**Advanced SQL  
Exercise 1: Ranking and Window Functions**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

Region VARCHAR(50)

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE OrderDetails (

OrderDetailID INT PRIMARY KEY,

OrderID INT,

ProductID INT,

Quantity INT,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

INSERT INTO Customers (CustomerID, Name, Region) VALUES

(1, 'Alice', 'North'),

(2, 'Bob', 'South'),

(3, 'Charlie', 'East'),

(4, 'David', 'West');

INSERT INTO Products (ProductID, ProductName, Category, Price) VALUES

(1, 'Laptop', 'Electronics', 1200.00),

(2, 'Smartphone', 'Electronics', 800.00),

(3, 'Tablet', 'Electronics', 600.00),

(4, 'Headphones', 'Accessories', 150.00);

INSERT INTO Orders (OrderID, CustomerID, OrderDate) VALUES

(1, 1, '2023-01-15'),

(2, 2, '2023-02-20'),

(3, 3, '2023-03-25'),

(4, 4, '2023-04-30');

INSERT INTO OrderDetails (OrderDetailID, OrderID, ProductID, Quantity) VALUES

(1, 1, 1, 1),

(2, 2, 2, 2),

(3, 3, 3, 1),

(4, 4, 4, 3);

/\* Row\_NUMBER() \*/

SELECT \* FROM (

SELECT

ProductID,

ProductName,

Category,

Price,

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

FROM Products

) AS RankedProducts

WHERE RowNum <= 3;

/\* RANK() \*/

SELECT \* FROM (

SELECT

ProductID,

ProductName,

Category,

Price,

RANK() OVER (PARTITION BY Category ORDER BY Price) AS RowNum

FROM Products

) AS Ranked

WHERE RowNum <= 3;

/\* DENSE\_RANK() \*/

SELECT \* FROM (

SELECT

ProductID,

ProductName,

Category,

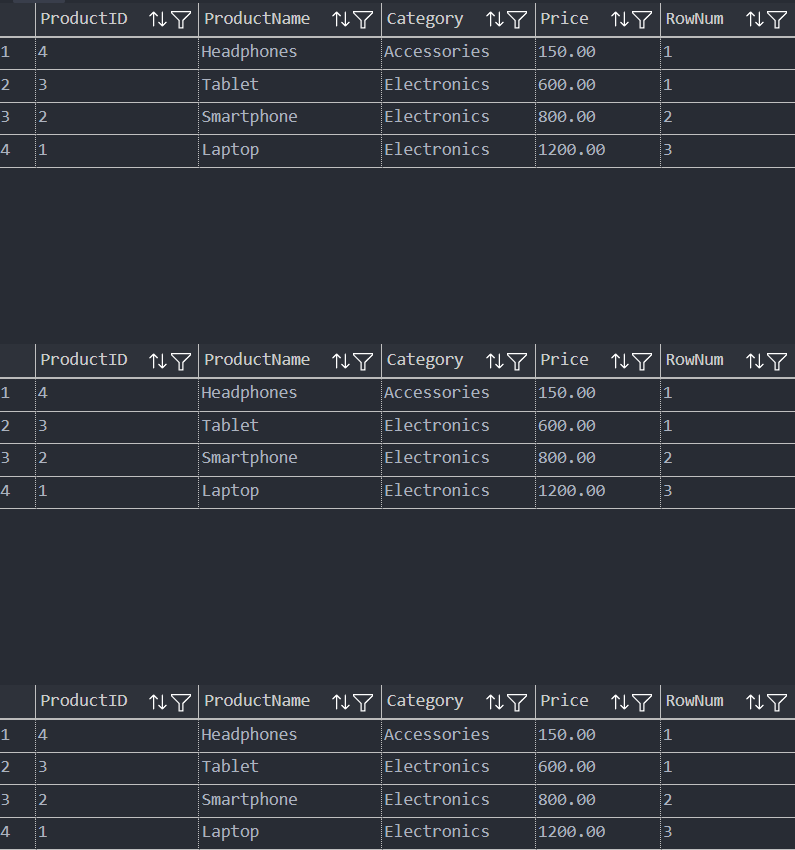
Price,

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

FROM Products

) AS RankedProd

WHERE RowNum <= 3;

  
  
**Exercise 1: Create a Stored Procedure**

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY IDENTITY(1,1),

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT,

Salary DECIMAL(10, 2),

JoinDate DATE,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'Human Resources'),

(2, 'Engineering'),

(3, 'Sales'),

(4, 'Marketing');

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

(101, 'Alice', 'Johnson', 1, 50000.00, '2020-01-10'),

(102, 'Bob', 'Smith', 2, 75000.00, '2019-03-15'),

(103, 'Charlie', 'Brown', 2, 80000.00, '2021-07-20'),

(104, 'David', 'Lee', 3, 60000.00, '2018-05-30'),

(105, 'Eve', 'Williams', 4, 65000.00, '2022-11-01');  
  
  
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;

GO

EXEC sp\_InsertEmployee

@FirstName = 'Rony',

@LastName = 'Gasper',

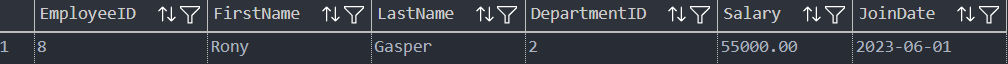
@DepartmentID = 2,

@Salary = 55000.00,

@JoinDate = '2023-06-01';

GO

SELECT \* FROM Employees WHERE FirstName = 'Rony' AND LastName = 'Gasper';



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

-- Create Departments table

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

-- Create Employees table

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 Departments

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

-- Insert Employees

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');

CREATE PROCEDURE sp\_GetEmployeeCountByDept @DeptID INT

AS

BEGIN

SELECT COUNT(\*) AS EmployeeCount

FROM Employees

WHERE DepartmentID = @DeptID

END;

GO

EXEC sp\_GetEmployeeCountByDept @DeptID = 1;



**NUnit and Moq**

**NUnit-Handson**

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; }

        }

    }

}

#TESTCASE

using NUnit.Framework;

using CalcLibrary;

namespace CalcLibraryTests

{

    [TestFixture]

    public class CalculatorTests

    {

        private SimpleCalculator calculator;

        [SetUp]

        public void Setup()

        {

            calculator = new SimpleCalculator();

        }

        [Test]

        [TestCase(2, 3, 5)]

        [TestCase(-1, 1, 0)]

        [TestCase(0, 0, 0)]

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

        {

            var result = calculator.Addition(a, b);

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

        }

        [Test]

        [TestCase(5, 3, 2)]

        [TestCase(0, 5, -5)]

        [TestCase(-1, -1, 0)]

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

        {

            var result = calculator.Subtraction(a, b);

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

        }

        [Test]

        [TestCase(4, 5, 20)]

        [TestCase(-3, 2, -6)]

        [TestCase(0, 10, 0)]

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

        {

            var result = calculator.Multiplication(a, b);

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

        }

        [Test]

        [TestCase(10, 2, 5)]

        [TestCase(-8, 2, -4)]

        [TestCase(5, -1, -5)]

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

        {

            var result = calculator.Division(a, b);

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

        }

        [Test]

        public void Division\_ByZero\_ThrowsArgumentException()

        {

            var ex = Assert.Throws<ArgumentException>(() => calculator.Division(10, 0));

            Assert.That(ex.Message, Is.EqualTo("Second Parameter Can't be Zero"));

        }

        [Test]

        public void AllClear\_ResetsResultToZero()

        {

            calculator.Addition(10, 5);

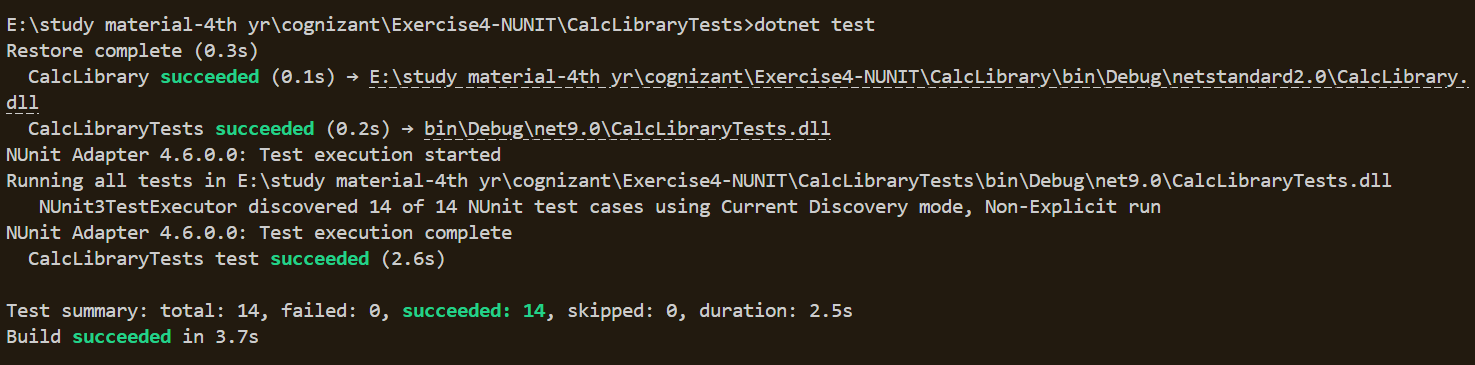
            calculator.AllClear();

            Assert.That(calculator.GetResult, Is.EqualTo(0));

        }

    }

}



**Moq-Handson**

#CustomerCommTests

using NUnit.Framework;

using Moq;

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

                .Returns(true);

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

        }

        [TestCase]

        public void SendMailToCustomer\_ReturnsTrue()

        {

            var result = \_customerComm!.SendMailToCustomer();

            Assert.That(result, Is.True);

        }

    }

}

#customercomm  
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;

        }

    }

}

#mailsender  
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 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;

        }

    }

}

