**Advanced SQL & NUnit and Moq**

**Name: Akshita Pattnaik**

**SUPERSET ID: 6360271**

**Question 1:**

**Exercise 1: Ranking and Window Functions**

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.

Steps:

1. Use ROW\_NUMBER() to assign a unique rank within each category.

2. Use RANK() and DENSE\_RANK() to compare how ties are handled.

3. Use PARTITION BY Category and ORDER BY Price DESC

**CODE:**

CREATE DATABASE ProductDB;

GO

USE ProductDB;

GO

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10,2)

);

GO

INSERT INTO Products (ProductID, ProductName, Category, Price) VALUES

(1, 'iPhone 14', 'Electronics', 999.99),

(2, 'Samsung TV', 'Electronics', 750.00),

(3, 'Sony Headphones', 'Electronics', 199.99),

(4, 'Laptop Dell', 'Electronics', 800.00),

(5, 'T-Shirt', 'Clothing', 19.99),

(6, 'Jacket', 'Clothing', 89.99),

(7, 'Jeans', 'Clothing', 49.99),

(8, 'Shoes', 'Clothing', 89.99),

(9, 'Oven', 'Appliances', 149.99),

(10, 'Washing Machine', 'Appliances', 499.99),

(11, 'Fridge', 'Appliances', 599.99),

(12, 'Toaster', 'Appliances', 49.99);

SELECT

Category,

ProductName,

Price,

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

RANK() OVER(PARTITION BY Category ORDER BY Price DESC) AS PriceRank,

DENSE\_RANK() OVER(PARTITION BY Category ORDER BY Price DESC) AS DensePriceRank

FROM Products;

SELECT \*

FROM (

SELECT

Category,

ProductName,

Price,

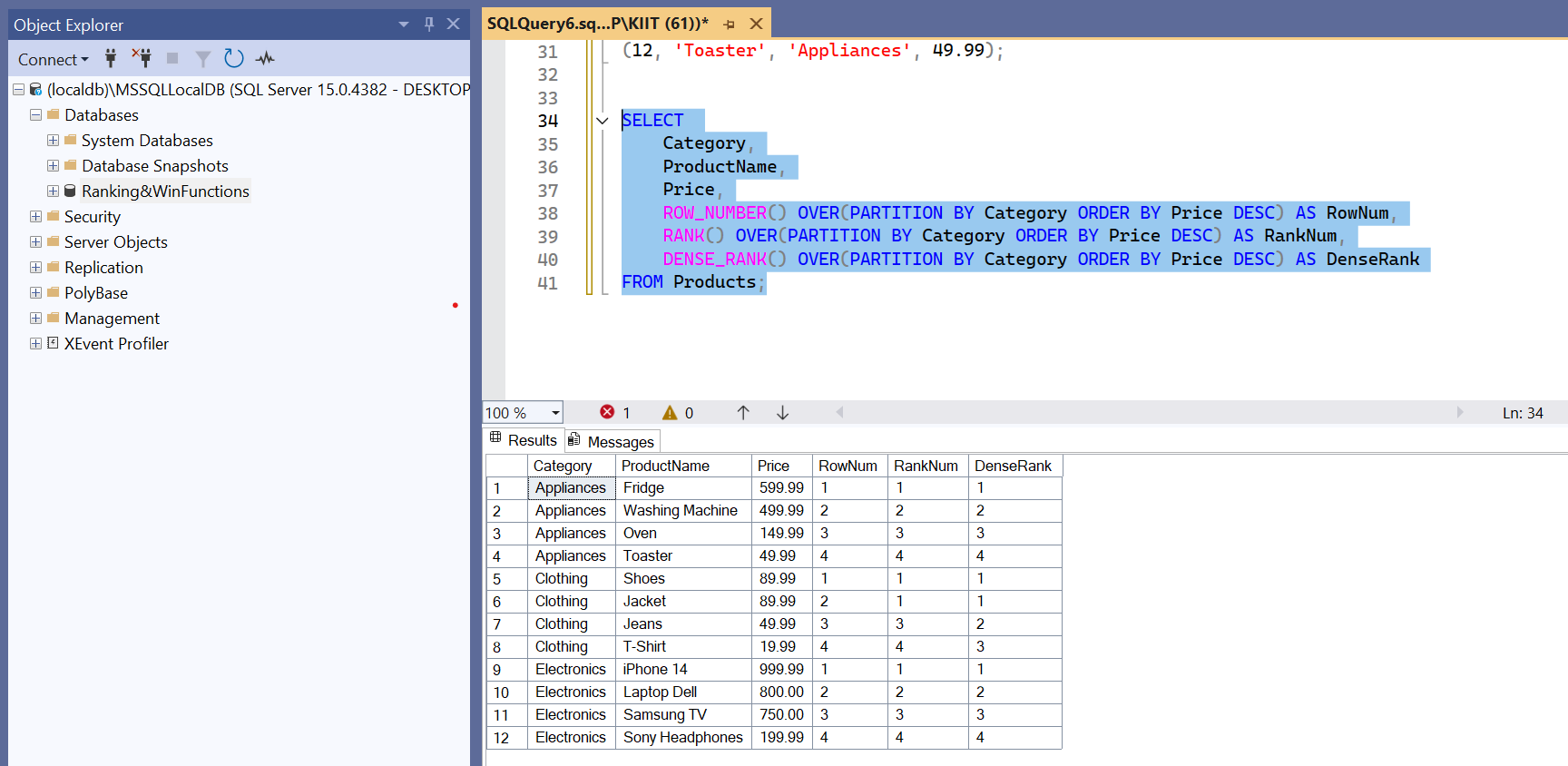
ROW\_NUMBER() OVER(PARTITION BY Category ORDER BY Price DESC) AS RowNum

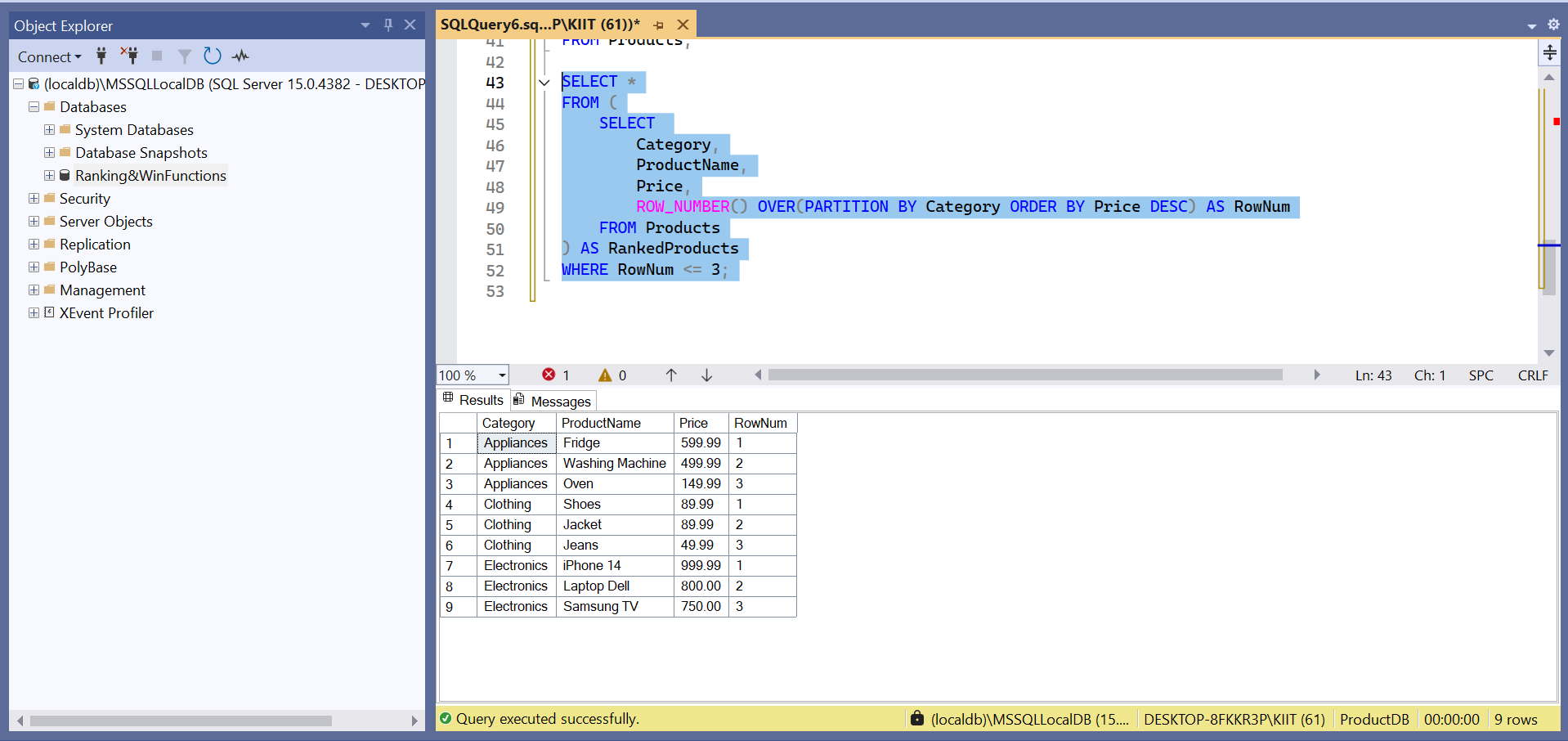
FROM Products

) AS RankedProducts

WHERE RowNum <= 3;

**OUTPUT:**





**Question 2:**

**Exercise 1: Create a Stored Procedure**

Goal: Create a stored procedure to retrieve employee details by department.

Steps:

1. Define the stored procedure with a parameter for DepartmentID.

2. Write the SQL query to select employee details based on the DepartmentID.

3. Create a stored procedure named `sp\_InsertEmployee` with the following 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;

**CODE:**

CREATE DATABASE EmployeeDB;

GO

USE EmployeeDB;

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');

GO

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

BEGIN

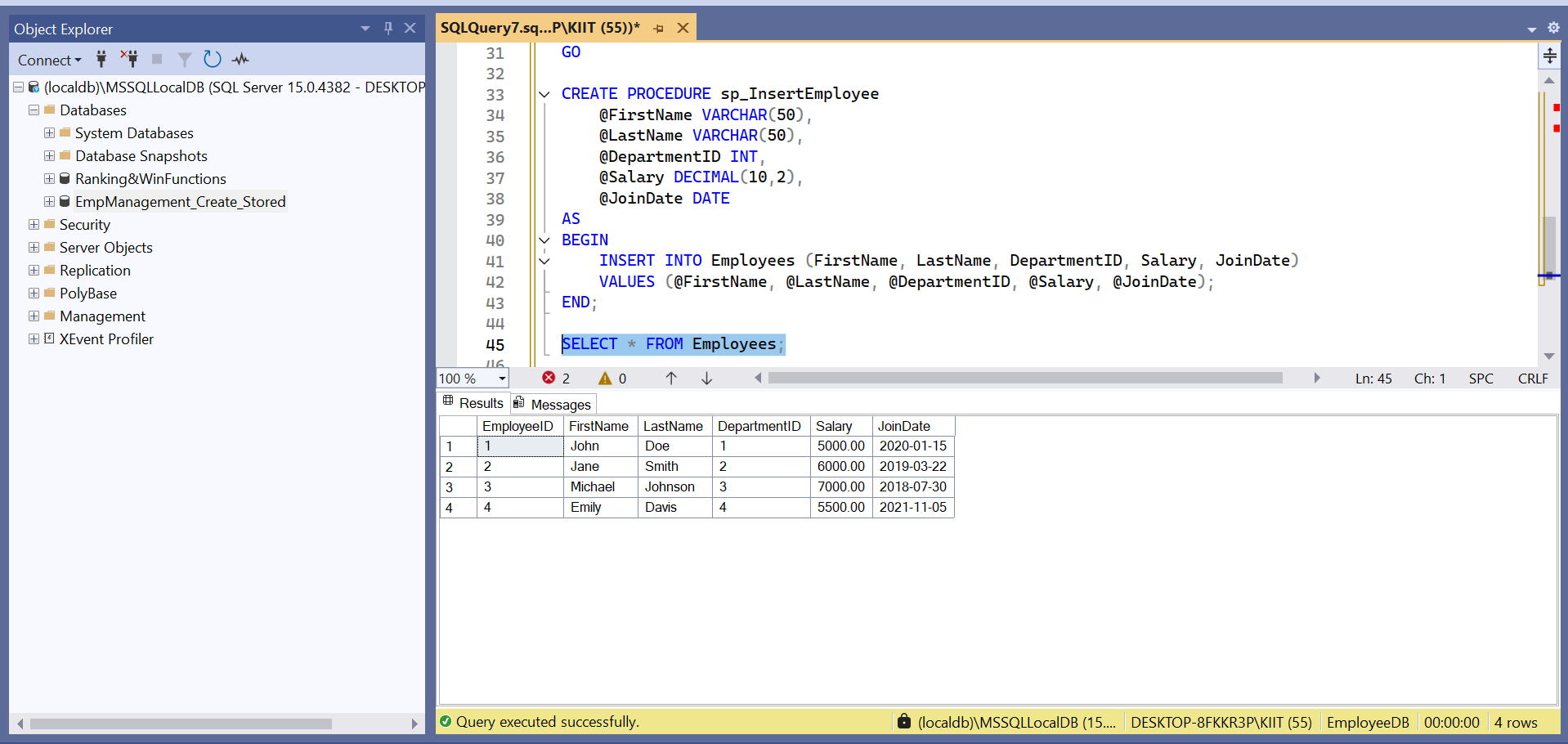
INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary, JoinDate)

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

END;

SELECT \* FROM Employees;

**OUTPUT:**



**Question 3:**

**Exercise 5: Return Data from a Stored Procedure**

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

department.

Steps:

1. Define the stored procedure with a parameter for DepartmentID.

2. Write the SQL query to count the number of employees in the specified department.

3. Save the stored procedure by executing the Stored procedure content.

**CODE:**

USE EmployeeDB;

GO

IF OBJECT\_ID('sp\_GetEmployeesByDepartment', 'P') IS NOT NULL

DROP PROCEDURE sp\_GetEmployeesByDepartment;

GO

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DeptID INT

AS

BEGIN

SELECT

E.EmployeeID,

E.FirstName,

E.LastName,

E.Salary,

E.JoinDate,

D.DepartmentName

FROM Employees E

INNER JOIN Departments D ON E.DepartmentID = D.DepartmentID

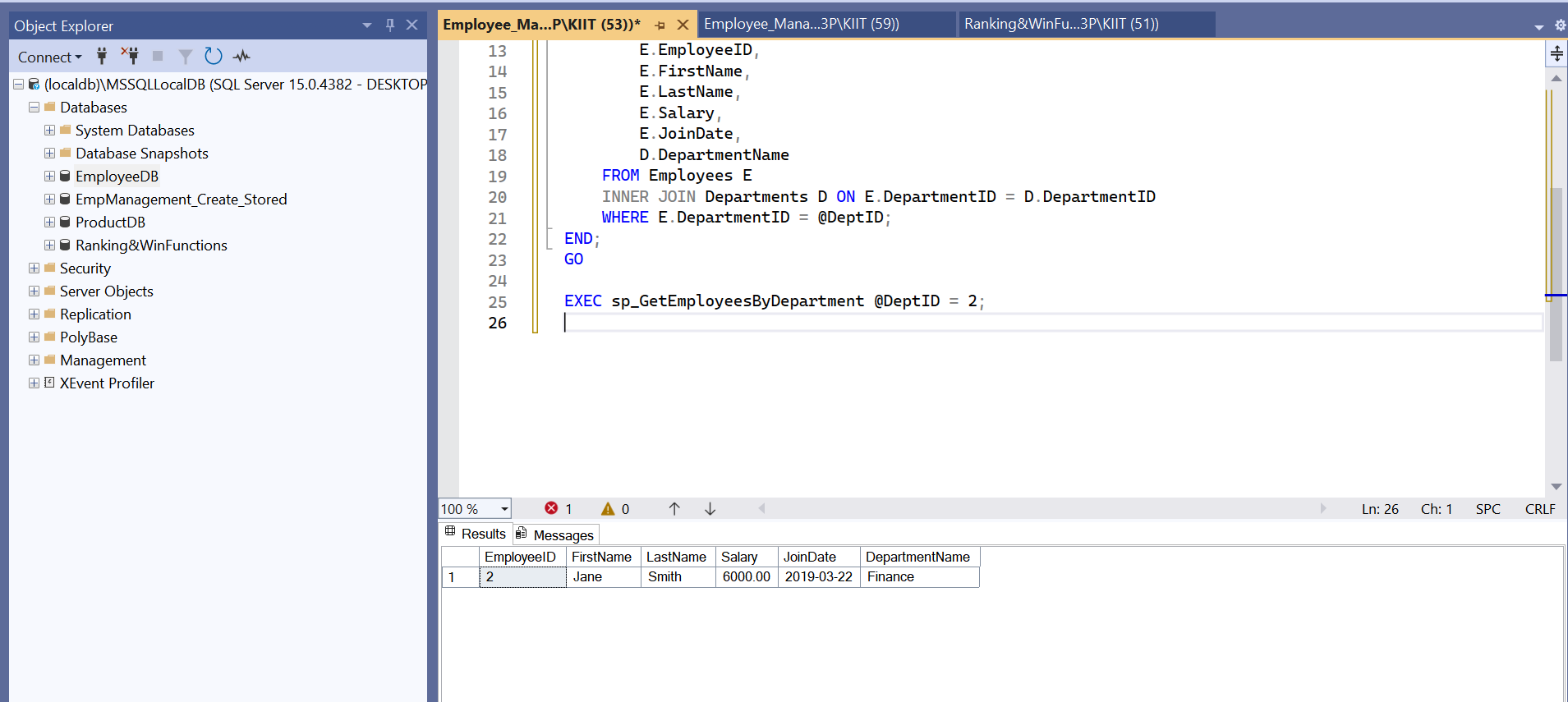
WHERE E.DepartmentID = @DeptID;

END;

GO

EXEC sp\_GetEmployeesByDepartment @DeptID = 2;

**OUTPUT:**



**Question 4:**

**1. NUnit-Handson**

**TestFixture & Test**  
Follow the steps listed below to write the NUnit test cases for the application.

* Create a Unit test project(.Net Framework) in the solution provided.
* Add the CalcLibrary project as reference
* Create a class “CalculatorTests” to write all the test cases for the methods in the solution
* Use the ‘TestFixture’, ‘SetUp’ and ‘TearDown’ attributes, to declare, initialize and cleanup activities respectively
* Create a Test method to check the addition functionality
* Use the ‘TestCase’ attribute to send the inputs and the expected result
* Use Assert.That to check the actual and expected result match

**CalculatorTests.cs:**

using NUnit.Framework;

using CalcLibrary;

namespace CalcLibrary.Tests

{

[TestFixture]

public class CalculatorTests

{

private SimpleCalculator \_calculator;

[SetUp]

public void Setup()

{

\_calculator = new SimpleCalculator();

}

[TearDown]

public void TearDown()

{

\_calculator.AllClear();

}

[Test]

[TestCase(5, 3, 8)]

[TestCase(-1, -4, -5)]

[TestCase(0, 0, 0)]

public void Addition\_ReturnsCorrectResult(double a, double b, double expected)

{

double actual = \_calculator.Addition(a, b);

Assert.That(actual, Is.EqualTo(expected));

}

[Test]

[Ignore("This is an ignored test demo")]

public void IgnoredTest\_Demo()

{

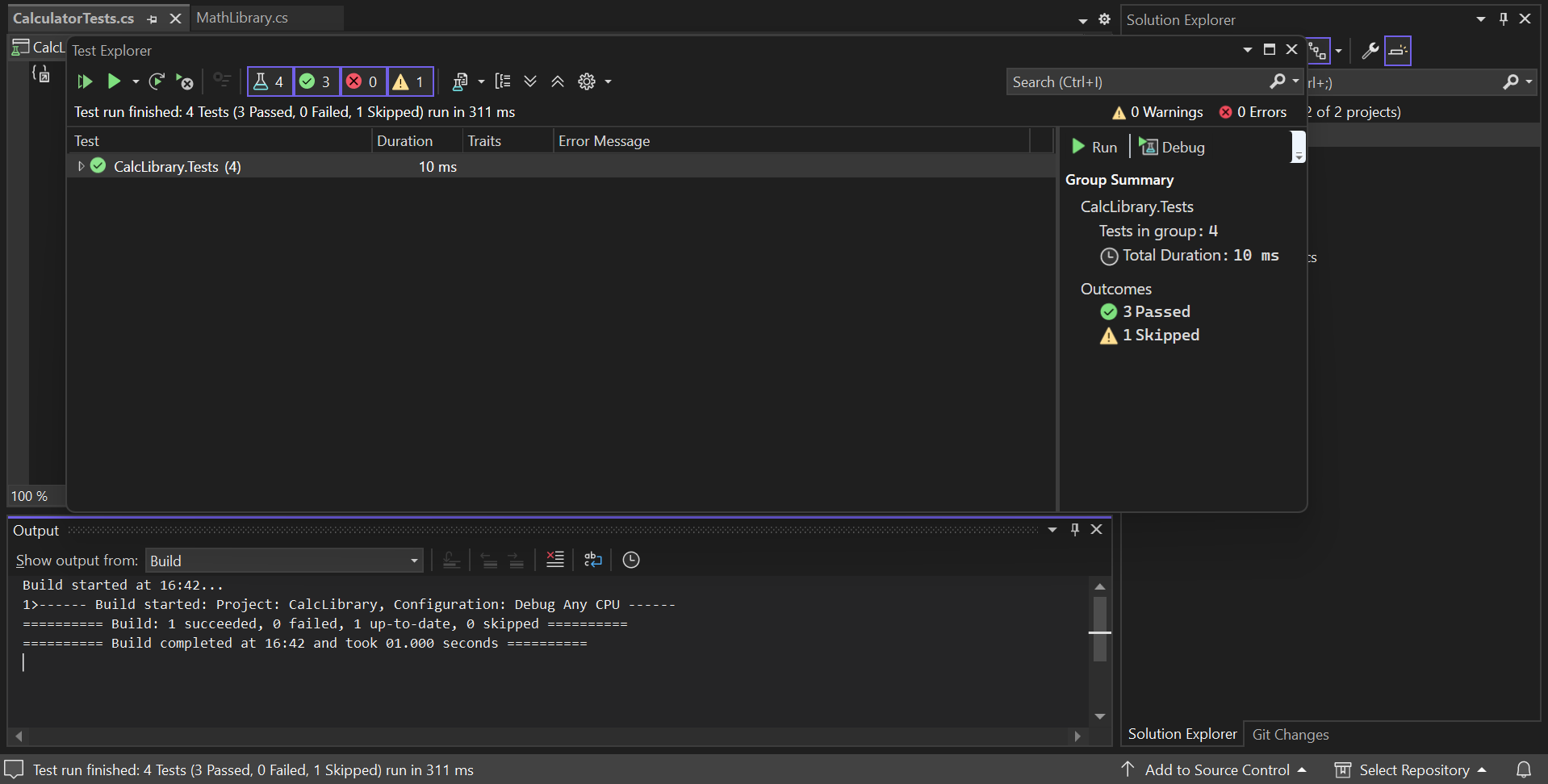
Assert.Fail("This should be ignored.");

}

}

}

**Output:**



**Question 5:**

1. **Moq-Handson:**
2. **Write Testable Code with Moq**

**CODE:**

**CustomerCommTests.cs:**

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerComm.Tests

{

[TestFixture]

public class CustomerCommTests

{

private Mock<IMailSender> \_mockMailSender;

private CustomerCommLib.CustomerComm \_customerComm;

[OneTimeSetUp]

public void Init()

{

\_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();

Assert.IsTrue(result);

}

}

}

**CustomerComm.cs:**

namespace CustomerCommLib

{

public class CustomerComm

{

IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

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

return true;

}

}

}

**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\_address@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;

}

}

}

**OUTPUT:**

