**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.

CREATE DATABASE orders;

USE orders;

CREATE TABLE customers( order\_id INT,

Fname VARCHAR(50), Lname VARCHAR(50), Email VARCHAR(60),

Ph\_no BIGINT,

Address VARCHAR(100)

);

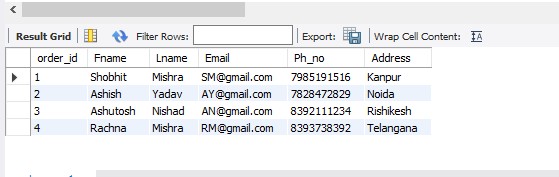
INSERT INTO customers VALUES (1,'Shobhit','Mishra','SM@gmail.com',7985191516,'Kanpur'),

(2,'Ashish','Yadav','AY@gmail.com',7828472829,'Noida'),

(3,'Ashutosh','Nishad','AN@gmail.com',8392111234,'Rishikesh'),

(4,'Rachna','Mishra','RM@gmail.com',8393738392,'Telangana');

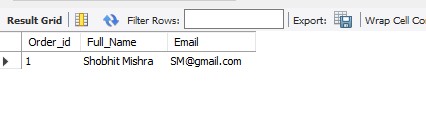
SELECT \* FROM customers;



# Return only the customer name and email address for customers in a specific city:

SELECT Order\_id,

concat(Fname," ",Lname) as Full\_Name,Email from customers where Address='Kanpur';



**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.

CREATE DATABASE orders;

USE orders;

CREATE TABLE customers( order\_id INT,

Fname VARCHAR(50), Lname VARCHAR(50), Email VARCHAR(60),

Ph\_no BIGINT,

Address VARCHAR(100)

);

INSERT INTO customers VALUES (1,'Shobhit','Mishra','SM@gmail.com',7985191516,'Kanpur'),

(2,'Ashish','Yadav','AY@gmail.com',7828472829,'Noida'),

(3,'Ashutosh','Nishad','AN@gmail.com',8392111234,'Rishikesh'),

(4,'Rachna','Mishra','RM@gmail.com',8393738392,'Telangana');

SELECT \* FROM customers;

SELECT Order\_id,

concat(Fname," ",Lname) as Full\_Name,Email from customers where Address='Kanpur';

CREATE TABLE orders ( order\_id INT,

product\_name VARCHAR(100), price INT

);

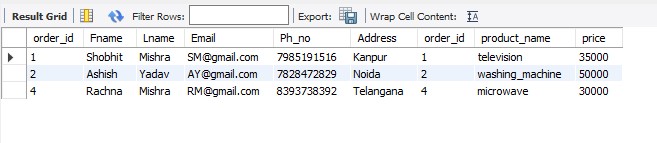
INSERT INTO orders VALUES

(1,'television', 35000),

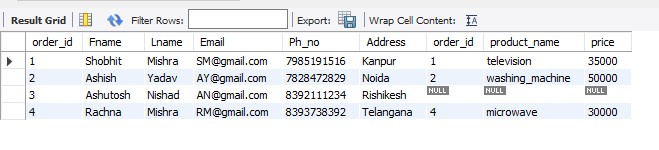
(2,'washing\_machine', 50000),

(4,'microwave', 30000);

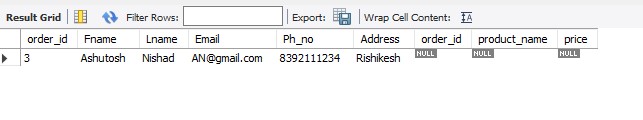
SELECT \* FROM customers c INNER JOIN orders o ON c.order\_id = o.order\_id;



SELECT \* FROM customers c LEFT JOIN orders o ON c.order\_id = o.order\_id;



SELECT \* FROM customers c LEFT JOIN orders o ON c.order\_id = o.order\_id WHERE o.order\_id IS NULL;



**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.

CREATE DATABASE orders;

USE orders;

CREATE TABLE customers( order\_id INT,

Fname VARCHAR(50), Lname VARCHAR(50), Email VARCHAR(60),

Ph\_no BIGINT,

Address VARCHAR(100)

);

INSERT INTO customers VALUES (1,'Shobhit','Mishra','SM@gmail.com',7985191516,'Kanpur'),

(2,'Ashish','Yadav','AY@gmail.com',7828472829,'Noida'),

(3,'Ashutosh','Nishad','AN@gmail.com',8392111234,'Rishikesh'),

(4,'Rachna','Mishra','RM@gmail.com',8393738392,'Telangana');

SELECT \* FROM customers;

SELECT Order\_id,

concat(Fname," ",Lname) as Full\_Name,Email from customers where Address='Kanpur';

CREATE TABLE orders ( order\_id INT,

product\_name VARCHAR(100), price INT

);

INSERT INTO orders VALUES

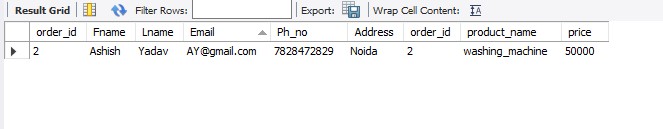
(1,'television', 35000),

(2,'washing\_machine', 50000),

(2,'washing\_machine', 30000),

(4,'microwave', 30000);

SELECT \* FROM customers c INNER JOIN orders o ON c.order\_id = o.order\_id WHERE o.price > (SELECT AVG(price) from orders);



**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.

CREATE DATABASE orders;

USE orders;

CREATE TABLE customers( order\_id INT,

Fname VARCHAR(50), Lname VARCHAR(50), Email VARCHAR(60),

Ph\_no BIGINT,

Address VARCHAR(100)

);

INSERT INTO customers VALUES (1,'Shobhit','Mishra','SM@gmail.com',7985191516,'Kanpur'),

(2,'Ashish','Yadav','AY@gmail.com',7828472829,'Noida'),

(3,'Ashutosh','Nishad','AN@gmail.com',8392111234,'Rishikesh'),

(4,'Rachna','Mishra','RM@gmail.com',8393738392,'Telangana');

SELECT \* FROM customers;

SELECT Order\_id,

concat(Fname," ",Lname) as Full\_Name,Email from customers where Address='Kanpur';

CREATE TABLE orders ( order\_id INT,

product\_name VARCHAR(100), price INT

);

INSERT INTO orders VALUES

(1,'television', 35000),

(2,'washing\_machine', 50000),

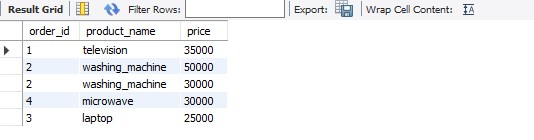
(2,'washing\_machine', 30000),

(4,'microwave', 30000);

start transaction;

insert into orders values(3,'laptop',25000);

commit;

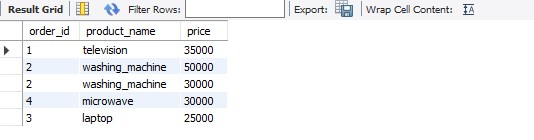


set sql\_safe\_updates=0;

update orders set price=3000 where order\_id=3;



Rollback;



**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.

CREATE TABLE orders ( order\_id VARCHAR(100), customer\_id INT, order\_date INT,

price INT

);

INSERT INTO orders VALUES (1,101,2024-05-20,150),

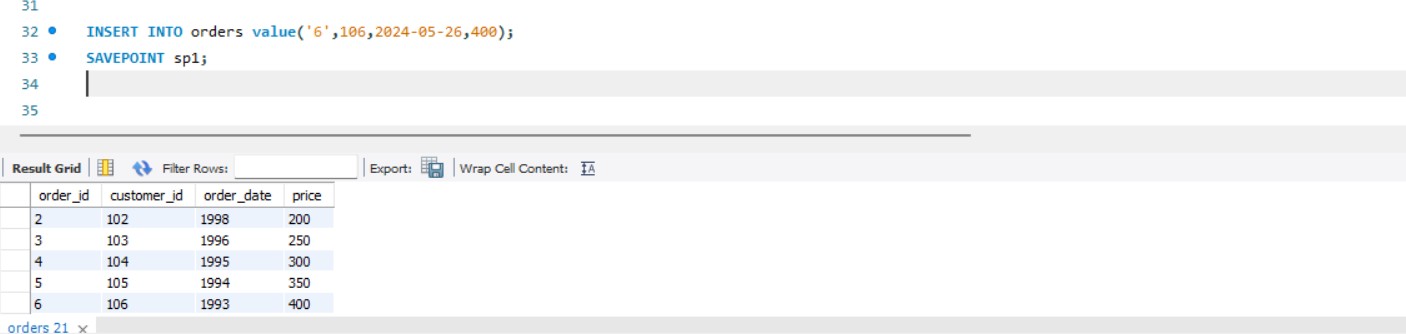
('2',102,2024-05-21,200),

('3',103,2024-05-23,250),

('4',104,2024-05-24,300);

('5',105,2024-05-25,350);

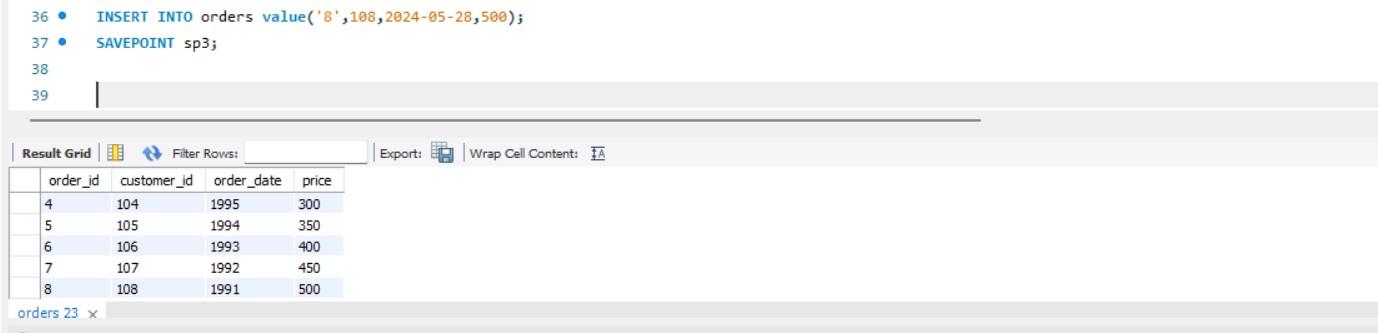
INSERT INTO orders value('6',106,2024-05-26,400); SAVEPOINT sp1;

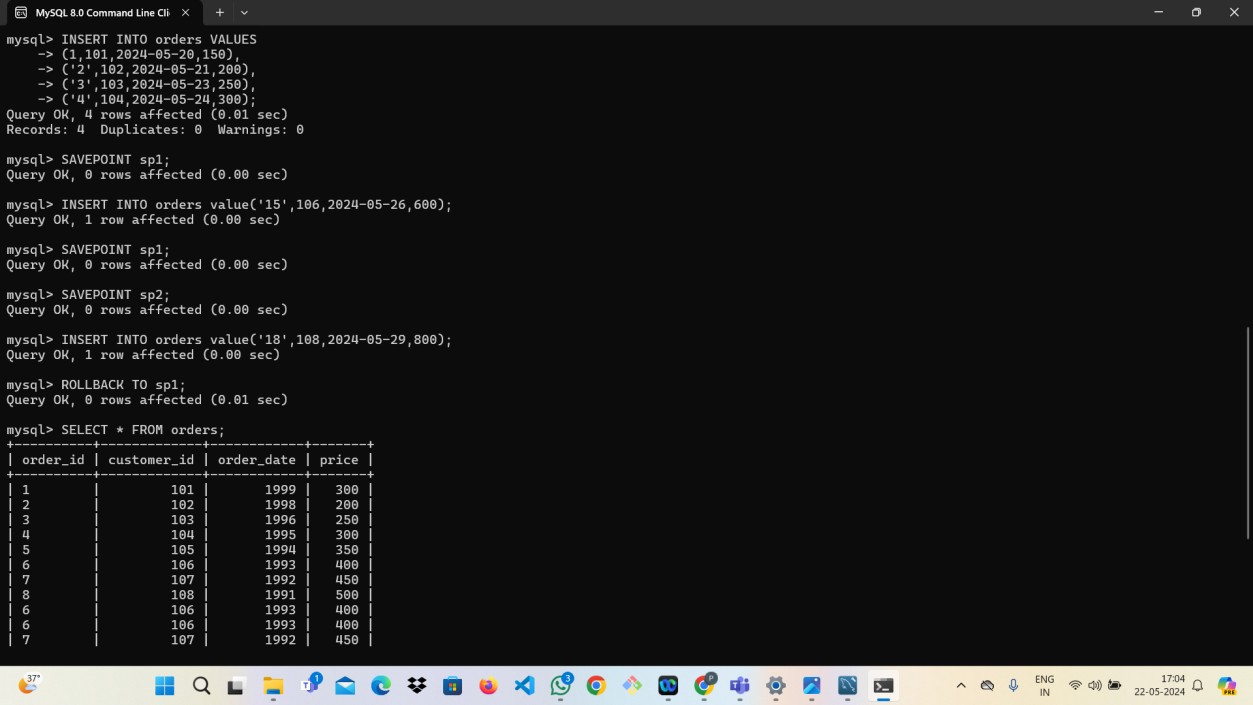
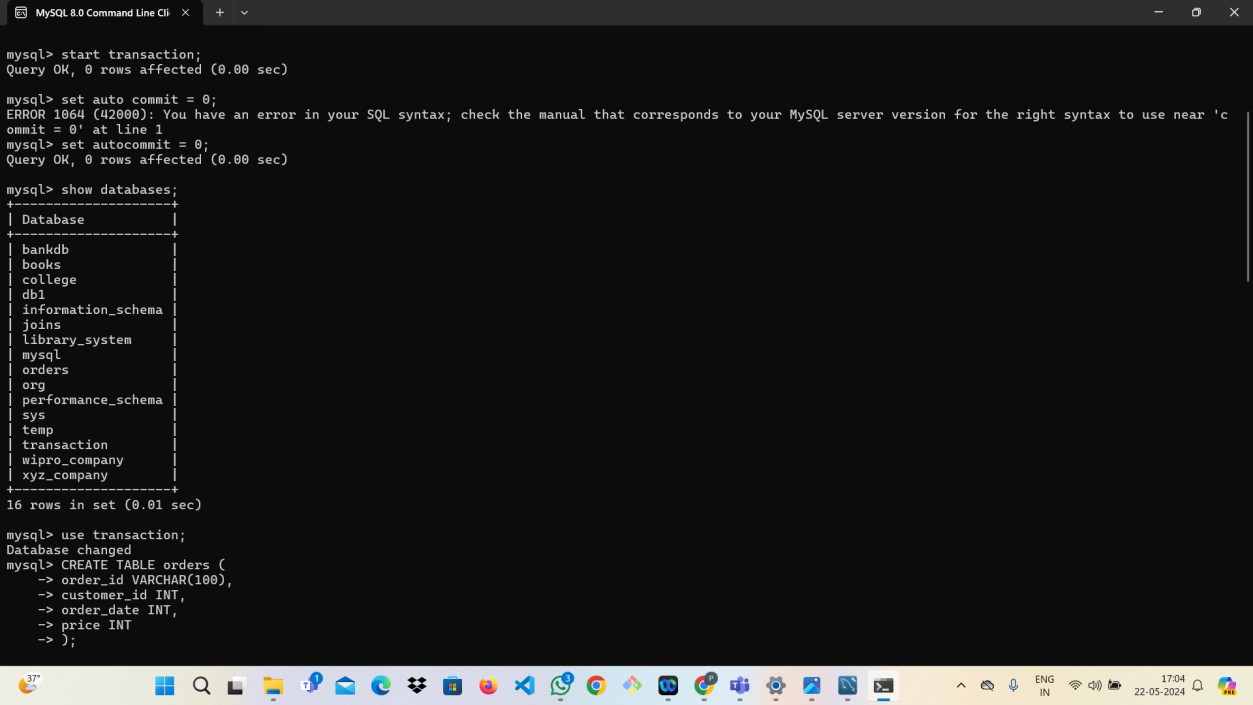


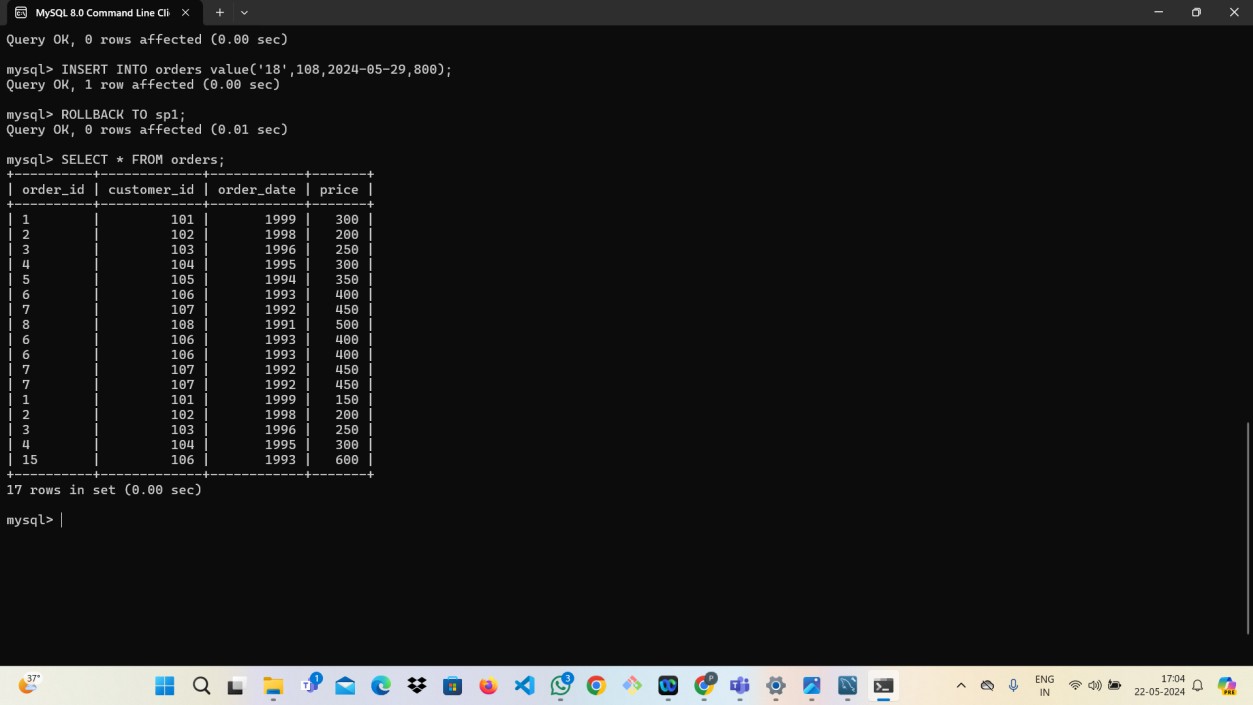
INSERT INTO orders value('7',107,2024-05-27,450); SAVEPOINT sp2;



INSERT INTO orders value('8',108,2024-05-28,500); SAVEPOINT sp3;







**Assignement 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.

# Purpose of Transaction Logs

Transaction logs serve several critical purposes:

1. Data Integrity: They ensure that all database transactions are either fully completed or fully rolled back, maintaining the atomicity of transactions.
2. Crash Recovery: In the event of a system crash, transaction logs enable the restoration of the database to its last consistent state.
3. Audit Trail: They provide a detailed record of all transactions, useful for auditing and forensic analysis.

# Transaction Logs Work

A transaction log records every transaction that modifies the database. Each entry in the log typically includes the following information:

* Transaction ID: A unique identifier for the transaction.
* Timestamp: The time at which the transaction occurred.
* Operation: The type of operation performed (INSERT, UPDATE, DELETE).
* Affected Data: The specific data that was changed, including the before and after states.

When a transaction is initiated, it is first recorded in the transaction log. The database engine then proceeds with the actual data modification. Once the transaction is complete, a commit record is added to the log. If a failure occurs before the commit, the database uses the log to roll back any partial changes.

# Hypothetical Scenario: Data Recovery After Unexpected Shutdown

Scenario:

Imagine a financial institution that relies on a MySQL database to manage customer transactions. The database is set to record all operations in a transaction log. During a busy day, the server experiences an unexpected shutdown due to a power failure. The database was in the middle of processing several transactions at the time of the shutdown.