# Week 2

Exercise 1: Ranking and window functions.

**SQL query:**

DROP TABLE IF EXISTS test2.Products;

CREATE TABLE Products (

ProductID INT PRIMARY KEY AUTO\_INCREMENT,

ProductName VARCHAR(100) NOT NULL,

Category VARCHAR(50) NOT NULL,

Price DECIMAL(10,2) NOT NULL

);

INSERT INTO Products (ProductName, Category, Price) VALUES

('Laptop Pro', 'Electronics', 2000),

('Phone X', 'Electronics', 1200),

('Tablet Z', 'Electronics', 800),

('Earbuds A', 'Electronics', 200),

('Vacuum Max', 'Home', 600),

('Blender Ultra', 'Home', 250),

('Coffee Maker', 'Home', 250),

('Toaster Slim', 'Home', 50),

('Leather Jacket', 'Clothing', 350),

('Running Shoes', 'Clothing', 120),

('Jeans Classic', 'Clothing', 80),

('T-Shirt Basic', 'Clothing', 20);

SELECT \*

FROM (

SELECT ProductID, ProductName, Category, Price,

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

FROM Products

) AS t

WHERE row\_num <= 3;

SELECT \*

FROM (

SELECT ProductID, ProductName, Category, Price,

RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS rnk

FROM Products

) AS t

WHERE rnk <= 3;

SELECT \*

FROM (

SELECT ProductID, ProductName, Category, Price,

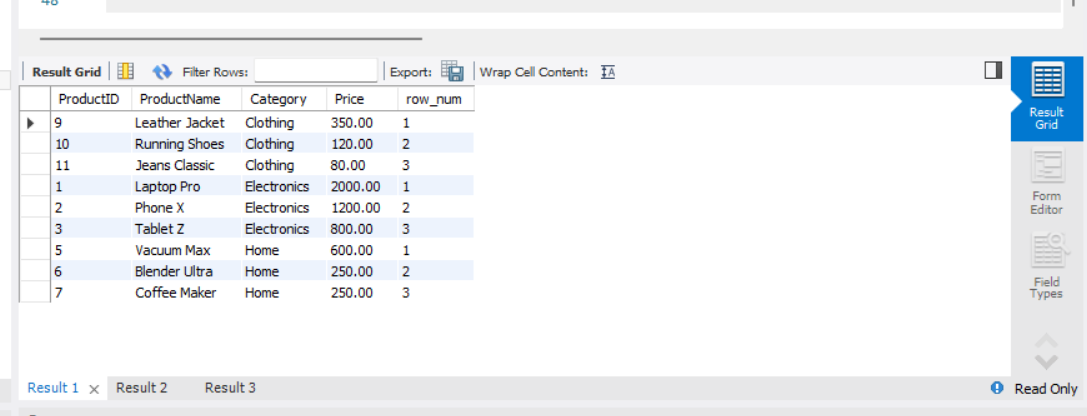
DENSE\_RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS drnk

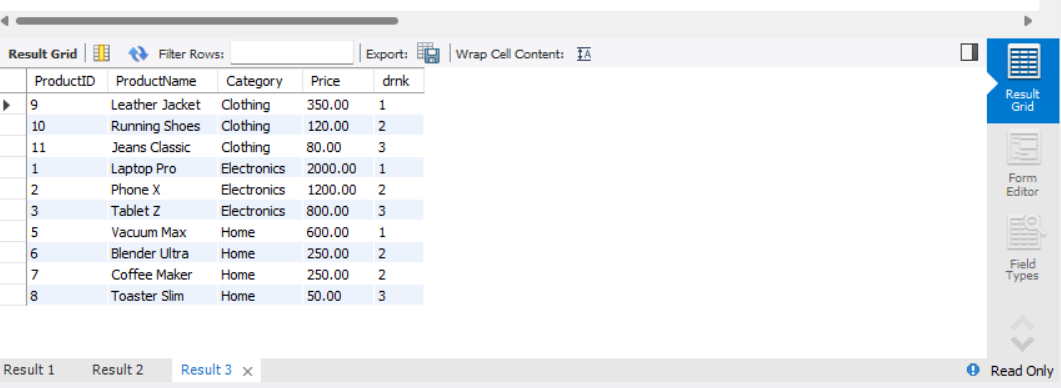
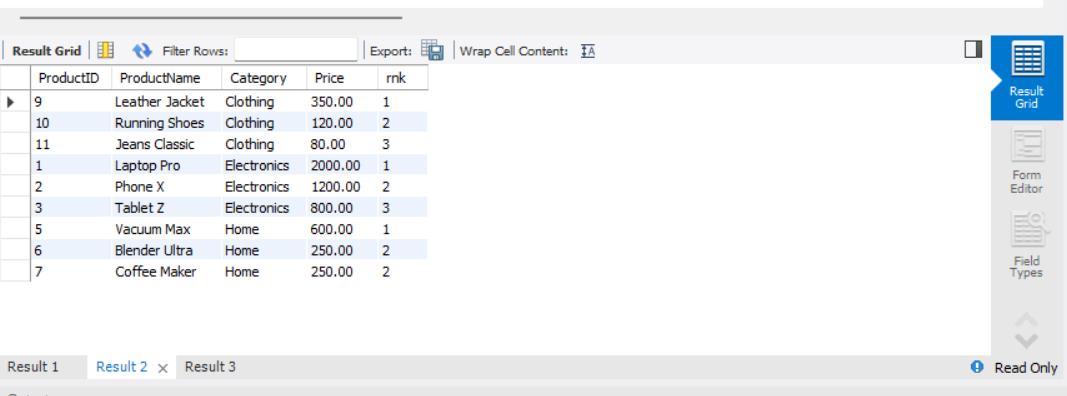
FROM Products

) AS t

WHERE drnk <= 3;

**Output:** Finding the top 3 most expensive products in each category using different ranking functions

****

****

**Exercise 2: Create a Stored Procedure**

**Sql query :**

DROP TABLE Departments;

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 (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID, Salary,

JoinDate) VALUES

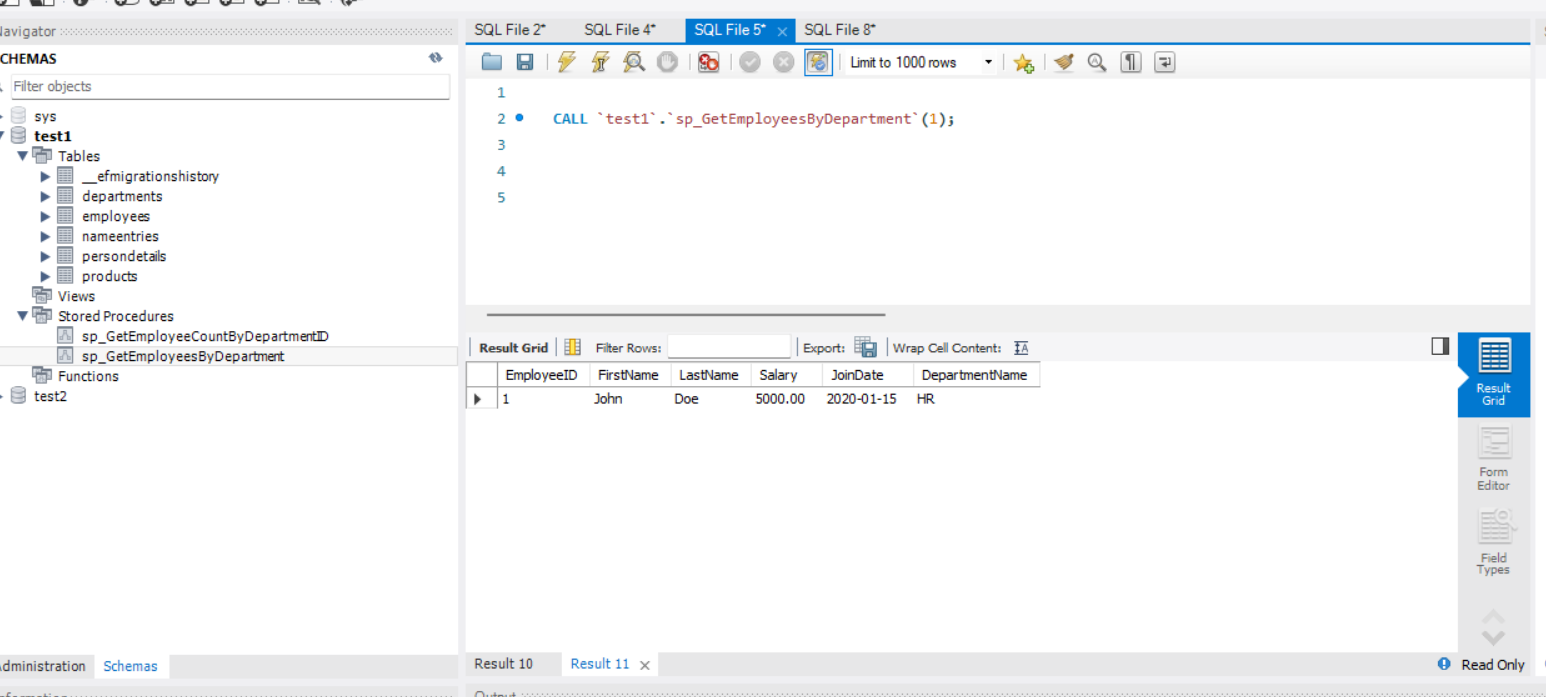
(1, 'John', 'Doe', 1, 5000.00, '2020-01-15'),

(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');

**Output:** Creating a stored procedure to retrieve employee details by department.



**Exercise 3: Return Data from a Stored Procedure**

**SQL query** : By using the same query used for creating a stored procedure

DROP TABLE Departments;

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 (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID, Salary,

JoinDate) VALUES

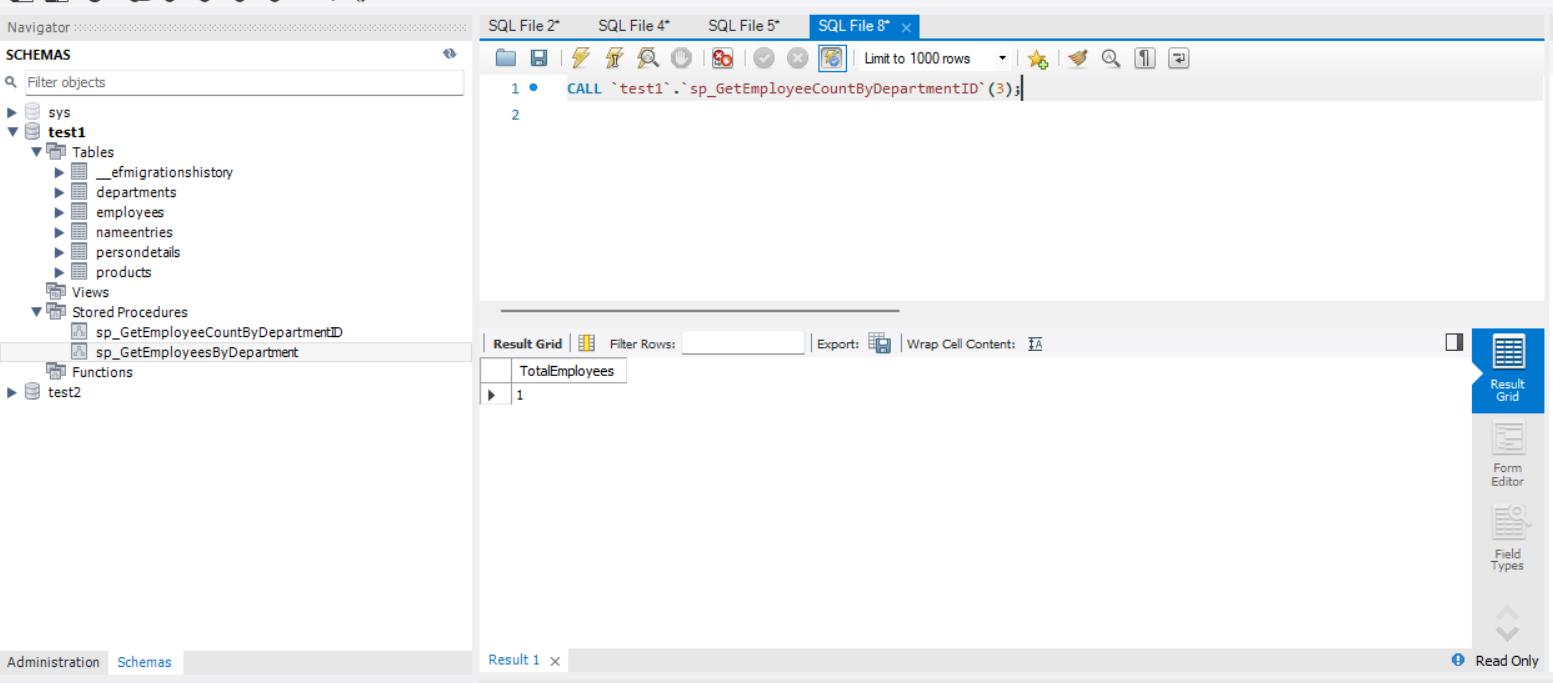
(1, 'John', 'Doe', 1, 5000.00, '2020-01-15'),

(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');

**Output:** Return Data from a Stored Procedure.

****

**Exercise 4 : NUnit.Handson .**

**Unit Testing vs Functional Testing :**

| Feature | Unit Testing | Functional Testing |
| --- | --- | --- |
| Scope | Individual units (methods/functions) | Whole features or workflows |
| Speed | Fast – no external dependencies | Slower – runs with real components |
| Tools | NUnit, xUnit, MSTest, Moq | Selenium, Postman, JMeter |
| Mocking | Common, for isolation | Rare, more end-to-end |
| Goal | Validate logic accuracy | Validate business outcomes |

**Types of Testing :**

* Unit Testing – Smallest pieces, tested in isolation
* Functional Testing – Ensures features behave as expected
* Automated Testing – Quick, repeatable tests run without manual effort
* Performance Testing – Measures responsiveness and scalability

**Benefits of Automated Testing :**

* Detect bugs early
* Improve developer confidence
* Enable CI/CD pipelines

**Test Project: NUnit + Moq**

**CODE :**

using NUnit.Framework;

using CalcLibrary;

namespace CalcLibrary.Tests

{

[TestFixture]

public class CalculatorTests

{

private Calculator \_calc;

[SetUp]

public void Init() => \_calc = new Calculator();

[TearDown]

public void Cleanup() => \_calc = null;

[TestCase(2, 3, 5)]

[TestCase(-1, 4, 3)]

[TestCase(0, 0, 0)]

public void Add\_ValidInput\_ReturnsSum(int a, int b, int expected)

{

int result = \_calc.Add(a, b);

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

}

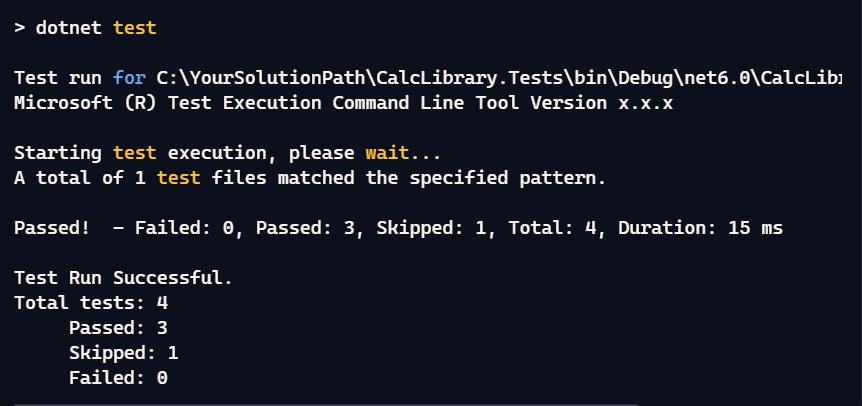
[Test, Ignore("Skipping demo test")]

public void IgnoredTest() => Assert.Fail("This test should not run.");

}

}

**OUTPUT :**

****

Exercise\_5 : Write Testable Code with Moq.

**Objective**:  
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.

**CODE :**

using Moq;

using NUnit.Framework;

using CustomerCommLib;

namespace CustomerCommLib.Tests

{

public class CustomerCommTests

{

[Test]

public void SendMailToCustomer\_ShouldReturnTrue\_WhenMailIsSentSuccessfully()

{

var mockMailSender = new Mock<IMailSender>();

mockMailSender

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

.Returns(true);

var comm = new CustomerComm(mockMailSender.Object);

var result = comm.SendMailToCustomer();

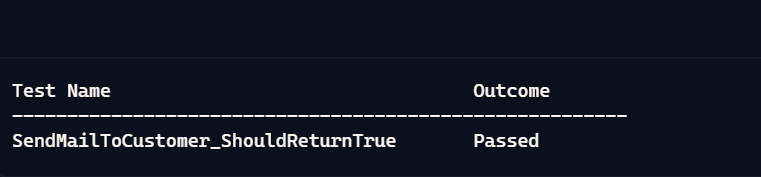
Assert.IsTrue(result);

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

}

}

}

**OUTPUT :  
**