**ADVANCED SQL**

**Exercise 1: Ranking and Window Functions**

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(100),

Price DECIMAL(10, 2)

);

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

VALUES

(1, 'Laptop A', 'Electronics', 1200.00),(2, 'Laptop B', 'Electronics', 1150.00),(3, 'Smartphone A', 'Electronics', 800.00),(4, 'Smartphone B', 'Electronics', 800.00),(5, 'TV A', 'Electronics', 700.00),(6, 'Shampoo A', 'Personal Care', 10.00),(7, 'Shampoo B', 'Personal Care', 8.00),(8, 'Conditioner A', 'Personal Care', 10.00),(9, 'Soap A','Personal Care', 5.00),(10, 'Soap B', 'Personal Care', 5.00),(11, 'T-Shirt A', 'Clothing', 20.00),(12, 'T-Shirt B', 'Clothing', 18.00),(13, 'Jeans A', 'Clothing', 40.00),(14, 'Jacket A', 'Clothing', 60.00),(15, 'Jacket B', 'Clothing', 60.00);

WITH RankedElectronics AS (

SELECT

ProductID,

ProductName,

Category,

Price,

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

FROM Products

)

SELECT \*

FROM RankedElectronics

WHERE Category = 'Electronics' AND RankNum <= 3

ORDER BY RankNum;

WITH RankedPersonalCare AS (

SELECT

ProductID,

ProductName,

Category,

Price,

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

FROM Products

)

SELECT \*

FROM RankedPersonalCare

WHERE Category = 'Personal Care' AND RankNum <= 3

ORDER BY RankNum;

WITH RankedClothing AS (

SELECT

ProductID,

ProductName,

Category,

Price,

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

FROM Products

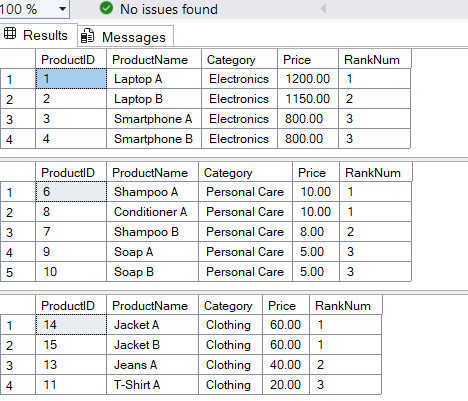
)

SELECT \*

FROM RankedClothing

WHERE Category = 'Clothing' AND RankNum <= 3

ORDER BY RankNum;

****

**Exercise 1: Create a Stored Procedure**

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 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 (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');

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

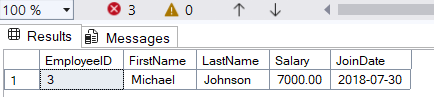
SELECT EmployeeID, FirstName, LastName, Salary, JoinDate

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 3;



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

CREATE PROCEDURE sp\_GetEmployeeCount

@DepartmentID INT

AS

BEGIN

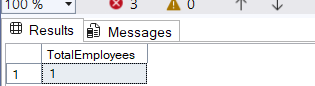
SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

EXEC sp\_GetEmployeeCount @DepartmentID = 1;



**NUNIT AND MOQ**

**1. Write Testable code with MOQ**

using System;

using System.Net;

using System.Net.Mail;

using NUnit.Framework;

using Moq;

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;

}

}

public class CustomerComm

{

IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

string toAddress = "cust123@abc.com";

string message = "Some Message";

return \_mailSender.SendMail(toAddress, message);

}

}

[TestFixture]

public class CustomerCommTests

{

[Test]

public void SendMailToCustomer\_MailSenderCalled\_ReturnsTrue()

{

var mockMailSender = new Mock<IMailSender>();

mockMailSender

.Setup(sender => sender.SendMail(It.IsAny<string>(), It.IsAny<string>()))

.Returns(true);

var customerComm = new CustomerComm(mockMailSender.Object);

var result = customerComm.SendMailToCustomer();

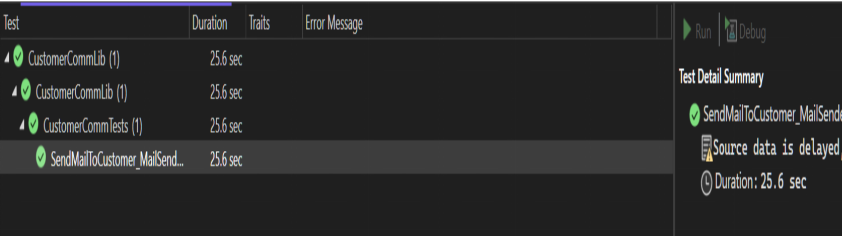
Assert.That(result, Is.True);

mockMailSender.Verify(sender => sender.SendMail("cust123@abc.com", "Some Message"), Times.Once);

}

}

}



**2. Test Fixture and Test**

using System;

using NUnit.Framework;

namespace CalcLibrary

{

public class Calculator

{

public int Add(int n1, int n2) => n1 + n2;

public int Subtract(int n1, int n2) => n1 - n2;

public int Multiply(int n1, int n2) => n1 \* n2;

public int Divide(int n1, int n2) => n1 / n2;

}

[TestFixture]

public class CalculatorTests

{

private Calculator calculator;

[SetUp]

public void Setup()

{

calculator = new Calculator();

}

[TearDown]

public void TearDown()

{

calculator = null;

}

[Test]

[TestCase(2, 3, 5)]

[TestCase(5, 10, 15)]

public void TestAdd(int n1, int n2, int expected)

{

Assert.That(calculator.Add(n1, n2), Is.EqualTo(expected));

}

[Test]

[TestCase(5, 3, 2)]

[TestCase(4, 4, 0)]

public void TestSub(int n1, int n2, int expected)

{

Assert.That(calculator.Subtract(n1, n2), Is.EqualTo(expected));

}

[Test]

[TestCase(10, 10, 100)]

[TestCase(4, 5, 20)]

public void TestMultiply(int n1, int n2, int expected)

{

Assert.That(calculator.Multiply(n1, n2), Is.EqualTo(expected));

}

[Test]

[TestCase(100,10,10)]

[TestCase(14,7,2)]

public void TestDivide(int n1, int n2, int expected)

{

Assert.That(calculator.Divide(n1, n2), Is.EqualTo(expected));

}

}

}

