**Advanced SQL Hands-on**

**Exercise1:Ranking and Window Functions**

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

Scenario:

Find the top 3 most expensive products in each category using different ranking functions.

Steps:

1. Use ROW\_NUMBER() to assign a unique rank within each category.

2. Use RANK() and DENSE\_RANK() to compare how ties are handled.

3. Use PARTITION BY Category and ORDER BY Price DESC.

SELECT

ProductName,

Category,

Price,

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 DenseRank

FROM Products;

WITH RankedProducts AS (

SELECT

ProductName,

Category,

Price,

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

FROM Products

)

SELECT \*

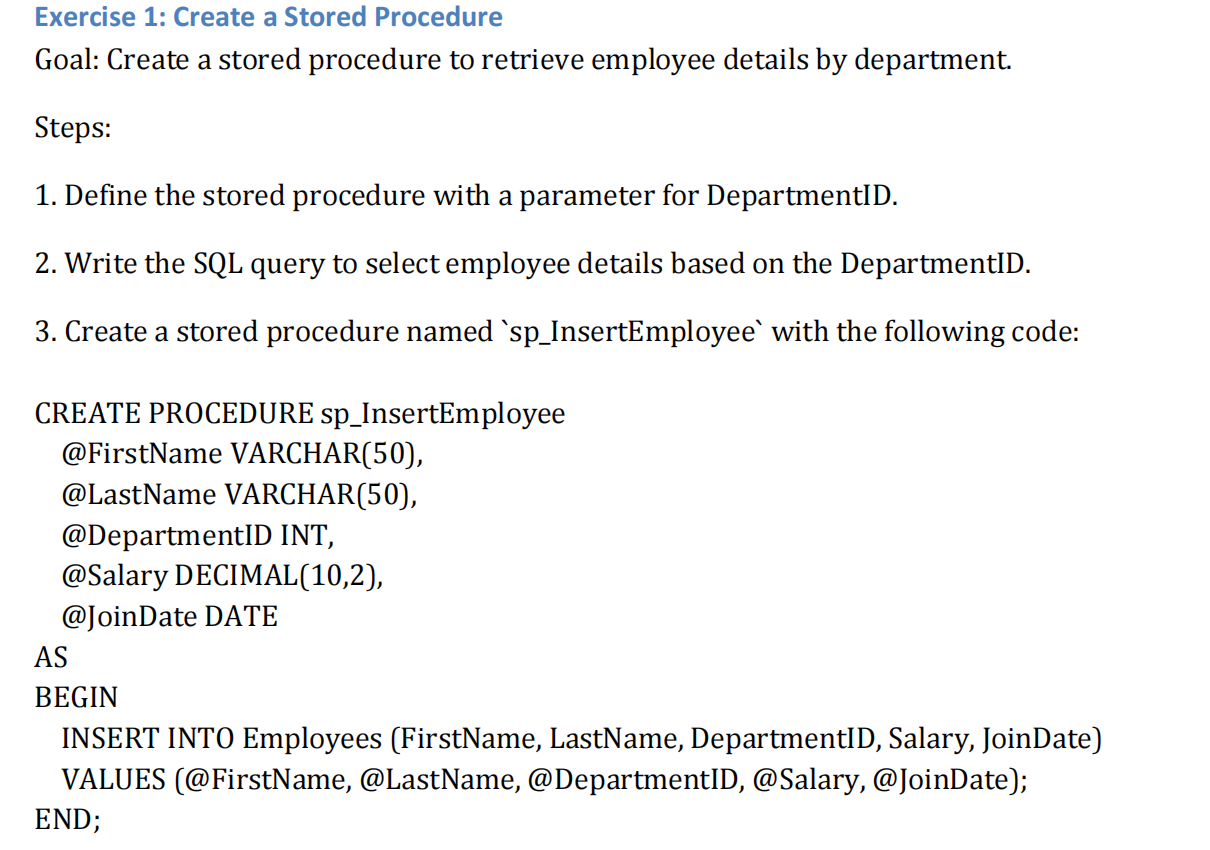
FROM RankedProducts

WHERE RowNum <= 3;

Output:



**SQL Exercise: Stored Concepts**



SQL QUERY:

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,

Salary DECIMAL(10,2),

JoinDate DATE,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

INSERT INTO Departments VALUES

(1,'HR'),(2,'Finance'),(3,'IT'),(4,'Marketing');

INSERT INTO Employees 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 sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary, JoinDate

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

GO

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 2;

OUTPUT:



1. **Unit Hands-on**

**1.N-unit**

**TestFixture & Test**

Please download the application available [here](https://cognizantonline.sharepoint.com/:u:/r/sites/GTP-Solutions/Gencsharepath/Shared%20Documents/Internship2020/FSE/DotNet/02%20-%20NUnit,%20C%23%204.5,%20ASP.Net%20Core/Handson/CalcLibrary.zip?csf=1&e=aLxB66). This will be used to write Unit test cases  
  
Follow the steps listed below to write the NUnit test cases for the application.

* 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

using NUnit.Framework;

namespace UnitTestProject1

{

[TestFixture]

public class CalculatorTests

{

private Clalculator calc;

[SetUp]

public void SetUp()

{

calc = new Clalculator();

}

[TearDown]

public void TearDown()

{

calc = null;

}

[Test]

[TestCase(2, 3, 5)]

[TestCase(-1, -1, -2)]

[TestCase(0, 0, 0)]

[TestCase(-5, 10, 5)]

public void Add\_AddsTwoNumbers\_ReturnsExpectedResult(int a, int b, int expected)

{

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

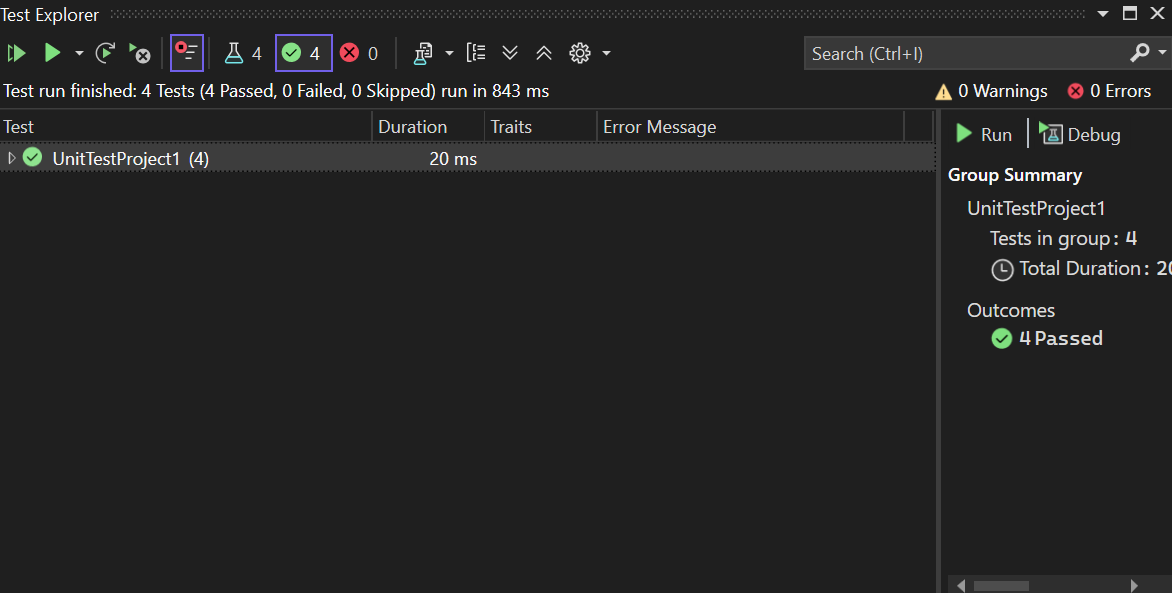
Assert.That(result, Is.EqualTo(expected));

}

}

}

Output:



1.Write Testable Code with Moq

## **Scenario**

You are tasked to write a unit test code for the below scenario.

The application in which you are teamed up with, deals with a mail server communication in which your application tries to send mail to its users upon every transaction. Your role is to write unit testing the module that contains send mail functionality. You wanted to perform testing the module without sending any email.

After investigating the problem scenario, you found a solution and that is creating **mock** objects of these external dependencies in the unit testing project so that you can achieve speedier test execution and loose coupling of code.

**Note:** Duration to complete this exercise is **30 min**.

## **Task1**

In this task, you will create a class library that will be used for unit testing.

* 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**
* Define an interface as follow.

public interface IMailSender

{

        bool SendMail(string toAddress, string message);

}

* And provide implementation of **IMailSender** in the **MailSender** class as seen below.

namespace CustomerCommLib

{

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);

}

}

}

The above class can’t be unit testing since the code access the STMP mail server.

* Create another class called **CustomeComm** which is the **class under test** in the given scenario.

namespace CustomerCommLib

{

public class CustomerComm

{

IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender=mailSender;

}

public bool SendMailToCustomer()

{

//Actual logic goes here

//define message and mail address

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

return true;

}

}

}

In the above code we **injected the dependency** (IMailSender) through **constructor** of **CustomerComm** class so that we can **pass the mock object** of the dependency wherever it is necessary.

We have successfully created a class that's written in such a way that we can run a unit test against it and an exception won't be thrown. We achieve this by mocking the call to IMailSender.SendMail() and adding a mocked return value of true to it.

* Finally **build** your project and be ready for the unit testing with NUnit and Moq.

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerCommLib.Tests

{

[TestFixture]

public class CustomerCommTests

{

[Test]

public void SendMailToCustomer\_ShouldReturnTrue\_WhenMailSentSuccessfully()

{

var mockMailSender = new Mock<IMailSender>();

mockMailSender

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

.Returns(true);

var customerComm = new CustomerComm(mockMailSender.Object);

var result = customerComm.SendMailToCustomer();

Assert.That(result, Is.True);

}

}

}

Output:

