**DN 4.0 Dotnet FSE**

**Name: Aditya Mohanty**

**Superset ID:6361043**

**Week 2-**

1. **SQL Exercise - Advanced concepts**

**Exercise 1: Ranking and Window Functions**

**Code:**

WITH RankedProducts AS (

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

)

SELECT

Category,

ProductName,

Price,

RowNum,

Rank,

DenseRank

FROM RankedProducts

WHERE DenseRank <= 3;

**Schema Code:**

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

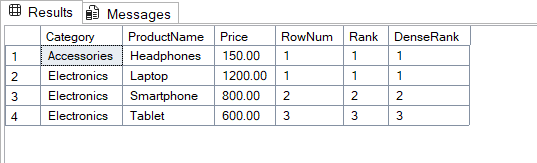
(1, 'Laptop', 'Electronics', 1200.00),

(2, 'Smartphone', 'Electronics', 800.00),

(3, 'Tablet', 'Electronics', 600.00),

(4, 'Headphones', 'Accessories', 150.00);

**Output:**



1. **SQL Exercise - Stored procedure**

**Exercise 1: Create a Stored Procedure**

**Schema Code:**

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT,

Salary DECIMAL(10,2),

JoinDate DATE

);

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

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

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

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

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

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

**Part 1:**

## **Get Employees by Department (simulate stored procedure for DepartmentID = 2):**

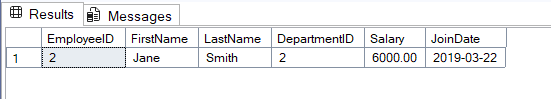
**Query Code:**

SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary, JoinDate

FROM Employees

WHERE DepartmentID = 2;

**Output:**



## Part 2: ****Insert a New Employee****

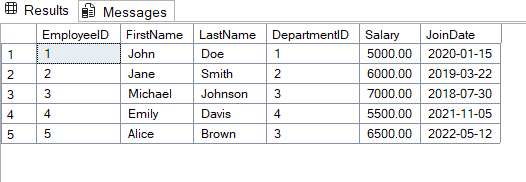
****SQL Query:****

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

**VALUES ('Alice', 'Brown', 3, 6500.00, '2022-05-12');**

**SELECT \* FROM Employees;**

****OUTPUT:****



#4 SQL Exercise - Stored procedure

**Exercise 5: Return Data from a Stored Procedure (Using SSMS now)**

**Code:**

CREATE PROCEDURE sp\_GetEmployeeCountByDepartment

    @DepartmentID INT

AS

BEGIN

    SELECT COUNT(\*) AS EmployeeCount

    FROM Employees

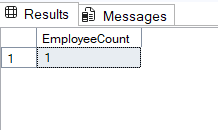
    WHERE DepartmentID = @DepartmentID;

END

GO

EXEC sp\_GetEmployeeCountByDepartment @DepartmentID = 2;

**OUTPUT:**

****

**NUnit-Handson**

**TestFixture & Test**

**Code for unit test:**

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 Cleanup()

{

\_calculator = null;

}

[TestCase(5, 3, 8)]

[TestCase(-2, 10, 8)]

[TestCase(0, 0, 0)]

[TestCase(2.5, 3.1, 5.6)]

public void TestAddition(double a, double b, double expected)

{

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

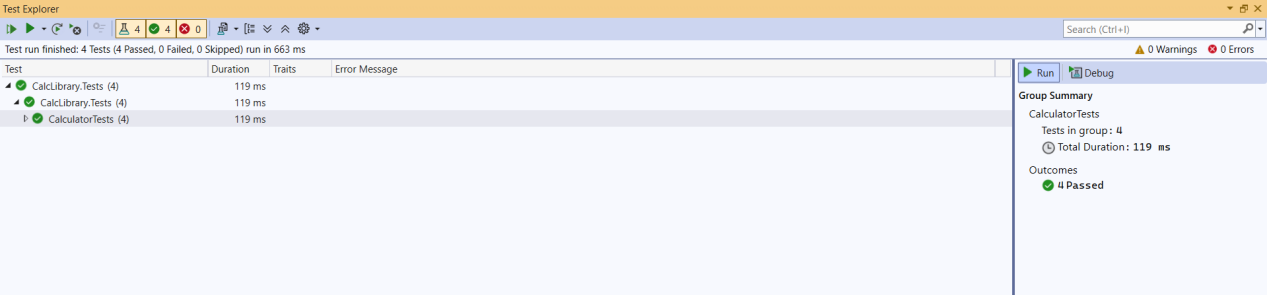
Assert.That(result, Is.EqualTo(expected));

}

}

}

**OUTPUT:**



**Moq-Handson**

**Write Testable Code with Moq**

**Codes:**

using CustomerCommLib;

using Moq;

using NUnit.Framework;

namespace CustomerComm.Tests

{

[TestFixture] // Marks class as test fixture

public class CustomerCommTests

{

private Mock<IMailSender> \_mockMailSender;

private CustomerComm \_customerComm;

[OneTimeSetUp] // Runs once before all tests

public void OneTimeSetup()

{

// Initialize mock and configure default behavior

\_mockMailSender = new Mock<IMailSender>();

\_mockMailSender.Setup(m =>

m.SendMail(It.IsAny<string>(), It.IsAny<string>())

).Returns(true); // Always return true for any parameters

}

[SetUp] // Runs before each test

public void Setup()

{

// Create CustomerComm with mocked dependency

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

}

[Test] // Test case without parameters

public void SendMailToCustomer\_WhenCalled\_ReturnsTrue()

{

// Act

bool result = \_customerComm.SendMailToCustomer();

// Assert

Assert.IsTrue(result);

}

[TestCase("test@domain.com", "Hello World")] // Parameterized test

[TestCase("user@example.org", "Another message")]

public void SendMailToCustomer\_WithDifferentParams\_AlwaysReturnsTrue(string address, string message)

{

bool result = \_customerComm.SendMailToCustomer();

Assert.IsTrue(result);

}

[Test]

public void SendMailToCustomer\_WhenCalled\_InvokesSendMail()

{

\_customerComm.SendMailToCustomer();

\_mockMailSender.Verify(m =>

m.SendMail(It.IsAny<string>(), It.IsAny<string>()),

Times.Once

);

}

}

}