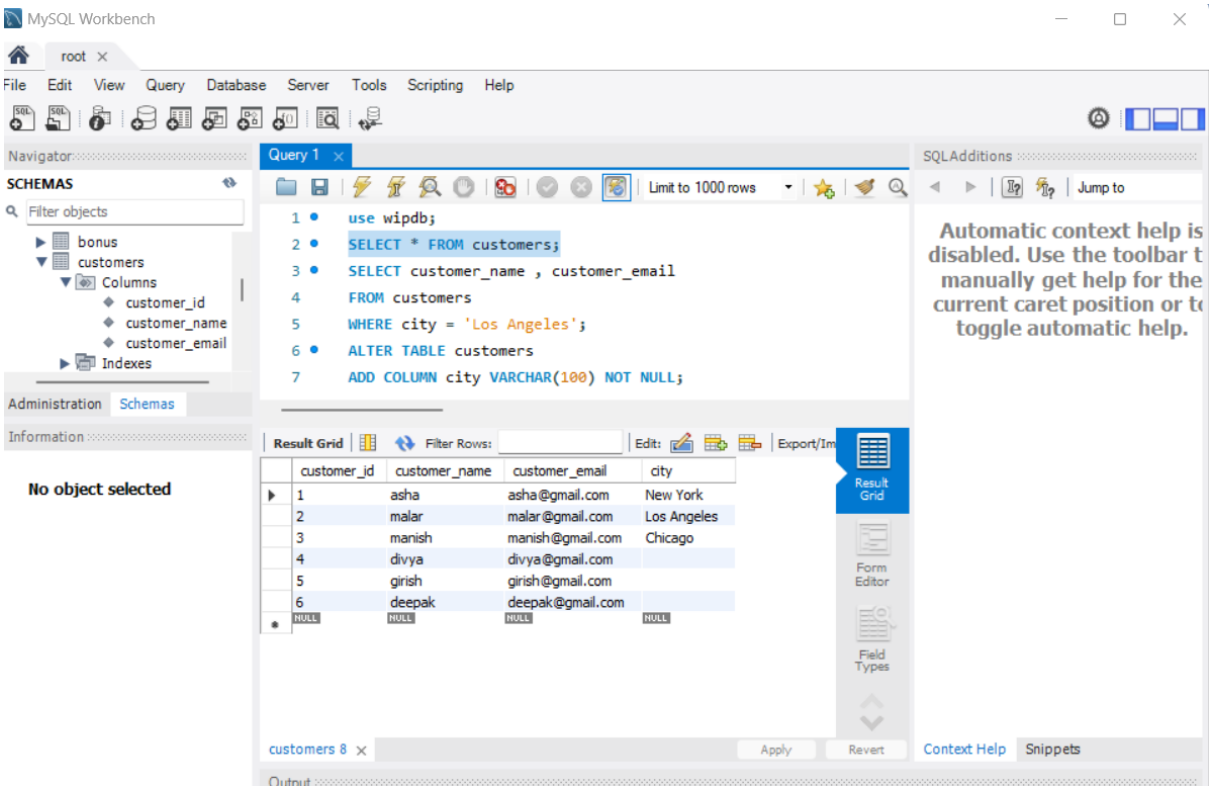
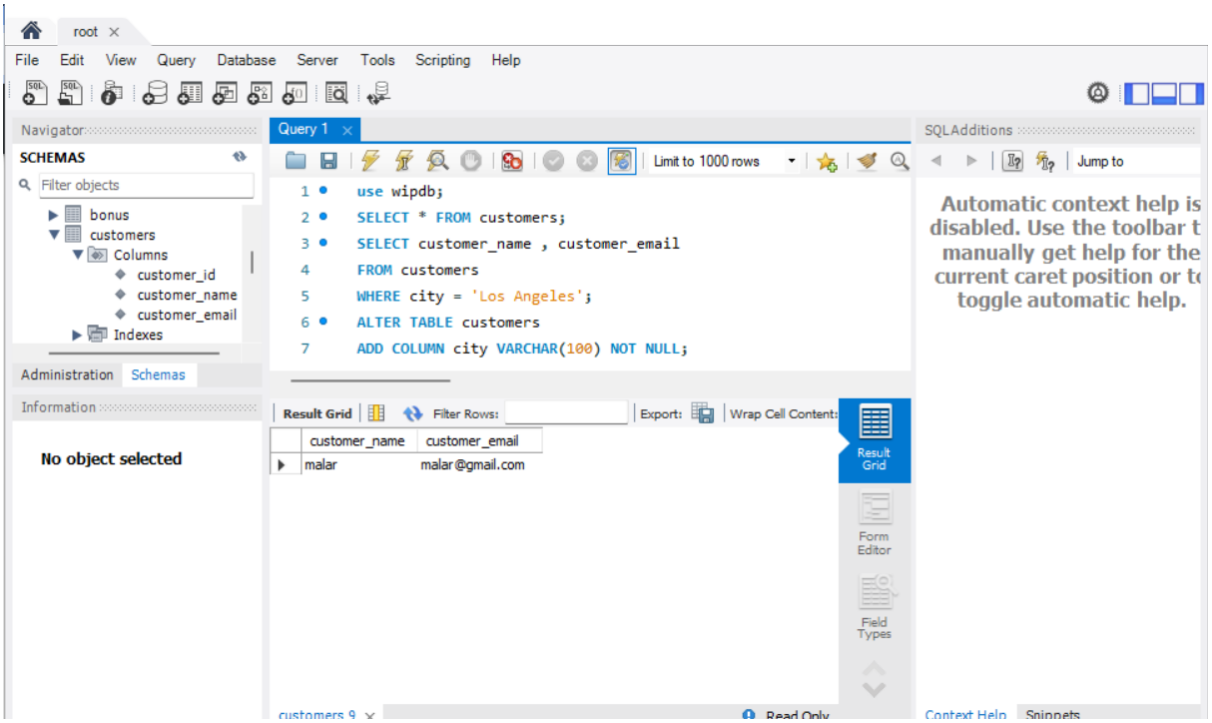
# ASSIGNMENT 1-6

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

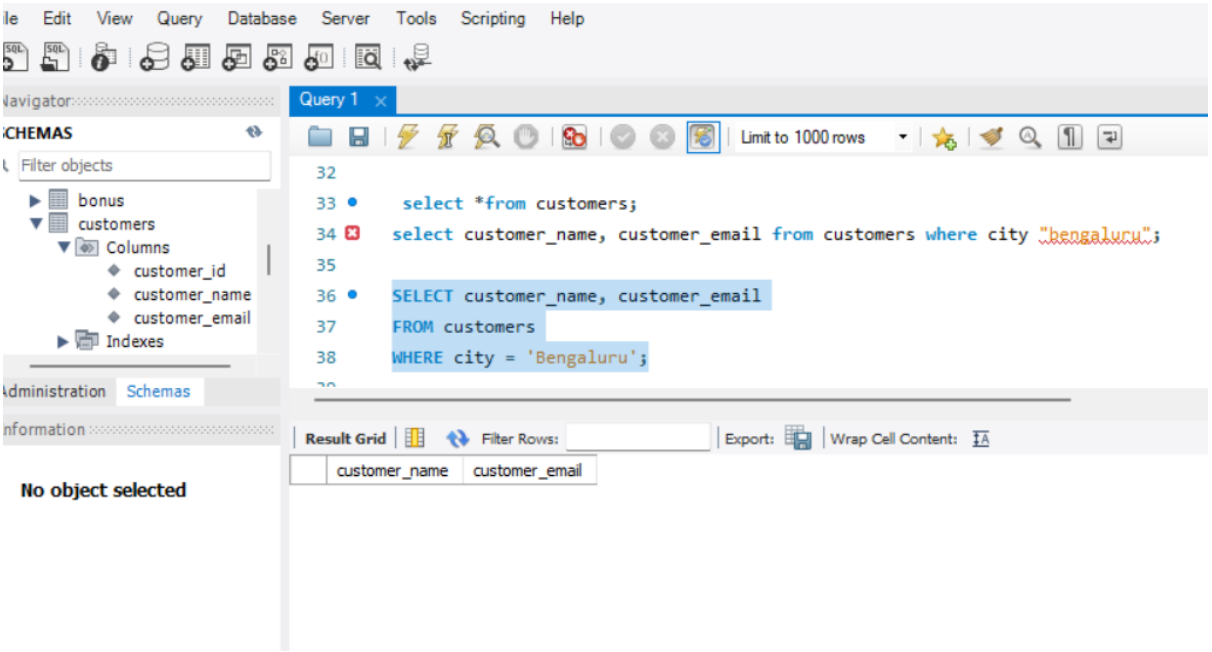
## ASSIGNMENT 1






ASSIGNMENT 2

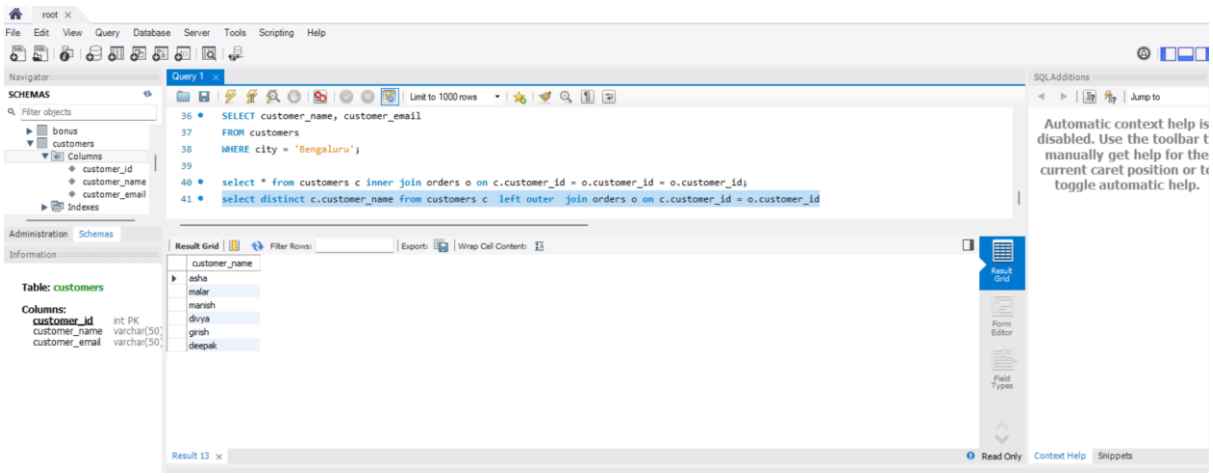
- Craft a query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region, and a LEFT JOIN to display all customers including those without orders.



```
35
36 •  SELECT customer_name, customer_email
37     FROM customers
38     WHERE city = 'Bengaluru';
39
40 •  select * from customers c inner join orders o on c.customer_id = o.customer_id = o.customer_id;
```

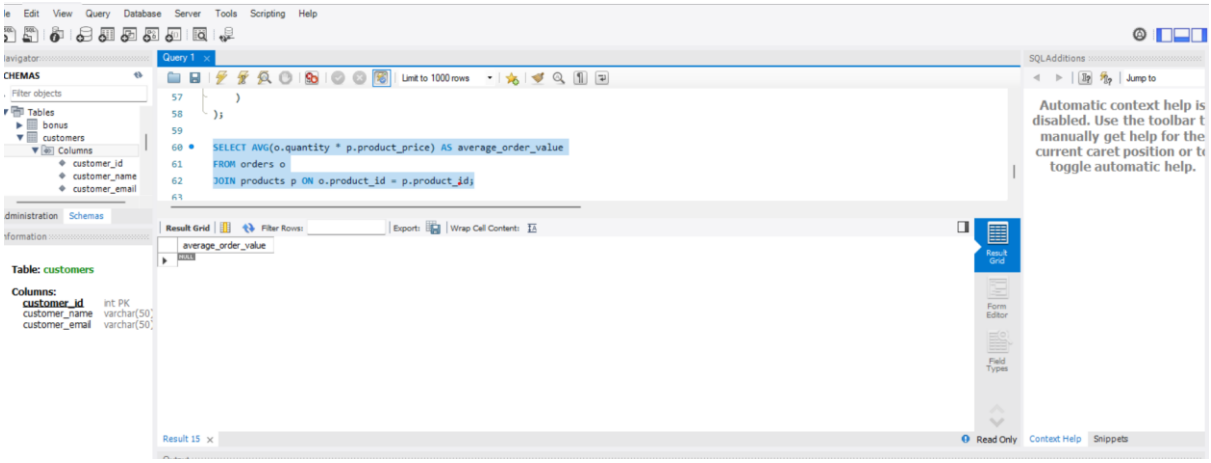
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| customer_id | customer_name | customer_email | city | order_id | customer_id | product_id | quantity | order_date |
|---|---|---|---|---|---|---|---|---|
| 1 | asha | asha@gmail.com | New York | 1 | 1 | 101 | 2 | 2024-01-01 |
| 1 | asha | asha@gmail.com | New York | 4 | 1 | 104 | 1 | 2024-01-04 |




## 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.
-

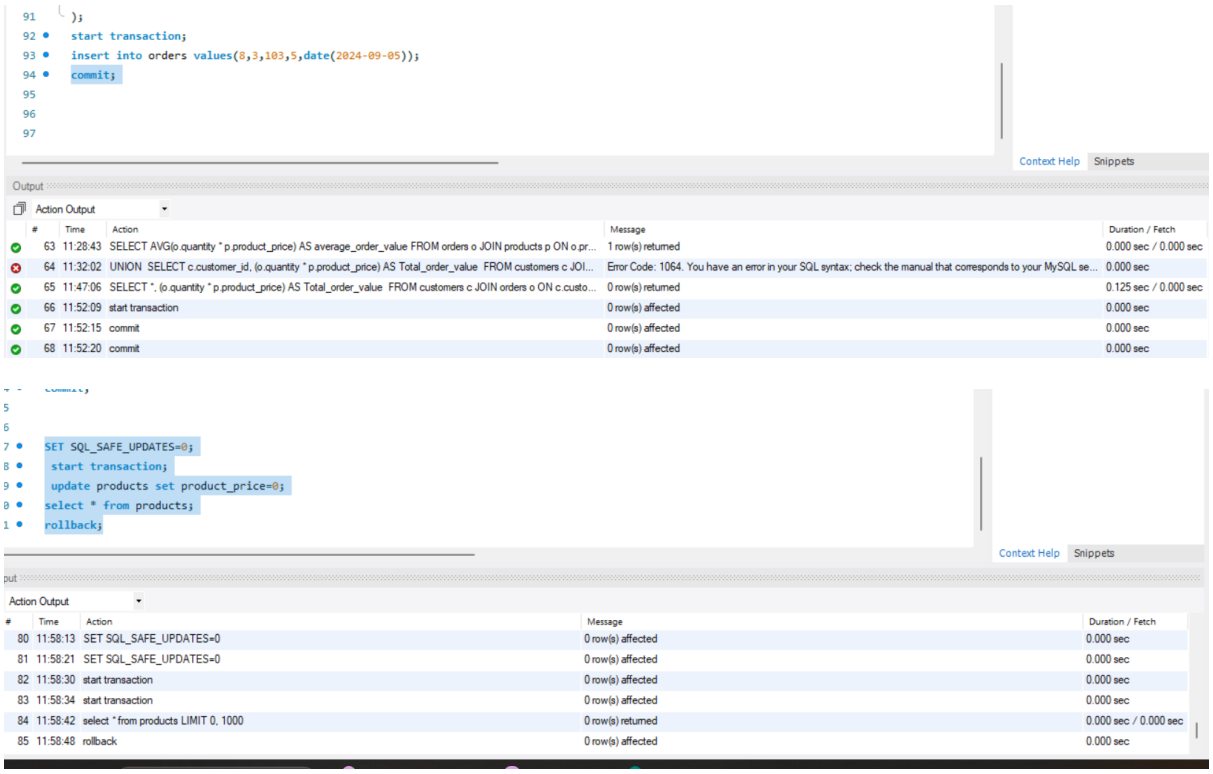



- a UNION query to combine two SELECT statements with the same number of columns.

```
SELECT *, (o.quantity * p.product_price) AS Total_order_value
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN products p ON p.product_id = o.product_id
WHERE (o.quantity * p.product_price) >
(
   SELECT AVG(o.quantity * p.product_price) AS average_order_value
   FROM orders o
   JOIN products p where o.product_id = p.product_id
)
union
SELECT * ,(o.quantity * p.product_price) AS Total_order_value
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN products p ON p.product_id = o.product_id
WHERE (o.quantity * p.product_price) > (
   SELECT AVG(o.quantity * p.product_price) AS average_order_value
   FROM orders o
   JOIN products p ON o.product_id = p.product_id
);
```

## 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



```
);
start transaction;
insert into orders values(8,3,103,5,date(2024-09-05));
commit;
```

Context Help | Snippets

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|---|---|---|---|
| 63 | 11:28:43 | SELECT AVG(o.quantity * p.product_price) AS average_order_value FROM orders o JOIN products p ON o.pr... | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 64 | 11:32:02 | UNION SELECT c.customer_id, (o.quantity * p.product_price) AS Total_order_value FROM customers c JOI... | Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL se... | 0.000 sec |
| 65 | 11:47:06 | SELECT *, (o.quantity * p.product_price) AS Total_order_value FROM customers c JOIN orders o ON c.custo... | 0 row(s) returned | 0.125 sec / 0.000 sec |
| 66 | 11:52:09 | start transaction | 0 row(s) affected | 0.000 sec |
| 67 | 11:52:15 | commit | 0 row(s) affected | 0.000 sec |
| 68 | 11:52:20 | commit | 0 row(s) affected | 0.000 sec |



```
SET SQL_SAFE_UPDATES=0;
 start transaction;
 update products set product_price=0;
select * from products;
rollback;
```

Context Help | Snippets

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|---|---|---|---|
| 80 | 11:58:13 | SET SQL_SAFE_UPDATES=0 | 0 row(s) affected | 0.000 sec |
| 81 | 11:58:21 | SET SQL_SAFE_UPDATES=0 | 0 row(s) affected | 0.000 sec |
| 82 | 11:58:30 | start transaction | 0 row(s) affected | 0.000 sec |
| 83 | 11:58:34 | start transaction | 0 row(s) affected | 0.000 sec |
| 84 | 11:58:42 | select * from products LIMIT 0, 1000 | 0 row(s) returned | 0.000 sec / 0.000 sec |
| 85 | 11:58:48 | rollback | 0 row(s) affected | 0.000 sec |

## Assignment 5

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

Begin;

INSERT INTO orders (order_id, customer_id, product_id, quantity, order_date) VALUES (10, 1, 101, 1, '2024-02-16');

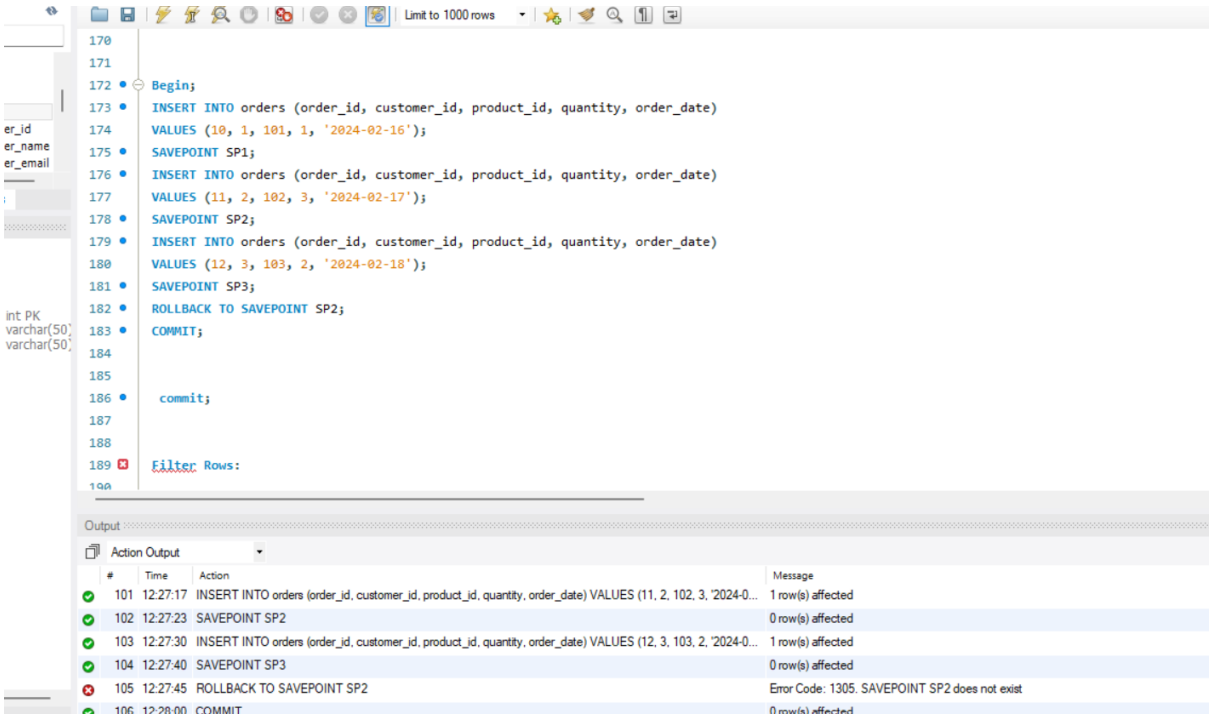
SAVEPOINT SP1; INSERT INTO orders (order_id, customer_id, product_id, quantity, order_date) VALUES (11, 2, 102, 3, '2024-02-17');

SAVEPOINT SP2; INSERT INTO orders (order_id, customer_id, product_id, quantity, order_date) VALUES (12, 3, 103, 2, '2024-02-18');

SAVEPOINT SP3;

ROLLBACK TO SAVEPOINT SP2;

COMMIT;



```
Begin;
INSERT INTO orders (order_id, customer_id, product_id, quantity, order_date)
VALUES (10, 1, 101, 1, '2024-02-16');
SAVEPOINT SP1;
INSERT INTO orders (order_id, customer_id, product_id, quantity, order_date)
VALUES (11, 2, 102, 3, '2024-02-17');
SAVEPOINT SP2;
INSERT INTO orders (order_id, customer_id, product_id, quantity, order_date)
VALUES (12, 3, 103, 2, '2024-02-18');
SAVEPOINT SP3;
ROLLBACK TO SAVEPOINT SP2;
COMMIT;


 commit;


Filter Rows:
```

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 101 | 12:27:17 | INSERT INTO orders (order_id, customer_id, product_id, quantity, order_date) VALUES (11, 2, 102, 3, '2024-0... | 1 row(s) affected |
| 102 | 12:27:23 | SAVEPOINT SP2 | 0 row(s) affected |
| 103 | 12:27:30 | INSERT INTO orders (order_id, customer_id, product_id, quantity, order_date) VALUES (12, 3, 103, 2, '2024-0... | 1 row(s) affected |
| 104 | 12:27:40 | SAVEPOINT SP3 | 0 row(s) affected |
| 105 | 12:27:45 | ROLLBACK TO SAVEPOINT SP2 | Error Code: 1305. SAVEPOINT SP2 does not exist |

# 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**

**Introduction:**

**Transaction logs are a vital aspect of database management systems, providing a record of all changes made to a database. They are instrumental in ensuring data integrity and facilitating data recovery in the event of system failures or disasters. This report delves into the significance of transaction logs for data recovery and presents a hypothetical scenario demonstrating their importance after an unexpected shutdown.**

**Importance of Transaction Logs for Data Recovery:**

**Transaction logs serve as a sequential record of all transactions executed on a database. They capture crucial details including the type of operation performed, the data affected, and the timestamp of the transaction. Transaction logs offer several key benefits for data recovery:**

**1. Point-in-Time Recovery: Transaction logs enable administrators to restore a database to a specific point in time**

before the occurrence of an error or failure. This granularity ensures minimal data loss and allows for precise recovery.

2. Rollback and Rollforward Operations: Transaction logs facilitate rollback operations to undo uncommitted transactions and rollforward operations to reapply committed transactions. These capabilities aid in restoring the database to a consistent state following a failure.

3. Data Integrity: By recording all database modifications, transaction logs ensure data integrity and consistency. In the event of a failure, the database can be restored to a stable state, preventing data corruption and preserving data accuracy.

Hypothetical Scenario:

Consider a scenario where a financial institution experiences an unexpected server crash during peak trading hours. The crash results in the loss of critical transactional data, posing a significant risk to the organization's operations and reputation.

Fortunately, the database system employed by the institution includes transaction logs that capture every transaction in real-time.

Upon detecting the system failure, the database administrator initiates the recovery process:

1. Identifying the Last Checkpoint: The administrator analyzes the transaction logs to identify the last checkpoint before the crash occurred, representing the most recent consistent state of the database.

2. Point-in-Time Recovery: Using the information from the transaction logs, the administrator performs a point-in-time recovery, restoring the database to the state it was in just before the crash.

3. Rollforward Operations: The administrator applies the transactions recorded in the transaction logs after the last checkpoint, ensuring that all committed transactions are reprocessed.

4. Data Verification:Once the recovery process is complete, the administrator verifies the integrity of the recovered data to ensure that all financial transactions are accurately restored.

5. Database Availability: With the database successfully recovered using transaction logs, the financial institution resumes its operations, ensuring uninterrupted service for its clients and maintaining trust in its systems.

In this hypothetical scenario, transaction logs prove indispensable in facilitating the swift and effective recovery of critical data, enabling the financial institution to mitigate the impact of the unexpected shutdown and resume normal operations promptly.

**Conclusion:**

**Transaction logs are an essential component of database management systems, providing a comprehensive record of all transactions and serving as a valuable tool for data recovery. By leveraging transaction logs, organizations can minimize data loss, maintain data integrity, and ensure business continuity in the face of unforeseen challenges.**