# **WEEK 2: Hands-On – Advanced SQL and NUnit & Moq**

## **Exercise 1: Ranking and Windows Function**

Goal: Use ROW\_NUMBER(), RANK(), DENSE\_RANK(), OVER(), and PARTITION BY.

Scenario: Find the top 3 most expensive products in each category using different ranking functions.

### **Creating a table and Inserting Values:**

#### **Code:**

CREATE DATABASE DN;

GO

USE DN;

GO

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName NVARCHAR(100),

Category NVARCHAR(50),

Price DECIMAL(10,2)

);

GO

INSERT INTO Products VALUES

(1, 'Mouse', 'Electronics', 799.99),

(2, 'Speaker', 'Electronics', 1499.50),

(3, 'Laptop', 'Electronics', 85000.00),

(4, 'Monitor', 'Electronics', 11999.00),

(5, 'Headphones', 'Electronics', 1499.50),

(6, 'Notebook', 'Stationery', 99.00),

(7, 'Bracelet', 'Jewellery', 450.50),

(8, 'Planner', 'Stationery', 249.00),

(9, 'Whiteboard', 'Stationery', 899.00),

(10, 'Marker Pack', 'Stationery', 199.00),

(11, 'Smartwatch', 'Electronics', 14999.00),

(12, 'Fountain Pen', 'Stationery', 899.00),

(13, 'Earrings', 'Jewellery', 200.00),

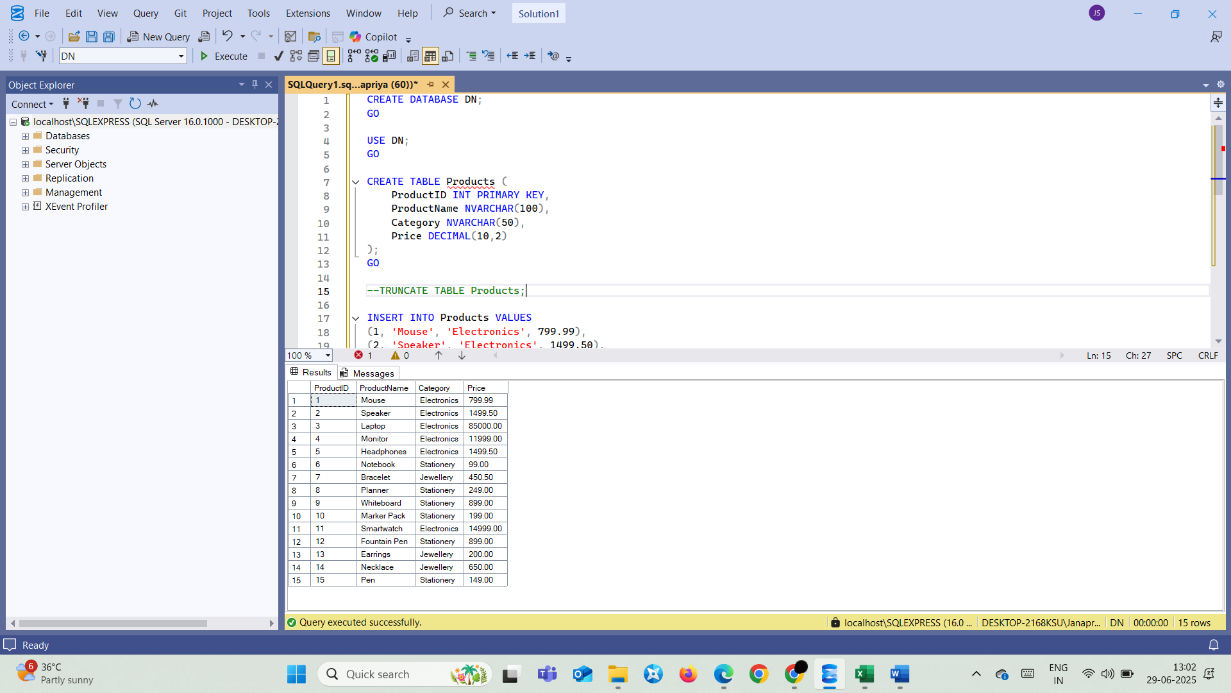
(14, 'Necklace', 'Jewellery', 650.00),

(15, 'Pen', 'Stationery', 149.00);

GO

SELECT \* FROM Products;

#### **Output:**



### **ROW\_NUMBER(), RANK(), DENSE\_RANK():**

#### **Code:**

SELECT

ProductID,

ProductName,

Category,

Price,

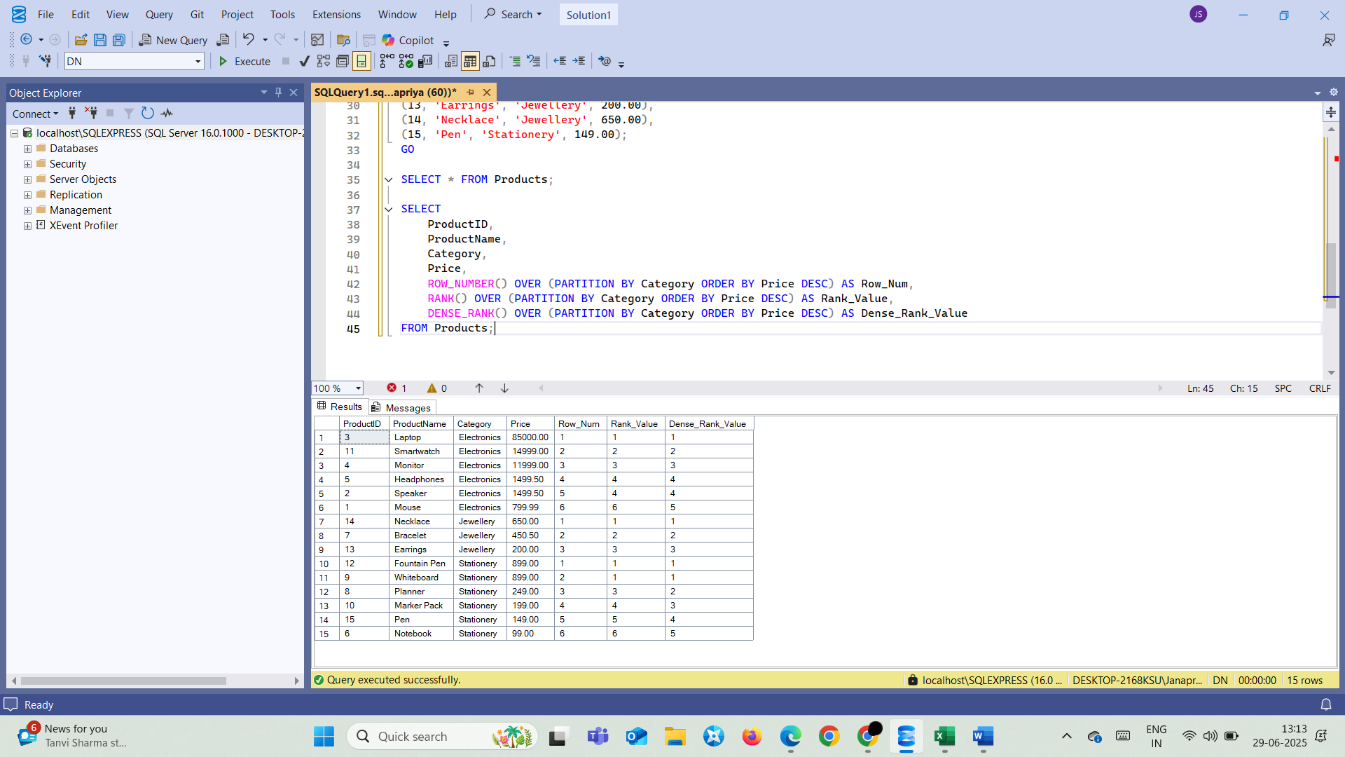
ROW\_NUMBER() OVER (PARTITION BY Category ORDER BY Price DESC) AS Row\_Num,

RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS Rank\_Value,

DENSE\_RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS Dense\_Rank\_Value

FROM Products;

#### **Output:**



### **Display Top Three Expensive Products in each category:**

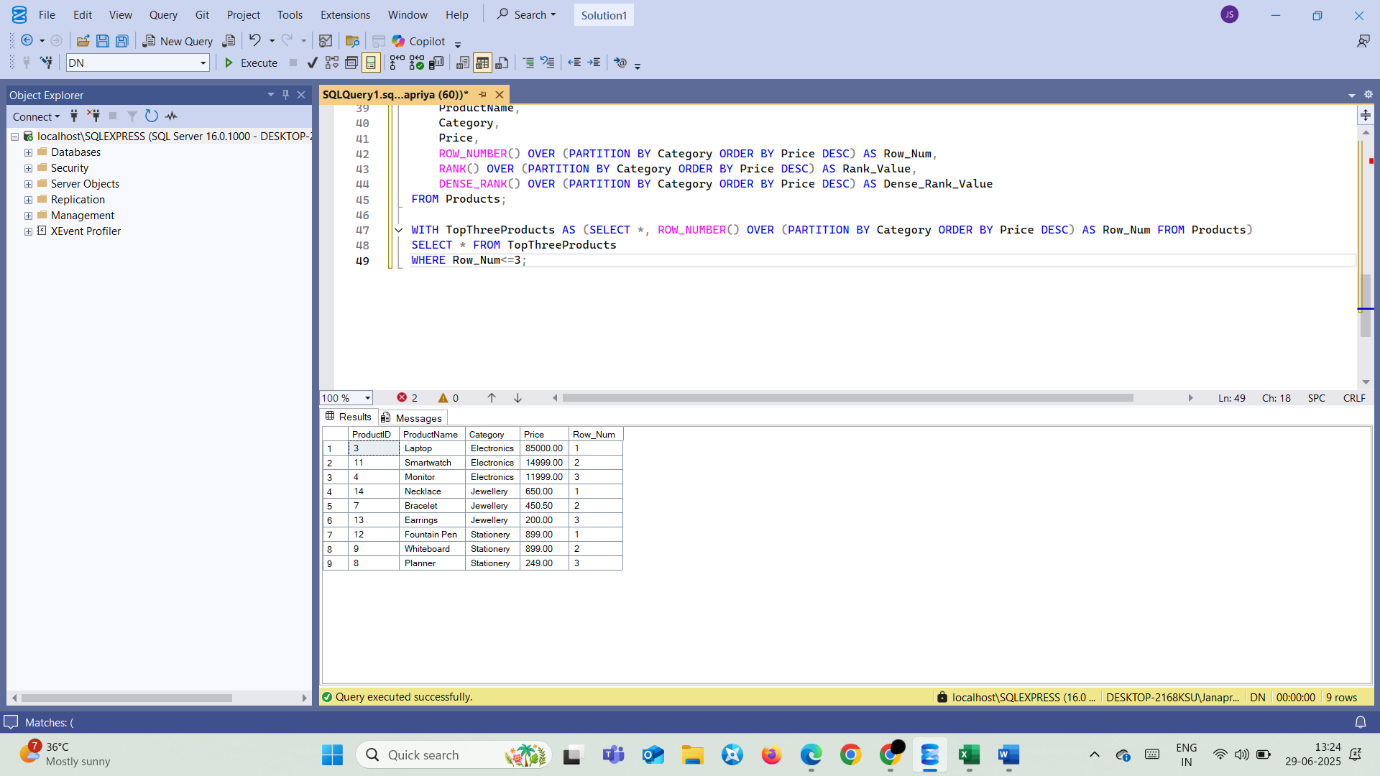
#### **Code:**

WITH TopThreeProducts AS (SELECT \*, ROW\_NUMBER() OVER (PARTITION BY Category ORDER BY Price DESC) AS Row\_Num FROM Products)

SELECT \* FROM TopThreeProducts

WHERE Row\_Num<=3;

#### **Output:**



## **Exercise 1: Create a Stored Procedure**

Goal: Create a Stored Procedure to retrieve employee details by department

### **Create Tables:**

#### **Code:**

USE DN;

GO

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY IDENTITY(1,1),

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT FOREIGN KEY REFERENCES Departments(DepartmentID),

Salary DECIMAL(10,2),

JoinDate DATE

);

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary,

JoinDate) VALUES

('John', 'Doe', 1, 5000.00, '2020-01-15'),

('Jane', 'Smith', 2, 6000.00, '2019-03-22'),

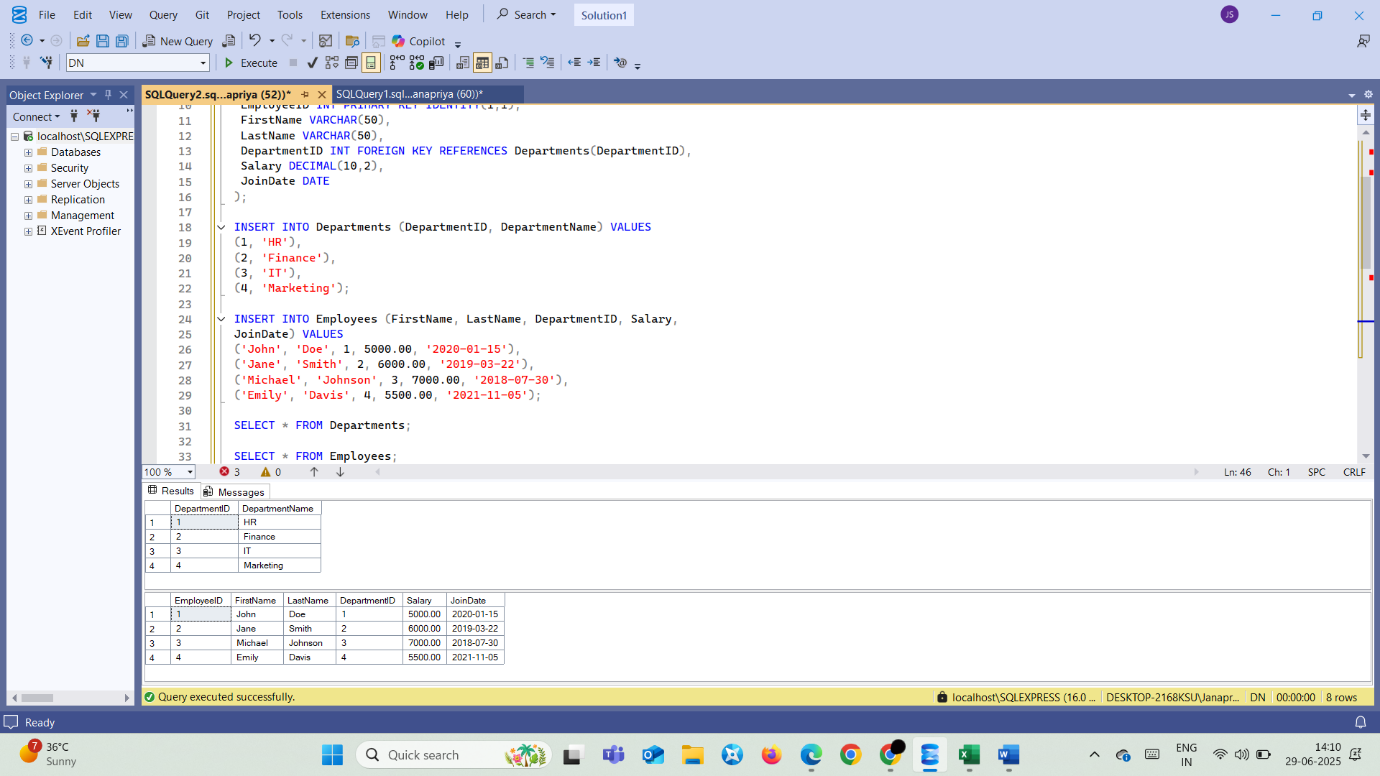
('Michael', 'Johnson', 3, 7000.00, '2018-07-30'),

('Emily', 'Davis', 4, 5500.00, '2021-11-05');

SELECT \* FROM Departments;

SELECT \* FROM Employees;

#### **Output:**



### **Create Procedure:**

#### **Code:**

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

BEGIN

INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary, JoinDate)

VALUES (@FirstName, @LastName, @DepartmentID, @Salary, @JoinDate);

END;

EXEC sp\_InsertEmployee

@FirstName = 'Lisa',

@LastName = 'Brown',

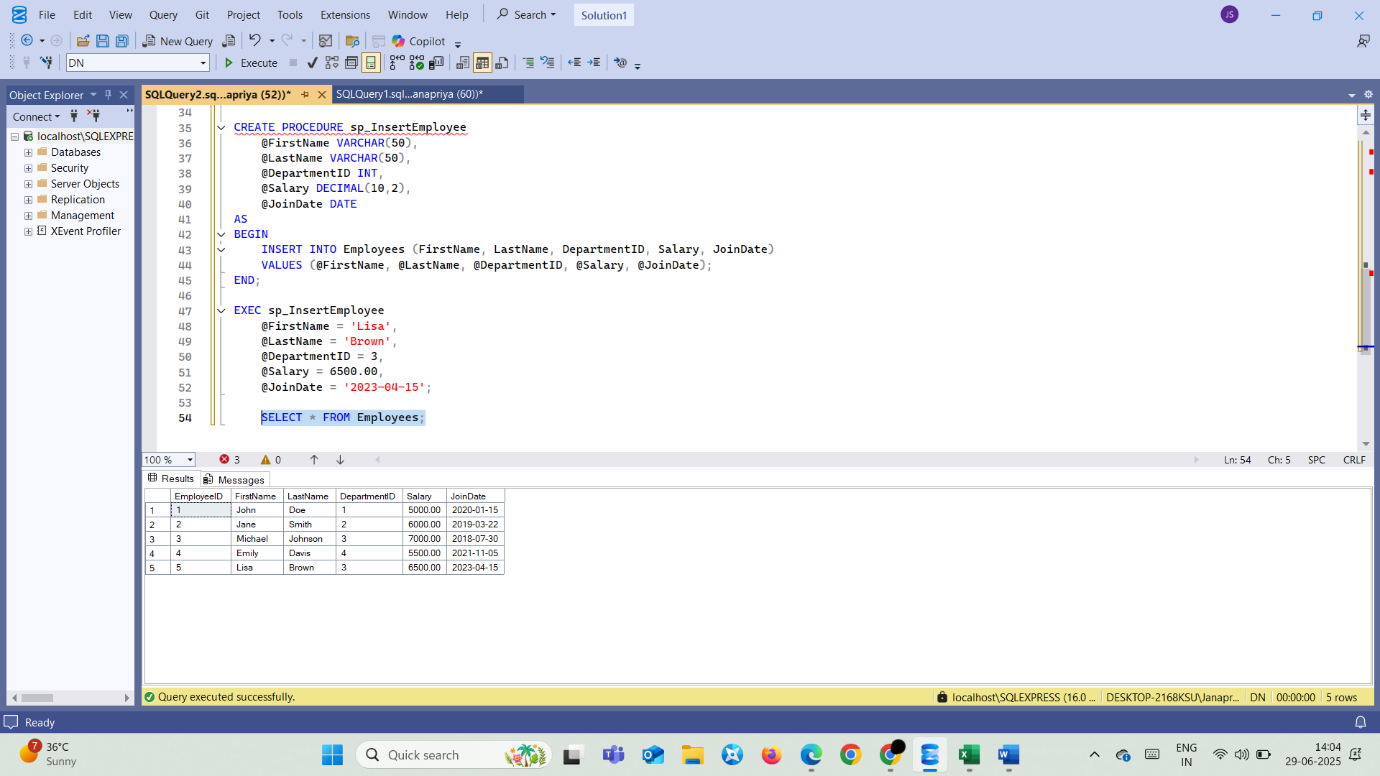
@DepartmentID = 3,

@Salary = 6500.00,

@JoinDate = '2023-04-15';

SELECT \* FROM Employees;

#### **Output:**



## **Exercise 5: Create a Stored Procedure**

Goal: Create a stored procedure that returns the total number of employees in a department

### **Procedure for returning the number of employees, given the department ID.**

#### **Code:**

CREATE PROCEDURE sp\_EmpCountByDept

@DepartmentID INT

AS

BEGIN

SELECT

COUNT(\*) AS TotalEmployees

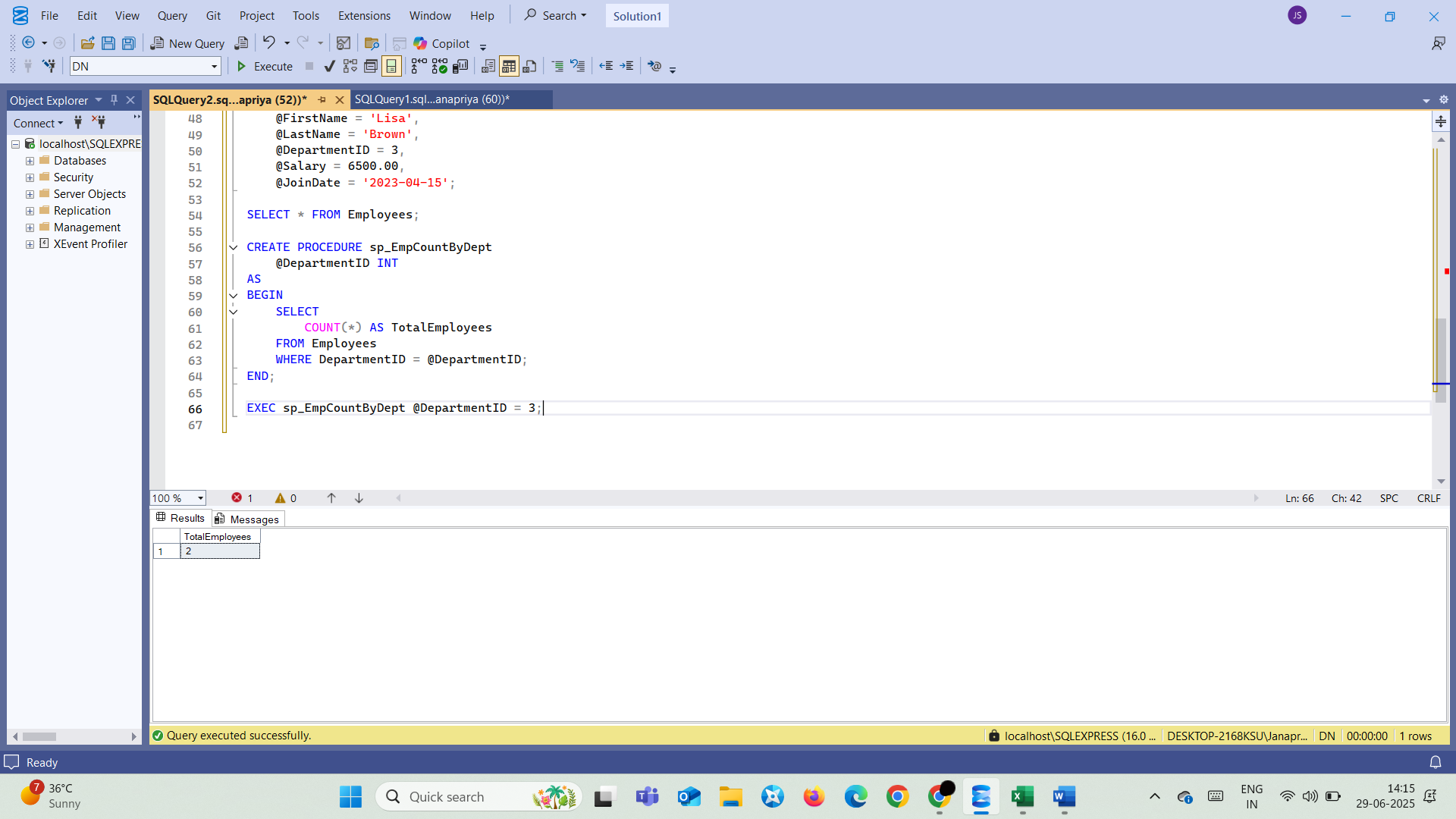
FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

EXEC sp\_EmpCountByDept @DepartmentID = 3;

#### **Output:**



## **Write NUnit test cases for Calculator Application**

### **Code:**

#### **MathLibrary.cs:**

using System;

namespace CalcLibrary

{

    interface IMathLibrary

    {

        double Addition(double a, double b);

        double Subtraction(double a, double b);

        double Multiplication(double a, double b);

        double Division(double a, double b);

    }

    public class SimpleCalculator : IMathLibrary

    {

        double result = 0;

        public double Addition(double a, double b)

        {

            result = a + b;

            return result;

        }

        public double Subtraction(double a, double b)

        {

            result = a - b;

            return result;

        }

        public double Multiplication(double a, double b)

        {

            result = a \* b;

            return result;

        }

        public double Division(double a, double b)

        {

            if (b == 0)

                throw new ArgumentException("Second Parameter Can't be Zero");

            result = a / b;

            return result;

        }

        public void AllClear()

        {

            result = 0;

        }

        public double GetResult

        {

            get { return result; }

        }

    }

}

#### **CalculatorTests.cs:**

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalcLibrary.Tests

{

    [TestFixture]

    public class SimpleCalculatorTests

    {

        private SimpleCalculator \_calc;

        [SetUp]

        public void Init()

        {

            \_calc = new SimpleCalculator();

        }

        [TearDown]

        public void Cleanup()

        {

            \_calc = null!;

        }

        [TestCase(2, 3, 5)]

        [TestCase(-1, 1, 0)]

        [TestCase(2.5, 3.5, 6.0)]

        [TestCase(-10.0, 5.0, -5.0)]

        public void Addition\_ReturnsExpectedResult(double a, double b, double exp)

        {

            Assert.That(\_calc.Addition(a, b), Is.EqualTo(exp).Within(0.0001));

        }

        [TestCase(5, 3, 2)]

        [TestCase(2, 5, -3)]

        [TestCase(7.5, 2.5, 5.0)]

        [TestCase(2.0, 7.0, -5.0)]

        public void Subtraction\_ReturnsExpectedResult(double a, double b, double exp)

        {

            Assert.That(\_calc.Subtraction(a, b), Is.EqualTo(exp).Within(0.0001));

        }

        [TestCase(2, 3, 6)]

        [TestCase(-2, 4, -8)]

        [TestCase(2.5, 4.0, 10.0)]

        [TestCase(-3.0, -2.0, 6.0)]

        public void Multiplication\_ReturnsExpectedResult(double a, double b, double exp)

        {

            Assert.That(\_calc.Multiplication(a, b), Is.EqualTo(exp).Within(0.0001));

        }

        [TestCase(6, 2, 3.0)]

        [TestCase(7, 2, 3.5)]

        [TestCase(10.0, 4.0, 2.5)]

        [TestCase(-9.0, 3.0, -3.0)]

        public void Division\_ReturnsExpectedResult(double a, double b, double exp)

        {

            Assert.That(\_calc.Division(a, b), Is.EqualTo(exp).Within(0.0001));

        }

        [Test]

        public void Division\_ByZero\_ThrowsArgumentException()

        {

            Assert.Throws<ArgumentException>(() => \_calc.Division(5, 0));

        }

        [Test]

        public void AllClear\_ResetsResultToZero()

        {

            \_calc.Addition(10, 5);

            Assert.That(\_calc.GetResult, Is.EqualTo(15));

            \_calc.AllClear();

            Assert.That(\_calc.GetResult, Is.EqualTo(0));

        }

        [Test]

        public void GetResult\_ReturnsCurrentResult()

        {

            \_calc.Addition(7, 3);

            Assert.That(\_calc.GetResult, Is.EqualTo(10));

            \_calc.Subtraction(20, 5);

            Assert.That(\_calc.GetResult, Is.EqualTo(15));

        }

        [Test, Ignore("Demo Test (not implemented yet)")]

        public void UnimplementedTest()

        {

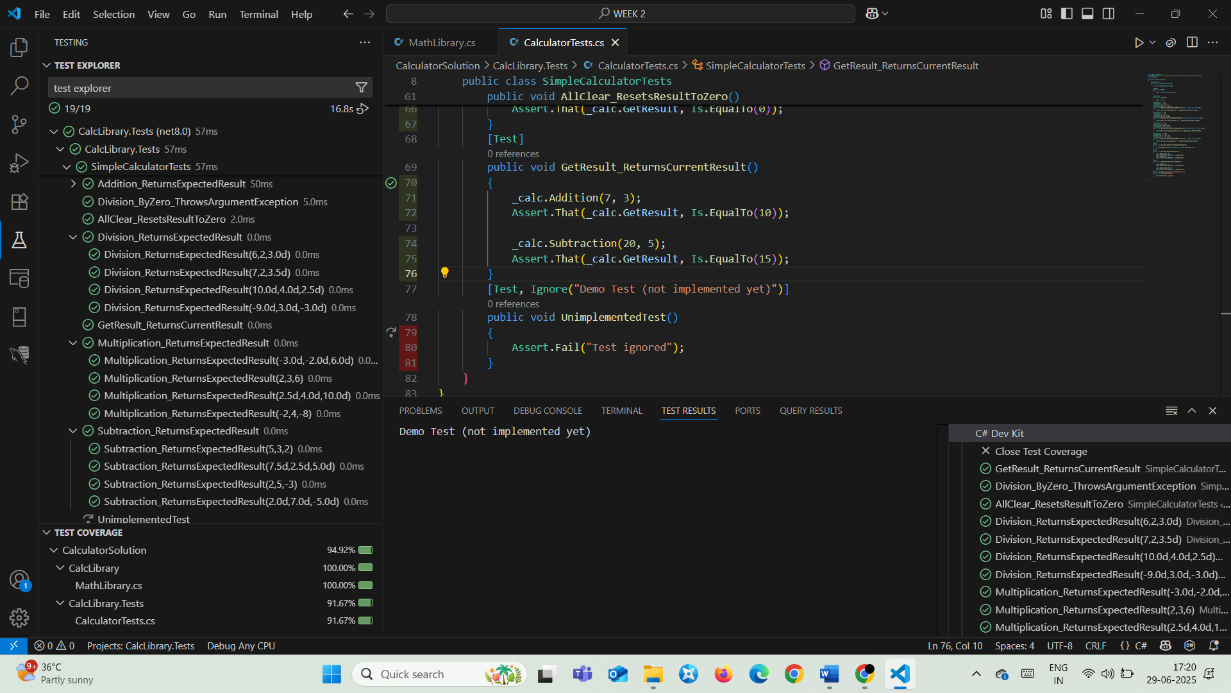
            Assert.Fail("Test ignored");

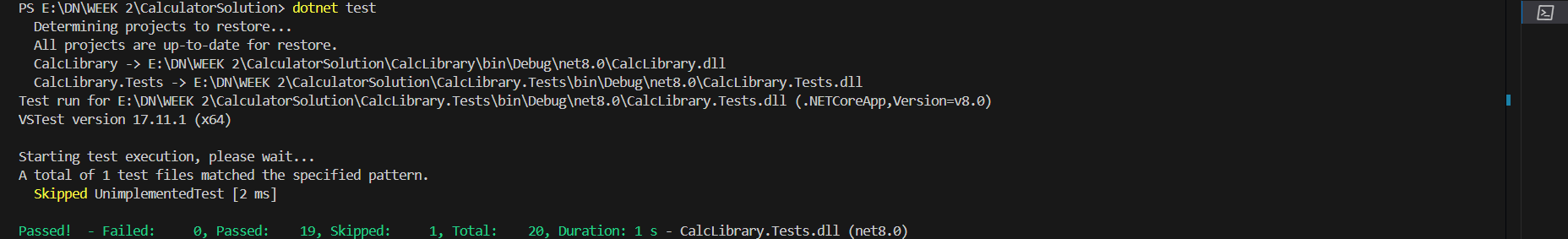
        }

    }

}

### **Output:**



****

## **Write Testable Code with Moq**

### **Code:**

#### **Task 1:**

**MailSender.cs:**

using System.Net;

using System.Net.Mail;

namespace CustomerCommLib

{

    public interface IMailSender

    {

        bool SendMail(string toAddress, string message);

    }

    public class MailSender : IMailSender

    {

        public bool SendMail(string toAddress, string message)

        {

            MailMessage mail = new MailMessage();

            SmtpClient smtpServer = new SmtpClient("smtp.gmail.com");

            mail.From = new MailAddress("your\_email@gmail.com");

            mail.To.Add(toAddress);

            mail.Subject = "Test Mail";

            mail.Body = message;

            smtpServer.Port = 587;

            smtpServer.Credentials = new NetworkCredential("username", "password");

            smtpServer.EnableSsl = true;

            smtpServer.Send(mail);

            return true;

        }

    }

}

**CustomerComm.cs:**

namespace CustomerCommLib

{

    public class CustomerComm

    {

        private readonly IMailSender \_mailSender;

        public CustomerComm(IMailSender mailSender)

        {

            \_mailSender = mailSender;

        }

        public bool SendMailToCustomer()

        {

            return \_mailSender.SendMail("cust123@abc.com", "Some Message");

        }

    }

}

#### **Task 2:**

**CustomerCommTests.cs:**

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerComm.Tests

{

    [TestFixture]

    public class CustomerCommTests

    {

        private Mock<IMailSender> \_mockMailSender = null!;

        private CustomerCommLib.CustomerComm \_customerComm = null!;

        [OneTimeSetUp]

        public void Setup()

        {

            \_mockMailSender = new Mock<IMailSender>();

            \_mockMailSender.Setup(m => m.SendMail(It.IsAny<string>(), It.IsAny<string>())).Returns(true);

            \_customerComm = new CustomerCommLib.CustomerComm(\_mockMailSender.Object);

        }

        [Test]

        public void SendMailToCustomer\_ReturnsTrue()

        {

            var result = \_customerComm.SendMailToCustomer();

              System.Console.WriteLine("Type: " + typeof(NUnit.Framework.Assert).FullName);

            Assert.That(result, Is.True);

        }

    }

}

### **Output:**

