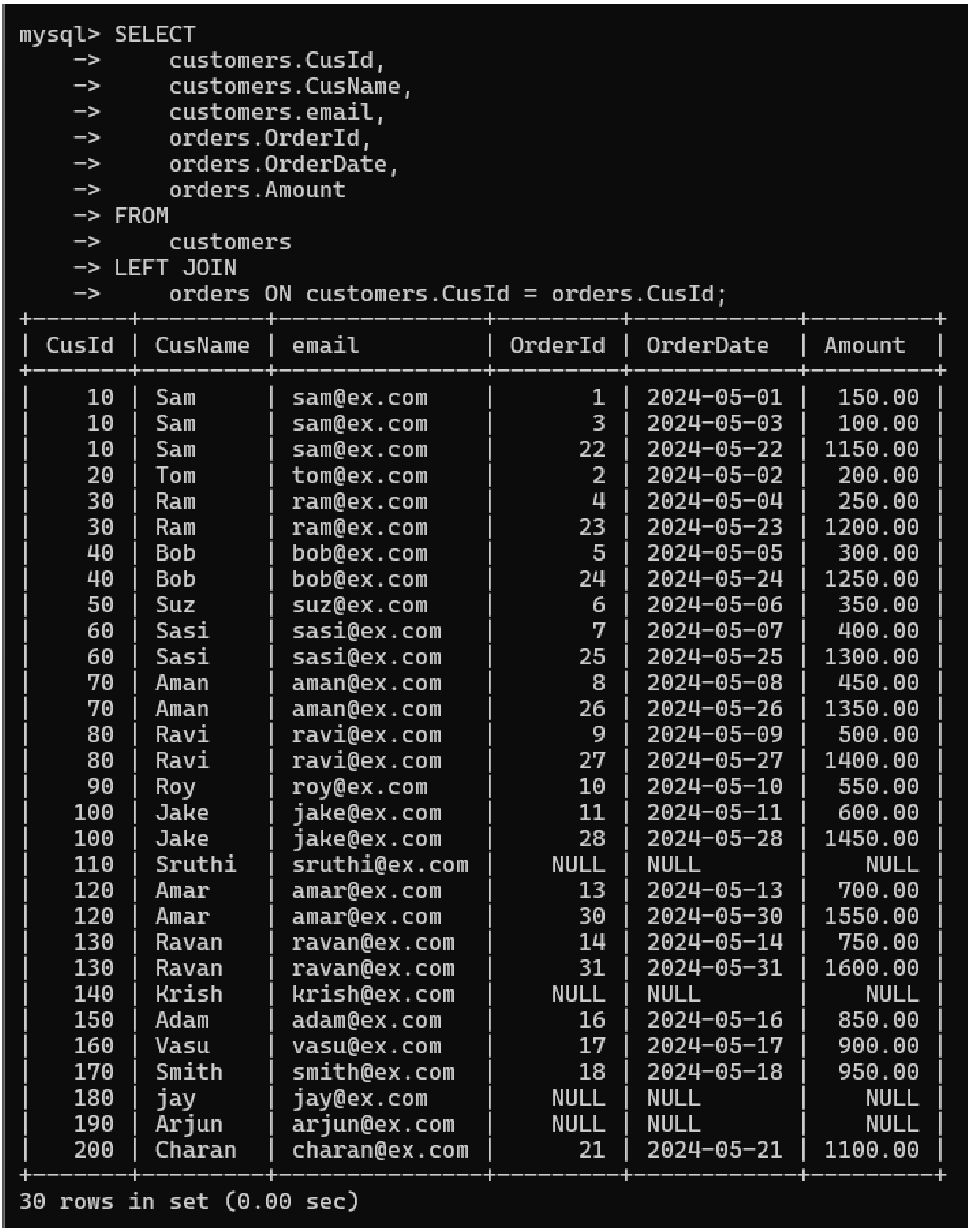
INNER JOIN for Customers in a Specified Region :



```
mysql> SELECT
    ->     customers.CusId,
    ->     customers.CusName,
    ->     customers.email,
    ->     orders.OrderId,
    ->     orders.OrderDate,
    ->     orders.Amount
    -> FROM
    ->     customers
    -> INNER JOIN
    ->     orders ON customers.CusId = orders.CusId
    -> WHERE
    ->     customers.region = 'South';
+-------+---------+-------------+---------+------------+---------+
| CusId | CusName | email       | OrderId | OrderDate  | Amount  |
+-------+---------+-------------+---------+------------+---------+
|    10 | Sam     | sam@ex.com  |       1 | 2024-05-01 |  150.00 |
|    10 | Sam     | sam@ex.com  |       3 | 2024-05-03 |  100.00 |
|    10 | Sam     | sam@ex.com  |      22 | 2024-05-22 | 1150.00 |
|    20 | Tom     | tom@ex.com  |       2 | 2024-05-02 |  200.00 |
|    40 | Bob     | bob@ex.com  |       5 | 2024-05-05 |  300.00 |
|    40 | Bob     | bob@ex.com  |      24 | 2024-05-24 | 1250.00 |
|    50 | Suz     | suz@ex.com  |       6 | 2024-05-06 |  350.00 |
|    60 | Sasi    | sasi@ex.com |       7 | 2024-05-07 |  400.00 |
|    60 | Sasi    | sasi@ex.com |      25 | 2024-05-25 | 1300.00 |
|    70 | Aman    | aman@ex.com |       8 | 2024-05-08 |  450.00 |
|    70 | Aman    | aman@ex.com |      26 | 2024-05-26 | 1350.00 |
|    80 | Ravi    | ravi@ex.com |       9 | 2024-05-09 |  500.00 |
|    80 | Ravi    | ravi@ex.com |      27 | 2024-05-27 | 1400.00 |
|    90 | Roy     | roy@ex.com  |      10 | 2024-05-10 |  550.00 |
|   100 | Jake    | jake@ex.com |      11 | 2024-05-11 |  600.00 |
|   100 | Jake    | jake@ex.com |      28 | 2024-05-28 | 1450.00 |
+-------+---------+-------------+---------+------------+---------+
16 rows in set (0.00 sec)
```

Query 2:

LEFT JOIN to Display All Customers Including Those Without Orders :

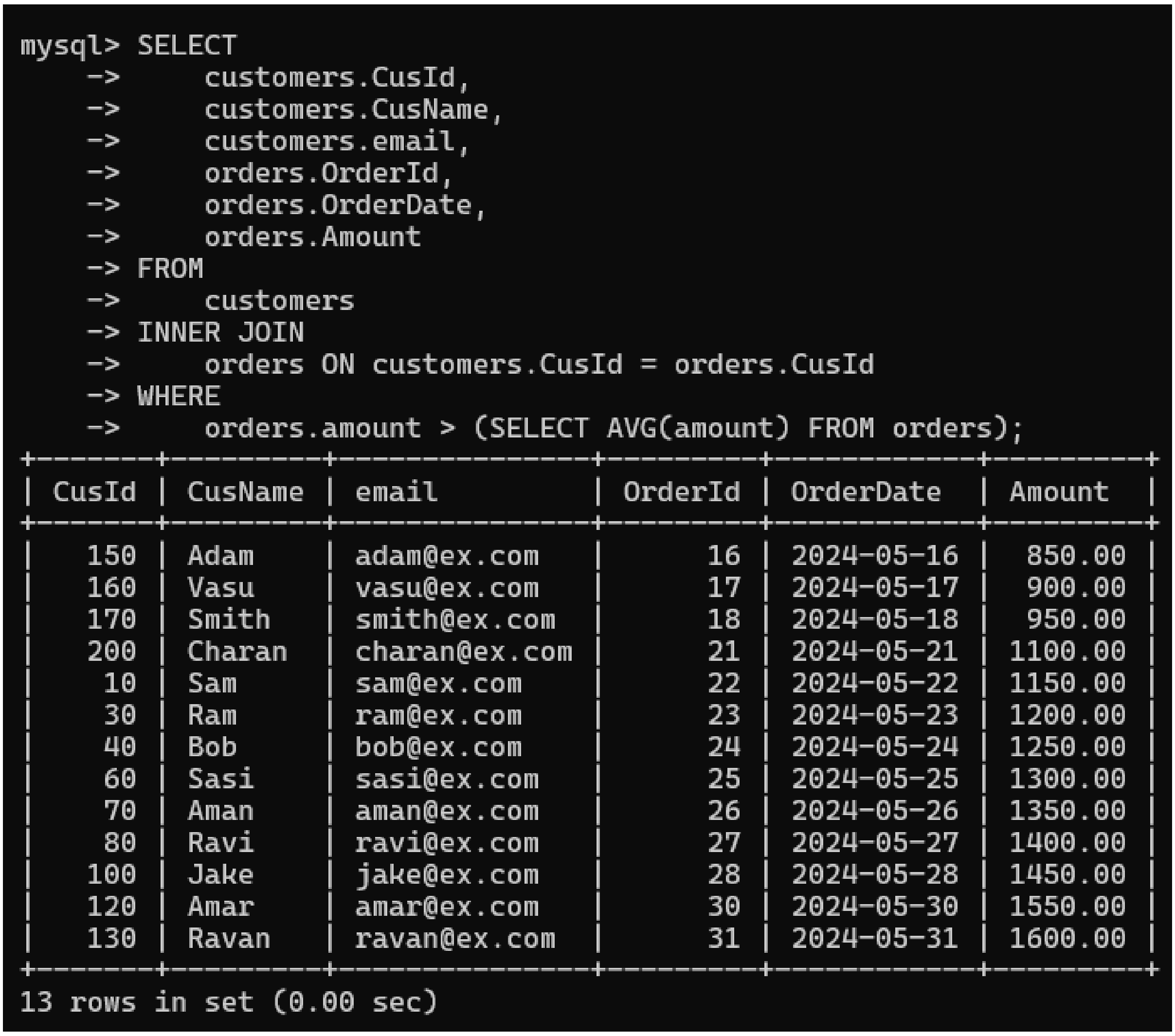


```
mysql> SELECT
    ->     customers.CusId,
    ->     customers.CusName,
    ->     customers.email,
    ->     orders.OrderId,
    ->     orders.OrderDate,
    ->     orders.Amount
    -> FROM
    ->     customers
    -> LEFT JOIN
    ->     orders ON customers.CusId = orders.CusId;
+-------+---------+---------------+---------+------------+---------+
| CusId | CusName | email         | OrderId | OrderDate  | Amount  |
+-------+---------+---------------+---------+------------+---------+
|    10 | Sam     | sam@ex.com    |       1 | 2024-05-01 |  150.00 |
|    10 | Sam     | sam@ex.com    |       3 | 2024-05-03 |  100.00 |
|    10 | Sam     | sam@ex.com    |      22 | 2024-05-22 | 1150.00 |
|    20 | Tom     | tom@ex.com    |       2 | 2024-05-02 |  200.00 |
|    30 | Ram     | ram@ex.com    |       4 | 2024-05-04 |  250.00 |
|    30 | Ram     | ram@ex.com    |      23 | 2024-05-23 | 1200.00 |
|    40 | Bob     | bob@ex.com    |       5 | 2024-05-05 |  300.00 |
|    40 | Bob     | bob@ex.com    |      24 | 2024-05-24 | 1250.00 |
|    50 | Suz     | suz@ex.com    |       6 | 2024-05-06 |  350.00 |
|    60 | Sasi    | sasi@ex.com   |       7 | 2024-05-07 |  400.00 |
|    60 | Sasi    | sasi@ex.com   |      25 | 2024-05-25 | 1300.00 |
|    70 | Aman    | aman@ex.com   |       8 | 2024-05-08 |  450.00 |
|    70 | Aman    | aman@ex.com   |      26 | 2024-05-26 | 1350.00 |
|    80 | Ravi    | ravi@ex.com   |       9 | 2024-05-09 |  500.00 |
|    80 | Ravi    | ravi@ex.com   |      27 | 2024-05-27 | 1400.00 |
|    90 | Roy     | roy@ex.com    |      10 | 2024-05-10 |  550.00 |
|   100 | Jake    | jake@ex.com   |      11 | 2024-05-11 |  600.00 |
|   100 | Jake    | jake@ex.com   |      28 | 2024-05-28 | 1450.00 |
|   110 | Sruthi  | sruthi@ex.com |    NULL | NULL       |    NULL |
|   120 | Amar    | amar@ex.com   |      13 | 2024-05-13 |  700.00 |
|   120 | Amar    | amar@ex.com   |      30 | 2024-05-30 | 1550.00 |
|   130 | Ravan   | ravan@ex.com  |      14 | 2024-05-14 |  750.00 |
|   130 | Ravan   | ravan@ex.com  |      31 | 2024-05-31 | 1600.00 |
|   140 | Krish   | krish@ex.com  |    NULL | NULL       |    NULL |
|   150 | Adam    | adam@ex.com   |      16 | 2024-05-16 |  850.00 |
|   160 | Vasu    | vasu@ex.com   |      17 | 2024-05-17 |  900.00 |
|   170 | Smith   | smith@ex.com  |      18 | 2024-05-18 |  950.00 |
|   180 | jay     | jay@ex.com    |    NULL | NULL       |    NULL |
|   190 | Arjun   | arjun@ex.com  |    NULL | NULL       |    NULL |
|   200 | Charan  | charan@ex.com |      21 | 2024-05-21 | 1100.00 |
+-------+---------+---------------+---------+------------+---------+
30 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.

Query 1:

Subquery to Find Customers with Orders Above the Average Order Value



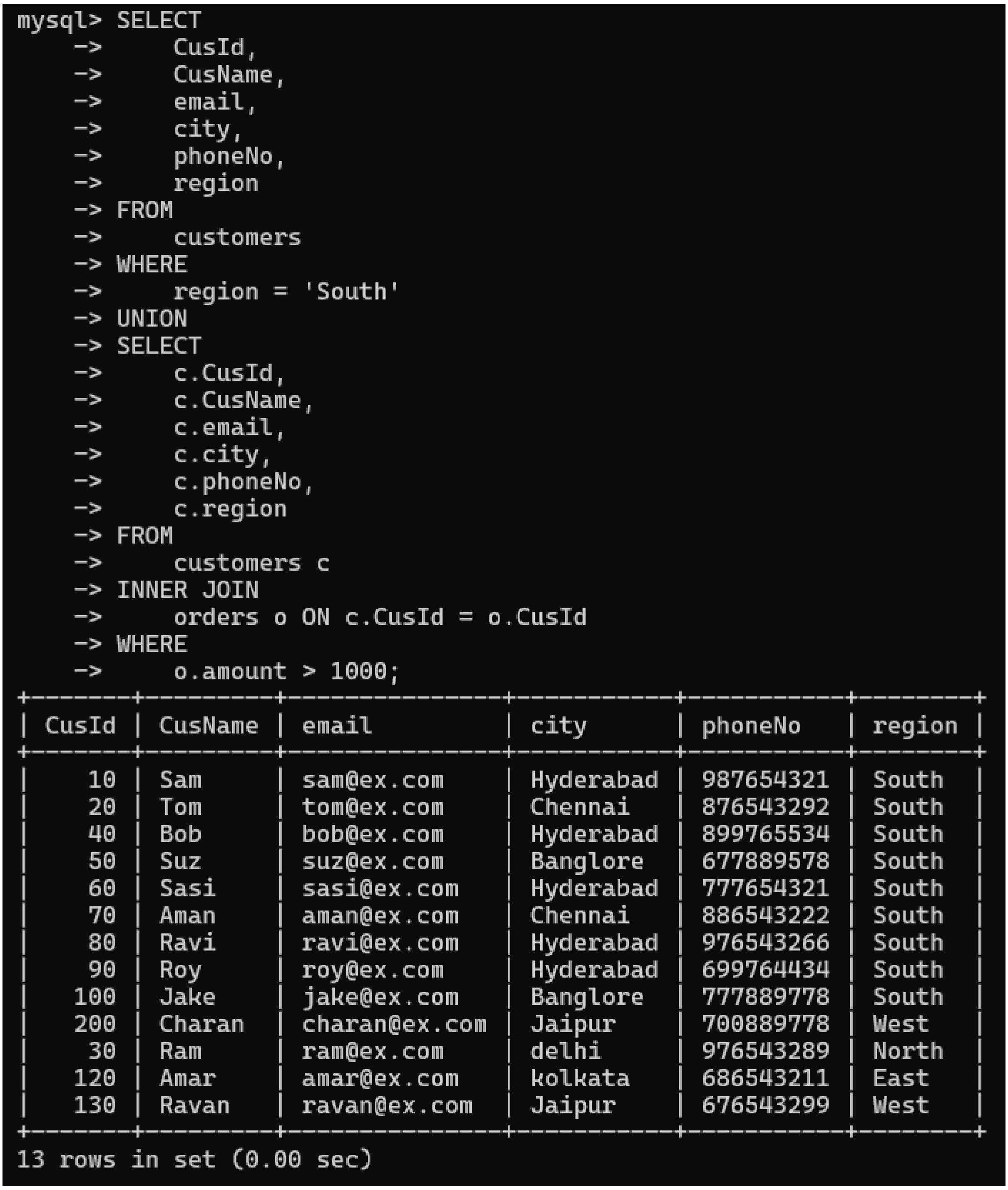
```
mysql> SELECT
    ->     customers.CusId,
    ->     customers.CusName,
    ->     customers.email,
    ->     orders.OrderId,
    ->     orders.OrderDate,
    ->     orders.Amount
    -> FROM
    ->     customers
    -> INNER JOIN
    ->     orders ON customers.CusId = orders.CusId
    -> WHERE
    ->     orders.amount > (SELECT AVG(amount) FROM orders);
+-------+---------+---------------+---------+------------+---------+
| CusId | CusName | email         | OrderId | OrderDate  | Amount  |
+-------+---------+---------------+---------+------------+---------+
|   150 | Adam    | adam@ex.com   |      16 | 2024-05-16 |  850.00 |
|   160 | Vasu    | vasu@ex.com   |      17 | 2024-05-17 |  900.00 |
|   170 | Smith   | smith@ex.com  |      18 | 2024-05-18 |  950.00 |
|   200 | Charan  | charan@ex.com |      21 | 2024-05-21 | 1100.00 |
|    10 | Sam     | sam@ex.com    |      22 | 2024-05-22 | 1150.00 |
|    30 | Ram     | ram@ex.com    |      23 | 2024-05-23 | 1200.00 |
|    40 | Bob     | bob@ex.com    |      24 | 2024-05-24 | 1250.00 |
|    60 | Sasi    | sasi@ex.com   |      25 | 2024-05-25 | 1300.00 |
|    70 | Aman    | aman@ex.com   |      26 | 2024-05-26 | 1350.00 |
|    80 | Ravi    | ravi@ex.com   |      27 | 2024-05-27 | 1400.00 |
|   100 | Jake    | jake@ex.com   |      28 | 2024-05-28 | 1450.00 |
|   120 | Amar    | amar@ex.com   |      30 | 2024-05-30 | 1550.00 |
|   130 | Ravan   | ravan@ex.com  |      31 | 2024-05-31 | 1600.00 |
+-------+---------+---------------+---------+------------+---------+
13 rows in set (0.00 sec)
```

Query 2:

UNION Query to Combine Two SELECT Statements

Let's assume we want to combine:

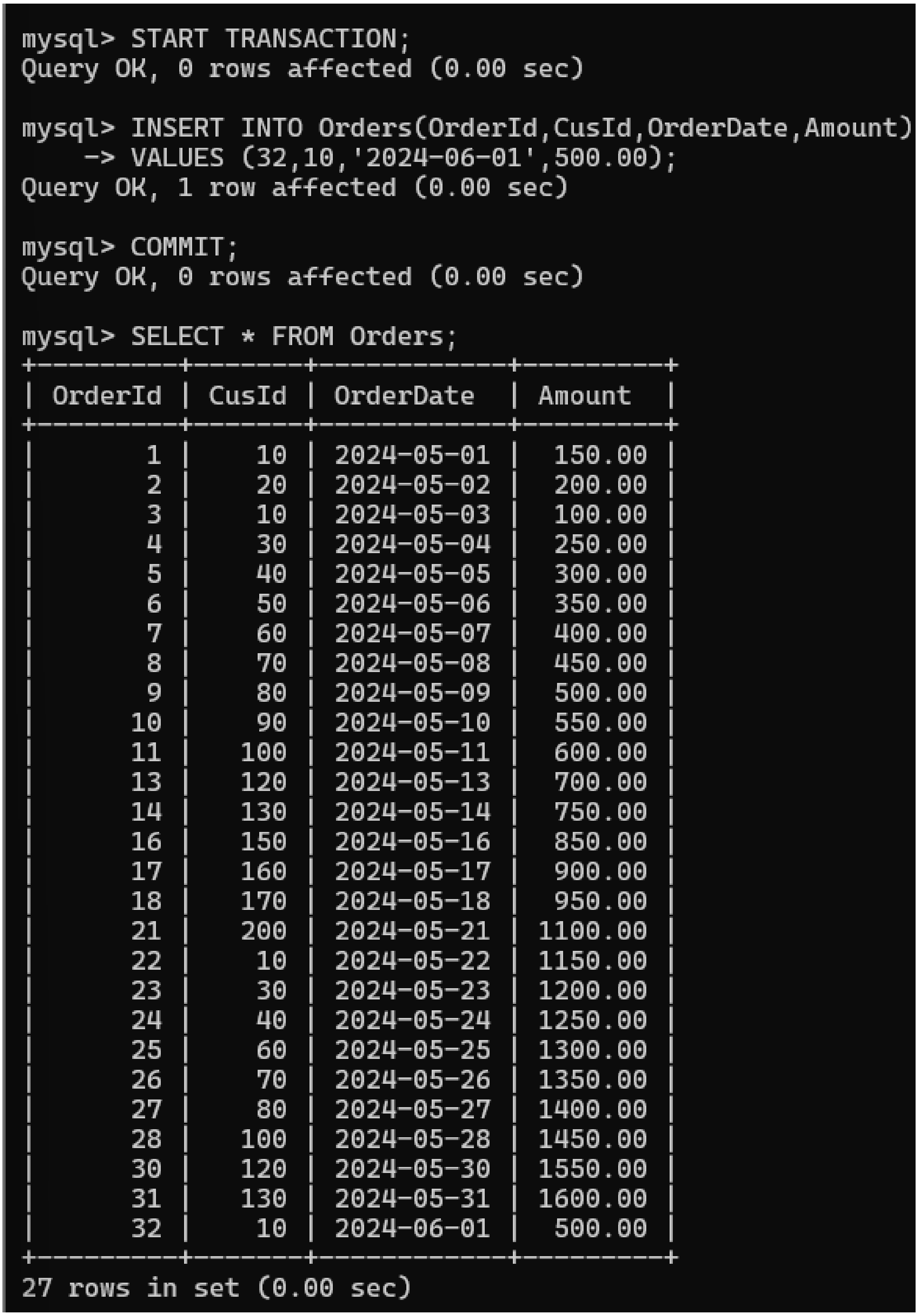
- A list of customers from the "South" region.
- A list of customers who have placed an order amount greater than $1000.



```
mysql> SELECT
    ->     CusId,
    ->     CusName,
    ->     email,
    ->     city,
    ->     phoneNo,
    ->     region
    -> FROM
    ->     customers
    -> WHERE
    ->     region = 'South'
    -> UNION
    -> SELECT
    ->     c.CusId,
    ->     c.CusName,
    ->     c.email,
    ->     c.city,
    ->     c.phoneNo,
    ->     c.region
    -> FROM
    ->     customers c
    -> INNER JOIN
    ->     orders o ON c.CusId = o.CusId
    -> WHERE
    ->     o.amount > 1000;
+-------+---------+---------------+-----------+-----------+--------+
| CusId | CusName | email         | city      | phoneNo   | region |
+-------+---------+---------------+-----------+-----------+--------+
|    10 | Sam     | sam@ex.com    | Hyderabad | 987654321 | South  |
|    20 | Tom     | tom@ex.com    | Chennai   | 876543292 | South  |
|    40 | Bob     | bob@ex.com    | Hyderabad | 899765534 | South  |
|    50 | Suz     | suz@ex.com    | Banglore  | 677889578 | South  |
|    60 | Sasi    | sasi@ex.com   | Hyderabad | 777654321 | South  |
|    70 | Aman    | aman@ex.com   | Chennai   | 886543222 | South  |
|    80 | Ravi    | ravi@ex.com   | Hyderabad | 976543266 | South  |
|    90 | Roy     | roy@ex.com    | Hyderabad | 699764434 | South  |
|   100 | Jake    | jake@ex.com   | Banglore  | 777889778 | South  |
|   200 | Charan  | charan@ex.com | Jaipur    | 700889778 | West   |
|    30 | Ram     | ram@ex.com    | delhi     | 976543289 | North  |
|   120 | Amar    | amar@ex.com   | kolkata   | 686543211 | East   |
|   130 | Ravan   | ravan@ex.com  | Jaipur    | 676543299 | West   |
+-------+---------+---------------+-----------+-----------+--------+
13 rows in set (0.00 sec)
```

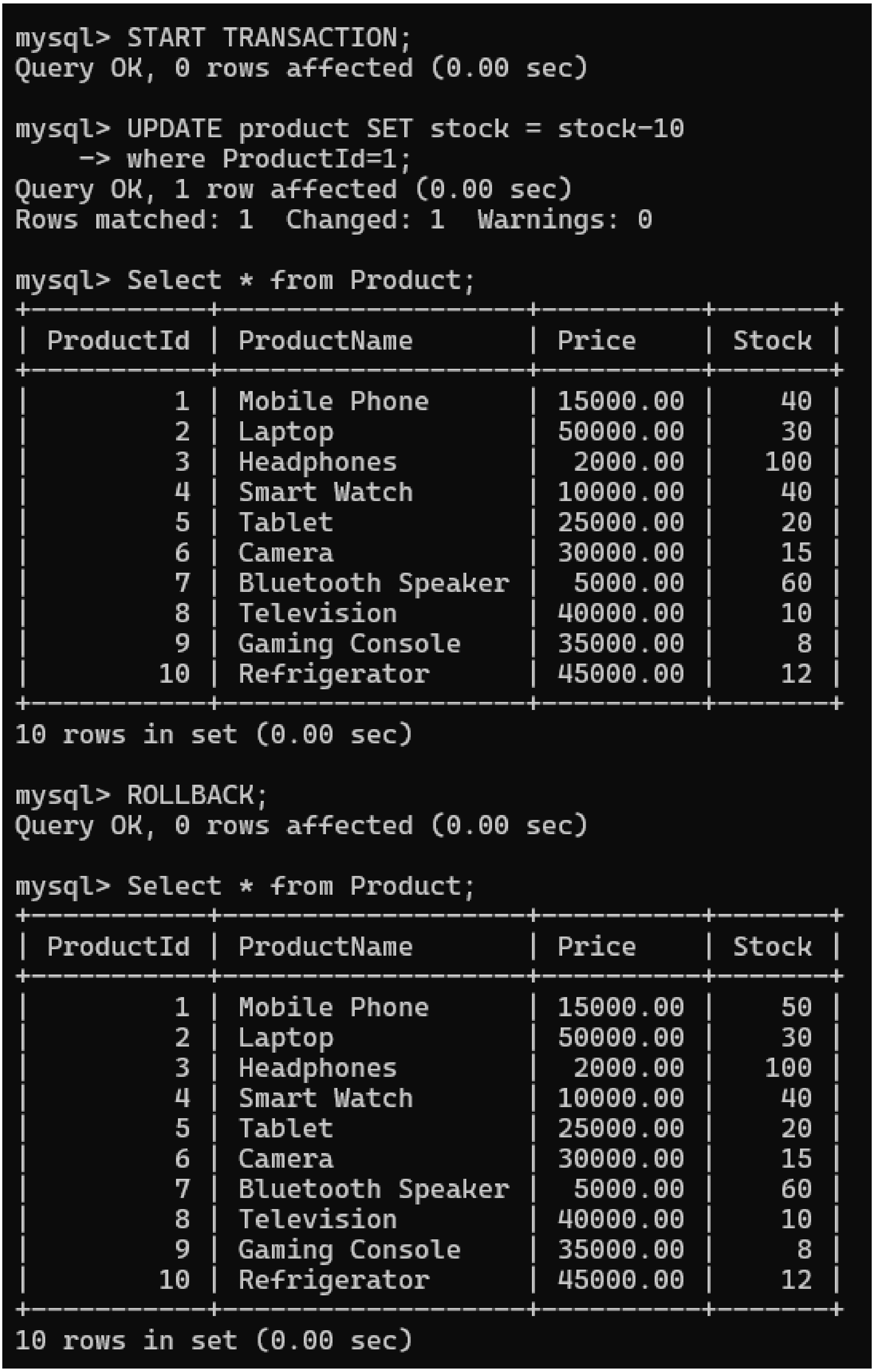
Assignment 4: Compose SQL statements to BEGIN a transaction, INSERT a new record into the 'orders' table, COMMIT the transaction, then UPDATE the 'products' table, and ROLLBACK the transaction.

Step 1: BEGIN a transaction and INSERT a new record into the 'orders' table, then COMMIT the transaction



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

mysql> INSERT INTO Orders(OrderId,CusId,OrderDate,Amount)
    -> VALUES (32,10,'2024-06-01',500.00);
Query OK, 1 row affected (0.00 sec)

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

mysql> SELECT * FROM Orders;
+---------+-------+------------+---------+
| OrderId | CusId | OrderDate  | Amount  |
+---------+-------+------------+---------+
|       1 |    10 | 2024-05-01 |  150.00 |
|       2 |    20 | 2024-05-02 |  200.00 |
|       3 |    10 | 2024-05-03 |  100.00 |
|       4 |    30 | 2024-05-04 |  250.00 |
|       5 |    40 | 2024-05-05 |  300.00 |
|       6 |    50 | 2024-05-06 |  350.00 |
|       7 |    60 | 2024-05-07 |  400.00 |
|       8 |    70 | 2024-05-08 |  450.00 |
|       9 |    80 | 2024-05-09 |  500.00 |
|      10 |    90 | 2024-05-10 |  550.00 |
|      11 |   100 | 2024-05-11 |  600.00 |
|      13 |   120 | 2024-05-13 |  700.00 |
|      14 |   130 | 2024-05-14 |  750.00 |
|      16 |   150 | 2024-05-16 |  850.00 |
|      17 |   160 | 2024-05-17 |  900.00 |
|      18 |   170 | 2024-05-18 |  950.00 |
|      21 |   200 | 2024-05-21 | 1100.00 |
|      22 |    10 | 2024-05-22 | 1150.00 |
|      23 |    30 | 2024-05-23 | 1200.00 |
|      24 |    40 | 2024-05-24 | 1250.00 |
|      25 |    60 | 2024-05-25 | 1300.00 |
|      26 |    70 | 2024-05-26 | 1350.00 |
|      27 |    80 | 2024-05-27 | 1400.00 |
|      28 |   100 | 2024-05-28 | 1450.00 |
|      30 |   120 | 2024-05-30 | 1550.00 |
|      31 |   130 | 2024-05-31 | 1600.00 |
|      32 |    10 | 2024-06-01 |  500.00 |
+---------+-------+------------+---------+
27 rows in set (0.00 sec)
```

Step 2: BEGIN another transaction, UPDATE the 'products' table, and ROLLBACK the transaction

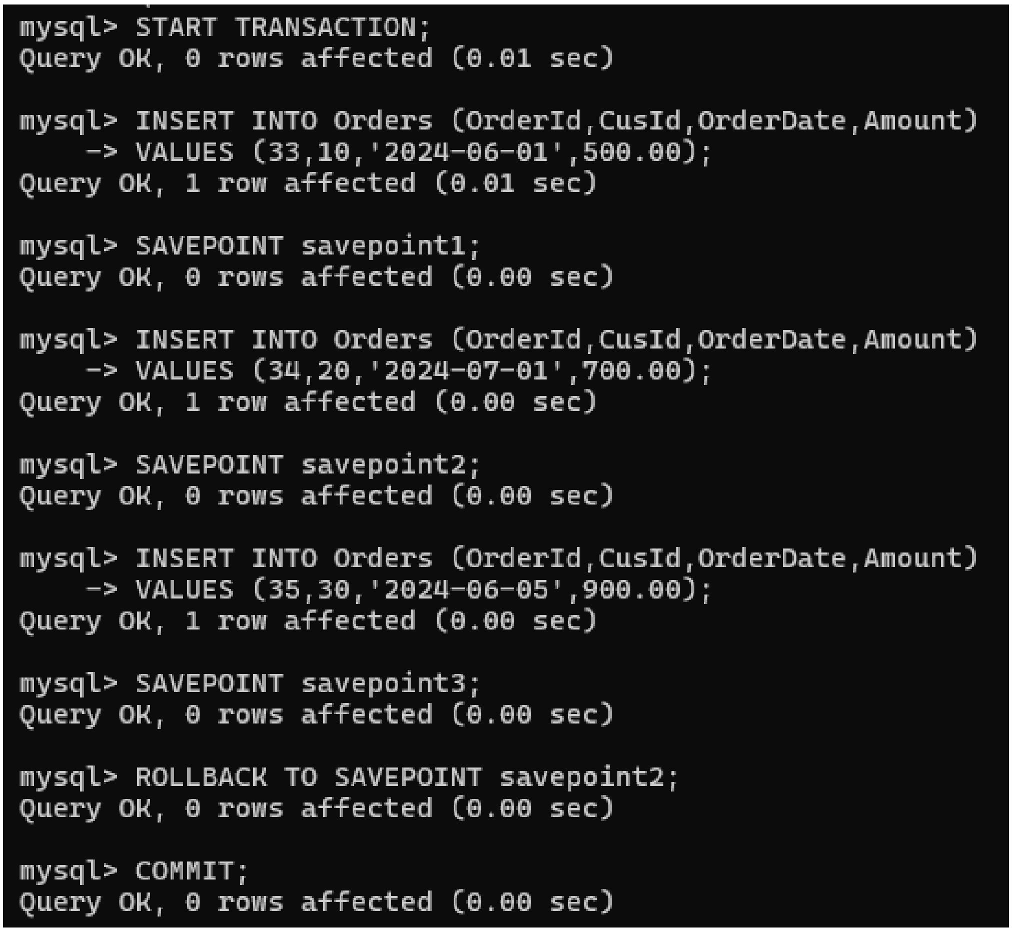


```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE product SET stock = stock-10
    -> where ProductId=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> Select * from Product;
+-----------+-------------------+----------+-------+
| ProductId | ProductName       | Price    | Stock |
+-----------+-------------------+----------+-------+
|         1 | Mobile Phone      | 15000.00 |    40 |
|         2 | Laptop            | 50000.00 |    30 |
|         3 | Headphones        |  2000.00 |   100 |
|         4 | Smart Watch       | 10000.00 |    40 |
|         5 | Tablet            | 25000.00 |    20 |
|         6 | Camera            | 30000.00 |    15 |
|         7 | Bluetooth Speaker |  5000.00 |    60 |
|         8 | Television        | 40000.00 |    10 |
|         9 | Gaming Console    | 35000.00 |     8 |
|        10 | Refrigerator      | 45000.00 |    12 |
+-----------+-------------------+----------+-------+
10 rows in set (0.00 sec)

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

mysql> Select * from Product;
+-----------+-------------------+----------+-------+
| ProductId | ProductName       | Price    | Stock |
+-----------+-------------------+----------+-------+
|         1 | Mobile Phone      | 15000.00 |    50 |
|         2 | Laptop            | 50000.00 |    30 |
|         3 | Headphones        |  2000.00 |   100 |
|         4 | Smart Watch       | 10000.00 |    40 |
|         5 | Tablet            | 25000.00 |    20 |
|         6 | Camera            | 30000.00 |    15 |
|         7 | Bluetooth Speaker |  5000.00 |    60 |
|         8 | Television        | 40000.00 |    10 |
|         9 | Gaming Console    | 35000.00 |     8 |
|        10 | Refrigerator      | 45000.00 |    12 |
+-----------+-------------------+----------+-------+
10 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.

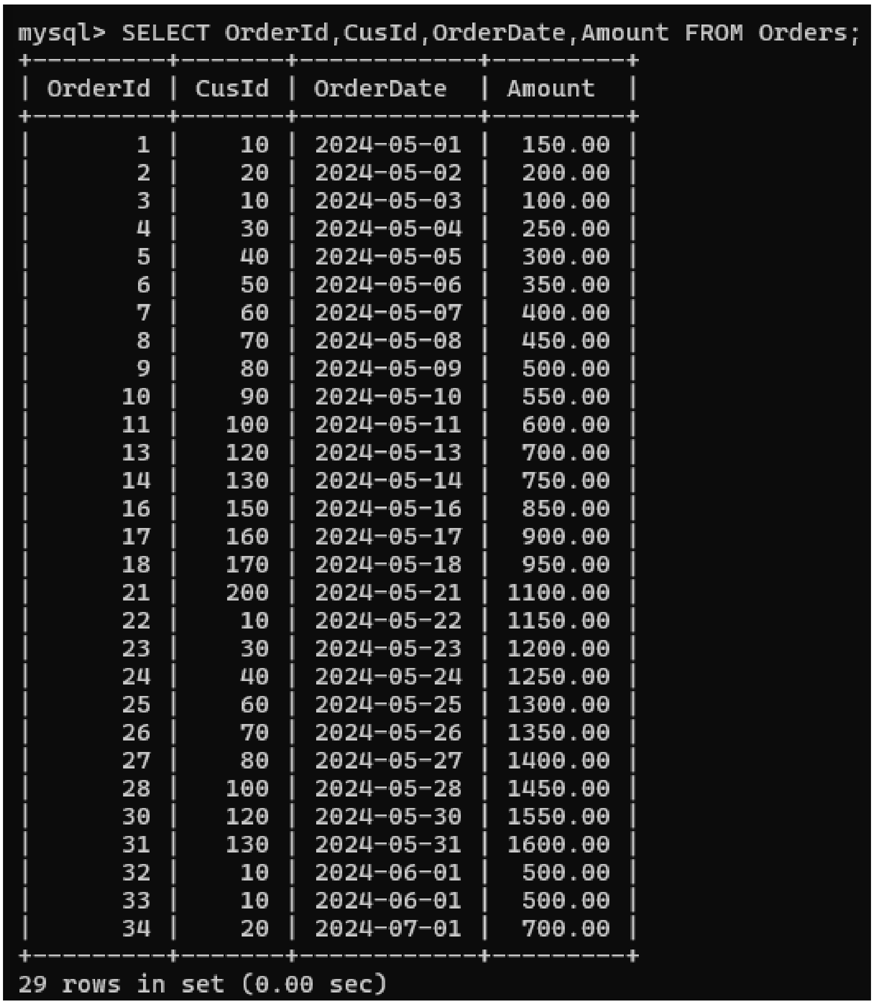
Steps :

- Begins a transaction using the **START**; statement.
- Inserts three orders into the 'orders' table, setting a **SAVEPOINT** after each **INSERT** operation.
- Rolls back to the second **SAVEPOINT** (savepoint2).
- Finally, commits the overall transaction using the **COMMIT**; statement.



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

mysql> INSERT INTO Orders (OrderId,CusId,OrderDate,Amount)
    -> VALUES (33,10,'2024-06-01',500.00);
Query OK, 1 row affected (0.01 sec)

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

mysql> INSERT INTO Orders (OrderId,CusId,OrderDate,Amount)
    -> VALUES (34,20,'2024-07-01',700.00);
Query OK, 1 row affected (0.00 sec)

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

mysql> INSERT INTO Orders (OrderId,CusId,OrderDate,Amount)
    -> VALUES (35,30,'2024-06-05',900.00);
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)
```

In the picture below, you can see that the first two insertions are done, but the third insertion is rolled back. That's why you can't see the OrderId with 35.



```
mysql> SELECT OrderId,CusId,OrderDate,Amount FROM Orders;
+---------+-------+------------+---------+
| OrderId | CusId | OrderDate  | Amount  |
+---------+-------+------------+---------+
|       1 |    10 | 2024-05-01 |  150.00 |
|       2 |    20 | 2024-05-02 |  200.00 |
|       3 |    10 | 2024-05-03 |  100.00 |
|       4 |    30 | 2024-05-04 |  250.00 |
|       5 |    40 | 2024-05-05 |  300.00 |
|       6 |    50 | 2024-05-06 |  350.00 |
|       7 |    60 | 2024-05-07 |  400.00 |
|       8 |    70 | 2024-05-08 |  450.00 |
|       9 |    80 | 2024-05-09 |  500.00 |
|      10 |    90 | 2024-05-10 |  550.00 |
|      11 |   100 | 2024-05-11 |  600.00 |
|      13 |   120 | 2024-05-13 |  700.00 |
|      14 |   130 | 2024-05-14 |  750.00 |
|      16 |   150 | 2024-05-16 |  850.00 |
|      17 |   160 | 2024-05-17 |  900.00 |
|      18 |   170 | 2024-05-18 |  950.00 |
|      21 |   200 | 2024-05-21 | 1100.00 |
|      22 |    10 | 2024-05-22 | 1150.00 |
|      23 |    30 | 2024-05-23 | 1200.00 |
|      24 |    40 | 2024-05-24 | 1250.00 |
|      25 |    60 | 2024-05-25 | 1300.00 |
|      26 |    70 | 2024-05-26 | 1350.00 |
|      27 |    80 | 2024-05-27 | 1400.00 |
|      28 |   100 | 2024-05-28 | 1450.00 |
|      30 |   120 | 2024-05-30 | 1550.00 |
|      31 |   130 | 2024-05-31 | 1600.00 |
|      32 |    10 | 2024-06-01 |  500.00 |
|      33 |    10 | 2024-06-01 |  500.00 |
|      34 |    20 | 2024-07-01 |  700.00 |
+---------+-------+------------+---------+
29 rows in set (0.00 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.

Report on the Use of Transaction Logs for Data Recovery :

Transaction logs play a vital role in ensuring data integrity and facilitating recovery in the event of system failures or unexpected shutdowns. These logs record all changes made to the database during transactions, providing a detailed history of data modifications.

Scenario: Imagine a scenario where an online retail company experiences a sudden power outage during a busy sales period. As a result, the database server abruptly shuts down, leading to potential data corruption and loss. However, due to the presence of transaction logs, the company can recover the lost data efficiently.

Utilization of Transaction Logs:

1. **Recovery Point:** Transaction logs serve as a recovery point, allowing the database to be restored to a specific point in time before the failure occurred.
2. **Redo and Undo Operations:** The transaction logs contain both redo and undo information. Redo logs help reapply committed transactions that were not yet written to disk before the failure. Undo logs assist in rolling back uncommitted or partially committed transactions to maintain data consistency.
3. **Consistency Check:** Before applying the redo and undo operations, the integrity of the transaction logs is verified to ensure their accuracy and completeness.

Conclusion: In conclusion, transaction logs are indispensable for data recovery in the event of unexpected system failures. By capturing all database modifications in real-time, these logs provide a reliable mechanism for restoring data integrity and minimizing downtime, thereby ensuring business continuity.