**Exercise 1: 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.

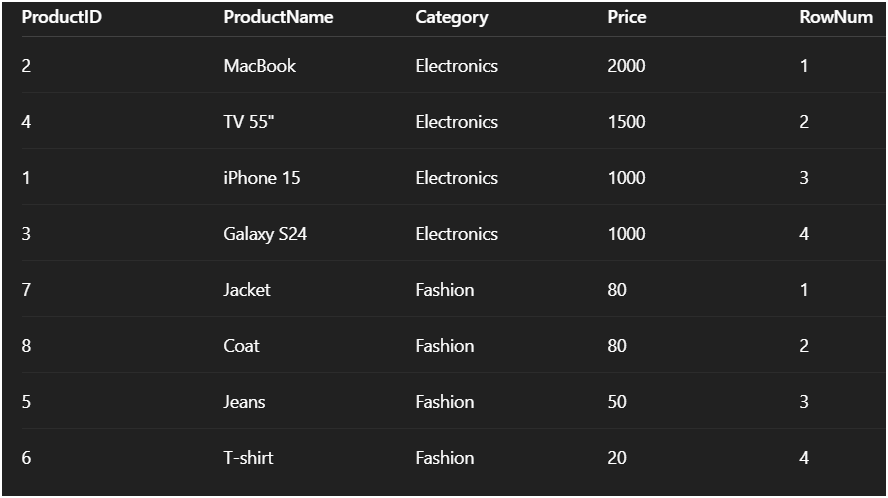
3. Use PARTITION BY Category and ORDER BY Price DESC

SELECT \*,

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

FROM Products;

Output:



**Exercise 1: Create a Stored Procedure**

Goal: Create a stored procedure to retrieve employee details by department.

Steps:

1. Define the stored procedure with a parameter for DepartmentID.

2. Write the SQL query to select employee details based on the DepartmentID.

3. Create a stored procedure named `sp\_InsertEmployee` with the following code: 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;

-- 1. Create Employees table (if not already exists)

CREATE TABLE Employees (

EmployeeID INT IDENTITY PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT,

Salary DECIMAL(10,2),

JoinDate DATE

);

-- 2. Create procedure to insert an employee

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;

-- 3. Create procedure to retrieve employees by department

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

SELECT

EmployeeID,

FirstName,

LastName,

DepartmentID,

Salary,

JoinDate

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

Example: Use the Procedures:

-- Insert example

EXEC sp\_InsertEmployee

@FirstName = 'John',

@LastName = 'Doe',

@DepartmentID = 2,

@Salary = 60000.00,

@JoinDate = '2024-05-01';

-- Retrieve example

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 2;

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

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

Steps:

1. Define the stored procedure with a parameter for DepartmentID.

2. Write the SQL query to count the number of employees in the specified department.

3. Save the stored procedure by executing the Stored procedure content.

**Exercise 6: Use Output Parameters in a Stored Procedure**

Goal: Create a stored procedure that returns the total salary of employees in a department using an output parameter.

-- Exercise 5: Stored Procedure to count employees in a department

CREATE PROCEDURE sp\_GetEmployeeCountByDepartment

@DepartmentID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

-- Exercise 6: Stored Procedure with OUTPUT parameter to get total salary

CREATE PROCEDURE sp\_GetTotalSalaryByDepartment

@DepartmentID INT,

@TotalSalary DECIMAL(18, 2) OUTPUT

AS

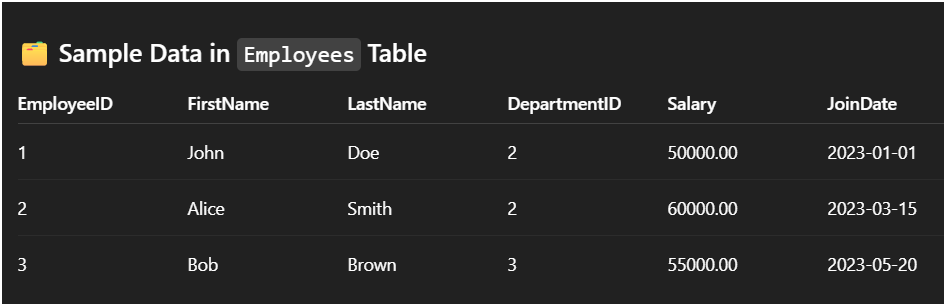
BEGIN

SELECT @TotalSalary = SUM(Salary)

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;



Nunits:

using NUnit.Framework;

using CalcLibrary;

namespace CalcTests

{

[TestFixture]

public class CalculatorTests

{

private Calculator calc;

[SetUp]

public void Setup()

{

calc = new Calculator();

}

[TearDown]

public void TearDown()

{

}

[Test]

[TestCase(2, 3, 5)]

[TestCase(0, 0, 0)]

[TestCase(-1, 1, 0)]

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

{

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

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

}

[Test]

[Ignore("This test is under development")]

public void Subtract\_ShouldBeImplemented()

{

}

}

}

Test Run Successful.

Total tests: 4

Passed: 3

Skipped: 1

Failed: 0

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()

{

\_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.
* **1. IMailSender.cs**
* csharp
* CopyEdit
* namespace CustomerCommLib
* {
* public interface IMailSender
* {
* bool SendMail(string toAddress, string message);
* }
* }

**2. MailSender.cs**

csharp

CopyEdit

using System.Net;

using System.Net.Mail;

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);

return true;

}

}

}

**CustomerComm.cs**

csharp

CopyEdit

namespace CustomerCommLib

{

public class CustomerComm

{

private readonly IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

// Normally you would fetch this from a DB or config

string email = "cust123@abc.com";

string message = "Some Message";

return \_mailSender.SendMail(email, message);

}

}

}

**CustomerCommTests.cs**

csharp

CopyEdit

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerCommTests

{

[TestFixture]

public class CustomerCommTests

{

private Mock<IMailSender> \_mailSenderMock;

private CustomerComm \_customerComm;

[SetUp]

public void Setup()

{

\_mailSenderMock = new Mock<IMailSender>();

\_customerComm = new CustomerComm(\_mailSenderMock.Object);

}

[Test]

public void SendMailToCustomer\_ShouldReturnTrue\_WhenMailIsSent()

{

\_mailSenderMock

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

.Returns(true);

// Act

bool result = \_customerComm.SendMailToCustomer();

// Assert

Assert.That(result, Is.True);

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

}

}

}

Output:

