**Week 2: Advanced SQL, Nunit and Moq**

**SQL Exercises: Advanced SQL Concepts**

**Exercise 1: RANKING AND WINDOW FUNCTIONS**

**QUERY:**

WITH RankedProducts AS (

SELECT

ProductID,

ProductName,

Category,

Price,

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

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

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

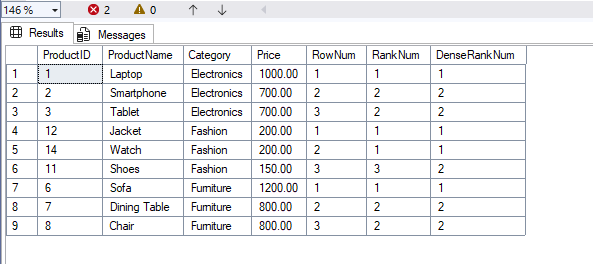
from

Products

)

select \* from RankedProducts where RowNum <= 3;

**Output:**

****

**SQL Exercise: Stored Procedure**

**Exercise 1: Create a Stored Procedure**

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

Create PROCEDURE sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

select \* from Employees

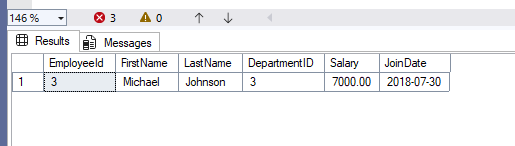
where DepartmentID = @DepartmentID;

END;

**2. Select employee based on DepartmentID**

SELECT \* FROM Employees WHERE DepartmentID = 3;

**Output:**



**3. Stored procedure named ‘sp\_InsertEmployee’**

EXEC sp\_InsertEmployee

@FirstName = 'Sachin',

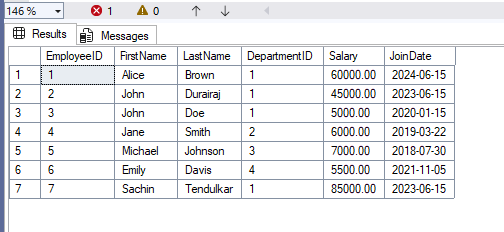
@LastName = 'Tendulkar',

@DepartmentID = 1,

@Salary = 85000.00,

@JoinDate = '2023-06-15';

**Output:**

****

**SQL Exercises: Functions**

**Exercise 7: Return data from Scalar function**

**1.Execute the `fn\_CalculateAnnualSalary` function for an employee with `EmployeeID = 1`.**

create FUNCTION dbo.fn\_CalculateAnnualSalary (@EmpID INT)

returns DECIMAL(12,2)

AS

BEGIN

DECLARE @Annual DECIMAL(12,2);

select @Annual = Salary \* 12

from Employees where EmployeeID = @EmpID;

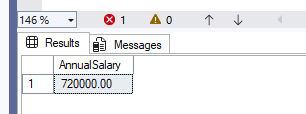
RETURN @Annual;

END;

GO

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

**Output:**

****

**2. Verify the result**

SELECT e.EmployeeID,

e.FirstName,

e.LastName,

d.DepartmentName,

e.Salary AS MonthlySalary,

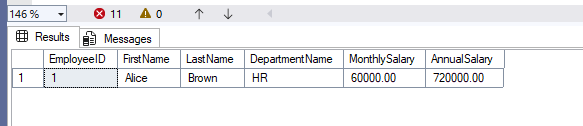
dbo.fn\_CalculateAnnualSalary(e.EmployeeID) AS AnnualSalary

FROM Employees e

JOIN Departments d ON d.DepartmentID = e.DepartmentID

WHERE e.EmployeeID = 1;

**Output:**



**SQL Exercise: Stored Procedure**

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

**1.Stored Procedure with DepartmentdID**

create PROCEDURE sp\_GetEmployeeCountByDepartment

@DepartmentID INT

AS

BEGIN

select COUNT(\*) AS EmployeeCount from Employees

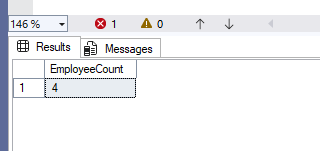
where DepartmentID = @DepartmentID;

END;

**2.Query for count the number of employees in the department**

EXEC sp\_GetEmployeeCountByDepartment @DepartmentID = 1;

**Output:**

****

**NUNIT HandsOn**

**Task:**

* 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

**Implementation:**

**Calculator.cs**

namespace CalcLibrary

{

    public class Calculator

    {

        public int Div(int a, int b)

        {

            return a /b;

        }

public int Mul(int a, int b){

return a\*b;

}

    }

}

**CalculatorTest.cs**

using NUnit.Framework;

using CalcLibrary;

namespace CalcLibrary.Tests

{

[TestFixture]

public class CalculatorTests

{

private Calculator calc;

[SetUp]

public void Setup()

{

calc = new Calculator();

}

[TearDown]

public void Teardown()

{

calc = null;

}

[Test]

[TestCase(2, 3, 6)]

[TestCase(0, 5, 0)]

[TestCase(-2, 4, -8)]

public void Multiply\_WhenCalled\_ReturnsExpectedResult(int a, int b, int expected)

{

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

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

}

[Test]

[TestCase(6, 3, 2)]

[TestCase(10, 5, 2)]

[TestCase(-8, 2, -4)]

public void Divide\_WhenCalled\_ReturnsExpectedResult(int a, int b, int expected)

{

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

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

}

[Test]

public void Divide\_ByZero\_ThrowsDivideByZeroException()

{

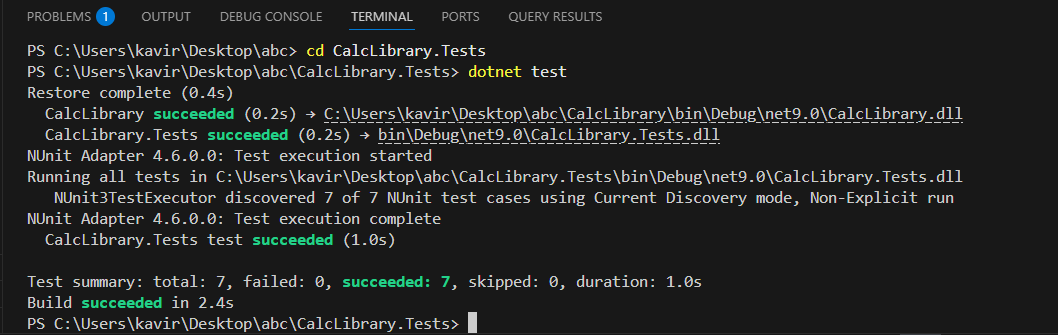
Assert.Throws<System.DivideByZeroException>(() => calc.Divide(5, 0));

}

}

}

**Output:**

****

**MOQ HandsOn**

**1.Write Testable Code with Moq**

**Scenario:**

You are tasked to write a unit test code for the below scenario.

The application in which you are teamed up with, deals with a mail server communication in which your application tries to send mail to its users upon every transaction. Your role is to write unit testing the module that contains send mail functionality. You wanted to perform testing the module without sending any email.

After investigating the problem scenario, you found a solution and that is creating mock objects of these external dependencies in the unit testing project so that you can achieve speedier test execution and loose coupling of code.

**Task 1:**

**IMailSender.cs**

namespace CustomerCommLib

{

    public interface IMailSender

    {

        bool SendMail(string toAddress, string message);

    }

}

**MailSender.cs**

using System.Net;

using System.Net.Mail;

namespace CustomerCommLib

{

    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;

        }

    }

}

**Task 2:**

**CustomerComm.cs**

namespace CustomerCommLib

{

    public class CustomerComm

    {

        IMailSender \_mailSender;

        public CustomerComm(IMailSender mailSender)

        {

            \_mailSender = mailSender;

        }

        public bool SendMailToCustomer()

        {

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

        }

    }

}

**CustomerCommTests.cs**

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerCommTests

{

    [TestFixture]

    public class CustomerCommTests

    {

        private Mock<IMailSender> mockMailSender;

        private CustomerCommLib.CustomerComm customerComm;

        [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]

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

        public void SendMailToCustomer\_ShouldReturnTrue(string email, string message)

        {

                   bool result = customerComm.SendMailToCustomer();

                   Assert.That(result, Is.True);

        }

    }

}

**Output:**

