Assignment no.1

**Aim:** Install and set up MySQL, Create a database and a table to store employee details. Perform basic operations like INSERT & DELETE

**Code:** CREATE TABLE employees (

Emp\_id INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(50),

Age INT,

Department VARCHAR(50),

Salary DECIMAL(10,2)

);

INSERT INTO employees (Name, Age, Department, Salary) VALUES

('shubham paithankar', 18, 'CY', 80000.00),

('rohit sharma', 42, 'HR', 4500.00),

('surya yadav', 35, 'Finance', 60000.00),

('Hardik pandya', 18, 'CSE', 70000.00),

('Shantanu sonawne', 42, 'AIML', 90000.00),

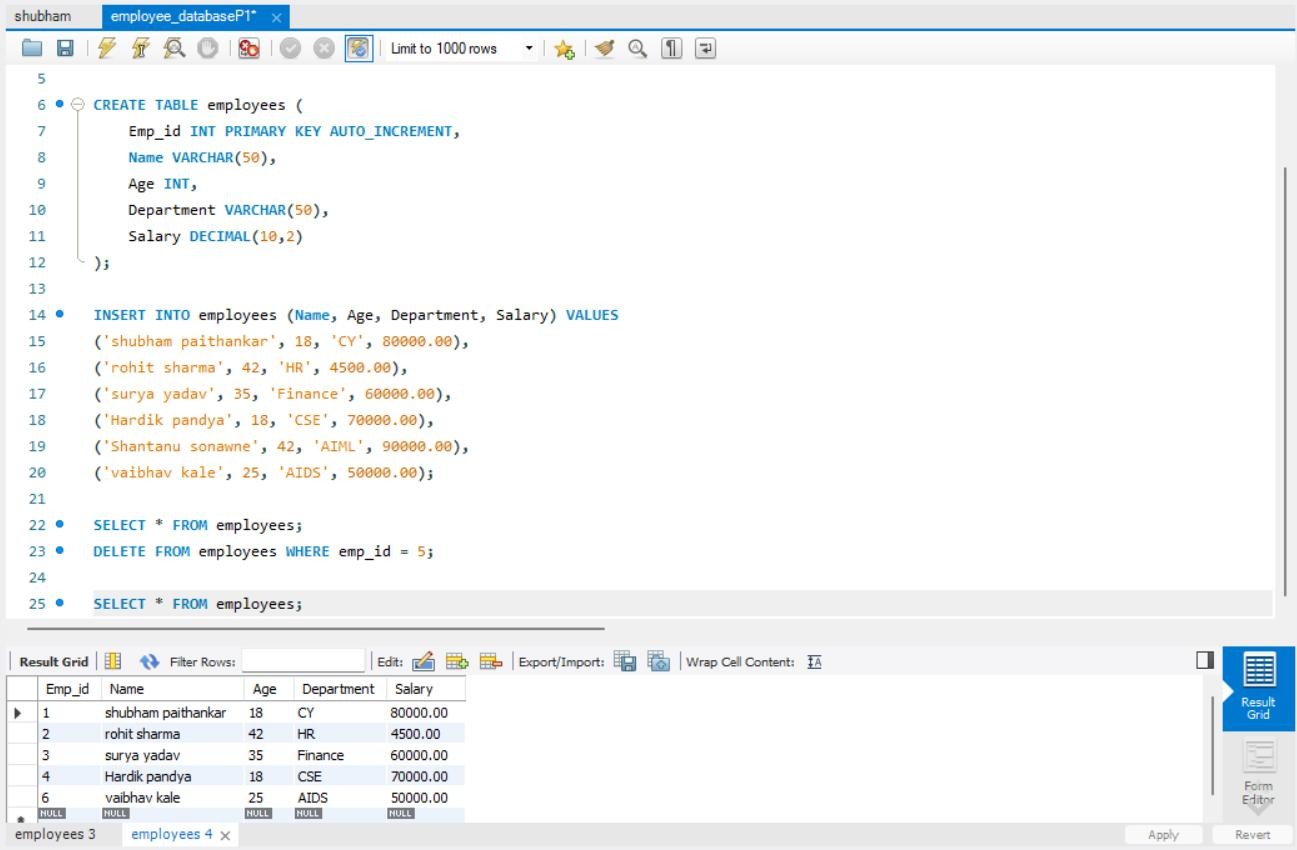
('vaibhav kale', 25, 'AIDS', 50000.00);

SELECT \* FROM employees;

DELETE FROM employees WHERE emp\_id = 5;

SELECT \* FROM employees;

**Screenshot(Output):**



# Assignment no.2

**Aim:** Create a table for storing student information. Insert sample data and perform basic operations: INSERT, UPDATE, DELETE, and SELECT

**Code:** CREATE TABLE student (

Student\_id INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(50),

Age INT,

Course VARCHAR(50),

Marks INT

);

INSERT INTO students (name, age, course, marks) VALUES

('Shubham Paithankar', 18, 'Cyber Security', 88),

('Suraj Chavhan', 21, 'Mechanical Engineering', 78),

('Prasad Kale', 23, 'Electrical Engineering', 90),

('Sanket Gaikwad', 17, 'Mechanical Engineering', 68),

('Vaibhav Thakur', 22, 'Mechanical Engineering', 78),

('Sushant Nagare', 22, 'Computer science an Engineering', 78);

SELECT \* FROM students;

UPDATE students

SET Marks = 80

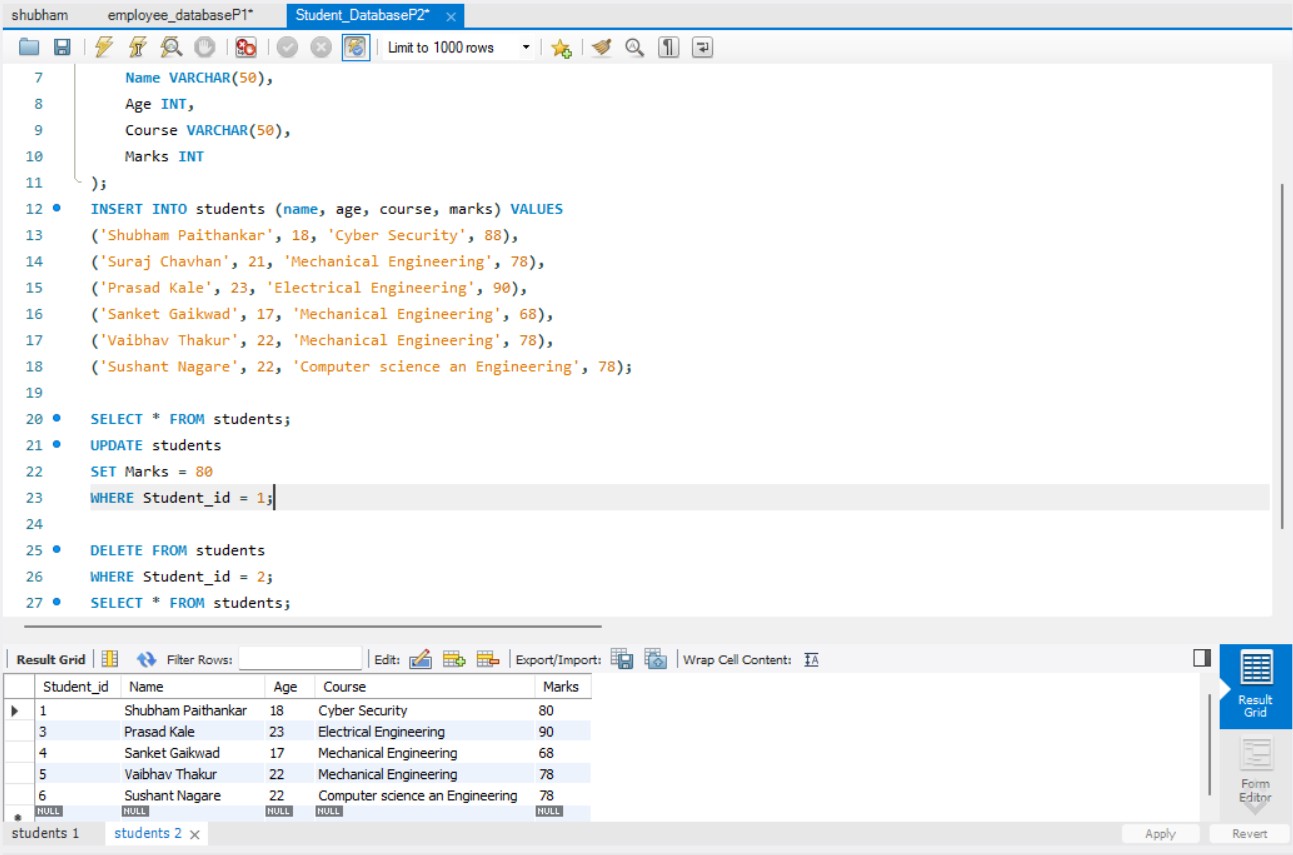
WHERE Student\_id = 1;

DELETE FROM students

WHERE Student\_id = 2;

SELECT \* FROM students;

**Screenshot(Output):**



# Assignment no.3

# 

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

**Code:**

CREATE TABLE employees (

EmployeeID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL,

Salary DECIMAL(20,2),

JoiningDate DATE,

ActiveStatus BOOLEAN

);

INSERT INTO employees (Name, Salary, JoiningDate, ActiveStatus) VALUES

('Shubham Paithankar', 95000.00, '2022-06-23', TRUE),

('Rohit Kolhe', 70000.50, '2021-07-25', TRUE),

('Aarav Wable', 50000.75, '2023-01-02', True),

('Gita rajwade', 70000.25, '2019-11-05', TRUE),

('Tejas Gade', 70000.25, '2020-11-05', TRUE);

SELECT \* FROM employees;

UPDATE employees

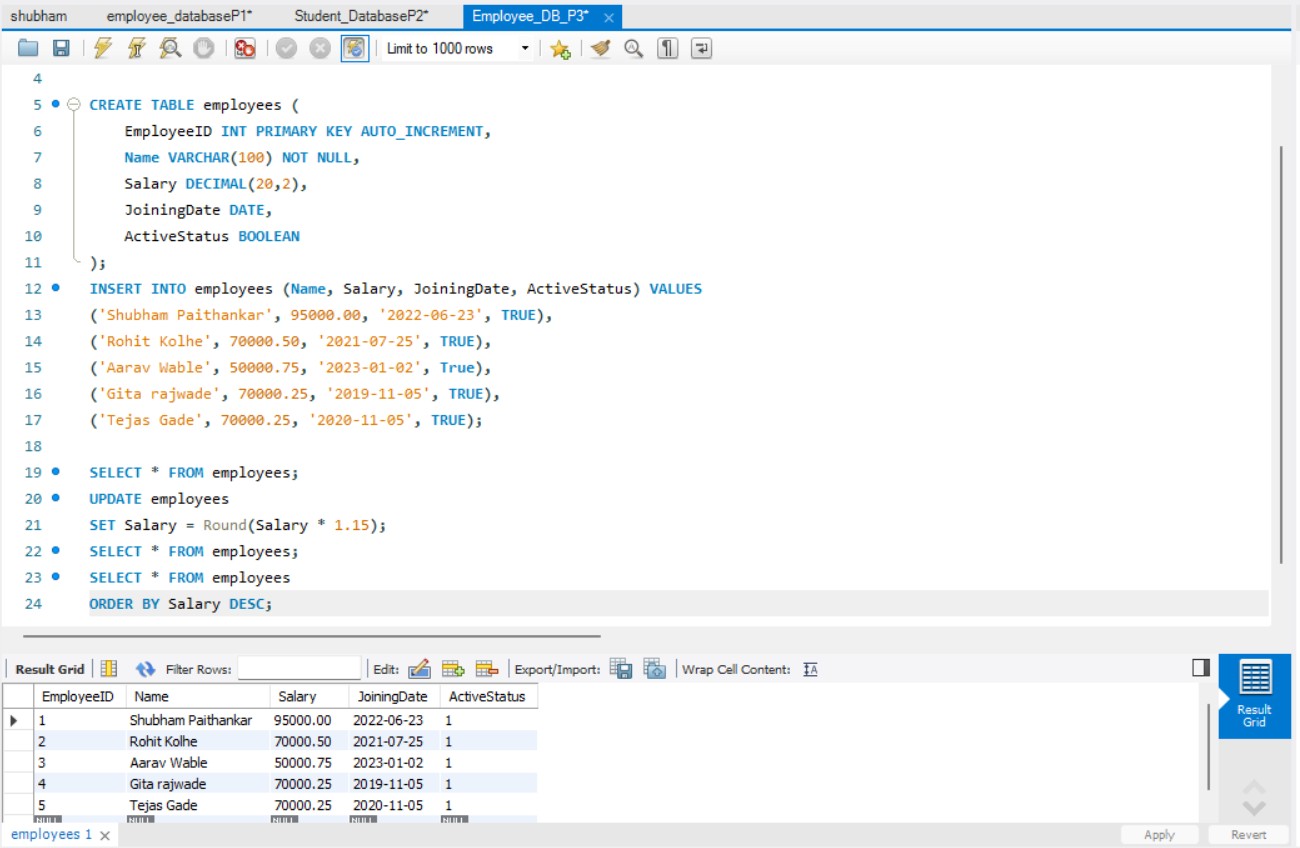
SET Salary = Round(Salary \* 1.15);

SELECT \* FROM employees;

SELECT \* FROM employees

ORDER BY Salary DESC;

**Screenshot(Output):**



# Assignment no.4

**Aim:** Create a table to store employee information with constraints like Primary Key, Foreign Key, and Unique. Insert valid and invalid data to test the constraints.

**Code**:

CREATE TABLE Department (

DeptID INT PRIMARY KEY,

DeptName VARCHAR(50) UNIQUE

);

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 INTO Department (DeptID, DeptName) VALUES (1, 'HR');

INSERT INTO Department (DeptID, DeptName) VALUES (2, 'IT');

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (101, 'Shubham','shubham.paithankar24@sanjivani.edu.in', 55000.00, 1);

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES

(102,'Shubham','shubham.sonawne24@sanjivani.edu.in',50000.00,2);

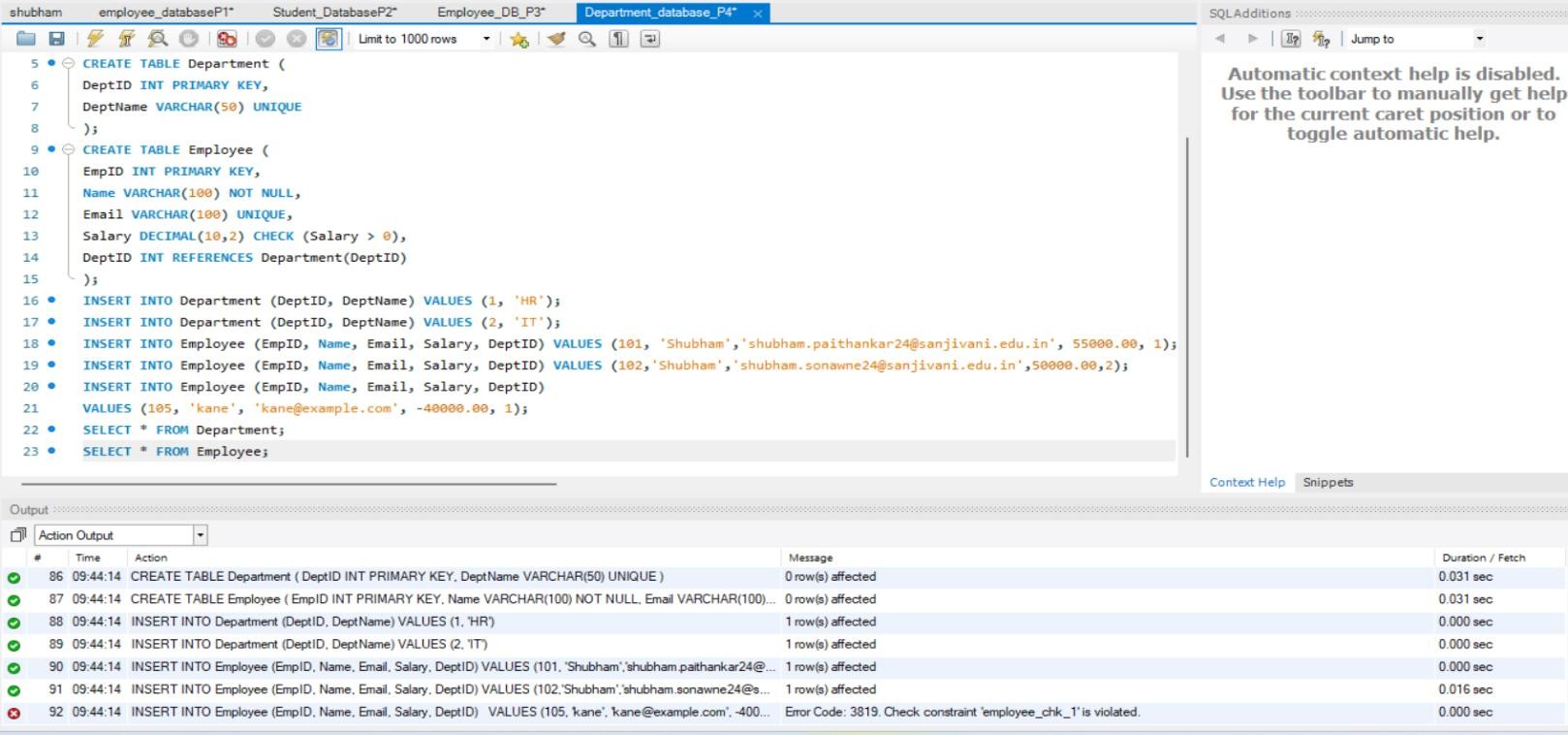
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)

VALUES (105, 'kane', 'kane@example.com', -40000.00, 1);

SELECT \* FROM Department;

SELECT \* FROM Employee;

**Output:(Error Screenshot):**



# Assignment no.5

**Aim:** Create a table for Customer details with various integrity constraints like NOT NULL, CHECK, and DEFAULT. Insert valid and invalid data to test these constraints and ensure data integrity.

**Code:** drop database Customer\_db; create database Customer\_db; use Customer\_db; 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

);

**--Inserting A Valid Data Into Customer table**

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age, IsActive)

VALUES (1, 'shubham','Paithankar','shubham.paithankar24@sanjivani.edu.in', '7020268056', 18, TRUE);

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)

VALUES (2, 'Rohit', 'Kale', 'rohit@example.com', '9956873490', 27);

SELECT \* FROM Customer;

**--Inserting invalid Data to test constrains**

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)

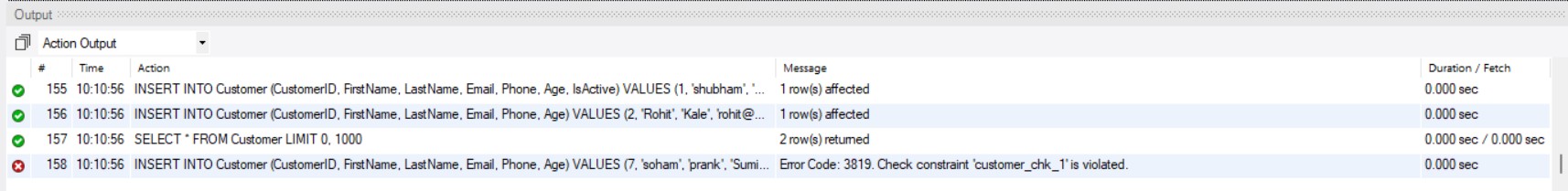
VALUES (7, 'soham', 'prank', 'Sumit@pass.com', '0071825067', -5);

SELECT \* FROM Customer;

**Screenshot(Output):**



**Error(Screenshot):**



# Assignment no.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 TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Age INT,

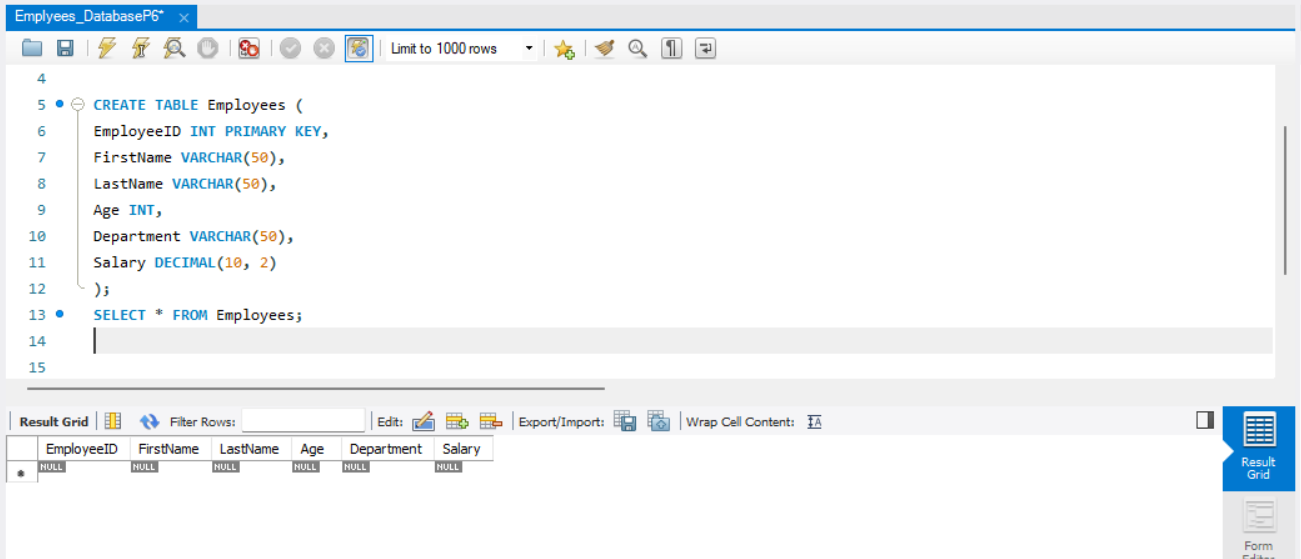
Department VARCHAR(50),

Salary DECIMAL(10, 2)

);

SELECT \* FROM Employees;

**Output:(Screenshot):**



**Data Insertion (Using DML Command):**

**--Inserting Data into Customer Table**

INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department,

Salary)

VALUES (1, 'Shubham', 'Paithankar', 18, 'HR', 95000.00);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department,

Salary)

VALUES (2, 'Sameer', 'shaik', 28, 'IT', 65000.00);

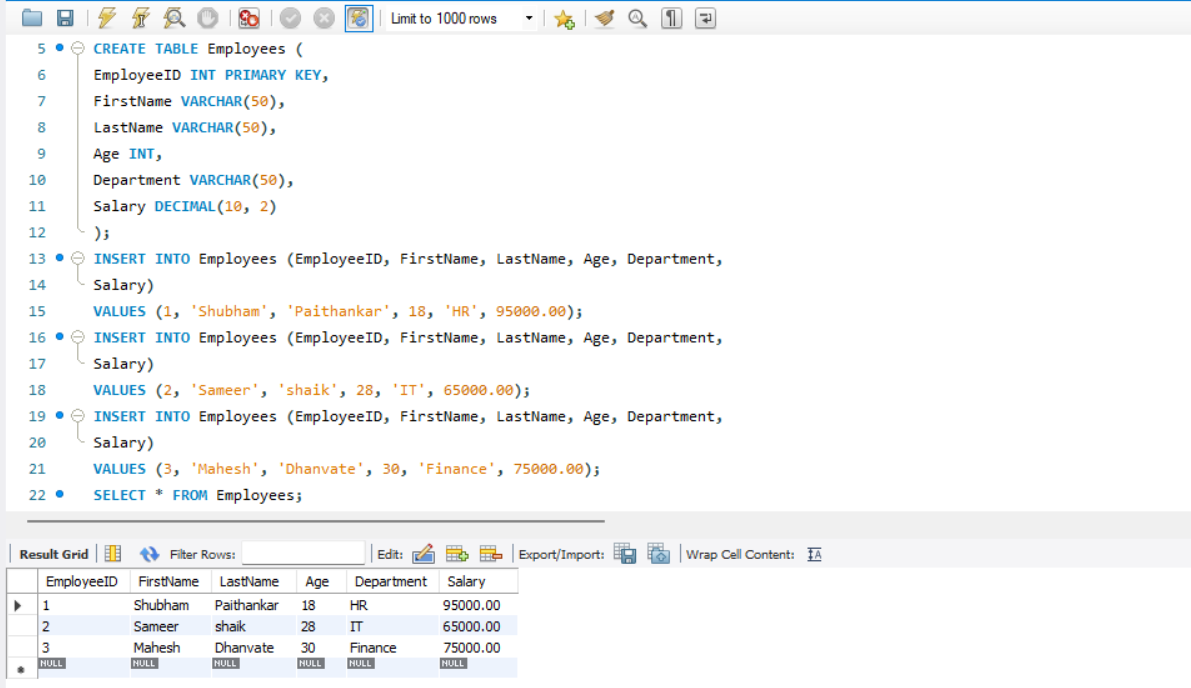
INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department,

Salary)

VALUES (3, 'Mahesh', 'Dhanvate', 30, 'Finance', 75000.00);

SELECT \* FROM Employees;

**Output:(Screenshot):**

****

**--Updating Data into Customer Table**

* Update a single column
* Update multiple columns for a specific row
* Update entire tuple
* Update with a condition
* Update with a subquery
* Update using a CASE statement

**Code:(Updation):**

**-- 1. Update a single column (e.g., update salary for EmployeeID 1)**

UPDATE Employees

SET Salary = 70000.00

WHERE EmployeeID = 1;

SELECT \* FROM Employees;

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

UPDATE Employees

SET FirstName = 'James', LastName = 'Will', Salary = 75000.00

WHERE EmployeeID = 2;

SELECT \* FROM Employees;

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

UPDATE Employees

SET FirstName = 'Michael', LastName = 'Bro', Age = 40, Department =

'Management', Salary = 80000.00

WHERE EmployeeID = 3;

SELECT \* FROM Employees;

**-- 4. Update with a condition (e.g., increase salary by 10% for EmployeeID=1)**

UPDATE Employees

SET Salary = Salary \* 1.10

WHERE EmployeeID = 1;

SELECT \* FROM Employees;

**-- 5. Update using a CASE statement (e.g., increase salary based on Employee Id)**

UPDATE Employees

SET Salary = CASE

WHEN EmployeeID = 1 THEN Salary \* 1.05

WHEN EmployeeID = 2 THEN Salary \* 1.08

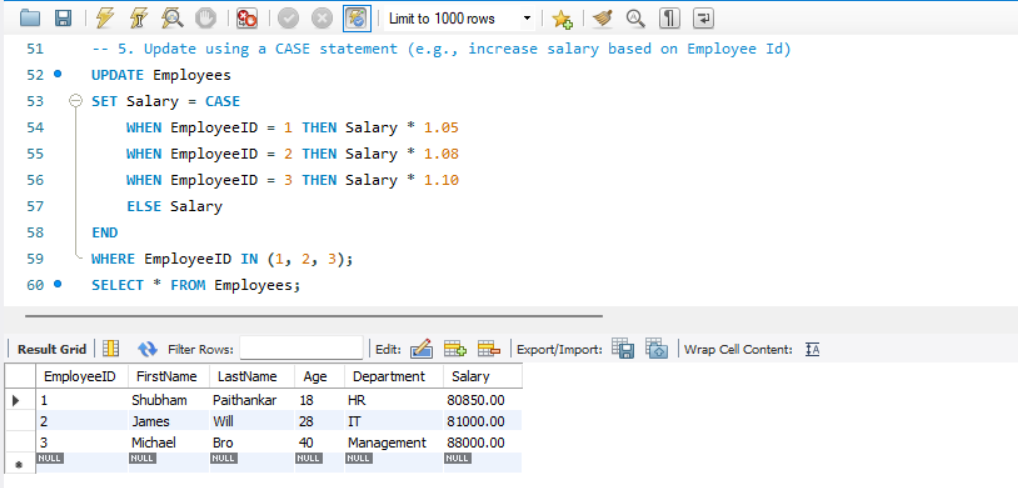
WHEN EmployeeID = 3 THEN Salary \* 1.10

ELSE Salary END

WHERE EmployeeID IN (1, 2, 3);

SELECT \* FROM Employees;

**Output:(Screenshot):**



**--Deleting Data into Customer Table (DML Command)**

DELETE FROM Employees

WHERE EmployeeID = 3;

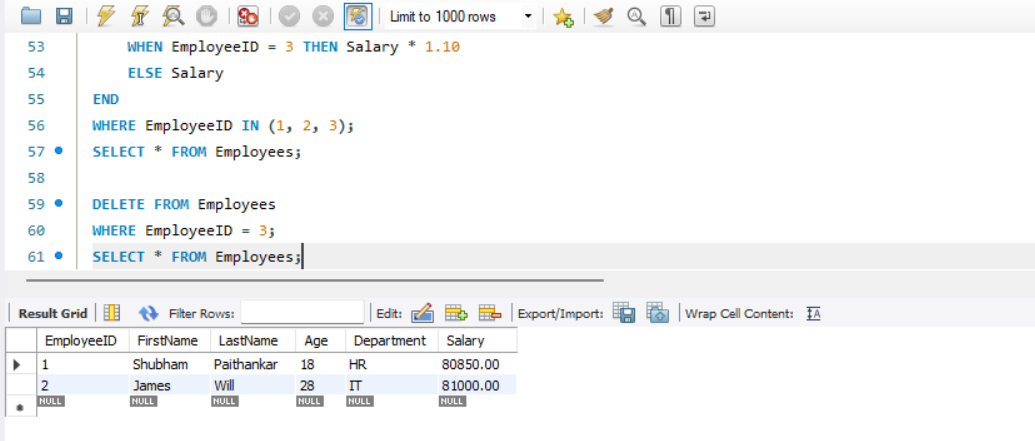
SELECT \* FROM Employees;

* **Select and Verify Data (SELECT Query)**

**-- To retrieve all data from the table**

SELECT \* FROM Employees;

**Output:(Screenshot):**

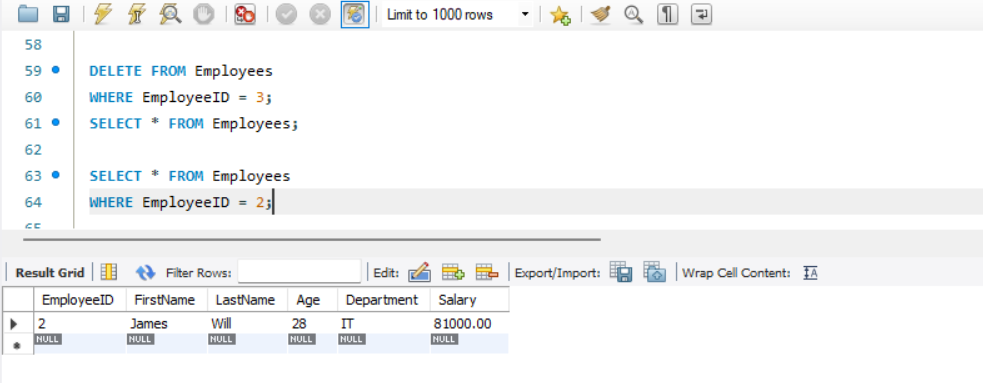


**-- To verify the update (checking updated values for EmployeeID 2)**

SELECT \* FROM Employees

WHERE EmployeeID = 2;

**Output:(Screenshot):**

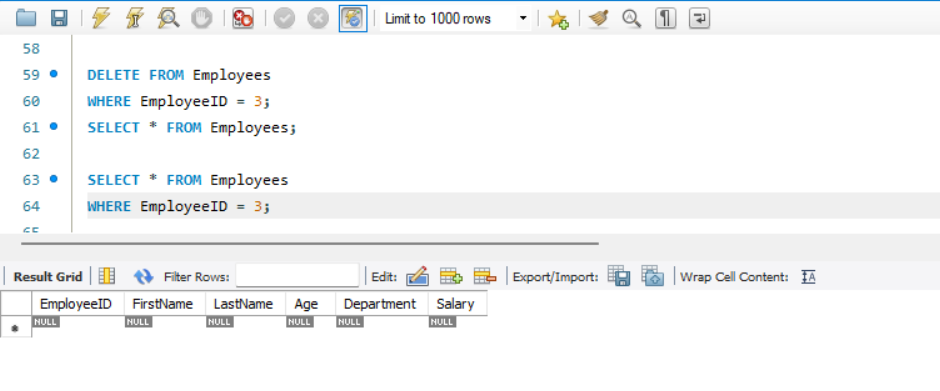


**-- To verify the deletion (checking if EmployeeID 1 exists)**

SELECT \* FROM Employees

WHERE EmployeeID = 3;

**Output:(Screenshot):**



# Assignment no.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 TABLE Sales (

SaleID INT PRIMARY KEY AUTO\_INCREMENT,

Product VARCHAR(50),

Quantity INT,

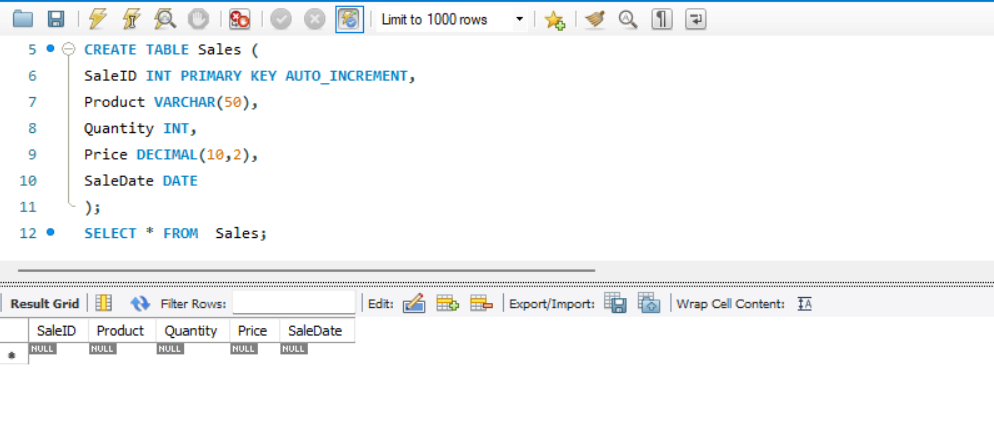
Price DECIMAL(10,2),

SaleDate DATE

);

SELECT \* FROM Sales;

**Output:(Screenshot):**



**Data Insertion :**

**--Inserting 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'),

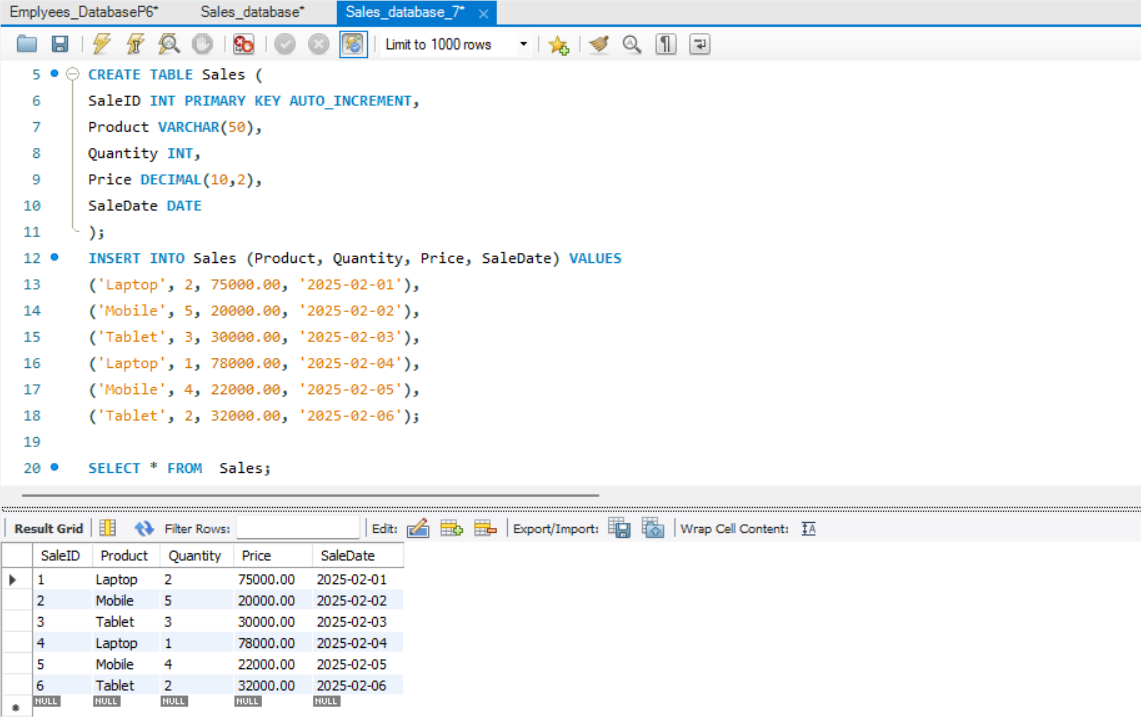
('Tablet', 3, 30000.00, '2025-02-03'),

('Laptop', 1, 78000.00, '2025-02-04'),

('Mobile', 4, 22000.00, '2025-02-05'),

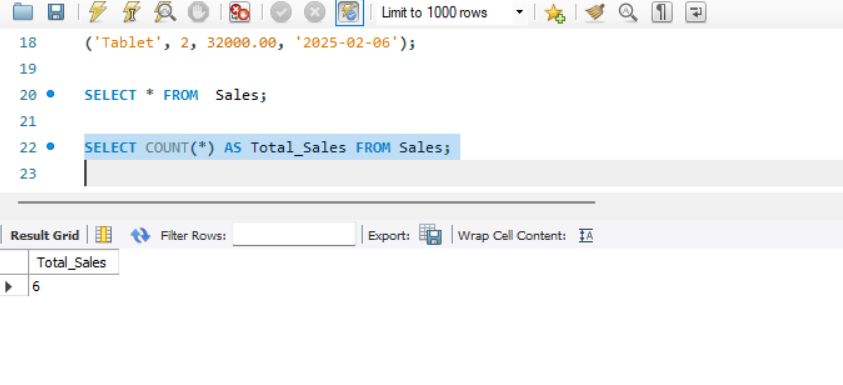
('Tablet', 2, 32000.00, '2025-02-06');

**Output:(Screenshot):**



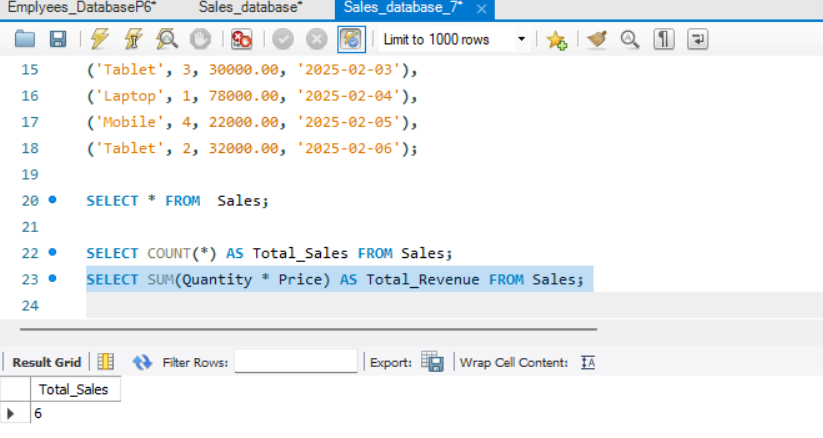
**-- Count the number of sales records** SELECT COUNT(\*) AS Total\_Sales FROM Sales;

**Output:(Screenshot):**

**-- Sum of total revenue generated**

SELECT SUM(Quantity \* Price) AS Total\_Revenue FROM Sales;

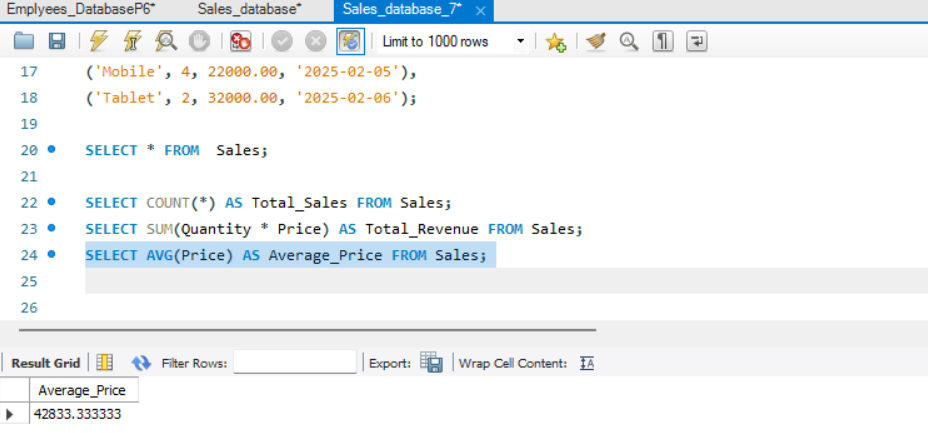
**Output:(Screenshot):**



**-- Average price of products sold**

SELECT AVG(Price) AS Average\_Price FROM Sales;

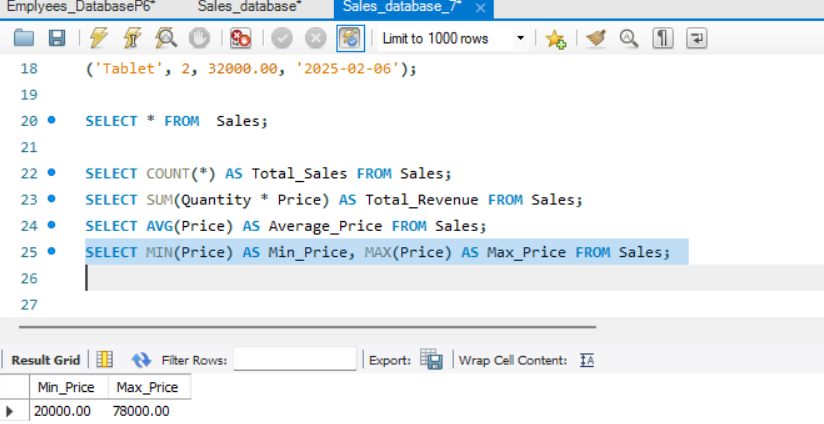
**Output:(Screenshot):**



**-- Minimum and Maximum price of a product sold**

SELECT MIN(Price) AS Min\_Price, MAX(Price) AS Max\_Price FROM Sales;

**Output:(Screenshot):**

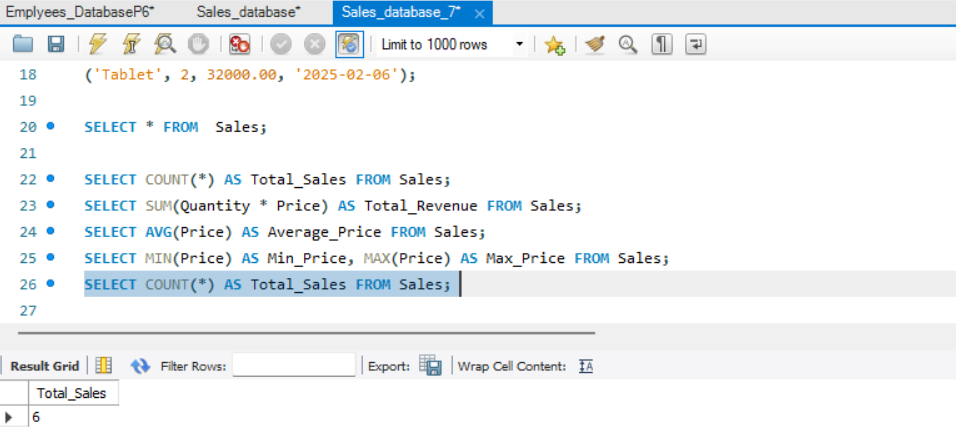


## **COUNT**-

**-- 1. Count the total number of sales records**

SELECT COUNT(\*) AS Total\_Sales FROM Sales;

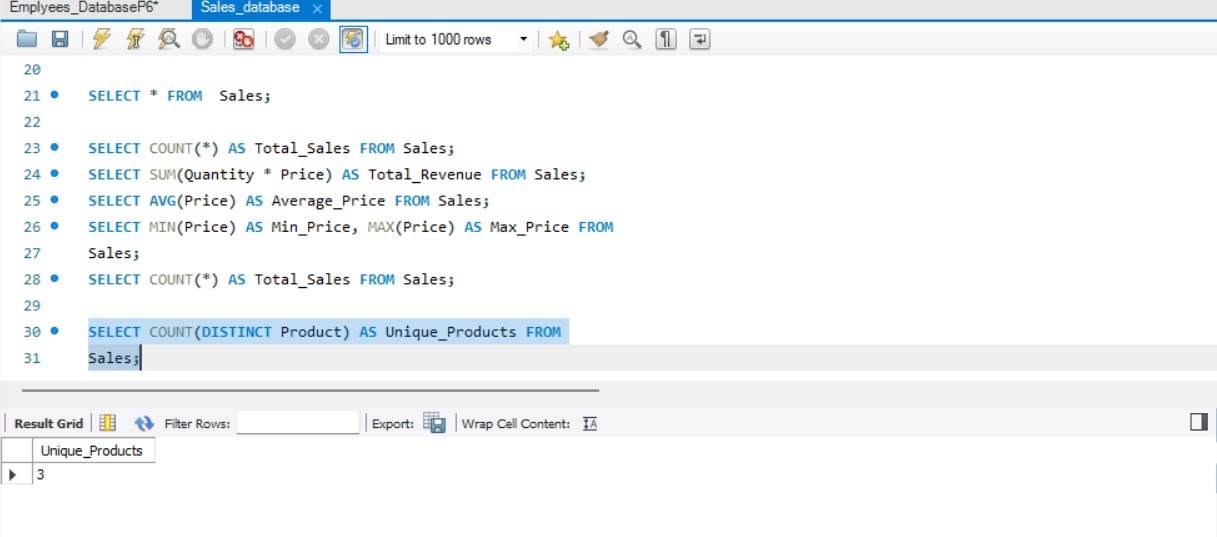
**Output:(Screenshot):**

****

**-- 2. Count the number of distinct products sold**

SELECT COUNT(DISTINCT Product) AS Unique\_Products FROM Sales;

**Output:(Screenshot):**



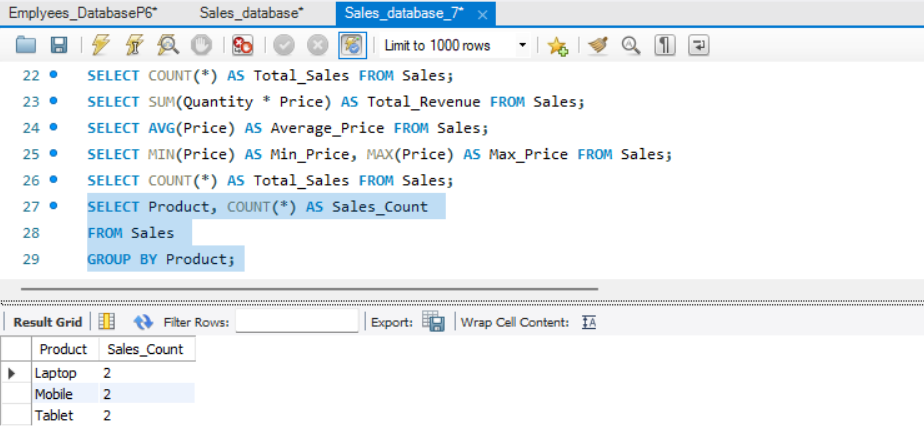
**3. Count the number of sales per product**

SELECT Product, COUNT(\*) AS Sales\_Count

FROM Sales

GROUP BY Product;

**Output:(Screenshot):**

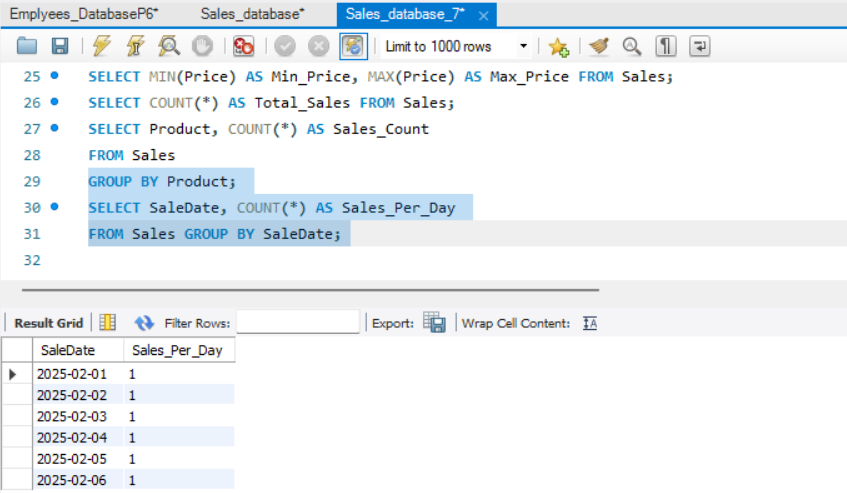


**-- 4. Count the number of sales per day**

SELECT SaleDate, COUNT(\*) AS Sales\_Per\_Day

FROM Sales GROUP BY SaleDate;

**Output:(Screenshot):**



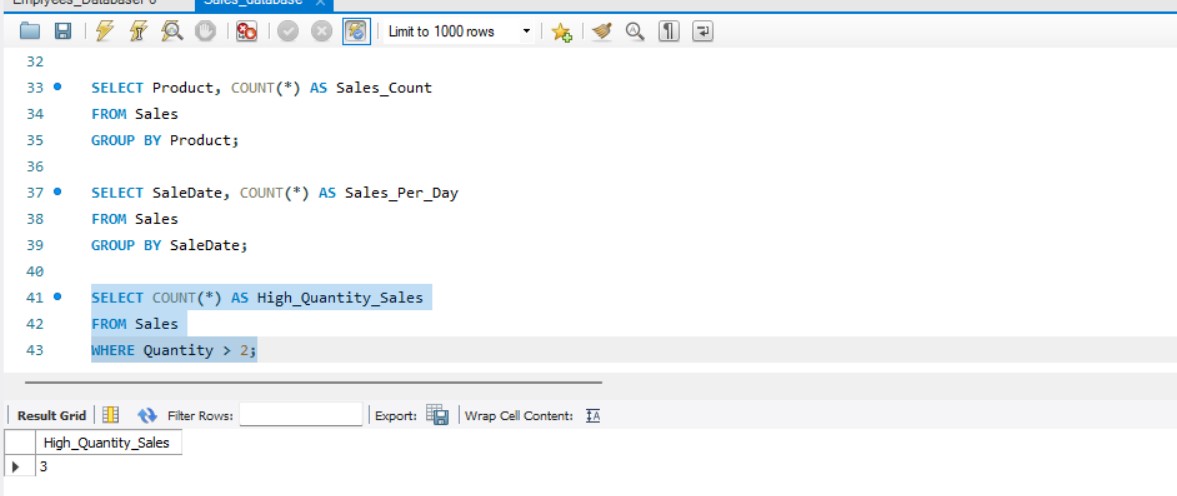
**5. Count the number of sales where more than 2 units were sold**

SELECT COUNT(\*) AS High\_Quantity\_Sales

FROM Sales

WHERE Quantity > 2;

**Output:(Screenshot):**



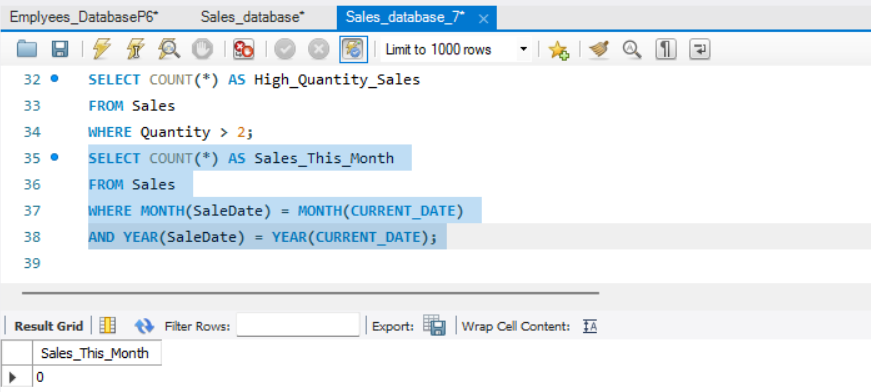
**-- 6. Count the number of sales in the current month**

SELECT COUNT(\*) AS Sales\_This\_Month

FROM Sales

WHERE MONTH(SaleDate) = MONTH(CURRENT\_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT\_DATE); **Output:(Screenshot):**

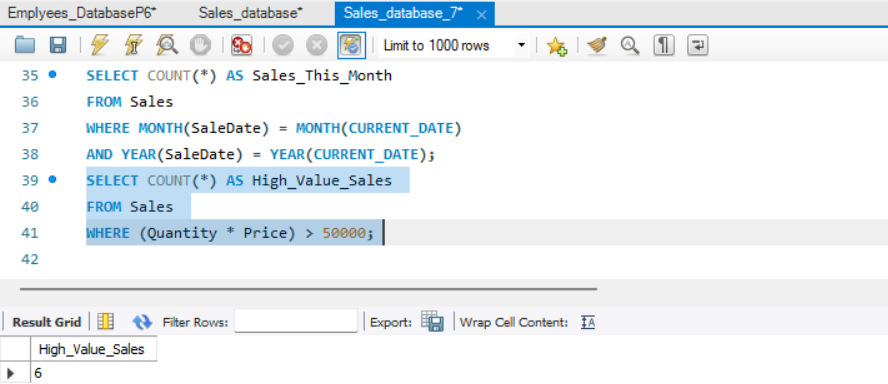


**7. Count the number of sales transactions where total sale value was more than 50,000**

SELECT COUNT(\*) AS High\_Value\_Sales

FROM Sales

WHERE (Quantity \* Price) > 50000; **Output:(Screenshot):**



**-- 8. Count the number of sales records for each product where total sale value is greater than ᴃ40,000**

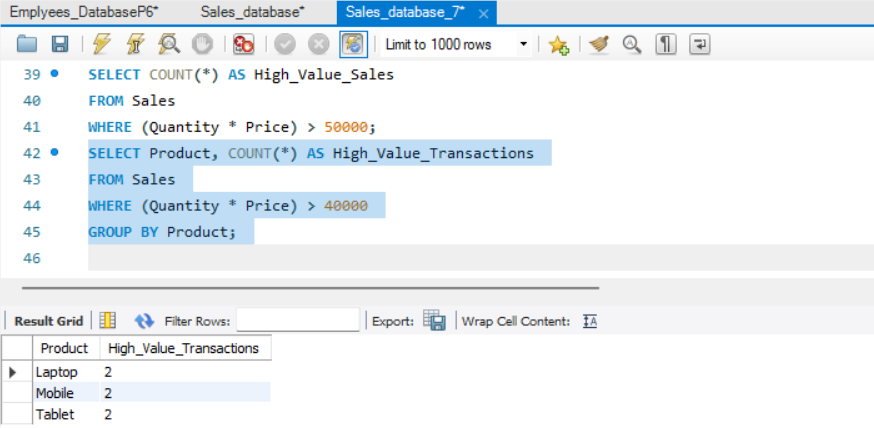
SELECT Product, COUNT(\*) AS High\_Value\_Transactions

FROM Sales

WHERE (Quantity \* Price) > 40000

GROUP BY Product;

**Output:(Screenshot):**



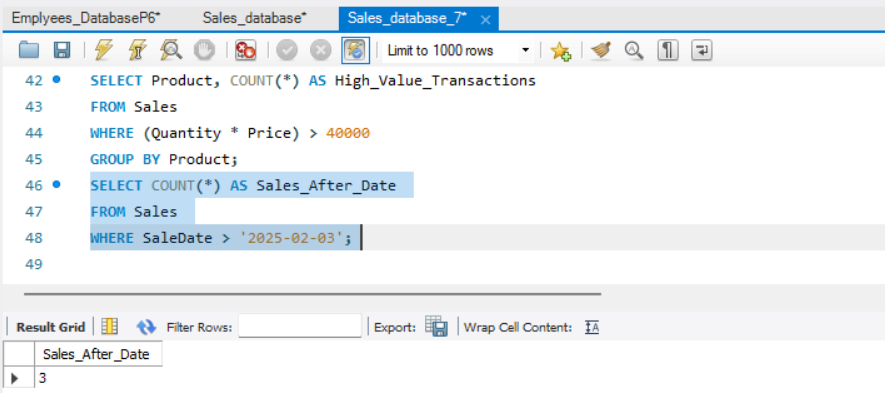
**9. Count the number of sales made after a specific date (e.g., Feb 3, 2025)**

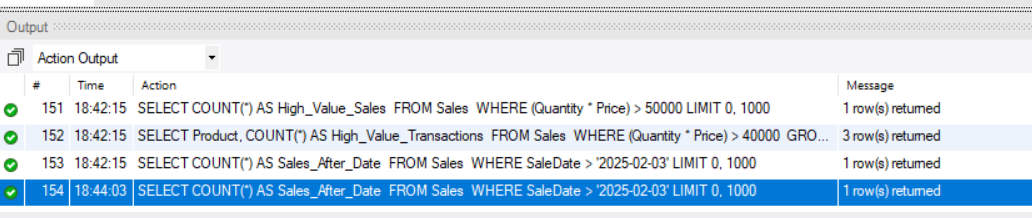
SELECT COUNT(\*) AS Sales\_After\_Date

FROM Sales

WHERE SaleDate > '2025-02-03';

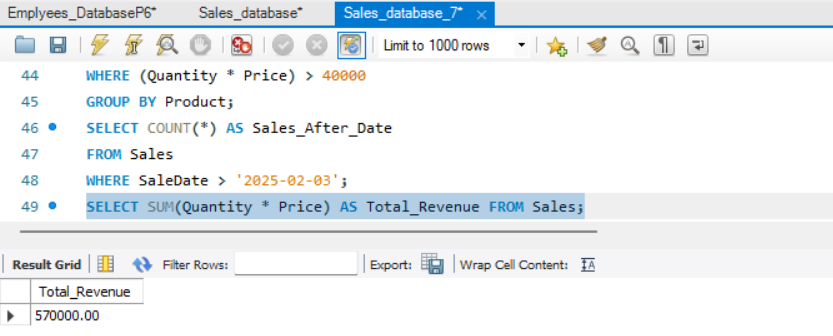
**Output:(Screenshot):**





## **SUM- -- 1. Sum of total revenue generated**

SELECT SUM(Quantity \* Price) AS Total\_Revenue FROM Sales; **Output:(Screenshot):**



**-- 2. Sum of total quantity of products sold** SELECT SUM(Quantity) AS Total\_Quantity\_Sold FROM

Sales;

**Output:(Screenshot):**



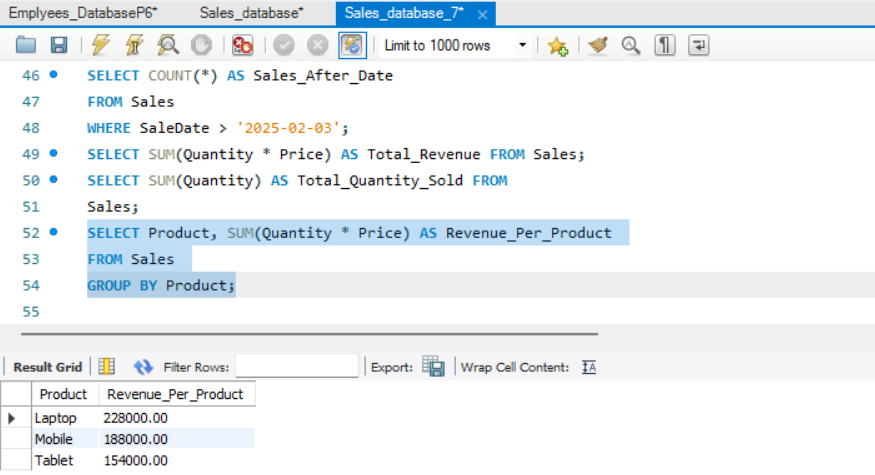
## **-- 3. Sum of total revenue per product**

SELECT Product, SUM(Quantity \* Price) AS Revenue\_Per\_Product

FROM Sales

GROUP BY Product;

**Output:(Screenshot):**



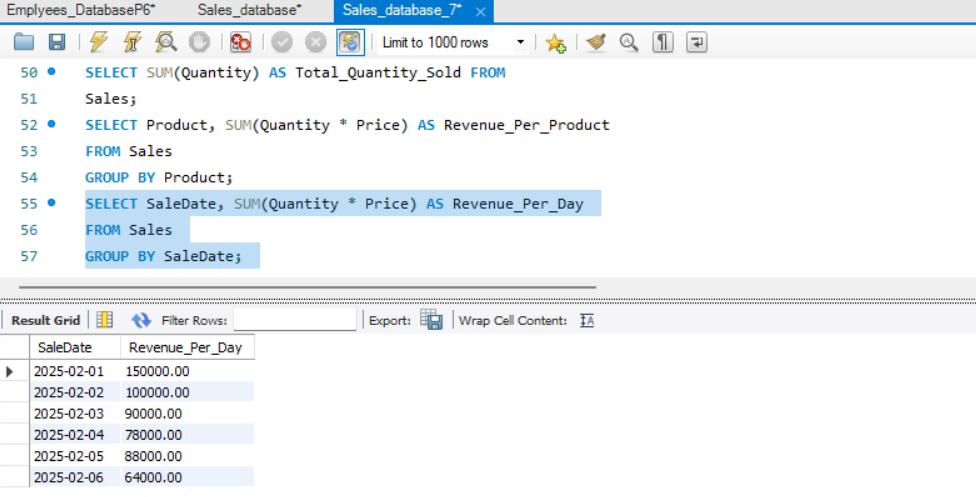
## **-- 4. Sum of total revenue per day**

SELECT SaleDate, SUM(Quantity \* Price) AS Revenue\_Per\_Day

FROM Sales

GROUP BY SaleDate;

**Output:(Screenshot):**



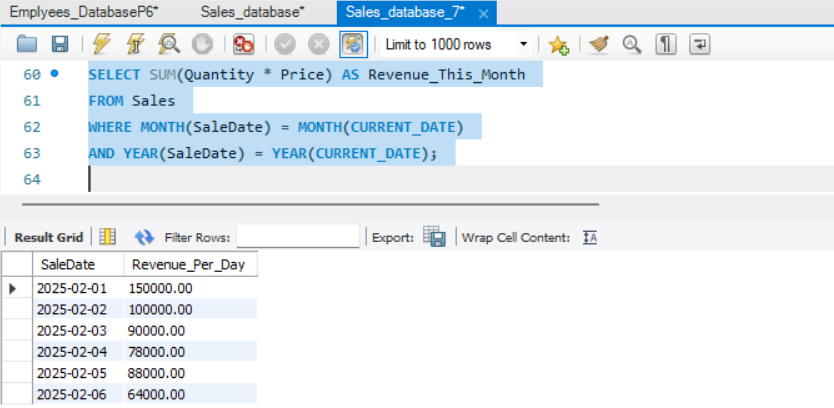
## **-- 5. Sum of total revenue in the current month**

SELECT SUM(Quantity \* Price) AS Revenue\_This\_Month

FROM Sales

WHERE MONTH(SaleDate) = MONTH(CURRENT\_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT\_DATE); **Output:(Screenshot):**



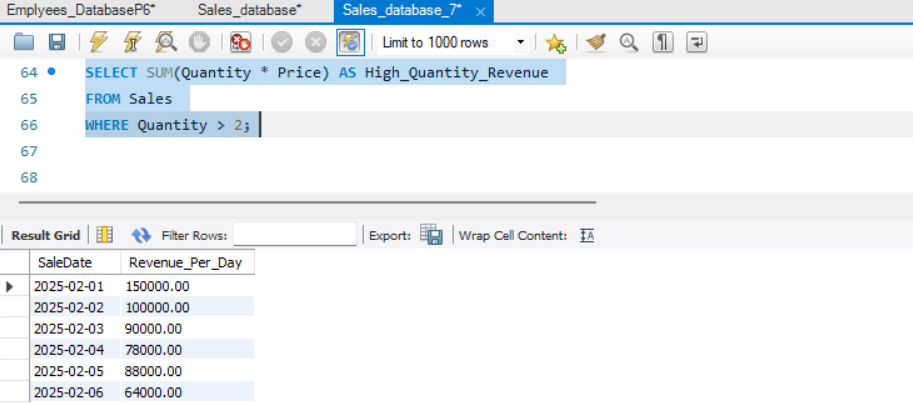
**--** **6. Sum of revenue for sales where quantity sold is greater than 2**

SELECT SUM(Quantity \* Price) AS High\_Quantity\_Revenue

FROM Sales

WHERE Quantity > 2;

**Output:(Screenshot):**

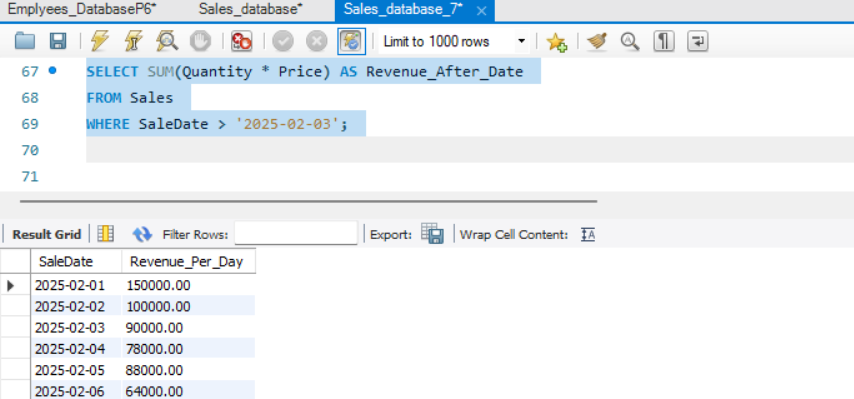


## **-- 7. Sum of total revenue generated after a specific date (e.g., Feb 3, 2025)**

SELECT SUM(Quantity \* Price) AS Revenue\_After\_Date

FROM Sales

WHERE SaleDate > '2025-02-03'; **Output:(Screenshot):**



**-- 8. Sum of revenue per product where the total revenue per transaction is greater than ᴃ40,000**

SELECT Product, SUM(Quantity \* Price) AS High\_Value\_Revenue

FROM Sales

WHERE (Quantity \* Price) > 40000

GROUP BY Product;

**Output:(Screenshot):**



## **AVG**-

**-- 1. Average price of products sold**

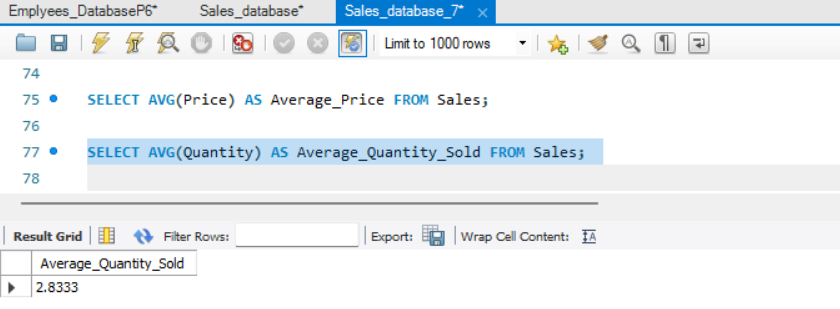
SELECT AVG(Price) AS Average\_Price FROM Sales;

**Output:(Screenshot):**



**-- 2. Average quantity of products sold per transaction** SELECT AVG(Quantity) AS Average\_Quantity\_Sold FROM Sales;

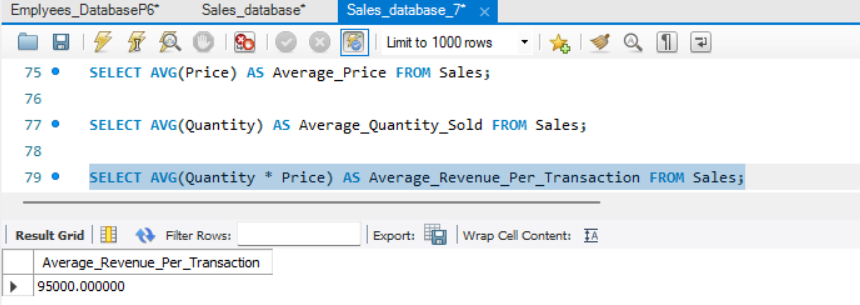
**Output:(Screenshot):**



**--3. Average revenue per transaction**

SELECT AVG(Quantity \* Price) AS Average\_Revenue\_Per\_Transaction FROM Sales;

**Output:(Screenshot):**



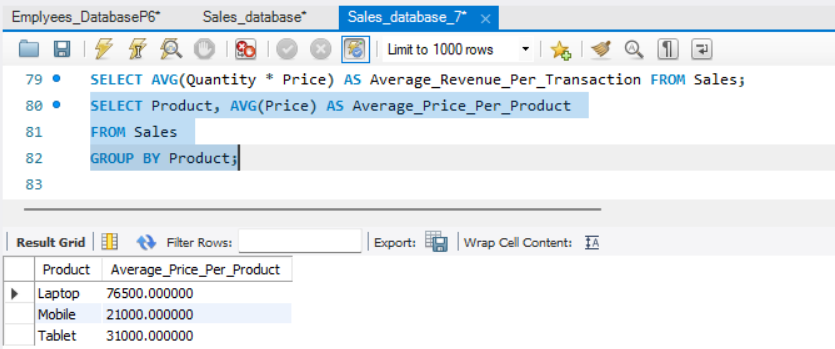
**-- 4. Average price per product**

SELECT Product, AVG(Price) AS Average\_Price\_Per\_Product

FROM Sales

GROUP BY Product;

**Output:(Screenshot):**



**--5. Average revenue per product**

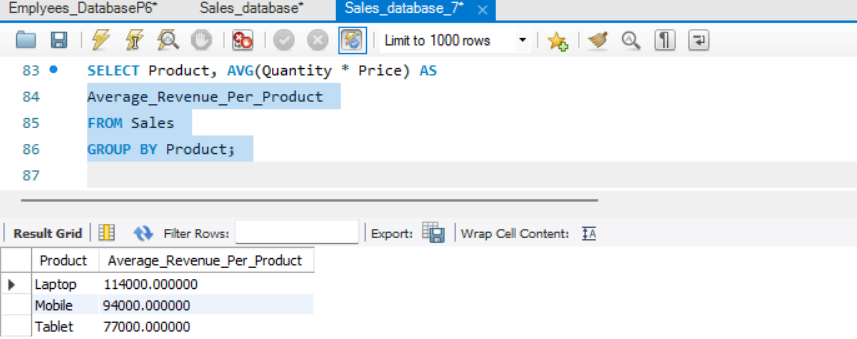
SELECT Product, AVG(Quantity \* Price) AS

Average\_Revenue\_Per\_Product

FROM Sales

GROUP BY Product;

**Output:(Screenshot):**



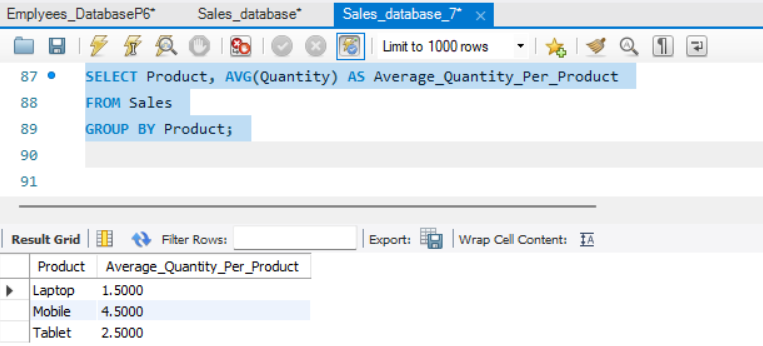
**-- 6. Average quantity sold per product**

SELECT Product, AVG(Quantity) AS Average\_Quantity\_Per\_Product

FROM Sales

GROUP BY Product;

**Output:(Screenshot):**



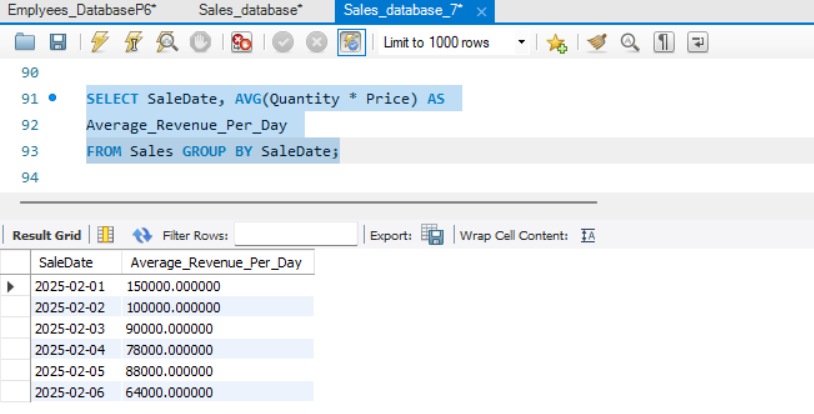
**--7. Average revenue per day**

SELECT SaleDate, AVG(Quantity \* Price) AS

Average\_Revenue\_Per\_Day

FROM Sales GROUP BY SaleDate;

**Output:(Screenshot):**



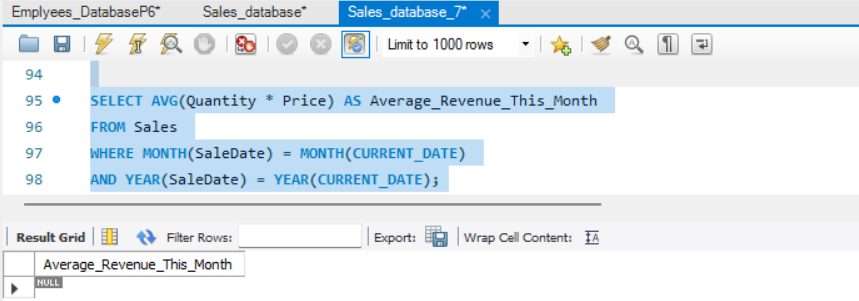
**-- 8. Average revenue in the current month**

SELECT AVG(Quantity \* Price) AS Average\_Revenue\_This\_Month

FROM Sales

WHERE MONTH(SaleDate) = MONTH(CURRENT\_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT\_DATE); **Output:(Screenshot):**



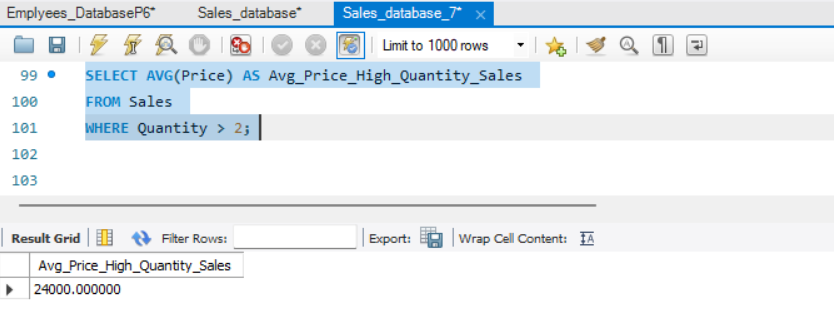
--**9. Average price of products where more than 2 units were sold**

SELECT AVG(Price) AS Avg\_Price\_High\_Quantity\_Sales

FROM Sales

WHERE Quantity > 2;

**Output:(Screenshot):**

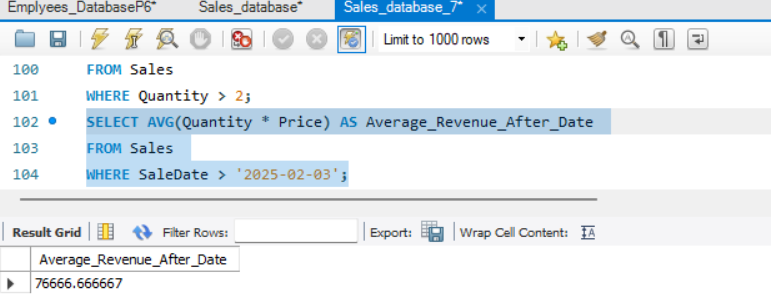


**-- 10. Average revenue after a specific date (e.g., Feb 3, 2025)**

SELECT AVG(Quantity \* Price) AS Average\_Revenue\_After\_Date

FROM Sales

WHERE SaleDate > '2025-02-03'; **Output:(Screenshot):**

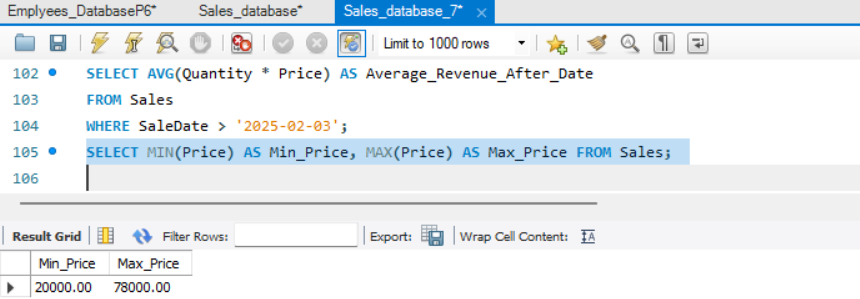


## **MIN**, **MAX**-

**-- 1. Minimum and Maximum price of a product sold**

SELECT MIN(Price) AS Min\_Price, MAX(Price) AS Max\_Price FROM Sales;

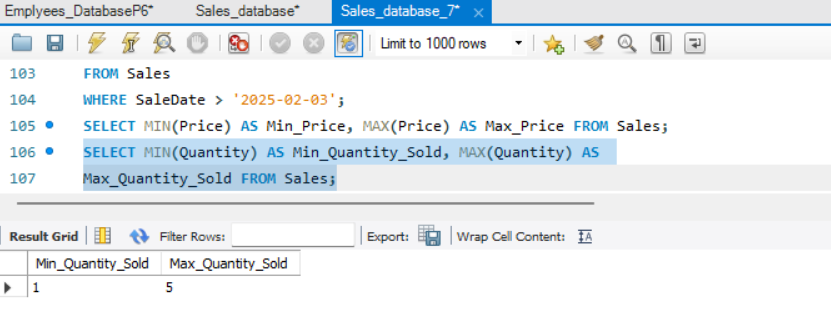
**Output:(Screenshot):**



**-- 2. 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; **Output:(Screenshot):**

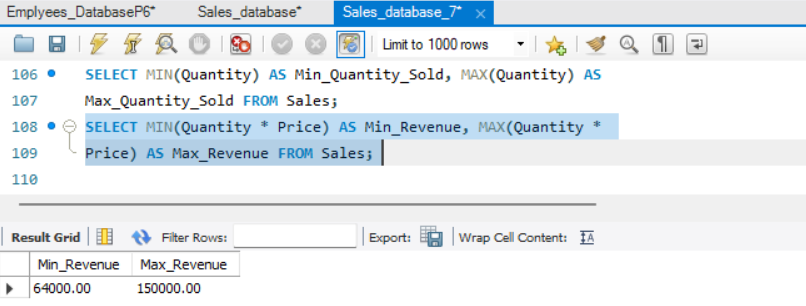
****

**--3. Minimum and Maximum revenue generated from a single transaction**

SELECT MIN(Quantity \* Price) AS Min\_Revenue, MAX(Quantity \*

Price) AS Max\_Revenue FROM Sales;

**Output:(Screenshot):**



**-- 4. Minimum and Maximum price per product**

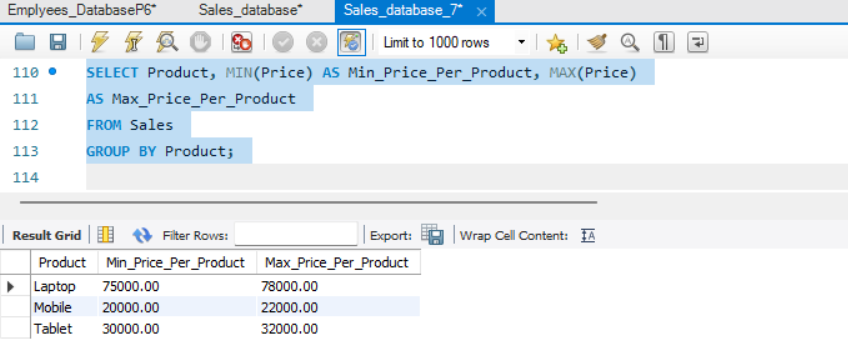
SELECT Product, MIN(Price) AS Min\_Price\_Per\_Product, MAX(Price)

AS Max\_Price\_Per\_Product

FROM Sales

GROUP BY Product;

**Output:(Screenshot):**



**-- 5. Minimum and Maximum revenue per product** SELECT Product, MIN(Quantity \* Price) AS

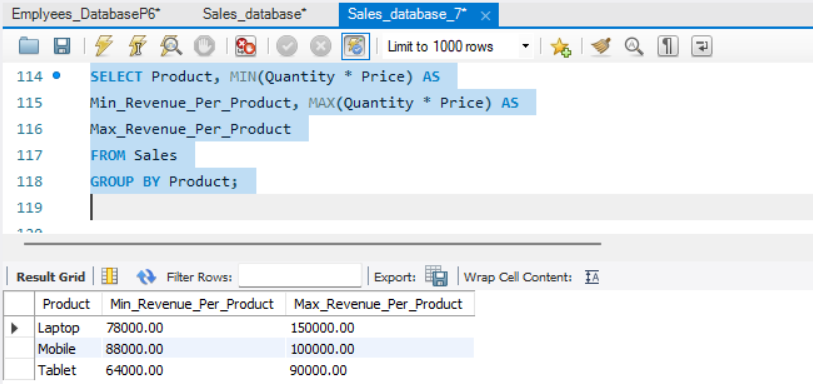
Min\_Revenue\_Per\_Product, MAX(Quantity \* Price) AS

Max\_Revenue\_Per\_Product

FROM Sales

GROUP BY Product;

**Output:(Screenshot):**



**-- 6. Minimum and Maximum quantity sold per product**

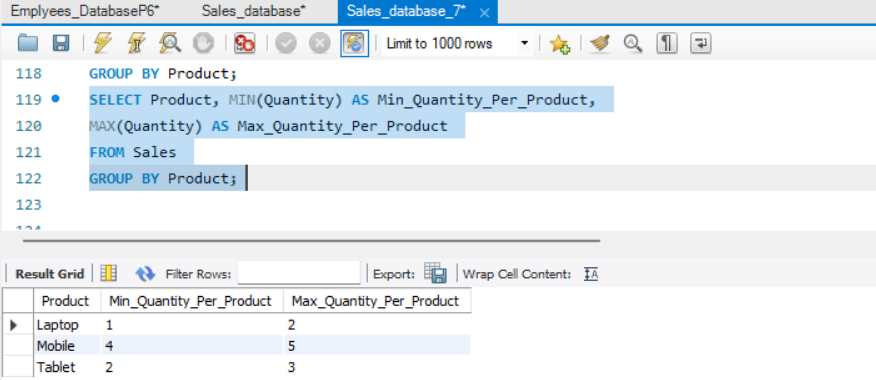
SELECT Product, MIN(Quantity) AS Min\_Quantity\_Per\_Product,

MAX(Quantity) AS Max\_Quantity\_Per\_Product

FROM Sales

GROUP BY Product;

**Output:(Screenshot):**



**--7. Minimum and Maximum revenue per day**

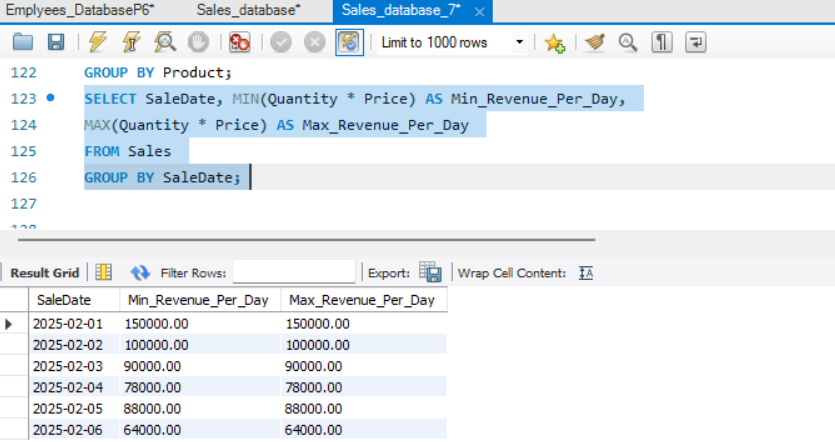
SELECT SaleDate, MIN(Quantity \* Price) AS Min\_Revenue\_Per\_Day,

MAX(Quantity \* Price) AS Max\_Revenue\_Per\_Day

FROM Sales

GROUP BY SaleDate;

**Output:(Screenshot):**



**-- 8. Minimum and Maximum revenue in the current month**

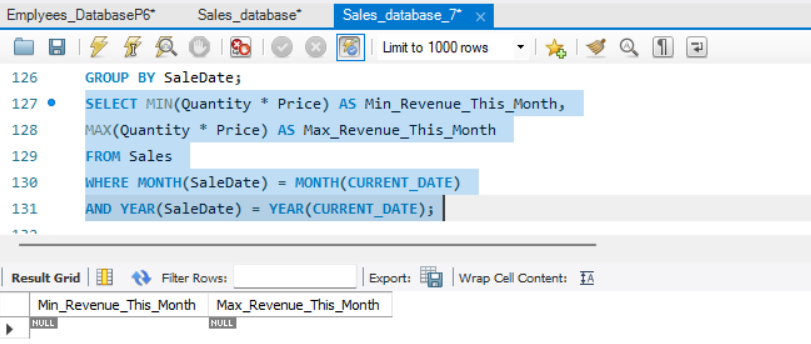
SELECT MIN(Quantity \* Price) AS Min\_Revenue\_This\_Month,

MAX(Quantity \* Price) AS Max\_Revenue\_This\_Month

FROM Sales

WHERE MONTH(SaleDate) = MONTH(CURRENT\_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT\_DATE); **Output:(Screenshot):**



**-- 9. Minimum and Maximum price of products where more than 2 units were sold**

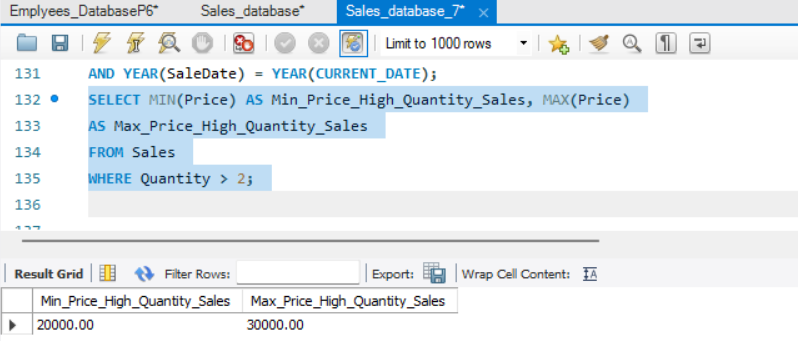
SELECT MIN(Price) AS Min\_Price\_High\_Quantity\_Sales, MAX(Price)

AS Max\_Price\_High\_Quantity\_Sales

FROM Sales

WHERE Quantity > 2;

**Output:(Screenshot):**



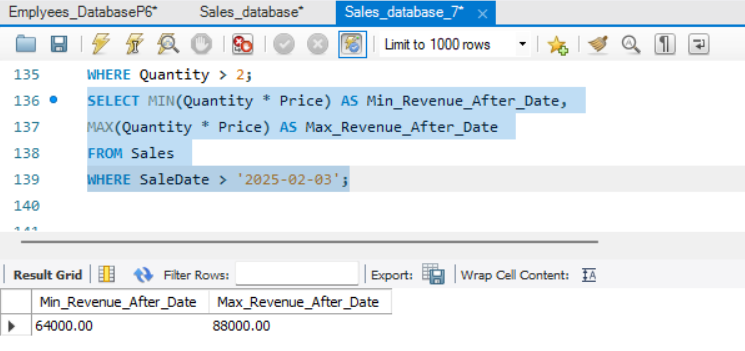
**-- 10. Minimum and Maximum revenue after a specific date (e.g., Feb 3, 2025)**

SELECT MIN(Quantity \* Price) AS Min\_Revenue\_After\_Date,

MAX(Quantity \* Price) AS Max\_Revenue\_After\_Date

FROM Sales

WHERE SaleDate > '2025-02-03'; **Output:(Screenshot):**



# Practical No.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: --For Creating Database And Table:**

CREATE DATABASE CompanyDB;

USE CompanyDB; CREATE TABLE Customers ( customer\_id INT PRIMARY KEY, customer\_name VARCHAR(100) NOT NULL

);

CREATE TABLE Orders ( order\_id INT PRIMARY KEY, order\_date DATE NOT NULL, customer\_id INT,

FOREIGN KEY (customer\_id) REFERENCES

Customers(customer\_id)

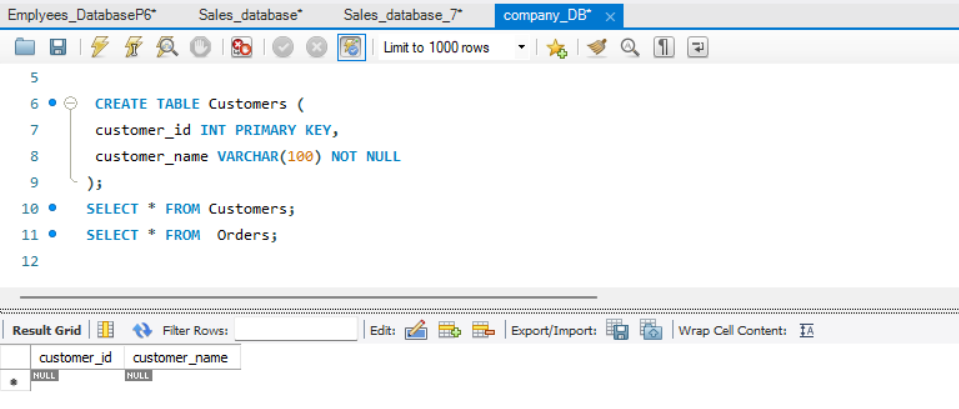
);

SELECT \* FROM Customers;

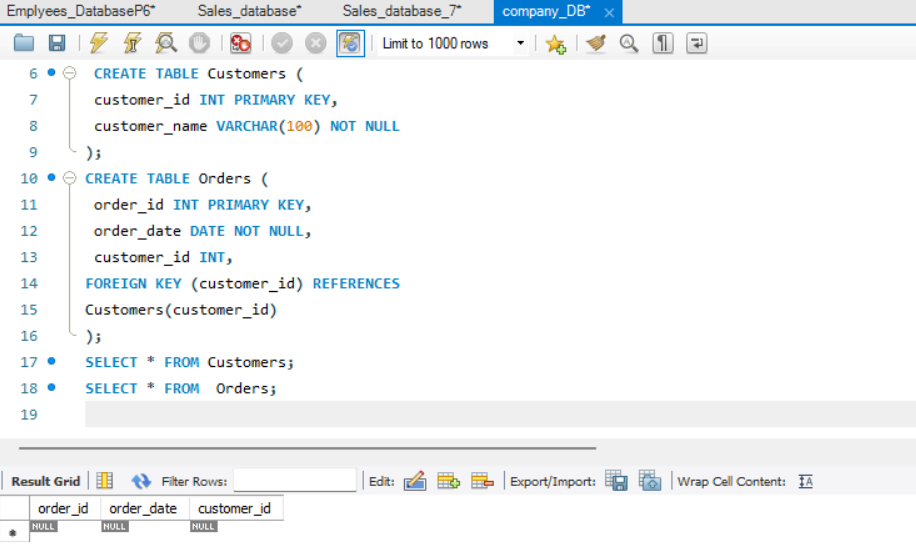
SELECT \* FROM Orders;

**Output:(Screenshot):**

**(Table 1)**



**(Table 2)**



**Data Insertion :**

**--Inserting Data into Customers and Orders Table**

INSERT INTO Customers (customer\_id, customer\_name) VALUES

(1, 'Shubham'),

(2, 'Bob'),

(3, 'Charlie'),

(4, 'David');

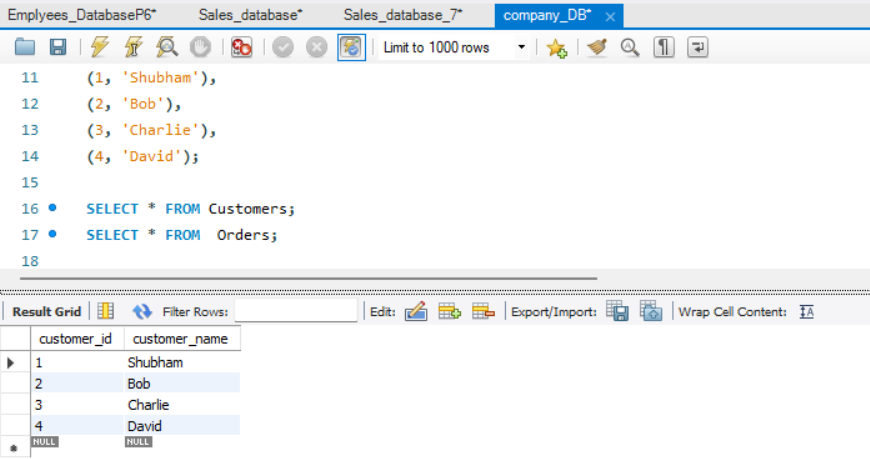
INSERT INTO Orders (order\_id, order\_date, customer\_id) VALUES

(101, '2024-01-01', 1),

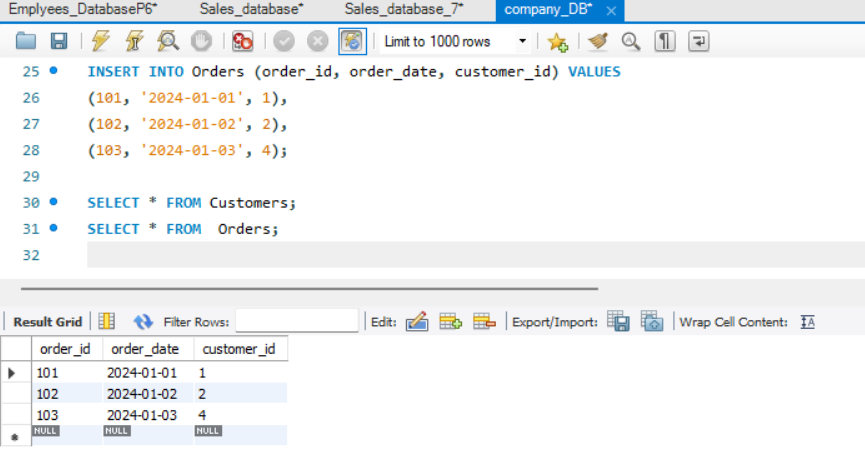
(102, '2024-01-02', 2),

(103, '2024-01-03', 4);

**Output:(Screenshot) : (Table 1)**



**Output:(Screenshot) : (Table 2)**



## **INNER** **JOIN** **Code:**

SELECT

c.customer\_id,

c.customer\_name,

o.order\_id,

o.order\_date

FROM

Customers c

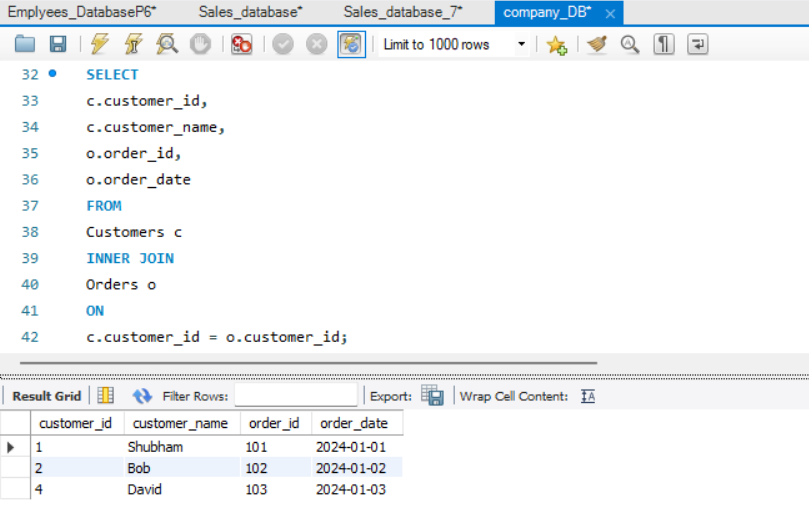
INNER JOIN

Orders o

ON

c.customer\_id = o.customer\_id;

**Output:(Screenshot) :**



## **LEFT JOIN - All Customers with their Orders (if any) :**

**Code:**

SELECT

c.customer\_id,

c.customer\_name,

o.order\_id,

o.order\_date

FROM

Customers c

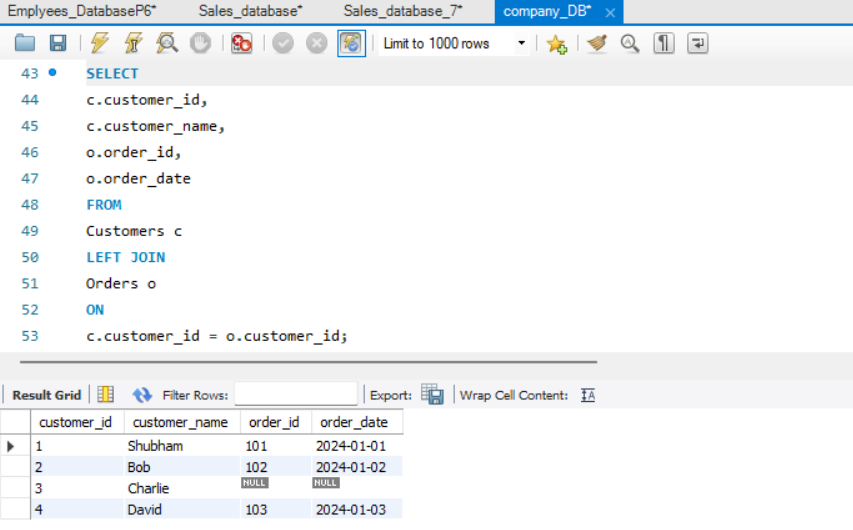
LEFT JOIN

Orders o

ON

c.customer\_id = o.customer\_id;

**Output:(Screenshot) :**



**RIGHT JOIN - All Orders with Customer details (if any) :**

**Code:**

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; **Output:(Screenshot) :**

