NAME:JANHAVI PAWAR

PRN:2124UCSF1074

Dept:Cyber Security

DBMS PRACTICAL

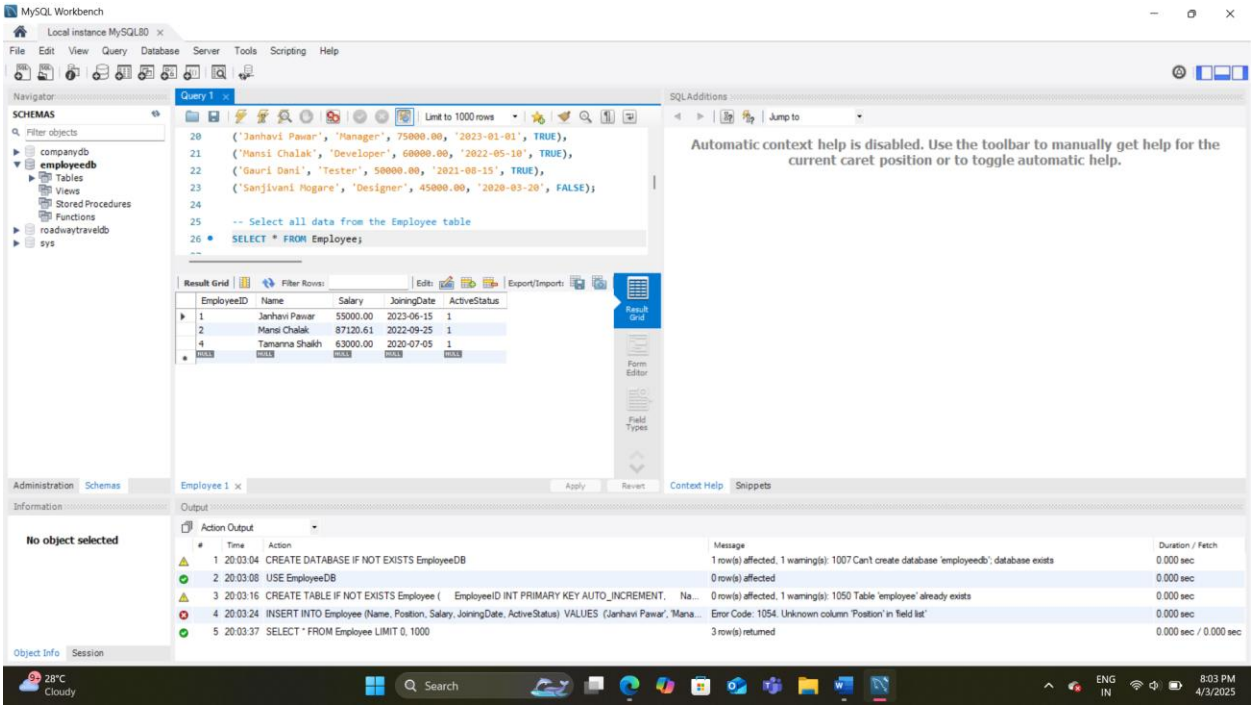
Practical:1

Aim: Create a database and a table to store employee details. Perform basic operations like INSERT, UPDATE, and DELETE using SELECT queries.

Code:

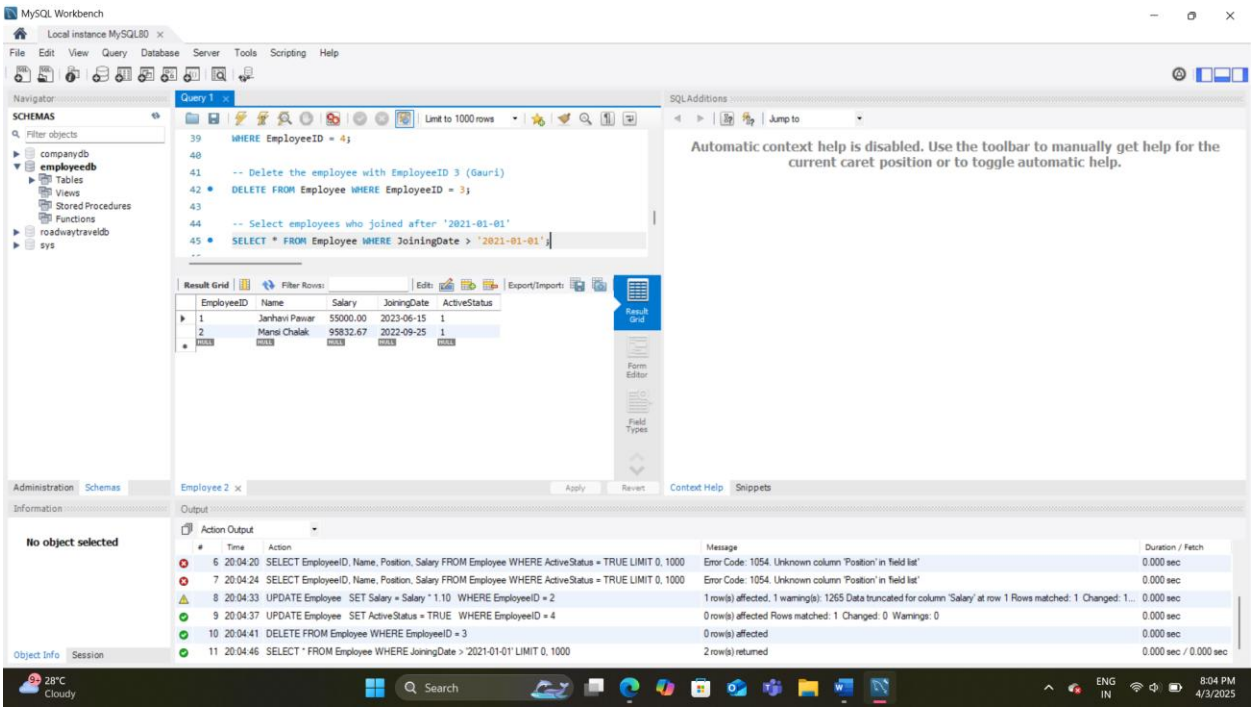
```
-- Create the database if it doesn't exist
CREATE DATABASE IF NOT EXISTS EmployeeDB;

-- Use the created database
USE EmployeeDB;

-- Create the Employee table
CREATE TABLE IF NOT EXISTS Employee (
  EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(100) NOT NULL,
  Position VARCHAR(100),
  Salary DECIMAL(10, 2) CHECK (Salary > 0),
  JoiningDate DATE NOT NULL,
  ActiveStatus BOOLEAN DEFAULT TRUE
);

-- Insert employee data into the Employee table
```

```
INSERT INTO Employee (Name, Position, Salary, JoiningDate, ActiveStatus)
VALUES
('Janhavi Pawar', 'Manager', 75000.00, '2023-01-01', TRUE),
('Mansi Chalak', 'Developer', 60000.00, '2022-05-10', TRUE),
('Gauri Dani', 'Tester', 50000.00, '2021-08-15', TRUE),
('Sanjivani Mogare', 'Designer', 45000.00, '2020-03-20', FALSE);

-- Select all data from the Employee table
SELECT * FROM Employee;
```

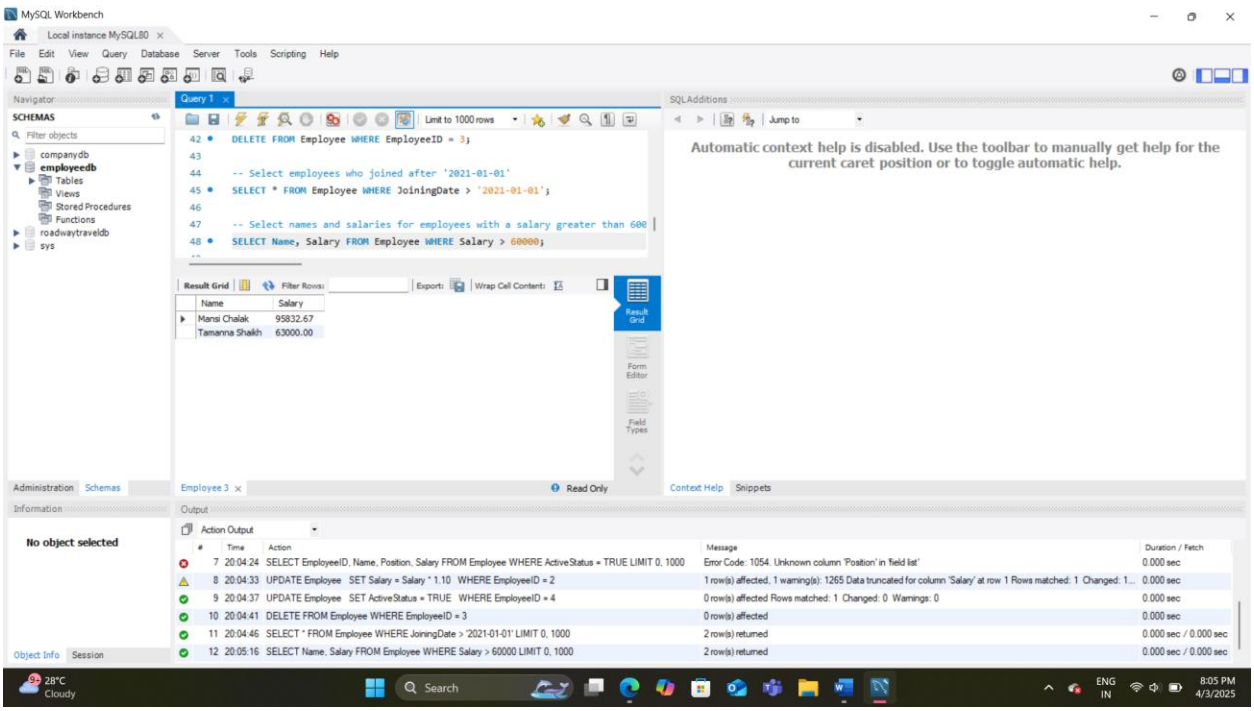



```
-- Select only active employees
SELECT EmployeeID, Name, Position, Salary FROM Employee WHERE ActiveStatus = TRUE;

-- Update salary for the employee with EmployeeID 2 (Mansi)
UPDATE Employee
```

```
SET Salary = Salary * 1.10

WHERE EmployeeID = 2;


-- Update the ActiveStatus for the employee with EmployeeID 4 (Sanjivani)

UPDATE Employee

SET ActiveStatus = TRUE

WHERE EmployeeID = 4;


-- Delete the employee with EmployeeID 3 (Gauri)

DELETE FROM Employee WHERE EmployeeID = 3;


-- Select employees who joined after '2021-01-01'

SELECT * FROM Employee WHERE JoiningDate > '2021-01-01';
```

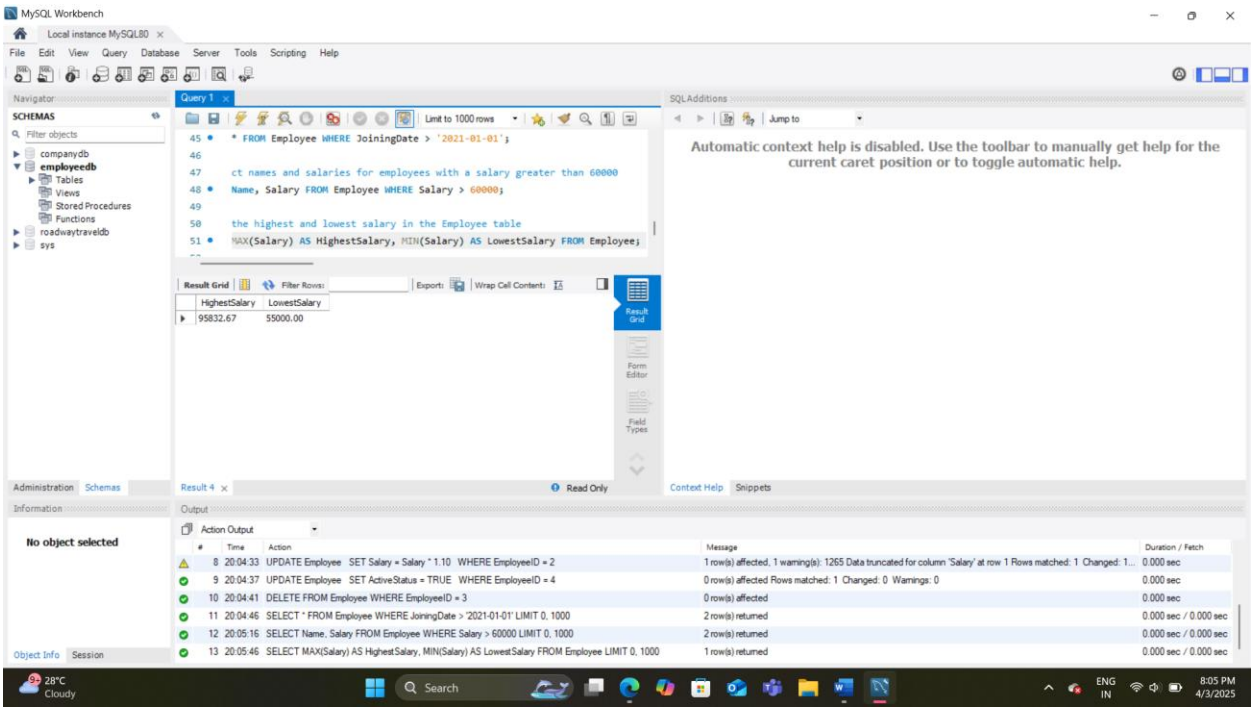



```
-- Select names and salaries for employees with a salary greater than 60000

SELECT Name, Salary FROM Employee WHERE Salary > 60000;
```



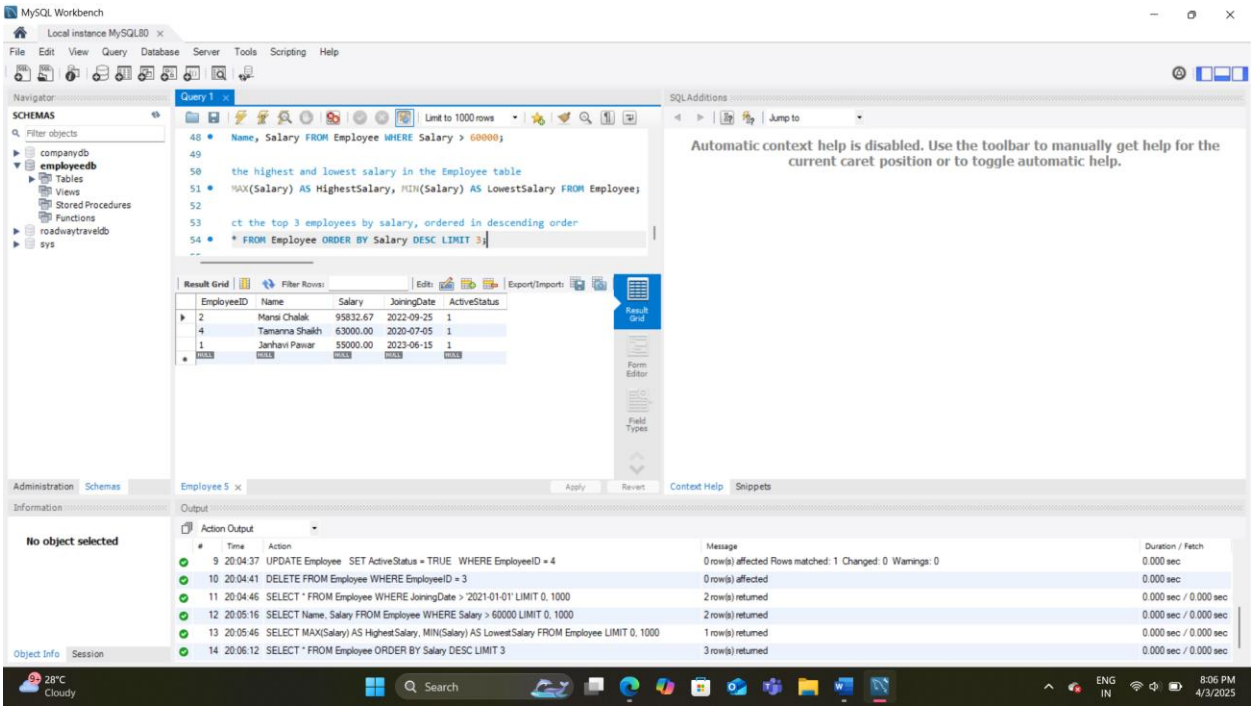
-- Get the highest and lowest salary in the Employee table

SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS LowestSalary FROM Employee;



-- Select the top 3 employees by salary, ordered in descending order

SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;



Practical 2:

Aim: Design an ER diagram for a Roadway Travel Management System with entities like Customer, Travel Route, and Booking. Create tables and perform operations such as bookings and route assignments.

Code:

```
-- Create Database

CREATE DATABASE IF NOT EXISTS RoadwayTravelDB;

USE RoadwayTravelDB;

 -- Create Customer Table

CREATE TABLE Customer (

   CustomerID INT PRIMARY KEY AUTO_INCREMENT,

   Name VARCHAR(100) NOT NULL,
```

```
    Email VARCHAR(100),
    PhoneNumber VARCHAR(15),
    Address VARCHAR(255)
);
 -- Create Travel Route Table
CREATE TABLE TravelRoute (
    RouteID INT PRIMARY KEY AUTO_INCREMENT,
    StartLocation VARCHAR(100),
    EndLocation VARCHAR(100),
    Distance DECIMAL(5, 2),
    Price DECIMAL(10, 2)
);
 -- Create Booking Table
CREATE TABLE Booking (
    BookingID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerID INT,
    RouteID INT,
    BookingDate DATE NOT NULL,
    SeatNumber INT,
    TotalPrice DECIMAL(10, 2),
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    FOREIGN KEY (RouteID) REFERENCES TravelRoute(RouteID)
);
 -- Insert Sample Data into Customer Table
INSERT INTO Customer (Name, Email, PhoneNumber, Address) VALUES
('Janhavi Pawar', 'janhavipawar@example.com', '1234567890', '123 Main St, City A'),
```

('Mansi Chalak', 'mansichalak@example.com', '9876543210', '456 Elm St, City B');

-- Insert Sample Data into TravelRoute Table

INSERT INTO TravelRoute (StartLocation, EndLocation, Distance, Price) VALUES

('City A', 'City B', 150.00, 50.00),

('City B', 'City C', 200.00, 70.00),

('City C', 'City D', 300.00, 90.00);

-- Insert Booking for a Customer

INSERT INTO Booking (CustomerID, RouteID, BookingDate, SeatNumber, TotalPrice)

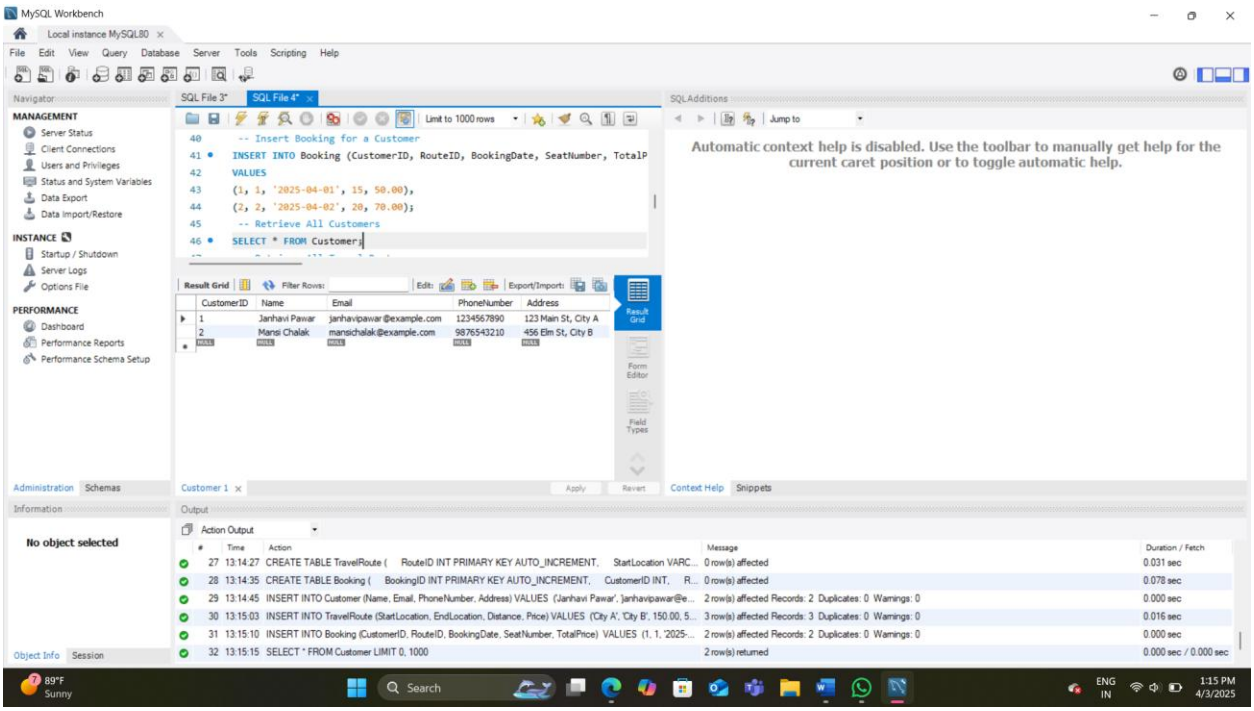
VALUES

(1, 1, '2025-04-01', 15, 50.00),

(2, 2, '2025-04-02', 20, 70.00);

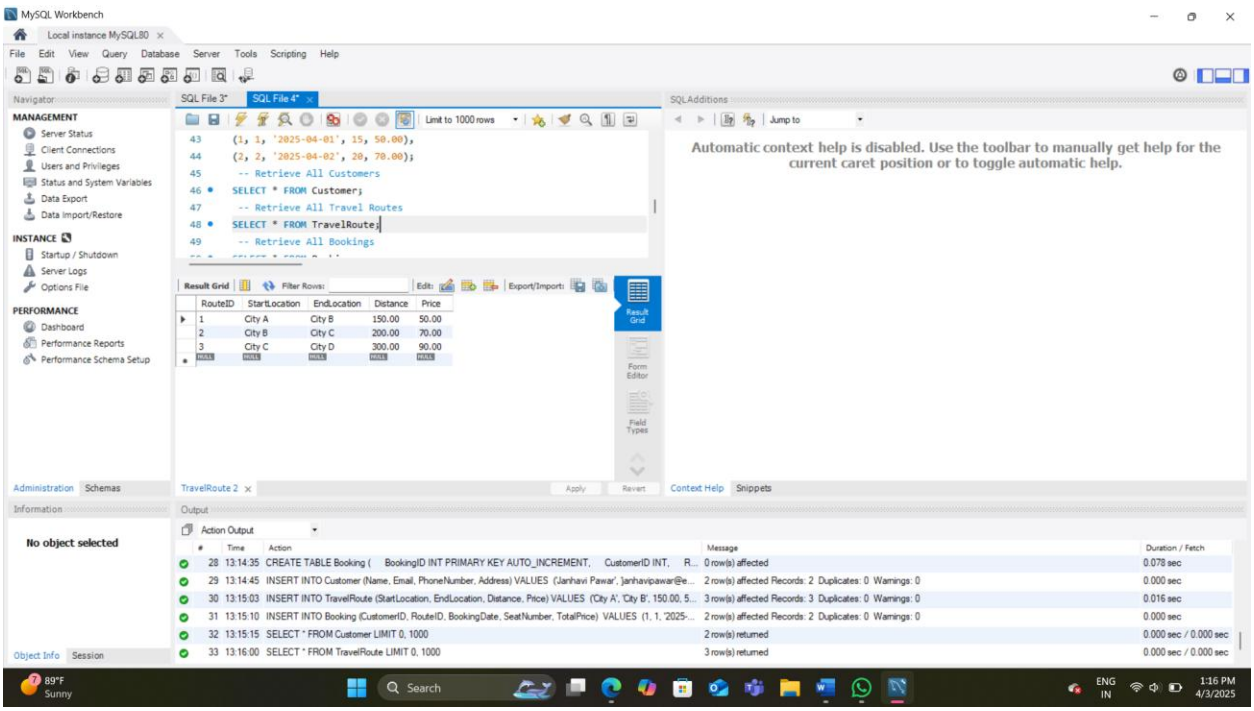
-- Retrieve All Customers

SELECT * FROM Customer;



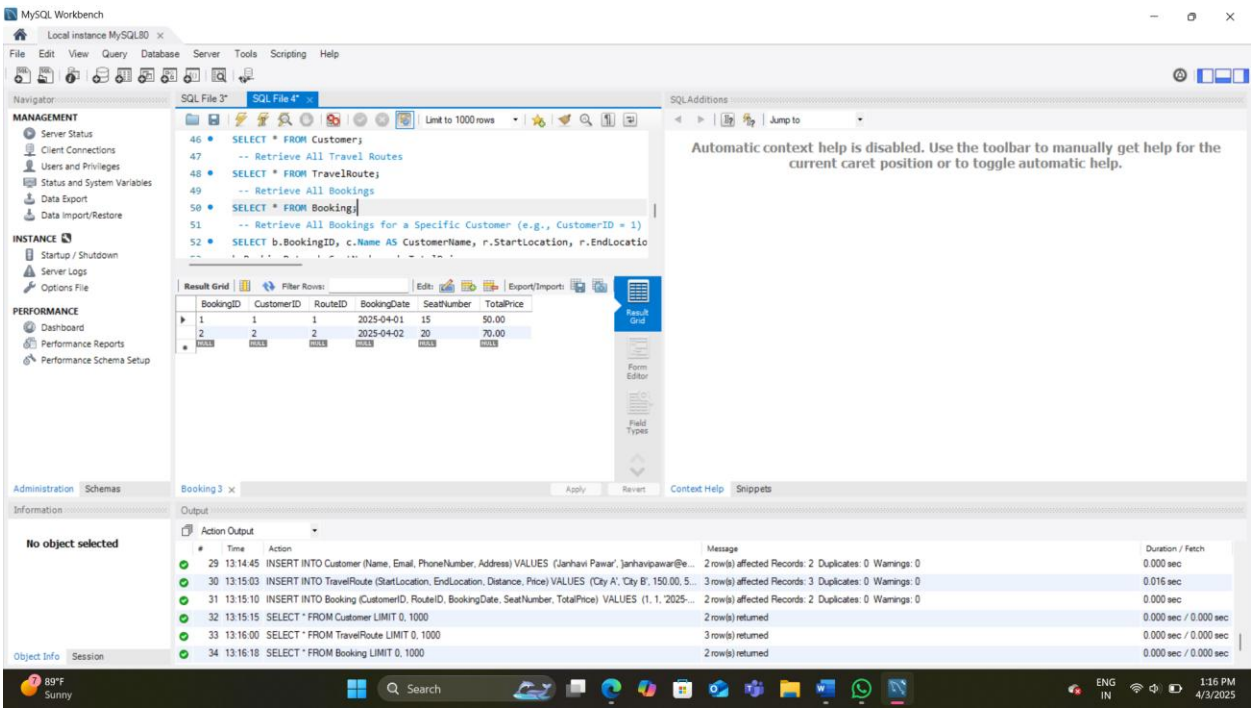
-- Retrieve All Travel Routes

SELECT * FROM TravelRoute;



-- Retrieve All Bookings

SELECT * FROM Booking;



-- Retrieve All Bookings for a Specific Customer (e.g., CustomerID = 1)

SELECT b.BookingID, c.Name AS CustomerName, r.StartLocation, r.EndLocation,

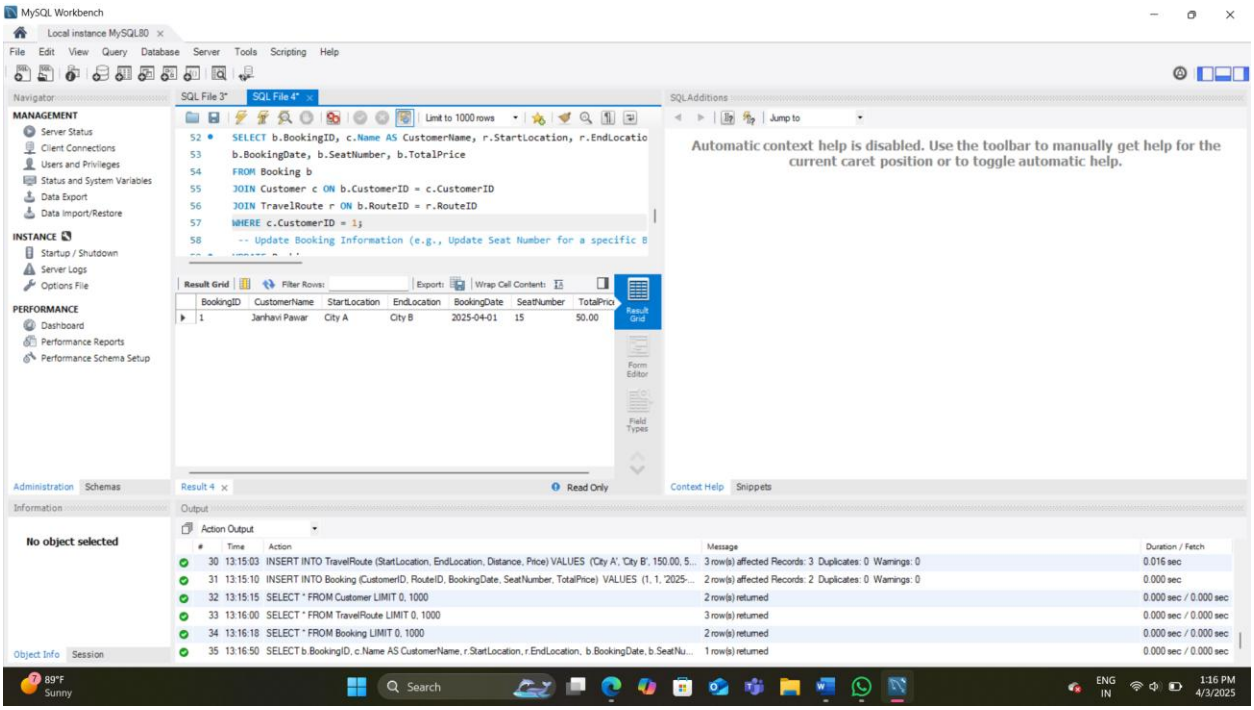
b.BookingDate, b.SeatNumber, b.TotalPrice

FROM Booking b

JOIN Customer c ON b.CustomerID = c.CustomerID

JOIN TravelRoute r ON b.RouteID = r.RouteID

WHERE c.CustomerID = 1;



-- Update Booking Information (e.g., Update Seat Number for a specific Booking)

UPDATE Booking

SET SeatNumber = 25, TotalPrice = 70.00

WHERE BookingID = 1;

-- Delete a Booking

DELETE FROM Booking WHERE BookingID = 2;

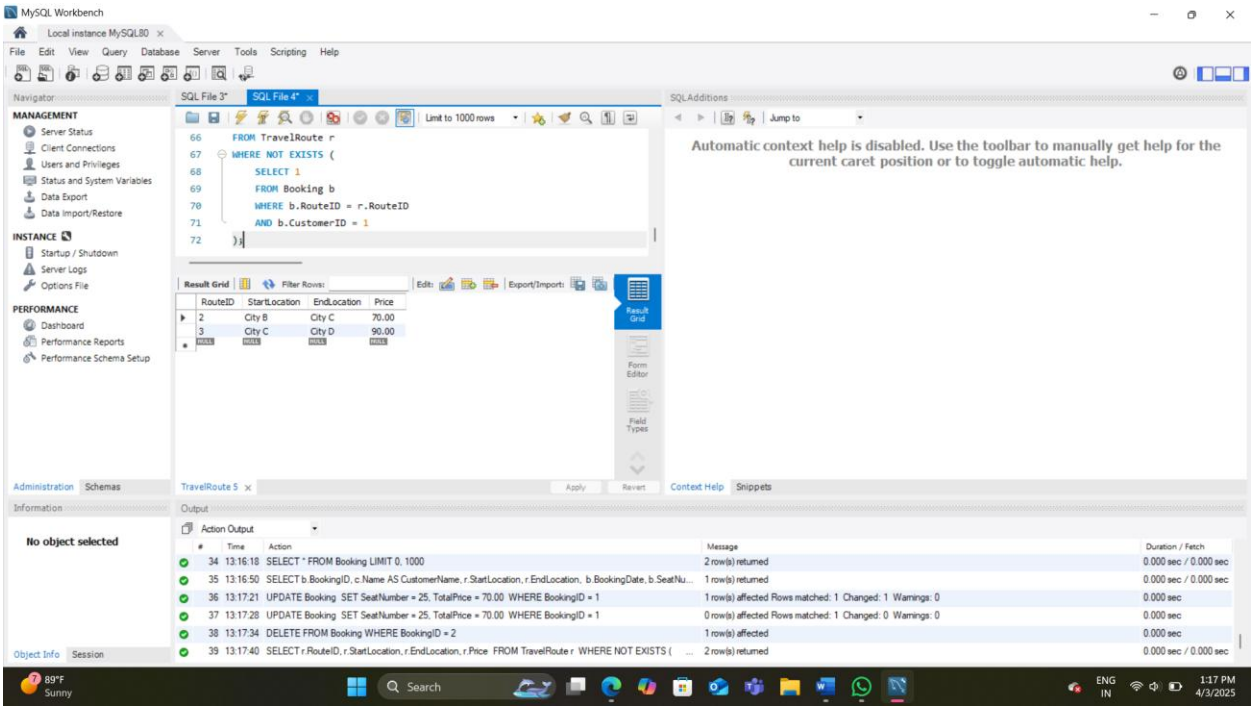
-- Display Available Routes for a Customer

SELECT r.RouteID, r.StartLocation, r.EndLocation, r.Price

FROM TravelRoute r

WHERE NOT EXISTS (

```
    SELECT 1
    FROM Booking b
    WHERE b.RouteID = r.RouteID
    AND b.CustomerID = 1
);
```



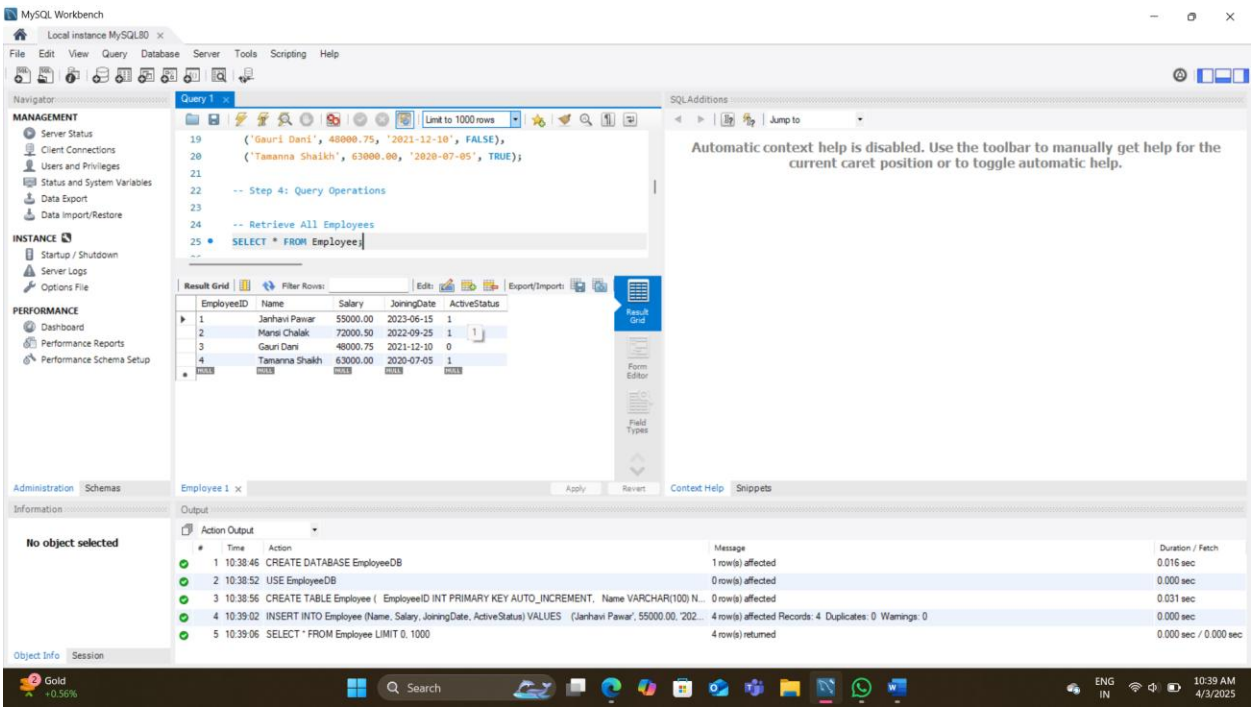
Practical 3:

Aim: Create a table with columns for EmployeeID, Name, Salary, JoiningDate, and ActiveStatus using different data types. Insert sample data and perform queries to manipulate and retrieve data.

Code:

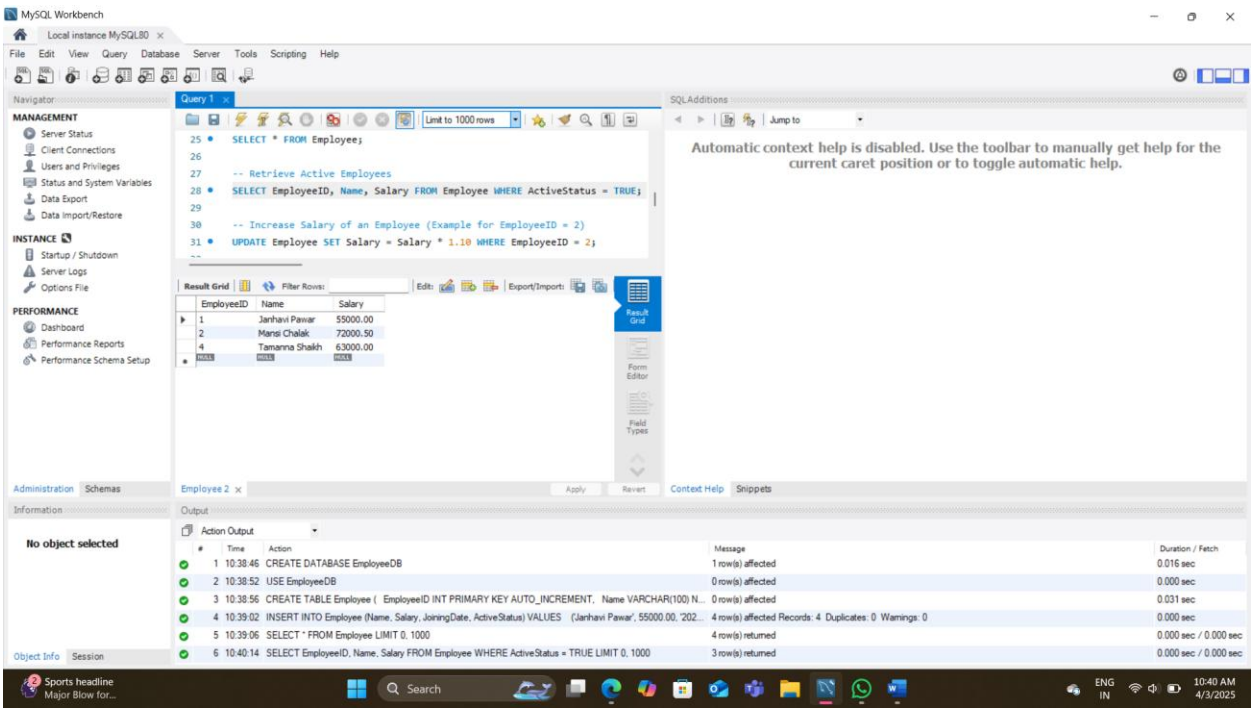
```
-- Step 1: Create the Database and Use It

CREATE DATABASE EmployeeDB;

USE EmployeeDB;
```

```
-- Step 2: Create the Employee Table
CREATE TABLE Employee (
  EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(100) NOT NULL,
  Salary DECIMAL(10,2) CHECK (Salary > 0),
  JoiningDate DATE NOT NULL,
  ActiveStatus BOOLEAN DEFAULT TRUE
);

-- Step 3: Insert Sample Data (Updated with New Names)
INSERT INTO Employee (Name, Salary, JoiningDate, ActiveStatus)
VALUES
  ('Janhavi Pawar', 55000.00, '2023-06-15', TRUE),
  ('Mansi Chalak', 72000.50, '2022-09-25', TRUE),
  ('Gauri Dani', 48000.75, '2021-12-10', FALSE),
  ('Tamanna Shaikh', 63000.00, '2020-07-05', TRUE);

-- Step 4: Query Operations
-- Retrieve All Employees
SELECT * FROM Employee;
```



-- Retrieve Active Employees

SELECT EmployeeID, Name, Salary FROM Employee WHERE ActiveStatus = TRUE;



-- Increase Salary of an Employee (Example for EmployeeID = 2)

UPDATE Employee SET Salary = Salary * 1.10 WHERE EmployeeID = 2;

-- Change Active Status of an Employee (Example for EmployeeID = 4)

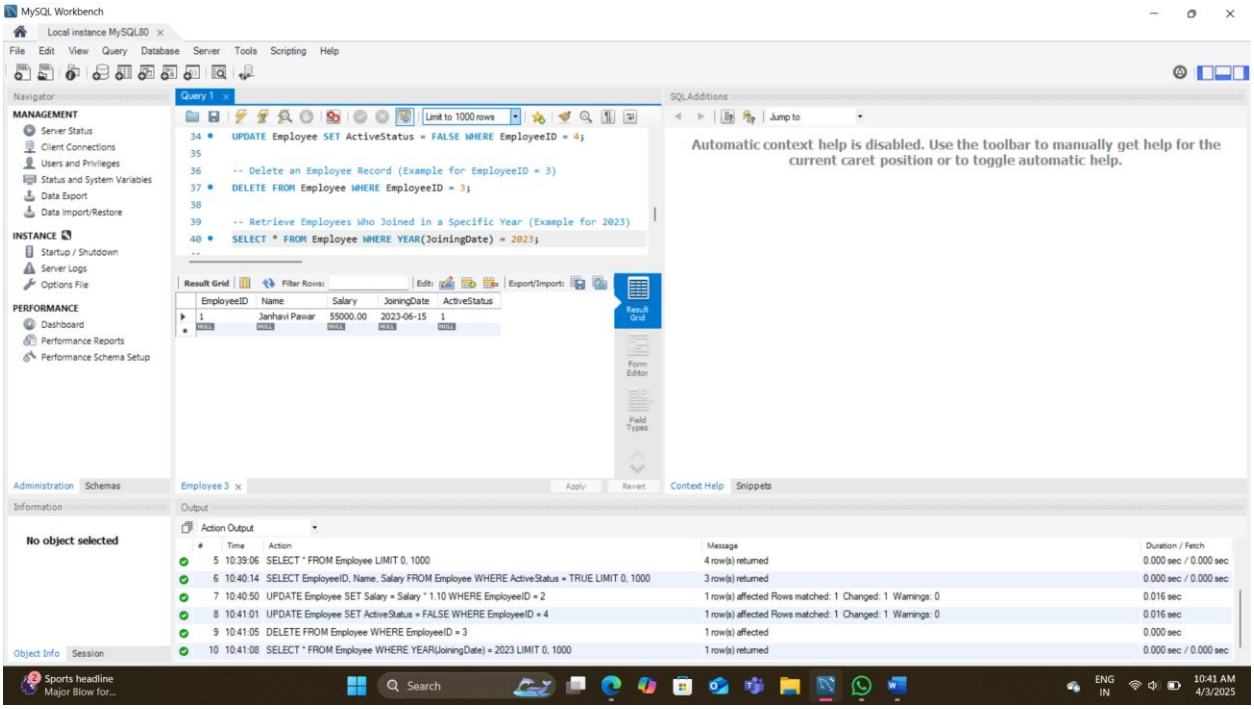
UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID = 4;

-- Delete an Employee Record (Example for EmployeeID = 3)

DELETE FROM Employee WHERE EmployeeID = 3;

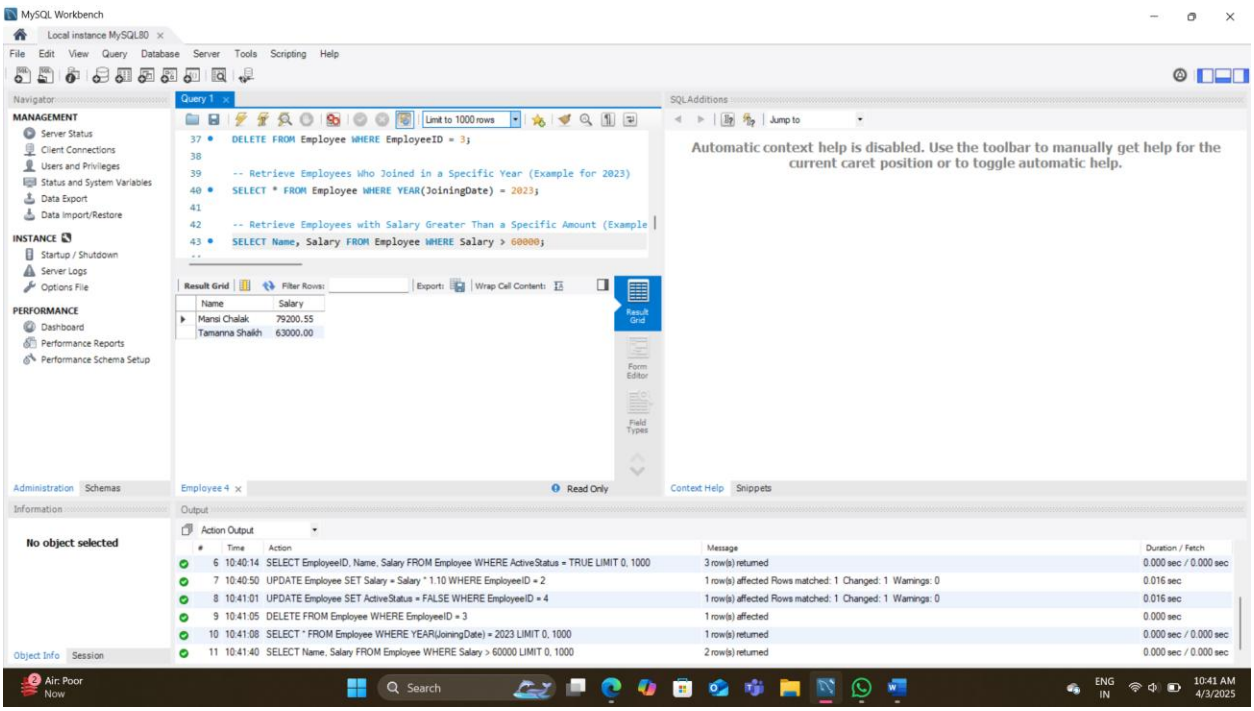
-- Retrieve Employees Who Joined in a Specific Year (Example for 2023)

SELECT * FROM Employee WHERE YEAR(JoiningDate) = 2023;



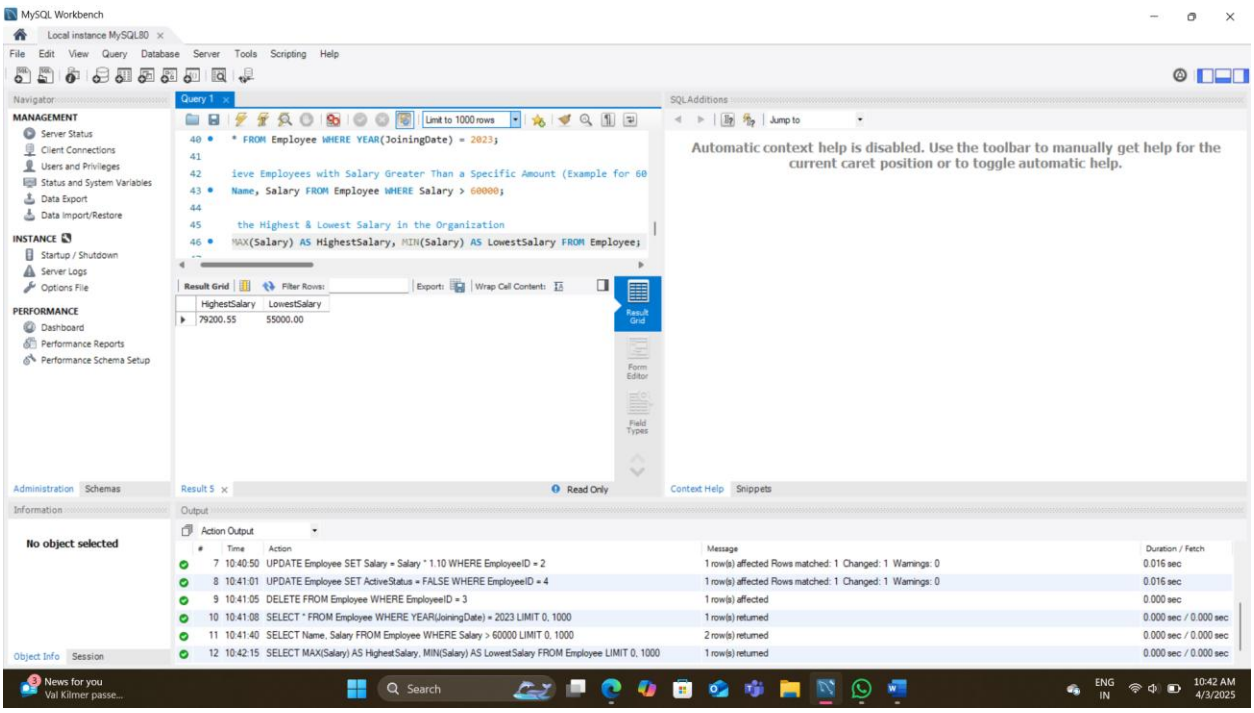
-- Retrieve Employees with Salary Greater Than a Specific Amount (Example for 60000)

SELECT Name, Salary FROM Employee WHERE Salary > 60000;



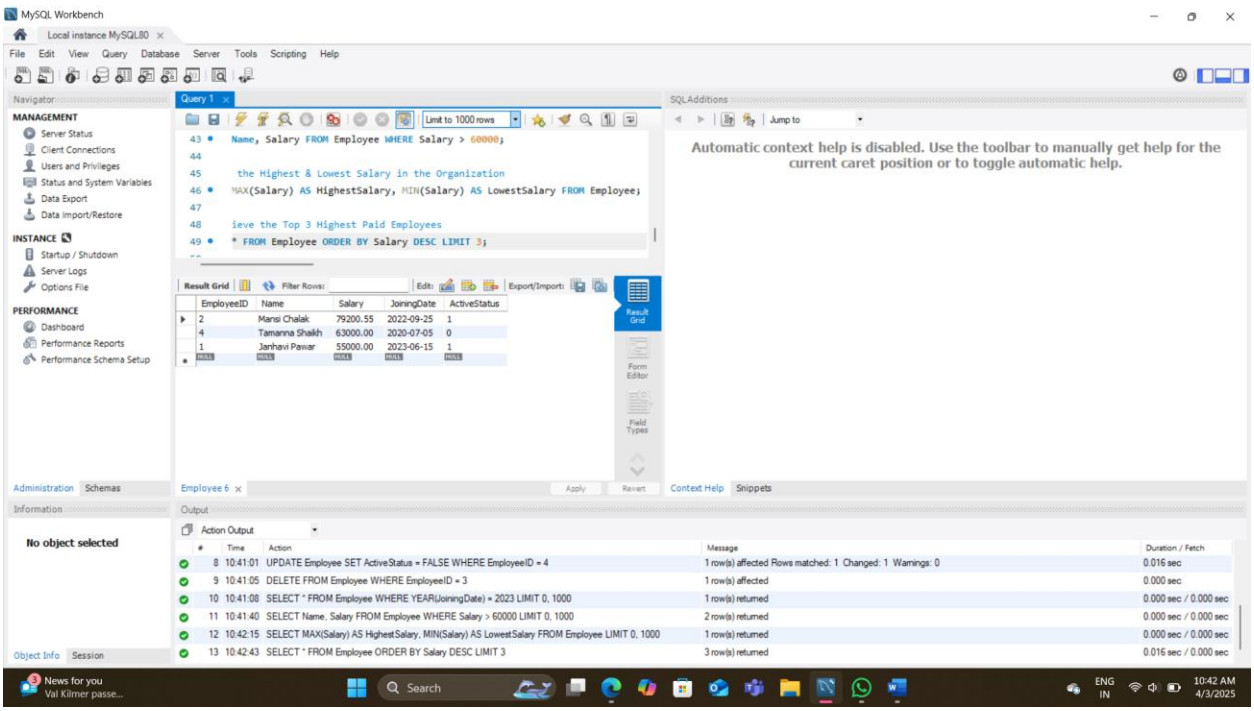
-- Find the Highest & Lowest Salary in the Organization

SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS LowestSalary FROM Employee;



-- Retrieve the Top 3 Highest Paid Employees

SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;



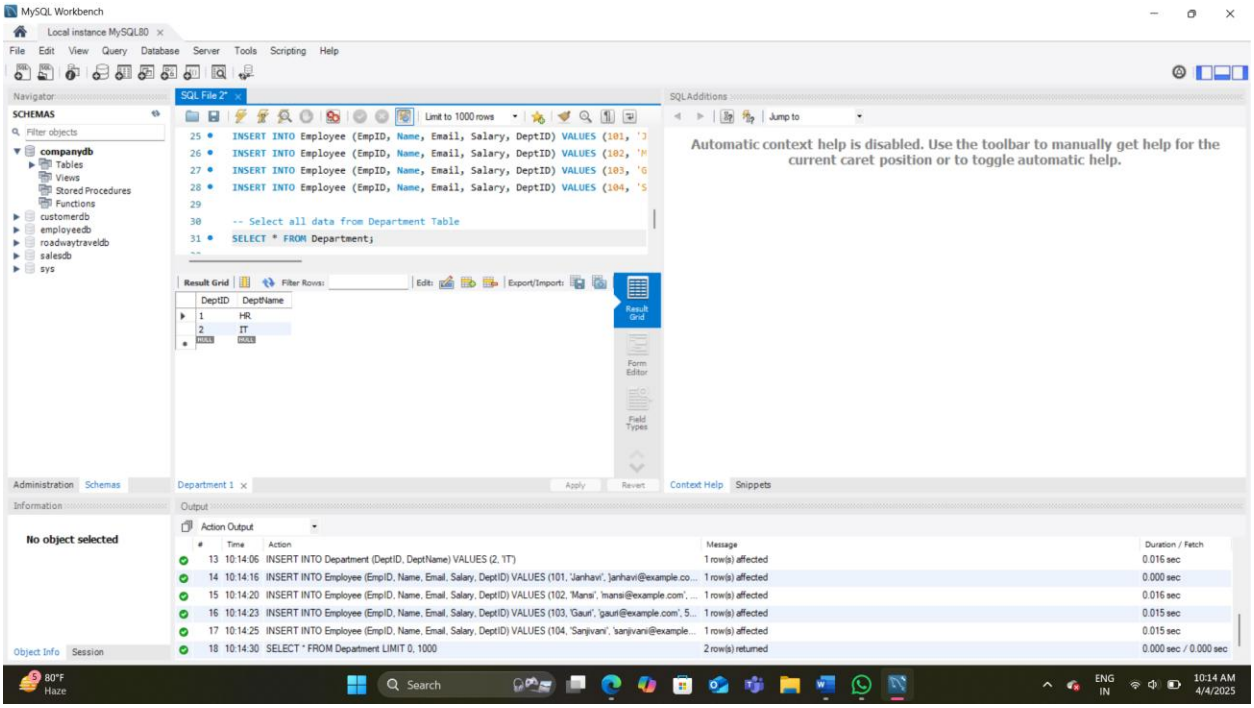
Practical 4:

Aim: Create a table to store employee information with constraints like Primary Key, Foreign Key, and Unique.

Code:

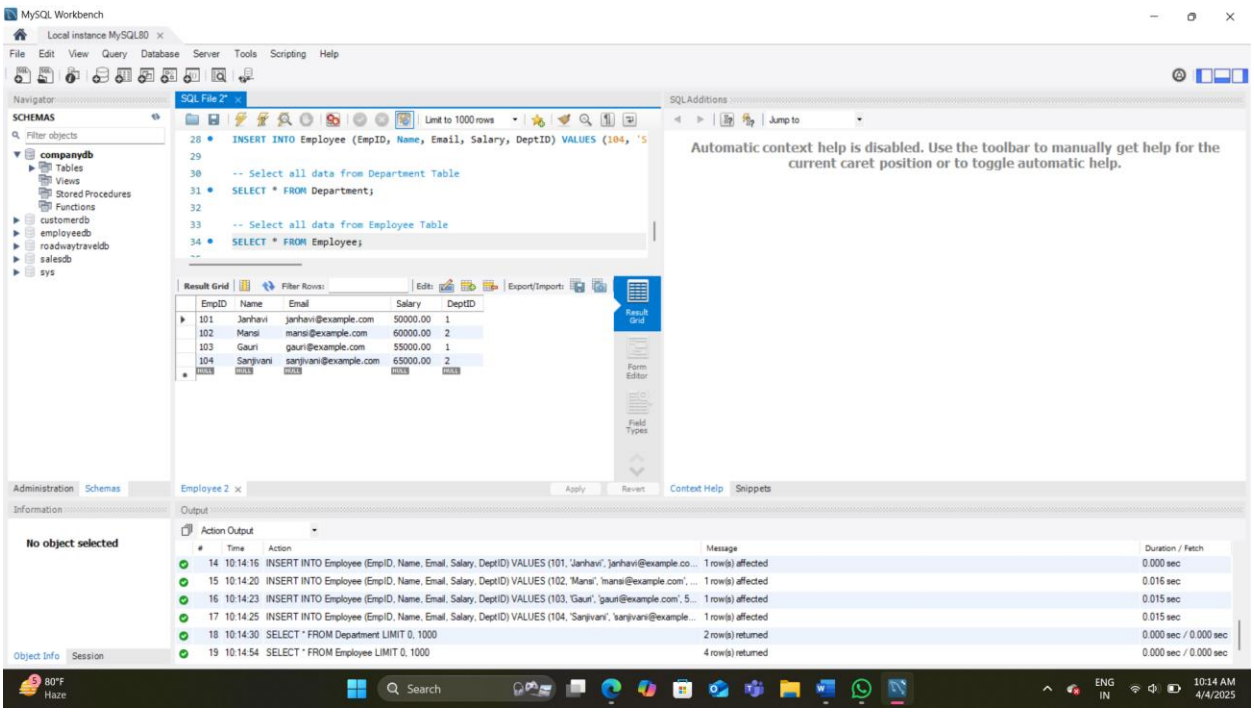
```
-- Drop existing tables if they exist

DROP TABLE IF EXISTS Employee;

DROP TABLE IF EXISTS Department;


-- Create the Department Table

CREATE TABLE Department (

  DeptID INT PRIMARY KEY,

  DeptName VARCHAR(50) UNIQUE

);
```

```
-- Create the Employee Table
CREATE TABLE Employee (
   EmpID INT PRIMARY KEY,
   Name VARCHAR(100) NOT NULL,
   Email VARCHAR(100) UNIQUE,
   Salary DECIMAL(10,2) CHECK (Salary > 0),
   DeptID INT REFERENCES Department(DeptID)
);

-- Insert Data into Department Table
INSERT INTO Department (DeptID, DeptName) VALUES (1, 'HR');
INSERT INTO Department (DeptID, DeptName) VALUES (2, 'IT');

-- Insert Data into Employee Table
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (101, 'Janhavi',
'janhavi@example.com', 50000.00, 1);
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (102, 'Mansi',
'mansi@example.com', 60000.00, 2);
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (103, 'Gauri',
'gauri@example.com', 55000.00, 1);
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (104, 'Sanjivani',
'sanjivani@example.com', 65000.00, 2);

-- Select all data from Department Table
SELECT * FROM Department;
```



-- Select all data from Employee Table

SELECT * FROM Employee;

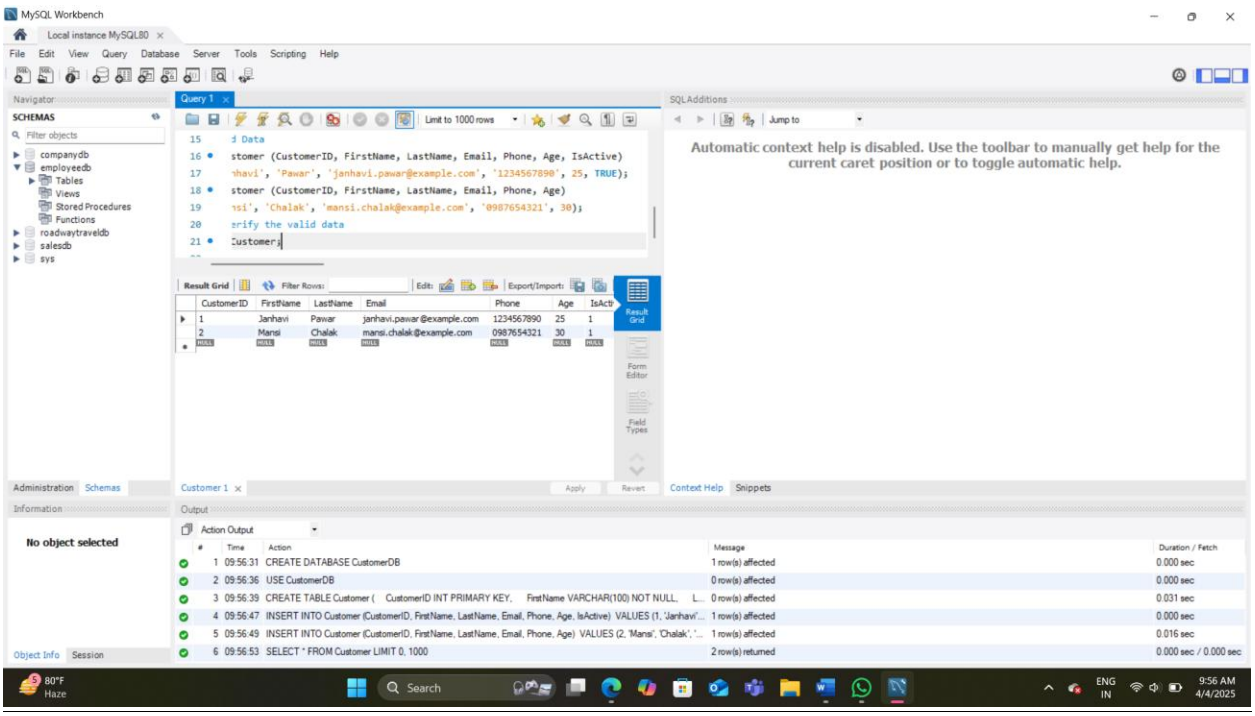


Practical 5:

Aim: To Test constraints like PRIMARY KEY, UNIQUE, and CHECK by inserting invalid data into the Employee table.

Code:

```
-- Create the Database
CREATE DATABASE CustomerDB;
-- Use the Database
USE CustomerDB;
-- Create the Customer Table
CREATE TABLE Customer (
  CustomerID INT PRIMARY KEY,
  FirstName VARCHAR(100) NOT NULL,
  LastName VARCHAR(100) NOT NULL,
  Email VARCHAR(100) UNIQUE,
  Phone VARCHAR(15),
  Age INT CHECK (Age >= 18),
  IsActive BOOLEAN DEFAULT TRUE
);
-- Insert Valid Data
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age, IsActive)
VALUES (1, 'Janhavi', 'Pawar', 'janhavi.pawar@example.com', '1234567890', 25, TRUE);
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)
VALUES (2, 'Mansi', 'Chalak', 'mansi.chalak@example.com', '0987654321', 30);
-- Select to verify the valid data
SELECT * FROM Customer;
```



Practical 6:

Aim: Use DDL commands to create tables and DML commands to insert, update, and delete data. Write SELECT queries to retrieve and verify data changes.

Code:

-- Create the Database

CREATE DATABASE EmployeeDB;

-- Use the Database

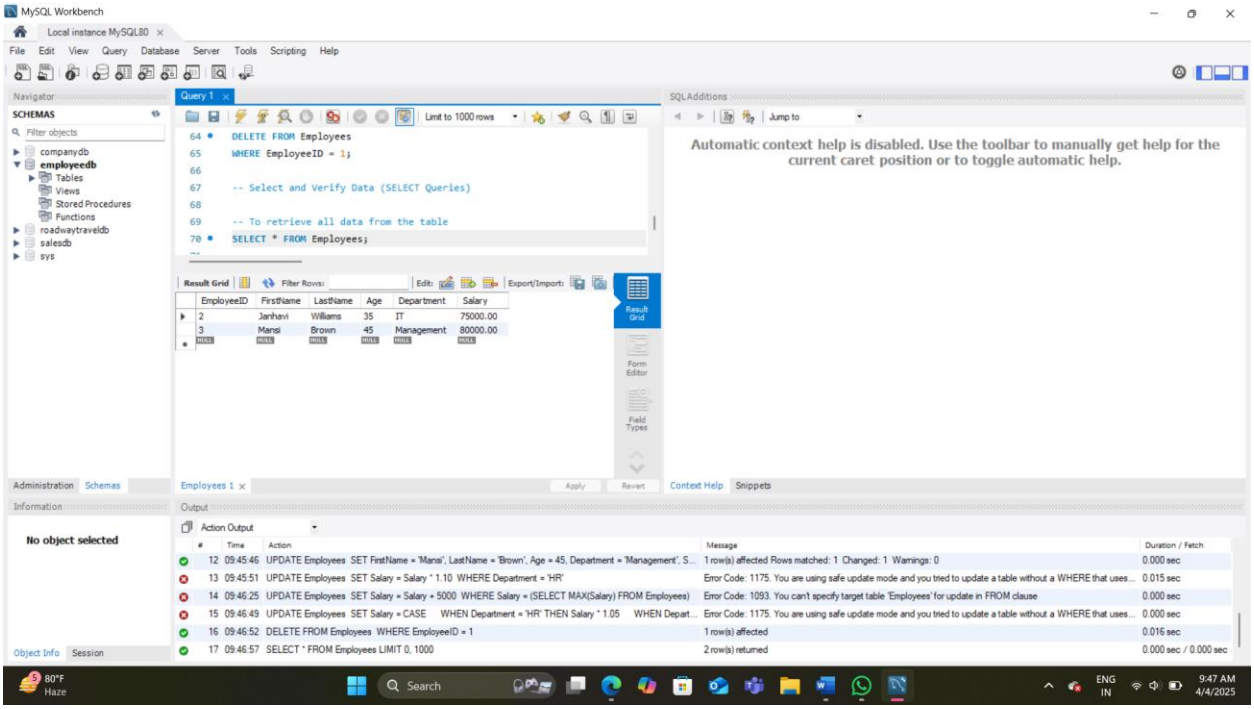
USE EmployeeDB;

-- Create the Employees Table

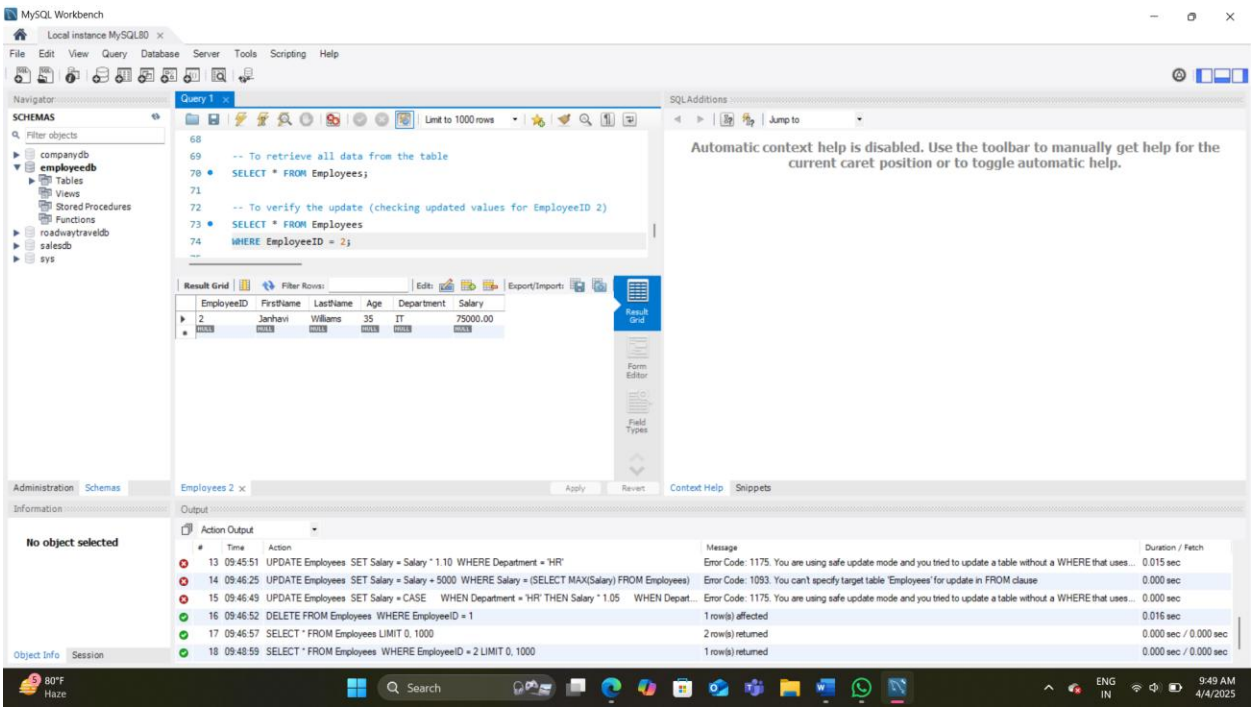
```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   Age INT,
   Department VARCHAR(50),
   Salary DECIMAL(10, 2)
);

-- Insert Data into Employees Table
INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department, Salary)
VALUES (1, 'Janhavi', 'Pawar', 28, 'HR', 50000.00);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department, Salary)
VALUES (2, 'Mansi', 'Chalak', 35, 'IT', 65000.00);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department, Salary)
VALUES (3, 'Gauri', 'Dani', 40, 'Finance', 75000.00);

-- Updates (DML Commands)

-- 1. Update a single column (e.g., update salary for EmployeeID 2)
UPDATE Employees
SET Salary = 70000.00
WHERE EmployeeID = 2;
```

```
-- 2. Update multiple columns for a specific row (e.g., update name and salary for EmployeeID 2)

UPDATE Employees

SET FirstName = 'Janhavi', LastName = 'Williams', Salary = 75000.00

WHERE EmployeeID = 2;


-- 3. Update entire tuple (all columns for EmployeeID 3)

UPDATE Employees

SET FirstName = 'Mansi', LastName = 'Brown', Age = 45, Department = 'Management', Salary = 80000.00

WHERE EmployeeID = 3;


-- 4. Update with a condition (e.g., increase salary by 10% for all employees in HR)

UPDATE Employees

SET Salary = Salary * 1.10

WHERE Department = 'HR';


-- 5. Update with a subquery (e.g., increase salary for Employee with highest salary)

UPDATE Employees

SET Salary = Salary + 5000

WHERE Salary = (SELECT MAX(Salary) FROM Employees);


-- 6. Update using a CASE statement (e.g., increase salary based on department)

UPDATE Employees

SET Salary = CASE

  WHEN Department = 'HR' THEN Salary * 1.05

  WHEN Department = 'IT' THEN Salary * 1.08
```

```
    WHEN Department = 'Finance' THEN Salary * 1.10
    ELSE Salary
END;
-- Delete Data from the Table (DML Command)
DELETE FROM Employees
WHERE EmployeeID = 1;


-- Select and Verify Data (SELECT Queries)
-- To retrieve all data from the table
SELECT * FROM Employees;
```



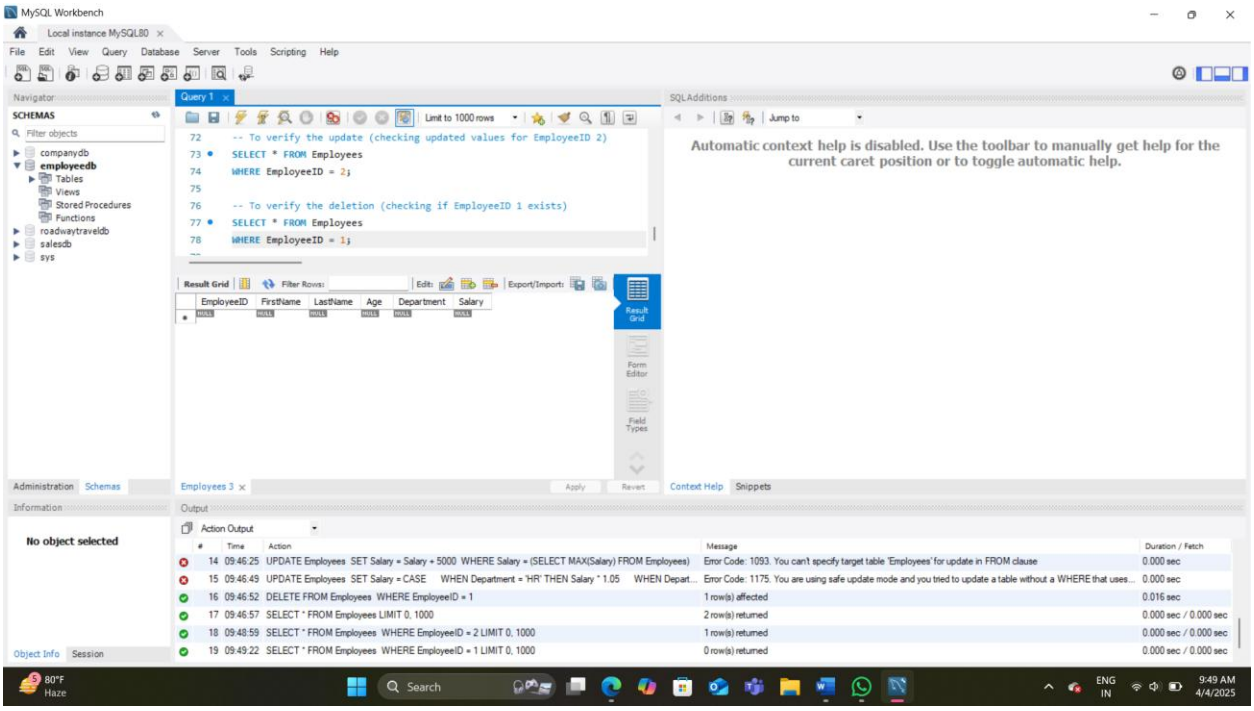

```
-- To verify the update (checking updated values for EmployeeID 2)
SELECT * FROM Employees
WHERE EmployeeID = 2;
```



-- To verify the deletion (checking if EmployeeID 1 exists)

SELECT * FROM Employees

WHERE EmployeeID = 1;



Practical 7:

Aim: Create a Sales table and use aggregate functions like COUNT, SUM, AVG, MIN, and MAX to summarize sales data and calculate statistics.

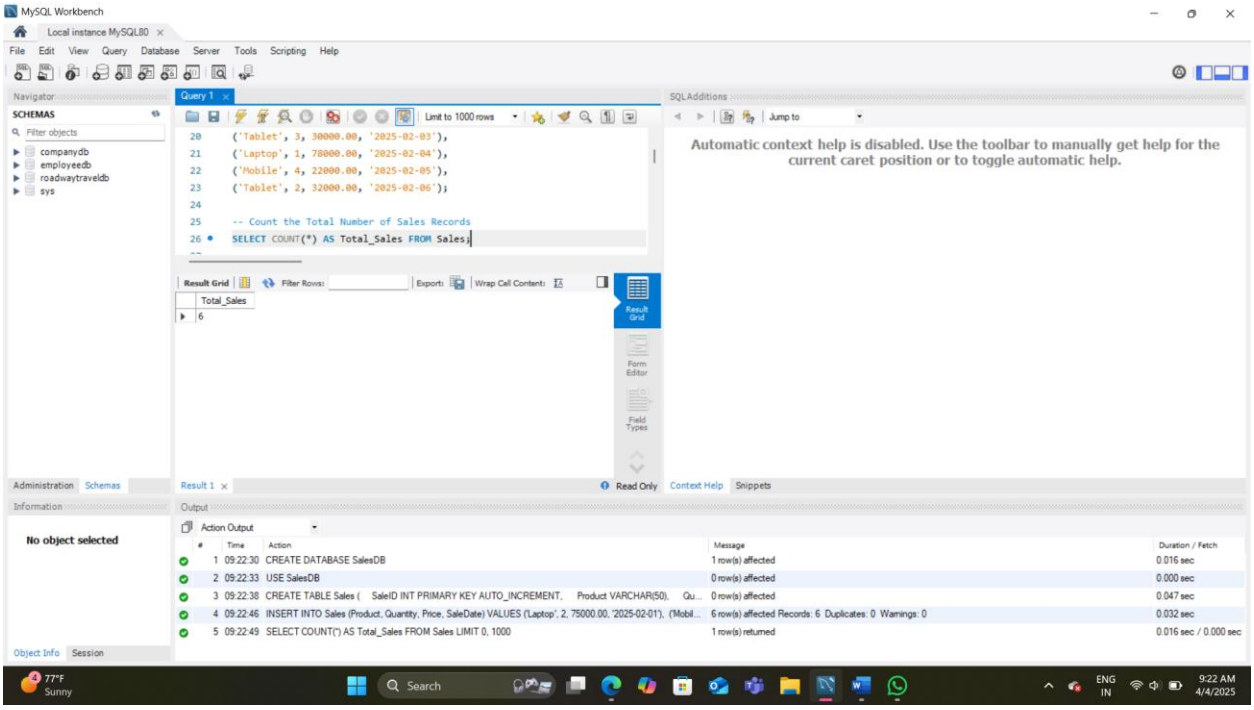
Code:

```
-- Create the Database
CREATE DATABASE SalesDB;

-- Use the Database
USE SalesDB;

-- Create the Sales Table
CREATE TABLE Sales (
  SaleID INT PRIMARY KEY AUTO_INCREMENT,
  Product VARCHAR(50),
  Quantity INT,
  Price DECIMAL(10,2),
  SaleDate DATE
);

-- Insert Sample Data into Sales Table
INSERT INTO Sales (Product, Quantity, Price, SaleDate) VALUES
('Laptop', 2, 75000.00, '2025-02-01'),
('Mobile', 5, 20000.00, '2025-02-02'),
('Tablet', 3, 30000.00, '2025-02-03'),
('Laptop', 1, 78000.00, '2025-02-04'),
('Mobile', 4, 22000.00, '2025-02-05'),
```

('Tablet', 2, 32000.00, '2025-02-06');

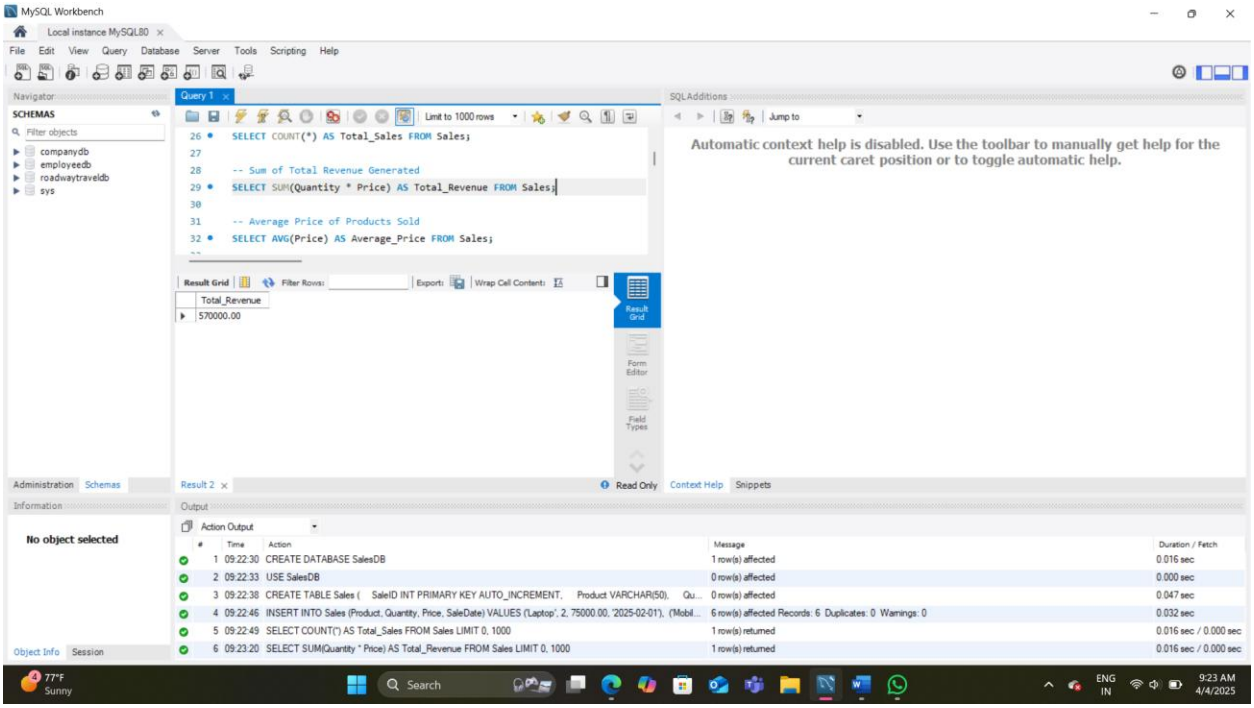
-- Count the Total Number of Sales Records

SELECT COUNT(*) AS Total_Sales FROM Sales;



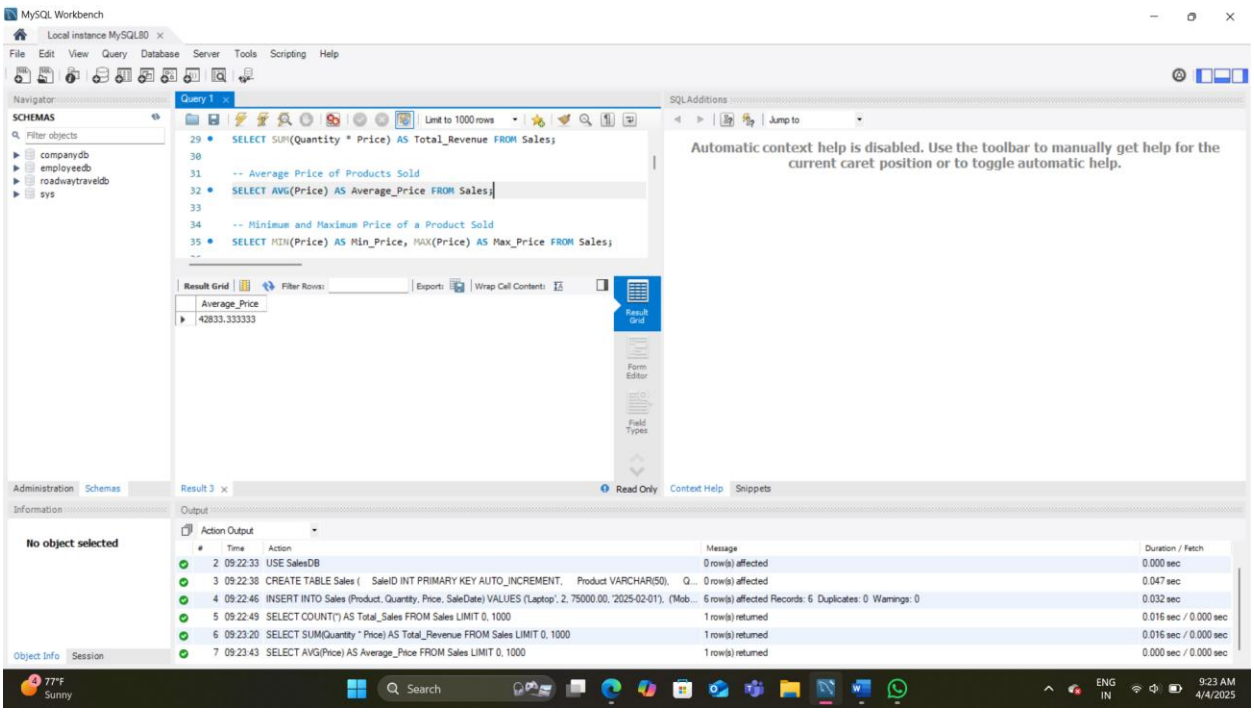
-- Sum of Total Revenue Generated

SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;



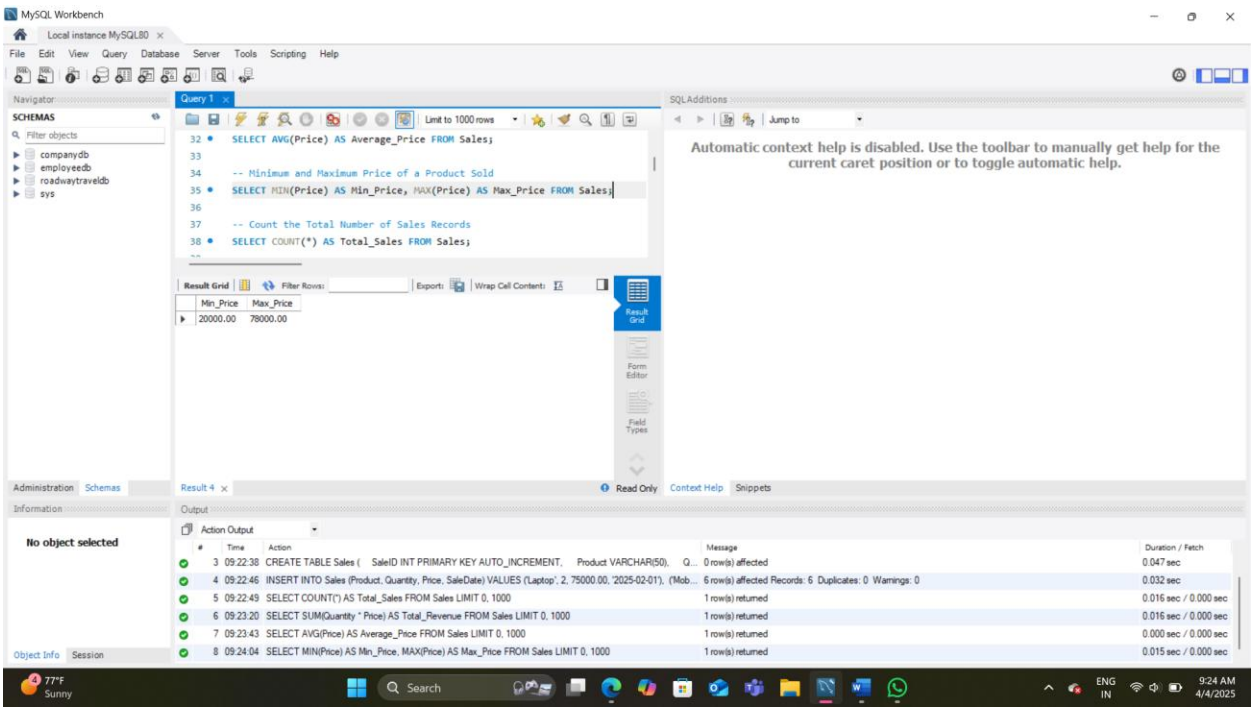
-- Average Price of Products Sold

SELECT AVG(Price) AS Average_Price FROM Sales;



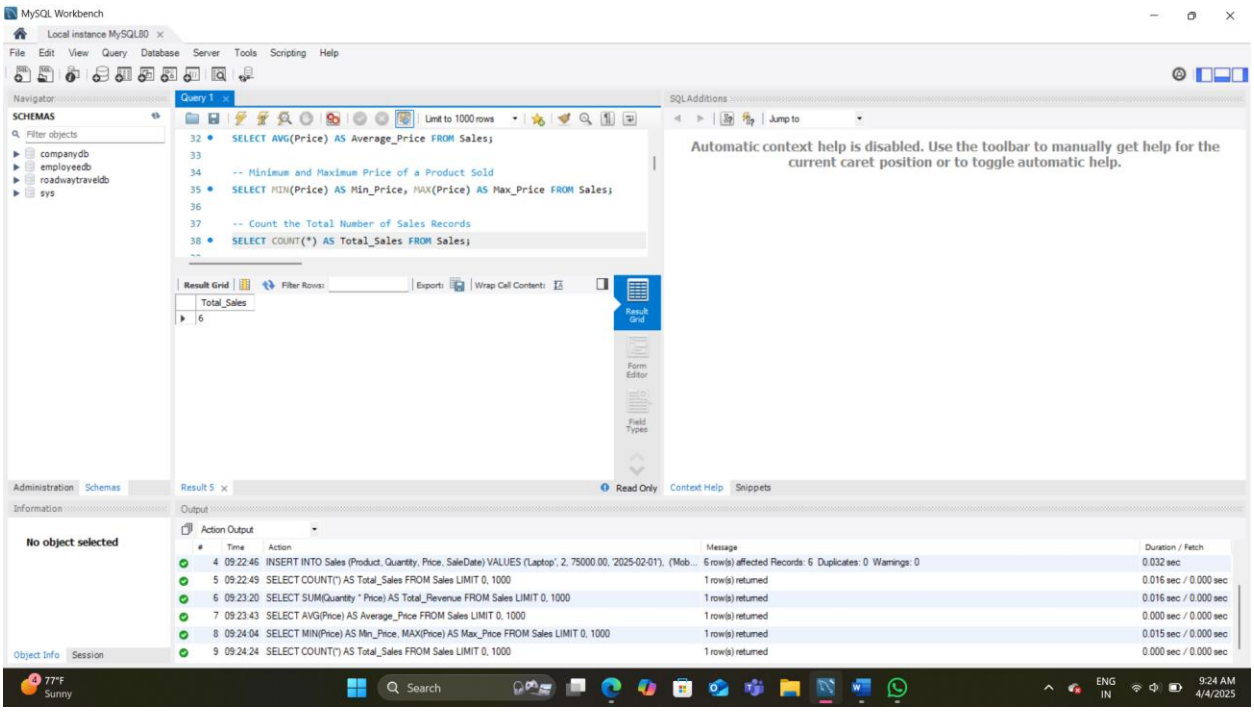
-- Minimum and Maximum Price of a Product Sold

SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;



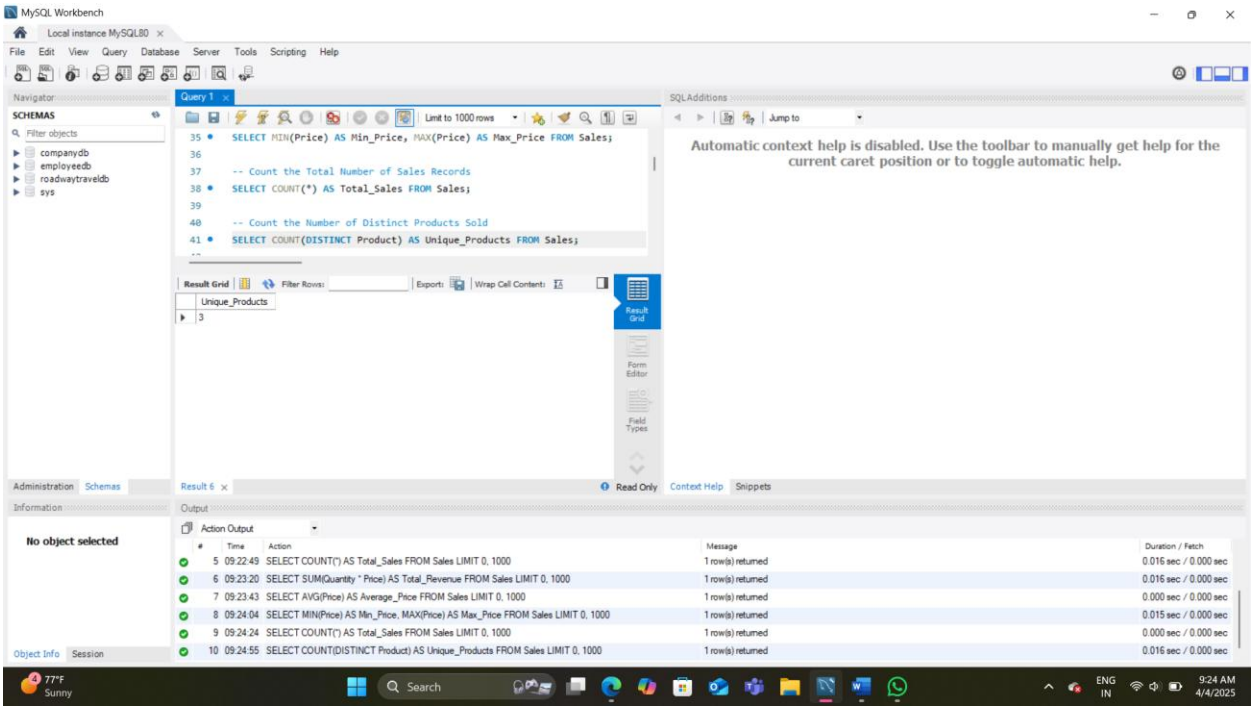
-- Count the Total Number of Sales Records

SELECT COUNT(*) AS Total_Sales FROM Sales;



-- Count the Number of Distinct Products Sold

SELECT COUNT(DISTINCT Product) AS Unique_Products FROM Sales;

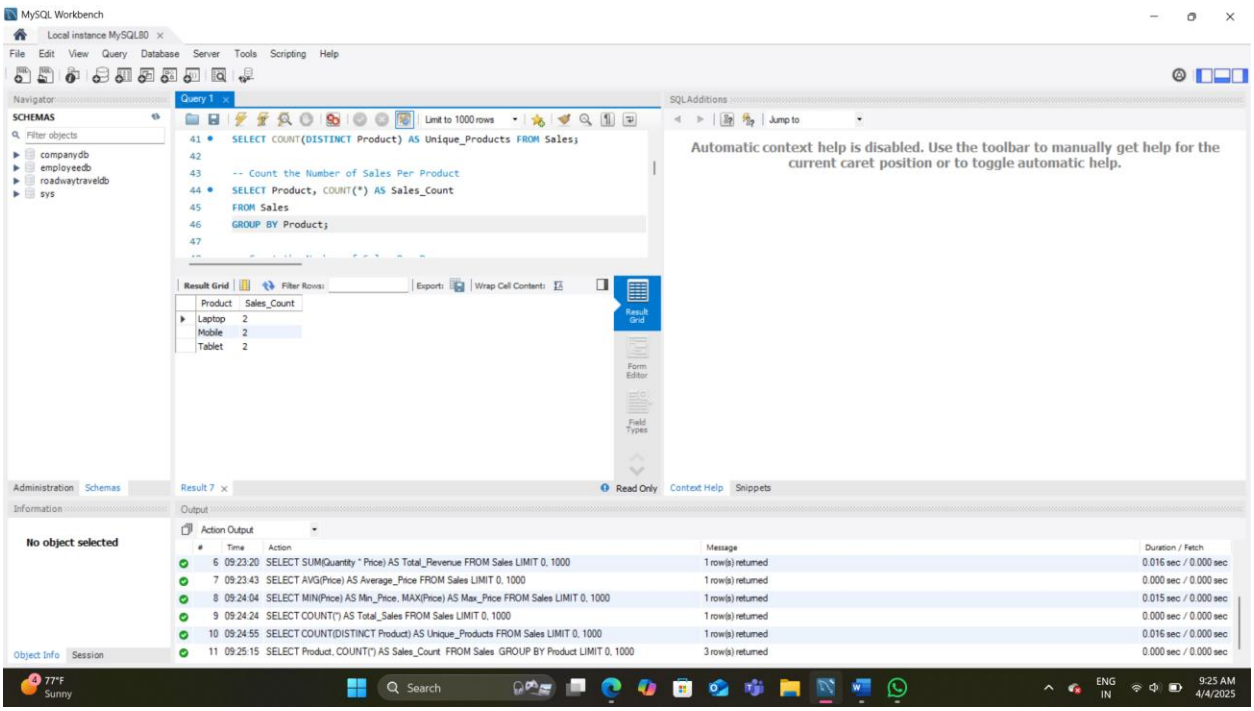


-- Count the Number of Sales Per Product

SELECT Product, COUNT(*) AS Sales_Count

FROM Sales

GROUP BY Product;

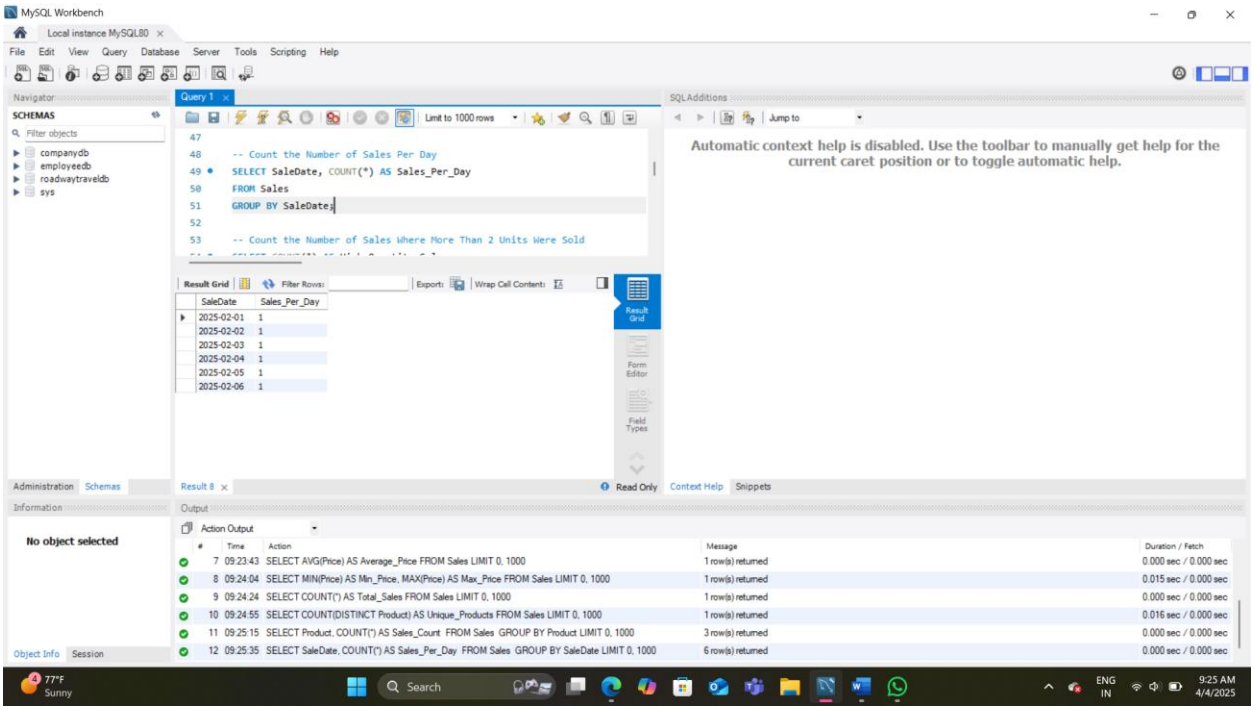


-- Count the Number of Sales Per Day

SELECT SaleDate, COUNT(*) AS Sales_Per_Day

FROM Sales

GROUP BY SaleDate;

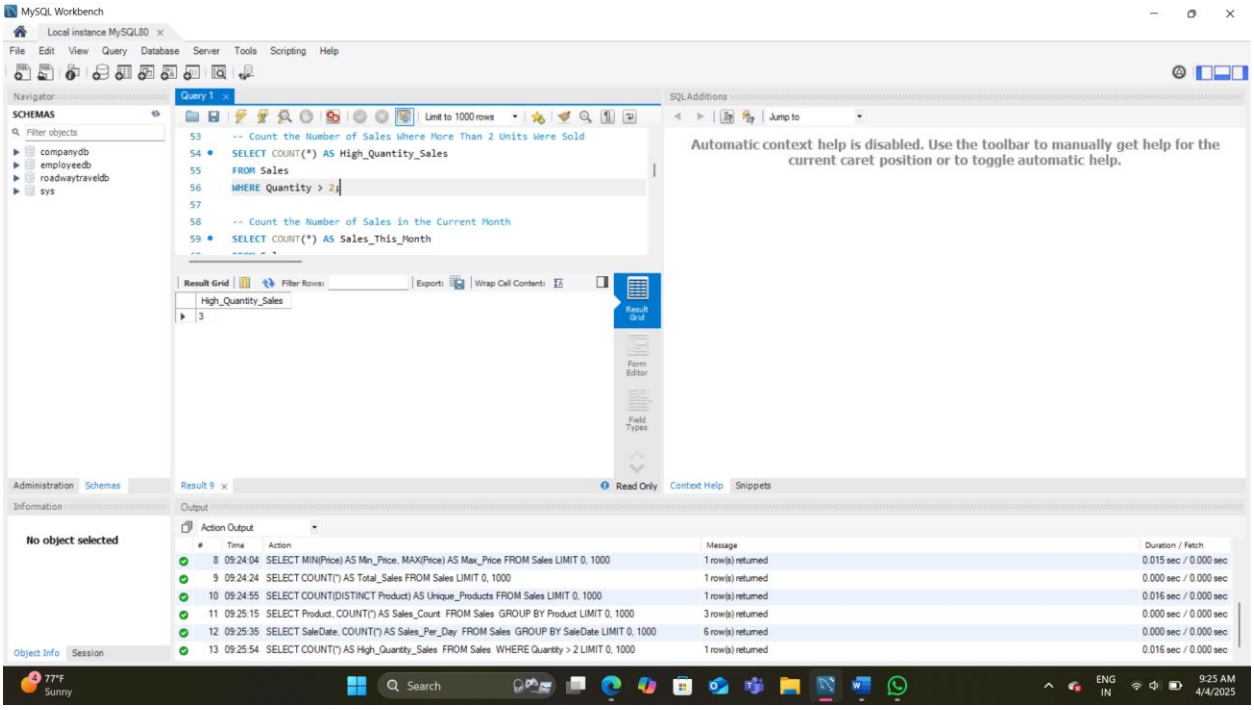


-- Count the Number of Sales Where More Than 2 Units Were Sold

SELECT COUNT(*) AS High_Quantity_Sales

FROM Sales

WHERE Quantity > 2;



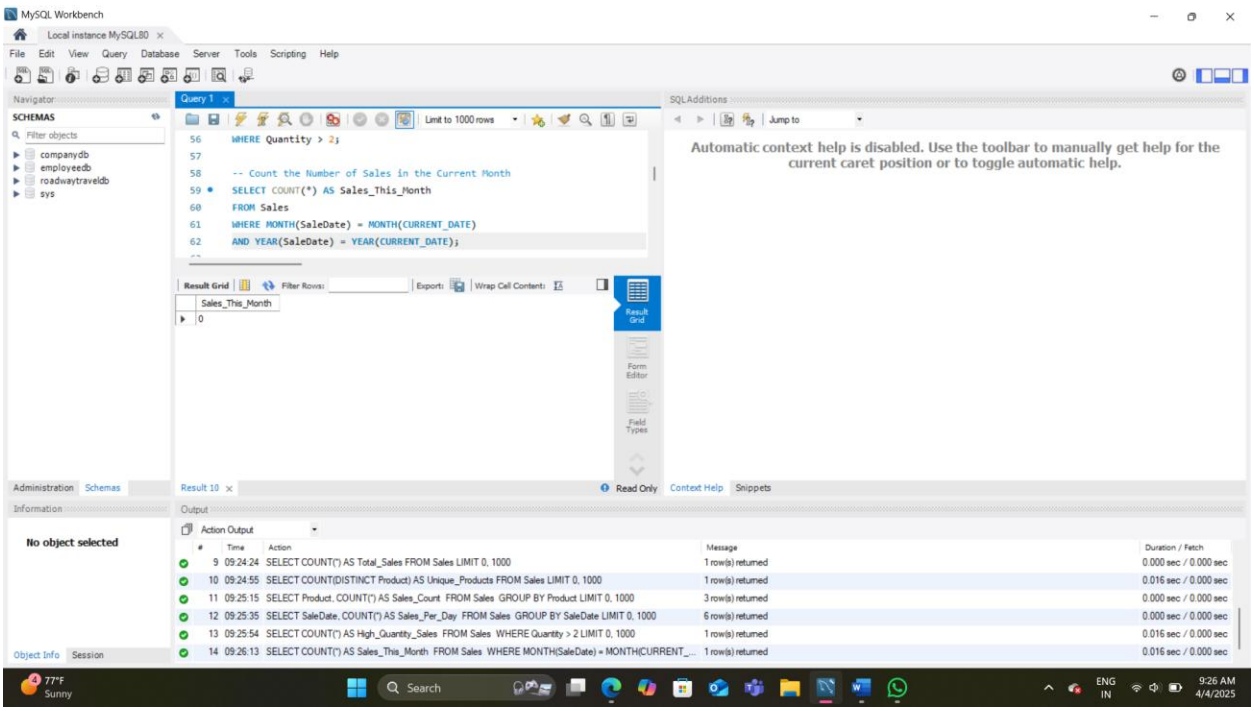
-- Count the Number of Sales in the Current Month

SELECT COUNT(*) AS Sales_This_Month

FROM Sales

WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT_DATE);

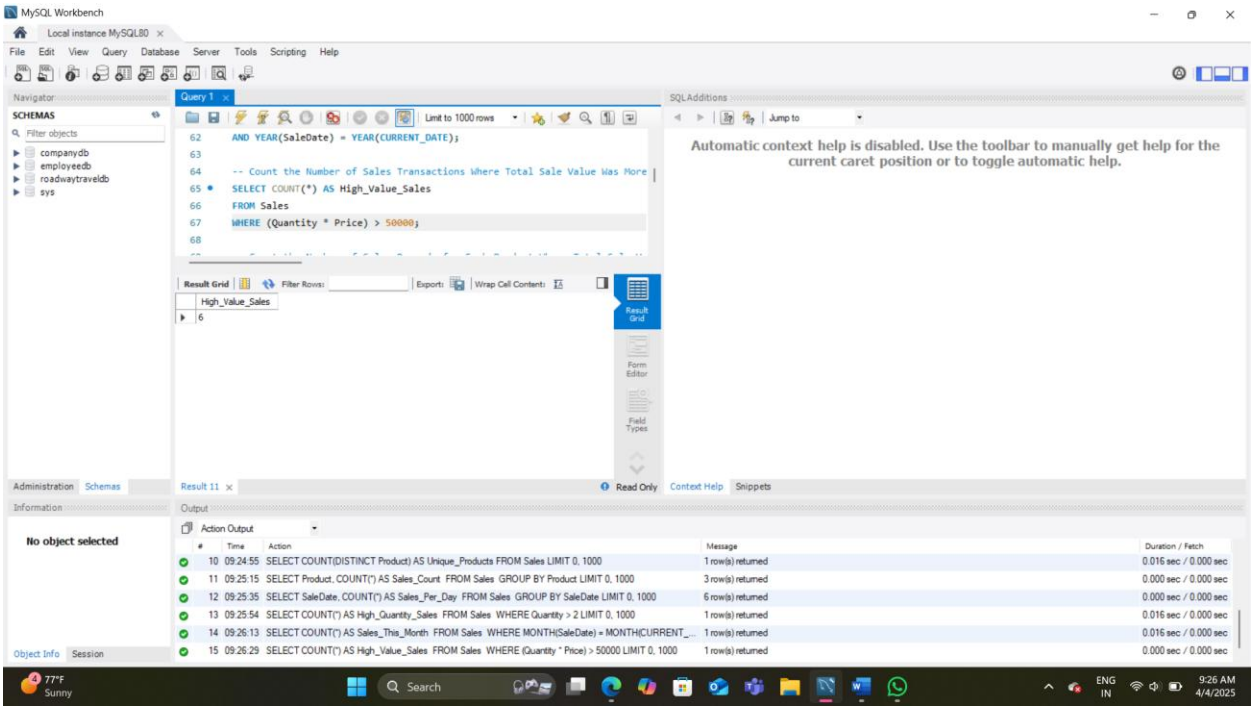


-- Count the Number of Sales Transactions Where Total Sale Value Was More Than ₹50,000

SELECT COUNT(*) AS High_Value_Sales

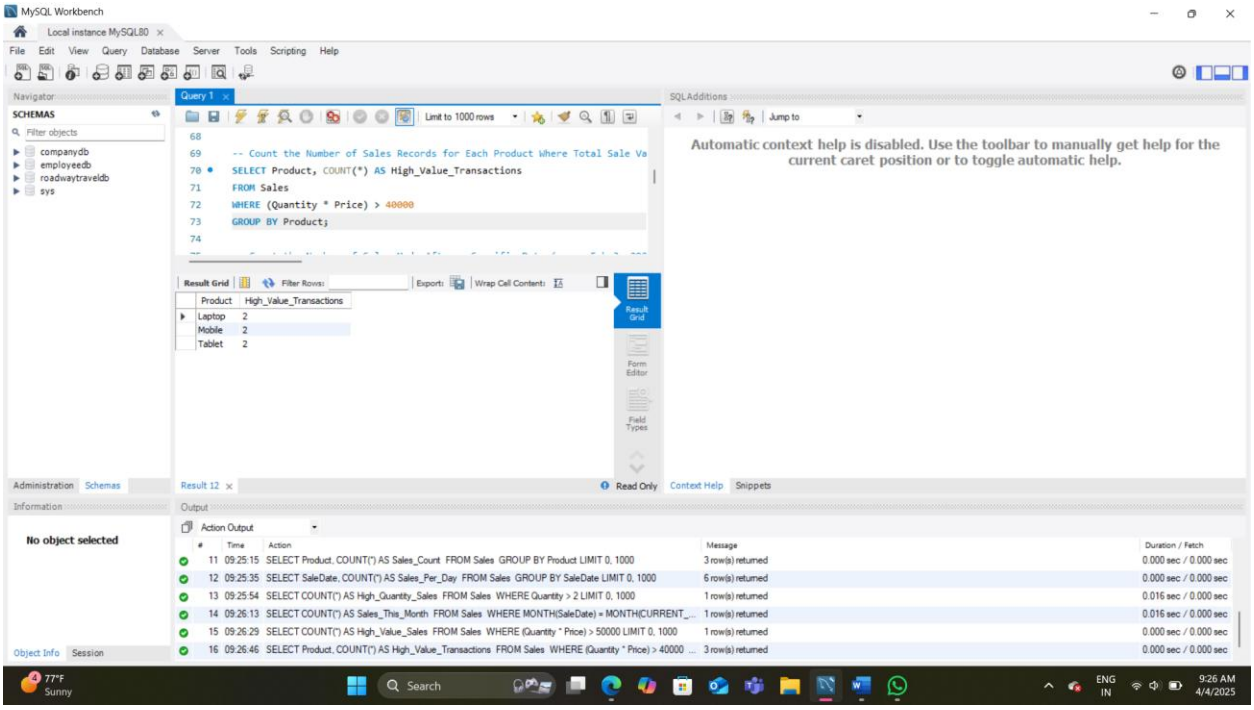
FROM Sales

WHERE (Quantity * Price) > 50000;



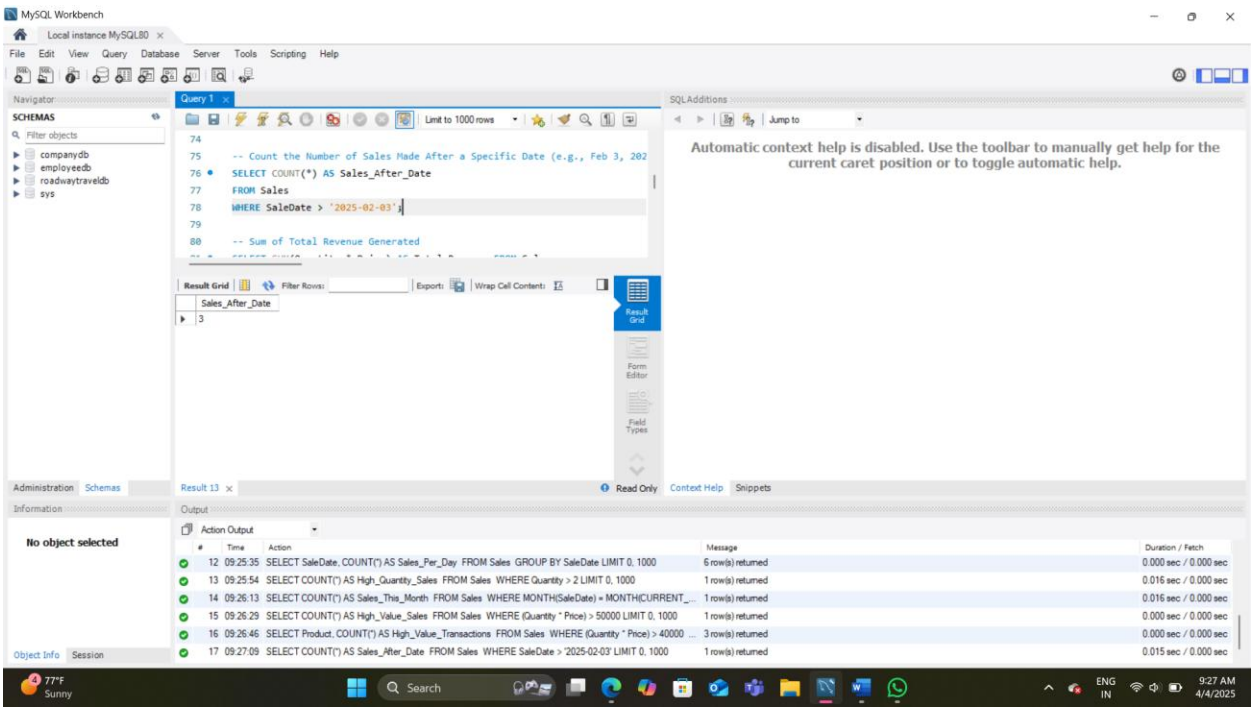
-- Count the Number of Sales Records for Each Product Where Total Sale Value Is Greater Than ₹40,000

```
SELECT Product, COUNT(*) AS High_Value_Transactions

FROM Sales

WHERE (Quantity * Price) > 40000

GROUP BY Product;
```



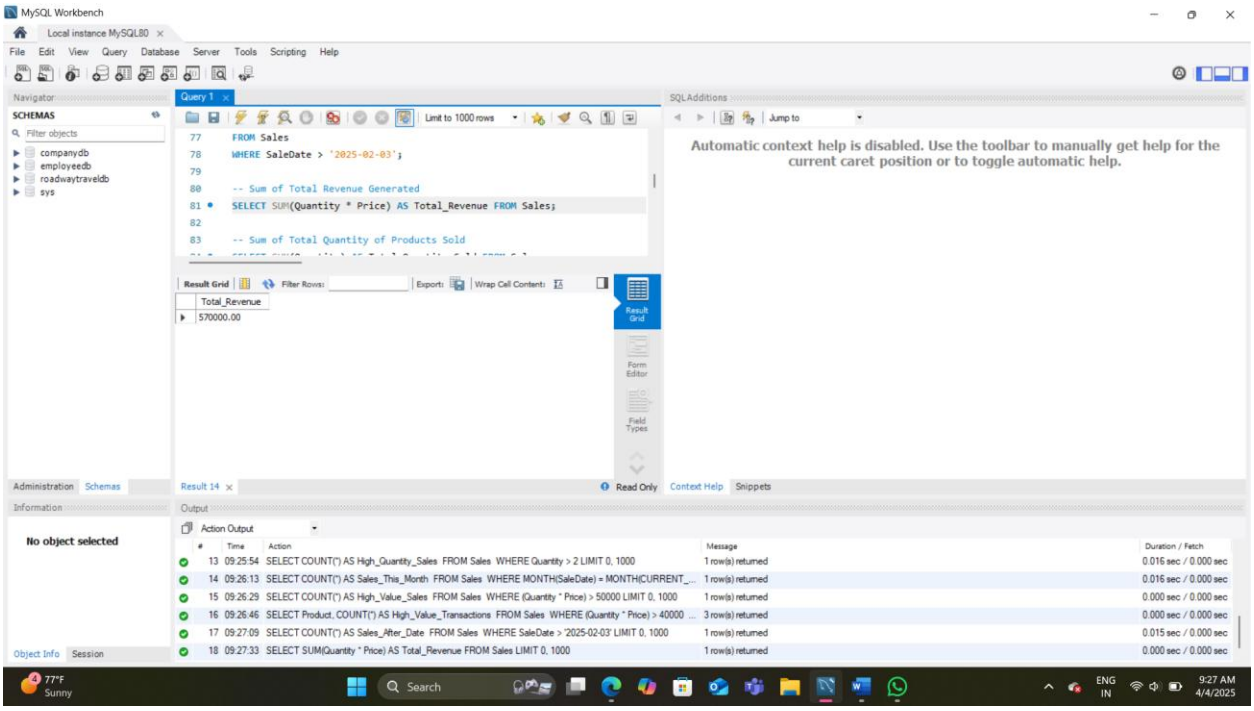
-- Count the Number of Sales Made After a Specific Date (e.g., Feb 3, 2025)

```
SELECT COUNT(*) AS Sales_After_Date

FROM Sales

WHERE SaleDate > '2025-02-03';
```



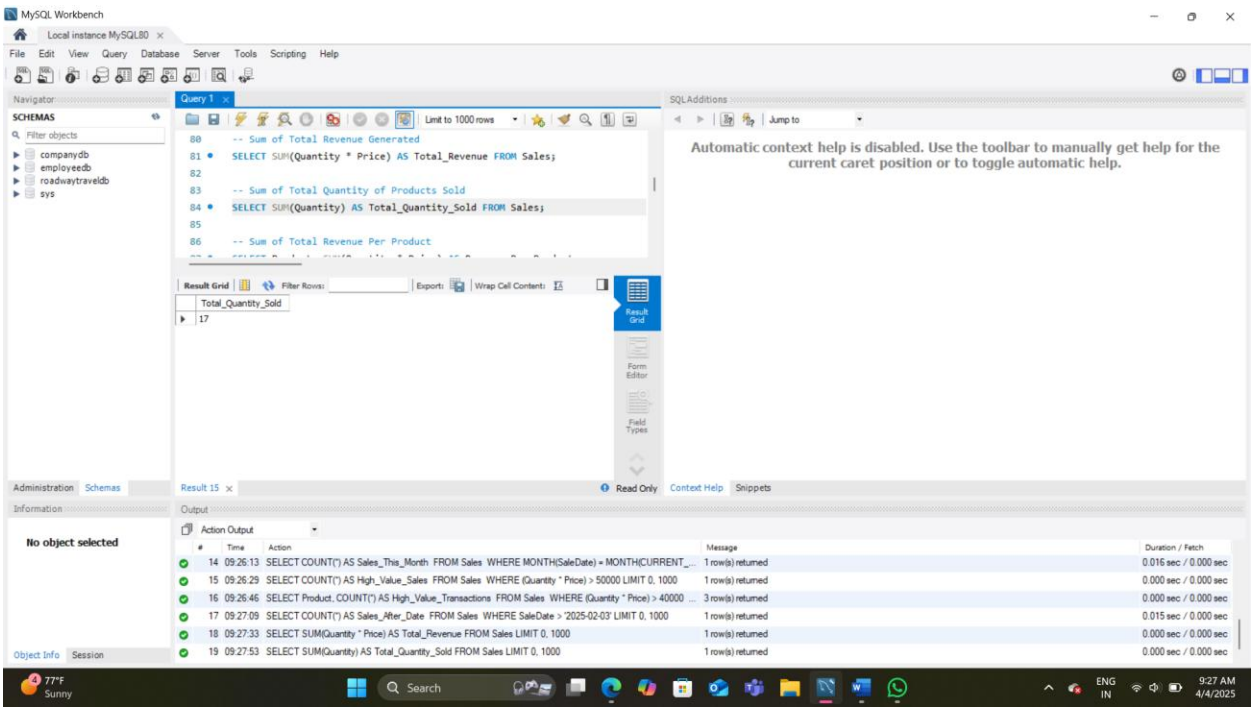
-- Sum of Total Revenue Generated

SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;



-- Sum of Total Quantity of Products Sold

SELECT SUM(Quantity) AS Total_Quantity_Sold FROM Sales;

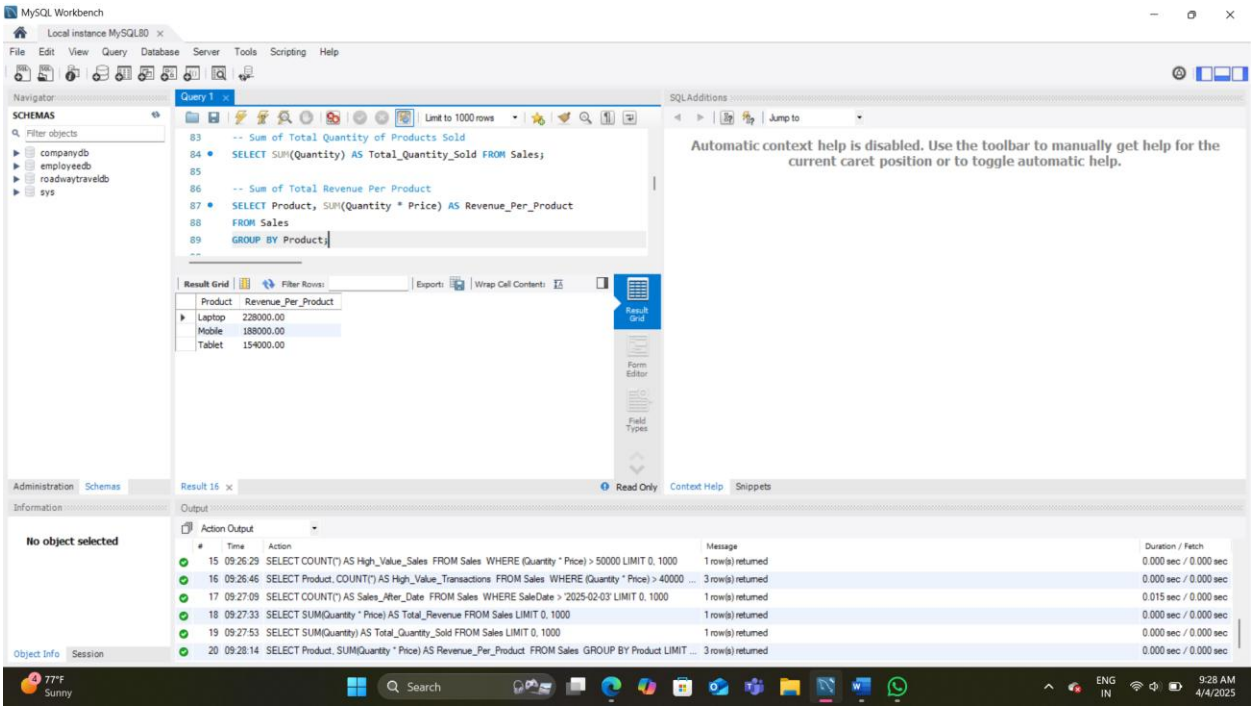


-- Sum of Total Revenue Per Product

SELECT Product, SUM(Quantity * Price) AS Revenue_Per_Product

FROM Sales

GROUP BY Product;

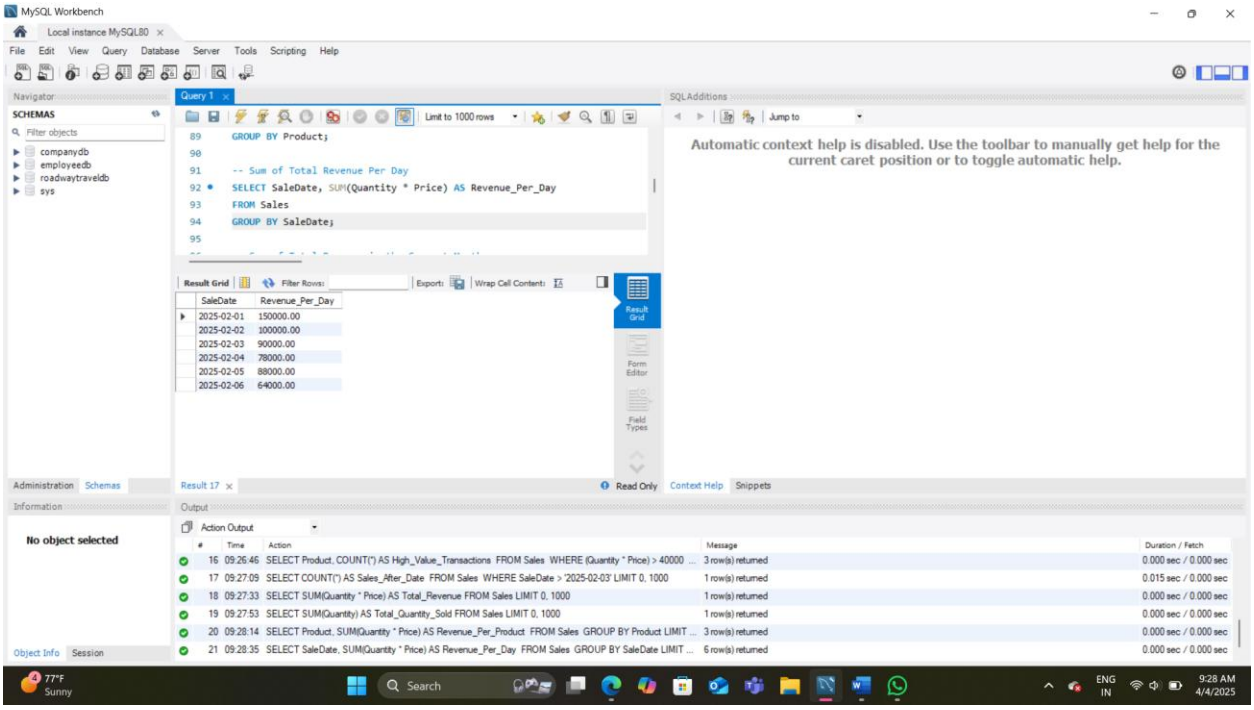


-- Sum of Total Revenue Per Day

SELECT SaleDate, SUM(Quantity * Price) AS Revenue_Per_Day

FROM Sales

GROUP BY SaleDate;



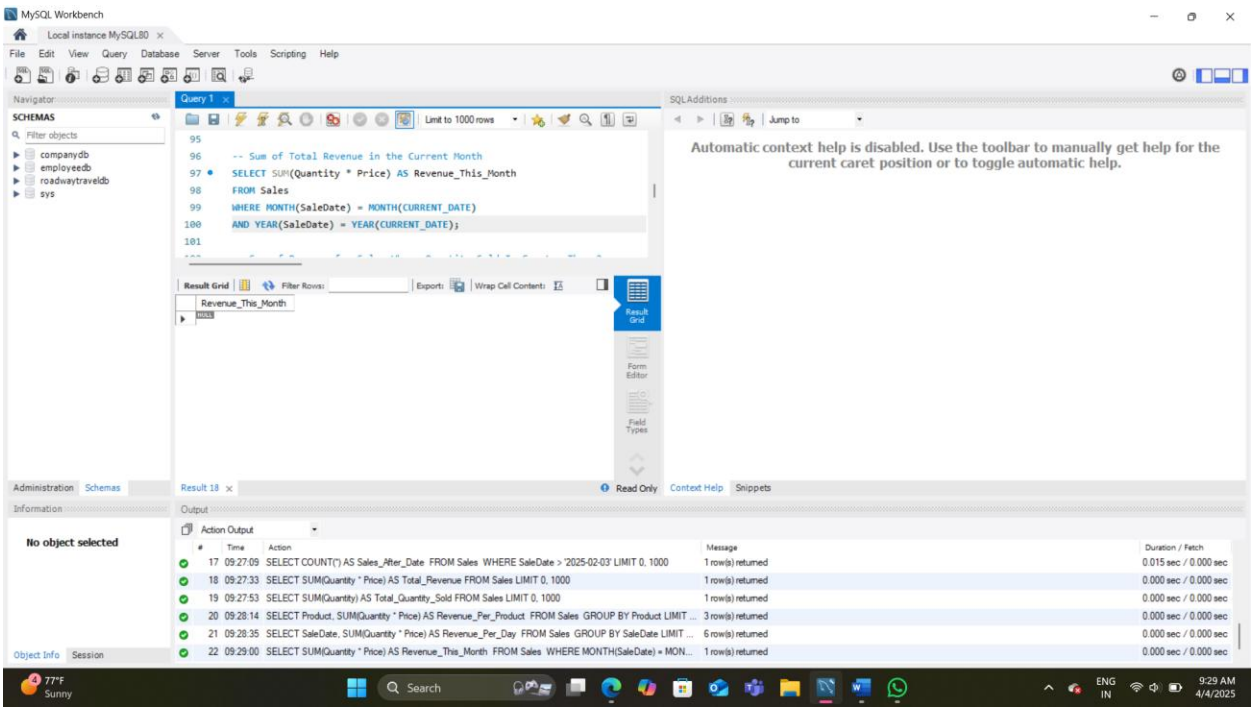
-- Sum of Total Revenue in the Current Month

SELECT SUM(Quantity * Price) AS Revenue_This_Month

FROM Sales

WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT_DATE);

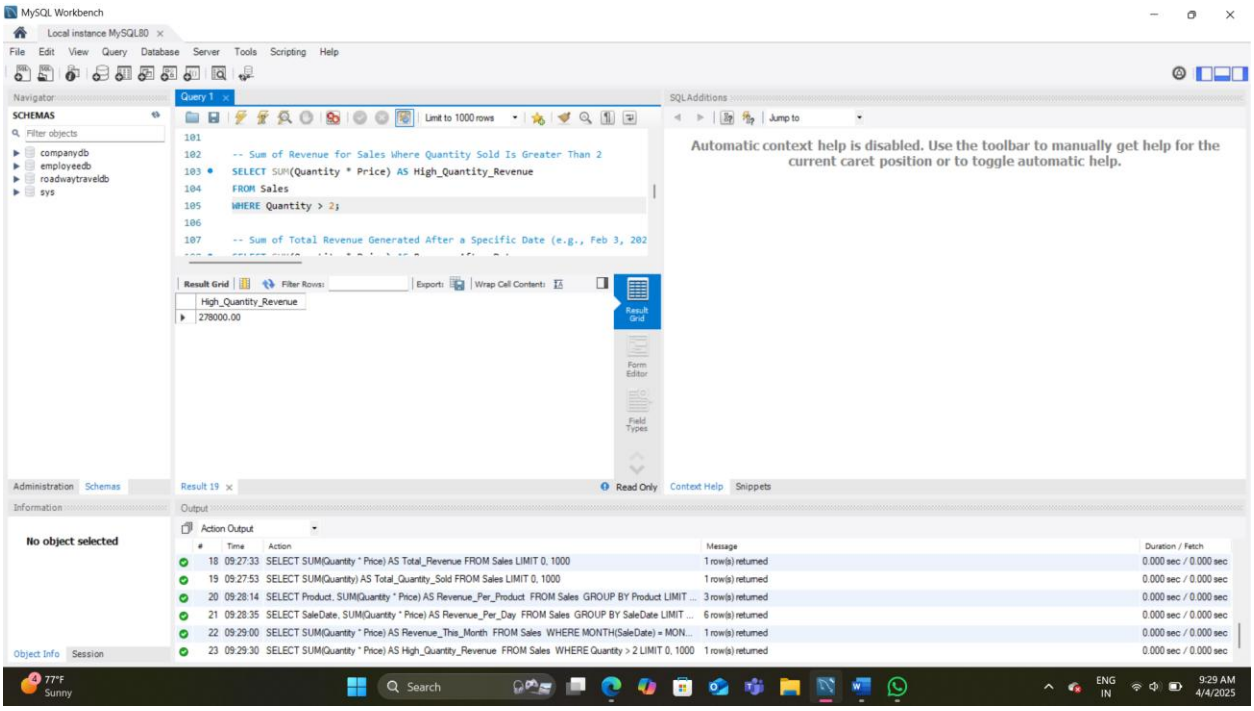


-- Sum of Revenue for Sales Where Quantity Sold Is Greater Than 2

SELECT SUM(Quantity * Price) AS High_Quantity_Revenue

FROM Sales

WHERE Quantity > 2;

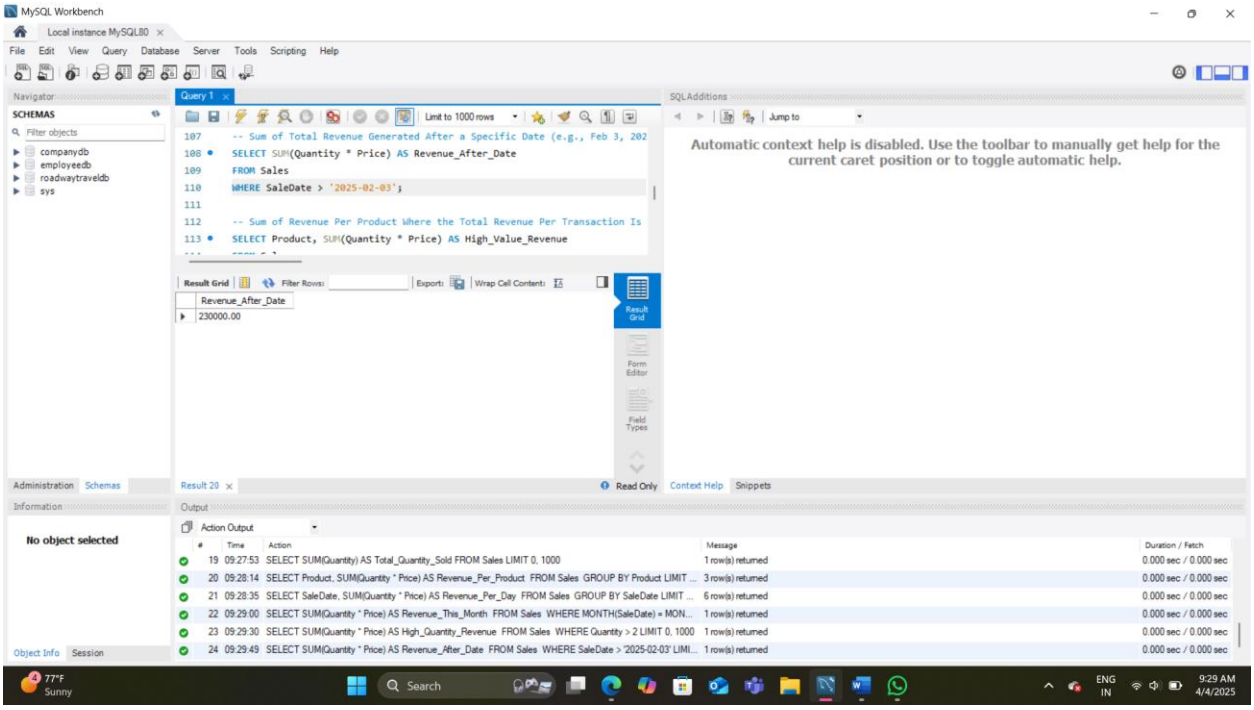


-- Sum of Total Revenue Generated After a Specific Date (e.g., Feb 3, 2025)

SELECT SUM(Quantity * Price) AS Revenue_After_Date

FROM Sales

WHERE SaleDate > '2025-02-03';



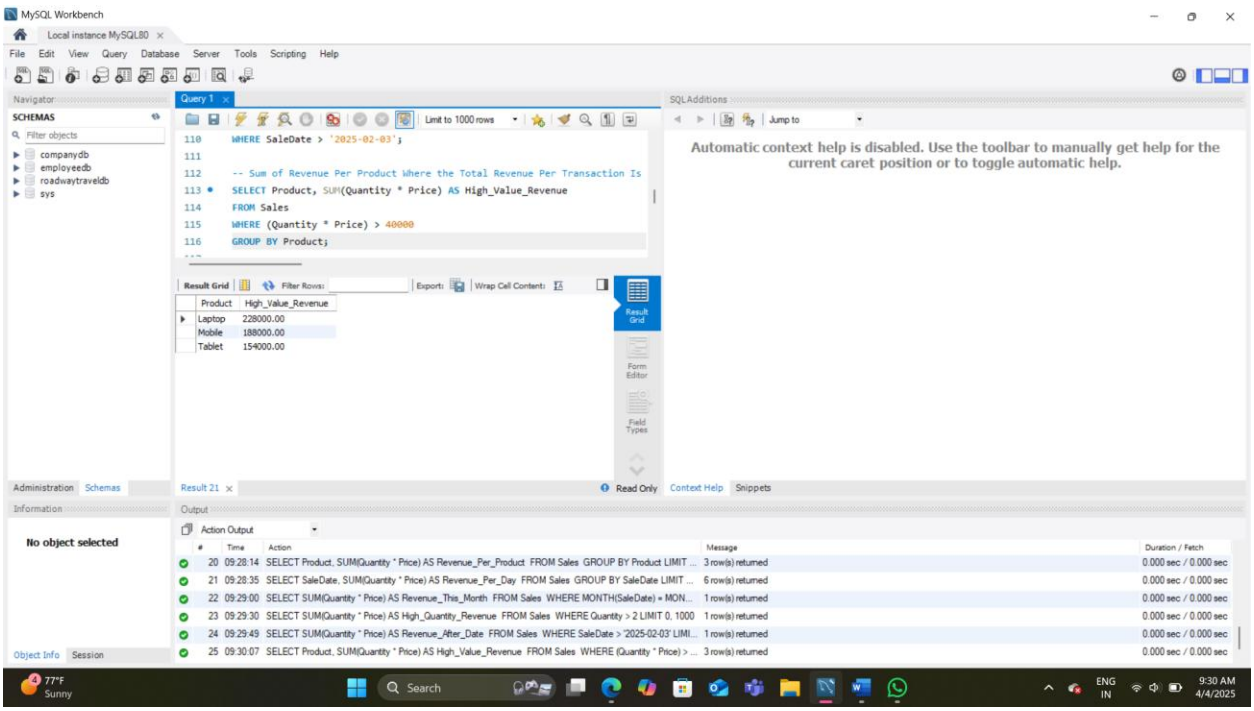
-- Sum of Revenue Per Product Where the Total Revenue Per Transaction Is Greater Than ₹40,000

SELECT Product, SUM(Quantity * Price) AS High_Value_Revenue

FROM Sales

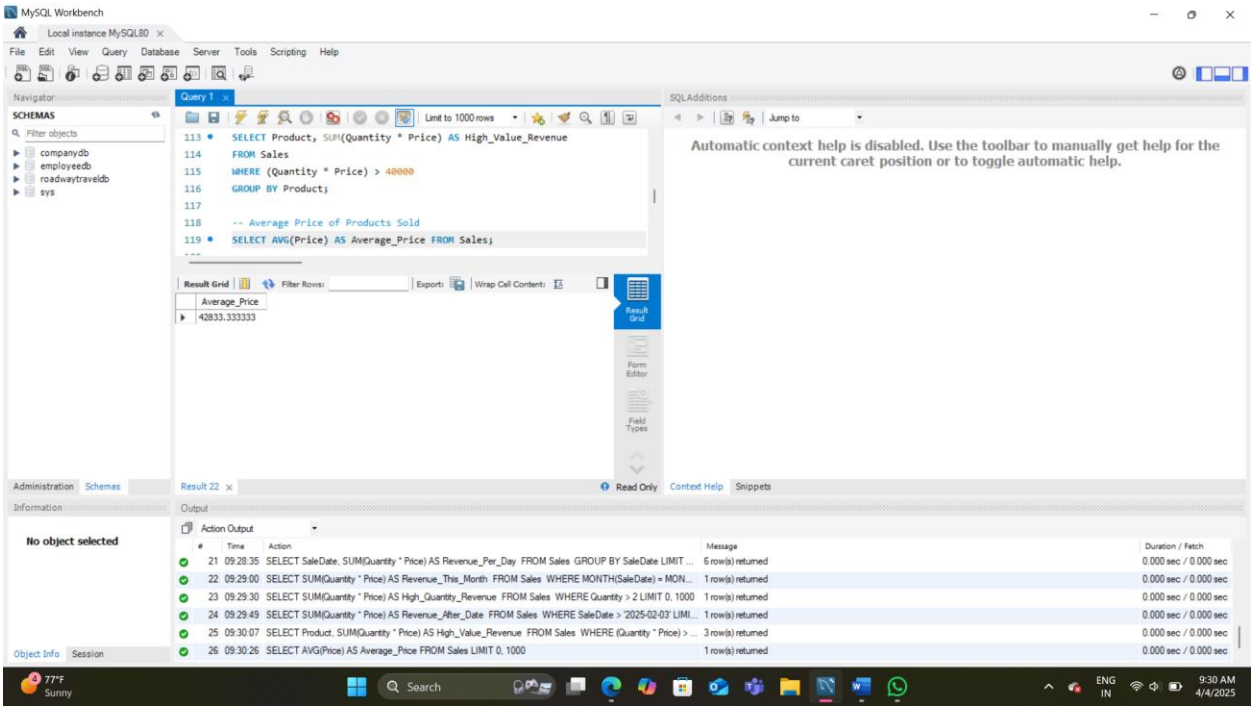
WHERE (Quantity * Price) > 40000

GROUP BY Product;



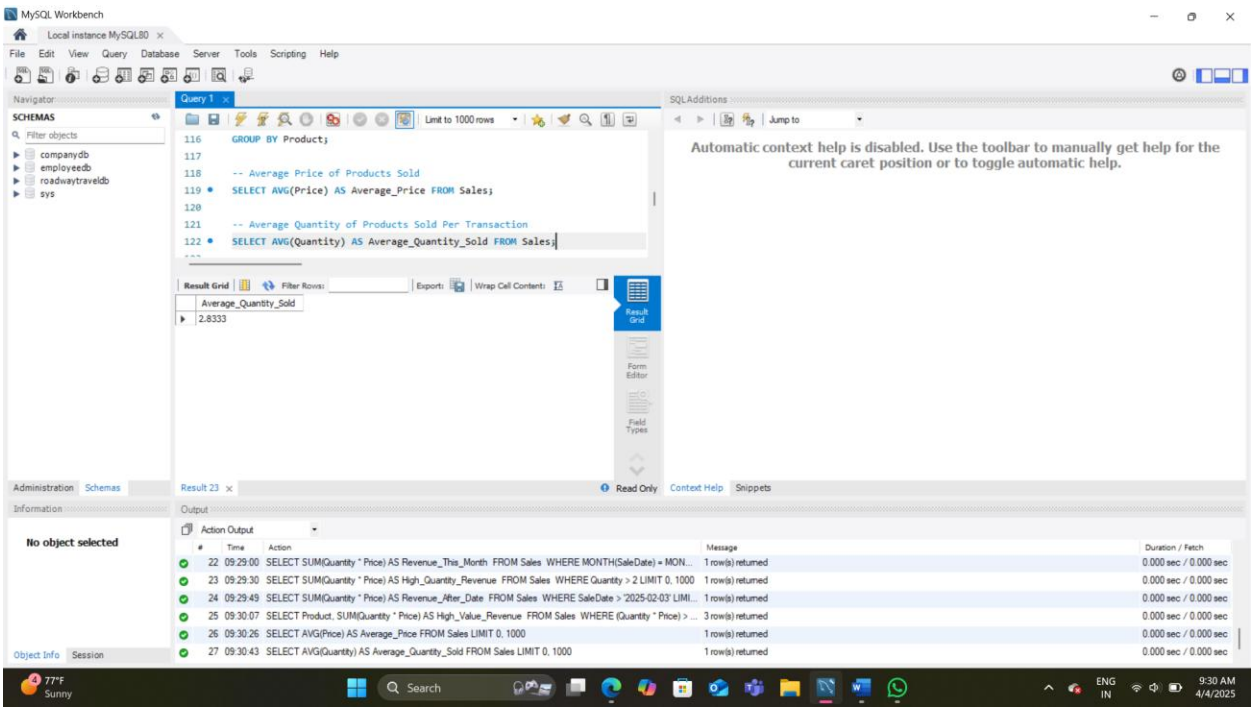
-- Average Price of Products Sold

SELECT AVG(Price) AS Average_Price FROM Sales;



-- Average Quantity of Products Sold Per Transaction

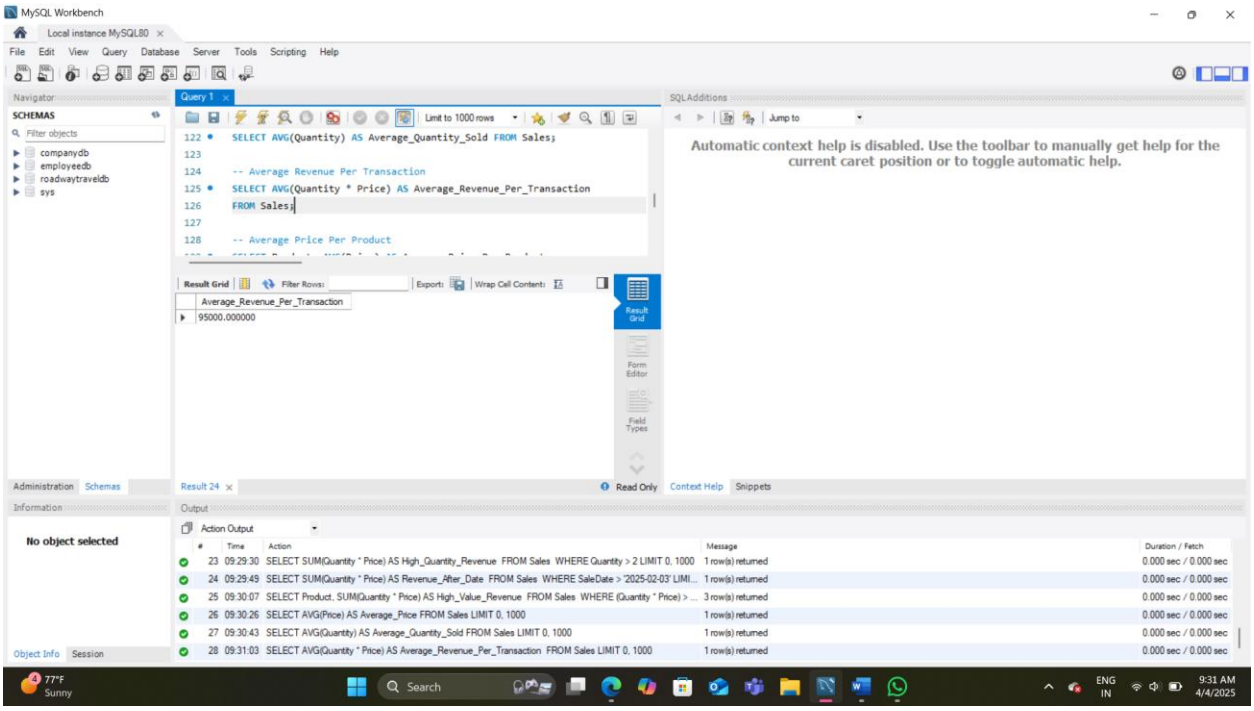
SELECT AVG(Quantity) AS Average_Quantity_Sold FROM Sales;



-- Average Revenue Per Transaction

SELECT AVG(Quantity * Price) AS Average_Revenue_Per_Transaction

FROM Sales;

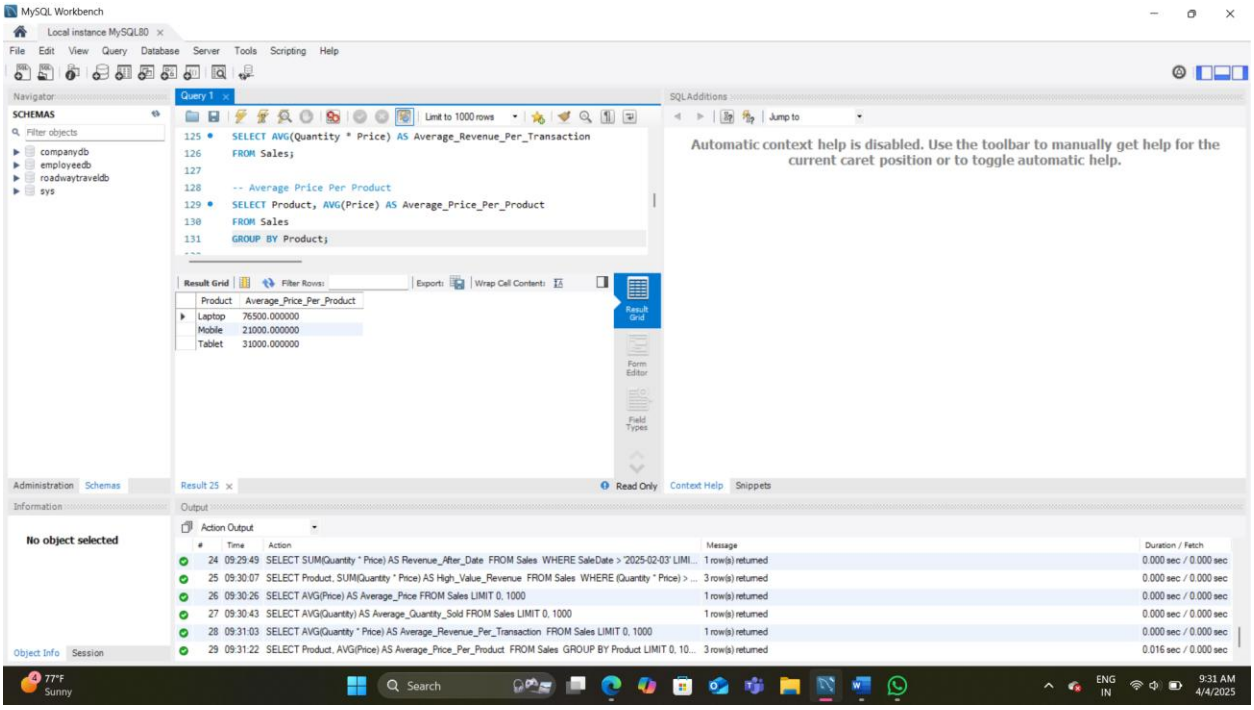


-- Average Price Per Product

SELECT Product, AVG(Price) AS Average_Price_Per_Product

FROM Sales

GROUP BY Product;

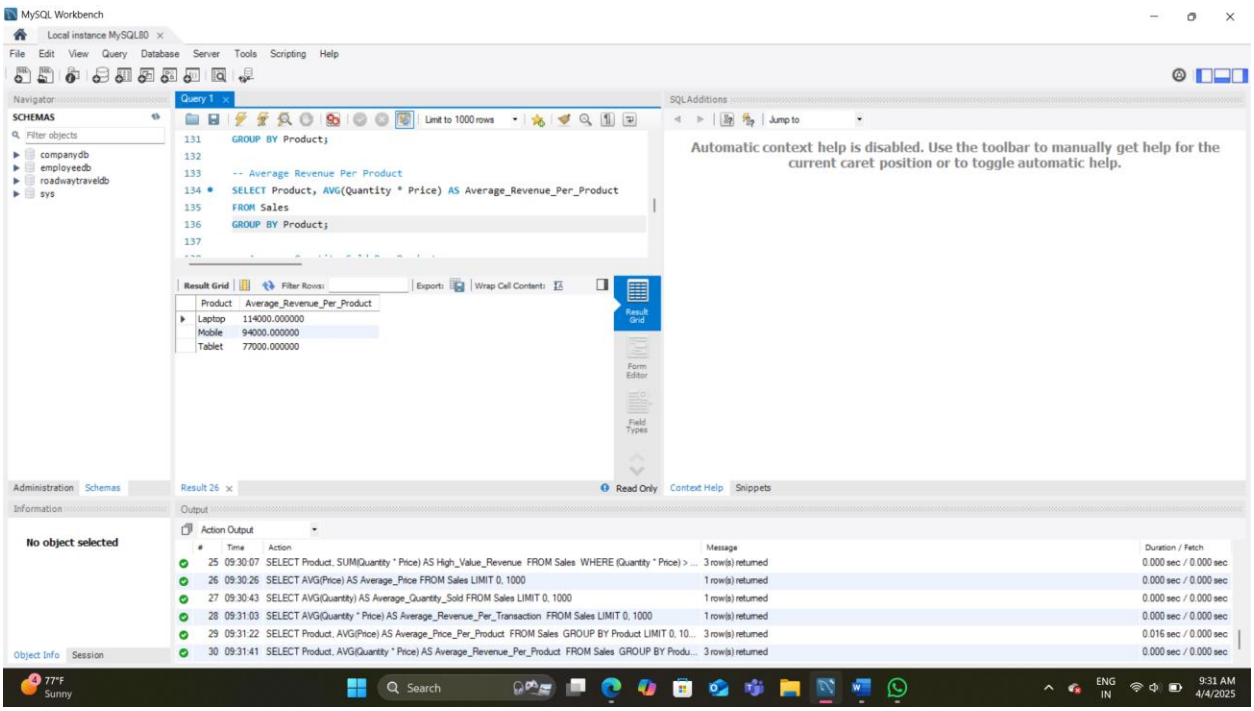


-- Average Revenue Per Product

SELECT Product, AVG(Quantity * Price) AS Average_Revenue_Per_Product

FROM Sales

GROUP BY Product;

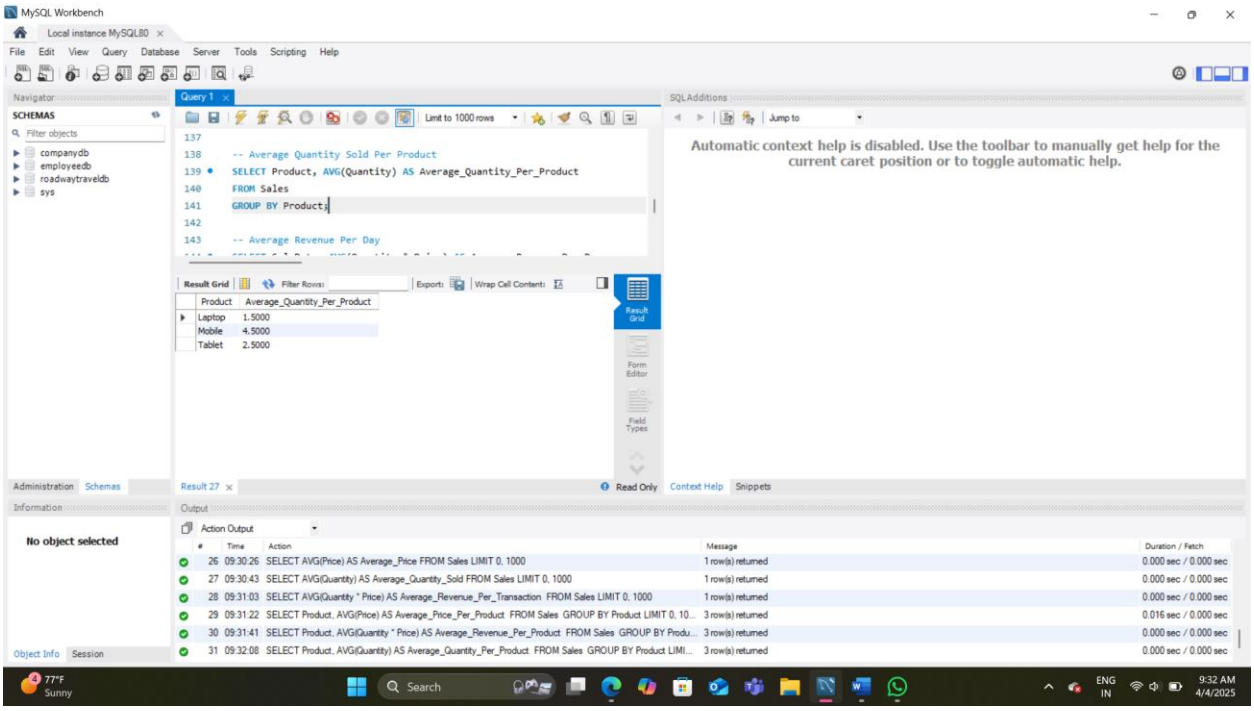


-- Average Quantity Sold Per Product

SELECT Product, AVG(Quantity) AS Average_Quantity_Per_Product

FROM Sales

GROUP BY Product;

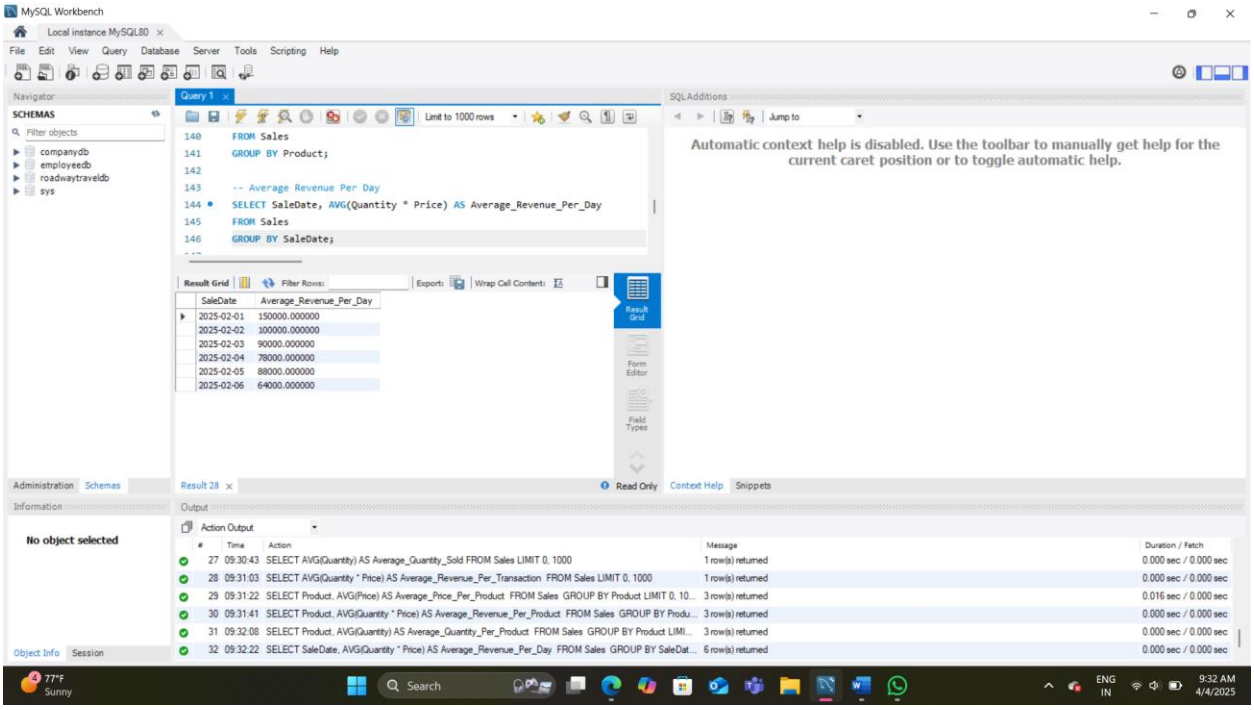


-- Average Revenue Per Day

SELECT SaleDate, AVG(Quantity * Price) AS Average_Revenue_Per_Day

FROM Sales

GROUP BY SaleDate;



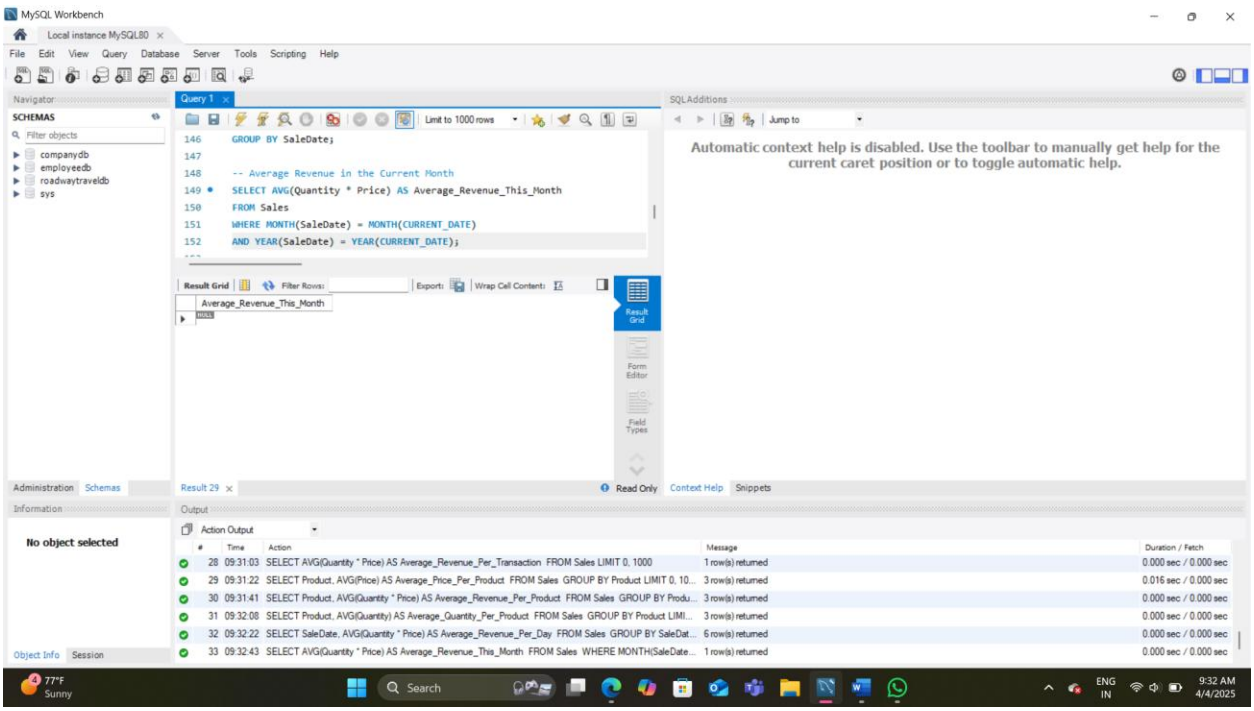
-- Average Revenue in the Current Month

SELECT AVG(Quantity * Price) AS Average_Revenue_This_Month

FROM Sales

WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT_DATE);

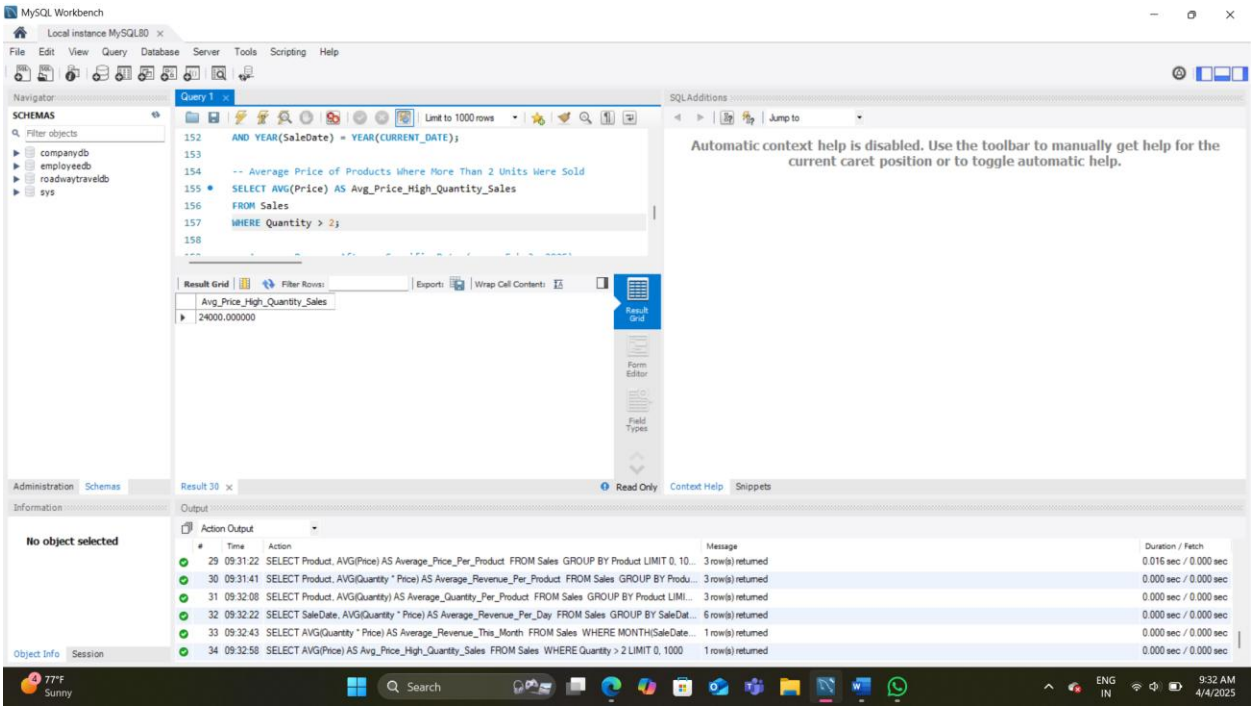


-- Average Price of Products Where More Than 2 Units Were Sold

SELECT AVG(Price) AS Avg_Price_High_Quantity_Sales

FROM Sales

WHERE Quantity > 2;

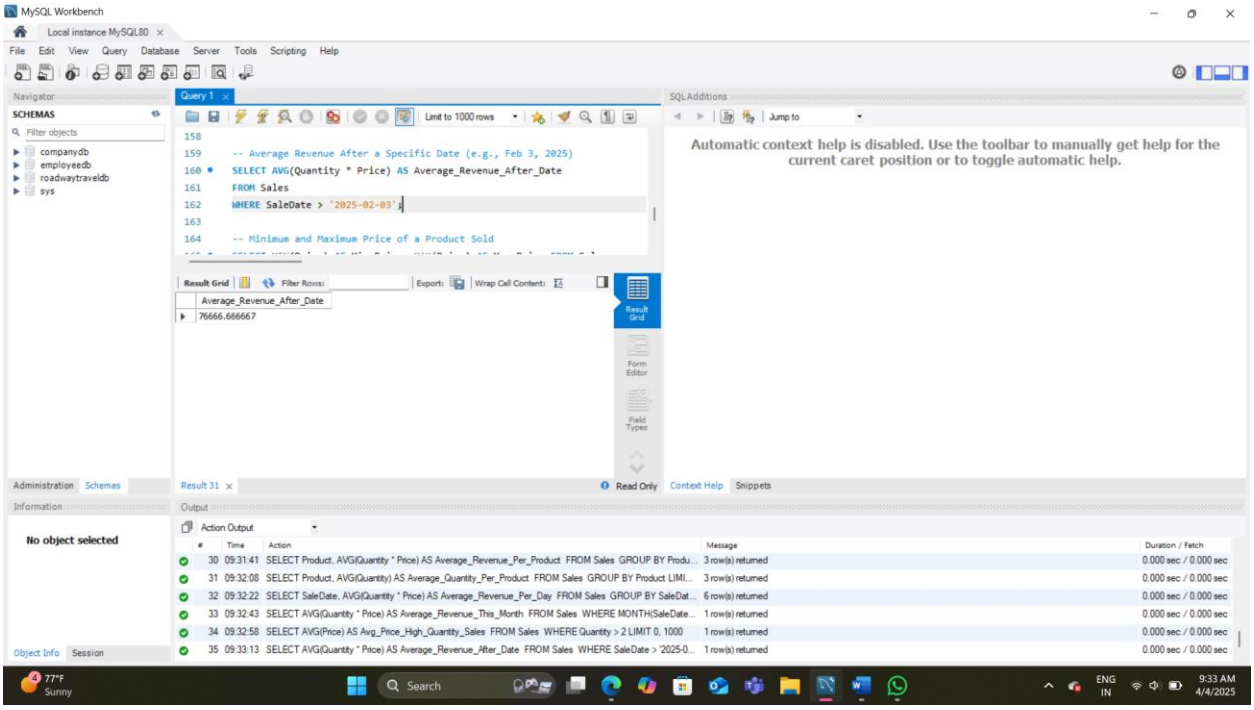


-- Average Revenue After a Specific Date (e.g., Feb 3, 2025)

SELECT AVG(Quantity * Price) AS Average_Revenue_After_Date

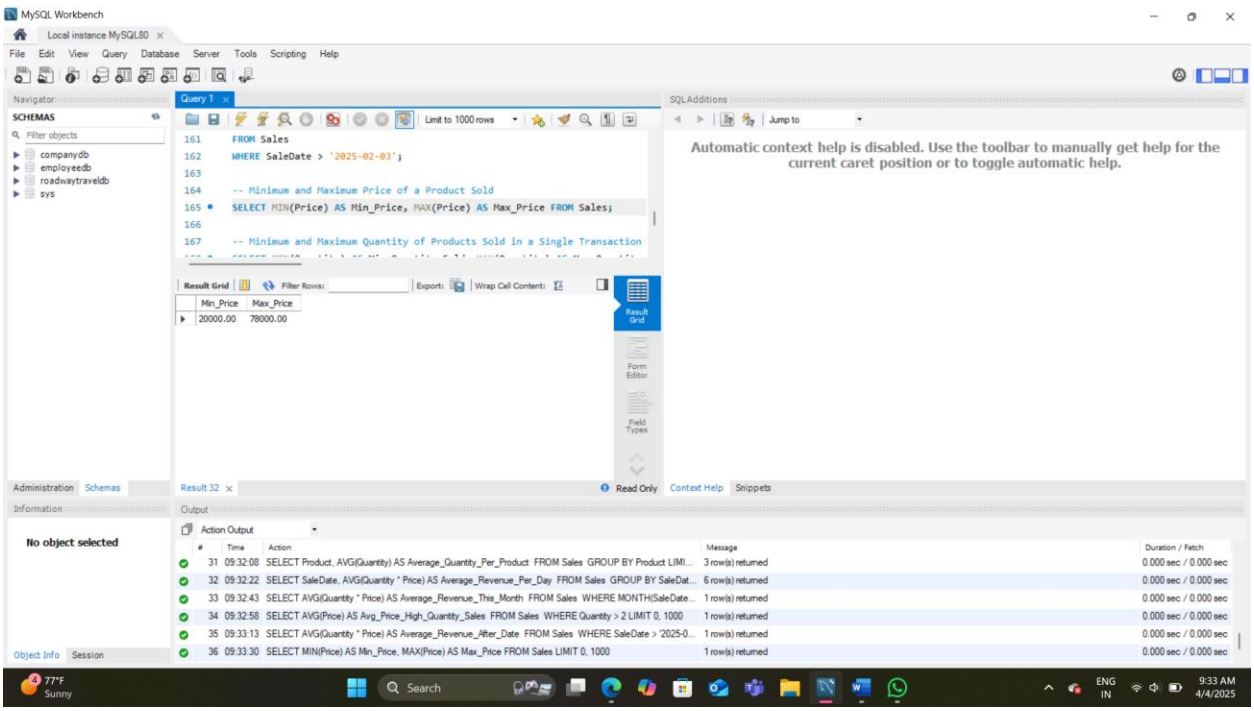
FROM Sales

WHERE SaleDate > '2025-02-03';



-- Minimum and Maximum Price of a Product Sold

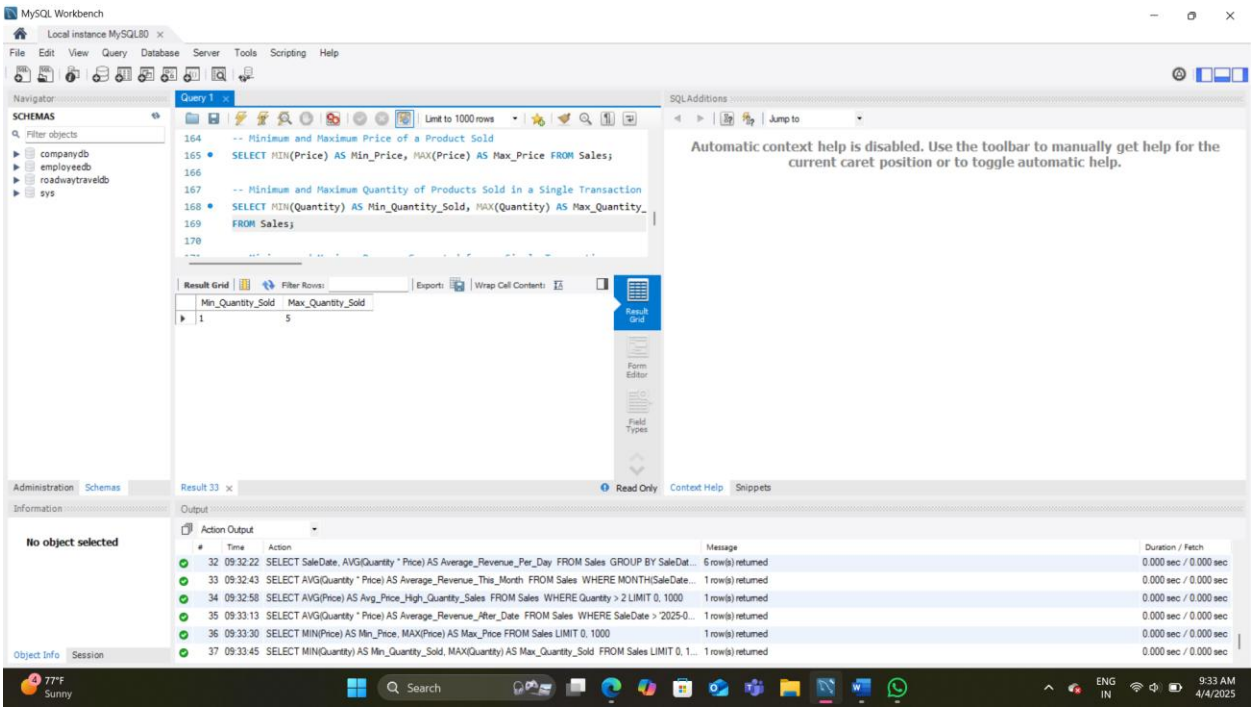
SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;



-- Minimum and Maximum Quantity of Products Sold in a Single Transaction

SELECT MIN(Quantity) AS Min_Quantity_Sold, MAX(Quantity) AS Max_Quantity_Sold

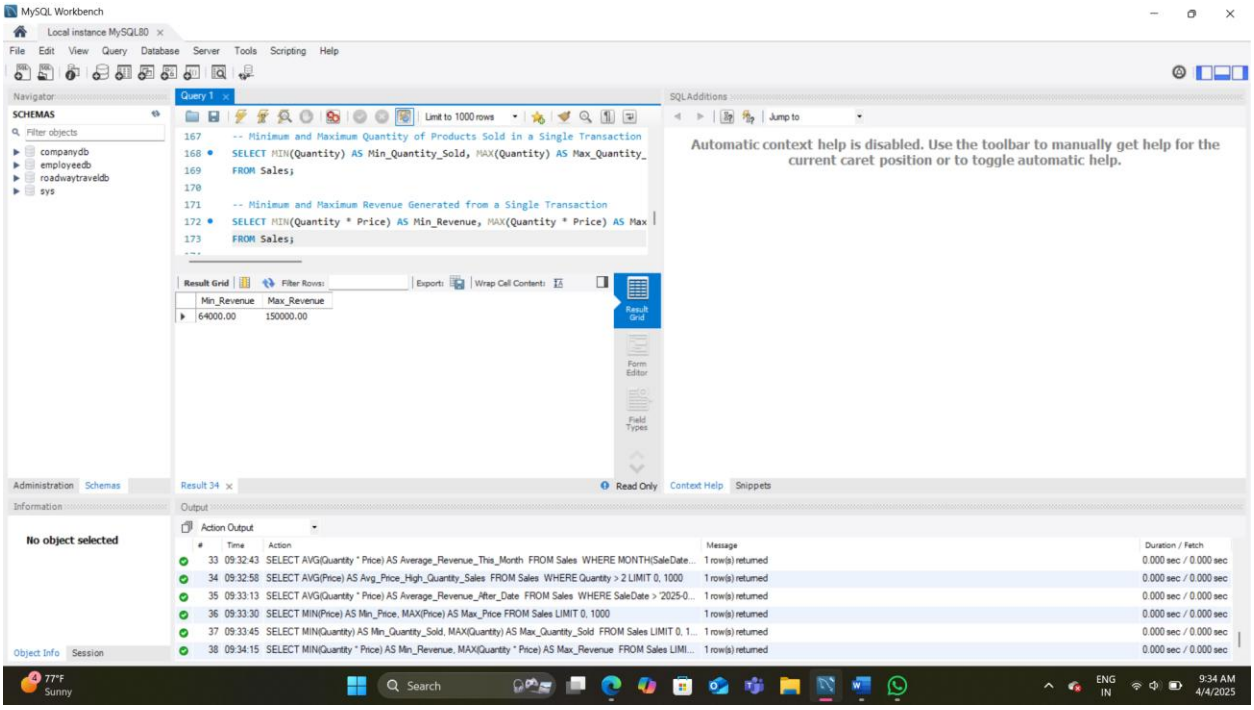
FROM Sales;



-- Minimum and Maximum Revenue Generated from a Single Transaction

SELECT MIN(Quantity * Price) AS Min_Revenue, MAX(Quantity * Price) AS Max_Revenue

FROM Sales;

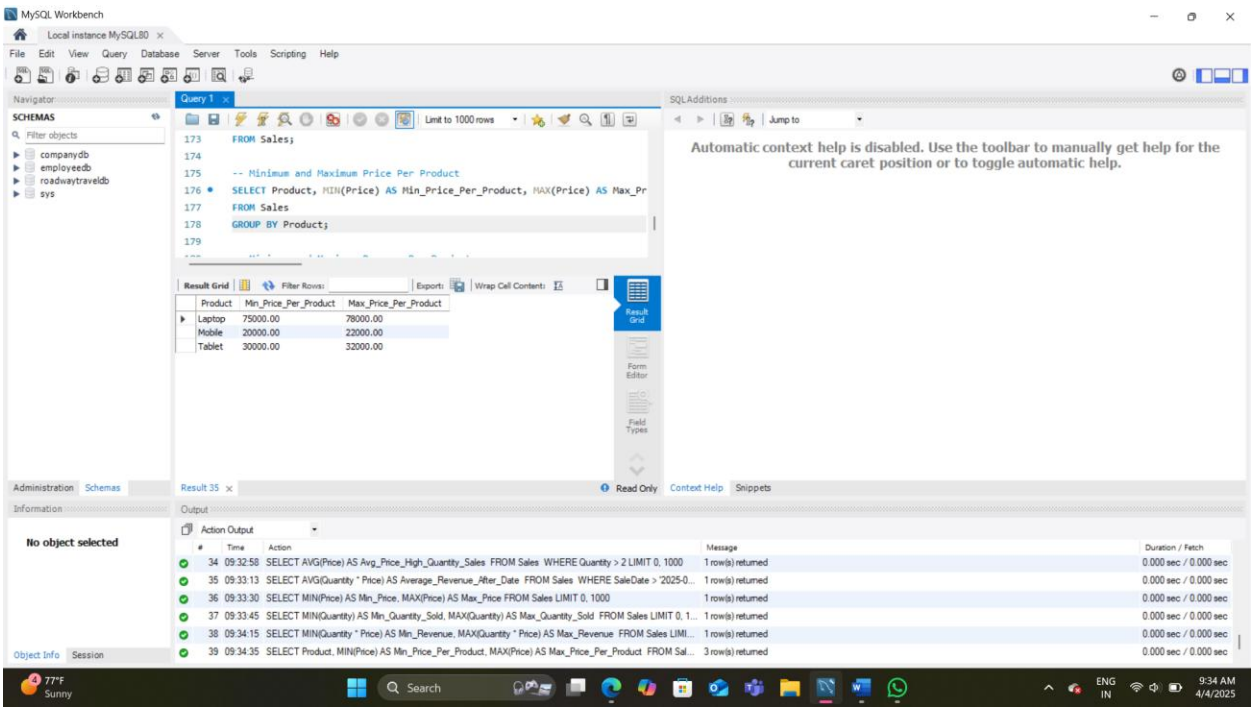


-- Minimum and Maximum Price Per Product

SELECT Product, MIN(Price) AS Min_Price_Per_Product, MAX(Price) AS Max_Price_Per_Product

FROM Sales

GROUP BY Product;

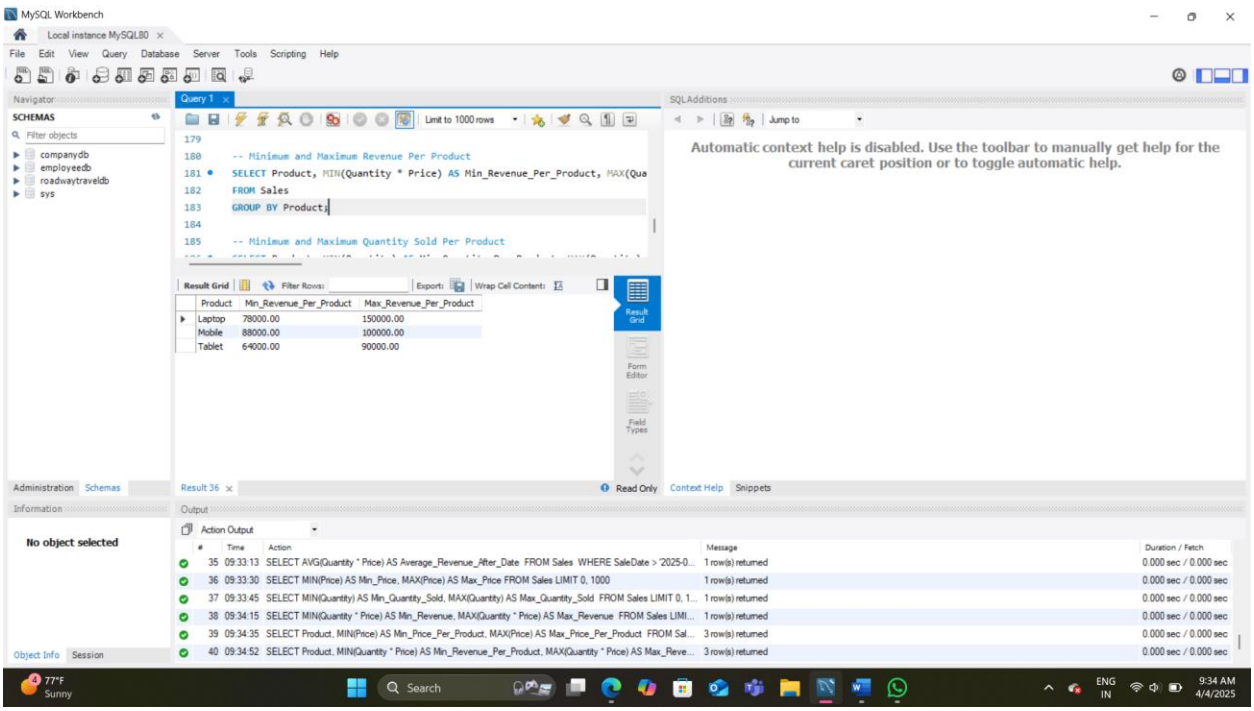


-- Minimum and Maximum Revenue Per Product

SELECT Product, MIN(Quantity * Price) AS Min_Revenue_Per_Product, MAX(Quantity * Price) AS Max_Revenue_Per_Product

FROM Sales

GROUP BY Product;

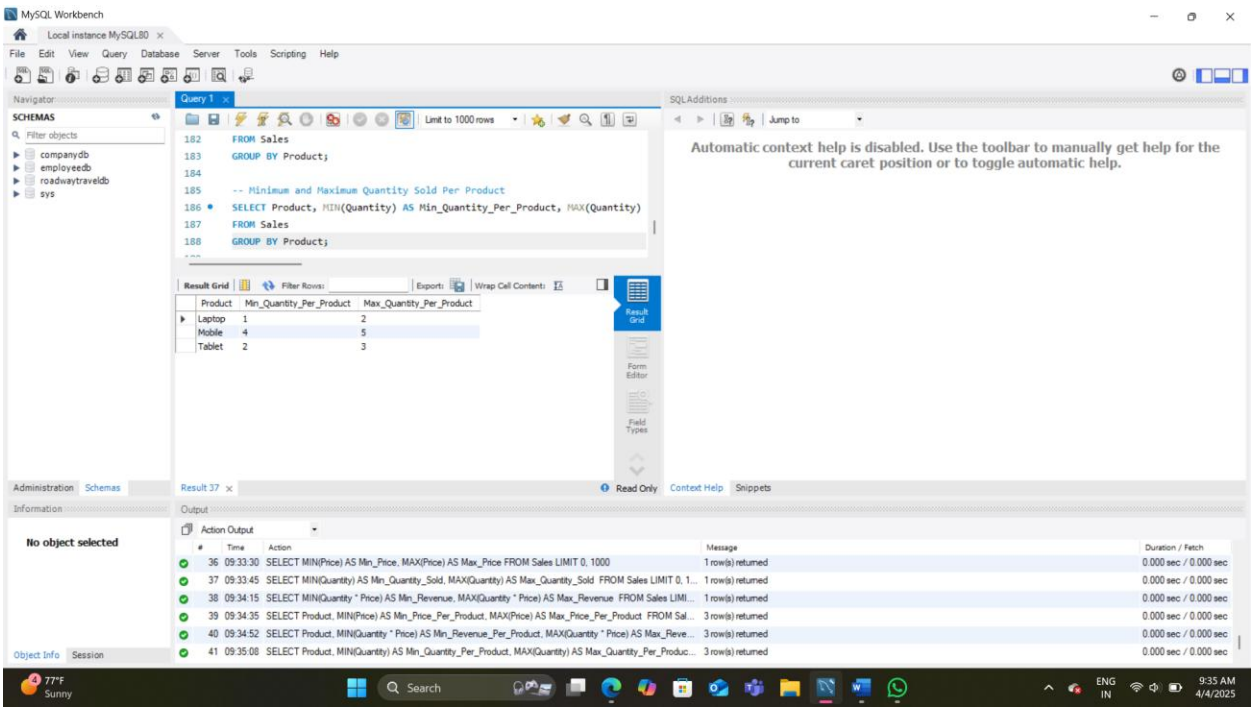


-- Minimum and Maximum Quantity Sold Per Product

SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity) AS Max_Quantity_Per_Product

FROM Sales

GROUP BY Product;

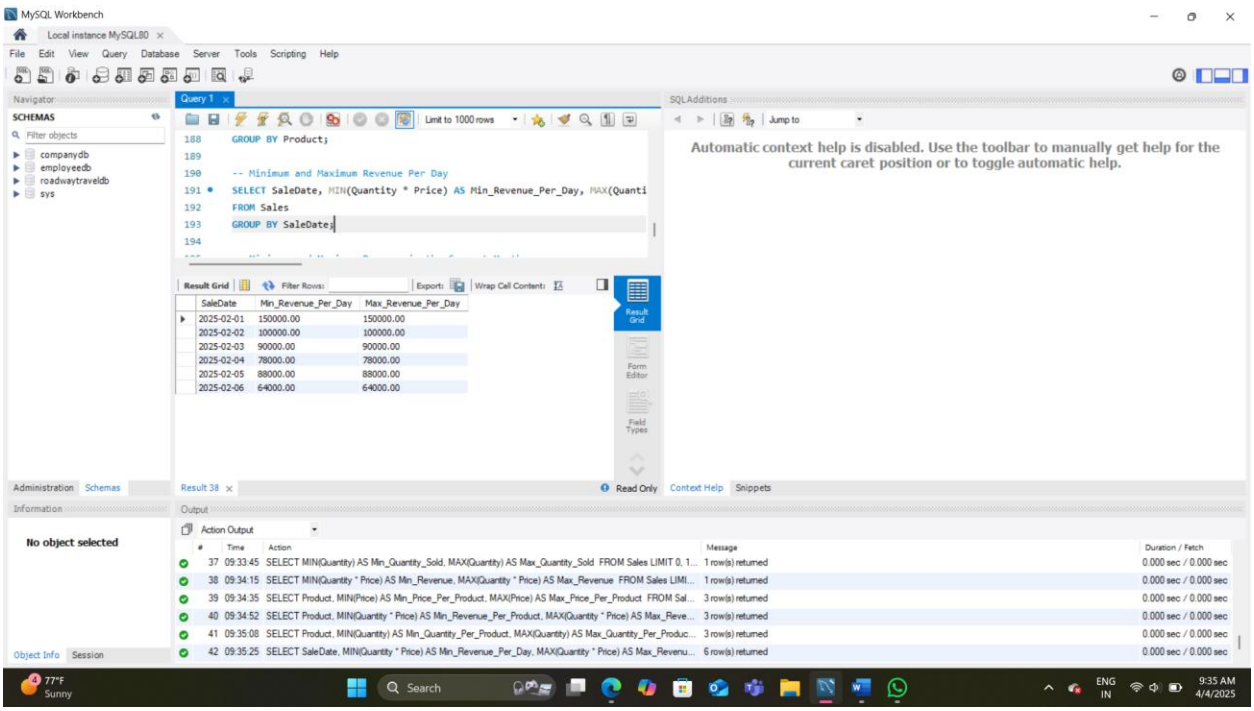


-- Minimum and Maximum Revenue Per Day

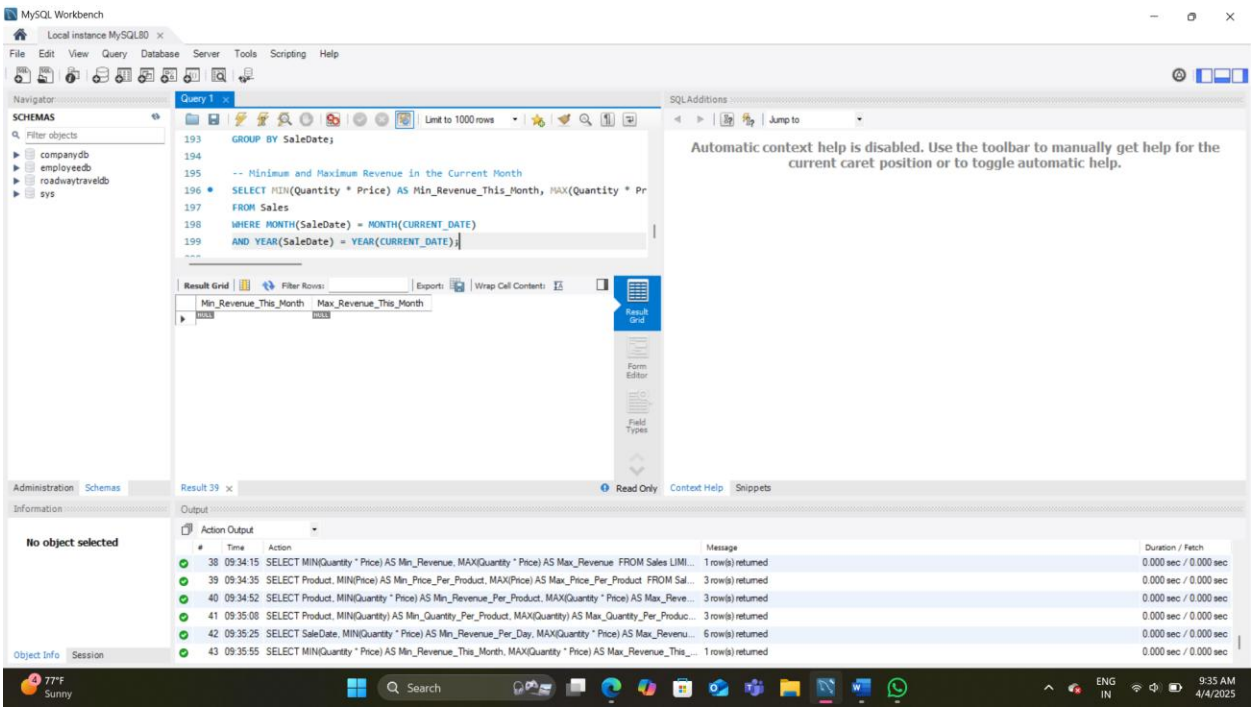
SELECT SaleDate, MIN(Quantity * Price) AS Min_Revenue_Per_Day, MAX(Quantity * Price) AS Max_Revenue_Per_Day

FROM Sales

GROUP BY SaleDate;




```
-- Minimum and Maximum Revenue in the Current Month

SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity * Price) AS Max_Revenue_This_Month

FROM Sales

WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
```

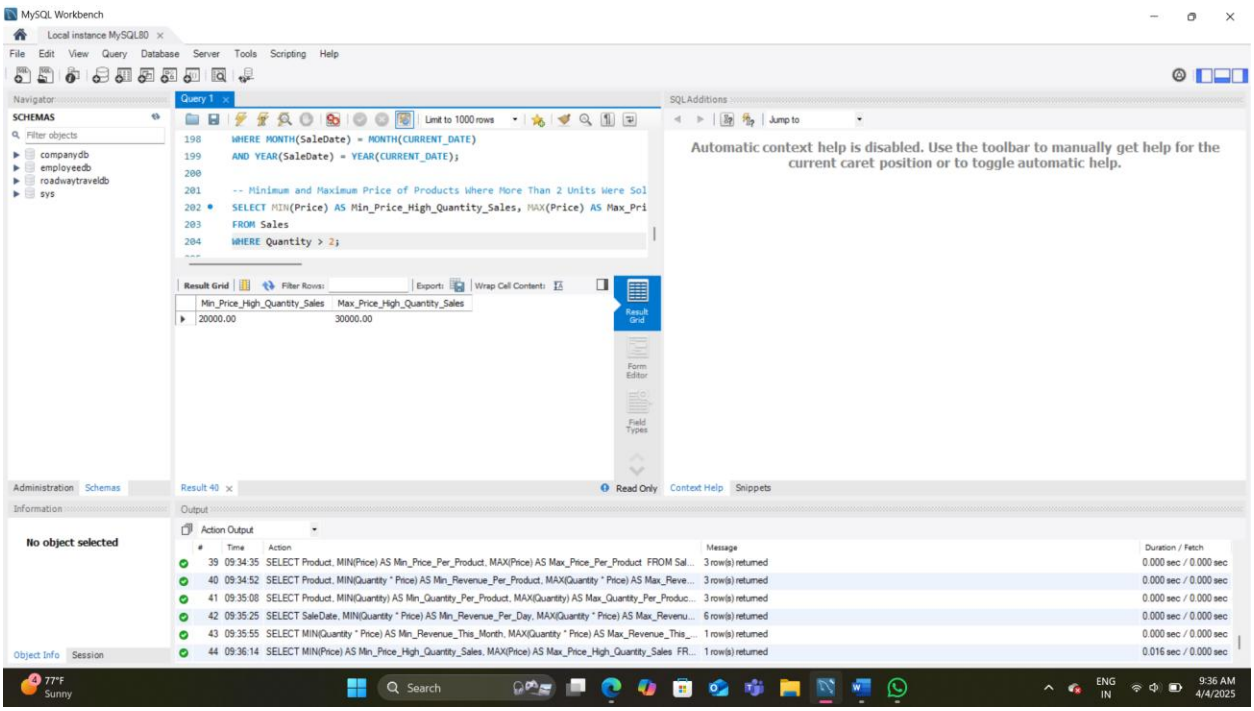


-- Minimum and Maximum Price of Products Where More Than 2 Units Were Sold

SELECT MIN(Price) AS Min_Price_High_Quantity_Sales, MAX(Price) AS Max_Price_High_Quantity_Sales

FROM Sales

WHERE Quantity > 2;

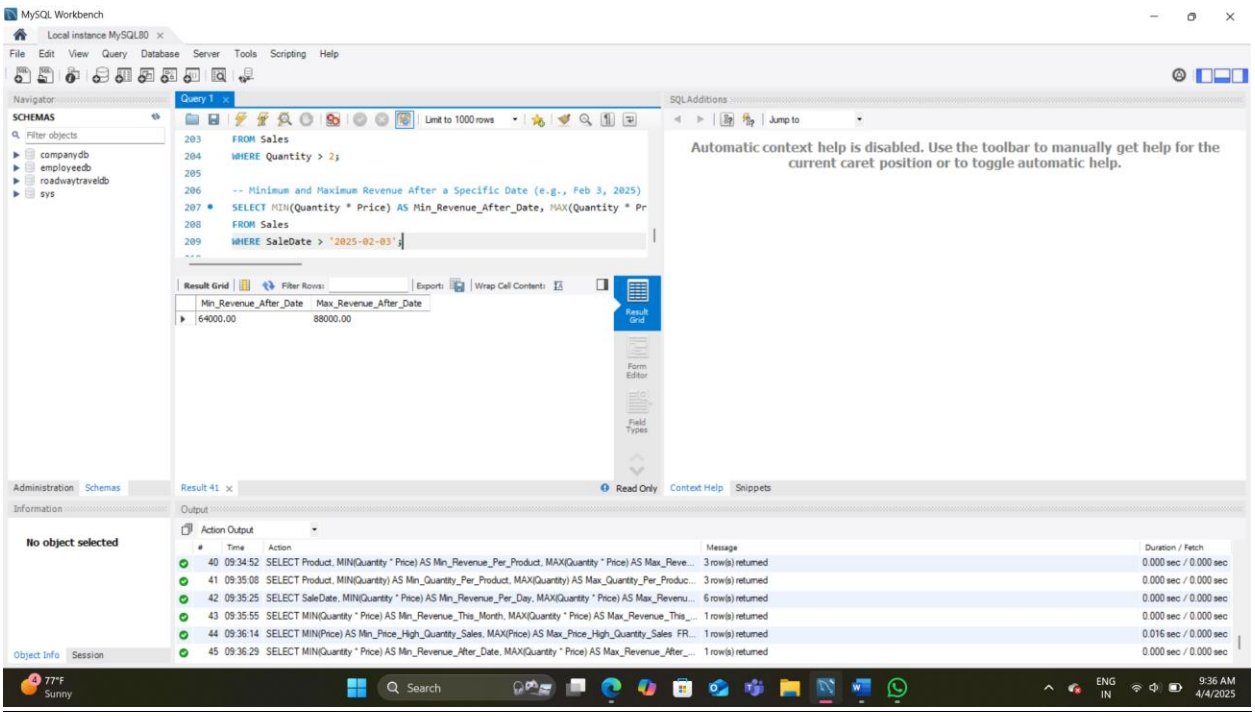


-- Minimum and Maximum Revenue After a Specific Date (e.g., Feb 3, 2025)

SELECT MIN(Quantity * Price) AS Min_Revenue_After_Date, MAX(Quantity * Price) AS Max_Revenue_After_Date

FROM Sales

WHERE SaleDate > '2025-02-03';



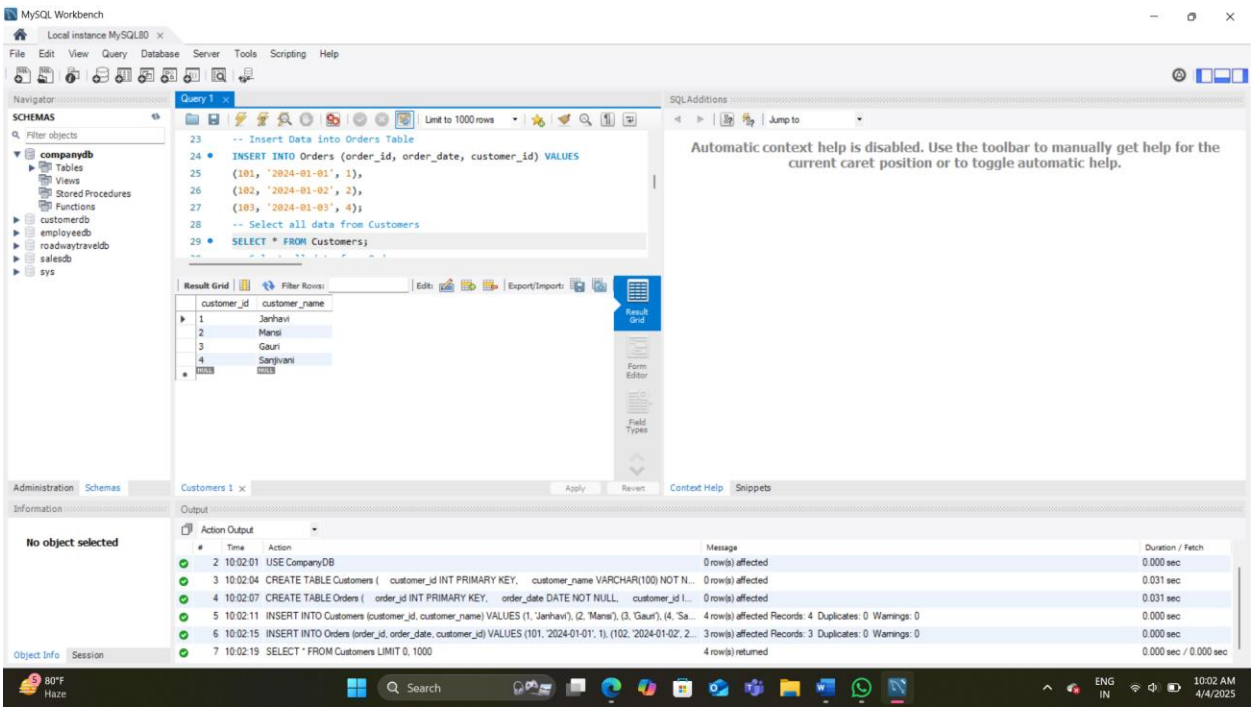
Practical 8:

Aim: Given Customers and Orders tables, write SQL queries to perform INNER JOIN, LEFT JOIN, and RIGHT JOIN to retrieve combined data for customer orders.

Code:

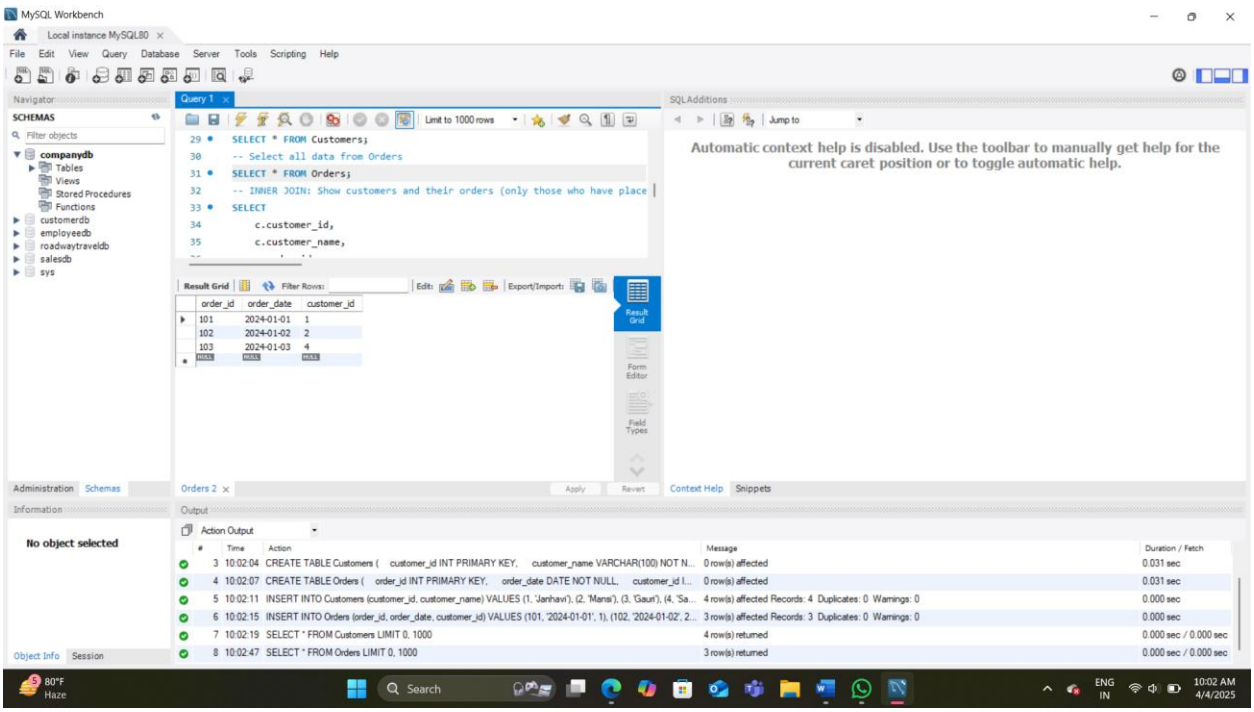
```
-- Create the Database

CREATE DATABASE CompanyDB;

-- Use the Database

USE CompanyDB;

-- Create the Customers Table

CREATE TABLE Customers (

   customer_id INT PRIMARY KEY,

   customer_name VARCHAR(100) NOT NULL
```

```
);
-- Create the Orders Table
CREATE TABLE Orders (
  order_id INT PRIMARY KEY,
  order_date DATE NOT NULL,
  customer_id INT,
  FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
-- Insert Data into Customers Table
INSERT INTO Customers (customer_id, customer_name) VALUES
(1, 'Janhavi'),
(2, 'Mansi'),
(3, 'Gauri'),
(4, 'Sanjivani');
-- Insert Data into Orders Table
INSERT INTO Orders (order_id, order_date, customer_id) VALUES
(101, '2024-01-01', 1),
(102, '2024-01-02', 2),
(103, '2024-01-03', 4);
-- Select all data from Customers
SELECT * FROM Customers;
```



-- Select all data from Orders

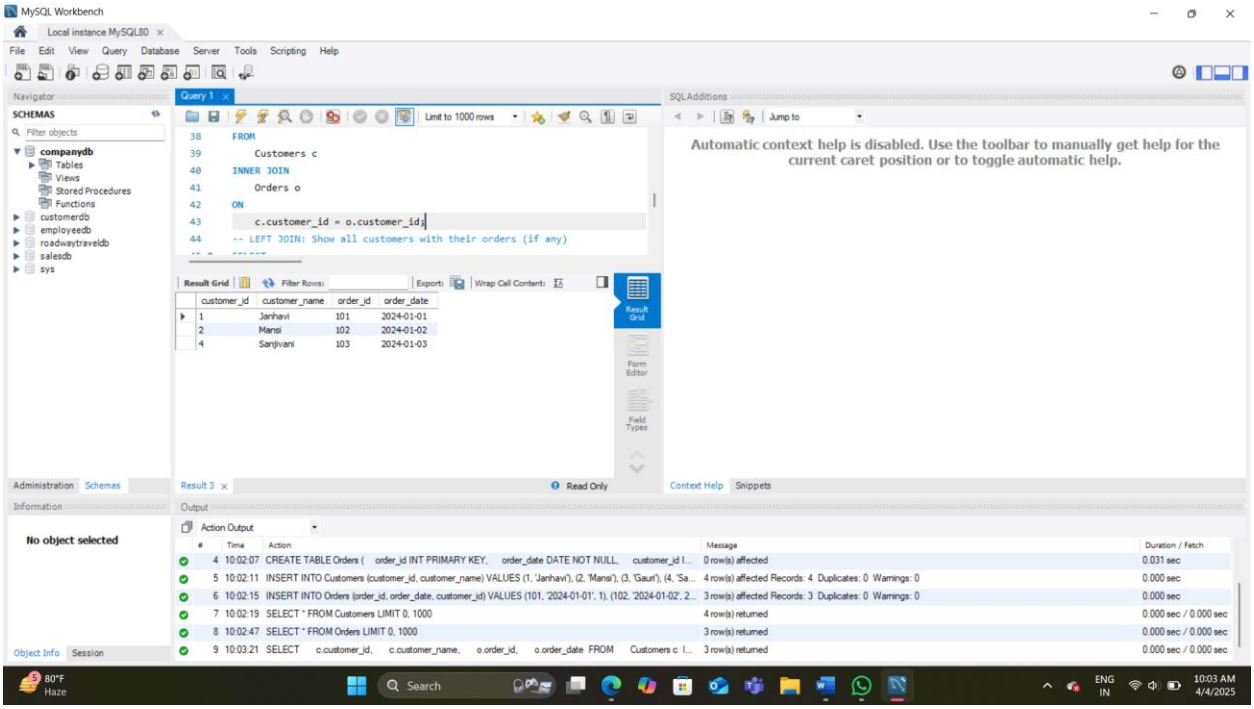
SELECT * FROM Orders;



-- INNER JOIN: Show customers and their orders (only those who have placed orders)

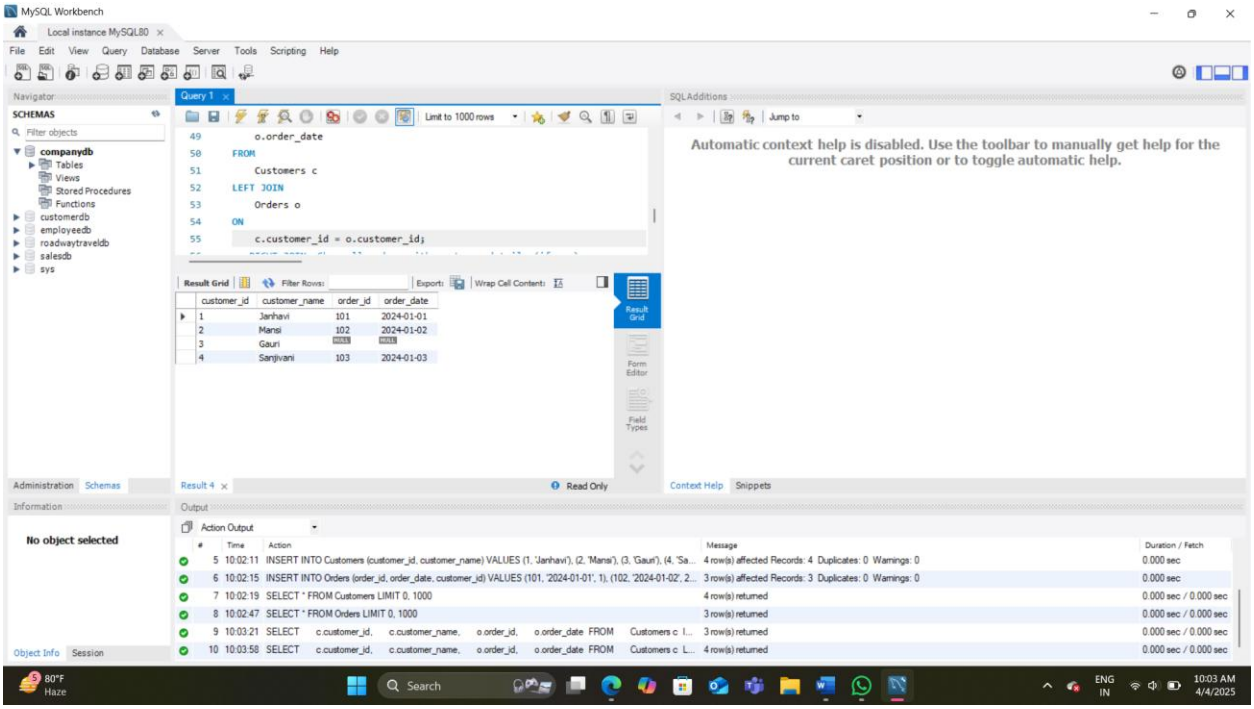
SELECT

```
    c.customer_id,
    c.customer_name,
    o.order_id,
    o.order_date
FROM
    Customers c
INNER JOIN
    Orders o
ON
    c.customer_id = o.customer_id;
```




```
-- LEFT JOIN: Show all customers with their orders (if any)
SELECT
    c.customer_id,
    c.customer_name,
    o.order_id,
```

```
    o.order_date
FROM
    Customers c
LEFT JOIN
    Orders o
ON
    c.customer_id = o.customer_id;
```




```
-- RIGHT JOIN: Show all orders with customer details (if any)
SELECT
    c.customer_id,
    c.customer_name,
    o.order_id,
    o.order_date
FROM
    Customers c
```

RIGHT JOIN

Orders o

ON

c.customer_id = o.customer_id;