Name:Vaishnavi Vinod Ingole

Mail:vaishnavingole54@gmail.com

"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.

Solution:

select * from customers;

select customer_name,email from customers where city="phoenix";

```
49
50 •   select * from customers;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

customer_id	customer_name	email	city	region
1	Amy Jackson	amy@mail.com	New York	East
2	Robin Scherbatsky	robin@mail.com	Los Angeles	West
3	Ted Mosby	ted@mail.com	Chicago	Midwest
4	Leonard Nimoy	leonard@mail.com	Houston	South
5	Phil Dunphy	phil@mail.com	Phoenix	West
6	alex Dunphy	alex@mail.com	Phoenix	West
NULL	NULL	NULL	NULL	NULL

```
49
50 •   select * from customers;
51 •   select customer_name,email from customers where city="phoenix";
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_name	email
Phil Dunphy	phil@mail.com
alex Dunphy	alex@mail.com

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.

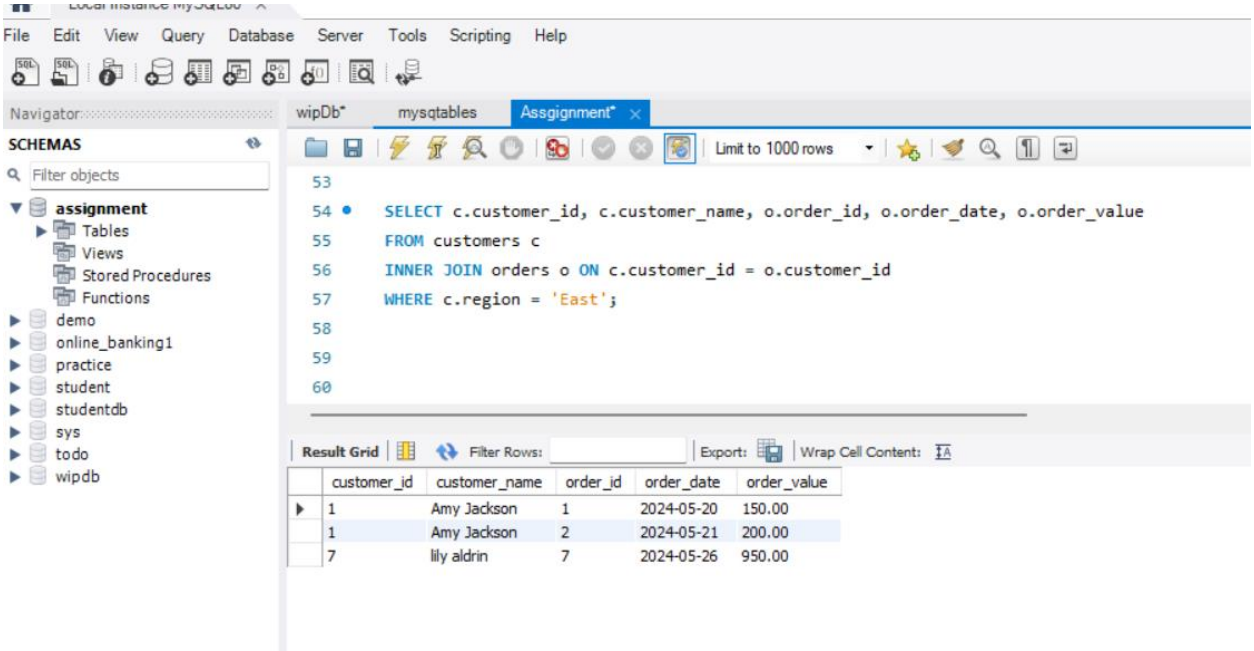
Solution:

SELECT c.customer_id, c.customer_name, o.order_id, o.order_date, o.order_value

FROM customers c

INNER JOIN orders o ON c.customer_id = o.customer_id

WHERE c.region = 'East';



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

FROM customers

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

Filter objects

- assignment
  - Tables
  - Views
  - Stored Procedures
  - Functions
- demo
- online_banking1
- practice
- student
- studentdb
- sys
- todo
- wipdb

```
WHERE c.region = 'East';


SELECT customers.customer_id, customers.customer_name, orders.o
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	customer_name	order_id	order_date	order_value
1	Amy Jackson	1	2024-05-20	150.00
1	Amy Jackson	2	2024-05-21	200.00
2	Robin Scherbatsky	3	2024-05-22	250.00
3	Ted Mosby	4	2024-05-23	300.00
4	Leonard Nimoy	5	2024-05-24	350.00
5	Phil Dunphy	NULL	NULL	NULL
6	alex Dunphy	6	2024-05-24	350.00
7	lily aldrin	7	2024-05-26	950.00

Result Grid
Form Editor
Field Types

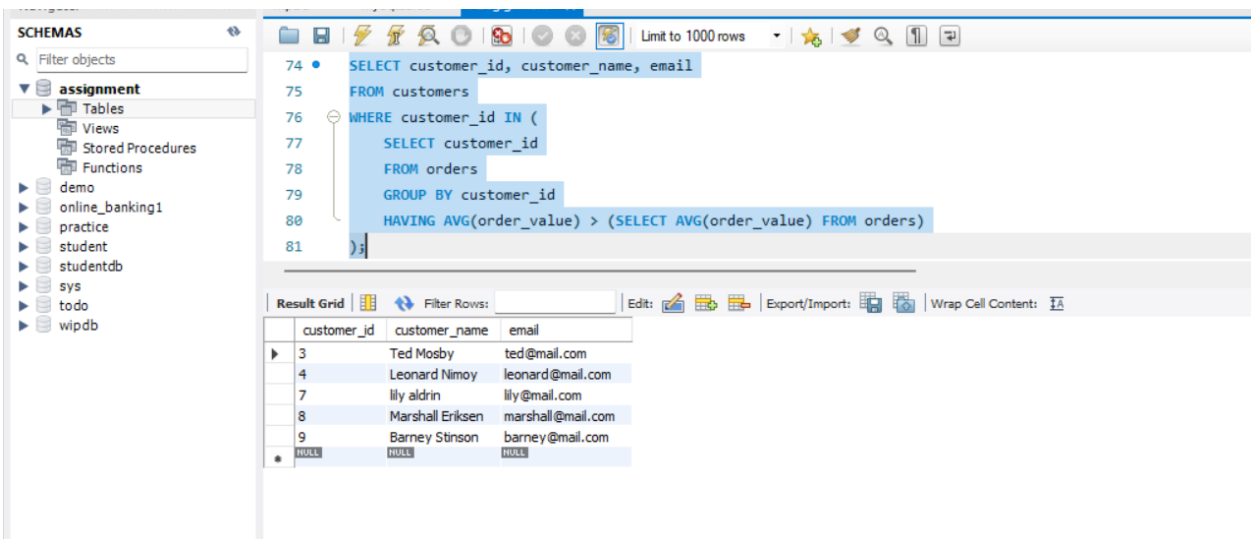
Result 2 ×

Read Only

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.

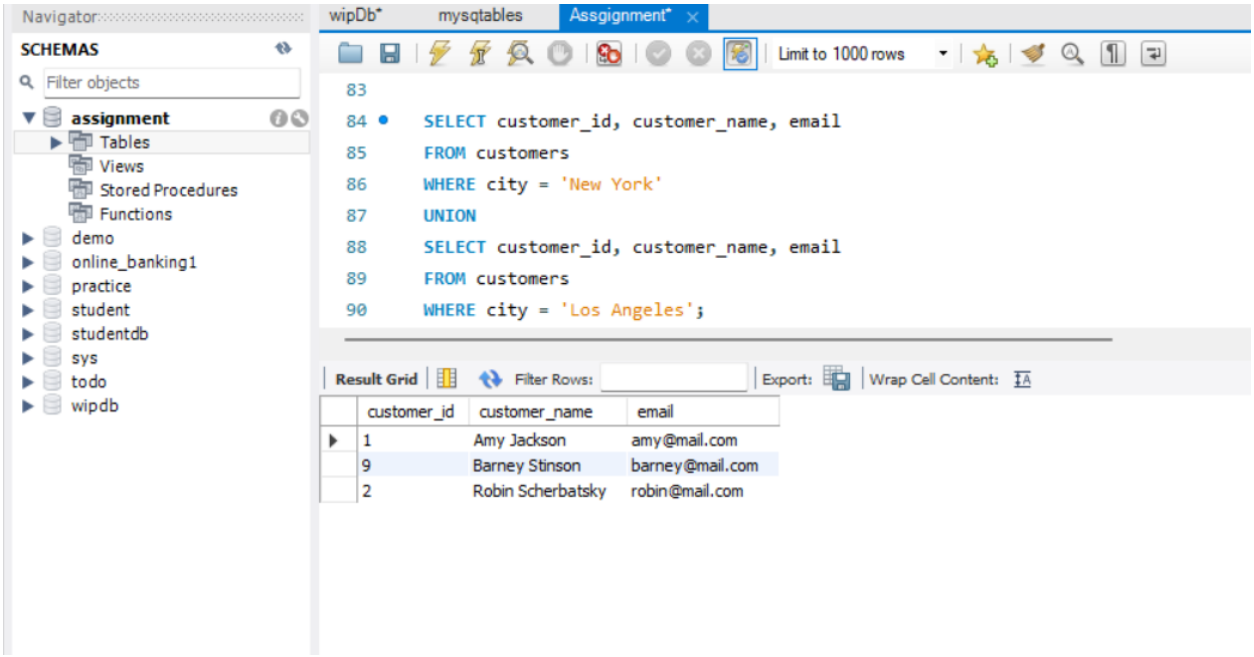
Solution:

```
SELECT customer_id, customer_name, email
FROM customers
WHERE customer_id IN (
    SELECT customer_id
    FROM orders
    GROUP BY customer_id
    HAVING AVG(order_value) > (SELECT AVG(order_value) FROM orders)
);
```



```
SELECT customer_id, customer_name, email
FROM customers
WHERE city = 'New York'
UNION
SELECT customer_id, customer_name, email
FROM customers
```

WHERE city = 'Los Angeles';



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.

use Assignment;

StartTRANSACTION;

INSERT INTO orders (order_id, customer_id, order_date, order_value)

VALUES (14, 9, '2024-06-01', 1000.00);

COMMIT;

```
Start TRANSACTION;


INSERT INTO orders (order_id, customer_id, order_date, order_value)
VALUES (14, 9, '2024-06-01', 1000.00);

COMMIT;
```

Output

Action Output

#	Time	Action	Message
1	08:36:01	Start TRANSACTION	0 row(s) affected
2	08:36:11	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (14, 9, '2024-06-01', 1000.00)	Error Code: 1046. No database selected Select the default DB to be used
3	08:36:21	use assignment	0 row(s) affected
4	08:36:25	Start TRANSACTION	0 row(s) affected
5	08:36:30	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (14, 9, '2024-06-01', 1000.00)	1 row(s) affected
6	08:36:35	COMMIT	0 row(s) affected

Start TRANSACTION;

UPDATE products

SET price = price + 100

WHERE product_id = 1;

ROLLBACK;

```
Start TRANSACTION;


UPDATE products
SET price = price + 100
WHERE product_id = 1;
select* from products;
```

Result Grid | Filter Rows: | Edit:

product_id	product_name	price	stock
1	Laptop	1099.99	50
2	Smart TV	799.99	30
3	AirPods	199.99	150
4	iPad	499.99	75
5	Headphones	99.99	200
NULL	NULL	NULL	NULL

roducts 1

Output

Action Output

#	Time	Action
4	08:36:25	Start TRANSACTION
5	08:36:30	INSERT INTO orders (order_id, customer_id, order_date, orde
6	08:36:35	COMMIT
7	08:37:40	Start TRANSACTION
8	08:37:47	UPDATE products SET price = price + 100 WHERE product_
9	08:38:01	select* from products LIMIT 0, 1000

```
117
118 •  ROLLBACK;
119
120 •  select* from products;
121
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

product_id	product_name	price	stock
1	Laptop	999.99	50
2	Smart TV	799.99	30
3	AirPods	199.99	150
4	iPad	499.99	75
5	Headphones	99.99	200
NULL	NULL	NULL	NULL

products 2

Output

Action Output

#	Time	Action	Message
6	08:36:35	COMMIT	0 row(s) affected
7	08:37:40	Start TRANSACTION	0 row(s) affected
8	08:37:47	UPDATE products SET price = price + 100 WHERE product_id = 1	1 row(s) affected Rows matched: 1
9	08:38:01	select* from products LIMIT 0, 1000	5 row(s) returned
10	08:38:42	ROLLBACK	0 row(s) affected
11	08:38:53	select* from products LIMIT 0, 1000	5 row(s) returned

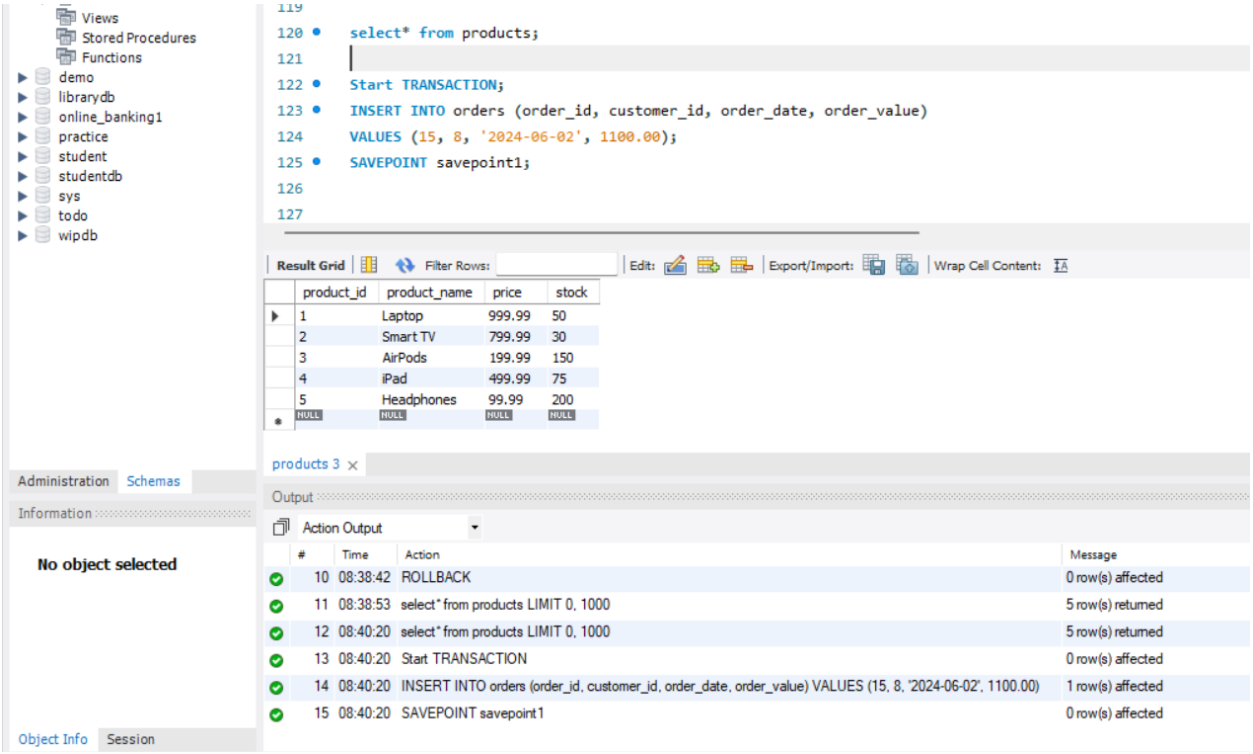
Assignment 5: Begin a transaction, perform a series of INSERTs into 'orders', setting a SAVEPOINT after each, rollback to the second SAVEPOINT, and COMMIT the overall transaction.

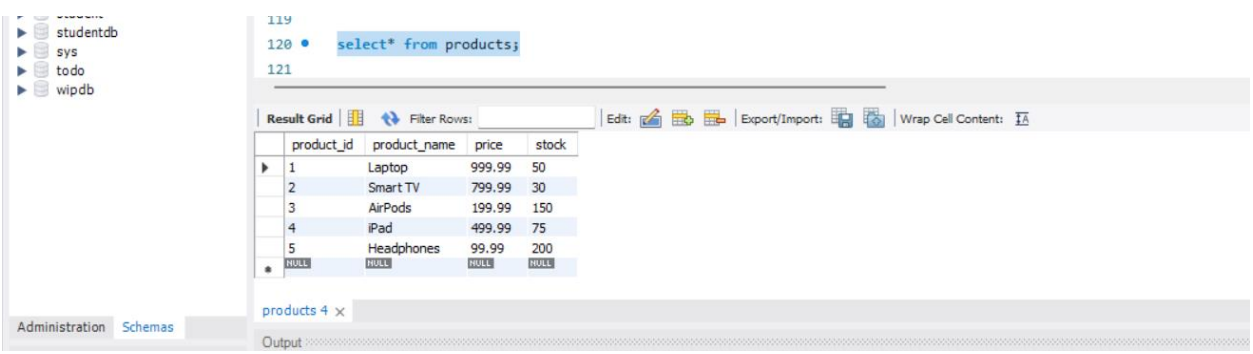
start TRANSACTION;

INSERT INTO orders (order_id, customer_id, order_date, order_value)

VALUES (15, 8, '2024-06-02', 1100.00);

SAVEPOINT savepoint1;

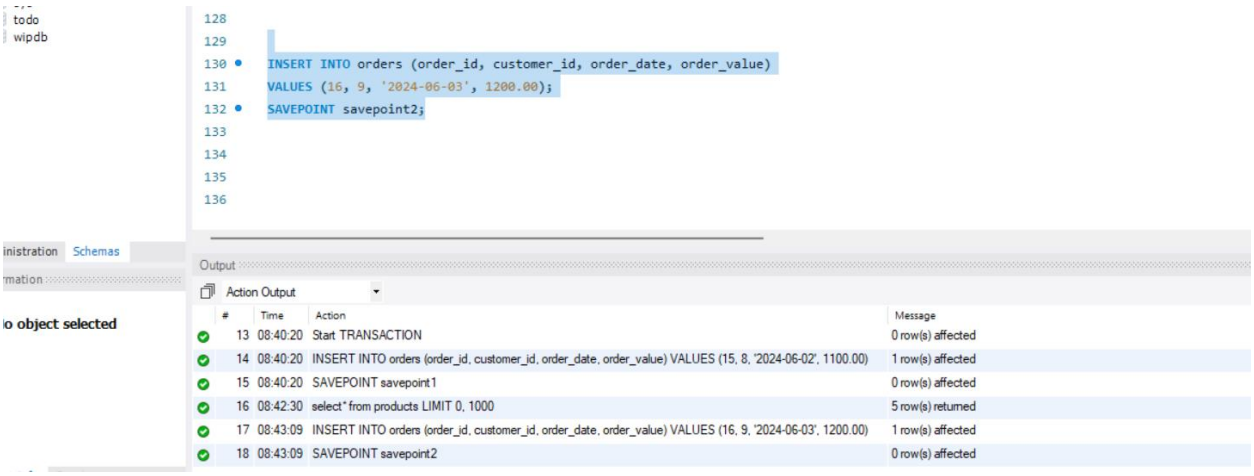


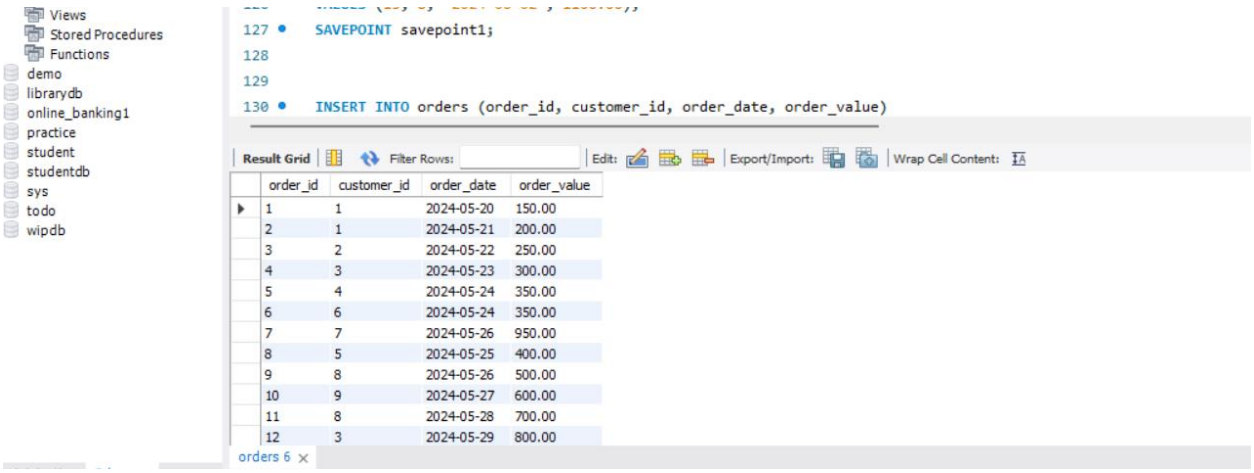


INSERT INTO orders (order_id, customer_id, order_date, order_value)

VALUES (16, 9, '2024-06-03', 1200.00);

SAVEPOINT savepoint2;

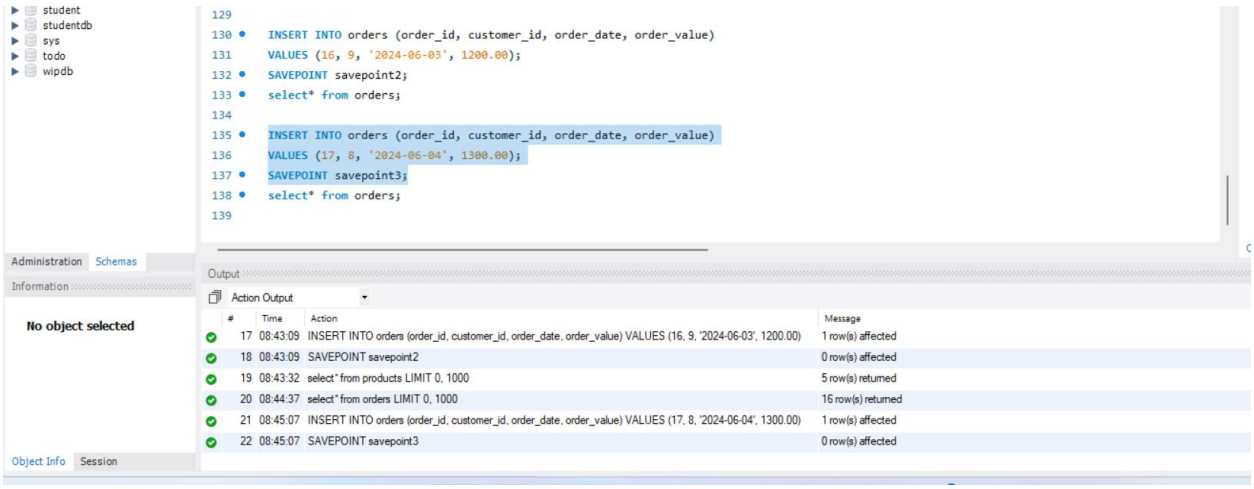


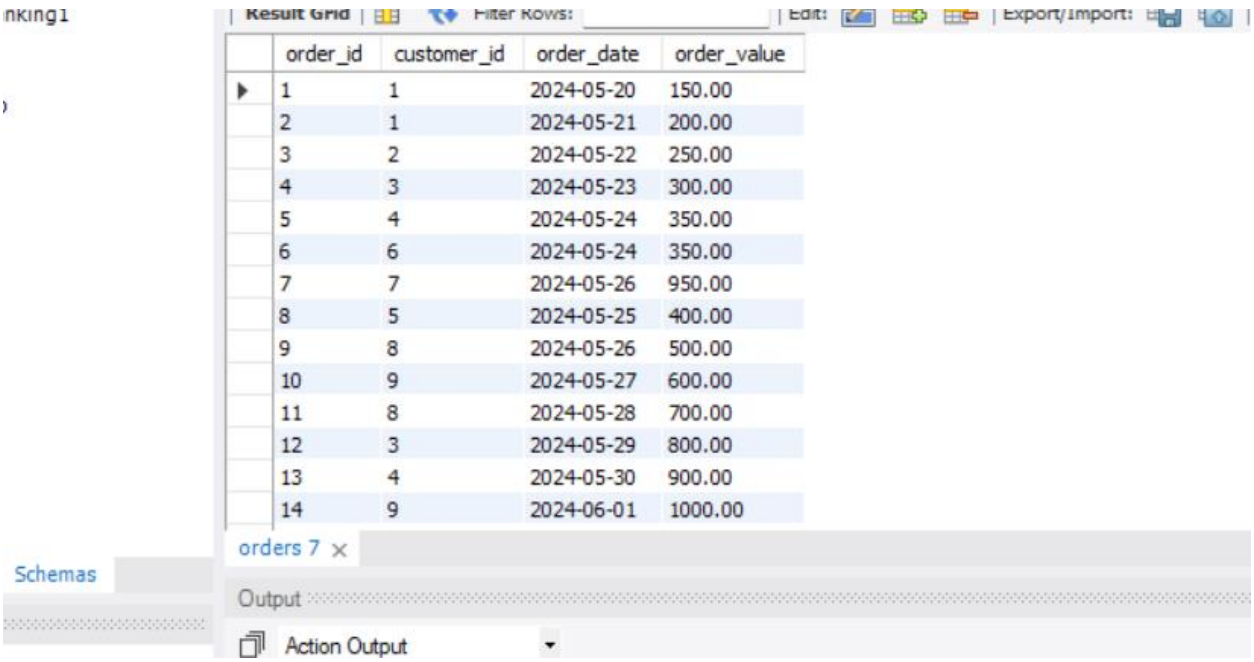


INSERT INTO orders (order_id, customer_id, order_date, order_value)

VALUES (17, 8, '2024-06-04', 1300.00);

SAVEPOINT savepoint3;





INSERT INTO orders (order_id, customer_id, order_date, order_value)

VALUES (18, 9, '2024-06-05', 1400.00);

SAVEPOINT savepoint4;

```
SAVEPOINT savepoint3;
select* from orders;

INSERT INTO orders (order_id, customer_id, order_date, order_value)
VALUES (18, 9, '2024-06-05', 1400.00);
SAVEPOINT savepoint4;
```

Output

Action Output

#	Time	Action	Message
20	08:44:37	select* from orders LIMIT 0, 1000	16 row(s) returned
21	08:45:07	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (17, 8, '2024-06-04', 1300.00)	1 row(s) affected
22	08:45:07	SAVEPOINT savepoint3	0 row(s) affected
23	08:45:25	select* from orders LIMIT 0, 1000	17 row(s) returned
24	08:46:02	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (18, 9, '2024-06-05', 1400.00)	1 row(s) affected
25	08:46:02	SAVEPOINT savepoint4	0 row(s) affected

order_id	customer_id	order_date	order_value
1	1	2024-05-20	150.00
2	1	2024-05-21	200.00
3	2	2024-05-22	250.00
4	3	2024-05-23	300.00
5	4	2024-05-24	350.00
6	6	2024-05-24	350.00
7	7	2024-05-26	950.00
8	5	2024-05-25	400.00
9	8	2024-05-26	500.00
10	9	2024-05-27	600.00
11	8	2024-05-28	700.00
12	3	2024-05-29	800.00
13	4	2024-05-30	900.00
14	9	2024-06-01	1000.00

ROLLBACK TO SAVEPOINT savepoint2;

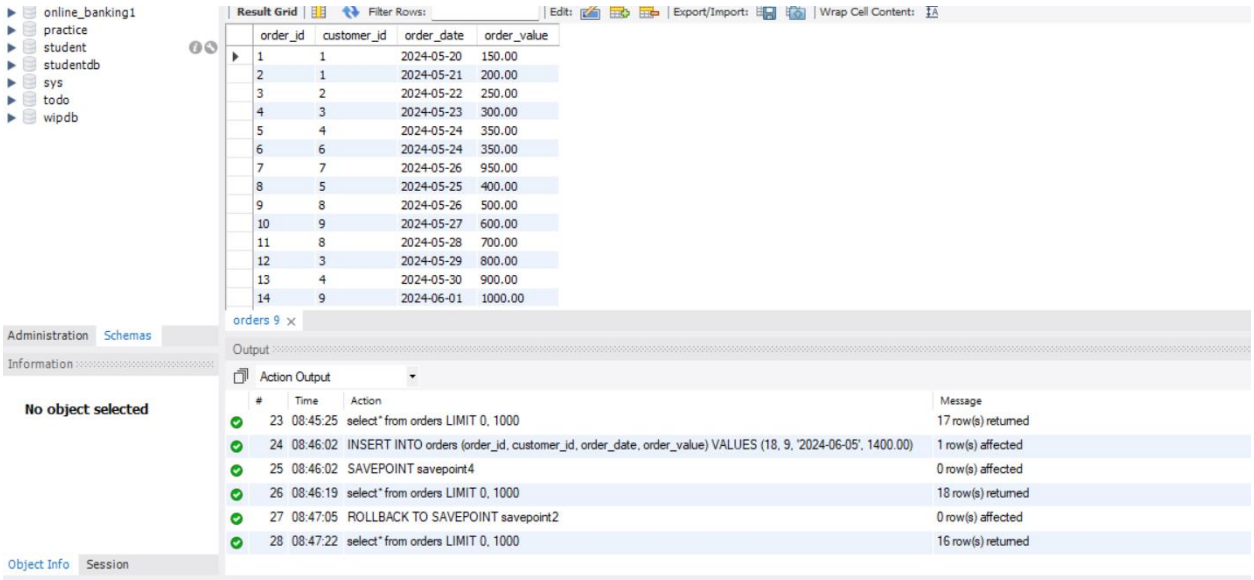
```
SAVEPOINT savepoint4;

ROLLBACK TO SAVEPOINT savepoint2;
```

Output

Action Output

#	Time	Action
22	08:45:07	SAVEPOINT savepoint3
23	08:45:25	select* from orders LIMIT 0, 1000
24	08:46:02	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (18, 9, '2024-06-05', 1400.00)
25	08:46:02	SAVEPOINT savepoint4
26	08:46:19	select* from orders LIMIT 0, 1000
27	08:47:05	ROLLBACK TO SAVEPOINT 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.

Solution:

Transaction logs are crucial components in database management systems (DBMS) that help ensure data integrity and enable recovery from unexpected failures. They record all changes made to the database, including inserts, updates, deletes, and transaction controls such as commits and rollbacks.

Importance of Transaction Logs

Data Integrity: Transaction logs ensure that all database operations are completed successfully. If a system failure occurs, the transaction log can help identify incomplete transactions and roll back or complete them to maintain data consistency.

Recovery from Failures: In the event of a system crash, power failure, or unexpected shutdown, transaction logs provide a way to restore the database to its last consistent state.

Audit Trail: Transaction logs keep a detailed record of all database operations, which can be used for auditing purposes to track changes and identify unauthorized access or modifications.

How Transaction Logs Work

Recording Transactions: Every database operation (insert, update, delete) is recorded in the transaction log before being applied to the database. Each log entry includes details like the transaction ID, type of operation, affected data, and timestamps.

Commit and Rollback: When a transaction is committed, a commit record is added to the log, indicating that the changes are permanent. If a rollback is issued, a rollback record is added, and any changes made by the transaction are undone.

Crash Recovery: After a crash, the DBMS uses the transaction log to determine the state of active transactions at the time of the failure. It replays committed transactions to ensure they are fully applied and rolls back uncommitted transactions to restore the database to a consistent state.

Hypothetical Scenario: Data Recovery After an Unexpected Shutdown

Imagine an online store with a database that handles customer orders. During a busy shopping day, the database server experiences an unexpected shutdown due to a power failure. Here's how transaction logs can help recover the database:

Before the Shutdown: Several transactions are in progress. Customers are placing orders (inserts), updating their shipping addresses (updates), and some transactions are being committed while others are pending.

Unexpected Shutdown: The power failure causes an abrupt shutdown, leaving some transactions incomplete. The database is now in an inconsistent state, with some operations partially applied.

Restart and Recovery:

Analyzing the Log: When the server restarts, the DBMS reads the transaction log to identify all transactions that were active at the time of the shutdown.

Rolling Forward: The DBMS replays all the committed transactions from the log, ensuring that their changes are fully applied to the database.

Rolling Back: It then identifies transactions that were not committed and rolls back their changes to undo any partial operations.

Restored State: The database is now restored to its last consistent state, with all completed transactions applied and incomplete transactions undone, ensuring data integrity and consistency.