**Week-02 HandsOn Solution**

* **Advanced SQL Server**

**File 1: Advanced Concepts**

**Exercise-1: Ranking and Window Functions**

* **Code:**

**//Database Creation:**

CREATE DATABASE OnlineRetailStore;

GO

USE OnlineRetailStore;

GO

CREATE TABLE Products (

    ProductID INT IDENTITY(1,1) PRIMARY KEY,

    ProductName NVARCHAR(100) NOT NULL,

    Category NVARCHAR(50) NOT NULL,

    Price DECIMAL(10,2) NOT NULL,

    Brand NVARCHAR(50) NOT NULL,

    StockQuantity INT DEFAULT 0,

    CreatedDate DATETIME2 DEFAULT GETDATE()

);

GO

INSERT INTO Products (ProductName, Category, Price, Brand, StockQuantity) VALUES

-- Electronics Category

('iPhone 15 Pro', 'Electronics', 1199.99, 'Apple', 50),

('Samsung Galaxy S24', 'Electronics', 999.99, 'Samsung', 75),

('MacBook Pro 16"', 'Electronics', 2499.99, 'Apple', 25),

('Dell XPS 13', 'Electronics', 1299.99, 'Dell', 30),

-- Clothing Category

('Nike Air Max', 'Clothing', 129.99, 'Nike', 100),

('Levi''s 501 Jeans', 'Clothing', 89.99, 'Levis', 150),

('Adidas Ultraboost', 'Clothing', 180.00, 'Adidas', 80),

('North Face Jacket', 'Clothing', 249.99, 'North Face', 45),

-- Home & Garden Category

('Dyson V15 Vacuum', 'Home & Garden', 749.99, 'Dyson', 35),

('KitchenAid Mixer', 'Home & Garden', 399.99, 'KitchenAid', 25),

('Instant Pot Pro', 'Home & Garden', 149.99, 'Instant Pot', 60),

('Roomba i7+', 'Home & Garden', 599.99, 'iRobot', 15),

-- Sports Category

('Peloton Bike', 'Sports', 1445.00, 'Peloton', 10),

('Bowflex Dumbbells', 'Sports', 349.99, 'Bowflex', 25),

('Yoga Mat Premium', 'Sports', 79.99, 'Manduka', 150),

('Resistance Bands Set', 'Sports', 29.99, 'Fit Simplify', 200);

GO

SELECT 'Database and table created successfully!' AS Status;

SELECT

    'Total Products: ' + CAST(COUNT(\*) AS VARCHAR(10)) AS Summary

FROM Products;

SELECT

    Category,

    COUNT(\*) AS ProductCount,

    MIN(Price) AS MinPrice,

    MAX(Price) AS MaxPrice

FROM Products

GROUP BY Category

ORDER BY Category;

**//Query:**

USE OnlineRetailStore;

GO

SELECT 'ROW\_NUMBER() - Assigns unique sequential ranks' AS RankingType;

SELECT

    ProductID,

    ProductName,

    Category,

    Price,

    Brand,

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

FROM Products

ORDER BY Category, Price DESC;

GO

SELECT 'RANK() - Leaves gaps after tied ranks' AS RankingType;

SELECT

    ProductID,

    ProductName,

    Category,

    Price,

    Brand,

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

FROM Products

ORDER BY Category, Price DESC;

GO

SELECT 'DENSE\_RANK() - No gaps after tied ranks' AS RankingType;

SELECT

    ProductID,

    ProductName,

    Category,

    Price,

    Brand,

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

FROM Products

ORDER BY Category, Price DESC;

GO

SELECT 'COMPARISON: All Three Ranking Functions' AS ComparisonType;

SELECT

    ProductID,

    ProductName,

    Category,

    Price,

    Brand,

    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

ORDER BY Category, Price DESC;

GO

SELECT 'TOP 3 Products per Category using ROW\_NUMBER()' AS ResultType;

WITH RankedProducts AS (

    SELECT

        ProductID,

        ProductName,

        Category,

        Price,

        Brand,

        StockQuantity,

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

    FROM Products

)

SELECT

    ProductID,

    ProductName,

    Category,

    Price,

    Brand,

    StockQuantity,

    RowNum

FROM RankedProducts

WHERE RowNum <= 3

ORDER BY Category, RowNum;

GO

SELECT 'TOP 3 Products per Category using RANK()' AS ResultType;

WITH RankedProducts AS (

    SELECT

        ProductID,

        ProductName,

        Category,

        Price,

        Brand,

        StockQuantity,

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

    FROM Products

)

SELECT

    ProductID,

    ProductName,

    Category,

    Price,

    Brand,

    StockQuantity,

    RankNum

FROM RankedProducts

WHERE RankNum <= 3

ORDER BY Category, RankNum;

GO

SELECT 'TOP 3 Products per Category using DENSE\_RANK()' AS ResultType;

WITH RankedProducts AS (

    SELECT

        ProductID,

        ProductName,

        Category,

        Price,

        Brand,

        StockQuantity,

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

    FROM Products

)

SELECT

    ProductID,

    ProductName,

    Category,

    Price,

    Brand,

    StockQuantity,

    DenseRankNum

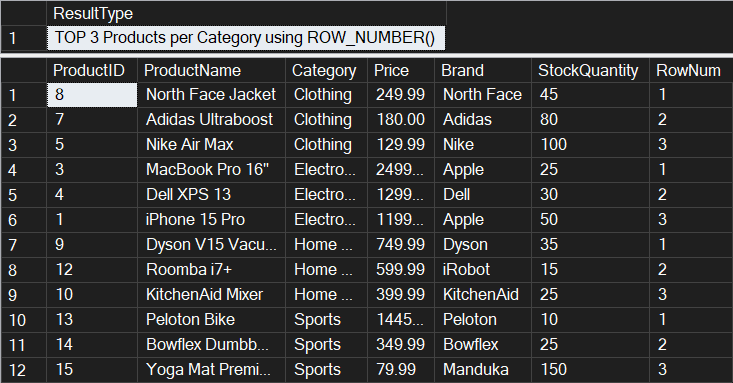
FROM RankedProducts

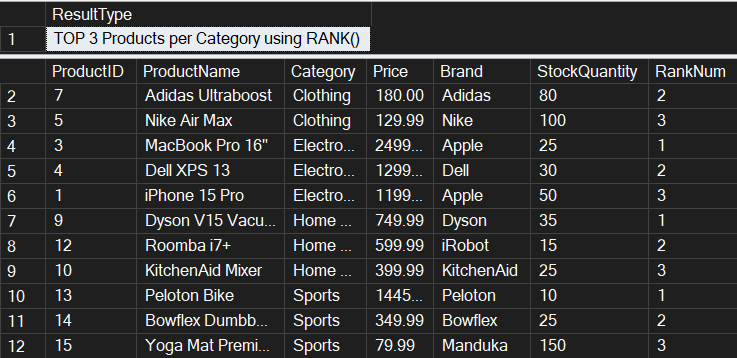
WHERE DenseRankNum <= 3

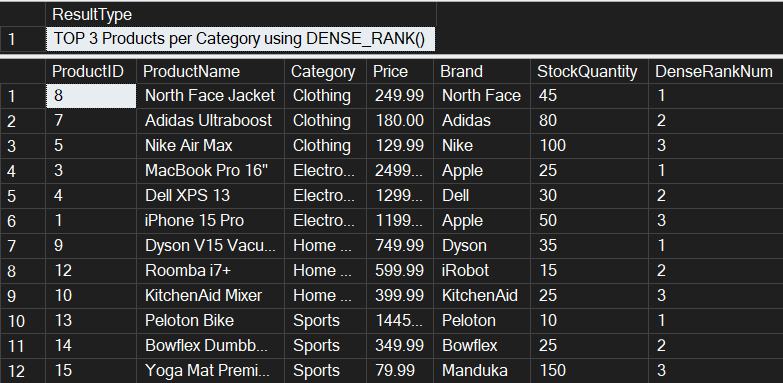
ORDER BY Category, DenseRankNum;

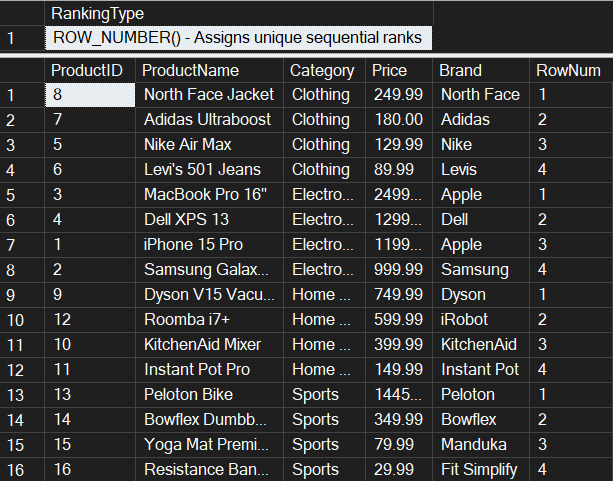
GO

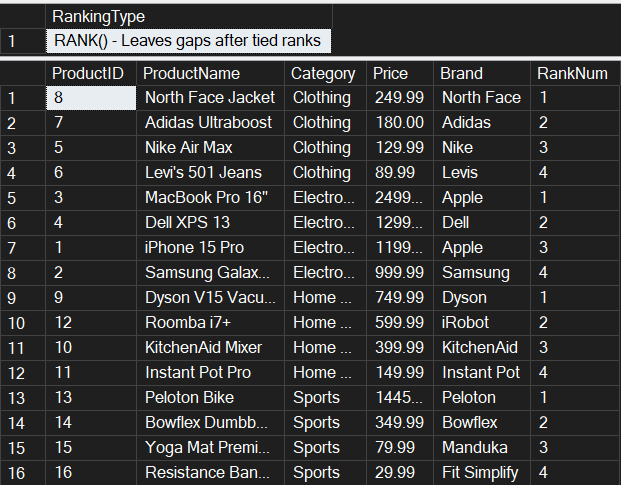
* **Output:**

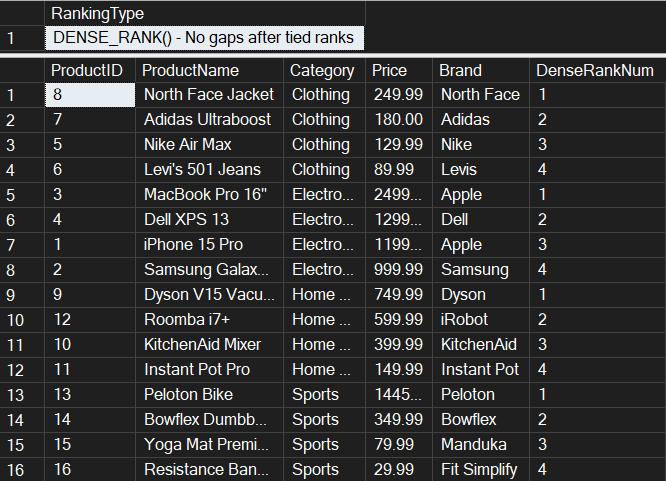












**File 4: Stored procedure**

**//Database Creation:**

CREATE DATABASE EmployeeManagementSystem;

GO

USE EmployeeManagementSystem;

GO

CREATE TABLE Departments (

    DepartmentID INT PRIMARY KEY,

    DepartmentName VARCHAR(100)

);

GO

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

);

GO

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

GO

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'),

(5, 'Robert', 'Wilson', 1, 4800.00, '2020-05-12'),

(6, 'Sarah', 'Brown', 2, 6200.00, '2019-08-14'),

(7, 'David', 'Miller', 3, 7500.00, '2018-12-01'),

(8, 'Lisa', 'Anderson', 1, 5200.00, '2021-02-28'),

(9, 'Kevin', 'Taylor', 3, 6800.00, '2020-09-15'),

(10, 'Maria', 'Garcia', 4, 5800.00, '2021-06-10');

GO

**Exercise-01: Create a Stored Procedure**

* **Code:**

Use EmployeeManagementSystem;

GO

PRINT 'Creating Exercise 1 stored procedures...';

IF OBJECT\_ID('sp\_GetEmployeesByDepartment', 'P') IS NOT NULL

    DROP PROCEDURE sp\_GetEmployeesByDepartment;

GO

IF OBJECT\_ID('sp\_InsertEmployee', 'P') IS NOT NULL

    DROP PROCEDURE sp\_InsertEmployee;

GO

CREATE PROCEDURE sp\_GetEmployeesByDepartment

    @DepartmentID INT

AS

BEGIN

    SET NOCOUNT ON;

    -- Validate input

    IF @DepartmentID IS NULL

    BEGIN

        RAISERROR('DepartmentID cannot be NULL', 16, 1);

        RETURN;

    END

    -- Check if department exists

    IF NOT EXISTS (SELECT 1 FROM Departments WHERE DepartmentID = @DepartmentID)

    BEGIN

        RAISERROR('Department with ID %d does not exist', 16, 1, @DepartmentID);

        RETURN;

    END

    -- Return employee details

    SELECT

        e.EmployeeID,

        e.FirstName,

        e.LastName,

        e.DepartmentID,

        d.DepartmentName,

        e.Salary,

        e.JoinDate

    FROM Employees e

    INNER JOIN Departments d ON e.DepartmentID = d.DepartmentID

    WHERE e.DepartmentID = @DepartmentID

    ORDER BY e.LastName, e.FirstName;

END;

GO

-- Create sp\_InsertEmployee

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

BEGIN

    -- Validate department exists

    IF NOT EXISTS (SELECT 1 FROM Departments WHERE DepartmentID = @DepartmentID)

    BEGIN

        RAISERROR('Department with ID %d does not exist', 16, 1, @DepartmentID);

        RETURN;

    END

    -- Get next available EmployeeID

    DECLARE @NextEmployeeID INT;

    SELECT @NextEmployeeID = ISNULL(MAX(EmployeeID), 0) + 1 FROM Employees;

    -- Insert new employee

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

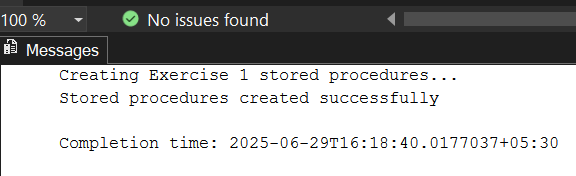
VALUES (@NextEmployeeID, @FirstName, @LastName, @DepartmentID, @Salary, @JoinDate);

END;

GO

PRINT 'Stored procedures created successfully';

* **Output:**



**Exercise 4: Execute a Stored Procedure**

* **Code:**

USE EmployeeManagementSystem;

GO

PRINT '=== EXECUTING STORED PROCEDURES ===';

PRINT '';

IF OBJECT\_ID('sp\_GetEmployeesByDepartment', 'P') IS NULL

BEGIN

    PRINT 'ERROR: sp\_GetEmployeesByDepartment procedure does not exist!';

    PRINT 'Please run Exercise 1 solution first to create the stored procedure.';

    RETURN;

END

ELSE

PRINT 'sp\_GetEmployeesByDepartment procedure found - Ready to execute';

PRINT '';

-- Execute for HR Department (DepartmentID = 1)

PRINT '--- Executing sp\_GetEmployeesByDepartment for HR Department (ID: 1) ---';

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 1;

PRINT '';

-- Execute for Finance Department (DepartmentID = 2)

PRINT '--- Executing sp\_GetEmployeesByDepartment for Finance Department (ID: 2) ---';

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 2;

PRINT '';

-- Execute for IT Department (DepartmentID = 3)

PRINT '--- Executing sp\_GetEmployeesByDepartment for IT Department (ID: 3) ---';

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 3;

PRINT '';

-- Execute for Marketing Department (DepartmentID = 4)

PRINT '--- Executing sp\_GetEmployeesByDepartment for Marketing Department (ID: 4) ---';

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 4;

PRINT '';

PRINT '--- Alternative Execution Methods ---';

PRINT 'Method 1 - Using positional parameter for IT Department:';

EXEC sp\_GetEmployeesByDepartment 3;

PRINT '';

PRINT 'Method 2 - Using EXECUTE keyword for HR Department:';

EXECUTE sp\_GetEmployeesByDepartment @DepartmentID = 1;

PRINT '';

PRINT 'Method 3 - Dynamic execution based on department name:';

-- Get department ID for 'Finance' and execute

DECLARE @DeptID INT;

SELECT @DeptID = DepartmentID FROM Departments WHERE DepartmentName = 'Finance';

PRINT 'Executing for Finance Department (Dynamic ID lookup):';

EXEC sp\_GetEmployeesByDepartment @DepartmentID = @DeptID;

PRINT '';

-- Test with invalid department ID

PRINT 'Testing with invalid Department ID (999):';

BEGIN TRY

    EXEC sp\_GetEmployeesByDepartment @DepartmentID = 999;

END TRY

BEGIN CATCH

    PRINT 'Error caught: ' + ERROR\_MESSAGE();

END CATCH

PRINT '';

-- Test with NULL parameter

PRINT 'Testing with NULL parameter:';

BEGIN TRY

    EXEC sp\_GetEmployeesByDepartment @DepartmentID = NULL;

END TRY

BEGIN CATCH

    PRINT 'Error caught: ' + ERROR\_MESSAGE();

END CATCH

PRINT '';

PRINT '--- Verification: Count of Results per Department ---';

-- Count employees returned by each department execution

SELECT

    'HR' AS Department,

    COUNT(\*) AS EmployeeCount

FROM Employees

WHERE DepartmentID = 1

UNION ALL

SELECT

    'Finance' AS Department,

    COUNT(\*) AS EmployeeCount

FROM Employees

WHERE DepartmentID = 2

UNION ALL

SELECT

    'IT' AS Department,

    COUNT(\*) AS EmployeeCount

FROM Employees

WHERE DepartmentID = 3

UNION ALL

SELECT

    'Marketing' AS Department,

    COUNT(\*) AS EmployeeCount

FROM Employees

WHERE DepartmentID = 4;

PRINT '';

PRINT '--- Advanced Usage: Capturing and Analyzing Results ---';

-- Create a temporary table to store results

IF OBJECT\_ID('tempdb..#TempEmployeeResults') IS NOT NULL

DROP TABLE #TempEmployeeResults;

CREATE TABLE #TempEmployeeResults (

    EmployeeID INT,

    FirstName VARCHAR(50),

    LastName VARCHAR(50),

    DepartmentID INT,

    DepartmentName VARCHAR(100),

    Salary DECIMAL(10,2),

    JoinDate DATE

);

-- Insert results from stored procedure into temp table

INSERT INTO #TempEmployeeResults

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 3;

-- Analyze the captured results

PRINT 'Analysis of IT Department employees (captured from stored procedure):';

SELECT

    COUNT(\*) AS TotalEmployees,

    AVG(Salary) AS AverageSalary,

    MIN(JoinDate) AS EarliestJoinDate,

    MAX(JoinDate) AS LatestJoinDate

FROM #TempEmployeeResults;

-- Show the captured data

SELECT 'Captured Employee Data:' AS Info;

SELECT \* FROM #TempEmployeeResults ORDER BY LastName;

-- Clean up

DROP TABLE #TempEmployeeResults;

PRINT '';

PRINT 'All stored procedure executions completed with results reviewed.';

-- Summary Report

PRINT '';

PRINT '--- SUMMARY REPORT ---';

SELECT

    d.DepartmentName,

    COUNT(e.EmployeeID) AS TotalEmployees,

    'EXEC sp\_GetEmployeesByDepartment @DepartmentID = ' + CAST(d.DepartmentID AS VARCHAR(5)) AS ExecutionCommand

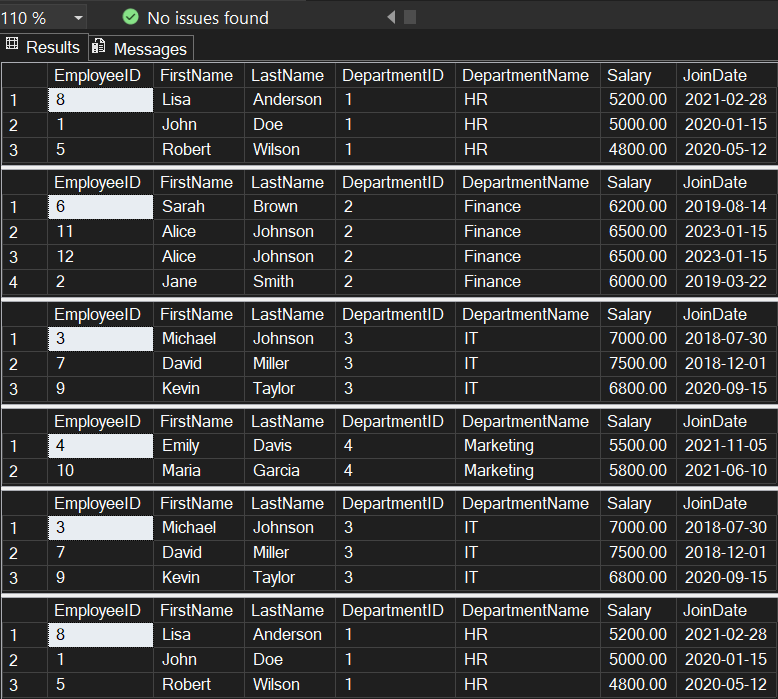
FROM Departments d

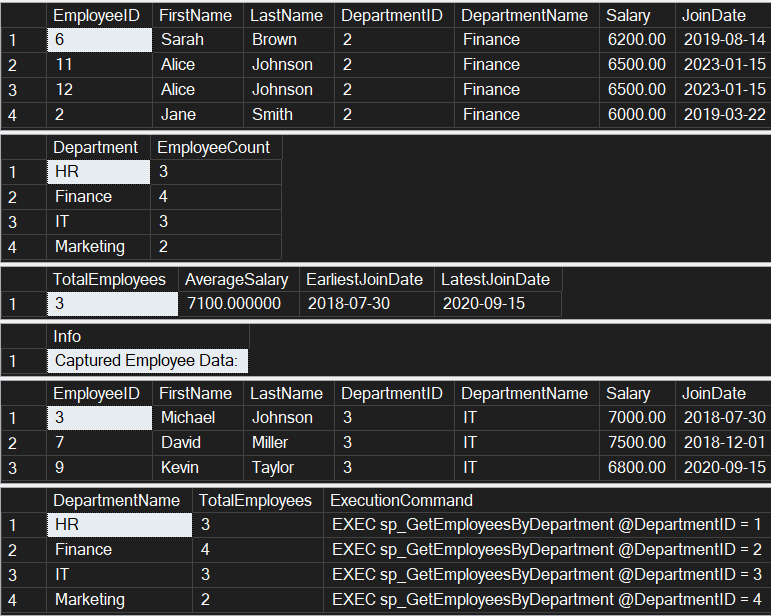
LEFT JOIN Employees e ON d.DepartmentID = e.DepartmentID

GROUP BY d.DepartmentID, d.DepartmentName

ORDER BY d.DepartmentID;

* **Output:**





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

* **Code:**

Use EmployeeManagementSystem;

GO

PRINT 'Creating Exercise 5 Stored Procedure...';

IF OBJECT\_ID('sp\_GetEmployeeCountByDepartment', 'P') IS NOT NULL

    DROP PROCEDURE sp\_GetEmployeeCountByDepartment;

GO

-- Create stored procedure that returns the total number of employees in a department

CREATE PROCEDURE sp\_GetEmployeeCountByDepartment

    @DepartmentID INT

AS

BEGIN

    SET NOCOUNT ON;

    -- Validate input parameter

    IF @DepartmentID IS NULL

    BEGIN

        PRINT 'Error: DepartmentID cannot be NULL';

        RETURN;

    END

    -- Check if department exists

    IF NOT EXISTS (SELECT 1 FROM Departments WHERE DepartmentID = @DepartmentID)

    BEGIN

        PRINT 'Error: Department with ID ' + CAST(@DepartmentID AS VARCHAR(10)) + ' does not exist';

        RETURN;

    END

    -- Declare variables for department info

    DECLARE @DepartmentName VARCHAR(100);

    DECLARE @EmployeeCount INT;

    -- Get department name

    SELECT @DepartmentName = DepartmentName

    FROM Departments

    WHERE DepartmentID = @DepartmentID;

    -- Count employees in the specified department

    SELECT @EmployeeCount = COUNT(\*)

    FROM Employees

    WHERE DepartmentID = @DepartmentID;

    -- Return the count as a result set

    SELECT

        @DepartmentID AS DepartmentID,

        @DepartmentName AS DepartmentName,

        @EmployeeCount AS TotalEmployees;

    -- Also return individual employee details for reference

    SELECT

        'Employee Details:' AS Info,

        e.EmployeeID,

        e.FirstName + ' ' + e.LastName AS FullName,

        e.Salary,

        e.JoinDate

    FROM Employees e

    WHERE e.DepartmentID = @DepartmentID

    ORDER BY e.LastName, e.FirstName;

    -- Print summary message

    PRINT 'Department: ' + @DepartmentName + ' has ' + CAST(@EmployeeCount AS VARCHAR(10)) + ' employees';

END;

GO

PRINT 'Exercise 5 stored procedure created successfully!';

-- TESTING THE STORED PROCEDURES

PRINT '=== TESTING EXERCISE 1 STORED PROCEDURES ===';

-- Test sp\_GetEmployeesByDepartment

PRINT 'Testing sp\_GetEmployeesByDepartment for HR Department (ID: 1):';

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 1;

PRINT '';

PRINT 'Testing sp\_GetEmployeesByDepartment for IT Department (ID: 3):';

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 3;

PRINT '';

PRINT 'Testing sp\_InsertEmployee - Adding a new employee:';

EXEC sp\_InsertEmployee

    @FirstName = 'Alice',

    @LastName = 'Johnson',

    @DepartmentID = 2,

    @Salary = 6500.00,

@JoinDate = '2023-01-15';

PRINT '';

PRINT '=== TESTING EXERCISE 5 STORED PROCEDURE ===';

-- Test sp\_GetEmployeeCountByDepartment for each department

PRINT 'Testing sp\_GetEmployeeCountByDepartment for HR Department (ID: 1):';

EXEC sp\_GetEmployeeCountByDepartment @DepartmentID = 1;

PRINT '';

PRINT 'Testing sp\_GetEmployeeCountByDepartment for Finance Department (ID: 2):';

EXEC sp\_GetEmployeeCountByDepartment @DepartmentID = 2;

PRINT '';

PRINT 'Testing sp\_GetEmployeeCountByDepartment for IT Department (ID: 3):';

EXEC sp\_GetEmployeeCountByDepartment @DepartmentID = 3;

PRINT '';

PRINT 'Testing sp\_GetEmployeeCountByDepartment for Marketing Department (ID: 4):';

EXEC sp\_GetEmployeeCountByDepartment @DepartmentID = 4;

-- ERROR TESTING

PRINT '';

PRINT '=== ERROR TESTING ===';

PRINT 'Testing with invalid department ID (999):';

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 999;

PRINT '';

PRINT 'Testing employee count with invalid department ID (999):';

EXEC sp\_GetEmployeeCountByDepartment @DepartmentID = 999;

-- SUMMARY REPORT

PRINT '';

PRINT '=== FINAL SUMMARY ===';

SELECT 'DEPARTMENT SUMMARY:' AS ReportType;

SELECT

    d.DepartmentID,

    d.DepartmentName,

    COUNT(e.EmployeeID) AS TotalEmployees,

    AVG(e.Salary) AS AverageSalary,

    MIN(e.Salary) AS MinSalary,

    MAX(e.Salary) AS MaxSalary,

    SUM(e.Salary) AS TotalSalaryBudget

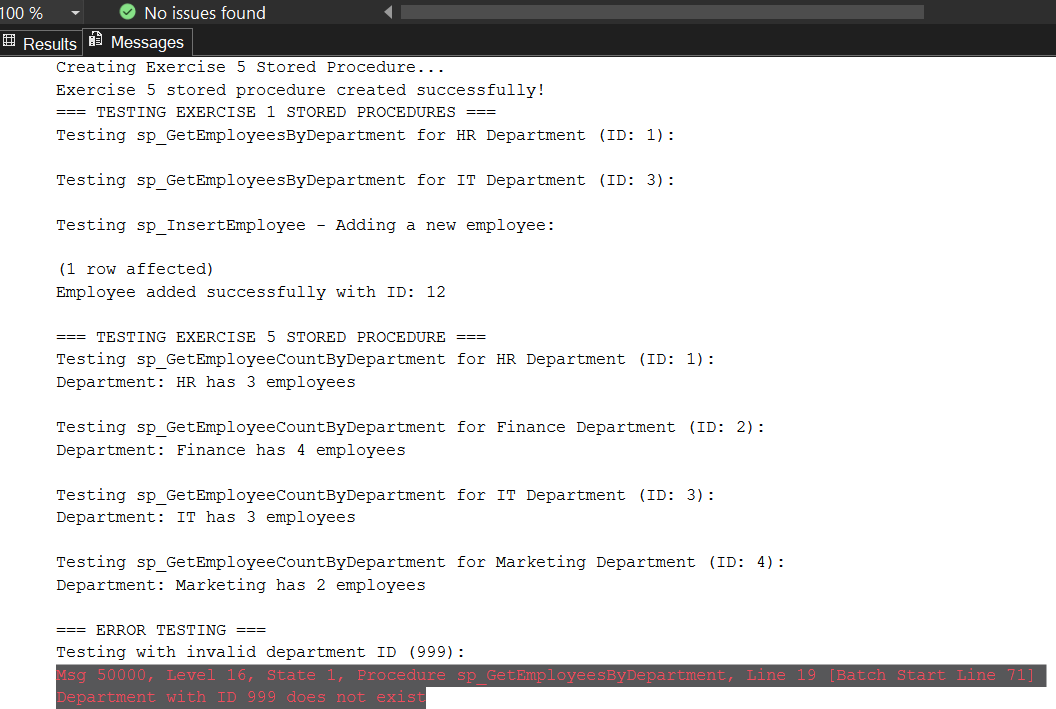
FROM Departments d

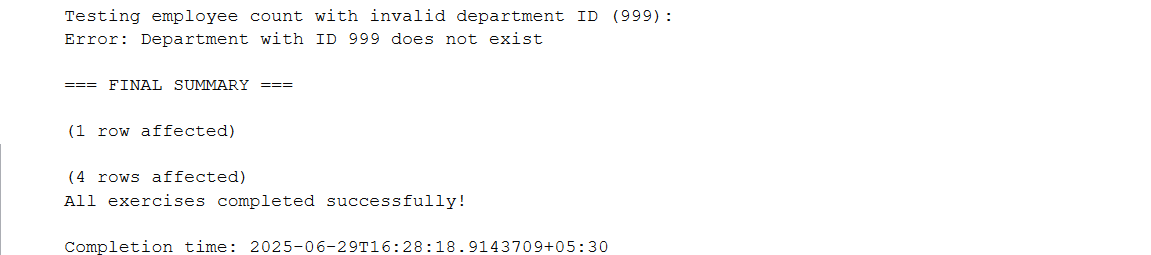
LEFT JOIN Employees e ON d.DepartmentID = e.DepartmentID

GROUP BY d.DepartmentID, d.DepartmentName

ORDER BY d.DepartmentID;

* **Output:**





**File 5: Functions**

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

* **Code:**

USE EmployeeManagementSystem;

GO

-- Drop function if it already exists

IF OBJECT\_ID('dbo.fn\_CalculateAnnualSalary', 'FN') IS NOT NULL

BEGIN

    DROP FUNCTION dbo.fn\_CalculateAnnualSalary;

    PRINT 'Existing function dropped';

END

GO

-- Create the fn\_CalculateAnnualSalary scalar function

CREATE FUNCTION dbo.fn\_CalculateAnnualSalary(@Salary DECIMAL(10,2))

RETURNS DECIMAL(12,2)

AS

BEGIN

    RETURN @Salary \* 12;

END;

GO

PRINT 'fn\_CalculateAnnualSalary function created successfully';

-- Solution 1: Basic query for EmployeeID = 1

SELECT

    EmployeeID,

    FirstName,

    LastName,

    Salary AS MonthlySalary,

    dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees

WHERE EmployeeID = 1;

-- Solution 2: More detailed query with department information

SELECT

    e.EmployeeID,

    e.FirstName + ' ' + e.LastName AS FullName,

    d.DepartmentName,

    e.Salary AS MonthlySalary,

    dbo.fn\_CalculateAnnualSalary(e.Salary) AS AnnualSalary,

    e.JoinDate

FROM Employees e

INNER JOIN Departments d ON e.DepartmentID = d.DepartmentID

WHERE e.EmployeeID = 1;

-- Solution 3: Direct function call with the salary of EmployeeID = 1

SELECT dbo.fn\_CalculateAnnualSalary(5000.00) AS AnnualSalaryForEmployee1;

-- Verification: Test the function with all employees to ensure it works correctly

PRINT 'Verification - Annual salary for all employees:';

SELECT

    e.EmployeeID,

    e.FirstName + ' ' + e.LastName AS EmployeeName,

    d.DepartmentName,

    e.Salary AS MonthlySalary,

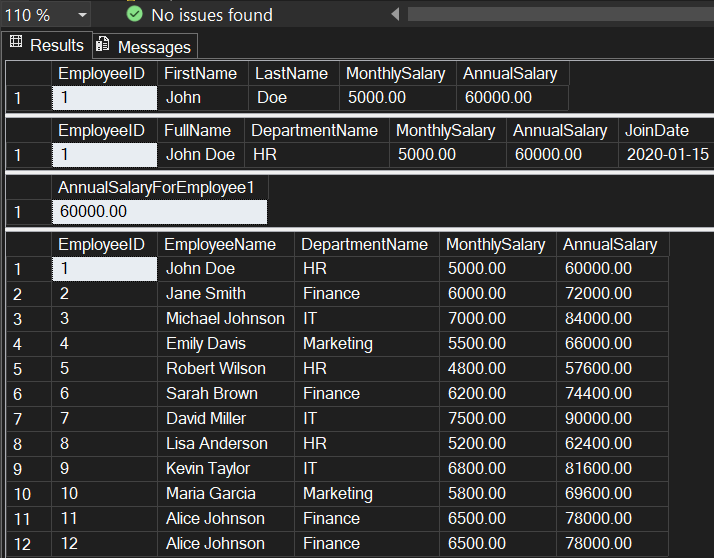
    dbo.fn\_CalculateAnnualSalary(e.Salary) AS AnnualSalary

FROM Employees e

INNER JOIN Departments d ON e.DepartmentID = d.DepartmentID

ORDER BY e.EmployeeID;

* **Output:**



* **Moq**

**1. Write Testable Code with Moq :**

**CustomerCommLib / MailSender.cs**

using System.Net;

using System.Net.Mail;

namespace CustomerCommLib

{

    public interface IMailSender

    {

        bool SendMail(string toAddress, string message);

}

    public class MailSender : IMaiSender

    {

        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()

        {

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

            return true;

        }

    }

}

**CustomerComm.Tests/CustomerCommTests.cs**

using Moq;

using NUnit.Framework;

using CustomerCommLib;

namespace CustomerComm.Tests

{

    [TestFixture]

    public class CustomerCommTests

    {

        private Mock<IMailSender> \_mockMailSender;

        private CustomerCommLib.CustomerComm \_customerComm;

        [OneTimeSetUp]

        public void Setup()

        {

            \_mockMailSender = new Mock<IMailSender>();

            \_mockMailSender.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>())).Returns(true);

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

        }

        [Test]

        public void SendMailToCustomer\_ShouldReturnTrue()

        {

            bool result = \_customerComm.SendMailToCustomer();

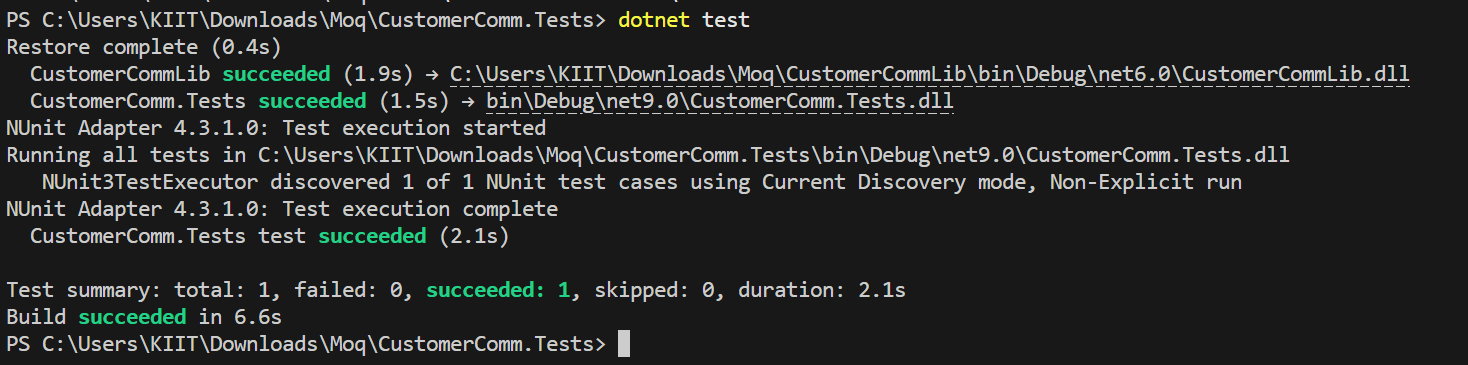
            Assert.IsTrue(result);

        }

    }

}

* **Output:**



1. **Mock file object for Unit Tests :**

**MagicFilesLib/DirectoryExplorer.cs**

using System.Collections.Generic;

using System.IO;

namespace MagicFilesLib

{

    public interface IDirectoryExplorer

    {

        ICollection<string> GetFiles(string path);

}

    public class DirectoryExplorer : IDirectoryExplorer

    {

        public ICollection<string> GetFiles(string path)

        {

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

            return files;

        }

    }

}

**DirectoryExplorer.Tests/DirectoryExplorerTests.cs**

using Moq;

using NUnit.Framework;

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 \_directoryExplorer;

        [OneTimeSetUp]

        public void Setup()

        {

            var mock = new Mock<IDirectoryExplorer>();

            mock.Setup(x => x.GetFiles(It.IsAny<string>())).Returns(new List<string> { \_file1, \_file2 });

            \_directoryExplorer = mock.Object;

        }

        [Test]

        public void GetFiles\_ShouldReturnTwoFiles()

        {

            var files = \_directoryExplorer.GetFiles("dummyPath");

            Assert.IsNotNull(files);

            Assert.AreEqual(2, files.Count);

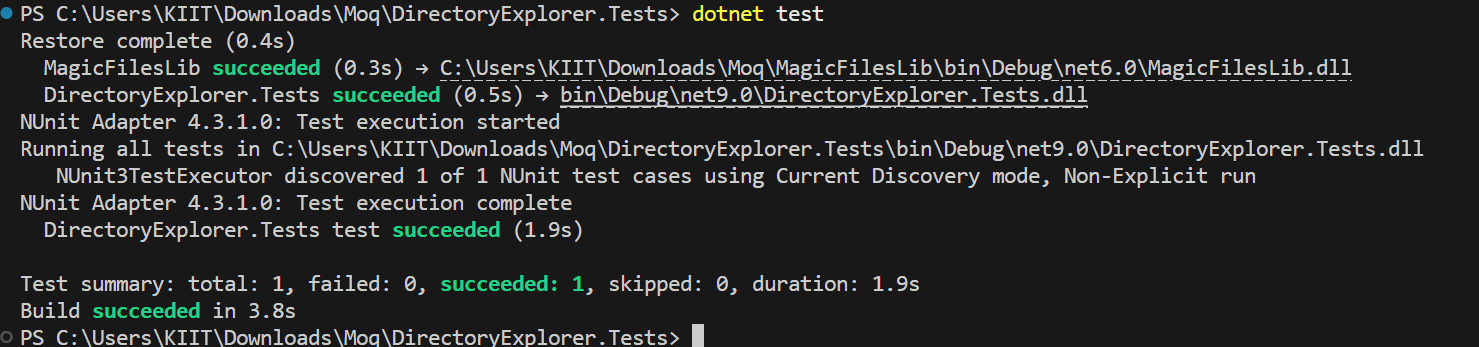
            Assert.Contains(\_file1, (System.Collections.ICollection)files);

        }

    }

}

* **Output:**



1. **Mock database for Unit Tests :**

**PlayersManagerLib/PlayerManager.cs**

using System;

using System.Data.SqlClient;

namespace PlayersManagerLib

{

    public interface IPlayerMapper

    {

        bool IsPlayerNameExistsInDb(string name);

        void AddNewPlayerIntoDb(string name);

}

    public class PlayerMapper : IPlayerMapper

    {

        private readonly string \_connectionString = "Data Source=(local);Initial Catalog=GameDB;Integrated Security=True";

        public bool IsPlayerNameExistsInDb(string name)

        {

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

            {

                conn.Open();

                using (SqlCommand cmd = conn.CreateCommand())

                {

                    cmd.CommandText = "SELECT COUNT(\*) FROM Player WHERE Name = @name";

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

                    return (int)cmd.ExecuteScalar() > 0;

                }

            }

        }

        public void AddNewPlayerIntoDb(string name)

        {

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

            {

                conn.Open();

                using (SqlCommand cmd = conn.CreateCommand())

                {

                    cmd.CommandText = "INSERT INTO Player (Name) VALUES (@name)";

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

                    cmd.ExecuteNonQuery();

                }

            }

        }

}

    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 matches)

        {

            Name = name;

            Age = age;

            Country = country;

            NoOfMatches = matches;

        }

        public static Player RegisterNewPlayer(string name, IPlayerMapper 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);

        }

    }

}

**PlayerManager.Tests/PlayerManagerTests.cs**

using Moq;

using NUnit.Framework;

using PlayersManagerLib;

using System;

namespace PlayerManager.Tests

{

    [TestFixture]

    public class PlayerManagerTests

    {

        private Mock<IPlayerMapper> \_mockMapper;

        [OneTimeSetUp]

        public void Setup()

        {

            \_mockMapper = new Mock<IPlayerMapper>();

            \_mockMapper.Setup(x => x.IsPlayerNameExistsInDb(It.IsAny<string>())).Returns(false);

        }

        [Test]

        public void RegisterNewPlayer\_ShouldReturnPlayerObject()

        {

            var player = Player.RegisterNewPlayer("Dhoni", \_mockMapper.Object);

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

            Assert.AreEqual(23, player.Age);

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

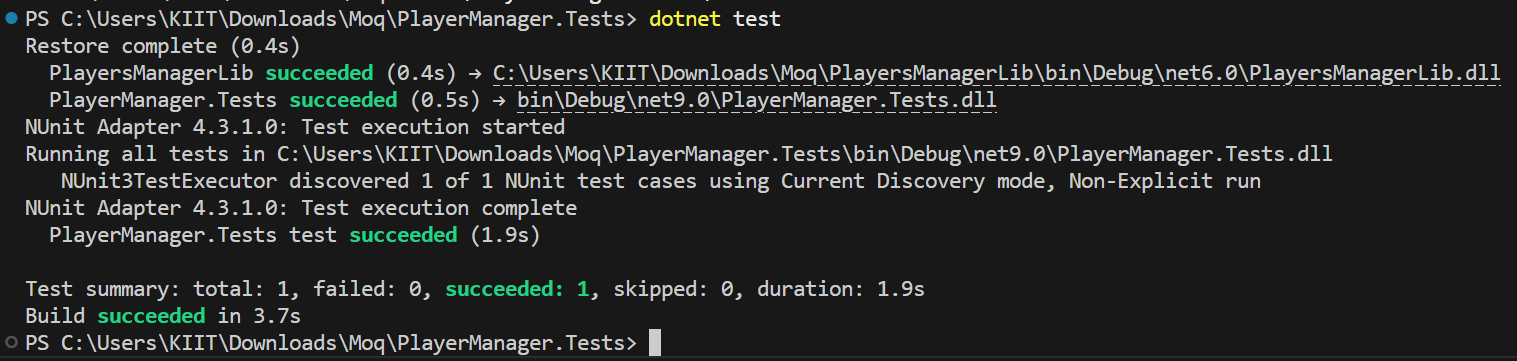
            Assert.AreEqual(30, player.NoOfMatches);

        }

    }

}

* **Output:**



* **NUnit**
* **NUnit Hands-On: Unit Testing Calculator Operations**

**CalcLibrary/Calc.cs**

using System;

namespace CalcLibrary

{

    interface IMathLibrary

    {

        double Addition(double a, double b);

        double Subtraction(double a, double b);

        double Multiplication(double a, double b);

        double Division(double a, double b);

}

    public class SimpleCalculator : IMathLibrary

    {

        double result = 0;

        public double Addition(double a, double b)

        {

            result = a + b;

            return result;

        }

        public double Subtraction(double a, double b)

        {

            result = a - b;

            return result;

        }

        public double Multiplication(double a, double b)

        {

            result = a \* b;

            return result;

        }

        public double Division(double a, double b)

        {

            if (b == 0)

                throw new ArgumentException("Second Parameter Can't be Zero");

            result = a / b;

            return result;

        }

        public void AllClear()

        {

            result = 0;

        }

        public double GetResult

        {

            get { return result; }

        }

    }

}

**CalcLibrary.Tests/CalculatorTests.cs**

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalcLibrary.Tests

{

    [TestFixture]

    public class CalculatorTests

    {

        private SimpleCalculator \_calc;

        [SetUp]

        public void Setup()

        {

            \_calc = new SimpleCalculator();

        }

        [TestCase(2, 3, 5)]

        [TestCase(-1, -2, -3)]

        public void Addition\_ShouldReturnCorrectSum(double a, double b, double expected)

        {

            var result = \_calc.Addition(a, b);

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

        }

        [TestCase(5, 3, 2)]

        [TestCase(0, -2, 2)]

        public void Subtraction\_ShouldReturnCorrectResult(double a, double b, double expected)

        {

            var result = \_calc.Subtraction(a, b);

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

        }

        [TestCase(4, 2, 8)]

        [TestCase(-1, 5, -5)]

        public void Multiplication\_ShouldReturnCorrectResult(double a, double b, double expected)

        {

            var result = \_calc.Multiplication(a, b);

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

        }

        [TestCase(10, 2, 5)]

        public void Division\_ShouldReturnCorrectResult(double a, double b, double expected)

        {

            var result = \_calc.Division(a, b);

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

        }

        [Test]

        public void Division\_ByZero\_ShouldThrowException()

        {

            Assert.Throws<ArgumentException>(() => \_calc.Division(5, 0));

        }

        [TearDown]

        public void Cleanup()

        {

            \_calc = null;

        }

    }

}

* **Output:**

