**SQL**

**Exercise 1: Ranking and Window Functions**

Products(

ProductID INT,

ProductName VARCHAR,

Category VARCHAR,

Price DECIMAL

)

SELECT

Category,

ProductName,

Price,

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

FROM Products;

SELECT

Category,

ProductName,

Price,

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

FROM Products;

SELECT

Category,

ProductName,

Price,

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

FROM Products;

SELECT \*

FROM (

SELECT

Category,

ProductName,

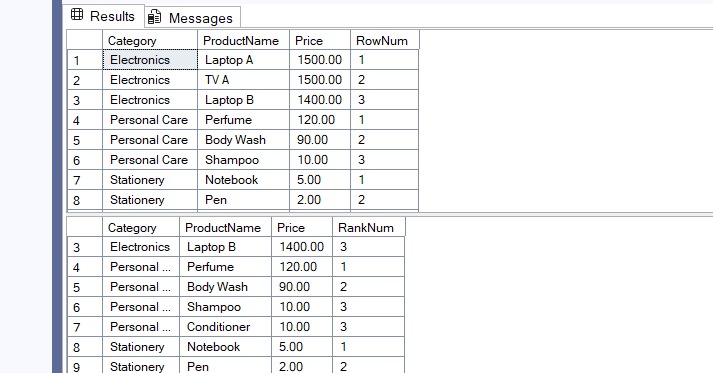
Price,

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

FROM Products

) Ranked

WHERE RowNum <= 3;



**SQL Exercise -Index:**

-- Database Schema

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

Region VARCHAR(50)

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE OrderDetails (

OrderDetailID INT PRIMARY KEY,

OrderID INT,

ProductID INT,

Quantity INT,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

-- Sample Data

INSERT INTO Customers (CustomerID, Name, Region) VALUES

(1, 'Alice', 'North'),

(2, 'Bob', 'South'),

(3, 'Charlie', 'East'),

(4, 'David', 'West');

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);

INSERT INTO Orders (OrderID, CustomerID, OrderDate) VALUES

(1, 1, '2023-01-15'),

(2, 2, '2023-02-20'),

(3, 3, '2023-03-25'),

(4, 4, '2023-04-30');

INSERT INTO OrderDetails (OrderDetailID, OrderID, ProductID, Quantity) VALUES

(1, 1, 1, 1),

(2, 2, 2, 2),

(3, 3, 3, 1),

(4, 4, 4, 3);

-- Exercise 1

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

CREATE NONCLUSTERED INDEX idx\_ProductName

ON Products (ProductName);

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

-- Exercise 2

CREATE NONCLUSTERED INDEX idx\_OrderDate

ON Orders (OrderDate);

-- Exercise 3

-- Step 1: Query before index creation

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

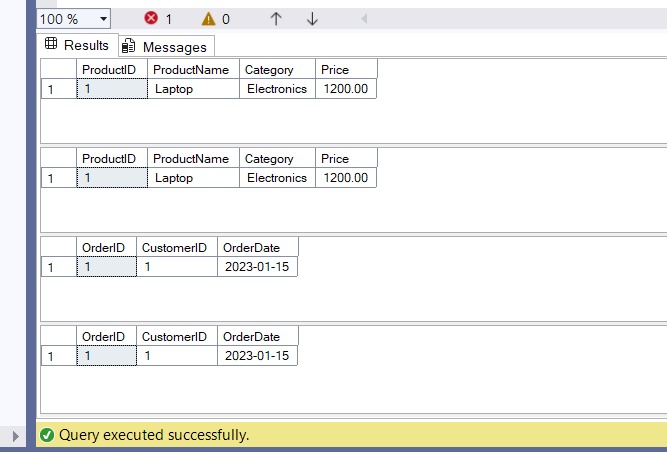
-- Step 2: Create a composite (non-clustered) index

CREATE NONCLUSTERED INDEX idx\_CustomerID\_OrderDate

ON Orders (CustomerID, OrderDate);

-- Step 3: Query after index creation

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



**Exercise 1: Create a Stored Procedure**

CREATE TABLE Employees (

EmployeeID INT IDENTITY(1,1) PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT,

Salary DECIMAL(10,2),

JoinDate DATE

);

GO

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

SELECT

EmployeeID,

FirstName,

LastName,

DepartmentID,

Salary,

JoinDate

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 101;

GO

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;

EXEC sp\_InsertEmployee

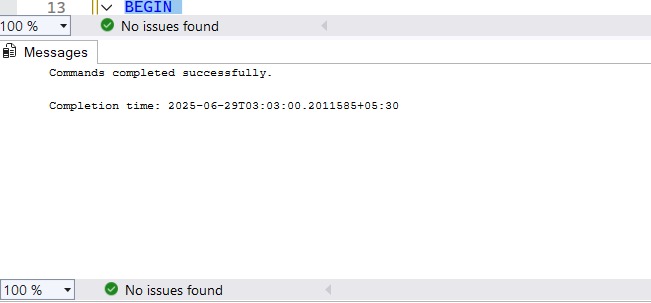
@FirstName = 'John',

@LastName = 'Doe',

@DepartmentID = 101,

@Salary = 50000.00,

@JoinDate = '2024-06-01';



**Exercise 5: Return Data & 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');

GO

CREATE PROCEDURE usp\_GetEmployeeCountByDepartment

@DepartmentID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

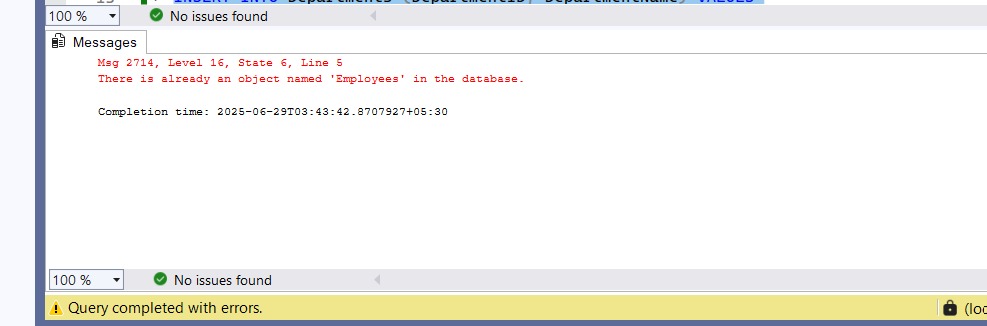
FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

-- Example: Count employees in Department 1

EXEC usp\_GetEmployeeCountByDepartment @DepartmentID = 1;

****

**SQL Exercise-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');

GO

CREATE PROCEDURE GetEmployeesByDepartment

@DeptID INT

AS

BEGIN

SELECT

E.EmployeeID,

E.FirstName,

E.LastName,

D.DepartmentName,

E.Salary,

E.JoinDate

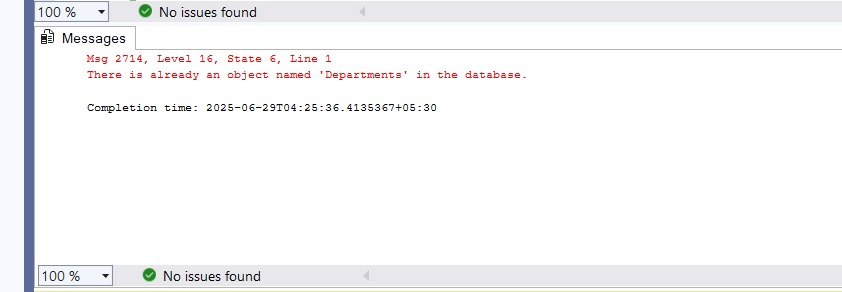
FROM Employees E

INNER JOIN Departments D ON E.DepartmentID = D.DepartmentID

WHERE E.DepartmentID = @DeptID;

END;

EXEC GetEmployeesByDepartment @DeptID = 3;



**MOQ HANDSON:**

**SCENARIO 1:**

**TASK01**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Mail;

using System.Text;

using System.Threading.Tasks;

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

{

private readonly IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

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

}

}

}

**SCENARIO 2:**

**TASK01**

using System.Collections.Generic;

namespace MagicFilesLib

{

public interface IDirectoryExplorer

{

ICollection<string> GetFiles(string path);

}

}

using System.Collections.Generic;

using System.IO;

namespace MagicFilesLib

{

public class DirectoryExplorer : IDirectoryExplorer

{

public ICollection<string> GetFiles(string path)

{

string[] files = Directory.GetFiles(path);

return files;

}

}

}

**TASK02**

using NUnit.Framework;

using Moq;

using MagicFilesLib;

using System.Collections.Generic;

namespace DirectoryExplorer.Tests

{

[TestFixture]

public class DirectoryExplorerTests

{

private readonly string \_file1 = "file.txt";

private readonly string \_file2 = "file2.txt";

private IDirectoryExplorer \_mockedExplorer;

[OneTimeSetUp]

public void Init()

{

var mock = new Mock<IDirectoryExplorer>();

mock.Setup(x => x.GetFiles(It.IsAny<string>()))

.Returns(new List<string> { \_file1, \_file2 });

\_mockedExplorer = mock.Object;

}

[TestCase]

public void Test\_GetFiles\_Returns\_CorrectFiles()

{

var files = \_mockedExplorer.GetFiles("C:\\dummy");

Assert.IsNotNull(files, "File collection should not be null.");

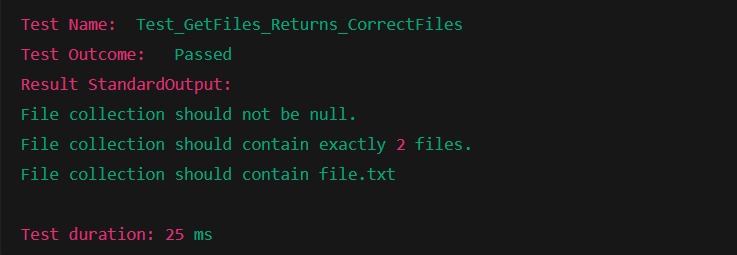
Assert.AreEqual(2, files.Count, "File collection should contain exactly 2 files.");

CollectionAssert.Contains(files, \_file1, "File collection should contain file.txt");

}

}

}



**SCENARIO 3:**

**TASK01**

namespace PlayersManagerLib

{

public interface IPlayerMapper

{

bool IsPlayerNameExistsInDb(string name);

void AddNewPlayerIntoDb(string name);

}

}

using System.Data;

using System.Data.SqlClient;

namespace PlayersManagerLib

{

public class PlayerMapper : IPlayerMapper

{

private readonly string \_connectionString =

"Data Source=(local);Initial Catalog=GameDB;Integrated Security=True";

public bool IsPlayerNameExistsInDb(string name)

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

using (SqlCommand command = connection.CreateCommand())

{

command.CommandText = "SELECT count(\*) FROM Player WHERE [Name] = @name";

command.Parameters.AddWithValue("@name", name);

int existingPlayersCount = (int)command.ExecuteScalar();

return existingPlayersCount > 0;

}

}

}

public void AddNewPlayerIntoDb(string name)

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

using (SqlCommand command = connection.CreateCommand())

{

command.CommandText = "INSERT INTO Player ([Name]) VALUES (@name)";

command.Parameters.AddWithValue("@name", name);

command.ExecuteNonQuery();

}

}

}

}

}

using System;

namespace PlayersManagerLib

{

public class Player

{

public string Name { get; private set; }

public int Age { get; private set; }

public string Country { get; private set; }

public int NoOfMatches { get; private set; }

public Player(string name, int age, string country, int noOfMatches)

{

Name = name;

Age = age;

Country = country;

NoOfMatches = noOfMatches;

}

public static Player RegisterNewPlayer(string name, IPlayerMapper playerMapper = null)

{

if (playerMapper == null)

playerMapper = new PlayerMapper();

if (string.IsNullOrWhiteSpace(name))

throw new ArgumentException("Player name can’t be empty.");

if (playerMapper.IsPlayerNameExistsInDb(name))

throw new ArgumentException("Player name already exists.");

playerMapper.AddNewPlayerIntoDb(name);

return new Player(name, 23, "India", 30);

}

}

}

**Task02**

using NUnit.Framework;

using Moq;

using PlayersManagerLib;

using System;

namespace PlayerManager.Tests

{

[TestFixture]

public class PlayerTests

{

private Mock<IPlayerMapper> \_mockMapper;

[OneTimeSetUp]

public void Init()

{

\_mockMapper = new Mock<IPlayerMapper>();

}

[TestCase]

public void RegisterNewPlayer\_WhenNameIsValid\_ReturnsPlayer()

{

// Arrange

string playerName = "Rohit";

\_mockMapper.Setup(m => m.IsPlayerNameExistsInDb(playerName)).Returns(false);

\_mockMapper.Setup(m => m.AddNewPlayerIntoDb(playerName));

// Act

Player player = Player.RegisterNewPlayer(playerName, \_mockMapper.Object);

// Assert

Assert.IsNotNull(player);

Assert.AreEqual("Rohit", player.Name);

Assert.AreEqual(23, player.Age);

Assert.AreEqual("India", player.Country);

Assert.AreEqual(30, player.NoOfMatches);

}

[TestCase]

public void RegisterNewPlayer\_WhenNameIsEmpty\_ThrowsArgumentException()

{

// Assert

var ex = Assert.Throws<ArgumentException>(() =>

Player.RegisterNewPlayer("", \_mockMapper.Object));

Assert.AreEqual("Player name can’t be empty.", ex.Message);

}

[TestCase]

public void RegisterNewPlayer\_WhenNameExistsInDb\_ThrowsArgumentException()

{

// Arrange

string playerName = "Virat";

\_mockMapper.Setup(m => m.IsPlayerNameExistsInDb(playerName)).Returns(true);

// Assert

var ex = Assert.Throws<ArgumentException>(() =>

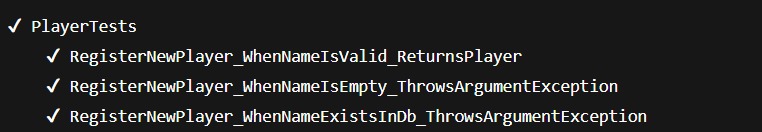
Player.RegisterNewPlayer(playerName, \_mockMapper.Object));

Assert.AreEqual("Player name already exists.", ex.Message);

}

}

}



A3

**NUnit-Handson:**

1. **NUnit-Handson**

using NUnit.Framework;

using CalcLibrary;

namespace CalcLibrary.Tests

{

[TestFixture]

public class CalculatorTests

{

private Calculator \_calculator;

[SetUp]

public void SetUp()

{

// Initialize before each test

\_calculator = new Calculator();

}

[TearDown]

public void TearDown()

{

// Cleanup after each test if needed

\_calculator = null;

}

[Test]

[TestCase(2, 3, 5)]

[TestCase(10, 20, 30)]

[TestCase(-5, 5, 0)]

public void Add\_WhenCalled\_ReturnsSum(int a, int b, int expectedResult)

{

// Act

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

// Assert

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

}

}

}

**--Assumed Calculator**

namespace CalcLibrary

{

public class Calculator

{

public int Add(int a, int b)

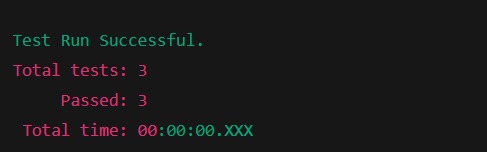
{

return a + b;

}

}

}



1. **NUnit-Handson**

**Step 1: Calculator/MathLibrary Class Assumption**

namespace MathLibrary

{

public class Calculator

{

private int result;

public int GetResult => result;

public int Add(int a, int b)

{

result = a + b;

return result;

}

public int Subtract(int a, int b)

{

result = a - b;

return result;

}

public int Multiply(int a, int b)

{

result = a \* b;

return result;

}

public int Divide(int a, int b)

{

if (b == 0)

throw new ArgumentException("Cannot divide by zero");

result = a / b;

return result;

}

public void AllClear()

{

result = 0;

}

}

}

**Step 2: Unit Test Code using NUnit**

using NUnit.Framework;

using MathLibrary;

using System;

namespace MathLibrary.Tests

{

[TestFixture]

public class CalculatorTests

{

private Calculator calculator;

[SetUp]

public void SetUp()

{

calculator = new Calculator();

}

[TearDown]

public void TearDown()

{

calculator = null;

}

// Subtraction Tests

[Test]

[TestCase(10, 5, 5)]

[TestCase(20, 30, -10)]

[TestCase(0, 0, 0)]

public void Subtract\_WithVariousInputs\_ReturnsExpected(int a, int b, int expected)

{

var actual = calculator.Subtract(a, b);

Assert.AreEqual(expected, actual);

}

// Multiplication Tests

[Test]

[TestCase(2, 3, 6)]

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

[TestCase(0, 10, 0)]

public void Multiply\_WithVariousInputs\_ReturnsExpected(int a, int b, int expected)

{

var actual = calculator.Multiply(a, b);

Assert.AreEqual(expected, actual);

}

// Division Tests

[Test]

[TestCase(10, 2, 5)]

[TestCase(9, 3, 3)]

public void Divide\_WithValidInputs\_ReturnsExpected(int a, int b, int expected)

{

var actual = calculator.Divide(a, b);

Assert.AreEqual(expected, actual);

}

[Test]

public void Divide\_ByZero\_ThrowsArgumentException()

{

try

{

calculator.Divide(10, 0);

Assert.Fail("Division by zero");

}

catch (ArgumentException ex)

{

Assert.AreEqual("Cannot divide by zero", ex.Message);

}

}

// Void Method Test (AllClear)

[Test]

public void TestAddAndClear()

{

int result = calculator.Add(5, 7);

Assert.AreEqual(12, result);

Assert.AreEqual(12, calculator.GetResult);

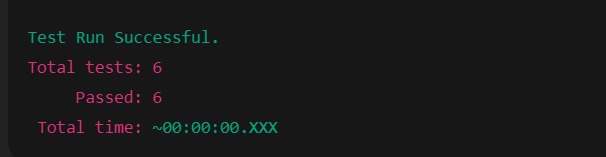
calculator.AllClear();

Assert.AreEqual(0, calculator.GetResult);

}

}

}



1. **NUnit-Handson**

namespace UtilLib

{

public class UrlHostNameParser

{

public string ParseHostName(string url)

{

if (string.IsNullOrWhiteSpace(url))

return "Invalid URL";

try

{

var uri = new Uri(url);

return uri.Host;

}

catch

{

return "Invalid URL";

}

}

}

}

**--write unit test**

using NUnit.Framework;

using UtilLib;

namespace UtilLib.Tests

{

[TestFixture]

public class UrlHostNameParserTests

{

private UrlHostNameParser \_parser;

[SetUp]

public void SetUp()

{

\_parser = new UrlHostNameParser();

}

[Test]

public void ParseHostName\_ValidUrl\_ReturnsHostName()

{

// Arrange

string url = "https://www.example.com/page";

// Act

string result = \_parser.ParseHostName(url);

// Assert

Assert.That(result, Is.EqualTo("www.example.com"));

}

[Test]

public void ParseHostName\_EmptyUrl\_ReturnsInvalidMessage()

{

// Arrange

string url = "";

// Act

string result = \_parser.ParseHostName(url);

// Assert

Assert.That(result, Is.EqualTo("Invalid URL"));

}

[Test]

public void ParseHostName\_MalformedUrl\_ReturnsInvalidMessage()

{

// Arrange

string url = "not a valid url";

// Act

string result = \_parser.ParseHostName(url);

// Assert

Assert.That(result, Is.EqualTo("Invalid URL"));

}

[Test]

public void ParseHostName\_NullInput\_ReturnsInvalidMessage()

{

// Arrange

string url = null;

// Act

string result = \_parser.ParseHostName(url);

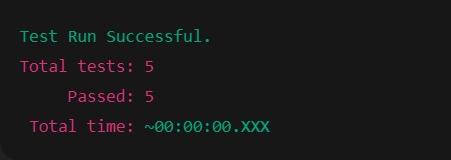
// Assert

Assert.That(result, Is.EqualTo("Invalid URL"));

}

}

}



1. **NUnit-Handson**

**---NUnit Test Code (AccountsManagerLib.Tests)**

using NUnit.Framework;

using AccountsManagerLib;

using System;

namespace AccountsManagerLib.Tests

{

[TestFixture]

public class AccountsManagerTests

{

private AccountsManager manager;

[SetUp]

public void SetUp()

{

manager = new AccountsManager();

}

[Test]

public void Login\_ValidCredentials\_ReturnsWelcomeMessage()

{

var result = manager.Login("user\_11", "secret@user11");

Assert.That(result, Is.EqualTo("Welcome user\_11!!!"));

}

[Test]

public void Login\_InvalidCredentials\_ReturnsInvalidMessage()

{

var result = manager.Login("user\_11", "wrongPassword");

Assert.That(result, Is.EqualTo("Invalid user id/password"));

}

[Test]

public void Login\_EmptyUserId\_ThrowsArgumentException()

{

var ex = Assert.Throws<ArgumentException>(() => manager.Login("", "secret@user11"));

Assert.That(ex.Message, Is.EqualTo("User ID and Password must be provided."));

}

[Test]

public void Login\_EmptyPassword\_ThrowsArgumentException()

{

var ex = Assert.Throws<ArgumentException>(() => manager.Login("user\_11", ""));

Assert.That(ex.Message, Is.EqualTo("User ID and Password must be provided."));

}

[Test]

public void Login\_BothEmpty\_ThrowsArgumentException()

{

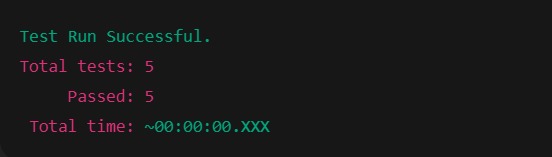
var ex = Assert.Throws<ArgumentException>(() => manager.Login("", ""));

Assert.That(ex.Message, Is.EqualTo("User ID and Password must be provided."));

}

}

}



1. **NUnit-Handson**

**NUnit Test Code**

using NUnit.Framework;

using CollectionsLib;

using System.Linq;

namespace CollectionsLib.Tests

{

[TestFixture]

public class EmployeeManagerTests

{

private EmployeeManager manager;

[SetUp]

public void SetUp()

{

manager = new EmployeeManager();

}

// Scenario 1: Ensure no null values

[Test]

public void GetEmployees\_CollectionContainsNoNulls\_Passes()

{

var employees = manager.GetEmployees();

Assert.That(employees, Has.None.Null);

}

// Scenario 2: Check employee with ID 100 exists

[Test]

public void GetEmployees\_ContainsEmployeeWithId100\_Passes()

{

var employees = manager.GetEmployees();

Assert.That(employees.Any(e => e.Id == 100), Is.True);

}

// Scenario 3: Ensure all employees are unique

[Test]

public void GetEmployees\_ContainsOnlyUniqueEmployees\_Passes()

{

var employees = manager.GetEmployees();

var distinct = employees.Distinct().ToList();

Assert.That(employees.Count, Is.EqualTo(distinct.Count));

}

// Scenario 4: GetEmployees and GetEmployeesWhoJoinedInPreviousYears return same items

[Test]

public void EmployeeLists\_AreSame\_UsingConstraintModel()

{

var current = manager.GetEmployees();

var previous = manager.GetEmployeesWhoJoinedInPreviousYears();

Assert.That(current, Is.EquivalentTo(previous));

}

[Test]

public void EmployeeLists\_AreSame\_UsingClassicModel()

{

var current = manager.GetEmployees();

var previous = manager.GetEmployeesWhoJoinedInPreviousYears();

CollectionAssert.AreEquivalent(current, previous);

}

}

}

1. **NUnit-Handson**

using NUnit.Framework;

using FourSeasonsLib;

using System.Collections.Generic;

namespace FourSeasonsLib.Tests

{

[TestFixture]

public class SeasonConverterTests

{

private SeasonConverter converter;

[SetUp]

public void SetUp()

{

converter = new SeasonConverter();

}

// Inline TestCaseSource using a static property

public static IEnumerable<TestCaseData> SeasonTestCases

{

get

{

yield return new TestCaseData("February", "Spring");

yield return new TestCaseData("March", "Spring");

yield return new TestCaseData("April", "Summer");

yield return new TestCaseData("May", "Summer");

yield return new TestCaseData("June", "Summer");

yield return new TestCaseData("July", "Monsoon");

yield return new TestCaseData("August", "Monsoon");

yield return new TestCaseData("September", "Autumn");

yield return new TestCaseData("October", "Autumn");

yield return new TestCaseData("November", "Autumn");

yield return new TestCaseData("December", "Winter");

yield return new TestCaseData("January", "Winter");

yield return new TestCaseData("NotAMonth", "Invalid month");

yield return new TestCaseData("", "Invalid month");

yield return new TestCaseData(null, "Invalid month");

}

}

[Test, TestCaseSource(nameof(SeasonTestCases))]

public void GetSeason\_MonthGiven\_ReturnsCorrectSeason(string month, string expectedSeason)

{

var actual = converter.GetSeason(month);

Assert.That(actual, Is.EqualTo(expectedSeason));

}

}

}

1. **NUnit-Handson**

using NUnit.Framework;

using LeapYearCalculatorLib;

namespace LeapYearCalculatorLib.Tests

{

[TestFixture]

public class LeapYearCalculatorTests

{

private LeapYearCalculator calculator;

[SetUp]

public void SetUp()

{

calculator = new LeapYearCalculator();

}

[TestCase(2000, 1)] // Leap year divisible by 400

[TestCase(2024, 1)] // Leap year divisible by 4, not by 100

[TestCase(1900, 0)] // Not a leap year: divisible by 100 but not 400

[TestCase(2023, 0)] // Regular non-leap year

[TestCase(1752, -1)] // Invalid: less than 1753

[TestCase(10000, -1)] // Invalid: more than 9999

public void IsLeapYear\_ValidAndInvalidInputs\_ReturnsExpected(int year, int expected)

{

var result = calculator.IsLeapYear(year);

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

}

}

}