Cognizant Week2 Handson:

Problem -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.

Sol:

CREATE DATABASE ProductRankingDemo;

USE ProductRankingDemo;

CREATE TABLE Products (

    ProductID INT PRIMARY KEY,

    ProductName VARCHAR(100),

    Category VARCHAR(50),

    Price DECIMAL(10, 2)

);

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

(1, 'Laptop Pro', 'Electronics', 1200.00),

(2, 'Smartphone X', 'Electronics', 999.99),

(3, 'Wireless Earbuds', 'Electronics', 199.99),

(4, 'Ultra HD TV', 'Electronics', 899.00),

(5, 'Bluetooth Speaker', 'Electronics', 149.99),

(6, 'Cotton T-Shirt', 'Clothing', 29.99),

(7, 'Denim Jeans', 'Clothing', 59.99),

(8, 'Winter Jacket', 'Clothing', 199.99),

(9, 'Running Shoes', 'Clothing', 89.99),

(10, 'Formal Shirt', 'Clothing', 49.99),

(11, 'Coffee Maker', 'Home', 79.99),

(12, 'Air Fryer', 'Home', 129.99),

(13, 'Blender', 'Home', 49.99),

(14, 'Toaster', 'Home', 39.99),

(15, 'Microwave', 'Home', 149.99),

(16, 'Novel - The Journey', 'Books', 14.99),

(17, 'Cookbook', 'Books', 24.99),

(18, 'Sci-Fi Trilogy', 'Books', 39.99),

(19, 'Biography', 'Books', 19.99),

(20, 'Children''s Book', 'Books', 12.99);

SELECT \* FROM Products;

-- Using ROW\_NUMBER() - assigns unique ranks within each category, even with ties

SELECT

    Category,

    ProductName,

    Price,

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

FROM

    Products

WHERE

    ROW\_NUMBER() OVER (PARTITION BY Category ORDER BY Price DESC) <= 3

ORDER BY

    Category, RowNumRank;

-- Note: The above query will fail because window functions can't be used in WHERE directly

-- Here's the correct way using a CTE:

WITH RankedProducts AS (

    SELECT

        Category,

        ProductName,

        Price,

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

    FROM

        Products

)

SELECT

    Category,

    ProductName,

    Price,

    RowNumRank

FROM

    RankedProducts

WHERE

    RowNumRank <= 3

ORDER BY

    Category, RowNumRank;

-- Using RANK() - handles ties by giving same rank, with gaps in ranking

WITH RankedProducts AS (

    SELECT

        Category,

        ProductName,

        Price,

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

    FROM

        Products

)

SELECT

    Category,

    ProductName,

    Price,

    Rank

FROM

    RankedProducts

WHERE

    Rank <= 3

ORDER BY

    Category, Rank;

-- Using DENSE\_RANK() - handles ties by giving same rank, with no gaps in ranking

WITH RankedProducts AS (

    SELECT

        Category,

        ProductName,

        Price,

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

    FROM

        Products

)

SELECT

    Category,

    ProductName,

    Price,

    DenseRank

FROM

    RankedProducts

WHERE

    DenseRank <= 3

ORDER BY

    Category, DenseRank;

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 Rank,

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

FROM

    Products

ORDER BY

    Category, Price DESC;

-- Add some products with identical prices

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

(21, 'Tablet A', 'Electronics', 899.00),

(22, 'Designer Jeans', 'Clothing', 59.99),

(23, 'Espresso Machine', 'Home', 129.99);

SELECT \* from Products;

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 Rank,

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

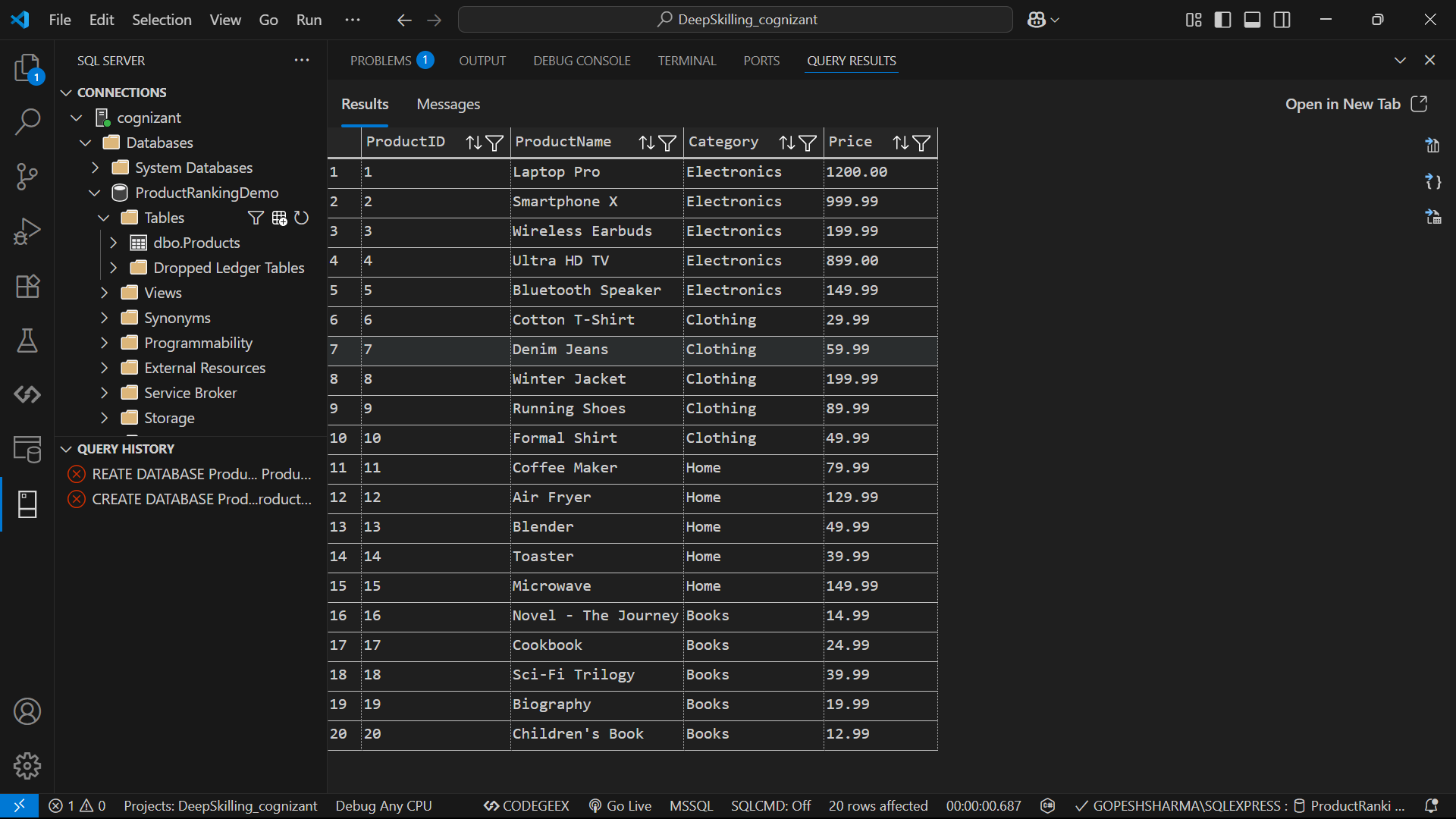
FROM

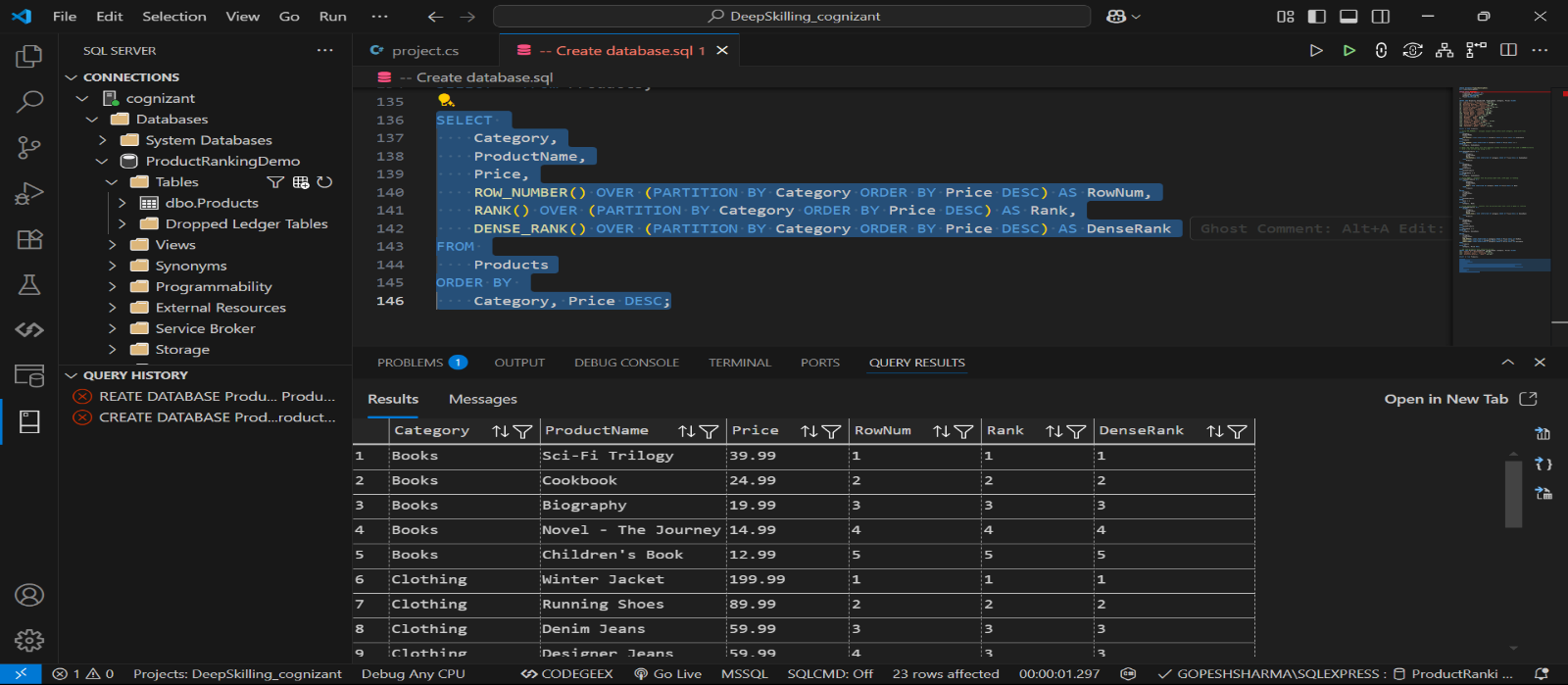
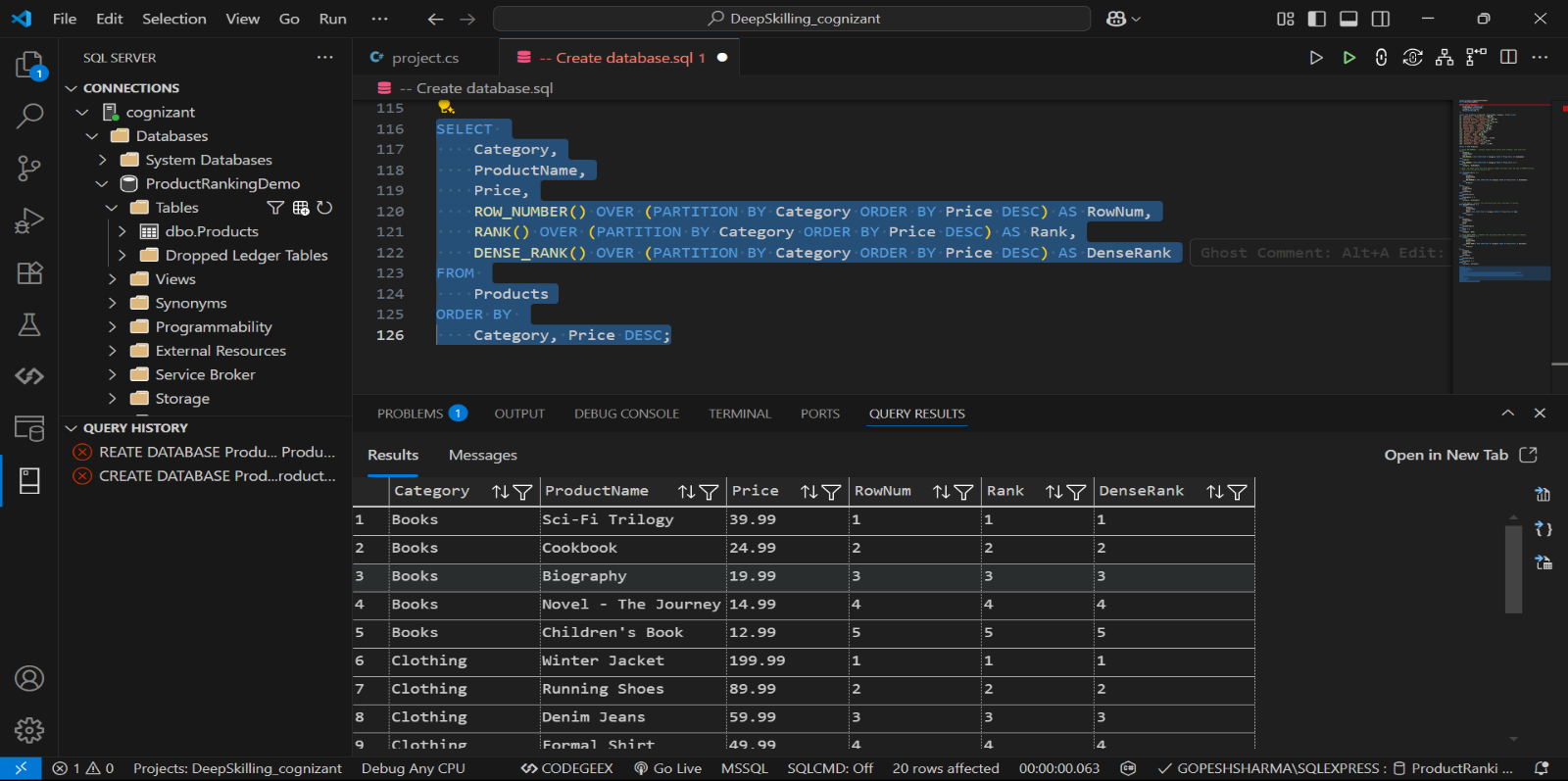
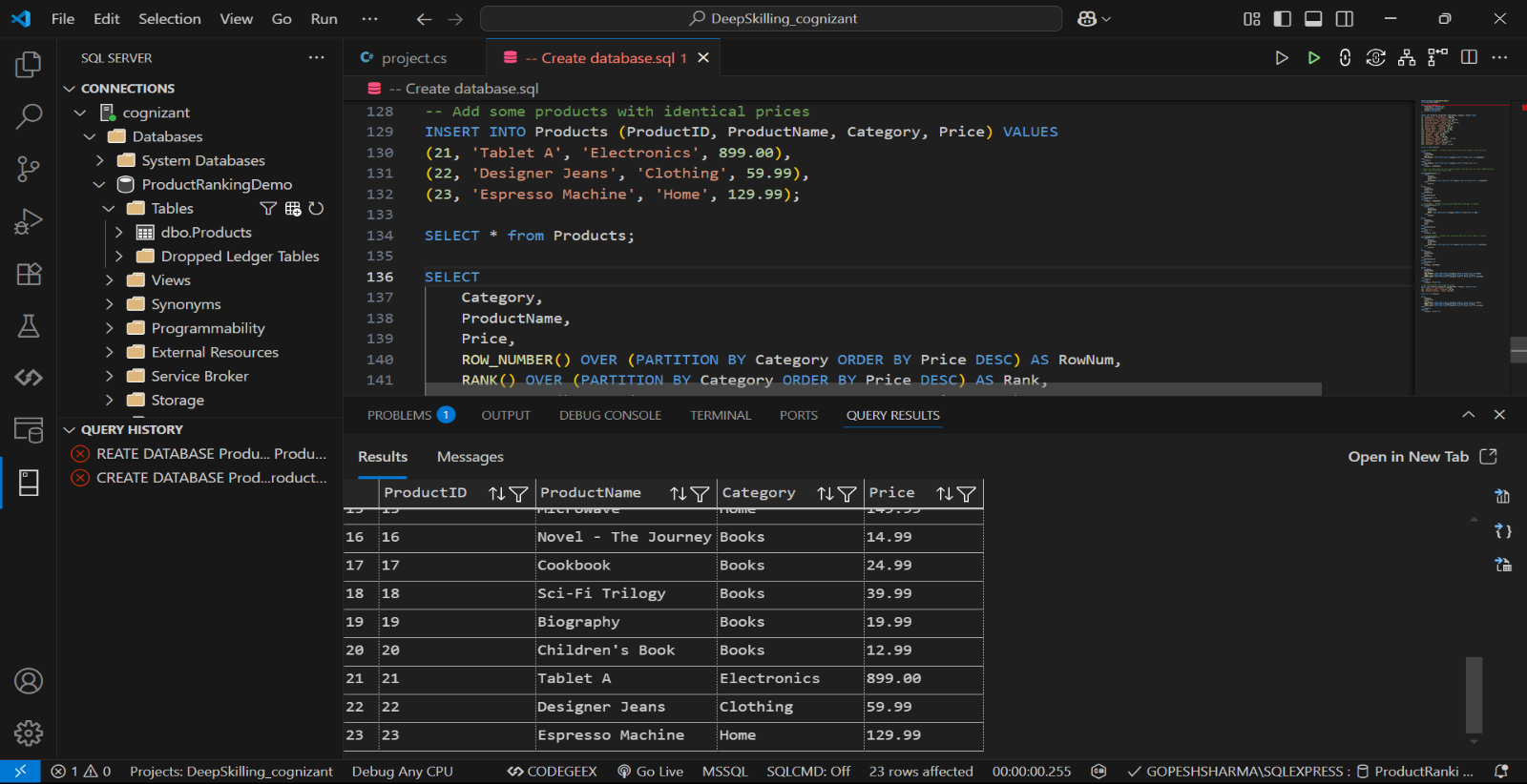
    Products

ORDER BY

    Category, Price DESC;

Output:





Problem – 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;

Sol:

CREATE DATABASE EmployeeManagement;

USE EmployeeManagement;

CREATE TABLE Departments (

    DepartmentID INT PRIMARY KEY,

    DepartmentName VARCHAR(50) NOT NULL

);

CREATE TABLE Employees (

    EmployeeID INT IDENTITY(1,1) PRIMARY KEY,

    FirstName VARCHAR(50) NOT NULL,

    LastName VARCHAR(50) NOT NULL,

    DepartmentID INT FOREIGN KEY REFERENCES Departments(DepartmentID),

    Salary DECIMAL(10,2),

    JoinDate DATE

);

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'Human Resources'),

(2, 'Information Technology'),

(3, 'Finance'),

(4, 'Marketing'),

(5, 'Operations');

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

('John', 'Smith', 2, 75000.00, '2020-01-15'),

('Sarah', 'Johnson', 1, 65000.00, '2019-05-20'),

('Michael', 'Williams', 2, 82000.00, '2021-03-10'),

('Emily', 'Brown', 3, 68000.00, '2020-11-05'),

('David', 'Jones', 4, 72000.00, '2018-07-22'),

('Jessica', 'Garcia', 2, 78000.00, '2022-02-18'),

('Robert', 'Miller', 5, 69000.00, '2019-09-30'),

('Jennifer', 'Davis', 1, 63000.00, '2021-01-12'),

('Thomas', 'Rodriguez', 3, 71000.00, '2020-08-25'),

('Lisa', 'Martinez', 4, 74000.00, '2018-12-05');

SELECT \* FROM Departments;

SELECT \* FROM Employees;

GO

CREATE PROCEDURE sp\_GetEmployeesByDepartment

    @DepartmentID INT

AS

BEGIN

    SELECT

        e.EmployeeID,

        e.FirstName,

        e.LastName,

        d.DepartmentName,

        e.Salary,

        e.JoinDate,

        DATEDIFF(YEAR, e.JoinDate, GETDATE()) AS YearsOfService

    FROM

        Employees e

    INNER JOIN

        Departments d ON e.DepartmentID = d.DepartmentID

    WHERE

        e.DepartmentID = @DepartmentID

    ORDER BY

        e.LastName, e.FirstName;

END;

GO

CREATE PROCEDURE sp\_InsertEmployee

    @FirstName VARCHAR(50),

    @LastName VARCHAR(50),

    @DepartmentID INT,

    @Salary DECIMAL(10,2),

    @JoinDate DATE

AS

BEGIN

    IF NOT EXISTS (SELECT 1 FROM Departments WHERE DepartmentID = @DepartmentID)

    BEGIN

        RAISERROR('Invalid DepartmentID. Department does not exist.', 16, 1)

        RETURN

    END

    IF @Salary <= 0

    BEGIN

        RAISERROR('Salary must be a positive value.', 16, 1)

        RETURN

    END

    IF @JoinDate > GETDATE()

    BEGIN

        RAISERROR('Join date cannot be in the future.', 16, 1)

        RETURN

    END

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

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

    SELECT SCOPE\_IDENTITY() AS NewEmployeeID;

END;

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 2;

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 1;

EXEC sp\_InsertEmployee

    @FirstName = 'Daniel',

    @LastName = 'Wilson',

    @DepartmentID = 3,

    @Salary = 67000.00,

    @JoinDate = '2023-04-15';

EXEC sp\_InsertEmployee

    @FirstName = 'Invalid',

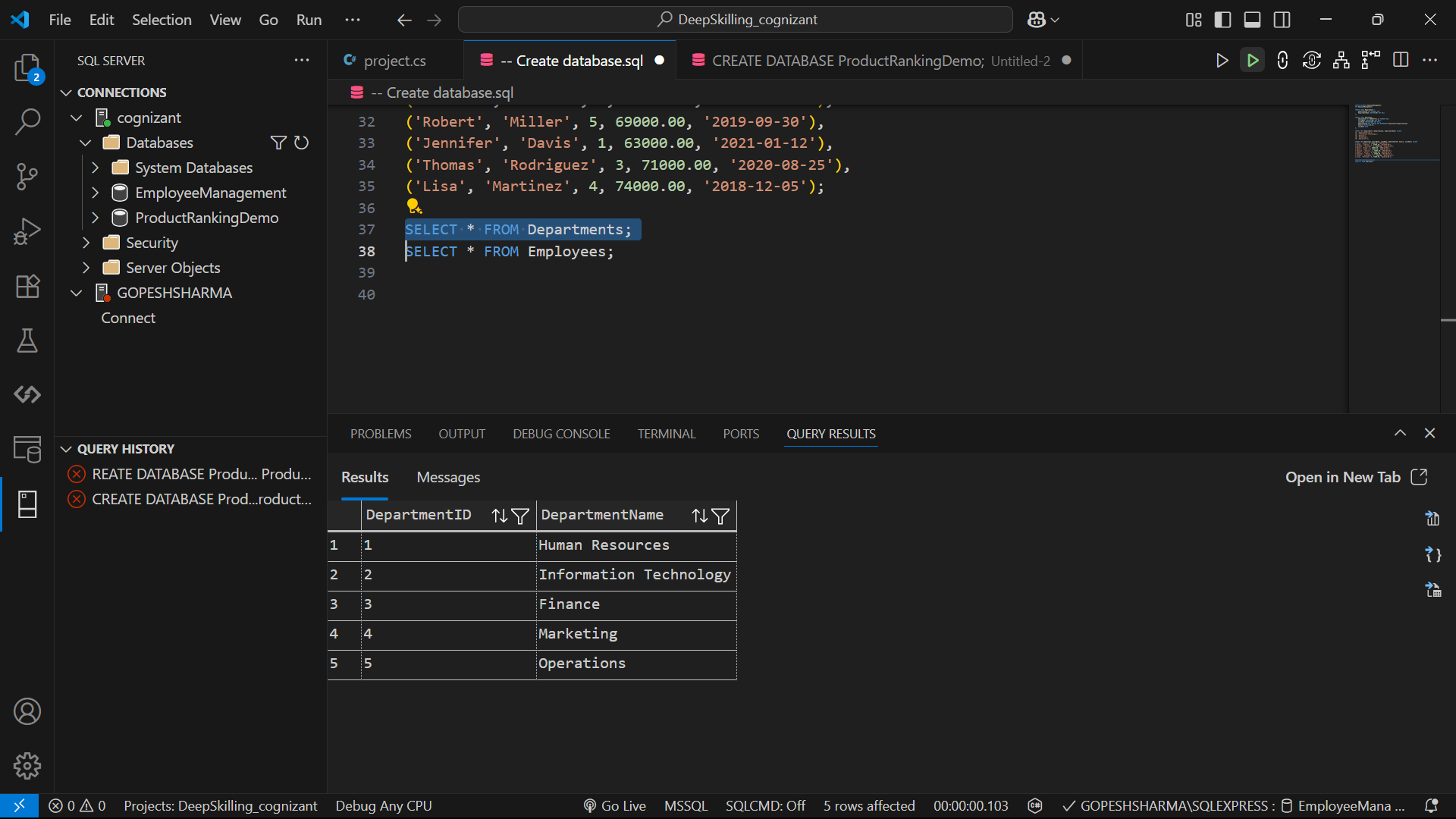
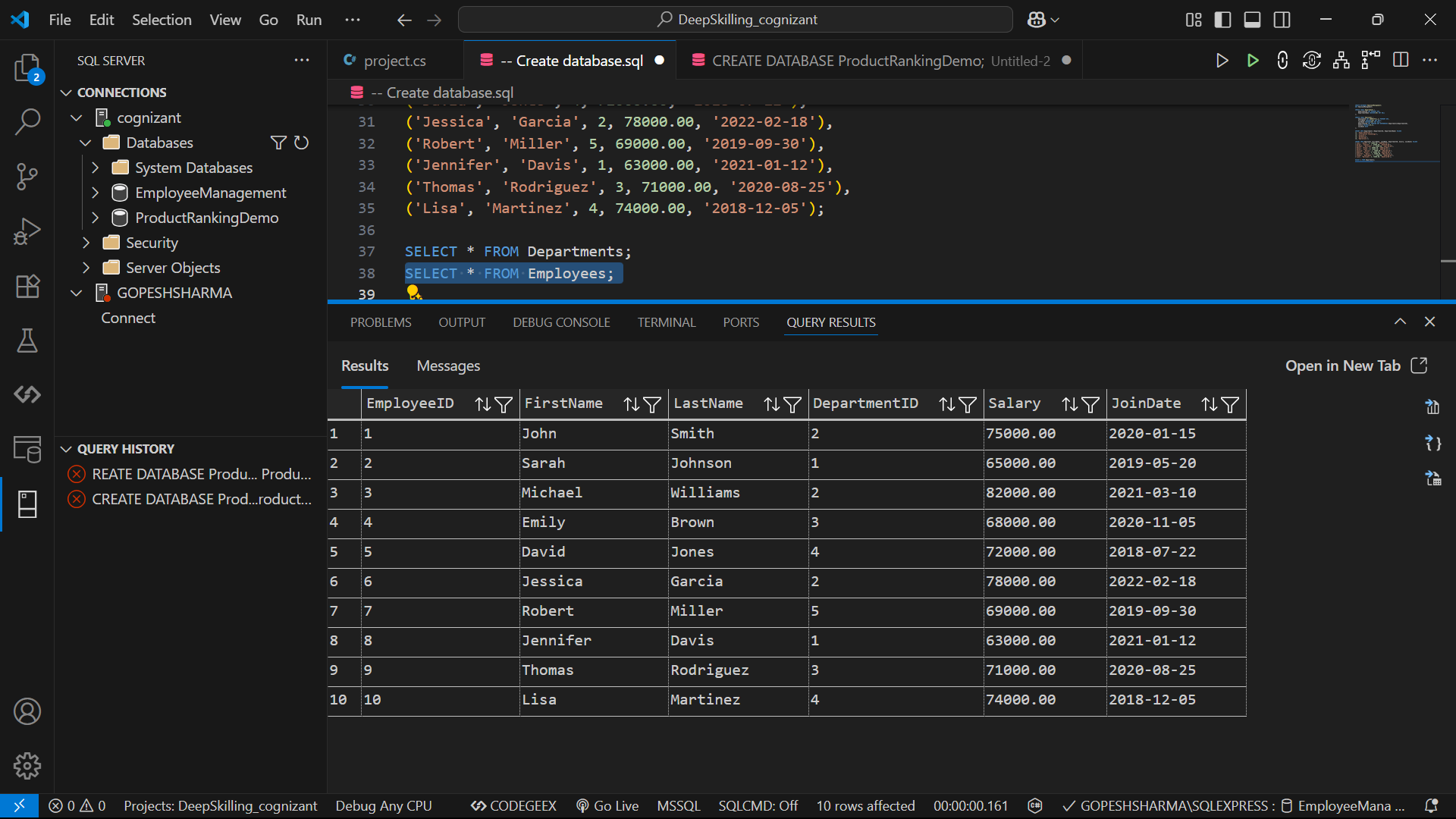
    @LastName = 'Employee',

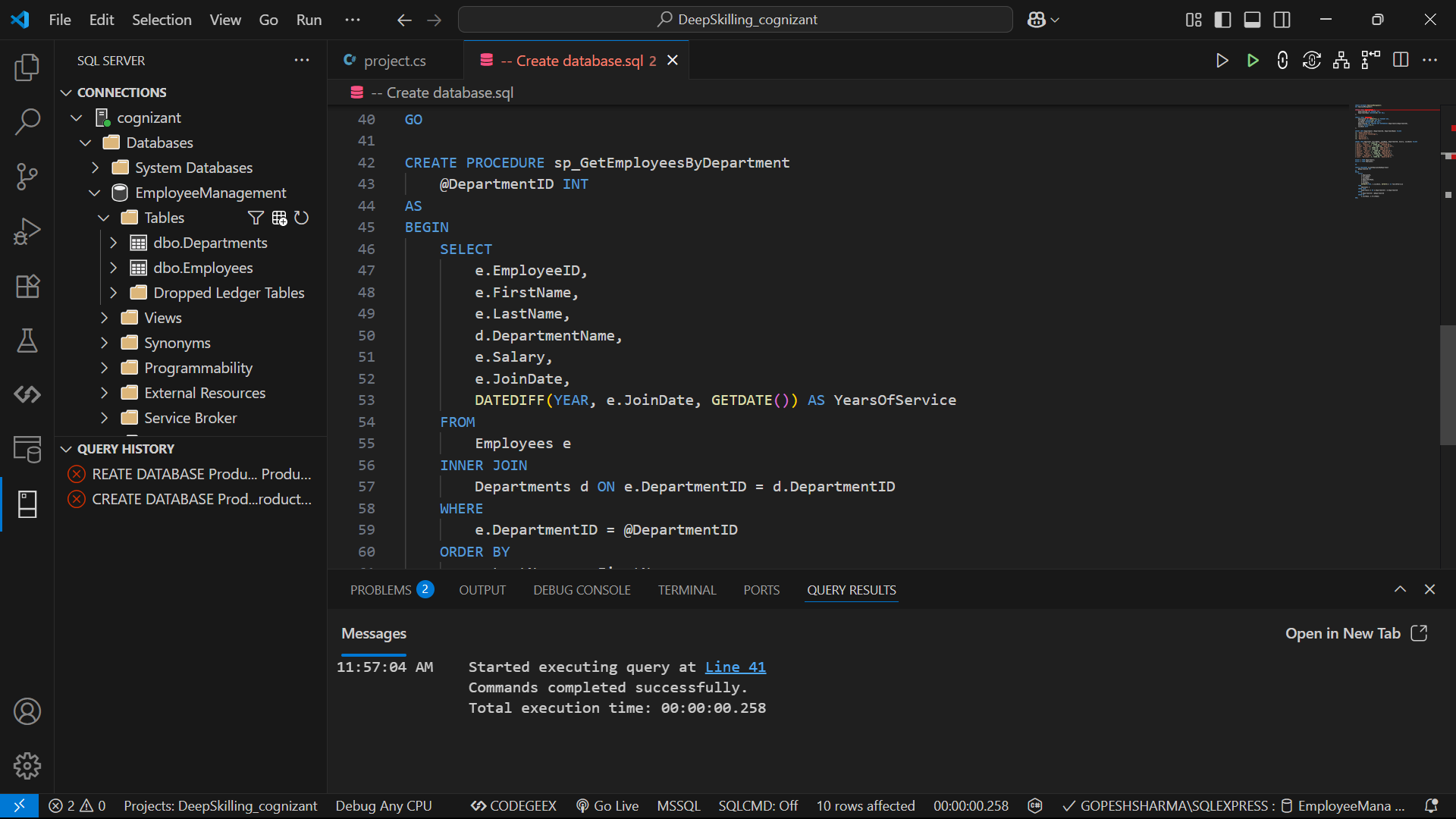
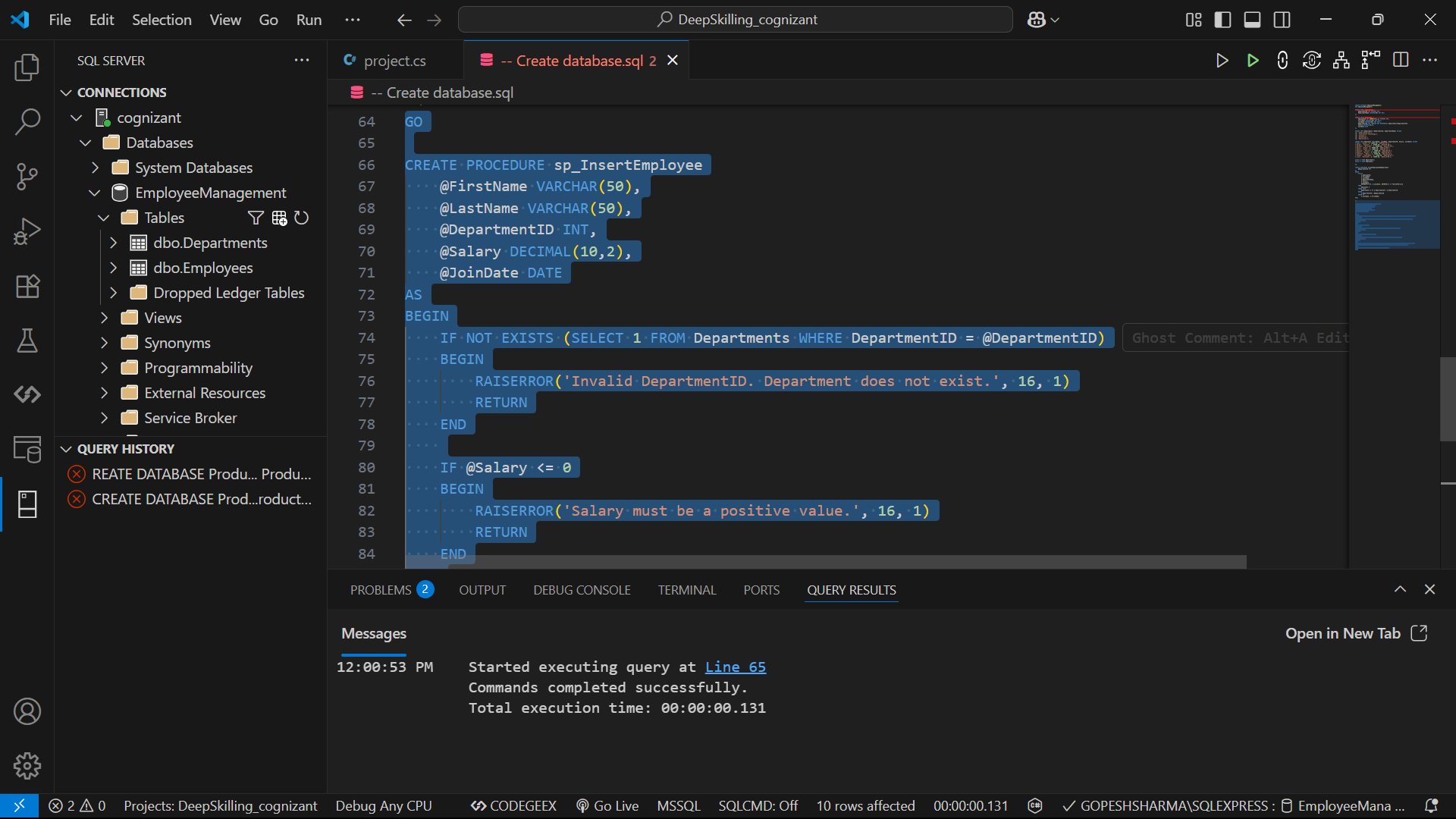
    @DepartmentID = 99,  -- Non-existent department

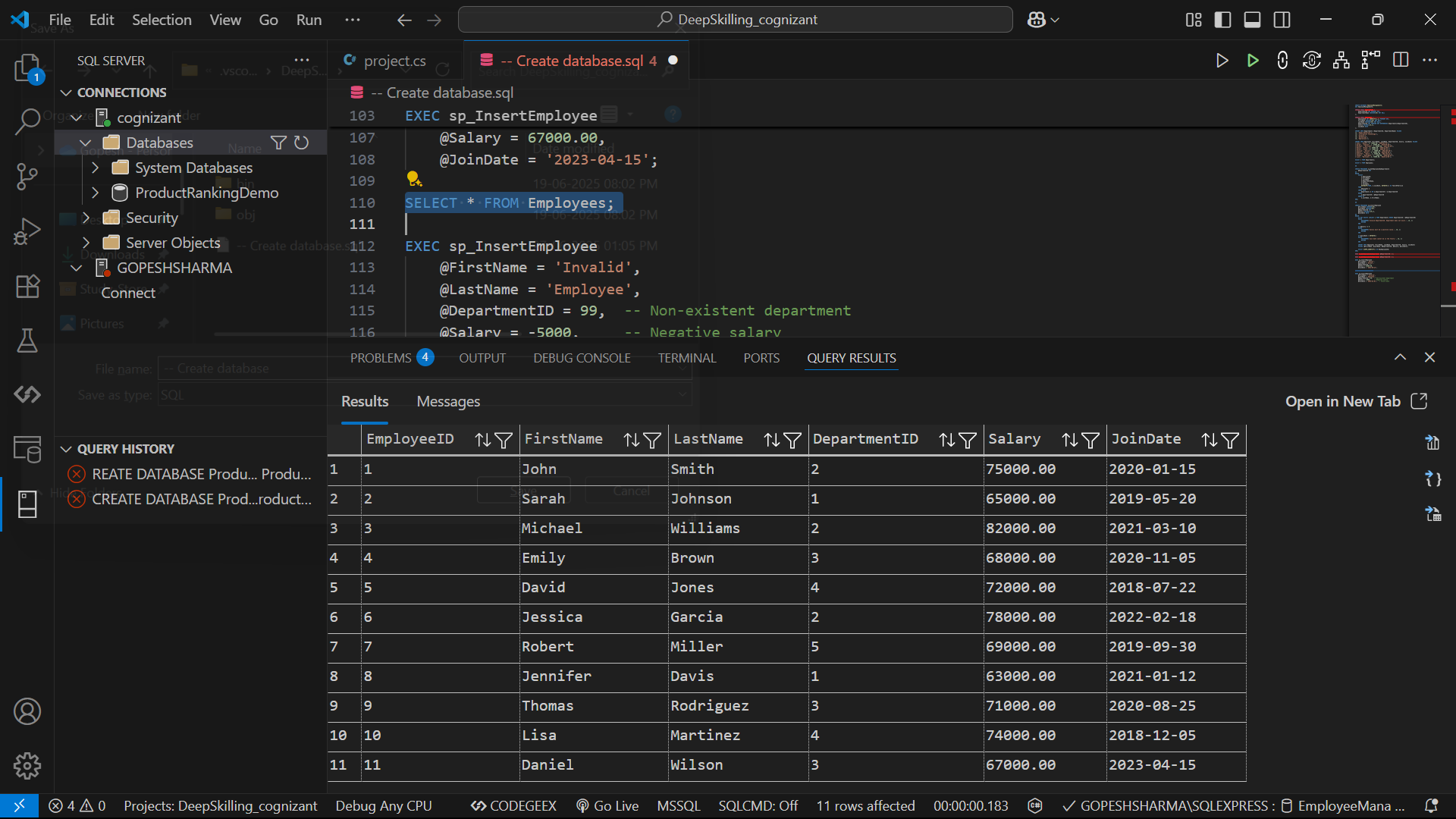
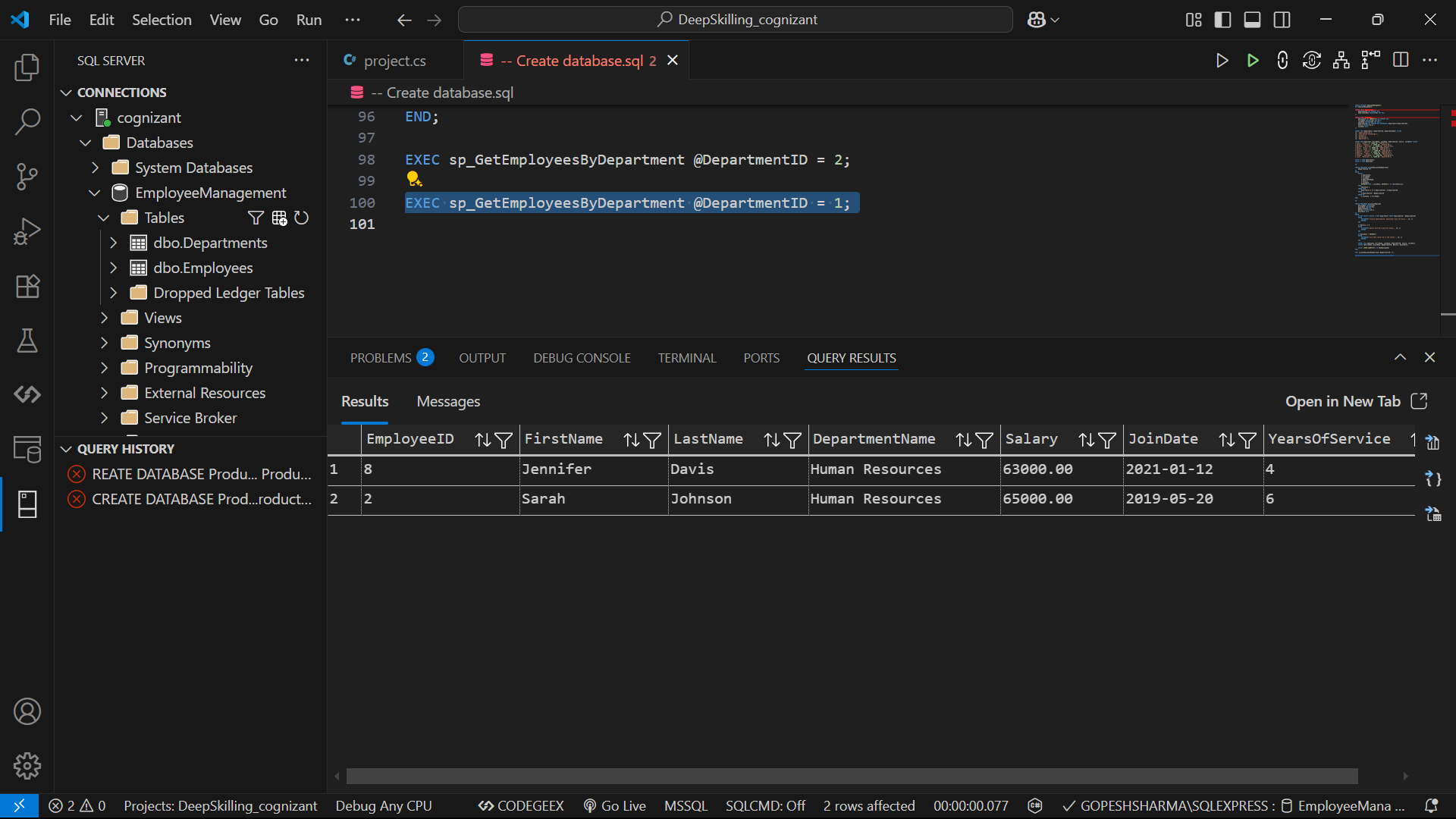
    @Salary = -5000,     -- Negative salary

    @JoinDate = '2025-01-01';  -- Future date

Output:







Problem – 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.

Sol:

GO

CREATE PROCEDURE sp\_GetEmployeeCountByDepartment

    @DepartmentID INT

AS

BEGIN

    DECLARE @EmployeeCount INT;

    SELECT @EmployeeCount = COUNT(\*)

    FROM Employees

    WHERE DepartmentID = @DepartmentID;

SELECT @EmployeeCount AS EmployeeCount;

END;

GO

CREATE PROCEDURE sp\_GetEmployeeCountByDepartmentEnhanced

    @DepartmentID INT

AS

BEGIN

    SELECT

        d.DepartmentName,

        COUNT(e.EmployeeID) AS EmployeeCount

    FROM

        Departments d

    LEFT JOIN

        Employees e ON d.DepartmentID = e.DepartmentID

    WHERE

        d.DepartmentID = @DepartmentID

    GROUP BY

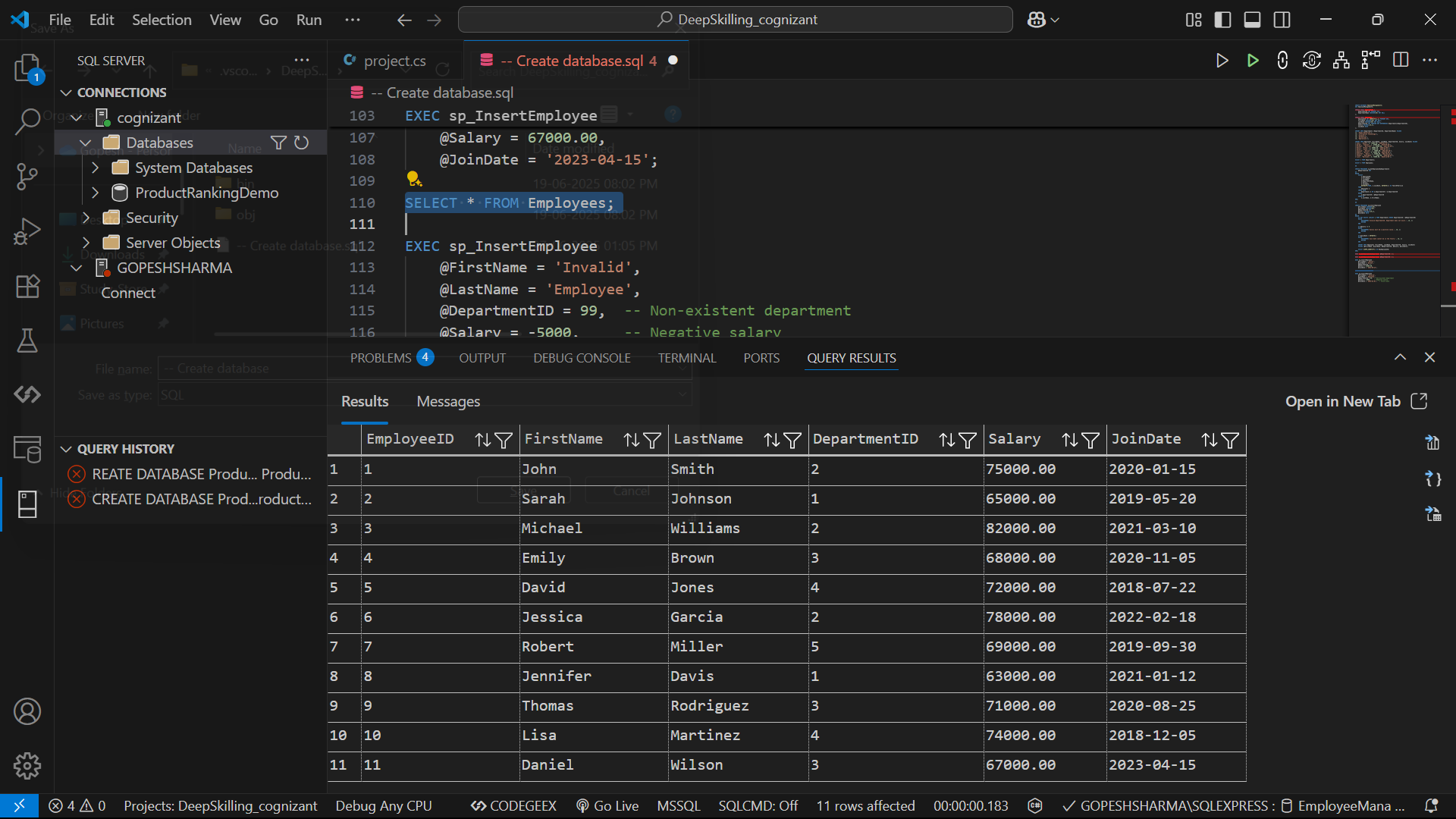
        d.DepartmentName;

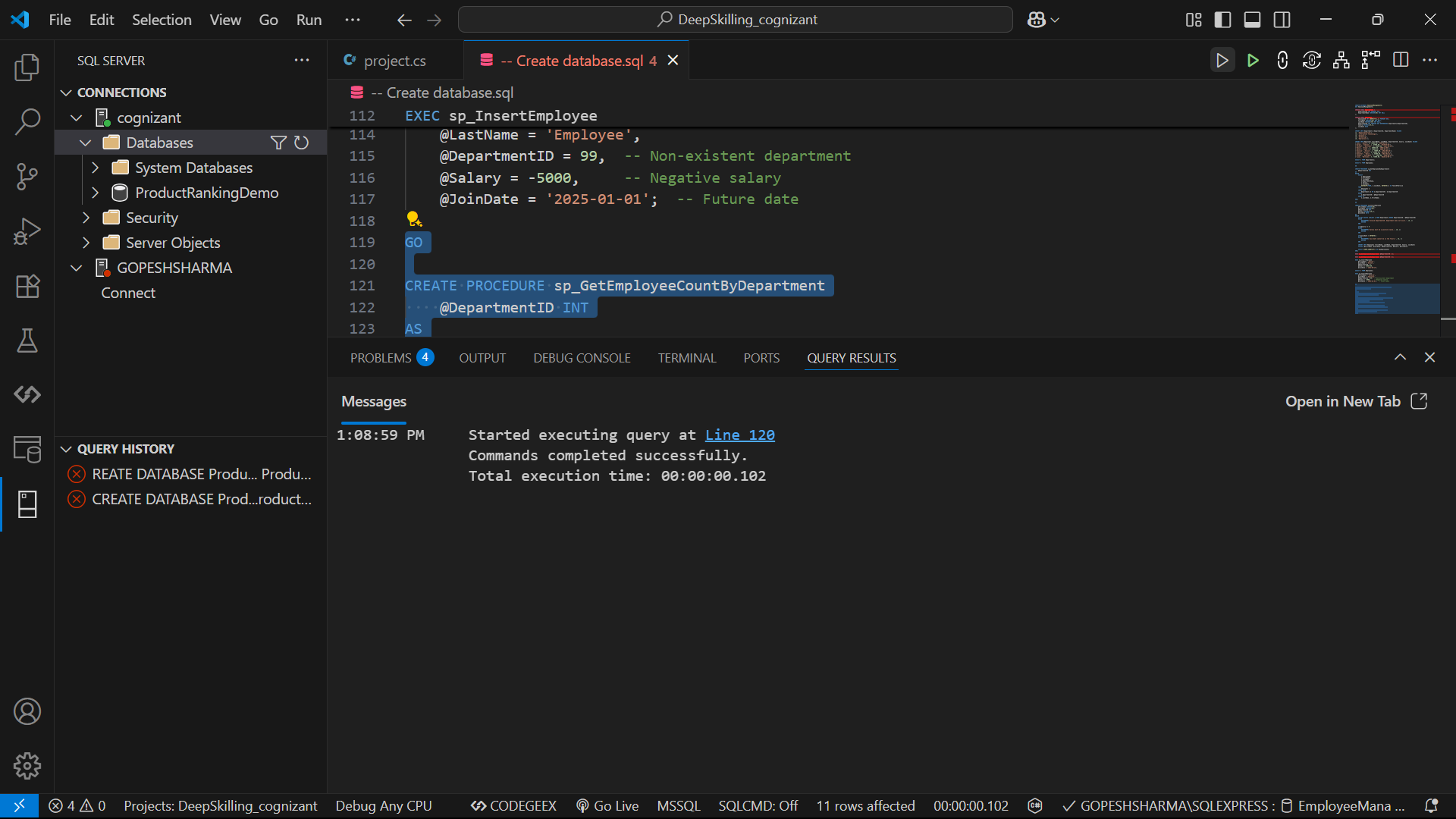
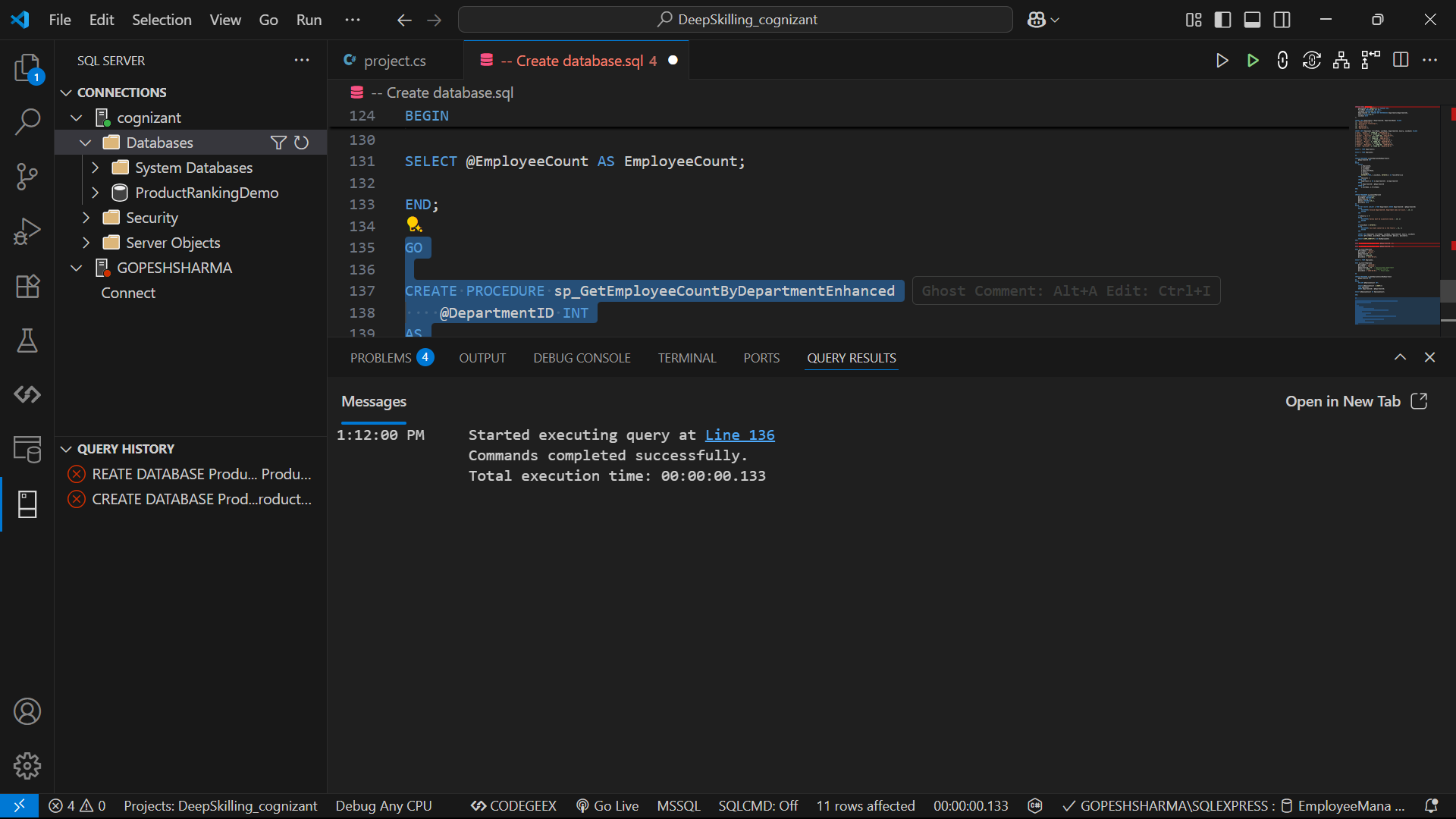
END;

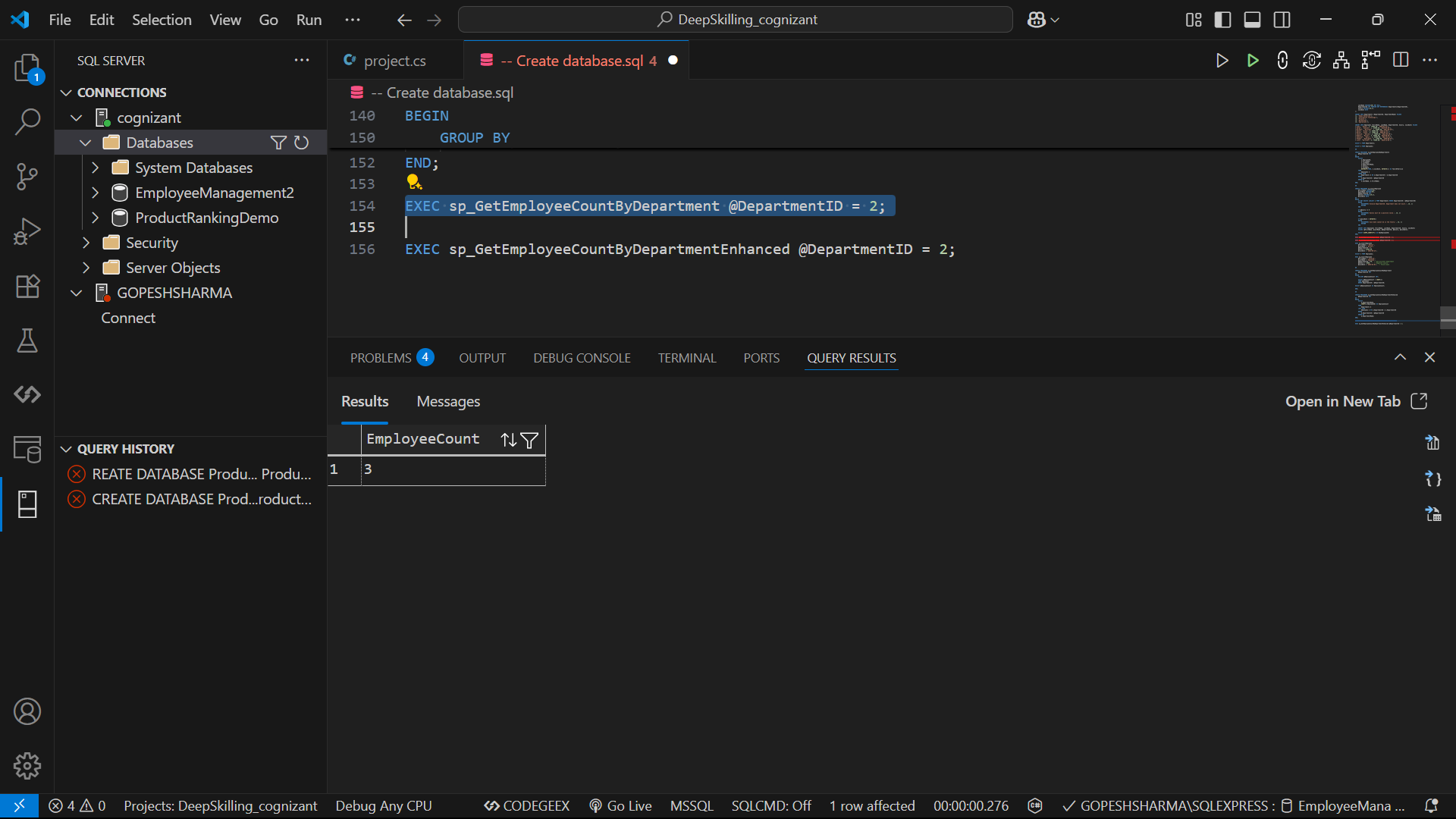
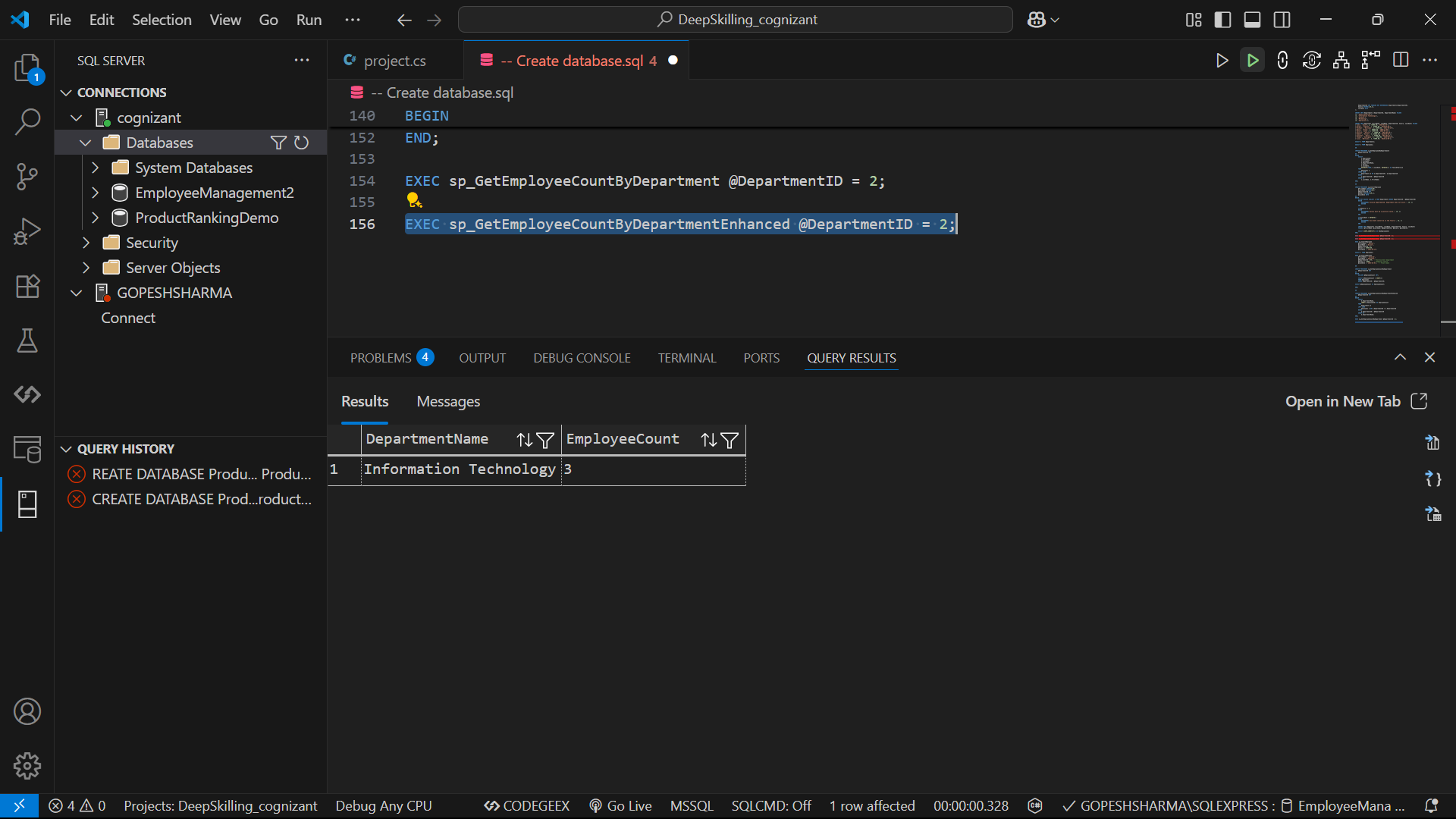
EXEC sp\_GetEmployeeCountByDepartment @DepartmentID = 2;

EXEC sp\_GetEmployeeCountByDepartmentEnhanced @DepartmentID = 2;

Output:







Problem – 4:

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

Sol:

using System;

namespace CalcLibrary

{

public interface ISimpleCalculator

{

double Add(double a, double b);

double Subtract(double a, double b);

double Multiply(double a, double b);

double Divide(double a, double b);

void Dispose();

}

public class SimpleCalculator : ISimpleCalculator, IDisposable

{

public double Add(double a, double b)

{

return a + b;

}

public double Subtract(double a, double b)

{

return a - b;

}

public double Multiply(double a, double b)

{

return a \* b;

}

public double Divide(double a, double b)

{

if (b == 0)

throw new DivideByZeroException("Division by zero is not allowed");

return a / b;

}

public void Dispose()

{

// Cleanup resources if needed

}

}

}

Test:

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalculatorTests

{

[TestFixture]

public class CalculatorTests

{

private ISimpleCalculator \_calculator;

[SetUp]

public void Setup()

{

// Initialize the calculator before each test

\_calculator = new SimpleCalculator();

}

[TearDown]

public void Cleanup()

{

// Clean up after each test

\_calculator.Dispose();

\_calculator = null;

}

// Addition Tests

[TestCase(5, 3, 8)]

[TestCase(-1, 1, 0)]

[TestCase(0, 0, 0)]

[TestCase(2.5, 3.5, 6)]

[TestCase(double.MaxValue, double.MaxValue, double.PositiveInfinity)]

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

{

var result = \_calculator.Add(a, b);

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

}

// Subtraction Tests

[TestCase(5, 3, 2)]

[TestCase(10, 2, 8)]

[TestCase(0, 0, 0)]

[TestCase(-5, -3, -2)]

public void Subtraction\_ValidInputs\_ReturnsCorrectResult(double a, double b, double expected)

{

var result = \_calculator.Subtract(a, b);

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

}

// Multiplication Tests

[TestCase(5, 3, 15)]

[TestCase(10, 0, 0)]

[TestCase(-5, 3, -15)]

[TestCase(2.5, 4, 10)]

public void Multiplication\_ValidInputs\_ReturnsCorrectResult(double a, double b, double expected)

{

var result = \_calculator.Multiply(a, b);

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

}

// Division Tests

[TestCase(10, 2, 5)]

[TestCase(9, 3, 3)]

[TestCase(1, 3, 0.33333, 0.0001)] // Allowing tolerance for floating point

public void Division\_ValidInputs\_ReturnsCorrectResult(double a, double b, double expected, double tolerance = 0)

{

var result = \_calculator.Divide(a, b);

Assert.That(result, Is.EqualTo(expected).Within(tolerance));

}

[Test]

public void Division\_ByZero\_ThrowsDivideByZeroException()

{

Assert.That(() => \_calculator.Divide(5, 0), Throws.TypeOf<DivideByZeroException>());

}

// Edge Cases

[Test]

public void Add\_MaxValuePlusOne\_ReturnsInfinity()

{

var result = \_calculator.Add(double.MaxValue, 1);

Assert.That(result, Is.EqualTo(double.PositiveInfinity));

}

[Test]

public void Multiply\_MaxValues\_ReturnsInfinity()

{

var result = \_calculator.Multiply(double.MaxValue, double.MaxValue);

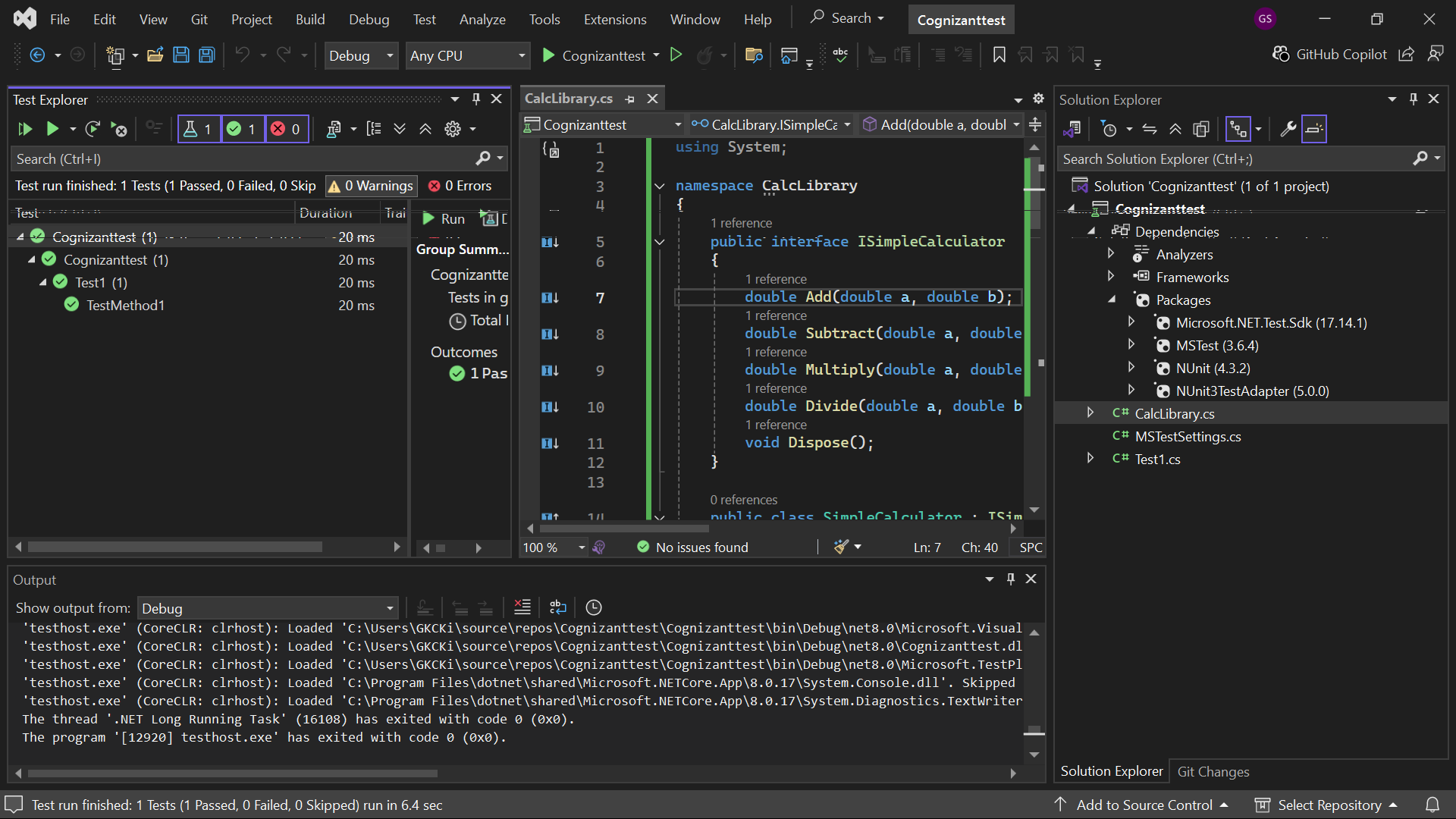
Assert.That(result, Is.EqualTo(double.PositiveInfinity));

}

}

}

Output:



Problem – 5:

MOQ:

Code:

// 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)

{

try

{

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;

}

catch

{

return false;

}

}

}

public class CustomerComm

{

private readonly IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

// Actual logic goes here

return \_mailSender.SendMail("cust123@abc.com", "This is my hadson test Message !!");

}

}

}

// CustomerCommTests.cs

using CustomerCommLib;

using Moq;

using NUnit.Framework;

namespace CustomerComm.Tests

{

[TestFixture]

public class CustomerCommTests

{

private Mock<IMailSender> \_mockMailSender;

private CustomerComm \_customerComm;

[OneTimeSetUp]

public void Setup()

{

// Create mock object

\_mockMailSender = new Mock<IMailSender>();

// Configure mock to always return true

\_mockMailSender.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>()))

.Returns(true);

// Inject mock dependency

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

}

[Test]

public void SendMailToCustomer\_WhenCalled\_ReturnsTrue()

{

// Act

var result = \_customerComm.SendMailToCustomer();

// Assert

Assert.IsTrue(result);

}

[Test]

public void SendMailToCustomer\_WhenCalled\_VerifyMailSenderInvoked()

{

// Act

\_customerComm.SendMailToCustomer();

// Verify that SendMail was called exactly once

\_mockMailSender.Verify(

x => x.SendMail(It.IsAny<string>(), It.IsAny<string>()),

Times.Once);

}

[Test]

[TestCase("cust123@abc.com", "Some Message")]

public void SendMailToCustomer\_WhenCalled\_VerifyCorrectParameters(string expectedAddress, string expectedMessage)

{

// Act

\_customerComm.SendMailToCustomer();

// Verify that SendMail was called with specific parameters

\_mockMailSender.Verify(

x => x.SendMail(

It.Is<string>(addr => addr == expectedAddress),

It.Is<string>(msg => msg == expectedMessage)

)

);

}

}

}