WEEK - 2 (Module - 2)

Aditya Raj

SuperSet ID - (6363544)

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

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2)

);

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

(1, 'Laptop A', 'Electronics', 1000),

(2, 'Laptop B', 'Electronics', 1000),

(3, 'Phone A', 'Electronics', 800),

(4, 'Phone B', 'Electronics', 800),

(5, 'Speaker', 'Electronics', 500),

(6, 'Chair A', 'Furniture', 300),

(7, 'Chair B', 'Furniture', 300),

(8, 'Table', 'Furniture', 200),

(9, 'Couch', 'Furniture', 200);

SELECT

ProductID,

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 DenseRankNum

FROM Products;

WITH Ranked AS (

SELECT

\*,

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

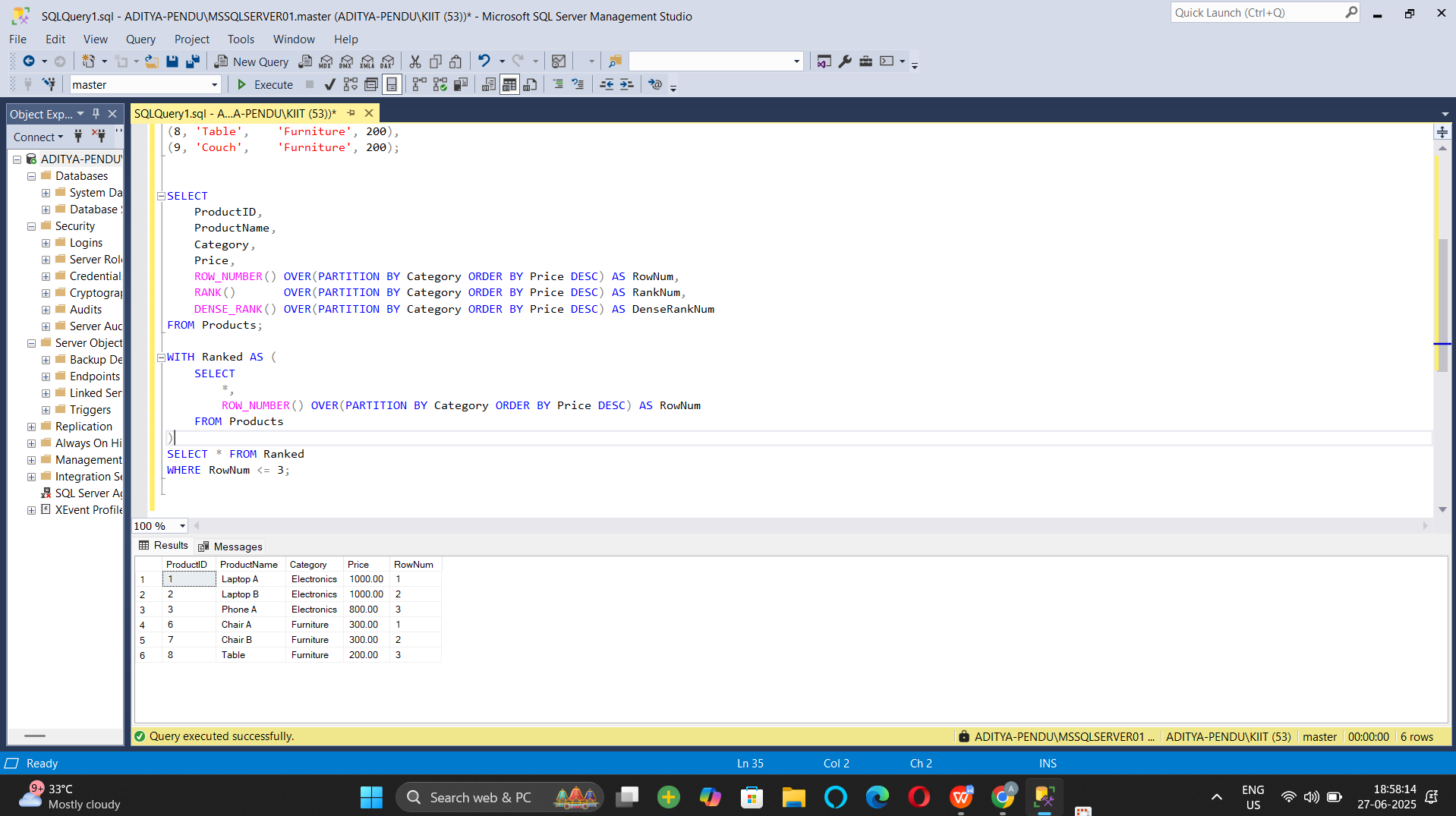
FROM Products

)

SELECT \* FROM Ranked

WHERE RowNum <= 3;

**Output:**

****

**Exercise 2: Aggregation with GROUPING SETS, CUBE, and ROLLUP:**

1. Join Orders, OrderDetails, Customers, and Products.

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

EmployeeName VARCHAR(100),

Department VARCHAR(50),

Region VARCHAR(50)

);

CREATE TABLE Sales (

SaleID INT PRIMARY KEY,

EmployeeID INT,

SaleAmount DECIMAL(10, 2),

SaleDate DATE,

FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)

);

INSERT INTO Employees VALUES

(1, 'Aditya', 'HR', 'North'),

(2, 'Uditya', 'HR', 'South'),

(3, 'Raj', 'Sales', 'North'),

(4, 'Rahul', 'Sales', 'West'),

(5, 'Rohit', 'IT', 'South');

INSERT INTO Sales VALUES

(101, 1, 1000.00, '2024-01-10'),

(102, 2, 1500.00, '2024-01-11'),

(103, 3, 2000.00, '2024-01-12'),

(104, 4, 2500.00, '2024-01-13'),

(105, 5, 1800.00, '2024-01-14'),

(106, 1, 1200.00, '2024-02-10'),

(107, 3, 800.00, '2024-02-12');

SELECT

E.Region,

E.Department,

SUM(S.SaleAmount) AS TotalSales

FROM Sales S

JOIN Employees E ON S.EmployeeID = E.EmployeeID

GROUP BY GROUPING SETS (

(E.Region),

(E.Department),

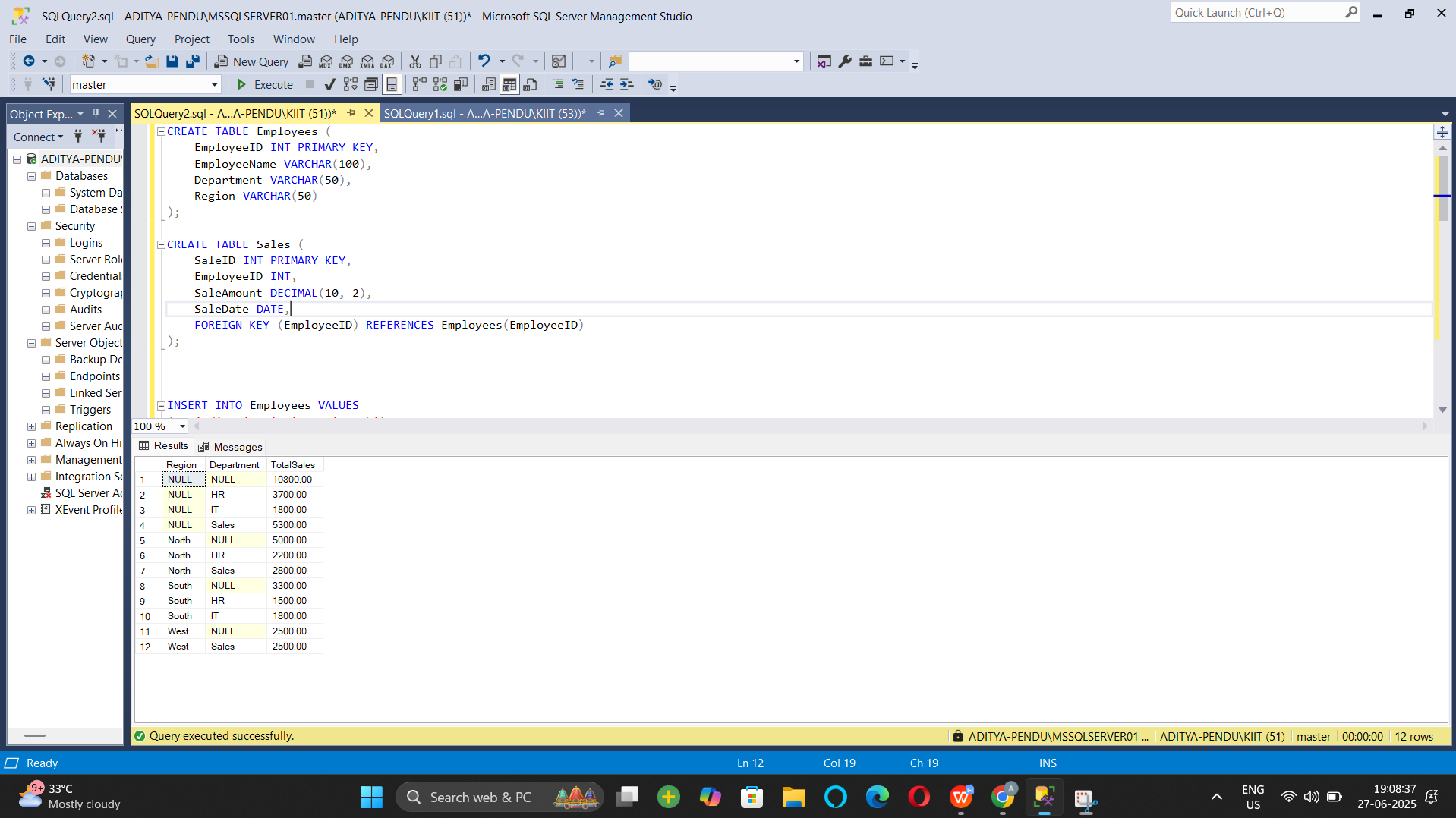
(E.Region, E.Department),

()

)

ORDER BY E.Region, E.Department;

Output :



1. **Use GROUPING SETS to get totals by Region, Category, and both.**

SELECT

E.Region,

E.Department,

SUM(S.SaleAmount) AS TotalSales

FROM Sales S

JOIN Employees E ON S.EmployeeID = E.EmployeeID

GROUP BY GROUPING SETS (

(E.Region),

(E.Department),

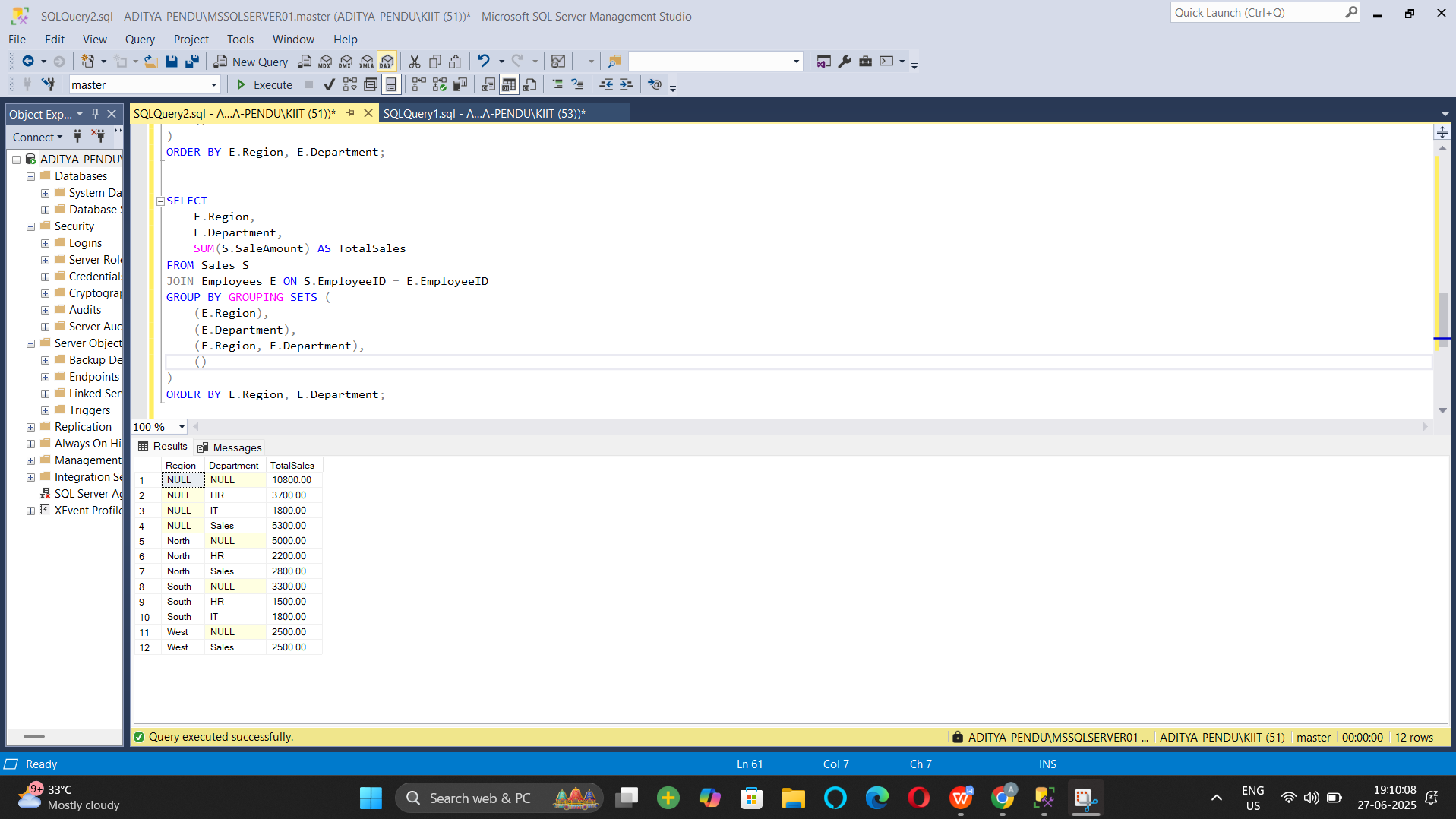
(E.Region, E.Department),

()

)

ORDER BY E.Region, E.Department;

**Output**



1. **Use ROLLUP to get subtotals and grand totals.**

SELECT

E.Region,

E.Department,

SUM(S.SaleAmount) AS TotalSales

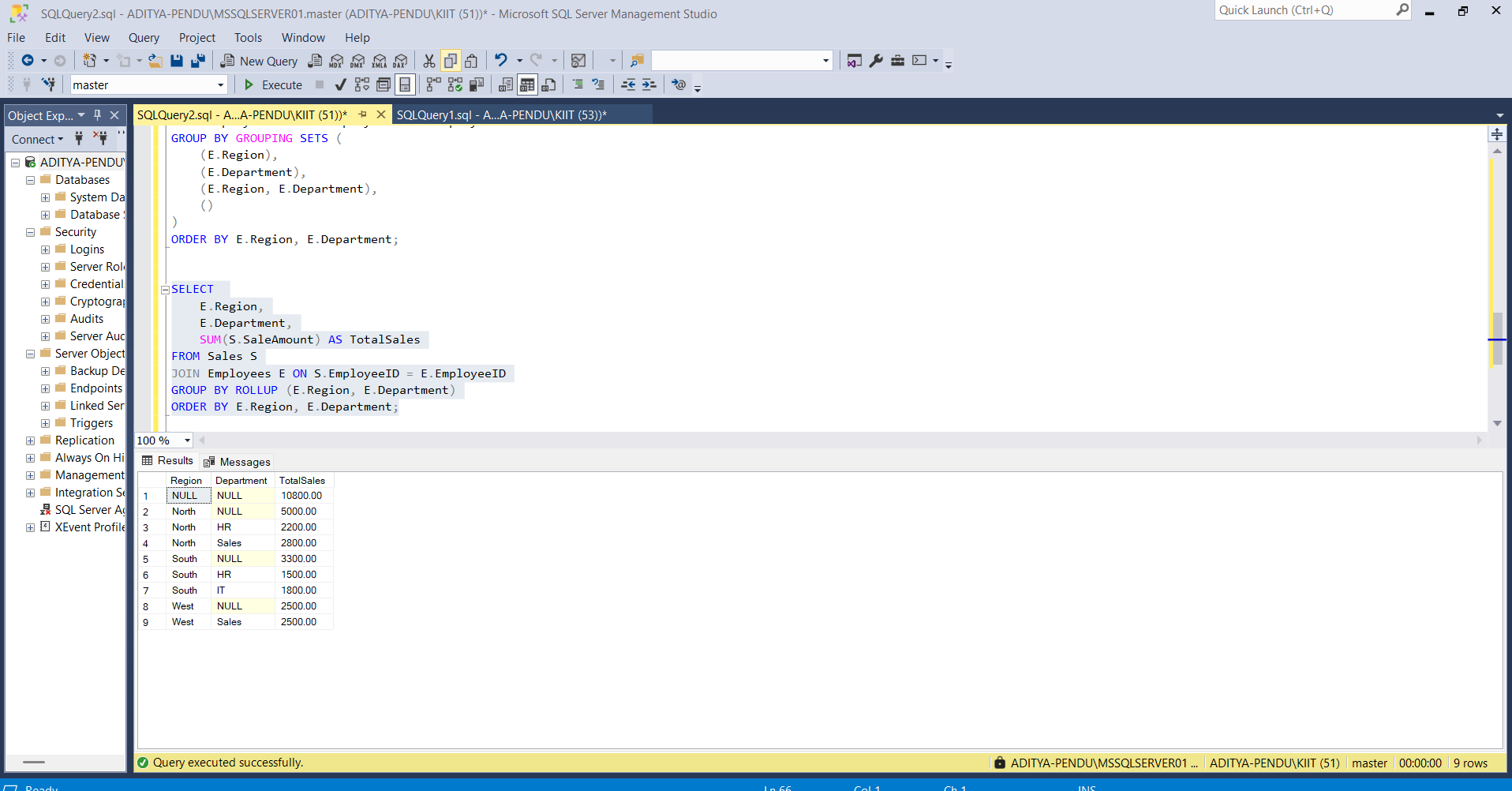
FROM Sales S

JOIN Employees E ON S.EmployeeID = E.EmployeeID

GROUP BY ROLLUP (E.Region, E.Department)

ORDER BY E.Region, E.Department;

**Output:**



1. Use CUBE to get all combinations of Region and Category.

SELECT

E.Region,

E.Department,

GROUPING(E.Region) AS RegionGrouped,

GROUPING(E.Department) AS DepartmentGrouped,

SUM(S.SaleAmount) AS TotalSales

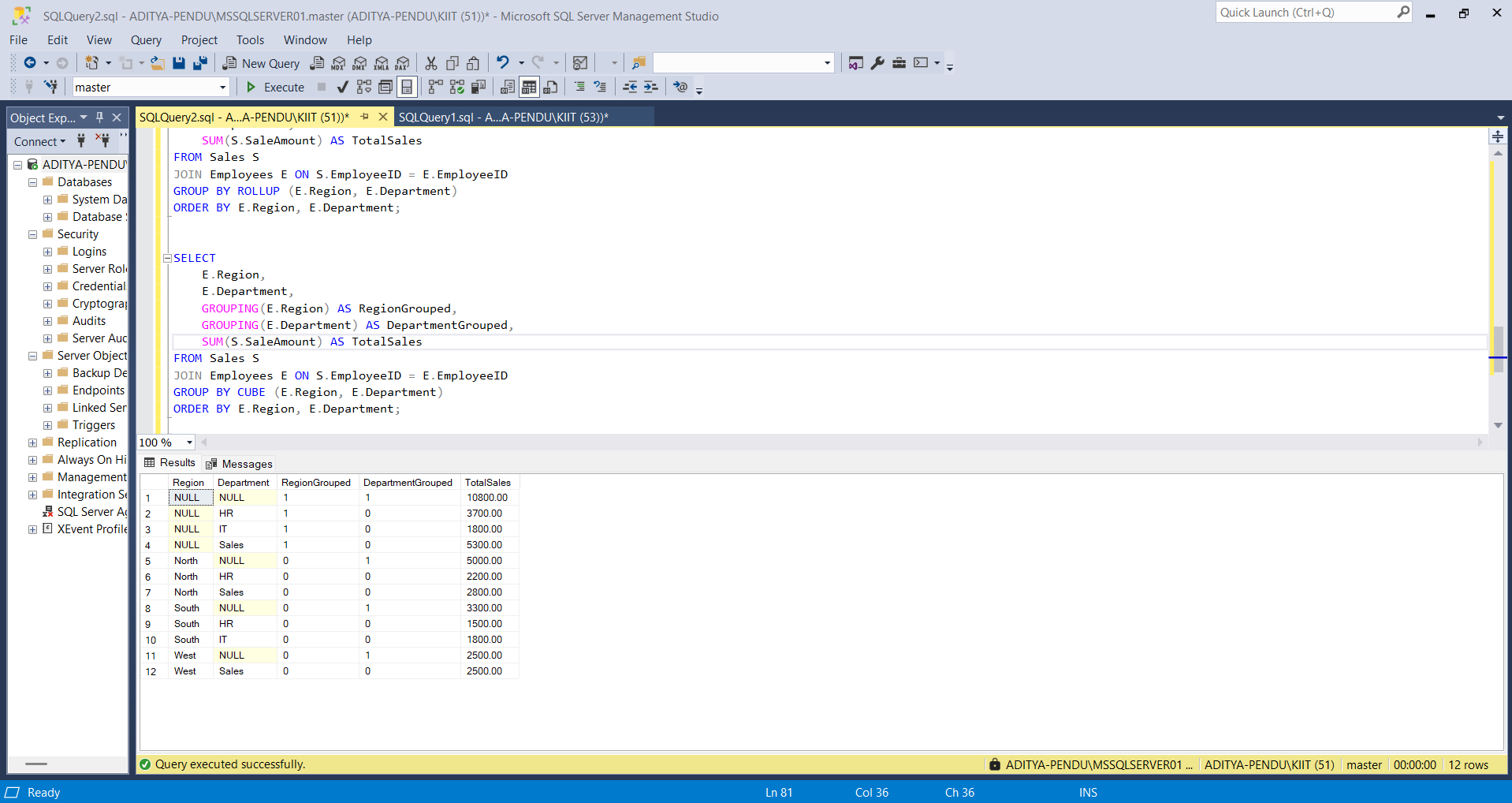
FROM Sales S

JOIN Employees E ON S.EmployeeID = E.EmployeeID

GROUP BY CUBE (E.Region, E.Department)

ORDER BY E.Region, E.Department;

**Output :**



**Exercise 3: CTEs and MERGE**

1. **Create a recursive CTE to generate dates from '2025-01-01' to '2025-01-31'**

WITH CalendarCTE AS (

SELECT CAST('2025-01-01' AS DATE) AS CalendarDate

UNION ALL

SELECT DATEADD(DAY, 1, CalendarDate)

FROM CalendarCTE

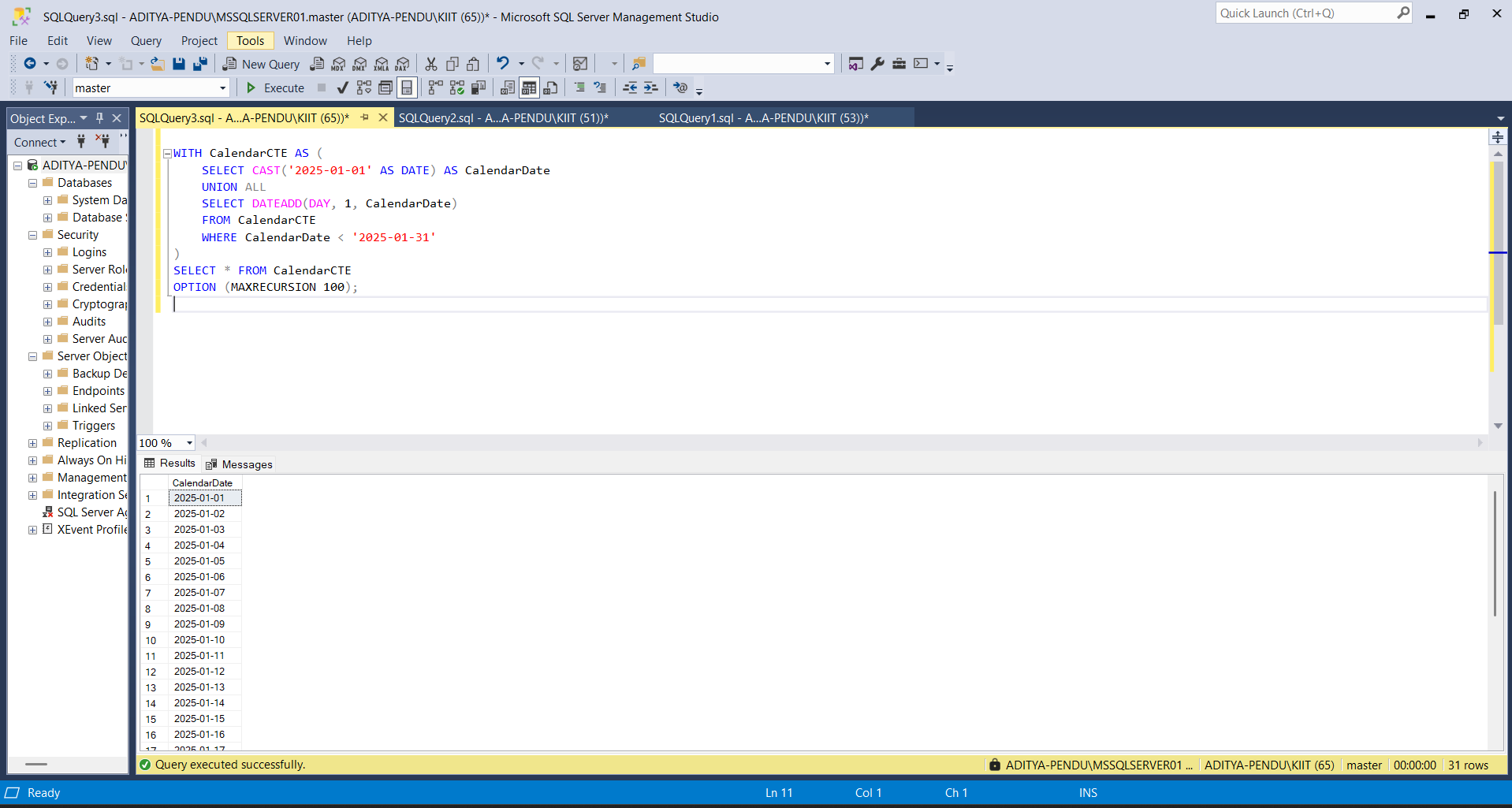
WHERE CalendarDate < '2025-01-31'

)

SELECT \* FROM CalendarCTE

OPTION (MAXRECURSION 100);

Output :



1. **Create a StagingProducts table with updated prices.**

CREATE TABLE Inventory (

ItemID INT PRIMARY KEY,

ItemName VARCHAR(100),

Price DECIMAL(10, 2)

);

INSERT INTO Inventory VALUES

(1, 'Mouse', 250.00),

(2, 'Keyboard', 450.00),

(3, 'Monitor', 5000.00);

CREATE TABLE StagingInventory (

ItemID INT,

ItemName VARCHAR(100),

Price DECIMAL(10, 2)

);

INSERT INTO StagingInventory VALUES

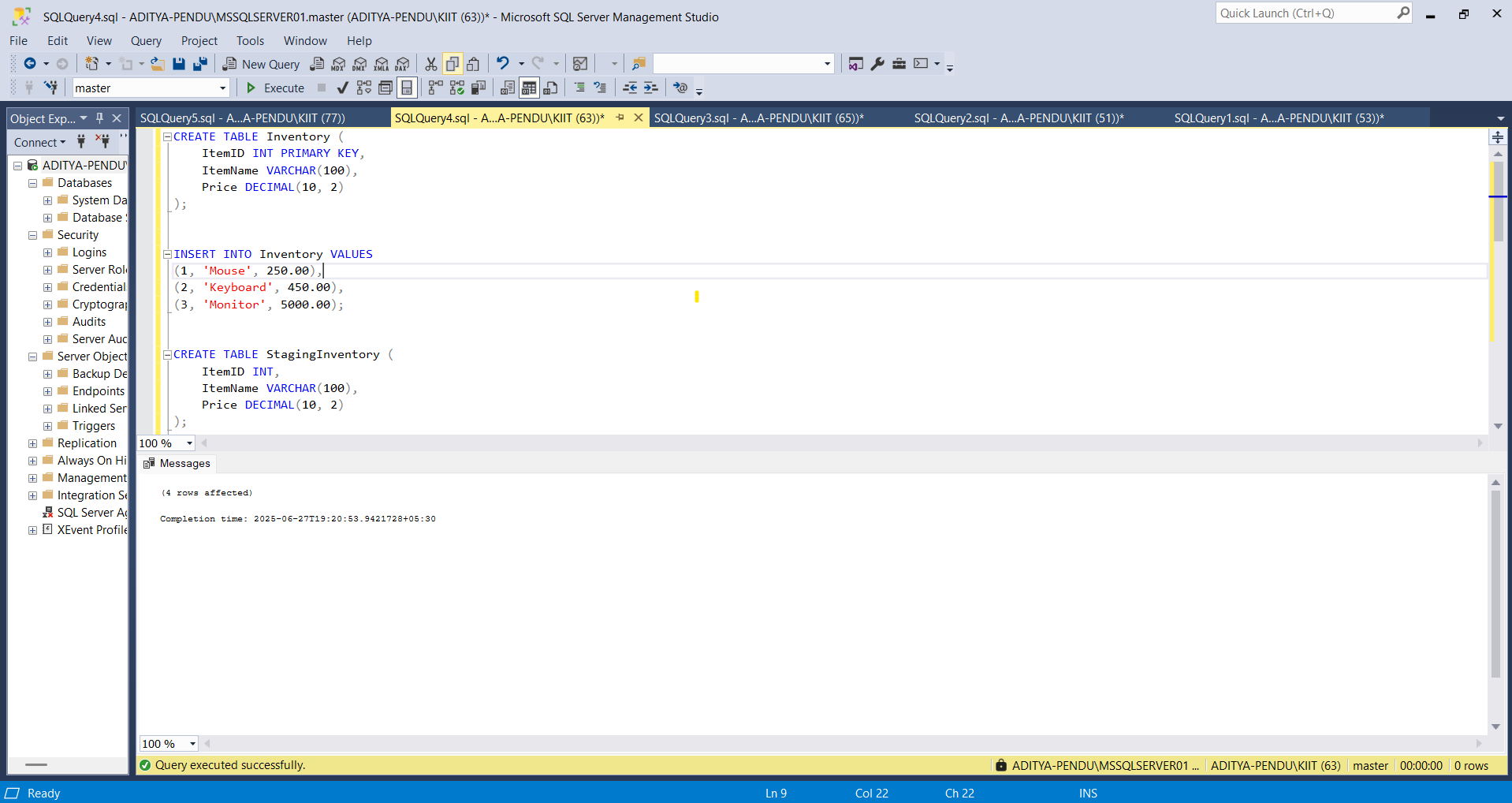
(1, 'Mouse', 275.00),

(3, 'Monitor', 4800.00),

(4, 'Webcam', 1200.00),

(5, 'Speaker', 1800.00);

**Output :**



1. **Use MERGE to update existing products or insert new ones.**

MERGE Inventory AS Target

USING StagingInventory AS Source

ON Target.ItemID = Source.ItemID

WHEN MATCHED THEN

UPDATE SET

Target.ItemName = Source.ItemName,

Target.Price = Source.Price

WHEN NOT MATCHED BY TARGET THEN

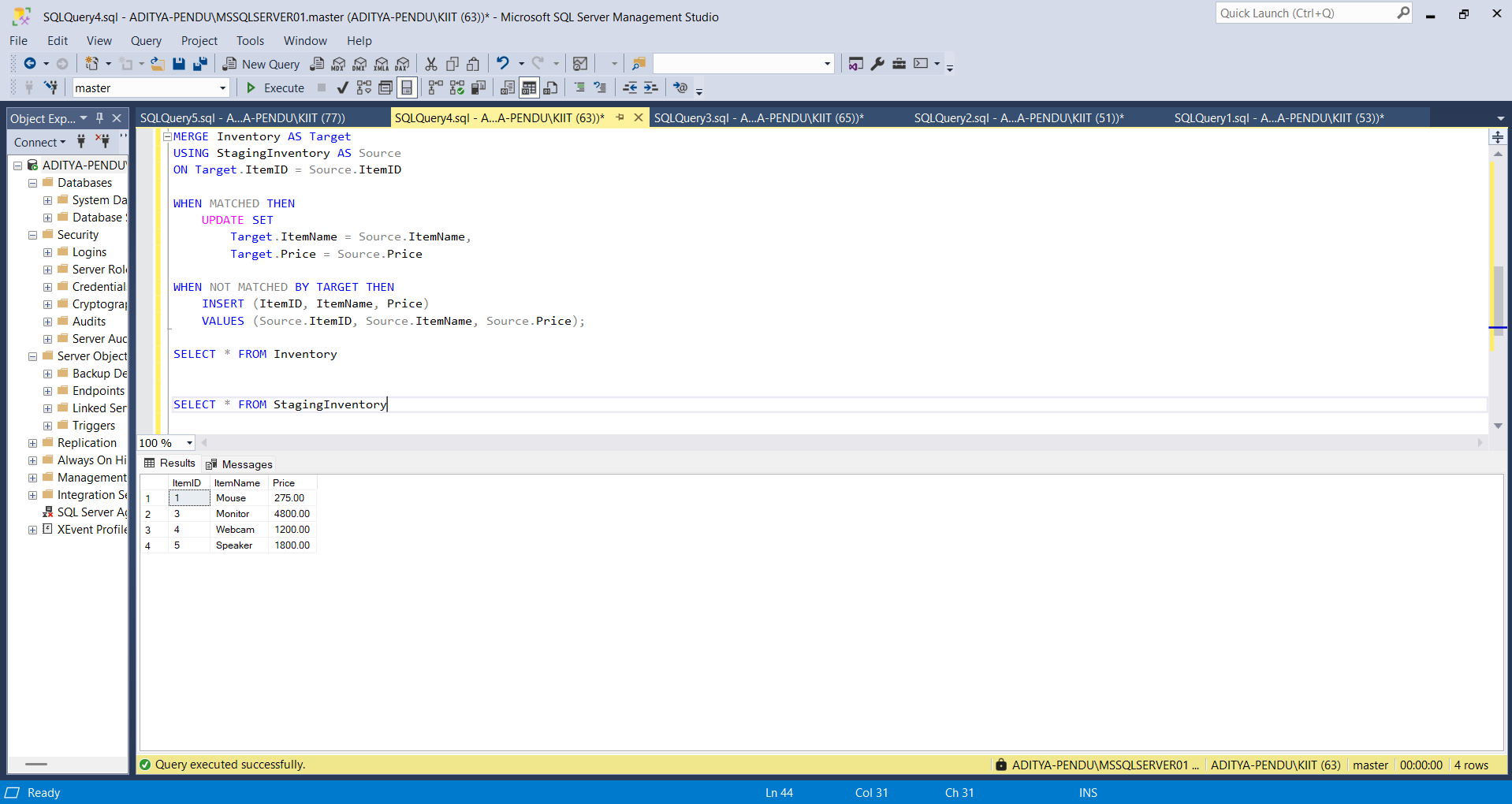
INSERT (ItemID, ItemName, Price)

VALUES (Source.ItemID, Source.ItemName, Source.Price);

SELECT \* FROM Inventory

SELECT \* FROM StagingInventory

Output :



**Exercise 4: PIVOT and UNPIVOT Goal: Transform data for reporting.**

1. Aggregate sales by Product and Month.

CREATE TABLE SalesData (

ProductName VARCHAR(100),

SaleDate DATE,

Quantity INT

);

INSERT INTO SalesData VALUES

('Laptop', '2025-01-05', 10),

('Laptop', '2025-02-12', 15),

('Laptop', '2025-03-20', 8),

('Phone', '2025-01-15', 20),

('Phone', '2025-02-17', 25),

('Phone', '2025-03-10', 12);

SELECT

ProductName,

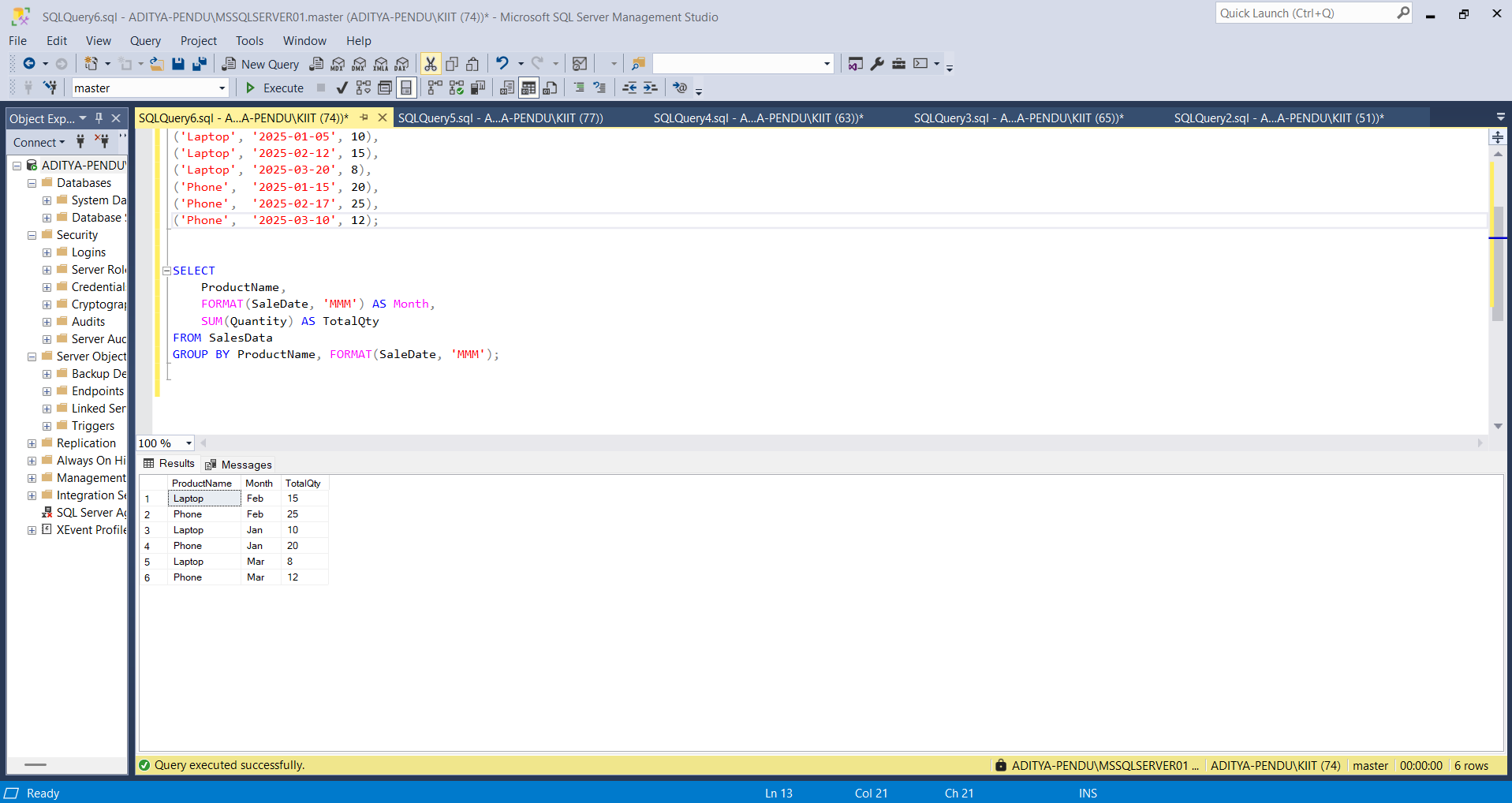
FORMAT(SaleDate, 'MMM') AS Month,

SUM(Quantity) AS TotalQty

FROM SalesData

GROUP BY ProductName, FORMAT(SaleDate, 'MMM');

Output :



1. **Use PIVOT to convert rows into columns (one column per month).**

SELECT \*

FROM (

SELECT

ProductName,

FORMAT(SaleDate, 'MMM') AS SaleMonth,

Quantity

FROM SalesData

) AS SourceTable

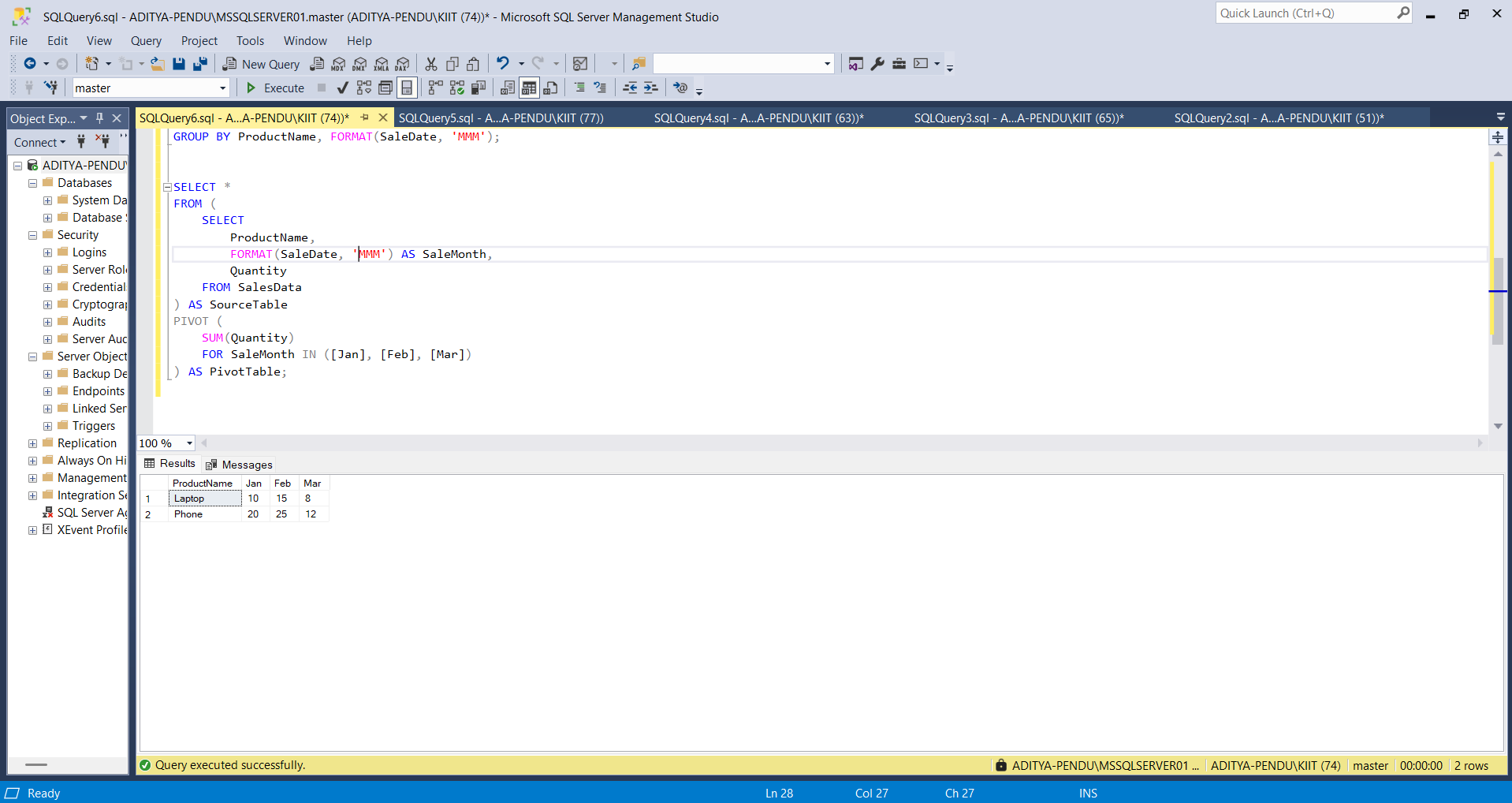
PIVOT (

SUM(Quantity)

FOR SaleMonth IN ([Jan], [Feb], [Mar])

) AS PivotTable;

Output :



1. **Use UNPIVOT to convert the pivoted data back into row format.**

SELECT

ProductName,

SaleMonth,

Quantity

FROM (

SELECT

ProductName, [Jan], [Feb], [Mar]

FROM (

SELECT

ProductName,

FORMAT(SaleDate, 'MMM') AS SaleMonth,

Quantity

FROM SalesData

) AS SourceTable

PIVOT (

SUM(Quantity)

FOR SaleMonth IN ([Jan], [Feb], [Mar])

) AS Pivoted

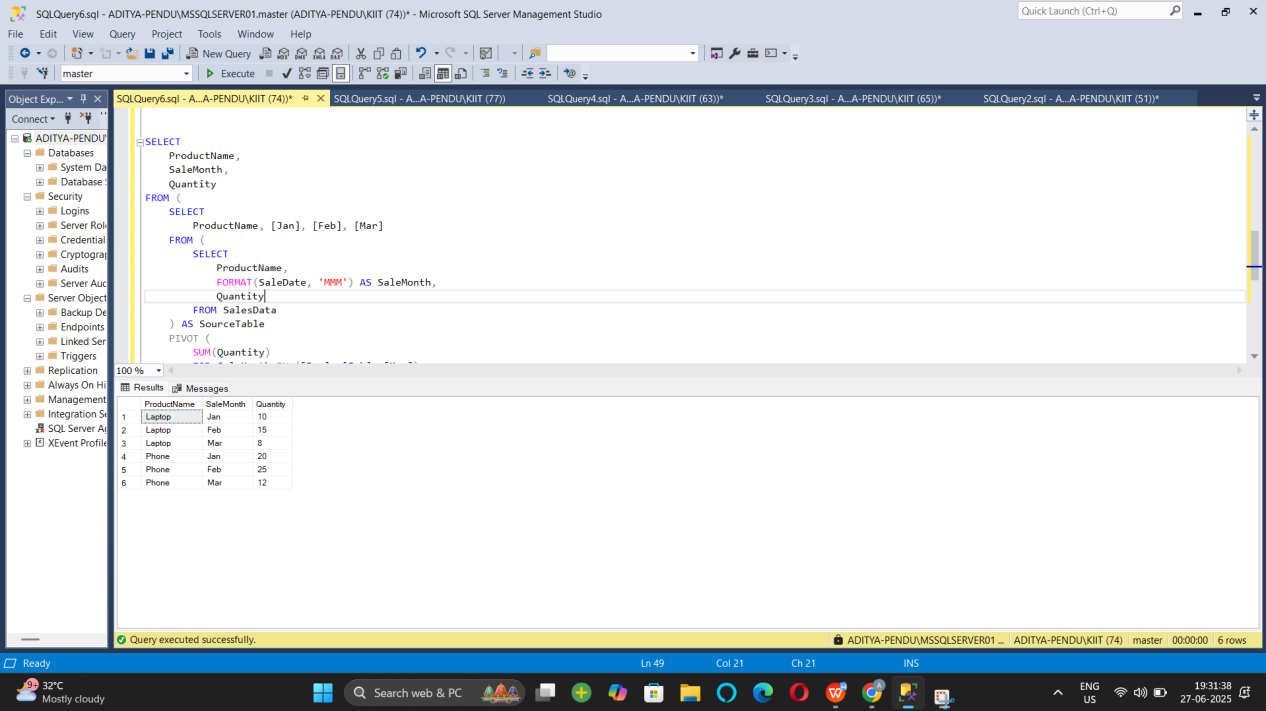
) AS PivotedTable

UNPIVOT (

Quantity FOR SaleMonth IN ([Jan], [Feb], [Mar])

) AS UnpivotedTable;

Output :



**Exercise 5: Using CTE to Simplify a Query**

CREATE TABLE DemoCustomers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100)

);

CREATE TABLE DemoOrders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES DemoCustomers(CustomerID)

);

INSERT INTO DemoCustomers VALUES

(1, 'Aditya'),

(2, 'Uditya'),

(3, 'Raj'),

(4, 'Rahul'),

(5, 'Rohit');

INSERT INTO DemoOrders VALUES

(101, 1, '2024-01-01'),

(102, 1, '2024-01-15'),

(103, 1, '2024-02-01'),

(104, 1, '2024-03-01'),

(105, 2, '2024-01-20'),

(106, 2, '2024-02-14'),

(107, 3, '2024-03-10'),

(108, 3, '2024-04-11'),

(109, 3, '2024-05-12'),

(110, 3, '2024-06-01'),

(111, 4, '2024-03-03'),

(112, 5, '2024-03-05'),

(113, 5, '2024-04-05'),

(114, 5, '2024-05-05');

WITH CustomerOrderCounts AS (

SELECT

CustomerID,

COUNT(OrderID) AS OrderCount

FROM DemoOrders

GROUP BY CustomerID

)

SELECT

c.CustomerID,

c.Name,

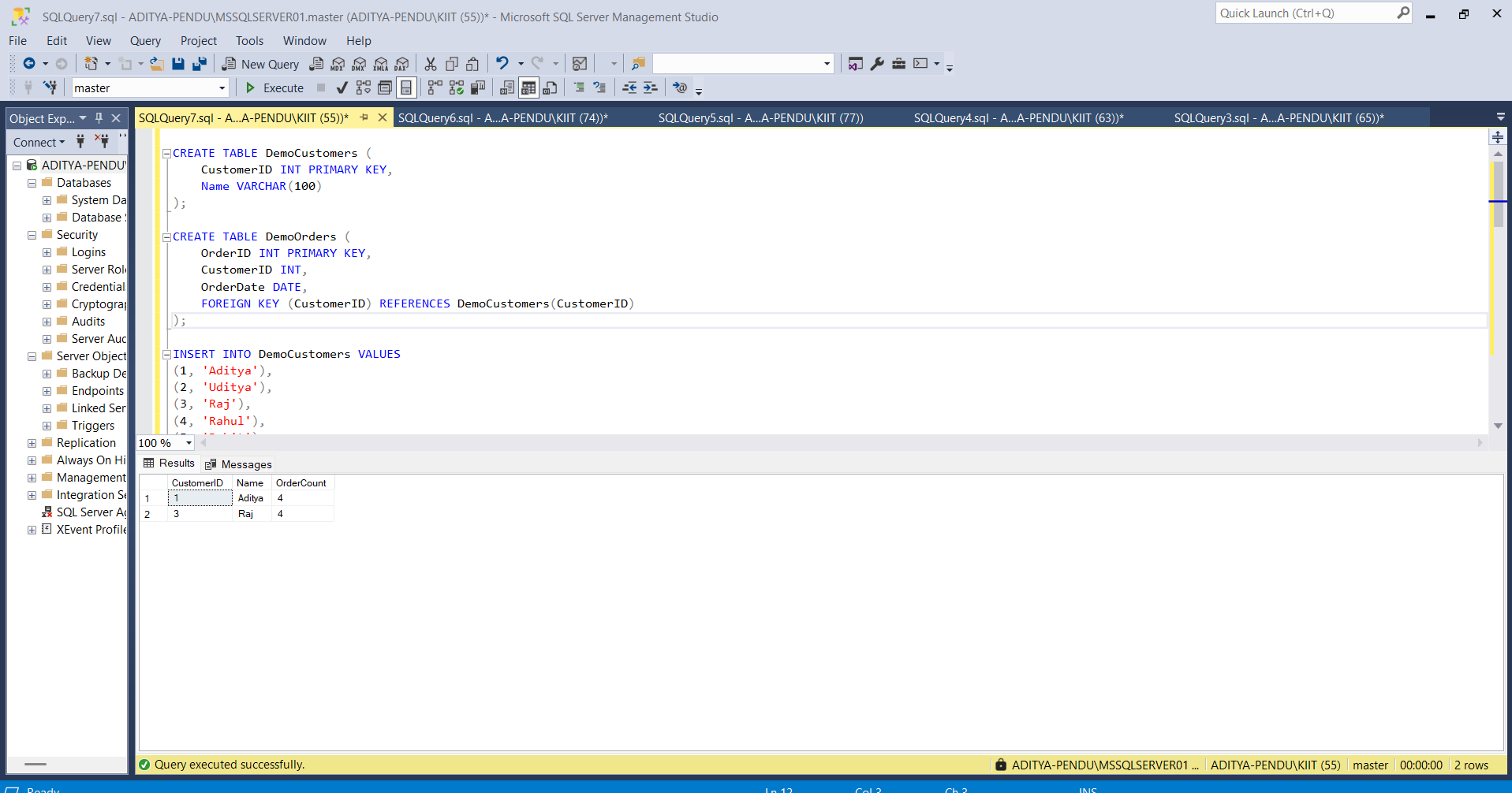
coc.OrderCount

FROM CustomerOrderCounts coc

JOIN DemoCustomers c ON c.CustomerID = coc.CustomerID

WHERE coc.OrderCount > 3;

Output :



**Exercise 2: SQL EXERCISE - Index.SQL :**

**- Question 1: Creating a Non-Clustered Index**

DROP TABLE IF EXISTS OrderDetails;

DROP TABLE IF EXISTS Orders;

DROP TABLE IF EXISTS Products;

DROP TABLE IF EXISTS Customers;

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, 'Aditya', 'North'),

(2, 'Uditya', 'South'),

(3, 'Raj', 'East'),

(4, 'Rahul', 'West'),

(5, 'Rohit', 'Central');

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'),

(5, 5, '2023-05-05');

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

(1, 1, 1, 1),

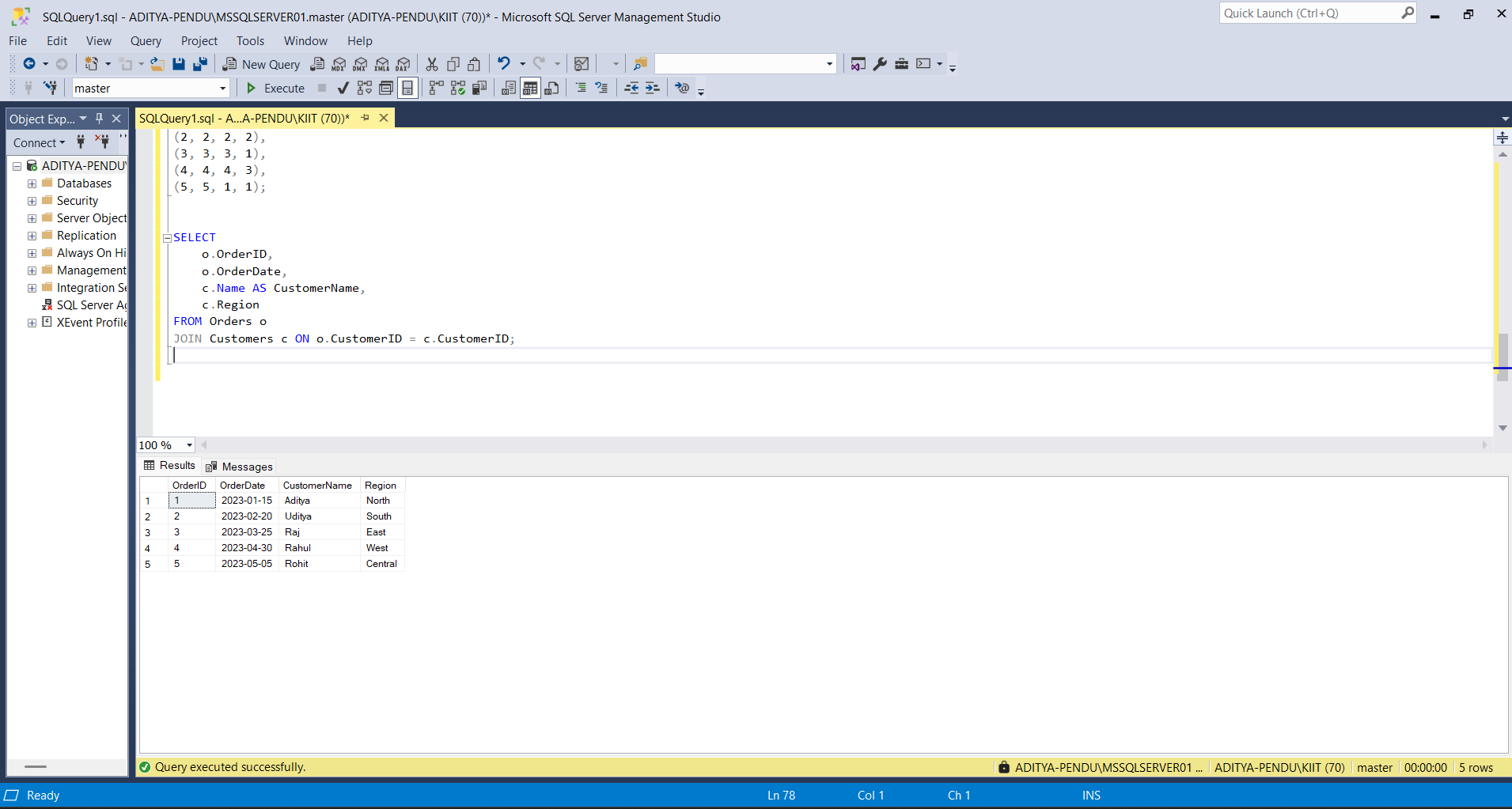
(2, 2, 2, 2),

(3, 3, 3, 1),

(4, 4, 4, 3),

(5, 5, 1, 1);

**Output:**



**Question 2: Creating a Clustered Index:**

SELECT

c.CustomerID,

c.Name,

SUM(od.Quantity) AS TotalQuantity

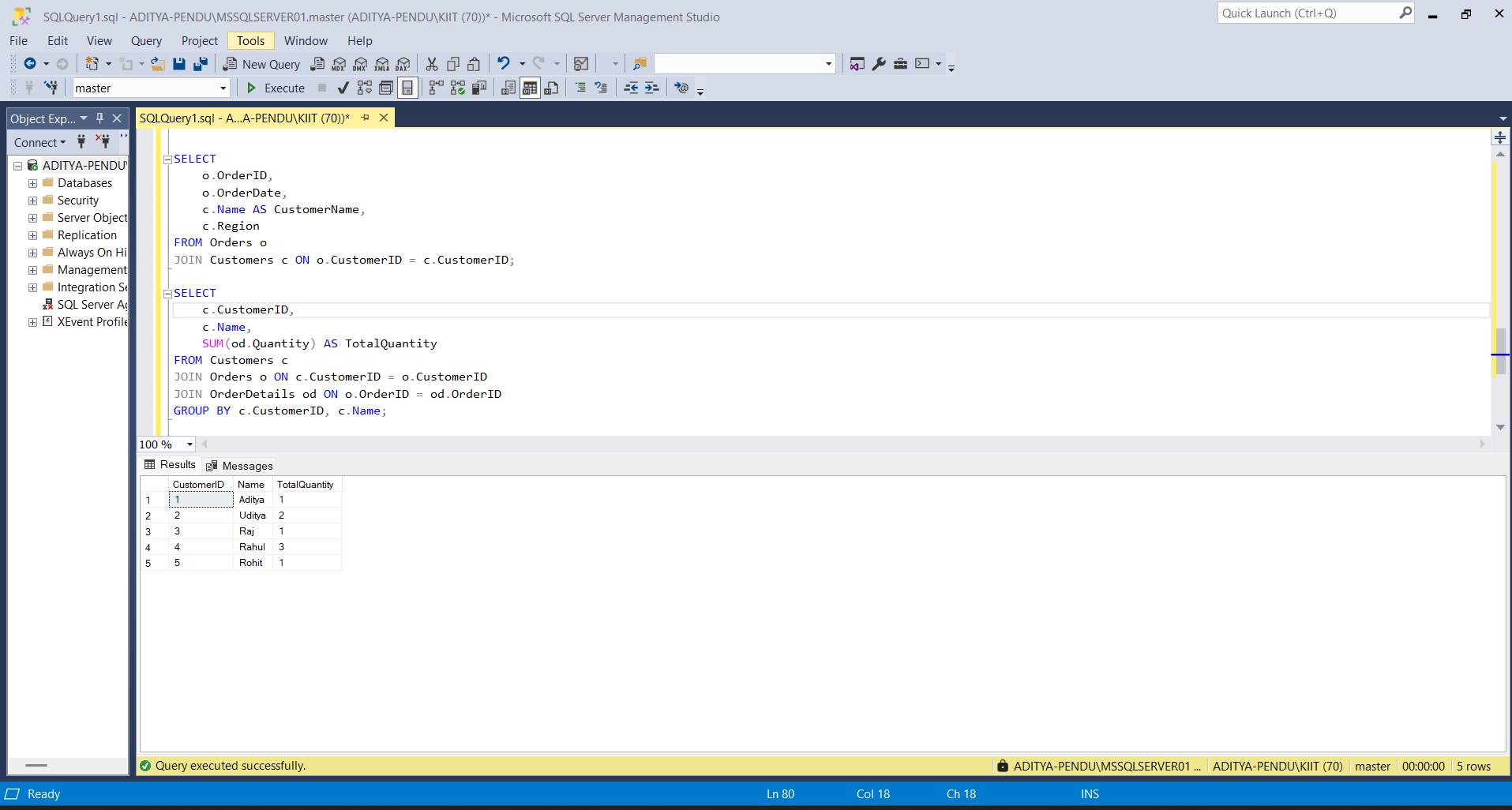
FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

JOIN OrderDetails od ON o.OrderID = od.OrderID

GROUP BY c.CustomerID, c.Name;

**Output :**



**Exercise 3: Creating a Composite Index**

SELECT

c.CustomerID,

c.Name,

SUM(od.Quantity \* p.Price) AS TotalSales

FROM Customers c

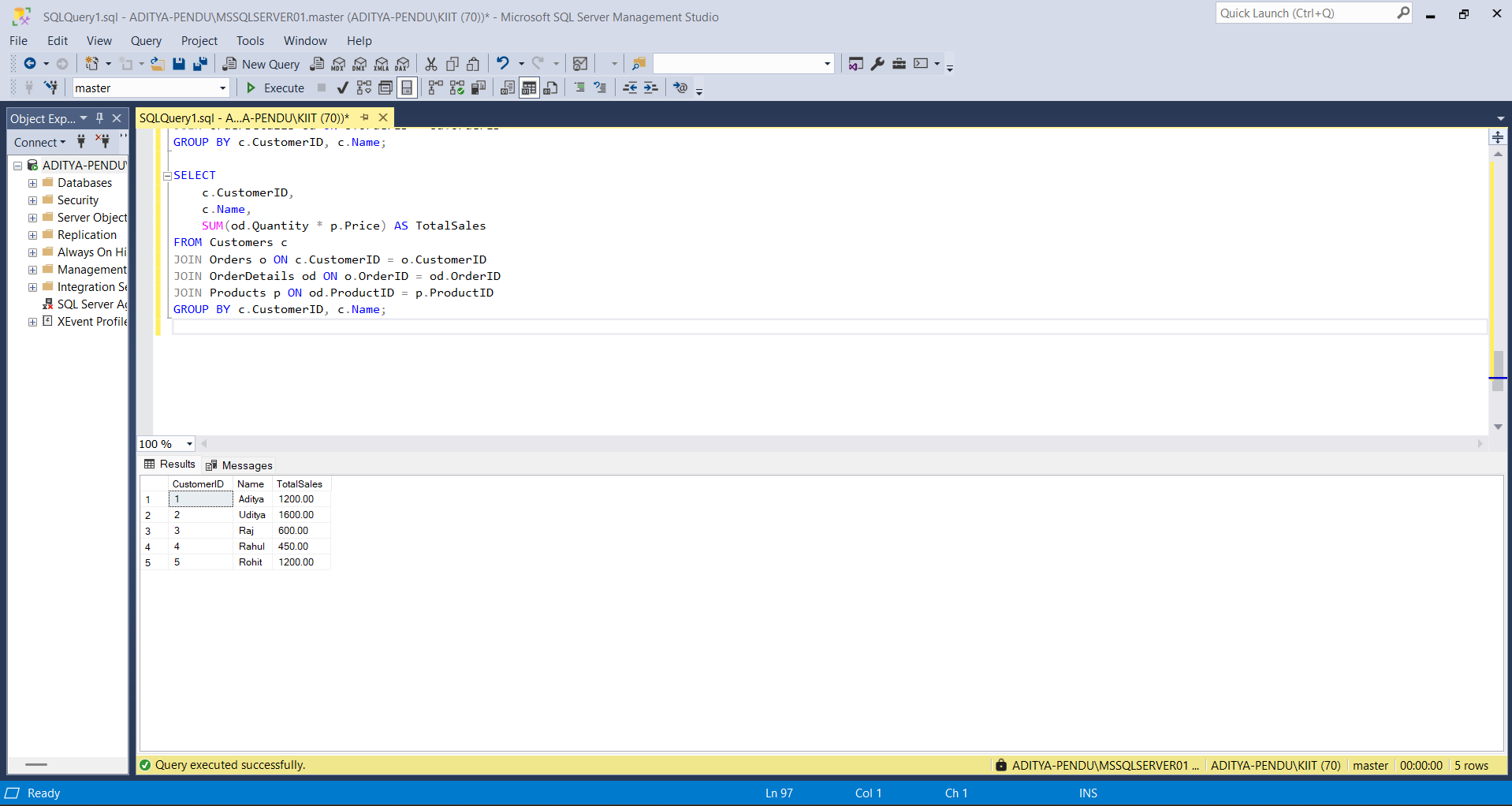
JOIN Orders o ON c.CustomerID = o.CustomerID

JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Products p ON od.ProductID = p.ProductID

GROUP BY c.CustomerID, c.Name;

Output :



**Exercise 3: SQL EXERCISE - Views.SQL :**

Question 1: Create a Simple View:

DROP VIEW IF EXISTS vw\_EmployeeBasicInfo;

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, 'Engineering'),

(2, 'Marketing'),

(3, 'HR');

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

(101, 'Aditya', 'Raj', 1, 60000.00, '2022-01-10'),

(102, 'Uditya', 'Kumar', 2, 55000.00, '2022-05-18'),

(103, 'Rahul', 'Mehta', 3, 50000.00, '2023-03-12');

CREATE VIEW vw\_EmployeeBasicInfo AS

SELECT

e.EmployeeID,

e.FirstName,

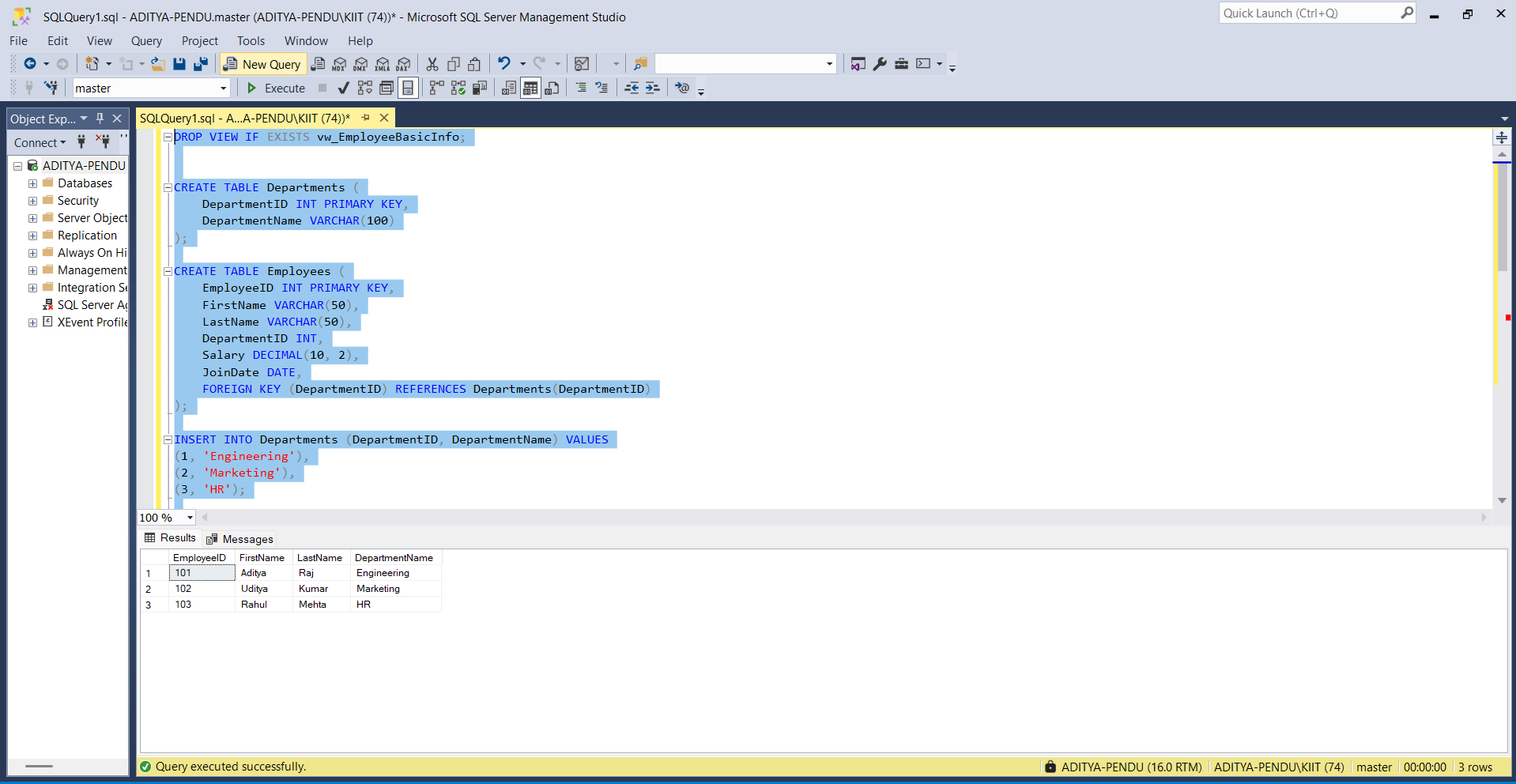
e.LastName,

d.DepartmentName

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID;

SELECT \* FROM vw\_EmployeeBasicInfo;

Output :

**Question2: Add Computed Column - Full Name**

CREATE VIEW vw\_EmployeeFullName AS

SELECT

e.EmployeeID,

e.FirstName,

e.LastName,

e.FirstName + ' ' + e.LastName AS FullName,

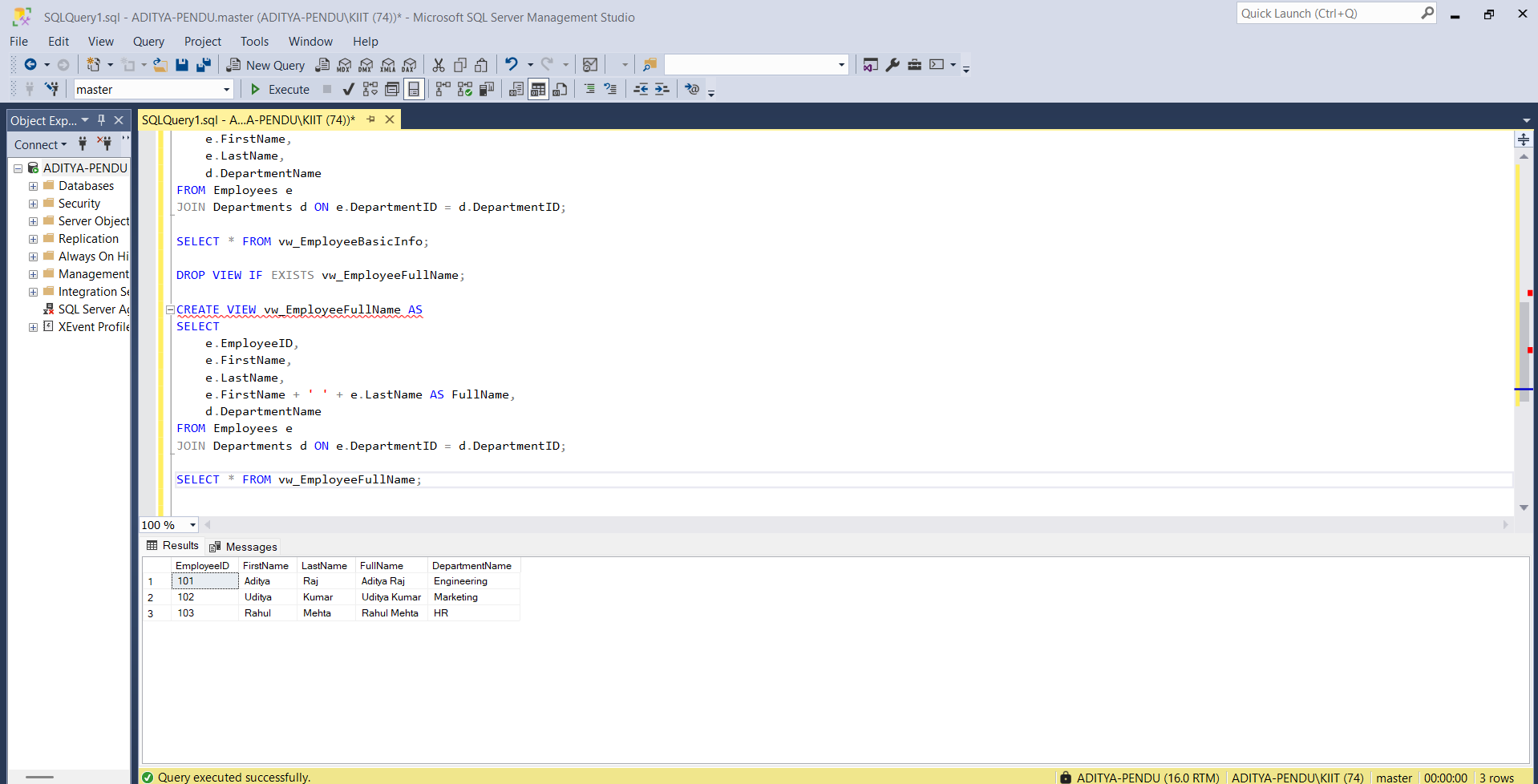
d.DepartmentName

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID;

SELECT \* FROM vw\_EmployeeFullName;

Output :



**Question 3: Add Computed Column - Annual Salary:**

CREATE VIEW vw\_EmployeeAnnualSalary AS

SELECT

e.EmployeeID,

e.FirstName,

e.LastName,

d.DepartmentName,

e.Salary,

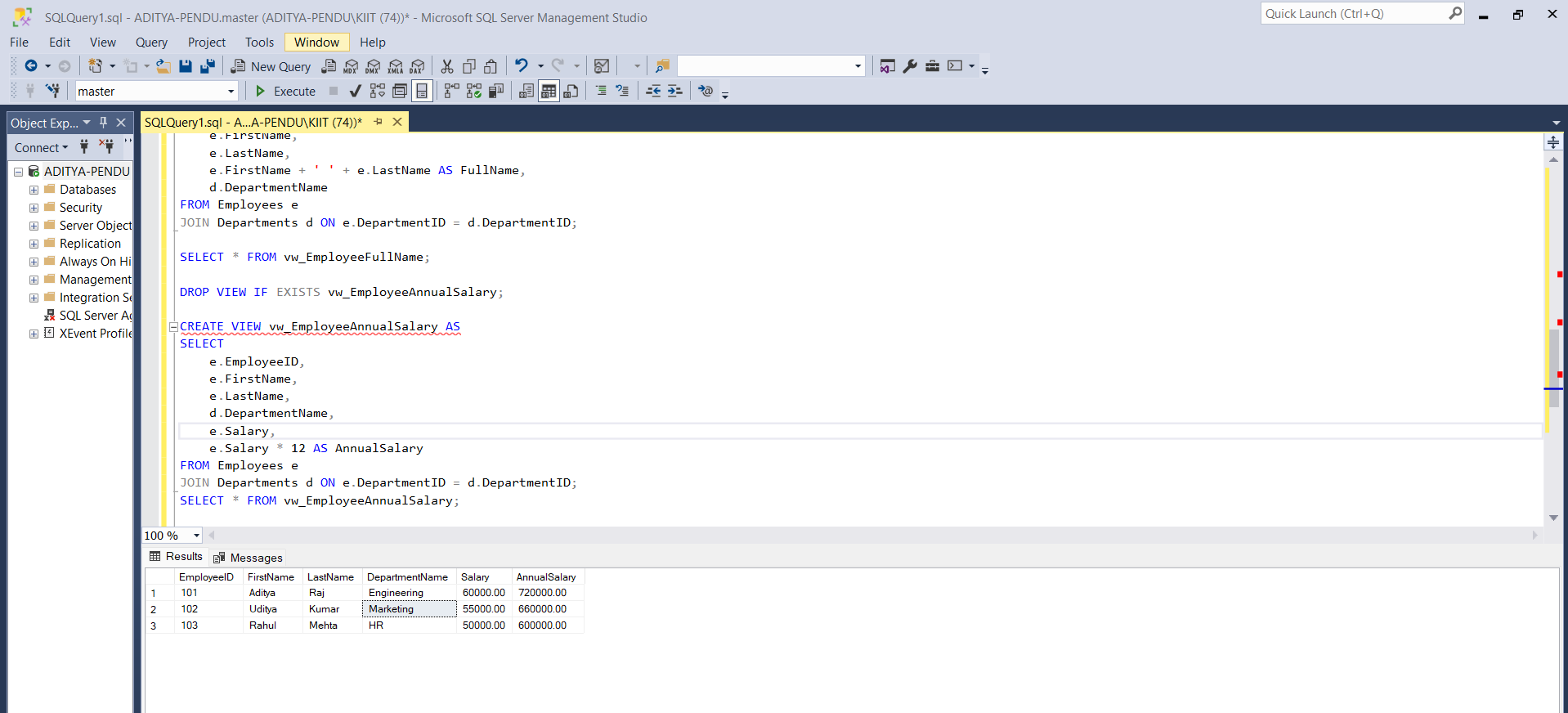
e.Salary \* 12 AS AnnualSalary

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID;

SELECT \* FROM vw\_EmployeeAnnualSalary;

Output:



Exercise 4: Add Multiple Computed Columns

CREATE VIEW vw\_EmployeeReport AS

SELECT

e.EmployeeID,

e.FirstName + ' ' + e.LastName AS FullName,

d.DepartmentName,

e.Salary \* 12 AS AnnualSalary,

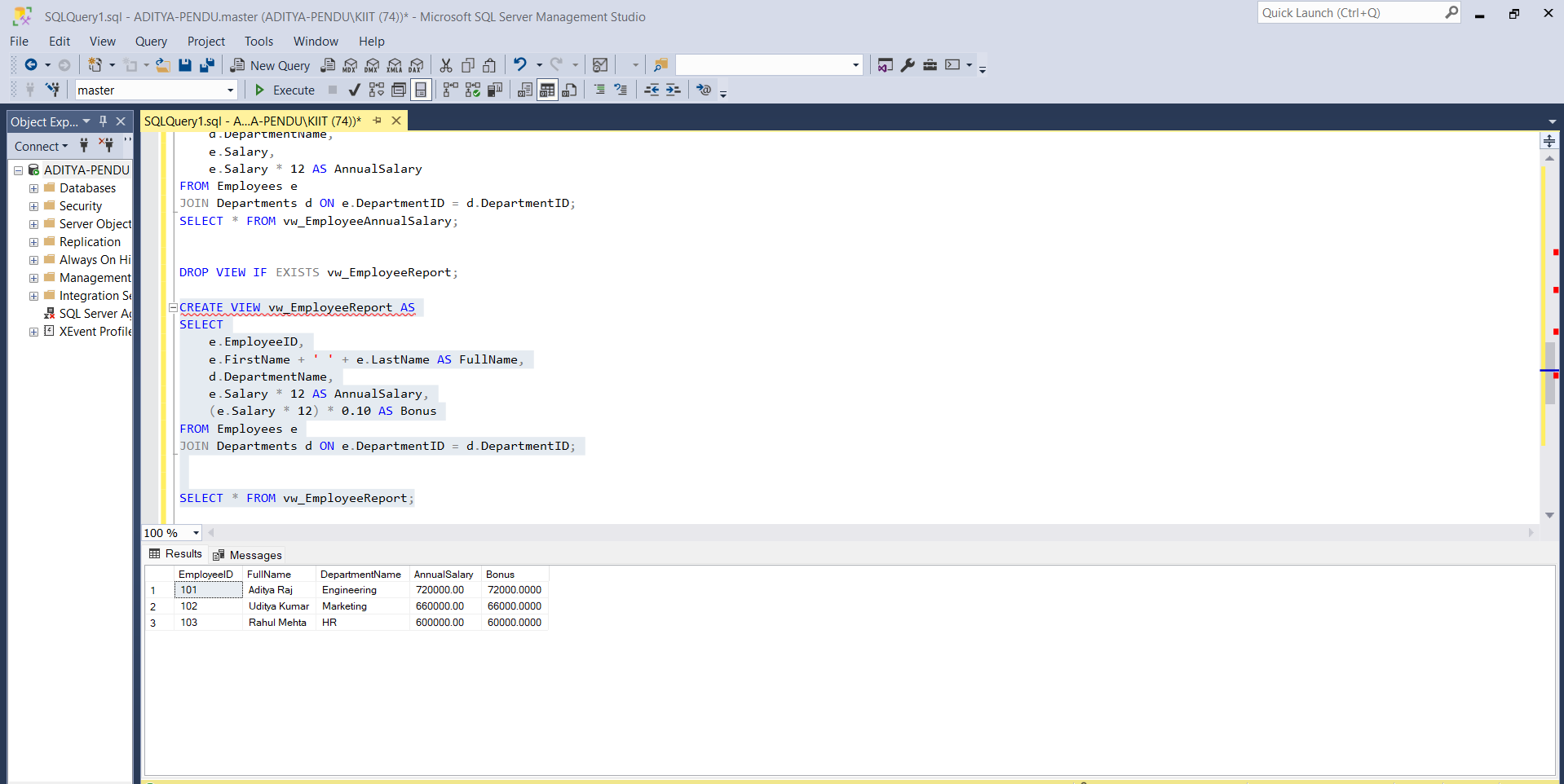
(e.Salary \* 12) \* 0.10 AS Bonus

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID;

SELECT \* FROM vw\_EmployeeReport;

Output



**Exercise 4: SQL EXERCISE - Stored Procedure.SQL :**

Question 1: Create a Stored Procedure

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

CREATE DATABASE CompanyDB;

GO

USE CompanyDB;

GO

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(50)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT FOREIGN KEY REFERENCES Departments(DepartmentID),

Salary DECIMAL(10,2),

JoinDate DATE

);

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');

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

SELECT

EmployeeID,

FirstName,

LastName,

DepartmentID,

Salary,

JoinDate

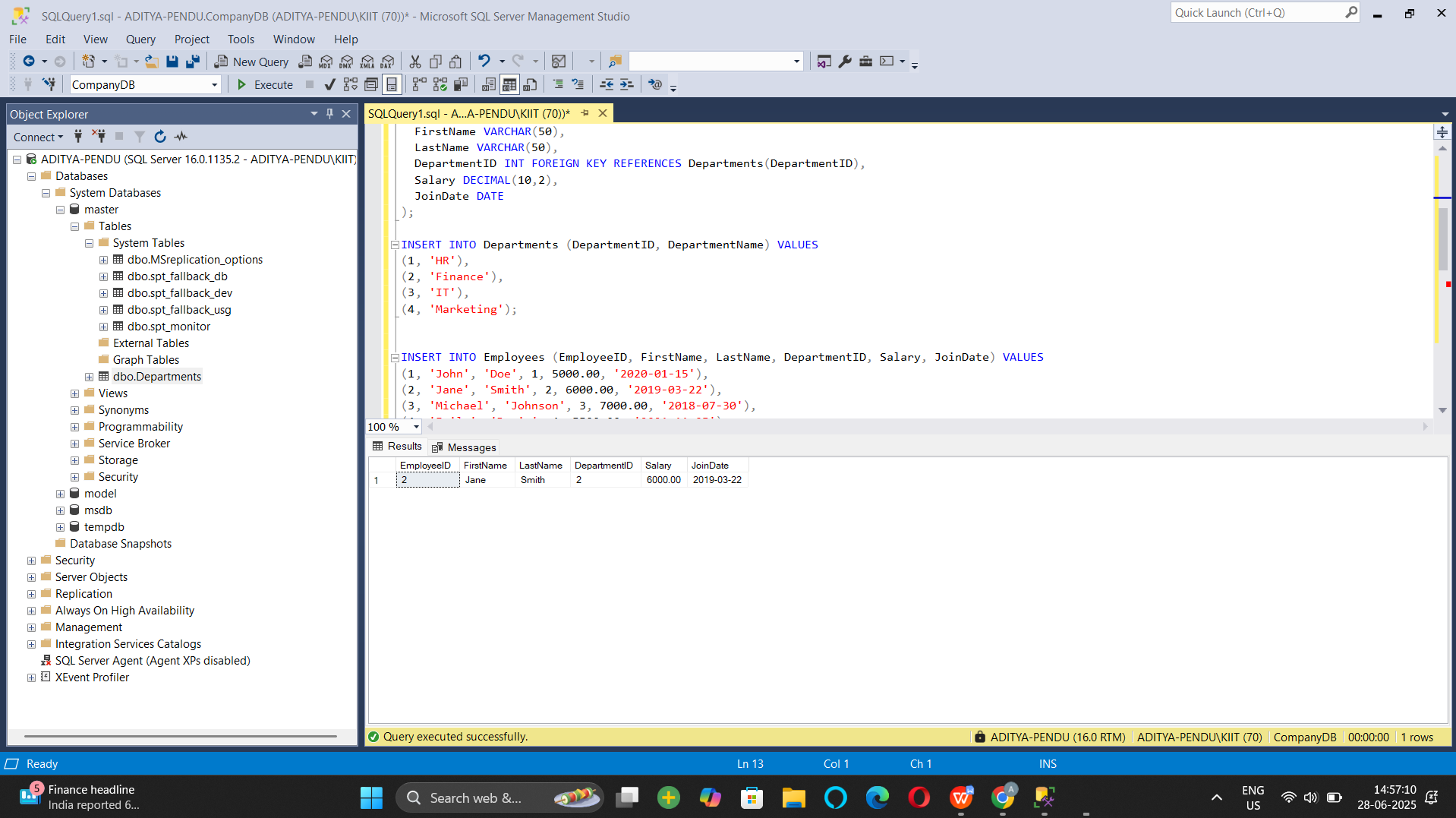
FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 2;

Output



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

CREATE TABLE Employees\_temp (

EmployeeID INT IDENTITY(1,1) PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT FOREIGN KEY REFERENCES Departments(DepartmentID),

Salary DECIMAL(10,2),

JoinDate DATE

);

INSERT INTO Employees\_temp (FirstName, LastName, DepartmentID, Salary, JoinDate)

SELECT FirstName, LastName, DepartmentID, Salary, JoinDate FROM Employees;

DROP TABLE Employees;

EXEC sp\_rename 'Employees\_temp', 'Employees';

EXEC sp\_InsertEmployee

@FirstName = 'Alice',

@LastName = 'Brown',

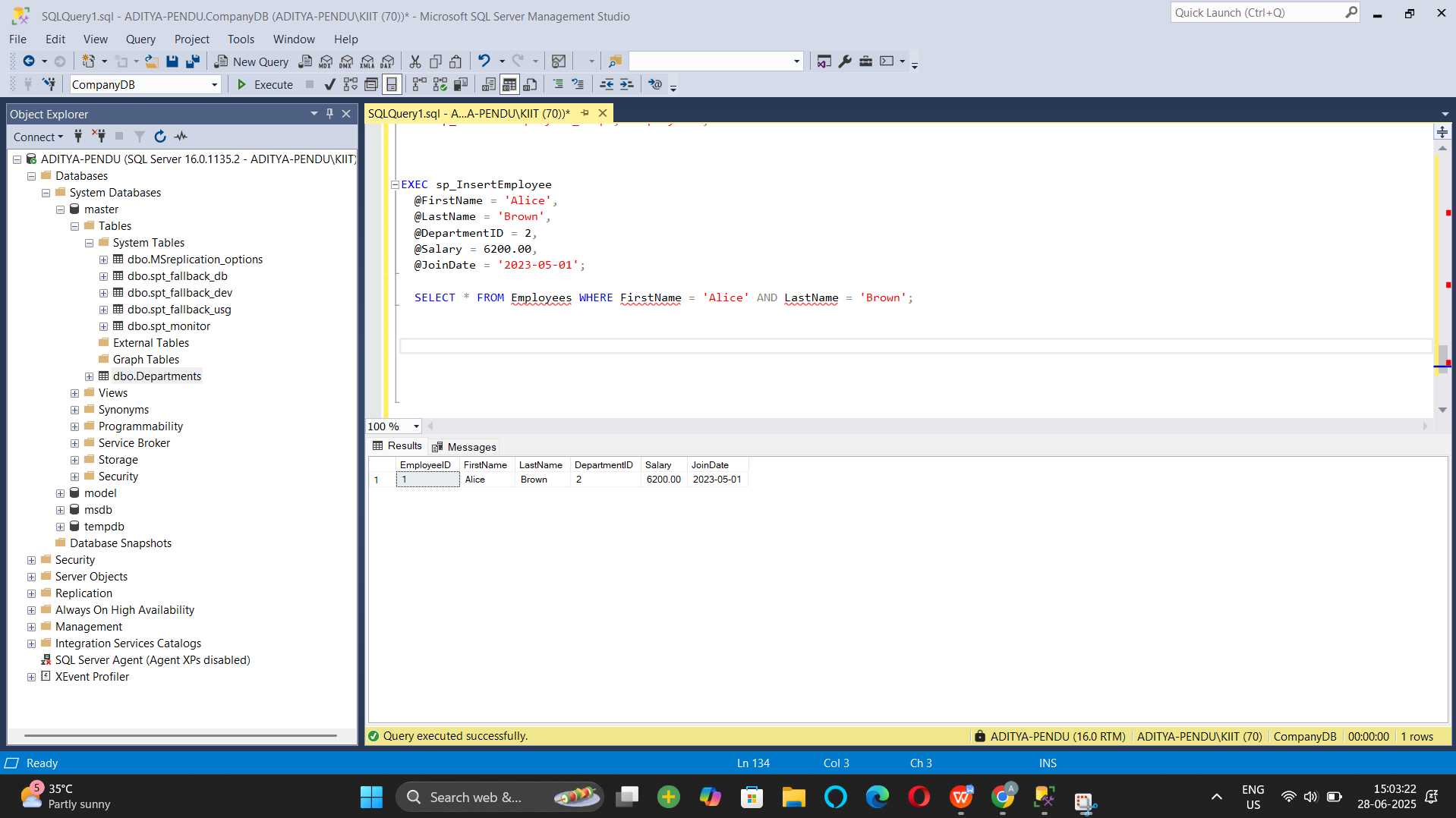
@DepartmentID = 2,

@Salary = 6200.00,

@JoinDate = '2023-05-01';

SELECT \* FROM Employees WHERE FirstName = 'Alice' AND LastName = 'Brown';

Output



1. **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;**

CREATE PROCEDURE sp\_InsertEmployee1

@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;

EXEC sp\_InsertEmployee1

@FirstName = 'David',

@LastName = 'Green',

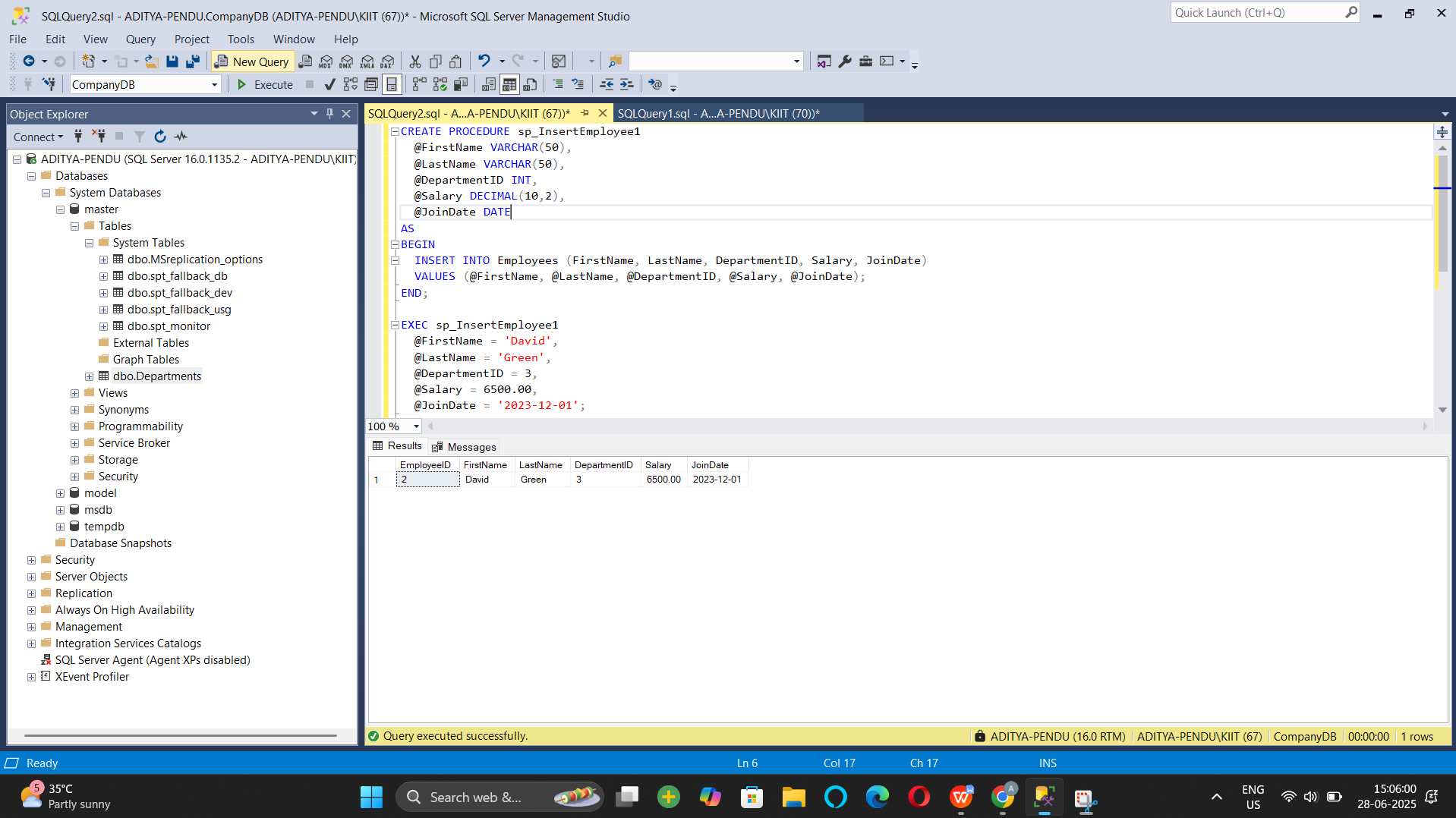
@DepartmentID = 3,

@Salary = 6500.00,

@JoinDate = '2023-12-01';

SELECT \* FROM Employees WHERE FirstName = 'David' AND LastName = 'Green';

Output



**Question 5: Return Data from a Stored Procedure Goal:**

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

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

CREATE PROCEDURE sp\_GetEmployeeCountByDepartment

@DepartmentID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

EXEC sp\_GetEmployeeCountByDepartment @DepartmentID = 2;

**Output :**



Nunit And Moq - Handson:

Exercise- 1:

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

namespace CalcLibrary

{

public class Calculator

{

public int Add(int a, int b)

{

return a + b;

}

}

}

using NUnit.Framework; // ✅ Make sure NUnit is installed

using CalcLibrary; // ✅ Add reference to CalcLibrary

namespace Nunit

{

[TestFixture]

public class CalculatorTests

{

private Calculator calc;

[SetUp]

public void SetUp()

{

calc = new Calculator(); // initialize before each test

}

[TearDown]

public void TearDown()

{

// Cleanup if needed after each test

}

[Test]

public void Add\_TwoNumbers\_ReturnsSum()

{

int result = calc.Add(2, 3);

Assert.That(result, Is.EqualTo(5));

}

[TestCase(1, 2, 3)]

[TestCase(5, 7, 12)]

[TestCase(-2, 3, 1)]

[TestCase(0, 0, 0)]

public void Add\_MultipleTestCases\_ReturnsExpected(int a, int b, int expected)

{

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

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

}

[Test, Ignore("This test is intentionally ignored")]

public void Subtract\_ThisIsPlaceholder()

{

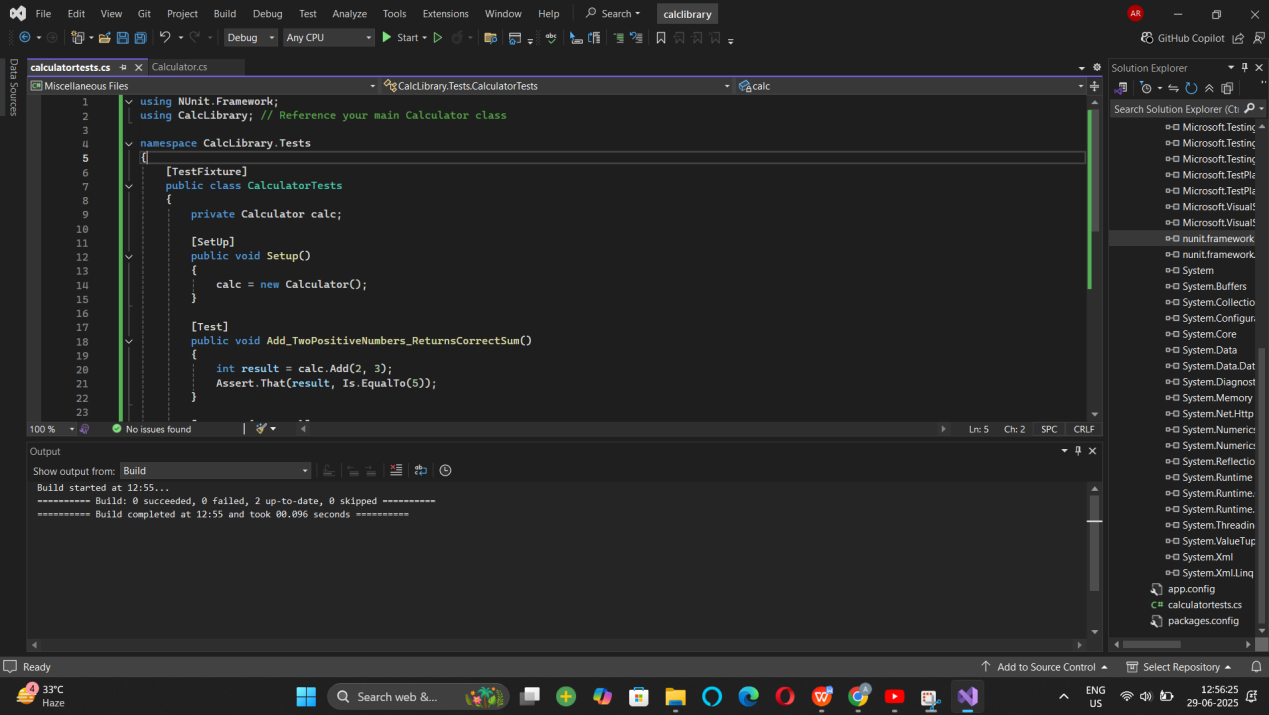
Assert.Fail("This is just a placeholder test.");

}

}

}

Output:



Exercise 2:

**Parameterized test cases**

1. Create test case to verify the subtraction feature of the calculator with various input types.

* Create test cases with ‘TestCase’ attribute to send in input parameters and the expected result.
* Add more than 1 ‘TestCase’ attributes to check various combinations for subtractions.
* Use Assert.Equal to check the actual and expected results

1. Create a test case to verify the multiplication concepts of calculator

* Create test cases with ‘TestCase’ attribute to send in input parameters and the expected result.
* Add more than 1 ‘TestCase’ attributes to check various combinations for subtractions.
* Use Assert.Equal to check the actual and expected results

1. Create a test case to verify the division logic of the calculator

* Create test cases with ‘TestCase’ attribute to send in input parameters and the expected result.
* Add more than 1 ‘TestCase’ attributes to check various combinations for subtractions.
* Use Assert.Equal to check the actual and expected results
* In one of the inputs, provide the divisor value to be 0
  + Use Try Catch block to catch the ArgumentException
  + Use Assert.Fail to notify the user that the test case has failed. Give the message “Division by zero” in the Assert.Fail, which will be notified to the user. This message will be seen in the test explorer.

**Test void methods**

In the MathLibrary class there is a property “GetResult”. The result of every operation is stored in a variable ‘result’. This value is accessed by the property.

The class also has a method “AllClear” that sets the value of the result variable to 0.

* Create a test method ‘TestAddAndClear’
* Invoke the Addition method of the math class library
* Verify if the expected and Actual results match using Assert.AreEqual
* Invoke the ‘AllClear’ method
* Use Assert.AreEqual to check if the result is 0 or not

using NUnit.Framework;

using CalcLibrary;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CalcLibrary.Tests

{

[TestFixture]

public class CalculatorTests

{

[Test]

public void Add\_TwoNumbers\_ReturnsCorrectSum()

{

var calc = new Calculator();

int result = calc.Add(5, 3);

Assert.That(result, Is.EqualTo(8));

; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CalcLibrary

{

public class Calculator

{

private int result;

public int GetResult => result;

public int Add(int a, int b)

{

result = a + b;

return result;

}

public int Subtract(int a, int b)

{

result = a - b;

return result;

}

public int Multiply(int a, int b)

{

result = a \* b;

return result;

}

public int Divide(int a, int b)

{

if (b == 0)

throw new ArgumentException("Cannot divide by zero");

result = a / b;

return result;

}

public void AllClear()

{

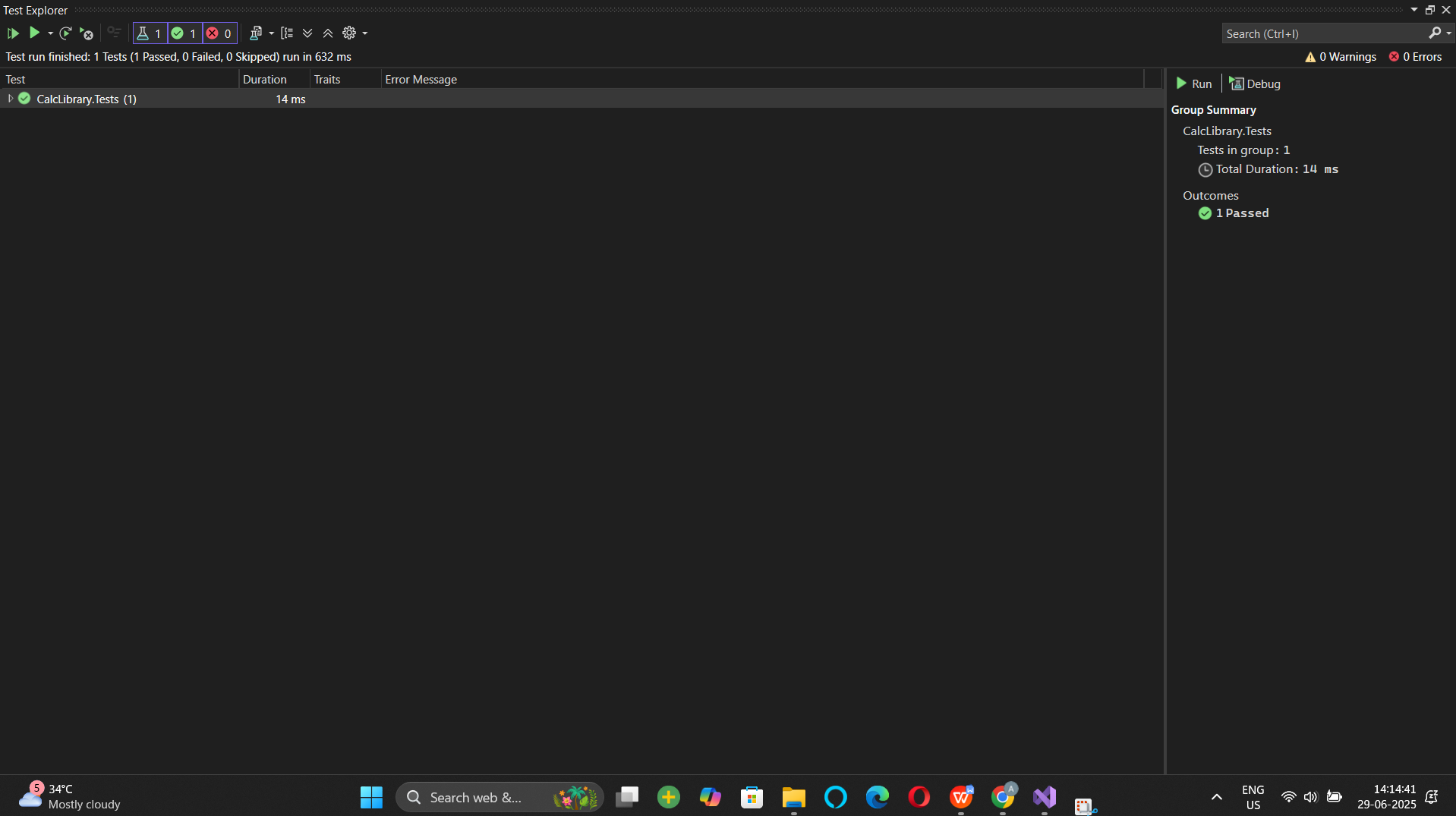
result = 0;

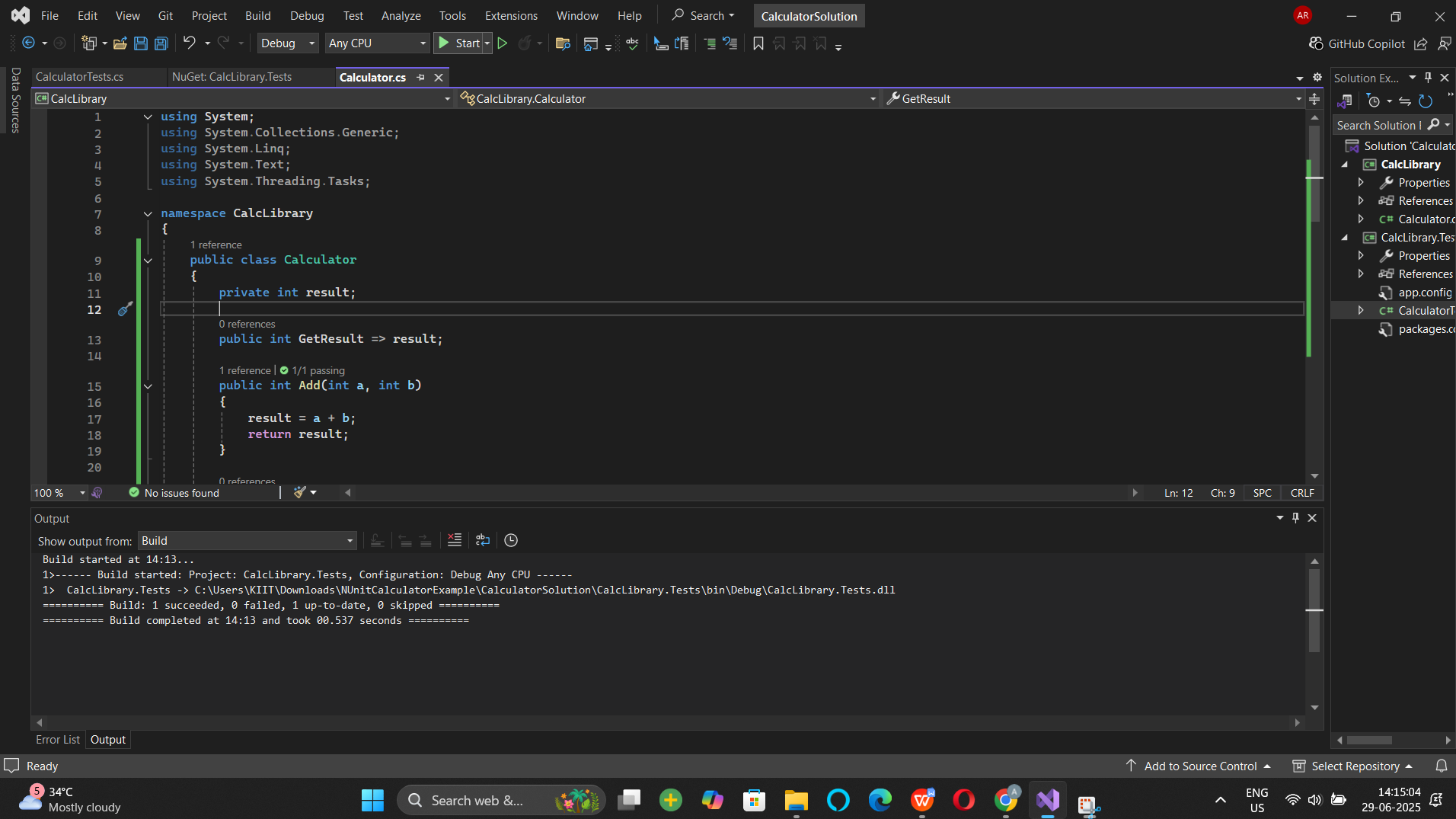
}

}

}

Output :





Exercise 3:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace UtilLib

{

public class UrlHostNameParser

{

public string ParseHostName(string url)

{

string protocol = url.Split(':')[0];

if (protocol.Equals("http")|| protocol.Equals("https"))

{

string hostName = url.Split(':')[1].Substring(2).Split('/')[0];

return hostName;

}

else

{

throw new FormatException("Url is not in correct format");

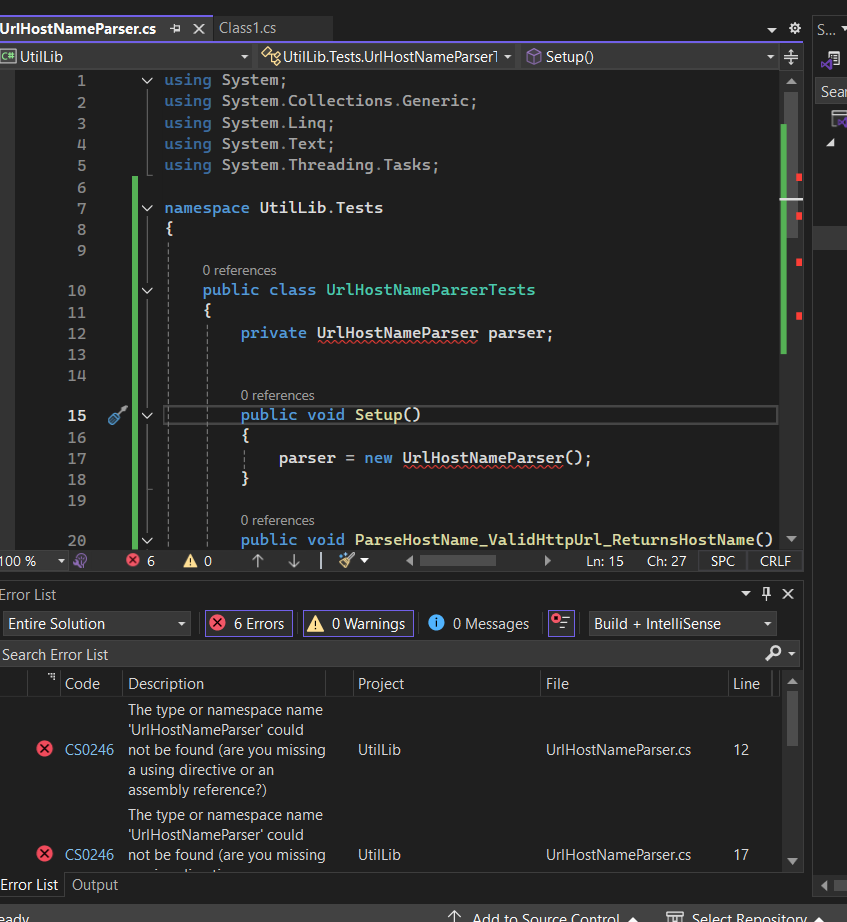
}

}

}

}

Output:



Exercise 4:

Create a Unit Test Project using NUnit Framework for the following requirement. Click [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/AccountsManagerLib.zip?csf=1&web=1&e=rtHrQD) to download the source project.

You are given a user account module called **AccountsManagerLib** using which an employee can login to the EMS portal using his/her credentials. If he/she provides valid login credentials, then the function should return the following message.

“Welcome <user\_id>!!!”

In case user provides invalid details, the function should return the following error message.

“Invalid user id/password”

If user doesn’t provide either userid or password, the function should throw an ArgumentException.

Valid Credentials are the following

|  |  |
| --- | --- |
| **User Id** | **Password** |
| user\_11 | secret@user11 |
| user\_22 | secret@user22 |

**Recommendations:**

Test Project Name:*<ClassLib\_Project>.Tests*

Test Class Name: *<SUT>Tests*

Test Method Name:  *UnitUnderTest\_Scenario\_ExpectedOutcome*

After writing the above test methods, run the tests and assert the results with that of the success/failure messages.

**Note:**

* *Enforce the Single Assertion Rule*
* *Use Assert.That()*

**Steps to perform:**

1. Create a Class Library project in the same solution which is provided and name it as suggested.
2. Rename the class file name (<SUT>Tests.cs).
3. Add the assembly reference of the UtilLib project to the test project.
4. Additionally add the reference of both NUnit and NUnit3TestAdapter in the test project using NuGet Package Manager (NPM).
5. Write the suggested test methods.
6. Run your tests.
7. Break the test by modifying the source project functionality.
8. Rerun the test.
9. Observe the test result.

Exercise 5:

# Objectives

* This lab will help you become skilled at writing automated unit tests using the NUnit framework.
* Explain & demonstrate various NUnit custom attributes to identify tests
* Explain & Demonstrate on CollectionAssert

Create a Unit Test Project using NUnit Framework for the following requirement. Click [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/CollectionsLib.zip?csf=1&web=1&e=LQC8Tc) to download the source project.

You have been given a source project called **CollectionsLib** that deals with set of collection objects. Write test methods for the below scenarios. Make sure that your tests pass. You may modify the collection values in the source project in order to make the test passed. Use the appropriate assert functionalities.

The method called **GetEmployees** returnsa collection of Employee object.

*Scenario 1*

* Ensure that there is no null value in the collection

*Scenario 2*

* Verify whether the employee having his/her id 100 exists in the collection.

*Scenario 3*

* Check whether the GetEmployees function returns only unique employees. If employee id is different in every employee object, then the list is considered as unique list. You may modify the source project so that you can achieve the goal here.

**Hint:** Override Equals() & GetHashCode() methods in Empoyee class.

*Scenario 3*

* Both **GetEmployees()** and **GetEmployeesWhoJoinedInPreviousYears()** return a set of employee object. Verify whether all items in both the collections are same or not.

Try both Classic Model as well as Constraint Model of Assertions while writing test cases.

**Recommendations:**

Test Project Name:*<ClassLib\_Project>.Tests*

Test Class Name: *<SUT>Tests*

Test Method Name:  *UnitUnderTest\_Scenario\_ExpectedOutcome*

**Note:**

* *Enforce the Single Assertion Rule*
* *Use Assert.That()*

**Steps to perform**

1. Create a Class Library project in the same solution which is provided and name it as suggested.
2. Rename the class file name (<SUT>Tests.cs).
3. Add the assembly reference of the UtilLib project to the test project.
4. Additionally add the reference of both NUnit and NUnit3TestAdapter in the test project using NuGet Package Manager (NPM).
5. Write the suggested test methods.
6. Run your tests.
7. Break the test by modifying the source project functionality.
8. Rerun the test.
9. Observe the test result.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CollectionsLib

{

public class Employee

{

public int EmpId { get; set; }

public string EmpName { get; set; }

public double Salary { get; set; }

public DateTime DOJ { get; set; }

}

public class EmployeeManager

{

private static readonly List<Employee> employees;

static EmployeeManager()

{

employees = new List<Employee>

{

new Employee { EmpId=100, EmpName="John",DOJ=DateTime.Now.AddYears(-5),Salary=30000},

new Employee { EmpId=101, EmpName="Mary",DOJ=DateTime.Now.AddYears(-2),Salary=10000},

new Employee { EmpId=102, EmpName="Steve",DOJ=DateTime.Now.AddYears(-2),Salary=10000},

new Employee { EmpId=103, EmpName="Allen",DOJ=DateTime.Now.AddYears(-7),Salary=50000},

};

}

public List<Employee> GetEmployees()

{

return employees;

}

public List<Employee> GetEmployeesWhoJoinedInPreviousYears()

{

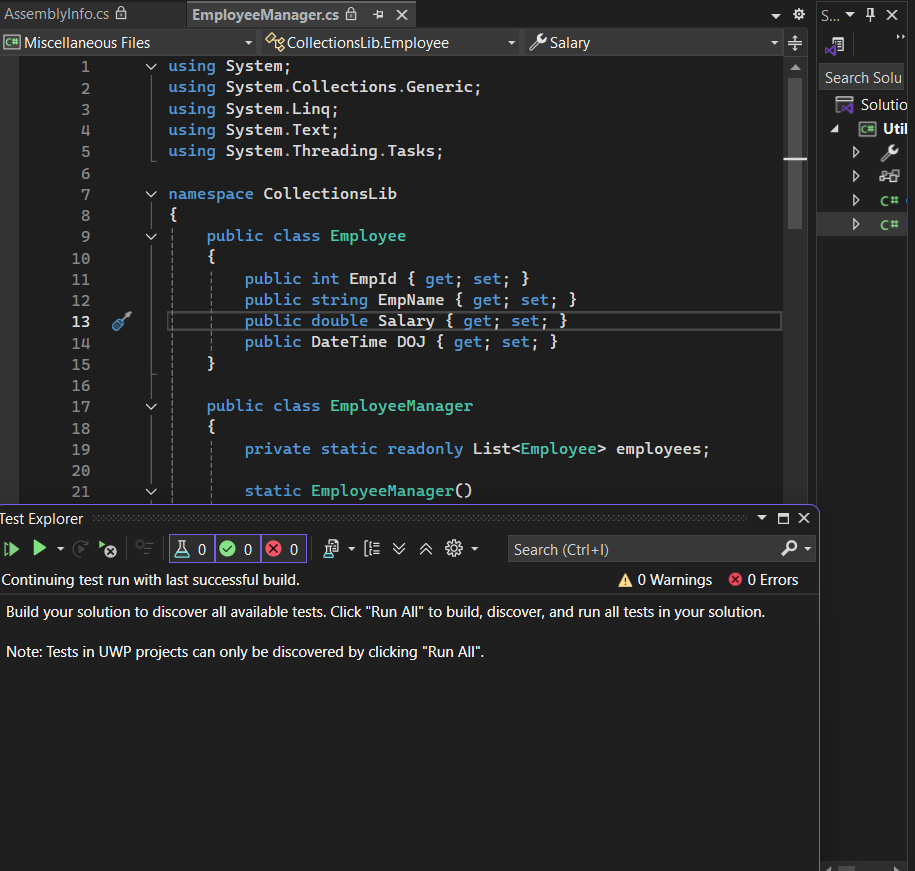
return employees.FindAll(x=>x.DOJ<DateTime.Now);

}

}

}

output



# Objectives

* This lab will help you become skilled at writing automated unit tests using the NUnit framework.
* Explain & Demonstrate TestCaseSource to minimize amount of code used for testing.

Create a Unit Test Project using NUnit Framework for the following requirement. Click [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/FourSeasonsLib.zip?csf=1&web=1&e=VY3YQa) to download the source project **FourSeasonsLib**.

Following is the business scenario implemented in the source project.

* The function accepts a month name and returns what season in that month (both inclusive) based on certain logic. The criteria is elaborated in the matrix below.

|  |  |  |
| --- | --- | --- |
| **Seasons** | **Month** | **Climate** |
| Spring | February to March | Sunny and pleasant |
| Summer | April to June | Hot |
| Monsoon | July to September | Wet, hot and humid |
| Autumn | September to November | Pleasant |
| Winter | December to January | Very Cool |

Make sure that you are not writing multiple test methods that are equal to the number of execution paths. Your focus should be of writing minimum code while unit testing.

Try both straight forward and alternate ways of working with the TestCaseSource attribute for the above scenario.

**Recommendations:**

Test Project Name:*<ClassLib\_Project>.Tests*

Test Class Name: *<SUT>Tests*

Test Method Name:  *UnitUnderTest\_Scenario\_ExpectedOutcome*

**Note:**

* *Enforce the Single Assertion Rule*
* *Use Assert.That()*

**Steps to perform**

1. Create a Class Library project in the same solution which is provided and name it as suggested.
2. Rename the class file name (<SUT>Tests.cs).
3. Add the assembly reference of the ConverterLib project to the test project.
4. Additionally add the reference of NUnit, NUnit3TestAdapter and Moq in the test project using NuGet Package Manager (NPM).
5. Write the suggested test methods.
6. Run your tests.
7. Break the test by modifying the source project functionality.
8. Rerun the test.
9. Observe the test result.

Exercise 6:

# Objectives

* This lab will help you become skilled at writing automated unit tests using the NUnit framework.
* Explain & Demonstrate TestCaseSource to minimize amount of code used for testing.

Create a Unit Test Project using NUnit Framework for the following requirement. Click [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/FourSeasonsLib.zip?csf=1&web=1&e=VY3YQa) to download the source project **FourSeasonsLib**.

Following is the business scenario implemented in the source project.

* The function accepts a month name and returns what season in that month (both inclusive) based on certain logic. The criteria is elaborated in the matrix below.

|  |  |  |
| --- | --- | --- |
| **Seasons** | **Month** | **Climate** |
| Spring | February to March | Sunny and pleasant |
| Summer | April to June | Hot |
| Monsoon | July to September | Wet, hot and humid |
| Autumn | September to November | Pleasant |
| Winter | December to January | Very Cool |

Make sure that you are not writing multiple test methods that are equal to the number of execution paths. Your focus should be of writing minimum code while unit testing.

Try both straight forward and alternate ways of working with the TestCaseSource attribute for the above scenario.

**Recommendations:**

Test Project Name:*<ClassLib\_Project>.Tests*

Test Class Name: *<SUT>Tests*

Test Method Name:  *UnitUnderTest\_Scenario\_ExpectedOutcome*

**Note:**

* *Enforce the Single Assertion Rule*
* *Use Assert.That()*

**Steps to perform**

1. Create a Class Library project in the same solution which is provided and name it as suggested.
2. Rename the class file name (<SUT>Tests.cs).
3. Add the assembly reference of the ConverterLib project to the test project.
4. Additionally add the reference of NUnit, NUnit3TestAdapter and Moq in the test project using NuGet Package Manager (NPM).
5. Write the suggested test methods.
6. Run your tests.
7. Break the test by modifying the source project functionality.
8. Rerun the test.
9. Observe the test result.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SeasonsLib

{

public class SeasonTeller

{

public string DisplaySeasonBy(string monthName)

{

string seasonName;

if (monthName.Equals("February", StringComparison.OrdinalIgnoreCase) || monthName.Equals("March", StringComparison.OrdinalIgnoreCase))

{

seasonName = "Spring";

}

else if (monthName.Equals("April", StringComparison.OrdinalIgnoreCase) || monthName.Equals("May", StringComparison.OrdinalIgnoreCase) || monthName.Equals("June", StringComparison.OrdinalIgnoreCase))

{

seasonName = "Summer";

}

else if (monthName.Equals("July", StringComparison.OrdinalIgnoreCase) || monthName.Equals("August", StringComparison.OrdinalIgnoreCase) || monthName.Equals("September", StringComparison.OrdinalIgnoreCase))

{

seasonName = "Monsoon";

}

else if (monthName.Equals("October", StringComparison.OrdinalIgnoreCase) || monthName.Equals("November", StringComparison.OrdinalIgnoreCase))

{

seasonName = "Autumn";

}

else if (monthName.Equals("December", StringComparison.OrdinalIgnoreCase) || monthName.Equals("January", StringComparison.OrdinalIgnoreCase))

{

seasonName = "Winter";

}

else

{

return "Invalid Season";

}

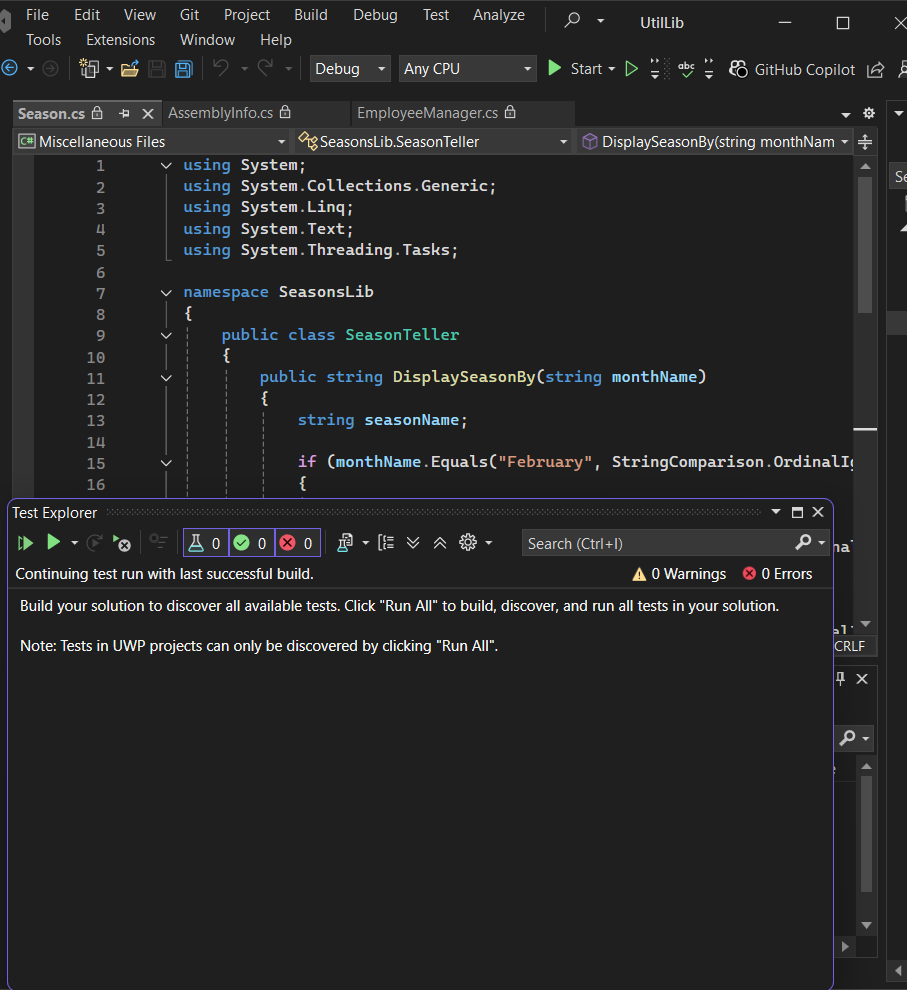
return seasonName;

}

}

}

Output



Exercise 7:

# bjectives

* This lab will help you become skilled at writing automated unit tests using the NUnit framework.
* Explain the benefit of writing parameterized test cases.
* Demonstrate on parameterized test cases using an example.

Create a Unit Test Project using NUnit Framework for the following requirement. Click [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/LeapYearCalculatorLib.zip?csf=1&web=1&e=hQwwAa) to download the source project.

The **LeapYearCalculatorLib** application tells the user whether the entered year is a leap year or not. Also it checks the input value to make sure that the data given is a valid year.

* If the given year is a Leap Year, the program will output 1. If it’s not, then the program will result 0.
* Any value between 1753 and 9999 (both inclusive) will be a valid year. Violation of this rule will result -1.

**Recommendations:**

Test Project Name:*<ClassLib\_Project>.Tests*

Test Class Name: *<SUT>Tests*

Test Method Name:  *UnitUnderTest\_Scenario\_ExpectedOutcome*

After writing the above test methods, run the tests and assert the results with that of the success/failure messages.

**Note:**

* *Enforce the Single Assertion Rule*
* *Use Assert.That()*

**Steps to perform:**

1. Create a Class Library project in the same solution which is provided and name it as suggested.
2. Rename the class file name (<SUT>Tests.cs).
3. Add the assembly reference of the UtilLib project to the test project.
4. Additionally add the reference of both NUnit and NUnit3TestAdapter in the test project using NuGet Package Manager (NPM).
5. Write the suggested test methods.
6. Run your tests.
7. Break the test by modifying the source project functionality.
8. Rerun the test.
9. Observe the test result.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LeapYearCalculatorLib

{

public class LeapYearCalculator

{

public int IsLeapYear(int year)

{

int output;

if (year<1753||year>9999)

{

return -1;

}

if (((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0))

output = 1;

else

output = 0;

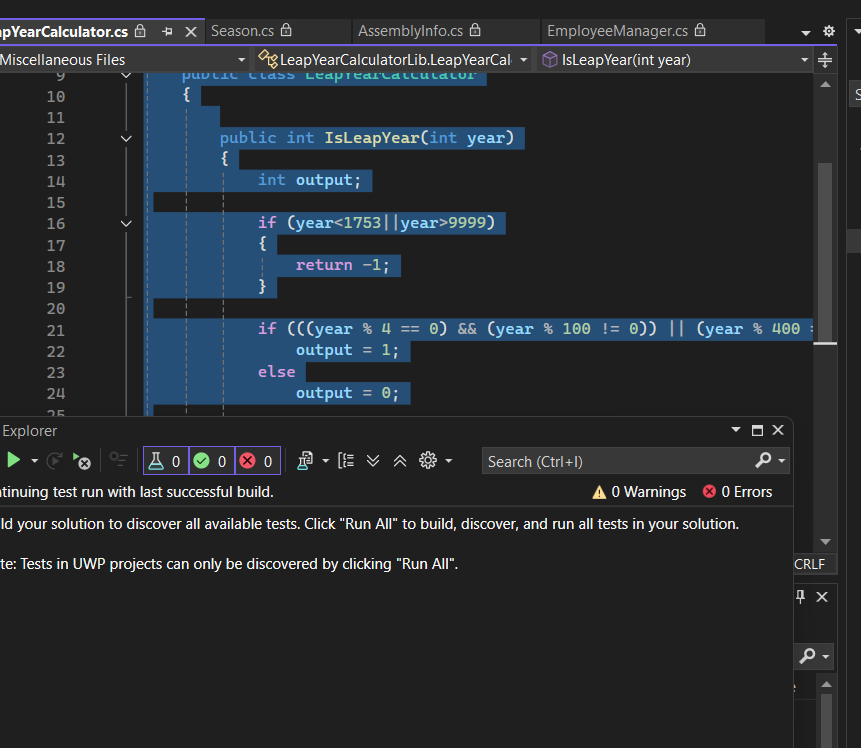
return output;

}

}

}

Output



Exercise 8:

# Objectives

* This lab will help you become skilled at writing automated unit tests using the NUnit framework.
* Demonstrate on testing methods that throw an exception.

Create a Unit Test Project using NUnit Framework for the following requirement. Click [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/UserManagerLib.zip?csf=1&web=1&e=cM5gWJ) to download the source project.

Following is the application logic of the given **UserManagerLib** source project.

User creation will be successful under the below given condition

* PANCardNo property reads only 10 characters length value from the user. It is a mandatory property while creating the user.

Following exceptions may occur while creating the user.

* + NullReferenceException- If the input value is empty or null
  + FormatException-If the input string does not meet the length criteria.
* While writing test cases for the above program, you need to ensure that you are handling all types of exception that may raise during the CreateUser method call. Also, write test method for happy path in the function.

**Recommendations:**

Test Project Name:*<ClassLib\_Project>.Tests*

Test Class Name: *<SUT>Tests*

Test Method Name:  *UnitUnderTest\_Scenario\_ExpectedOutcome*

**Note:**

* *Enforce the Single Assertion Rule*
* *Use Assert.That()*

**Steps to perform**

1. Create a Class Library project in the same solution which is provided and name it as suggested.
2. Rename the class file name (<SUT>Tests.cs).
3. Add the assembly reference of the UtilLib project to the test project.
4. Additionally add the reference of both NUnit and NUnit3TestAdapter in the test project using NuGet Package Manager (NPM).
5. Write the suggested test methods.
6. Run your tests.
7. Break the test by modifying the source project functionality.
8. Rerun the test.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace UserManagerLib

{

public class User

{

public Guid Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string EmailId { get; set; }

public string PANCardNo

{

get;

set;

}

public string ValidatePANCardNumber(string panCard)

{

if (string.IsNullOrEmpty(panCard))

{

throw new NullReferenceException("Invalid Pan Card Number");

}

else if (panCard.Length != 10)

{

throw new FormatException("Pan Card Number Should contain only 10 characters");

}

else

{

return "Valid";

}

}

public void CreateUser(User user)

{

if (ValidatePANCardNumber(user.PANCardNo).Equals("Valid"))

{

//Do something

}

}

}

}

Output

