**Exercise 1: Ranking and Window Functions**

-- 1.1 ROW\_NUMBER: Top 3 most expensive products per category

SELECT \*

FROM (

SELECT

ProductID,

ProductName,

Category,

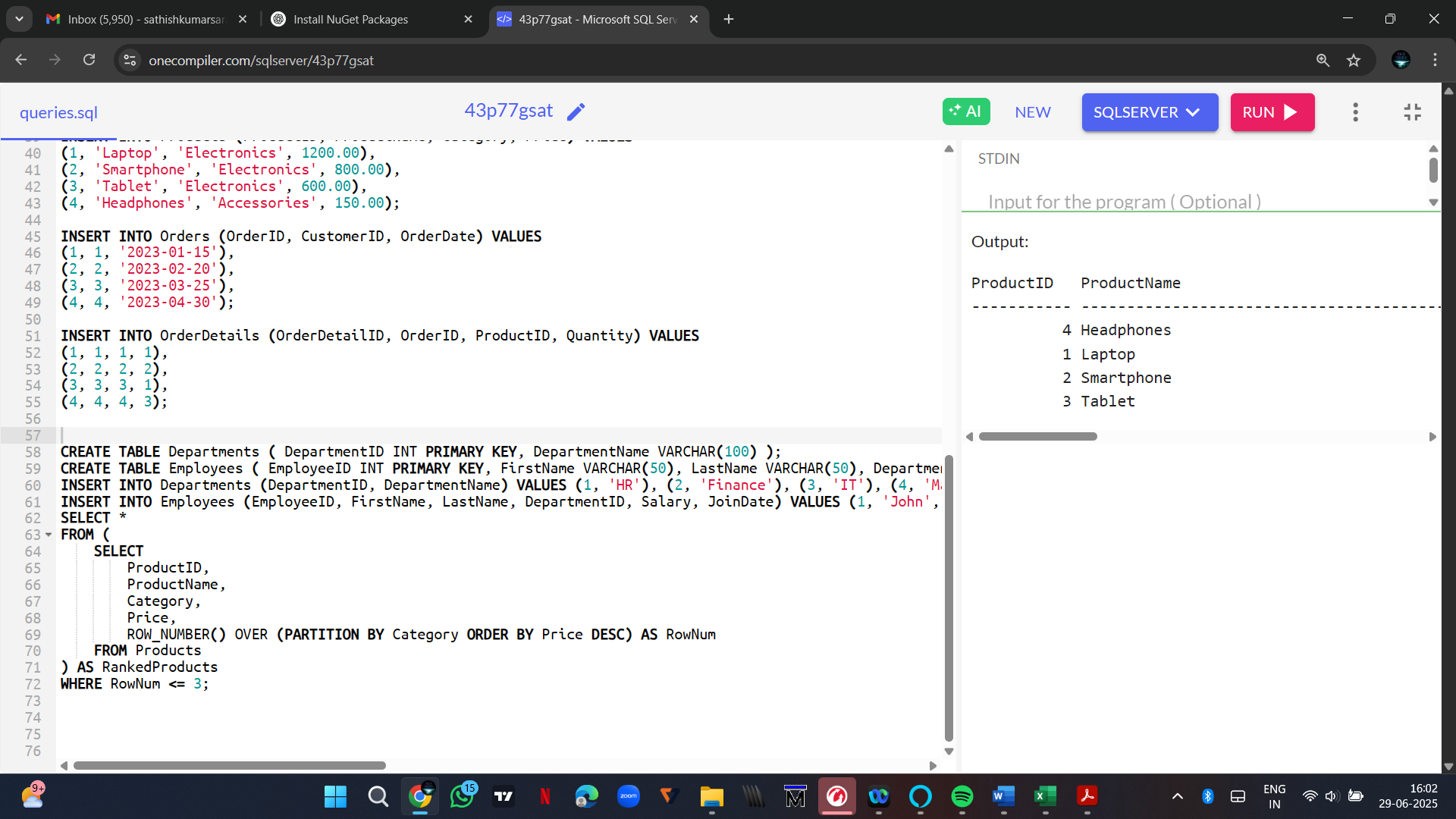
Price,

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

FROM Products

) AS RankedProducts

WHERE RowNum <= 3;



-- 1.2 RANK: To observe tie behavior

SELECT \*

FROM (

SELECT

ProductID,

ProductName,

Category,

Price,

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

FROM Products

) AS RankedProducts

WHERE PriceRank <= 3;

-- 1.3 DENSE\_RANK: No gaps in ranking

SELECT \*

FROM (

SELECT

ProductID,

ProductName,

Category,

Price,

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

FROM Products

) AS RankedProducts

WHERE DensePriceRank <= 3;

**2. SQL Exercise - Index**

-- =============================================

-- EXERCISE 1: NON-CLUSTERED INDEX

-- =============================================

-- 2.1 Query before index

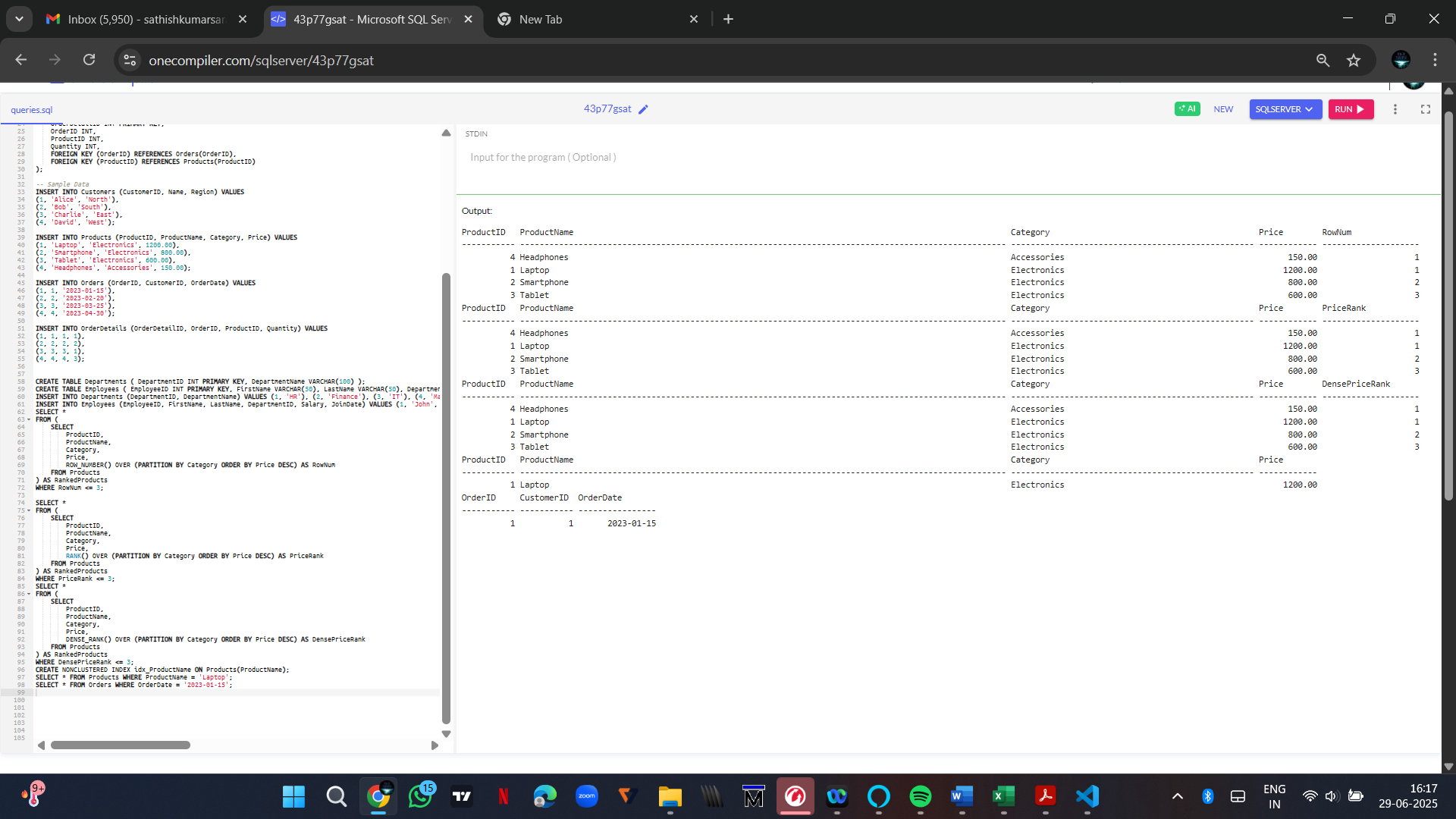
SELECT \* FROM Products WHERE ProductName = 'Laptop';

-- 2.2 Create non-clustered index

CREATE NONCLUSTERED INDEX idx\_ProductName ON Products(ProductName);

-- 2.3 Query after index

SELECT \* FROM Products WHERE ProductName = 'Laptop';



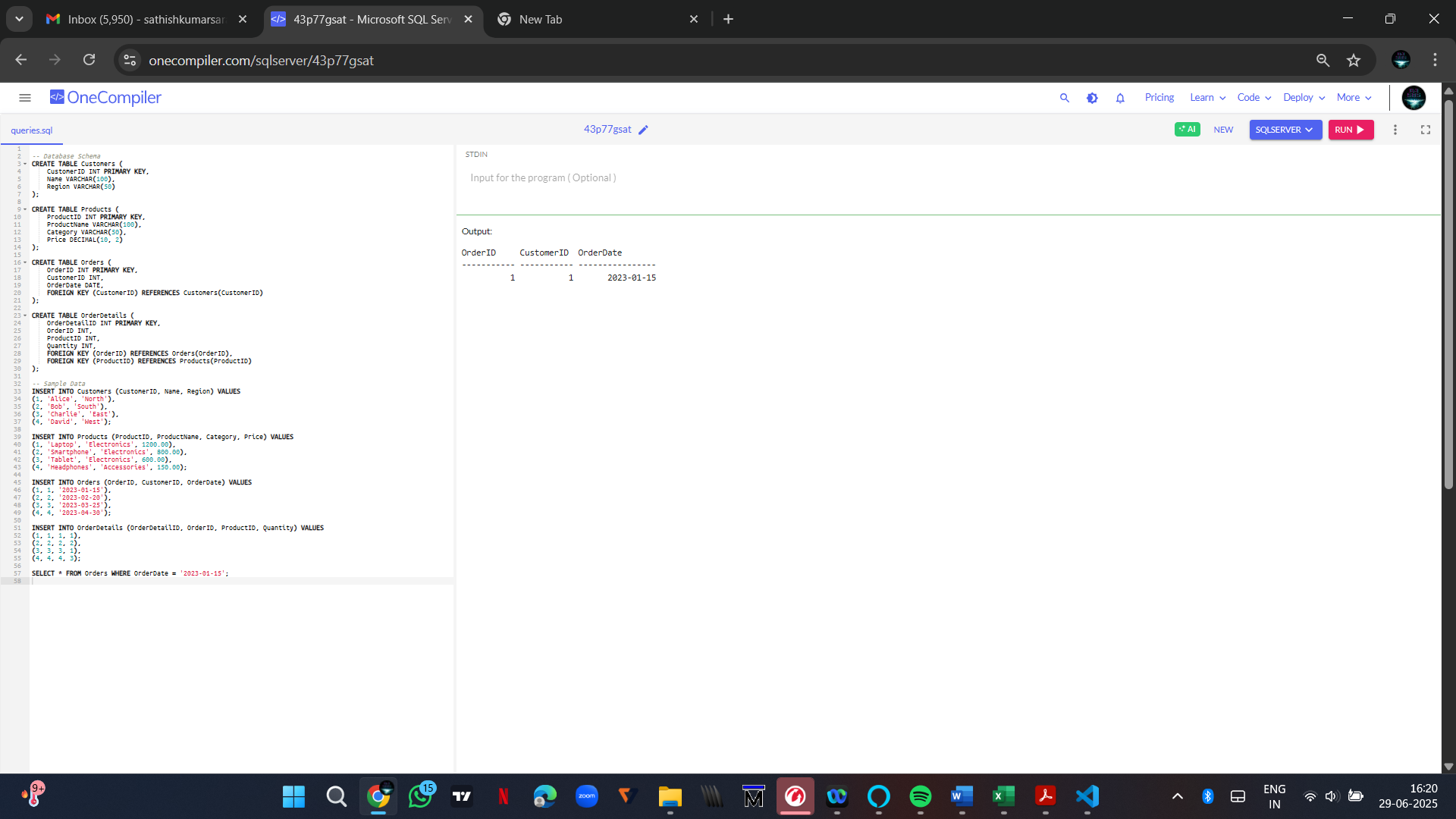
-- =============================================

-- EXERCISE 2: CLUSTERED INDEX

-- =============================================

-- 3.1 Query before index

SELECT \* FROM Orders WHERE OrderDate = '2023-01-15';



-- 3.2 Create clustered index

-- Warning: Only one clustered index per table. Skip if primary key is already clustered.

-- You may need to drop existing clustered index before creating this.

-- Example: DROP INDEX PK\_Orders ON Orders;

CREATE CLUSTERED INDEX idx\_OrderDate ON Orders(OrderDate);

-- 3.3 Query after index

SELECT \* FROM Orders WHERE OrderDate = '2023-01-15';

-- =============================================

-- EXERCISE 3: COMPOSITE INDEX

-- =============================================

-- 4.1 Query before index

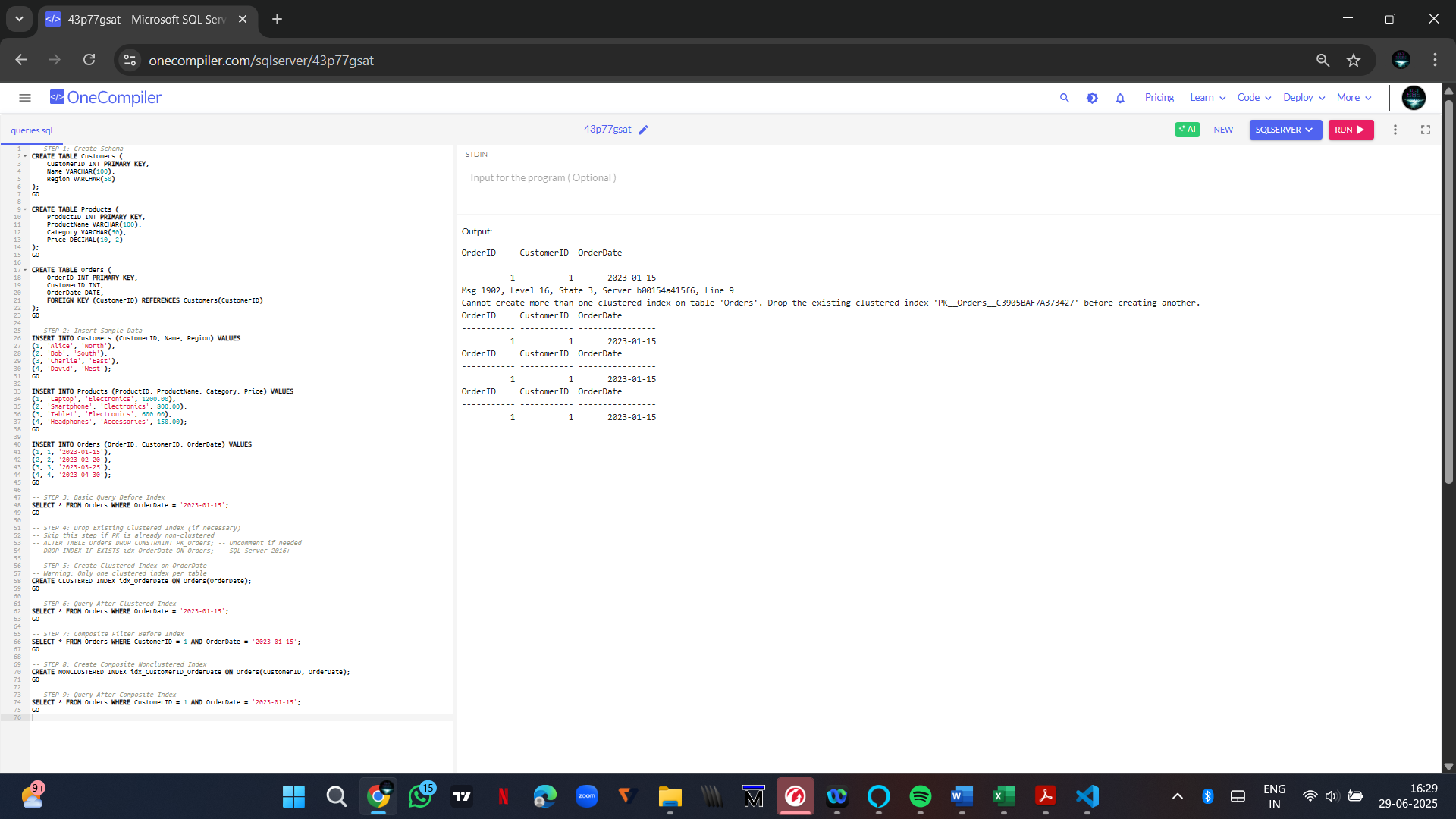
SELECT \* FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';

-- 4.2 Create composite index

CREATE NONCLUSTERED INDEX idx\_CustomerID\_OrderDate ON Orders(CustomerID, OrderDate);

-- 4.3 Query after index

SELECT \* FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';



**-- EXERCISE 1: Stored Procedure to Retrieve Employees by Department**

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

SELECT EmployeeID, FirstName, LastName, Salary, JoinDate

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

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;

**-- EXERCISE 5: Stored Procedure to Count Employees in a Department**

CREATE PROCEDURE sp\_CountEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

**Exercise 7: Return Data from a Scalar Function**

-- Create the scalar function to calculate annual salary

CREATE FUNCTION fn\_CalculateAnnualSalary (@EmployeeID INT)

RETURNS DECIMAL(10,2)

AS

BEGIN

DECLARE @MonthlySalary DECIMAL(10,2);

DECLARE @AnnualSalary DECIMAL(10,2);

SELECT @MonthlySalary = Salary

FROM Employees

WHERE EmployeeID = @EmployeeID;

SET @AnnualSalary = @MonthlySalary \* 12;

RETURN @AnnualSalary;

END;

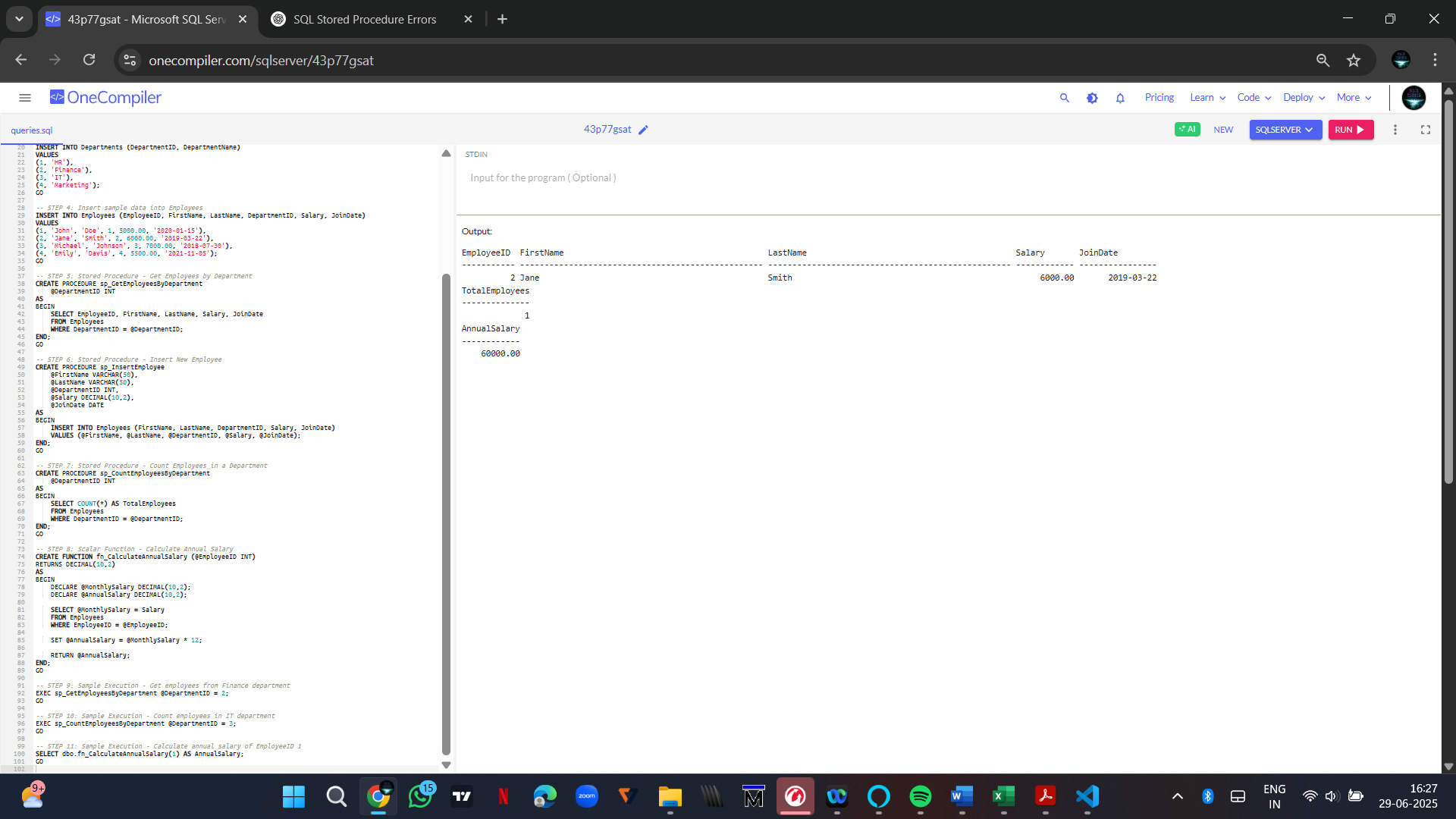
GO

-- Execute the function for EmployeeID = 1

SELECT dbo.fn\_CalculateAnnualSalary(1) AS AnnualSalary;

**Exercise 4: Execute a Stored Procedure**

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 2;



|  |  |
| --- | --- |
| **1. NUnit-Handson** | **Hands-on in this document** |

using NUnit.Framework;

using CalcLibrary; // Ensure this matches your actual library namespace

namespace CalcTests

{

[TestFixture]

public class CalculatorTests

{

private Calculator calc;

[SetUp]

public void SetUp()

{

calc = new Calculator();

}

[TearDown]

public void TearDown()

{

calc = null;

}

[Test]

public void Test\_Addition\_Simple()

{

int result = calc.Add(2, 3);

Assert.That(result, Is.EqualTo(5));

}

[TestCase(1, 1, 2)]

[TestCase(0, 0, 0)]

[TestCase(-1, -1, -2)]

public void Test\_Addition\_WithMultipleCases(int a, int b, int expected)

{

int result = calc.Add(a, b);

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

}

[Test, Ignore("This test is ignored for demonstration purposes.")]

public void Test\_IgnoredExample()

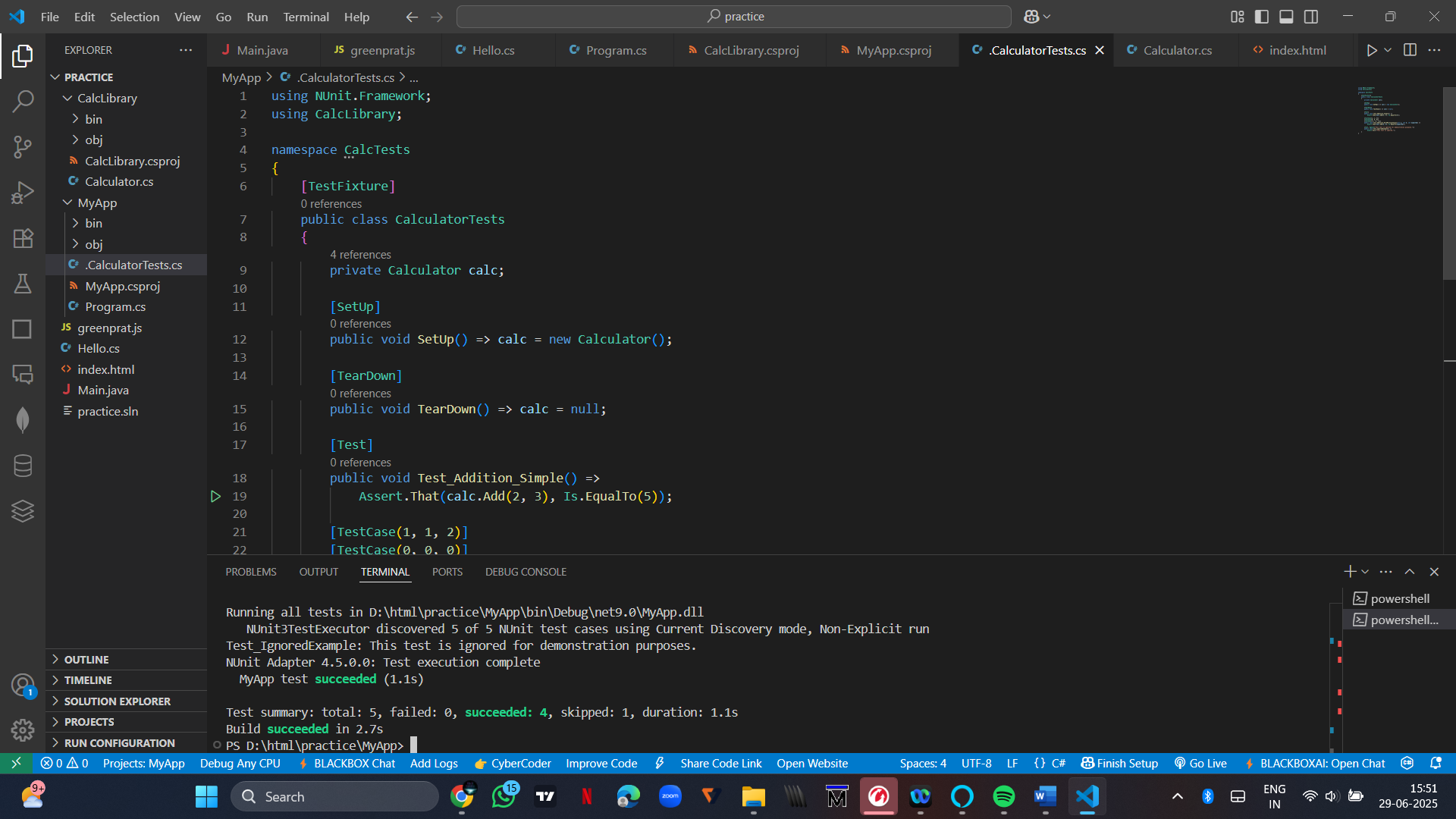
{

Assert.Fail("This test is ignored.");

}

}

}



|  |  |
| --- | --- |
| **1. Moq-Handson** | **1. Write Testable Code with Moq** |

// Required NuGet Packages:

// - NUnit

// - Moq

// - NUnit3TestAdapter

using System;

using System.Net;

using System.Net.Mail;

using NUnit.Framework;

using Moq;

namespace CustomerCommApp

{

// ========== Task 1 ==========

// Define the interface

public interface IMailSender

{

bool SendMail(string toAddress, string message);

}

// Actual mail sender class (external dependency)

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;

}

}

// Class under test

public class CustomerComm

{

private readonly IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

string to = "cust123@abc.com";

string message = "Some Message";

return \_mailSender.SendMail(to, message);

}

}

// ========== Task 2 ==========

// Unit test using Moq and NUnit

[TestFixture]

public class CustomerCommTests

{

private Mock<IMailSender> mockMailSender;

private 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 CustomerComm(mockMailSender.Object);

}

[TestCase]

public void SendMailToCustomer\_ShouldReturnTrue\_WhenMailIsSent()

{

bool result = customerComm.SendMailToCustomer();

Assert.That(result, Is.True);

}

}

}

