**WEEK 2**

**ADVANCED SQL**

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

Use ROW\_NUMBER(), RANK(), DENSE\_RANK(), OVER(), and PARTITION BY.

create table Students(sno int, Sname varchar(50), Sid int, Course varchar(50), Smarks int);

insert into Students(sno, Sname, Sid, Course, Smarks) values (1, 'Gayu', 225, 'Computer Science', 95),

(2, 'Avi', 226, 'Biology', 94),

(3, 'Bhava', 227, 'Biology', 92),

(4, 'Ashmi', 228, 'Computer Science', 92),

(5, 'Afrin', 229, 'Computer Science', 86),

(6, 'Abi', 230, 'Biology', 89);

select\*from Students;

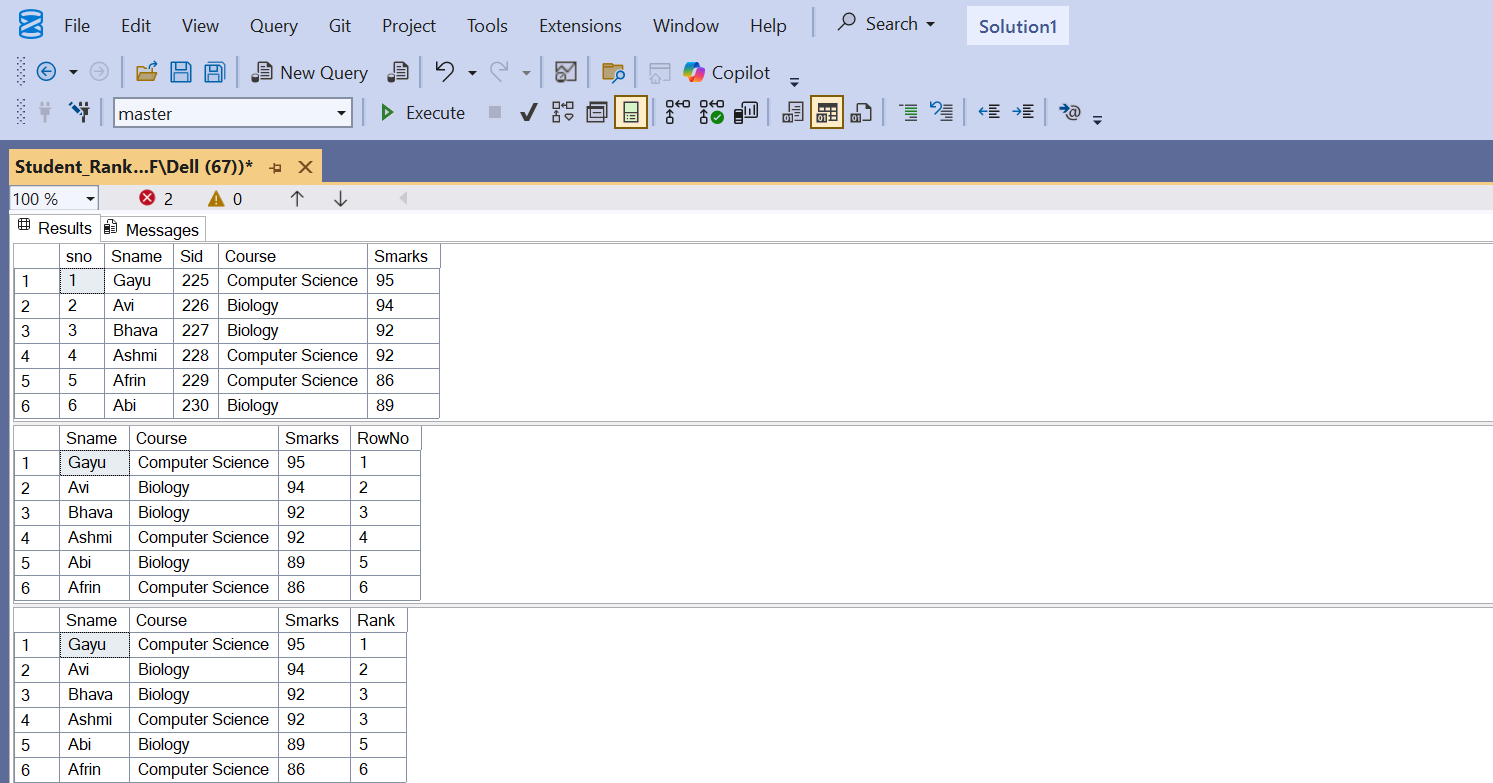
select Sname, Course, Smarks, row\_number() over(ORDER BY Smarks DESC) as RowNo from Students;

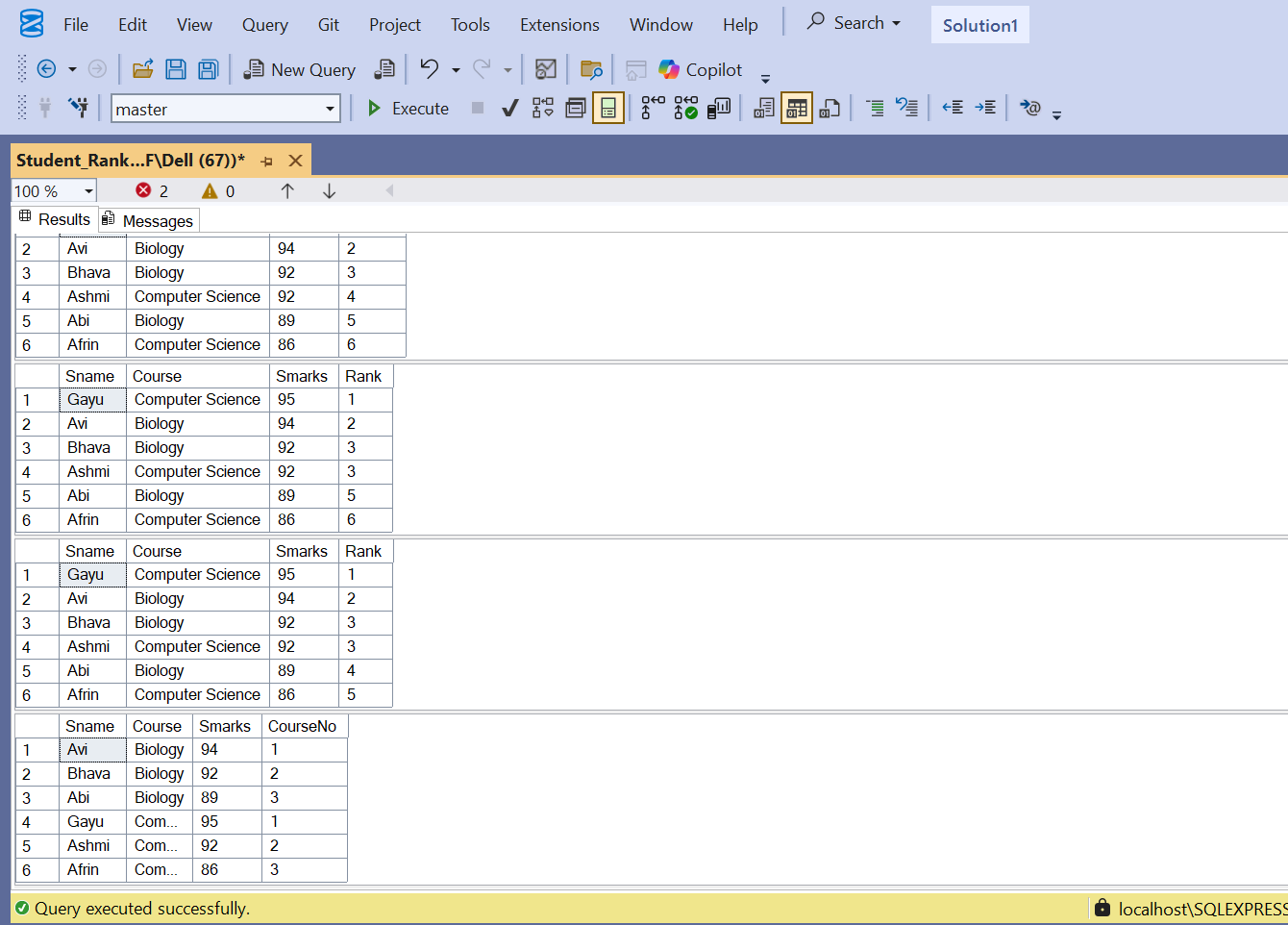
select Sname, Course, Smarks, RANK() over(ORDER BY Smarks DESC) as 'Rank' from Students;

select Sname, Course, Smarks, DENSE\_RANK() over(ORDER BY Smarks DESC) as 'Rank' from Students;

select Sname, Course, Smarks, ROW\_NUMBER() over (PARTITION BY Course ORDER BY Smarks DESC) as CourseNo from Students;

OUTPUT:





**TASK 2: Exercise 2: Create A Stored Procedure**

Create a stored procedure to retrieve employee details by department.

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

DROP PROCEDURE sp\_InsertEmployee;

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

DROP PROCEDURE getbydept;

IF OBJECT\_ID('Employee', 'U') IS NOT NULL

DROP TABLE Employee;

IF OBJECT\_ID('Dept', 'U') IS NOT NULL

DROP TABLE Dept;

CREATE TABLE Dept (

DeptID INT PRIMARY KEY,

Deptname VARCHAR(100)

);

CREATE TABLE Employee (

EmpID INT IDENTITY(1,1) PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DeptID INT,

Salary DECIMAL(10,2),

JoinDate DATE,

FOREIGN KEY (DeptID) REFERENCES Dept(DeptID)

);

INSERT INTO Dept (DeptID, Deptname)

VALUES

(1,'HR'),

(2,'Finance'),

(3,'IT'),

(4,'Marketing');

INSERT INTO Employee (FirstName, LastName, DeptID, Salary, JoinDate)

VALUES

('John', 'Doe', 1, 15000.00, '2020-01-15'),

('Jane', 'Smith', 2, 26000.00, '2019-03-22'),

('Michael', 'Johnson', 3, 37000.00, '2018-07-30'),

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

GO

CREATE PROCEDURE getbydept

@DeptID INT

AS

BEGIN

SELECT \* FROM Employee

WHERE DeptID = @DeptID;

END;

GO

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

BEGIN

INSERT INTO Employee (FirstName, LastName, DeptID, Salary, JoinDate)

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

END;

GO

EXEC getbydept @DeptID = 3;

EXEC sp\_InsertEmployee

@FirstName = 'Anne',

@LastName = 'Jason',

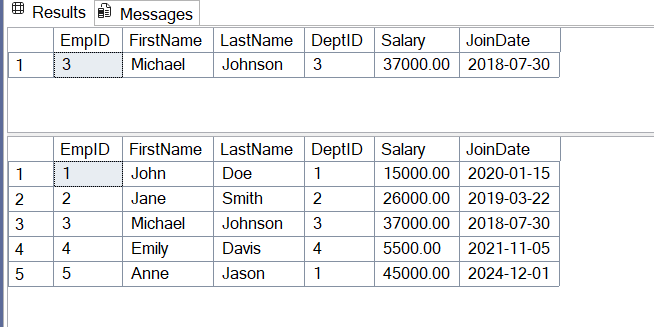
@DepartmentID = 1,

@Salary = 45000.00,

@JoinDate = '2024-12-01';

SELECT \* FROM Employee;

OUTPUT:



**TASK 3: Exercise 5: Return Data from Stored Procedure:**

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

Go

CREATE PROCEDURE GetEmployeeCountByDept

@DeptID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

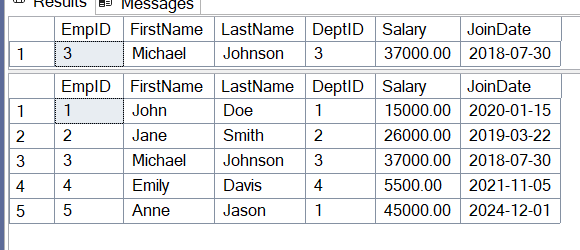
FROM Employee

WHERE DeptID = @DeptID;

END;

EXEC GetEmployeeCountByDept @DeptID = 2;

OUTPUT:



**NUNIT**

**TestFixture & Test**

* 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.

**CalculatorTests.cs**

using CalcLibrary;

using NUnit.Framework;

using System;

using NUnit.Framework;

using CalcLibrary;

namespace CalcLibraryTests

{

[TestFixture]

public class CalculatorTests

{

private Calculator calc;

[SetUp]

public void Init()

{

calc = new Calculator();

}

[TearDown]

public void Cleanup()

{

calc = null;

}

[TestCase(10, 5, 15)]

[TestCase(-3, 3, 0)]

public void Add\_TestCases(int a, int b, int expected)

{

var result = calc.Add(a, b);

TestContext.WriteLine($"Addition Result: {a} + {b} = {result}");

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

}

[TestCase(10, 5, 5)]

public void Subtract\_TestCases(int a, int b, int expected)

{

var result = calc.Subtract(a, b);

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

}

[TestCase(2, 3, 6)]

public void Multiply\_TestCases(int a, int b, int expected)

{

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

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

}

[TestCase(10.0, 2.0, 5.0)]

public void Divide\_TestCases(double a, double b, double expected)

{

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

Assert.That(result, Is.EqualTo(expected).Within(0.0001));

}

[Test]

public void Divide\_ByZero\_ThrowsException()

{

Assert.Throws<DivideByZeroException>(() => calc.Divide(10, 0));

}

}

}

**CalcLibrary.cs**

using System;

namespace CalcLibrary

{

public class Calculator

{

public int Add(int a, int b) => a + b;

public int Subtract(int a, int b) => a - b;

public int Multiply(int a, int b) => a \* b;

public double Divide(double a, double b)

{

if (b == 0) throw new DivideByZeroException("Cannot divide by zero.");

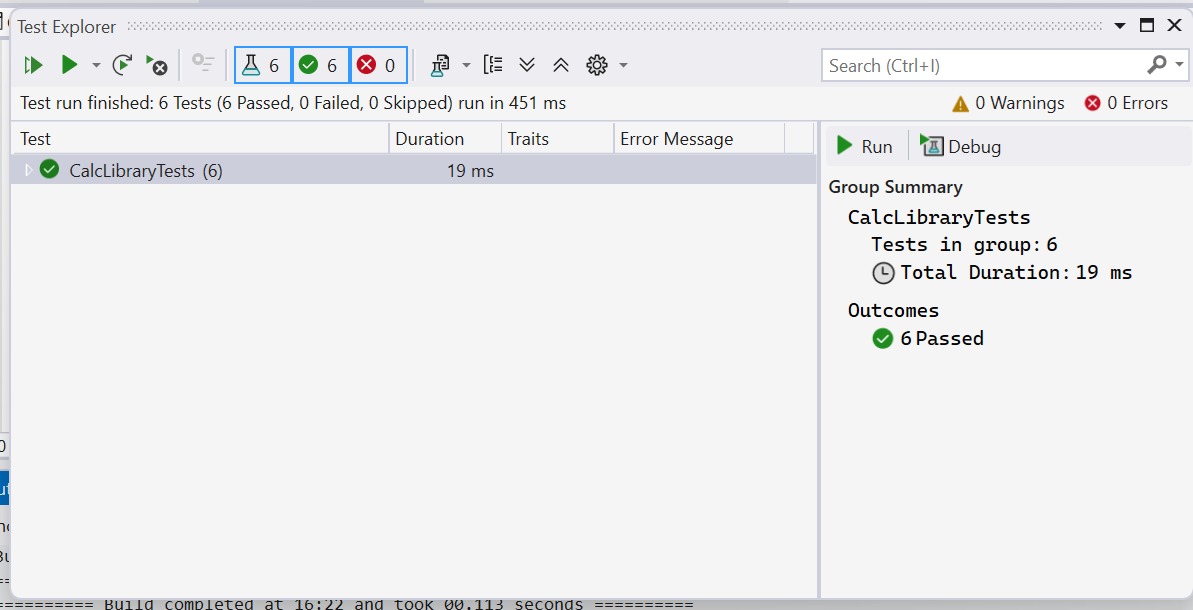
return a / b;

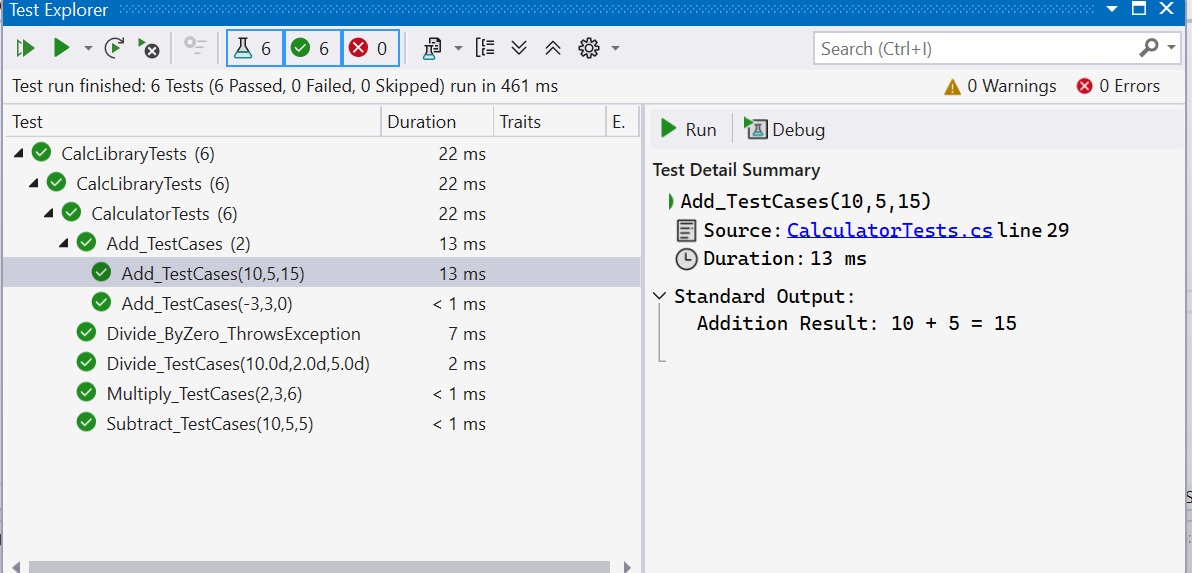
}

}

}

OUTPUT:





**Moq**

**Task-1: Write Testable Code with Moq**

* Create a **Class Library (Language C#)** project using Visual Studio IDE, and name it as **CustomerCommLib.**
* Rename the default **Class1** class name as **MailSender.**
* Include the following namespaces with ‘using’ directive.
  + **System.Net**
  + **System.Net.Mail**

**CustomerComm.cs**

namespace CustomerCommLib

{

public class CustomerComm

{

private IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

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

}

}

}

**MainSender.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CustomerCommLib

{

public class Class1

{

}

}

**IMailSender.cs**

namespace CustomerCommLib

{

public interface IMailSender

{

bool SendMail(string toAddress, string message);

}

}

**Task 2:** Create a new class library project called **CustomerComm.Tests** and add the following external dependencies to it using **NuGet Package Manager.**

* + NUnit
  + NUnit Test Adapter
  + Moq

**CustomerCommTests.cs**

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerCommLib.Tests

{

[TestFixture]

public class CustomerCommTests

{

private Mock<IMailSender> mockMailSender;

private CustomerComm customerComm;

[OneTimeSetUp]

public void Init()

{

mockMailSender = new Mock<IMailSender>();

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

.Returns(true);

customerComm = new CustomerCommLib.CustomerComm(mockMailSender.Object);

}

[TestCase]

public void SendMailToCustomer\_ShouldReturnTrue\_WhenMailIsSent()

{

// Act

var result = customerComm.SendMailToCustomer();

// Assert

Assert.That(result, Is.True);

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

}

}

}

OUTPUT:

