**WEEK 1 HANDS ON**

Submitted by:Simran Raghav(6362006)

**ADVANCED SQL**

**Exercise 1.1:Ranking and Window Functions**

**A)using row\_number()**

QUERY:

WITH ordered\_prod AS

(SELECT ProductID,

ProductName,

Category,

Price,

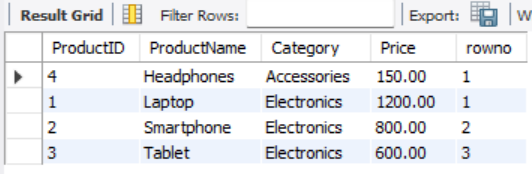
ROW\_NUMBER() OVER (PARTITION BY CATEGORY ORDER BY PRICE DESC) AS rowno

FROM PRODUCTS

)

SELECT \* FROM ordered\_prod WHERE rowno <=3;

OUTPUT:



**B) -using rank()**

QUERY:

WITH ordered\_prod AS

(SELECT ProductID,

ProductName,

Category,

Price,

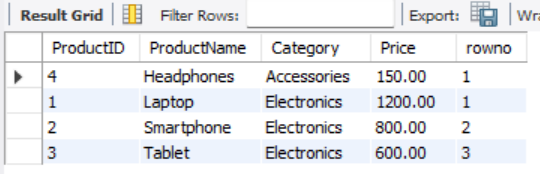
RANK() OVER (PARTITION BY CATEGORY ORDER BY PRICE DESC) AS rowno

FROM PRODUCTS

)

SELECT \* FROM ordered\_prod WHERE rowno <=3;

OUTPUT:



**C)-using dense\_rank()**

QUERY:

WITH ordered\_prod AS

(SELECT ProductID,

ProductName,

Category,

Price,

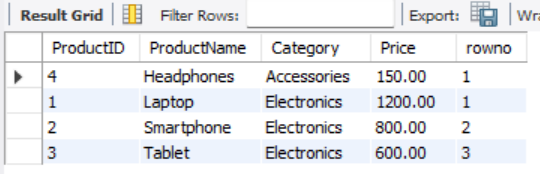
DENSE\_RANK() OVER (PARTITION BY CATEGORY ORDER BY PRICE DESC) AS rowno

FROM PRODUCTS

)

SELECT \* FROM ordered\_prod WHERE rowno <=3;

OUTPUT:



**Note:**

→ There are **no ties** (i.e., no products in the **same category** with the **same price**).,hence all 3 return identical values.However, when ties exist:

* ROW\_NUMBER() assigns a **unique**, sequential number regardless of ties.
* RANK() gives the same rank to tied values but **skips** subsequent ranks.
* DENSE\_RANK() also gives the same rank to tied values but **does not skip** ranks.

This distinction affects how many rows are returned when filtering by top n ranks.

**Exercise 4.1: Create a Stored Procedure**

**A)stored procedure to retrieve employee details by department**

QUERY:

CREATE PROCEDURE dbo.sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

SET NOCOUNT ON;

SELECT e.EmployeeID,

e.FirstName,

e.LastName,

d.DepartmentName,

e.Salary,

e.JoinDate

FROM dbo.Employees AS e

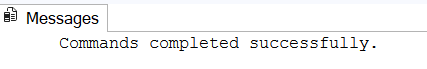
JOIN dbo.Departments AS d

ON d.DepartmentID = e.DepartmentID

WHERE e.DepartmentID = @DepartmentID;

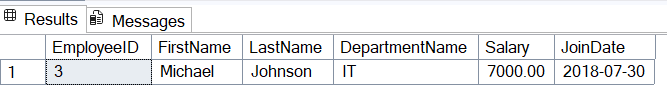
END;

OUTPUT:



**Optional check:**

EXEC dbo.sp\_GetEmployeesByDepartment @DepartmentID = 3;



**B)stored procedure named `sp\_InsertEmployee`**

QUERY:

CREATE PROCEDURE dbo.sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

BEGIN

SET NOCOUNT ON;

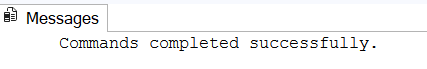
INSERT INTO dbo.Employees

(FirstName, LastName, DepartmentID, Salary, JoinDate)

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

END;

OUTPUT:



**Exercise 4.2: Return data from Stored Procedure**

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

**department.**

QUERY:

CREATE PROCEDURE dbo.sp\_GetEmployeeCountByDepartment

@DepartmentID INT

AS

BEGIN

SET NOCOUNT ON;

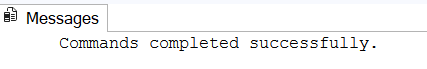
SELECT COUNT(\*) AS EmployeeCount

FROM dbo.Employees

WHERE DepartmentID = @DepartmentID;

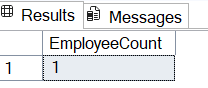
END;

OUTPUT:



**Optional check:**

EXEC dbo.sp\_GetEmployeesByDepartment @DepartmentID = 3;



**NUNIT AND MOQ**

**Exercise 1:NUnit**

**- testing program to validate a calculator addition operation**

CODE:

[1 . Production Code (already supplied in CalcLibrary)]

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; }

}

}

}

[2. Unit Test Project Code (created - CalcLibrary. Tests)]

using NUnit.Framework;

using CalcLibrary;

namespace CalcLibrary.Tests

{

[TestFixture]

public class SimpleCalculatorTests

{

private SimpleCalculator calc;

[SetUp]

public void Setup()

{

calc = new SimpleCalculator();

}

[TearDown]

public void TearDown()

{

calc = null;

}

[Test]

[TestCase(2, 3, 5)]

[TestCase(-1, 1, 0)]

[TestCase(0, 0, 0)]

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

{

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

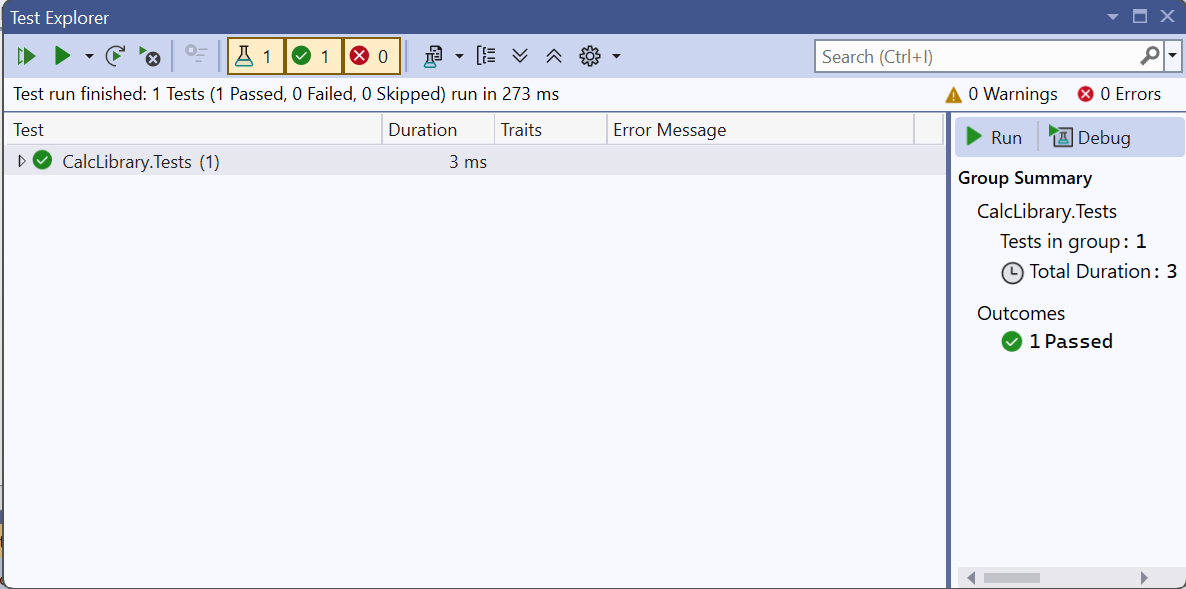
Assert.That(result, Is.EqualTo(expected));

}

}

}

OUTPUT:



**Exercise 2:MOQ**

**- write testable code with MOQ**

CODE:

[1a . Production Code (MailSender.cs)]

using System.Net;

using System.Net.Mail;

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 smtp = 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;

smtp.Port = 587;

smtp.Credentials = new NetworkCredential("username", "password");

smtp.EnableSsl = true;

smtp.Send(mail);

return true;

}

}

}

[1b . Production Code (CustomerComm.cs)]

namespace CustomerCommLib

{

public class CustomerComm

{

private readonly IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

string email = "cust123@abc.com";

string message = "Some Message";

return \_mailSender.SendMail(email, message);

}

}

}

[2 . Unit Test Code (CustomerCommTests.cs)]

using NUnit.Framework;

using Moq;

using Assert = NUnit.Framework.Assert;

namespace CustomerCommLib

{

[TestFixture]

public class CustomerCommTests

{

[Test]

public void SendMailToCustomer\_ShouldCallSendMailOnce\_AndReturnTrue()

{

var mockSender = new Mock<IMailSender>();

mockSender

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

.Returns(true);

var comm = new CustomerComm(mockSender.Object);

var result = comm.SendMailToCustomer();

Assert.IsTrue(result);

mockSender.Verify(

m => m.SendMail("cust123@abc.com", "Some Message"),

Times.Once);

}

}

}

OUTPUT:

