LAB FILE
Database Management Systems
(DBMS) Academic Year: 2024-2025

Name: Aditya Mhaismale
PRN Number: 2124UCSM1070
Branch: Cyber Security
Subject: DBMS

Index

Sr.No.	Practical Name
1	Install and set up MySQL. Create a database and table for employee details.
2	Create a table for storing student information and perform basic operations.
3	Create a table with various data types and perform queries.
4	Create a table with constraints like Primary Key, Foreign Key, and Unique.
5	Create a table with integrity constraints like NOT NULL, CHECK, and DEFAULT.
6	Use DDL and DML commands to create tables and manipulate data.
7	Create a Sales table and use aggregate functions to summarize sales data.
8	Perform JOIN operations on Customers and Orders tables.

Practical 1

Aim:

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

Code:

```
-- Step 1: Create a database

CREATE DATABASE EmployeeDB;


-- Step 2: Use the database

USE EmployeeDB;


-- Step 3: Create a table for employee details

CREATE TABLE Employees (

   EmployeeID INT PRIMARY KEY,

   Name VARCHAR(100),

   Department VARCHAR(100)

);


-- Step 4: Insert data into the table

INSERT INTO Employees (EmployeeID, Name, Department)

VALUES (1, 'Aditya', 'HR'),

            (2, 'Vishnu', 'Finance');
```

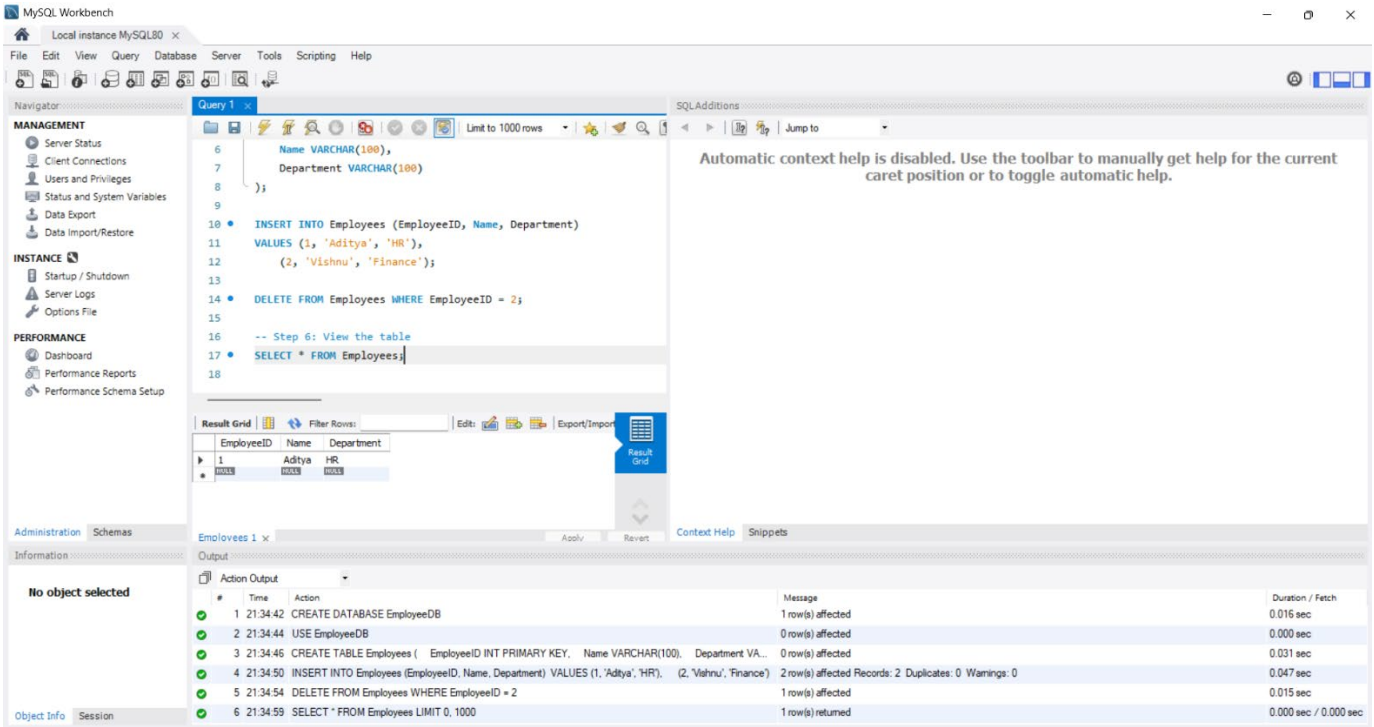
-- Step 5: Delete a record from the table

DELETE FROM Employees WHERE EmployeeID = 2;

-- Step 6: View the table

SELECT * FROM Employees;

Screenshots of Output:



Practical 2

Aim:

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

Code:

```
-- Step 1: Create a table for student information
CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   Name VARCHAR(100),
   Age INT,
   Grade CHAR(2)
);

-- Step 2: Insert sample data
INSERT INTO Students (StudentID, Name, Age, Grade)
VALUES (1, 'Aditya', 18, 'A'),
            (2, 'Urmi', 17, 'B');
```

-- Step 3: Update a record

UPDATE Students SET Grade = 'A+' WHERE StudentID = 1;

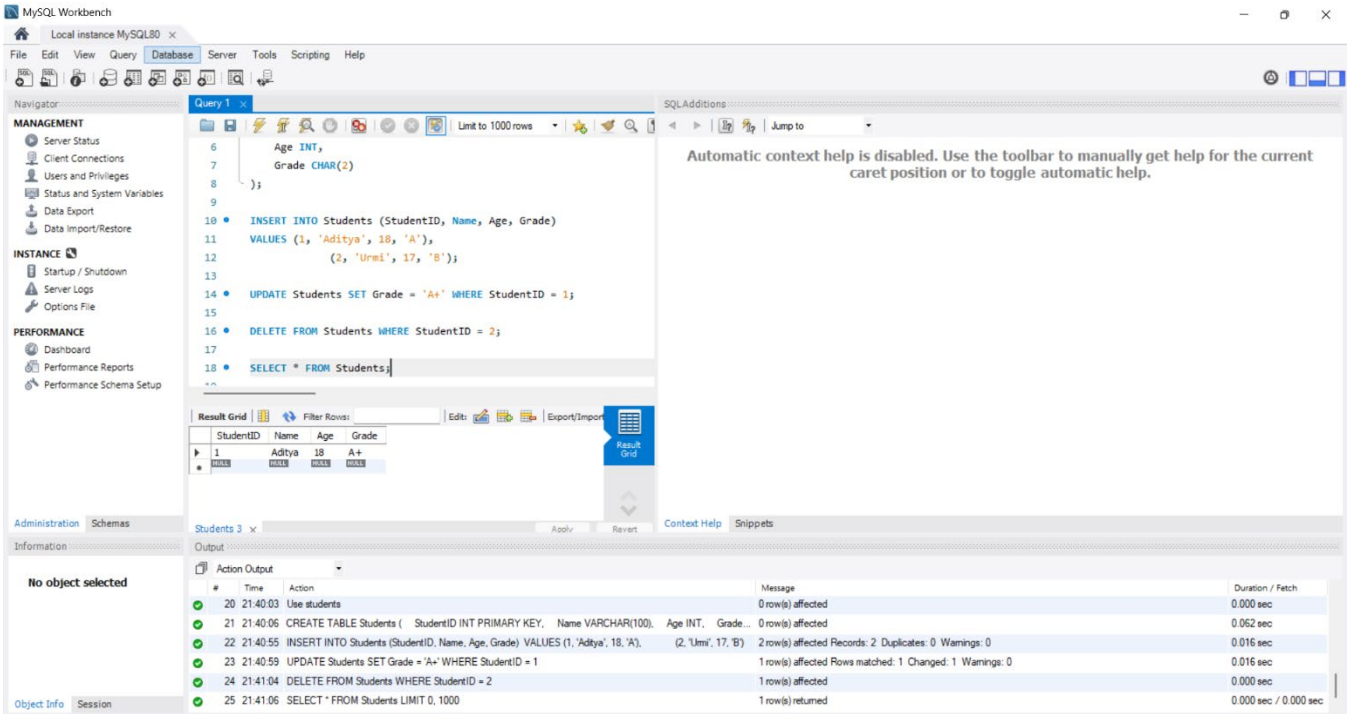
-- Step 4: Delete a record

DELETE FROM Students WHERE StudentID = 2;

-- Step 5: Retrieve all records

SELECT * FROM Students;

Screenshots of Output:



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 a table with various data types
CREATE TABLE EmployeeDetails (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    Salary DECIMAL(10, 2),
    JoiningDate DATE,
    ActiveStatus BOOLEAN
);

-- Step 2: Insert sample data
INSERT INTO EmployeeDetails (EmployeeID, Name, Salary, JoiningDate,
ActiveStatus)
VALUES (1, 'Michael Jordan', 50000.00, '2023-01-15', TRUE),
            (2, 'Aditya', 60000.00, '2022-06-20', FALSE);
```

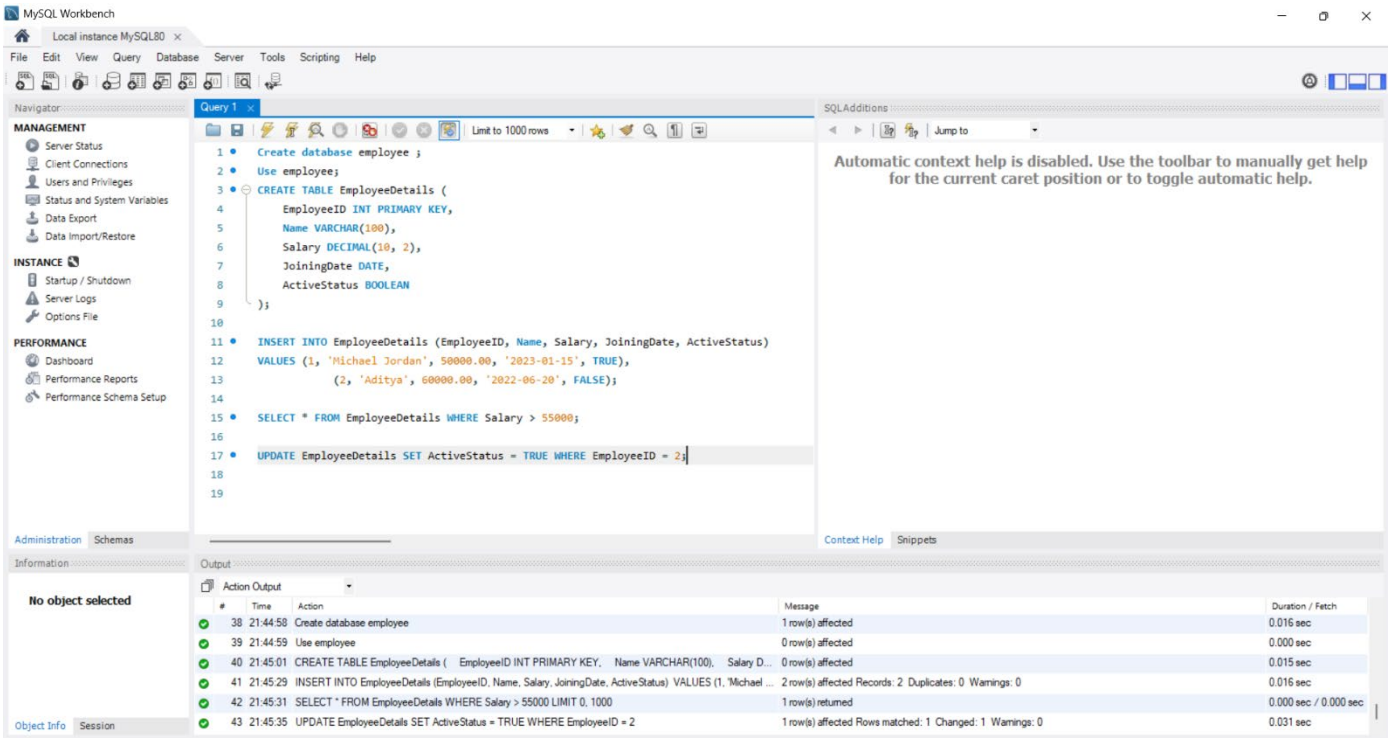
-- Step 3: Retrieve employees with salary > 55000

SELECT * FROM EmployeeDetails WHERE Salary > 55000;

-- Step 4: Update ActiveStatus

UPDATE EmployeeDetails SET ActiveStatus = TRUE WHERE EmployeeID = 2;

Screenshots of Output:



Practical 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:

```
-- Step 1: Create a department table
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(100) UNIQUE
);

-- Step 2: Create an employee table with foreign key
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

-- Step 3: Insert valid data

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES (1, 'HR'), (2, 'Finance');

INSERT INTO Employees (EmployeeID, Name, DepartmentID) VALUES (1, 'Aditya', 1);

-- Step 4: Insert invalid data (test constraints)

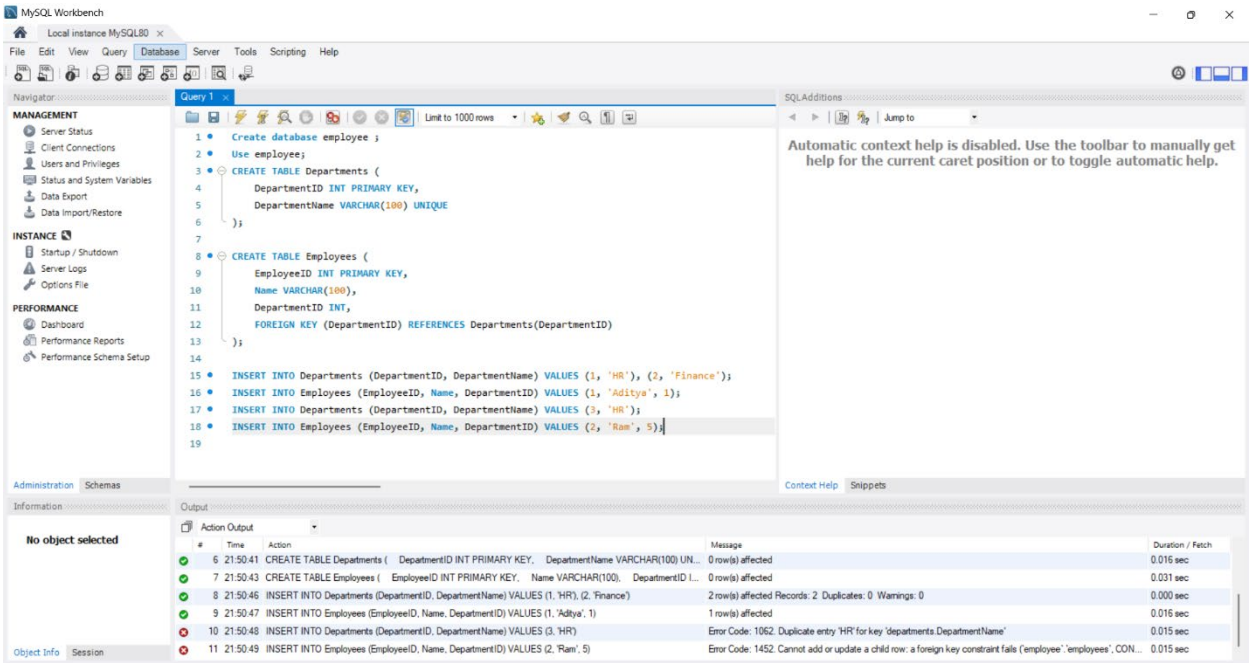
-- This will fail due to UNIQUE constraint

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES (3, 'HR');

-- This will fail due to FOREIGN KEY constraint

INSERT INTO Employees (EmployeeID, Name, DepartmentID) VALUES (2, 'Ram', 5);

Screenshots of Output:



Practical 5

Aim:

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

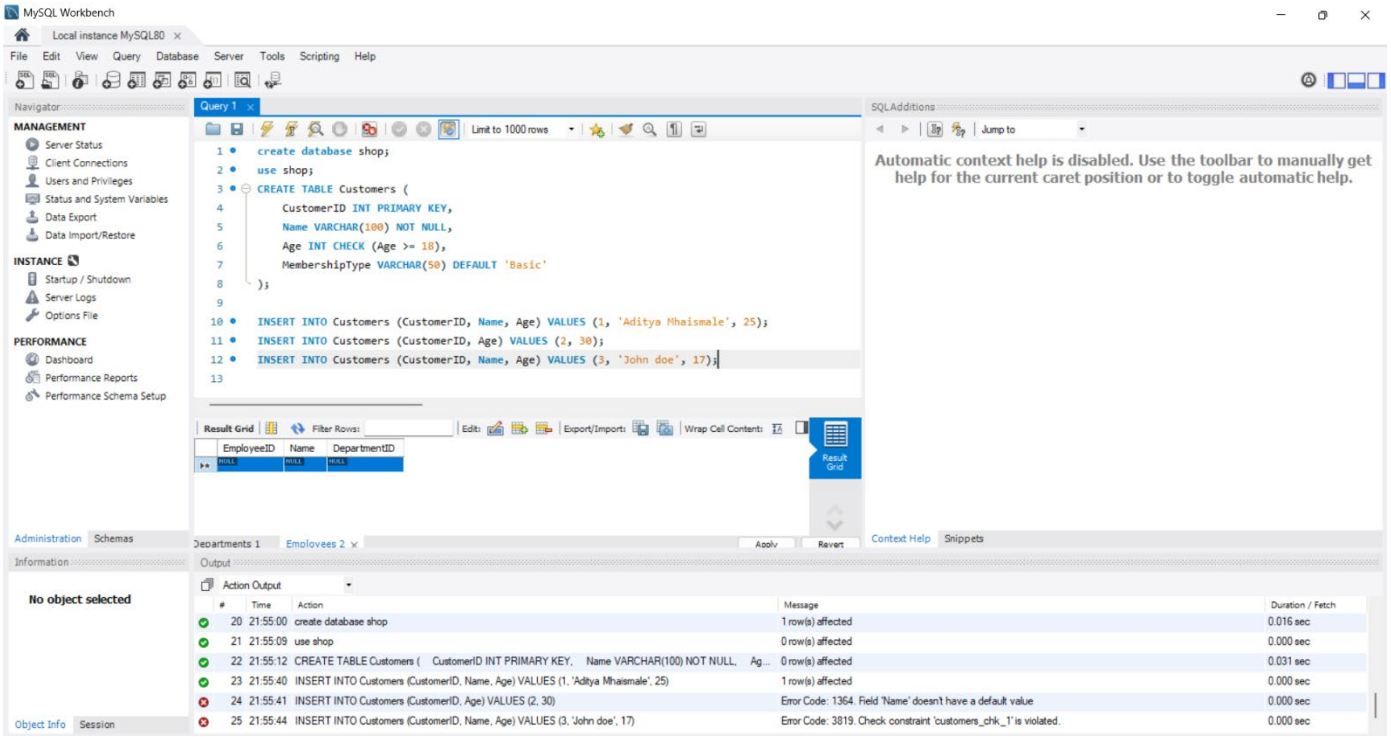
Code:

```
-- Step 1: Create a table with constraints
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Age INT CHECK (Age >= 18),
    MembershipType VARCHAR(50) DEFAULT 'Basic'
);

-- Step 2: Insert valid data
INSERT INTO Customers (CustomerID, Name, Age) VALUES (1, 'Aditya Mhaismale', 25);

-- Step 3: Insert invalid data (test constraints)
-- This will fail due to NOT NULL constraint
INSERT INTO Customers (CustomerID, Age) VALUES (2, 30);
```

-- This will fail due to CHECK constraint

INSERT INTO Customers (CustomerID, Name, Age) VALUES (3, 'John doe', 17);

Screenshots of Output:



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.

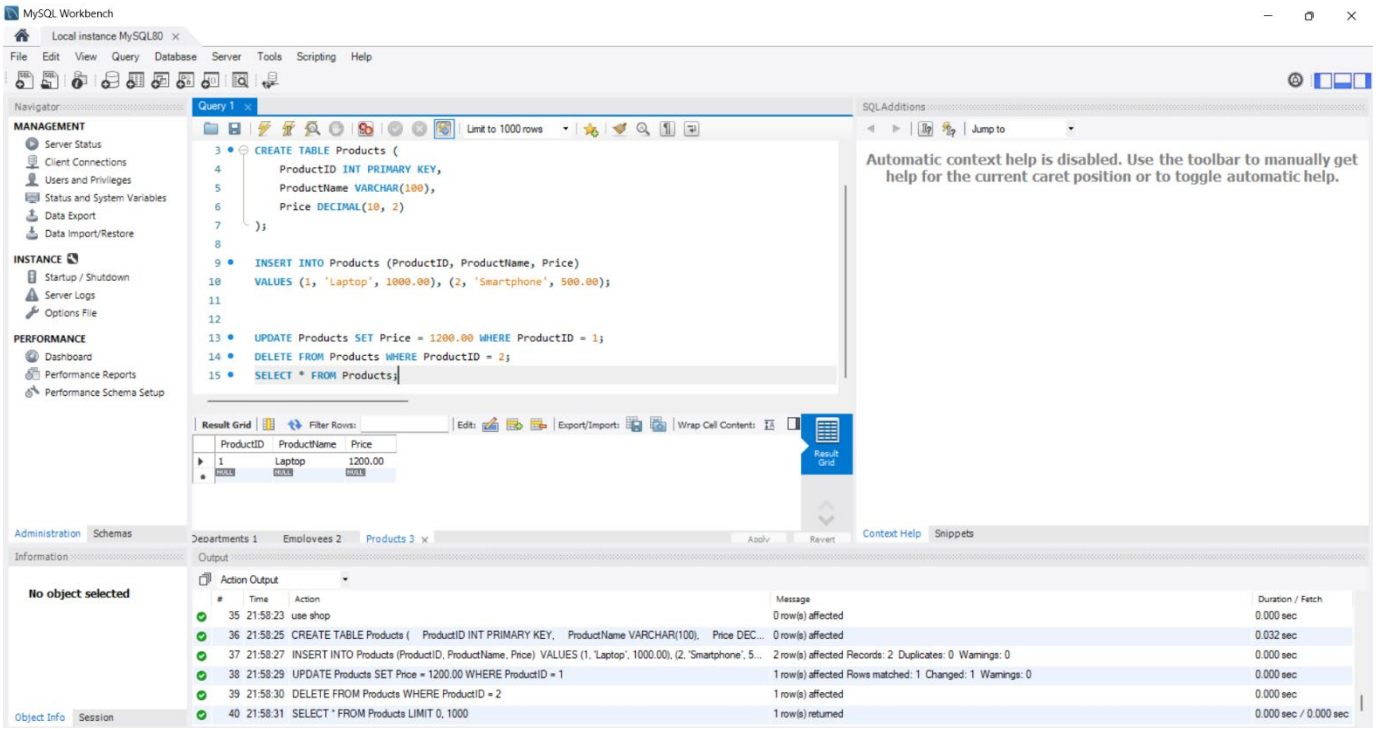
Code:

```
-- Step 1: Create a table
CREATE TABLE Products (
   ProductID INT PRIMARY KEY,
   ProductName VARCHAR(100),
   Price DECIMAL(10, 2)
);

-- Step 2: Insert data
INSERT INTO Products (ProductID, ProductName, Price)
VALUES (1, 'Laptop', 1000.00), (2, 'Smartphone', 500.00);

-- Step 3: Update data
UPDATE Products SET Price = 1200.00 WHERE ProductID = 1;

-- Step 4: Delete data
DELETE FROM Products WHERE ProductID = 2;
```

-- Step 5: Retrieve data

SELECT * FROM Products;

Screenshots of Output:



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:

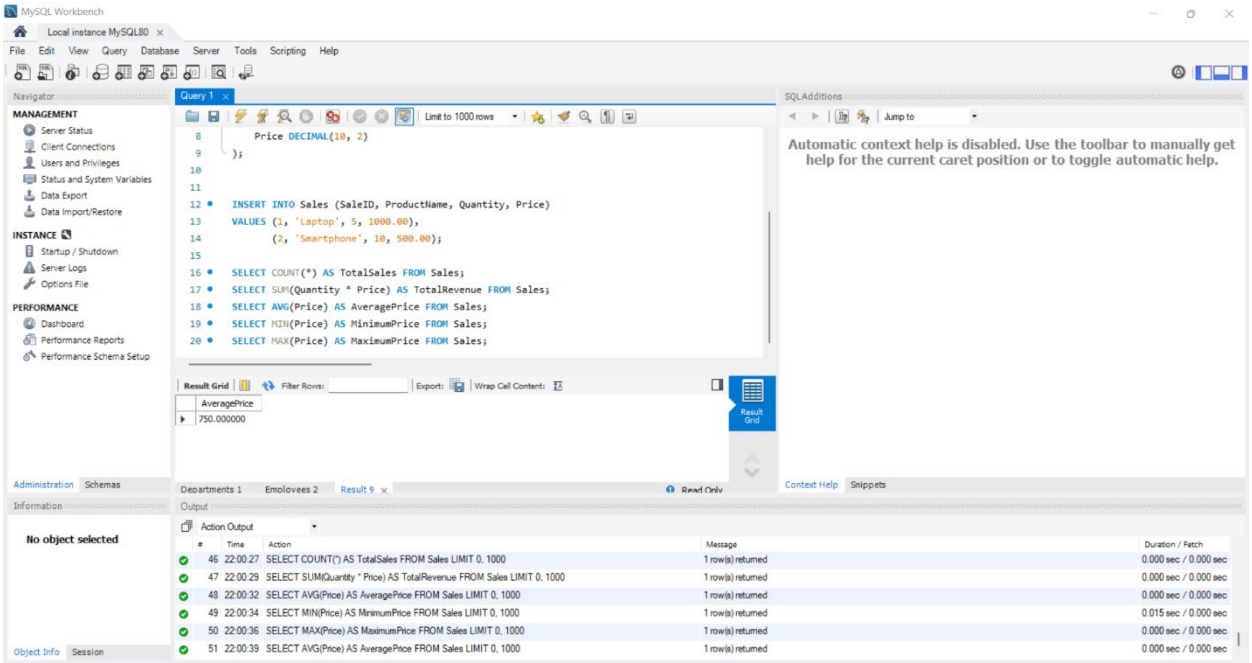
```
-- Step 1: Create a Sales table
CREATE TABLE Sales (
   SaleID INT PRIMARY KEY,
   ProductName VARCHAR(100),
   Quantity INT,
   Price DECIMAL(10, 2)
);

-- Step 2: Insert data
INSERT INTO Sales (SaleID, ProductName, Quantity, Price)
VALUES (1, 'Laptop', 5, 1000.00),
      (2, 'Smartphone', 10, 500.00);

-- Step 3: Use aggregate functions
SELECT COUNT(*) AS TotalSales FROM Sales;
SELECT SUM(Quantity * Price) AS TotalRevenue FROM Sales;
```

SELECT AVG(Price) AS AveragePrice FROM Sales;

SELECT MIN(Price) AS MinimumPrice FROM Sales;

SELECT MAX(Price) AS MaximumPrice FROM Sales;

Screenshots of Output:



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:

```
-- Step 1: Create Customers and Orders tables
CREATE TABLE Customers (
   CustomerID INT PRIMARY KEY,
   Name VARCHAR(100)
);

CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   CustomerID INT,
   OrderAmount DECIMAL(10, 2),
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

-- Step 2: Insert data
INSERT INTO Customers (CustomerID, Name) VALUES (1, 'Aditya'), (2, 'Bob');
INSERT INTO Orders (OrderID, CustomerID, OrderAmount) VALUES (101, 1, 500.00), (102, 2, 1000.00);
```

-- Step 3: Perform JOIN operations

-- INNER JOIN

SELECT c.Name, o.OrderAmount

FROM Customers c INNER JOIN Orders o ON c.CustomerID = o.CustomerID;

-- LEFT JOIN

SELECT c.Name, o.OrderAmount

FROM Customers c LEFT JOIN Orders o ON c.CustomerID = o.CustomerID;

-- RIGHT JOIN

SELECT c.Name, o.OrderAmount

FROM Customers c RIGHT JOIN Orders o ON c.CustomerID = o.CustomerID;

Screenshots of Output: