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

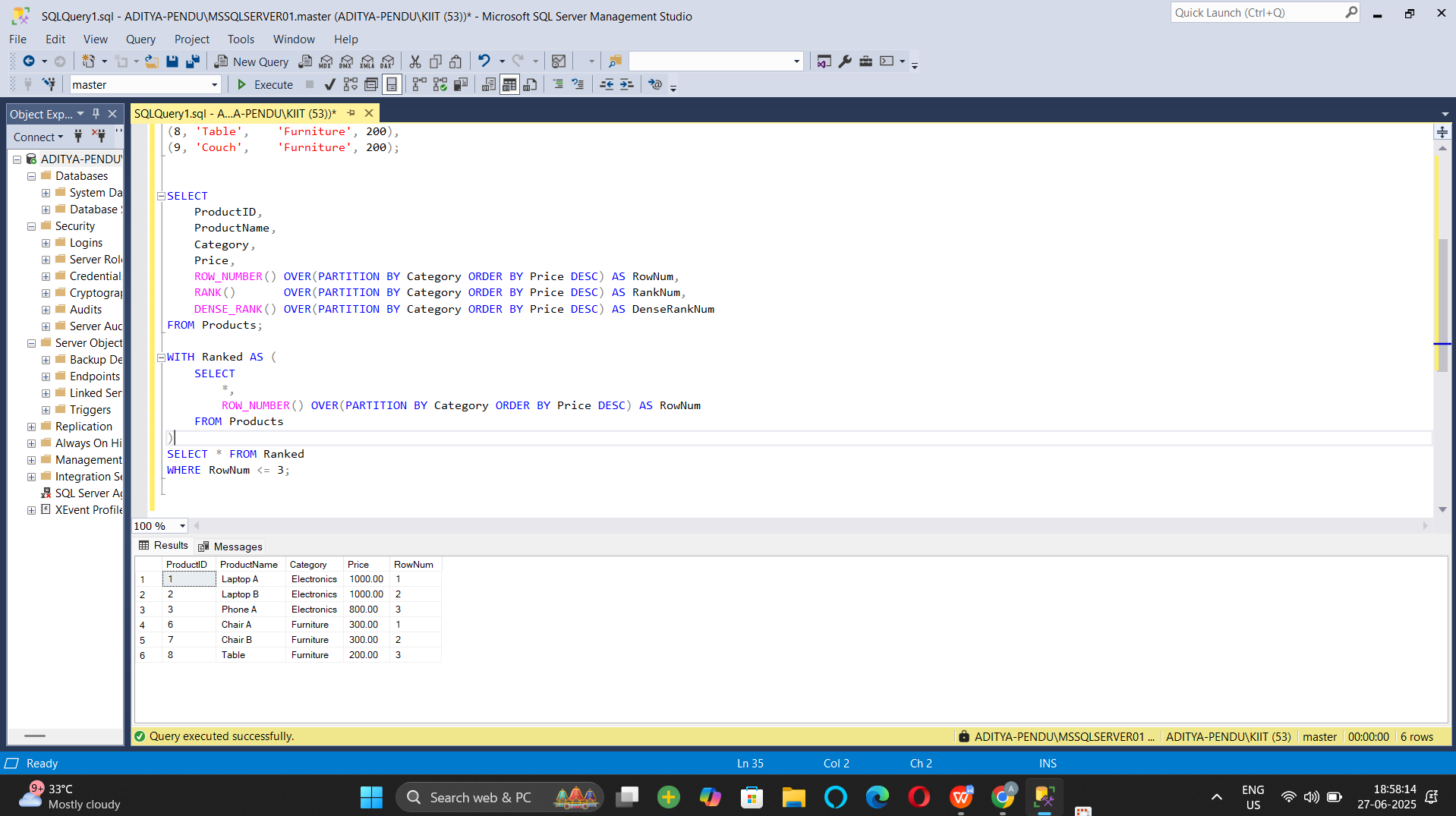
FROM Products

)

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),

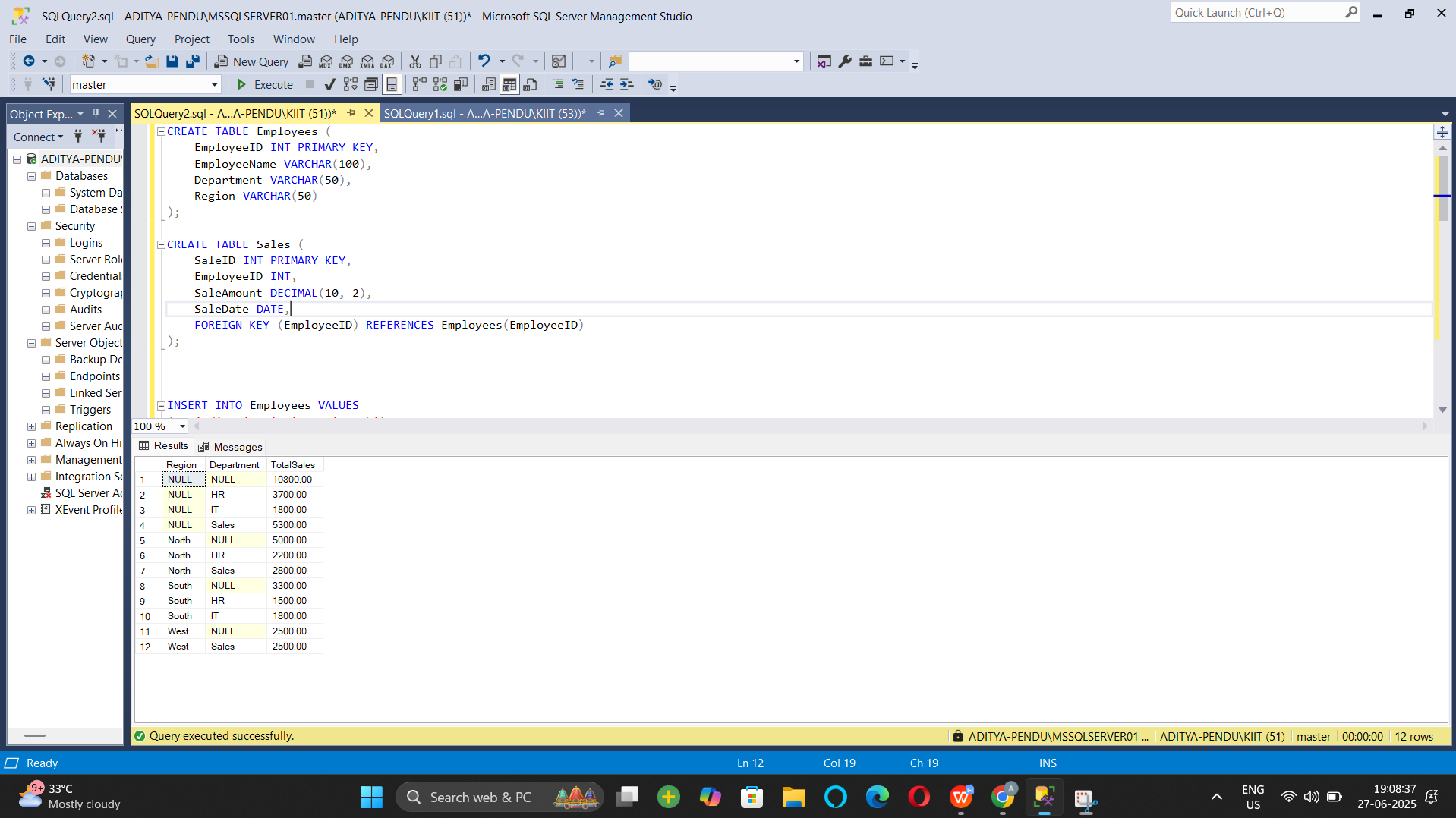
(E.Region, E.Department),

()

)

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),

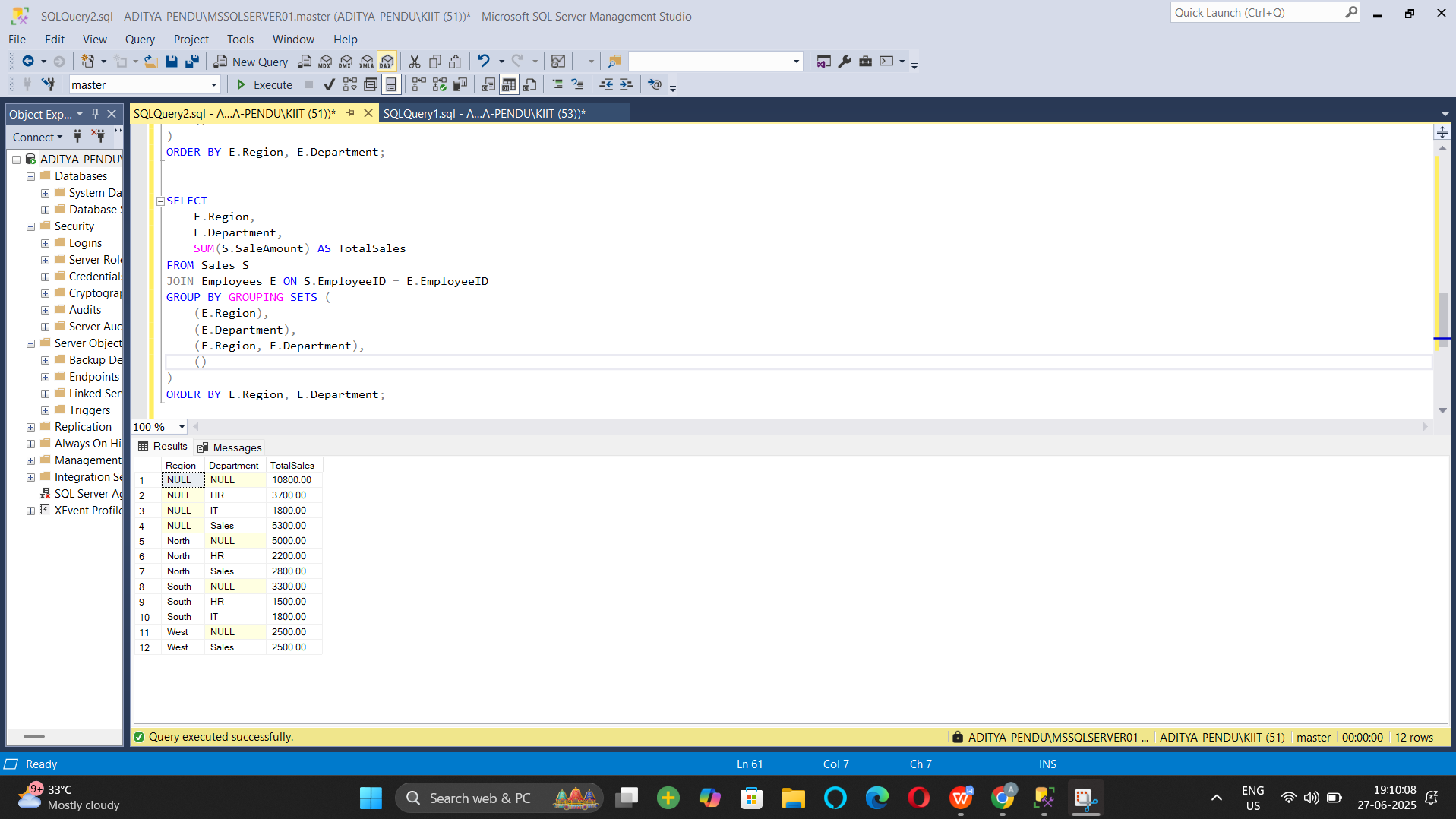
(E.Region, E.Department),

()

)

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

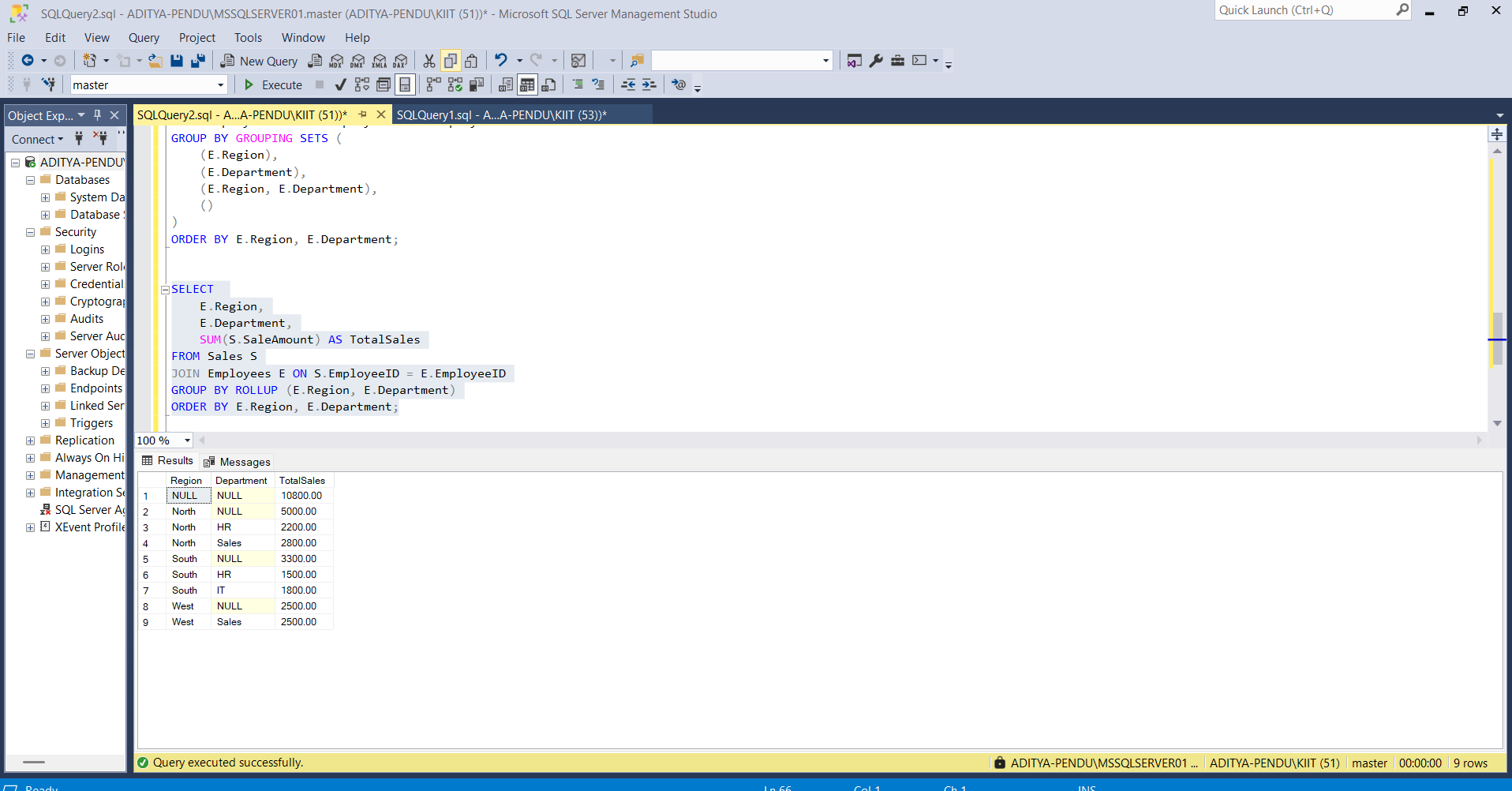
FROM Sales S

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

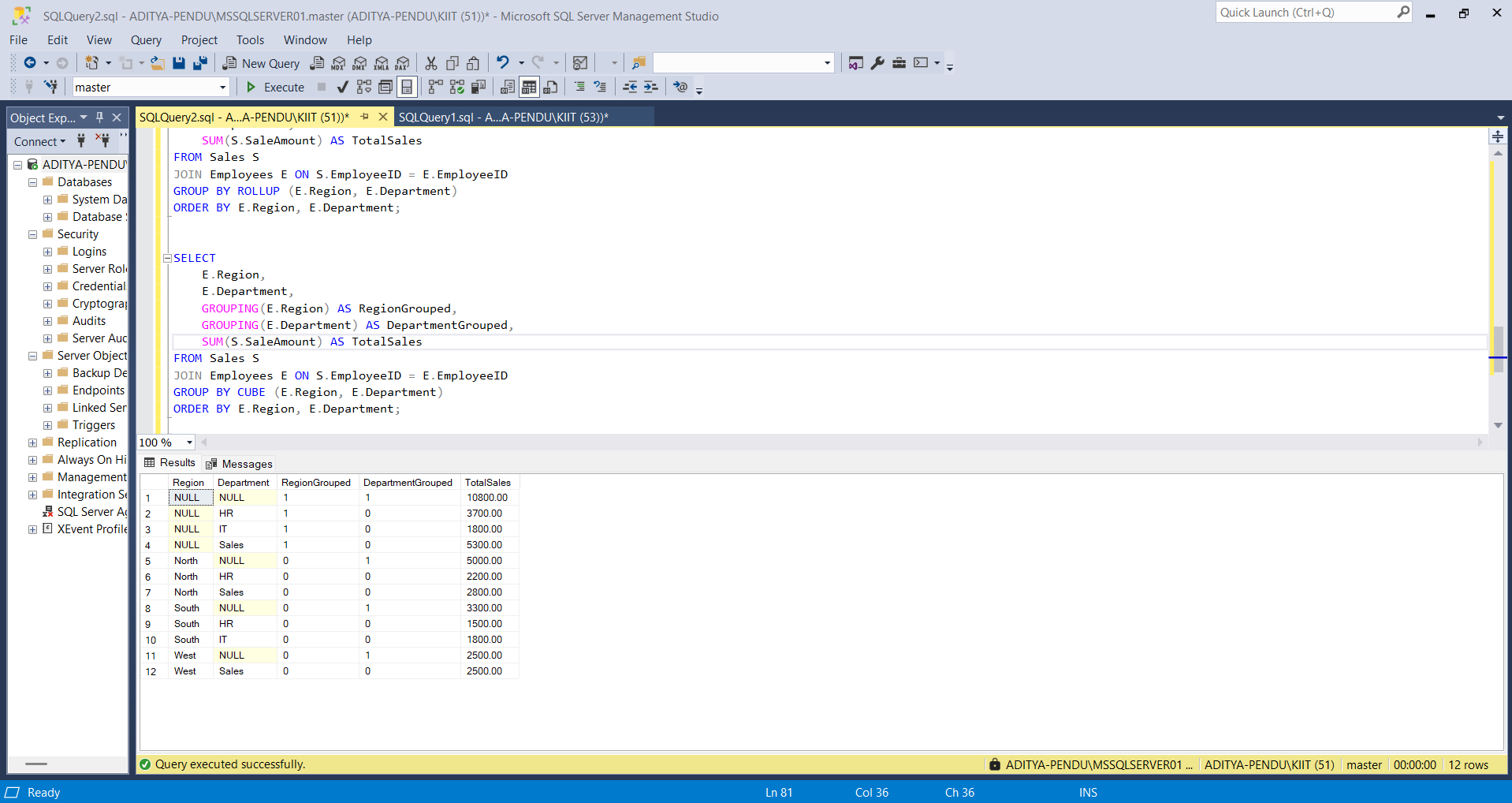
FROM Sales S

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

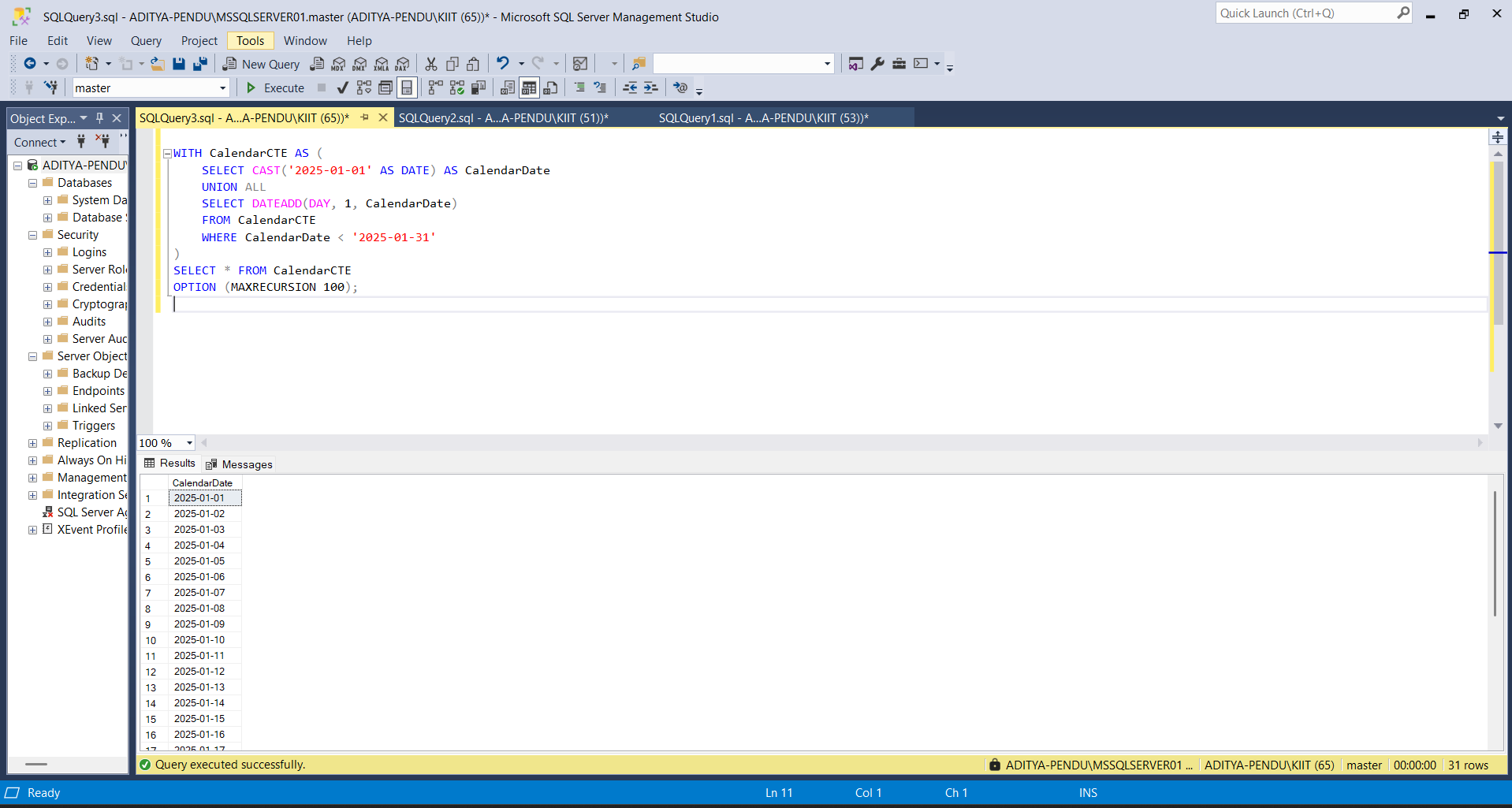
WHERE CalendarDate < '2025-01-31'

)

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

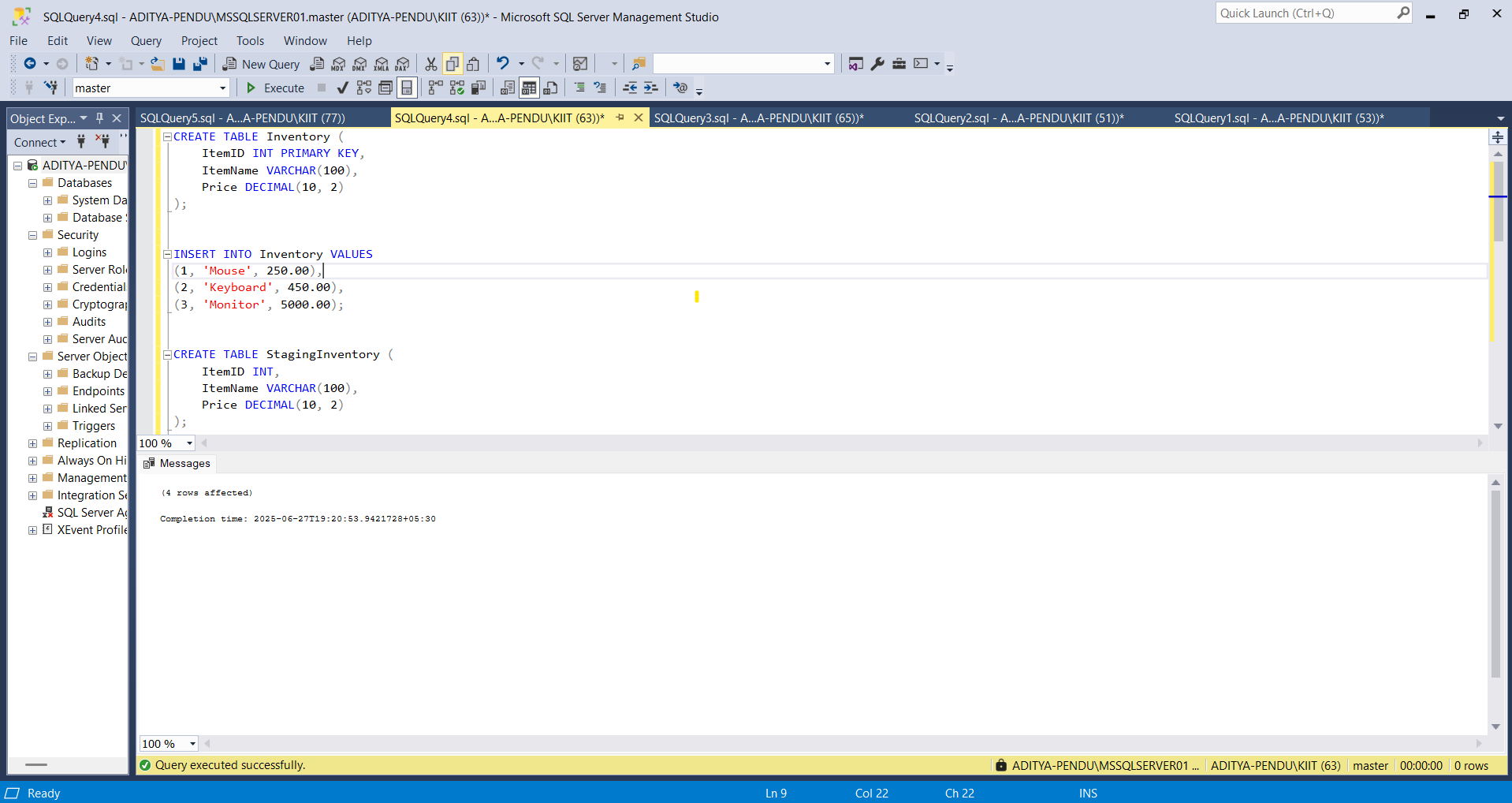
(1, 'Mouse', 275.00),

(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

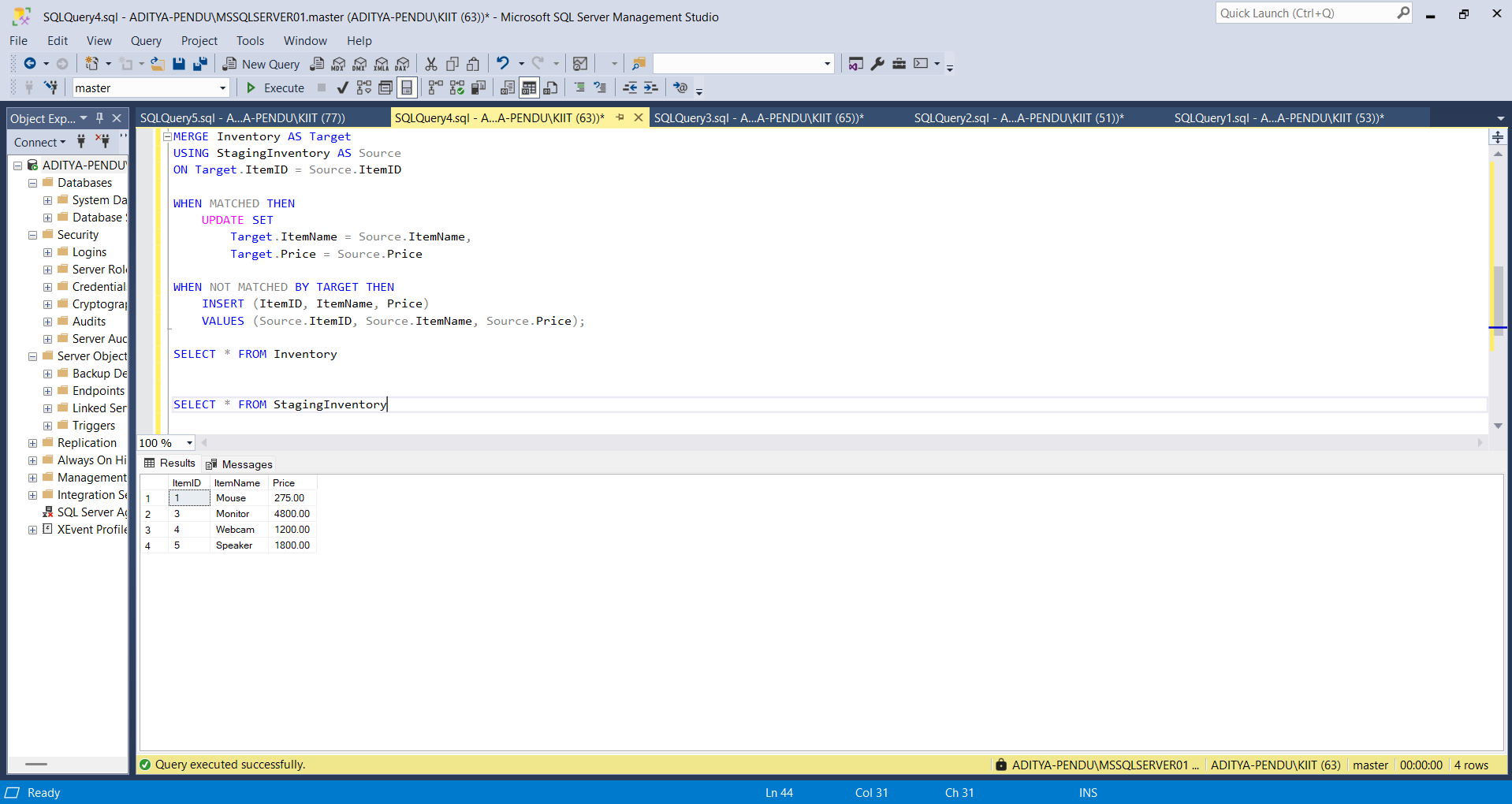
INSERT (ItemID, ItemName, Price)

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,

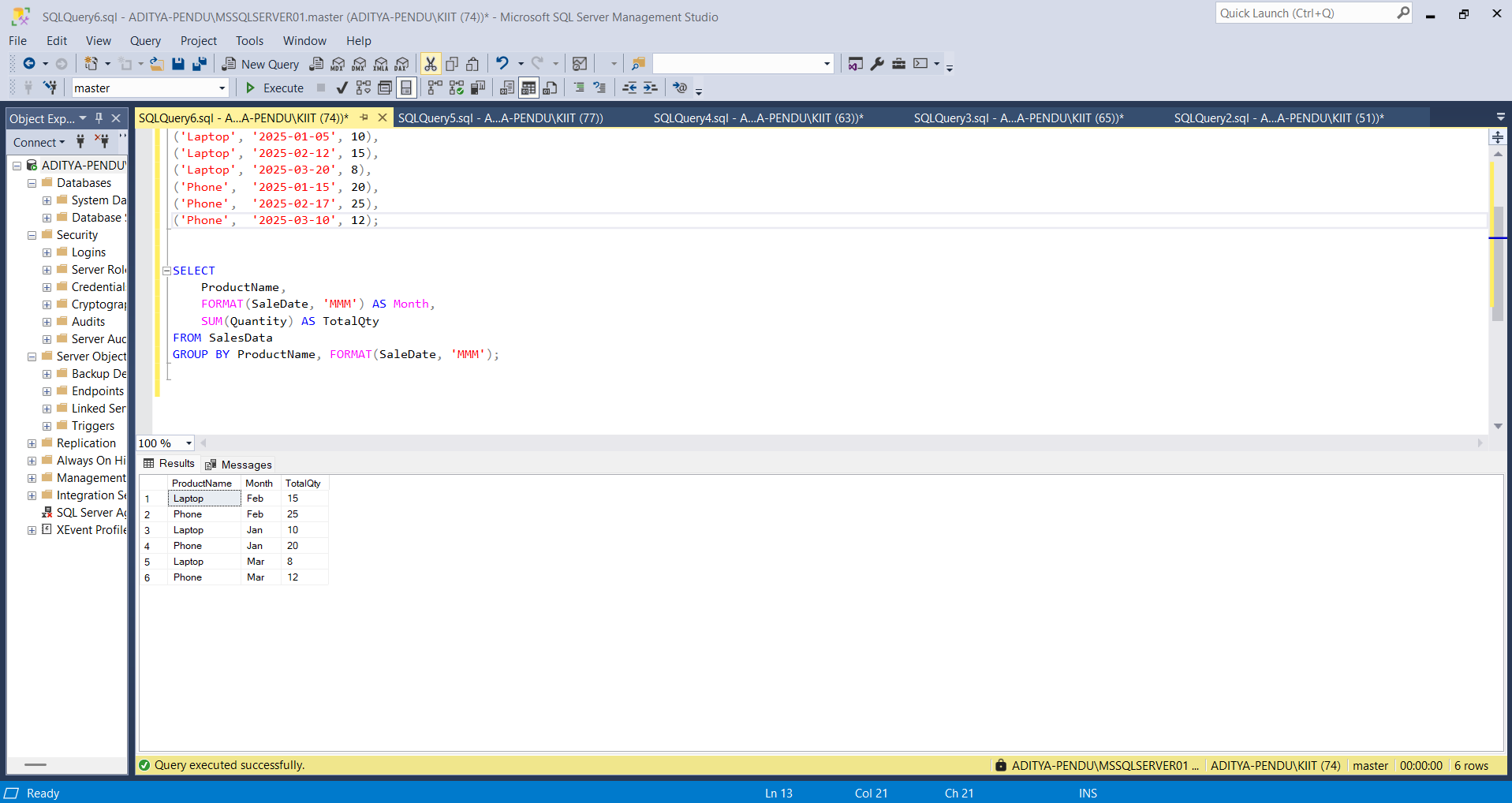
FORMAT(SaleDate, 'MMM') AS Month,

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

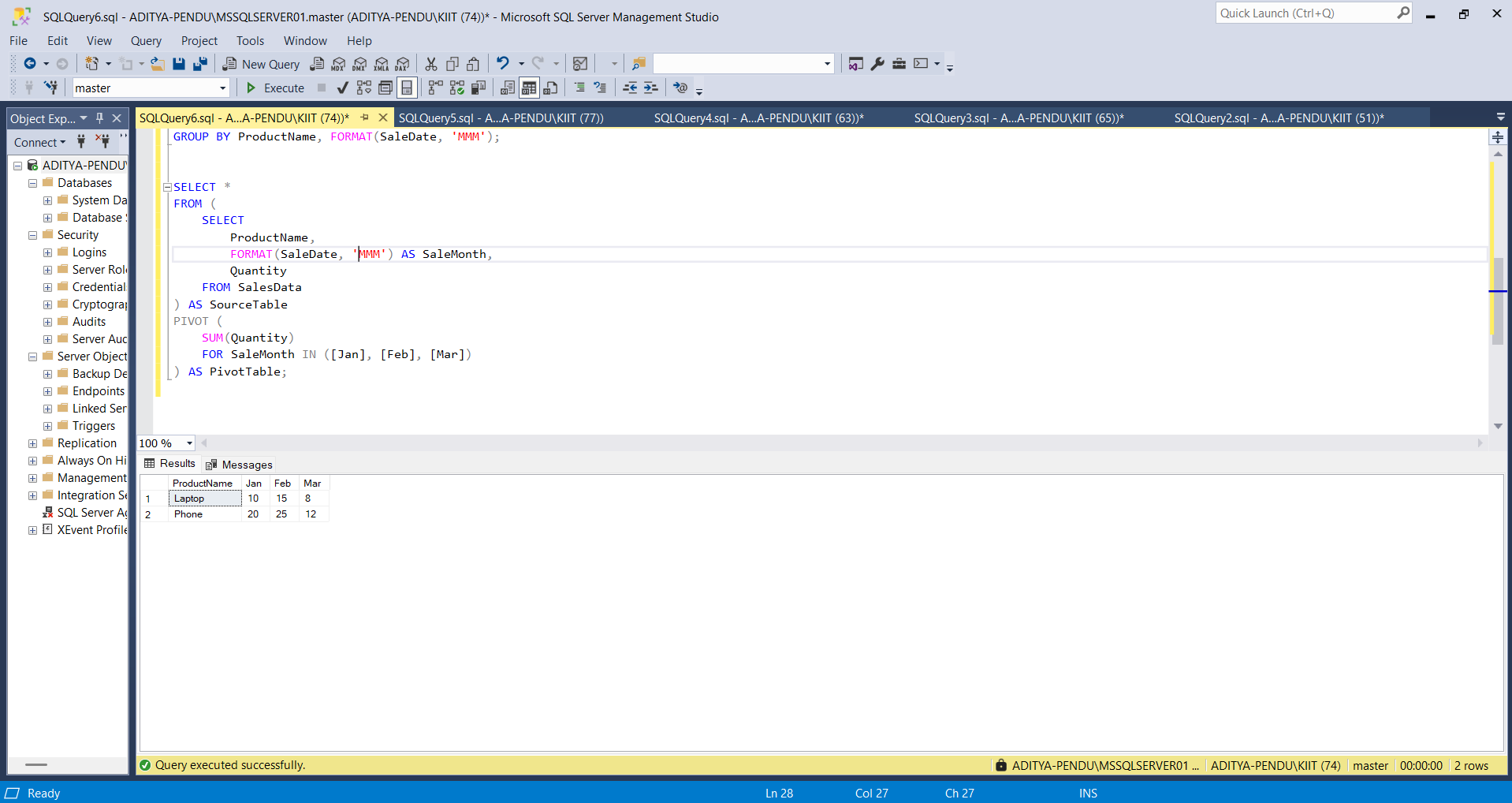
PIVOT (

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

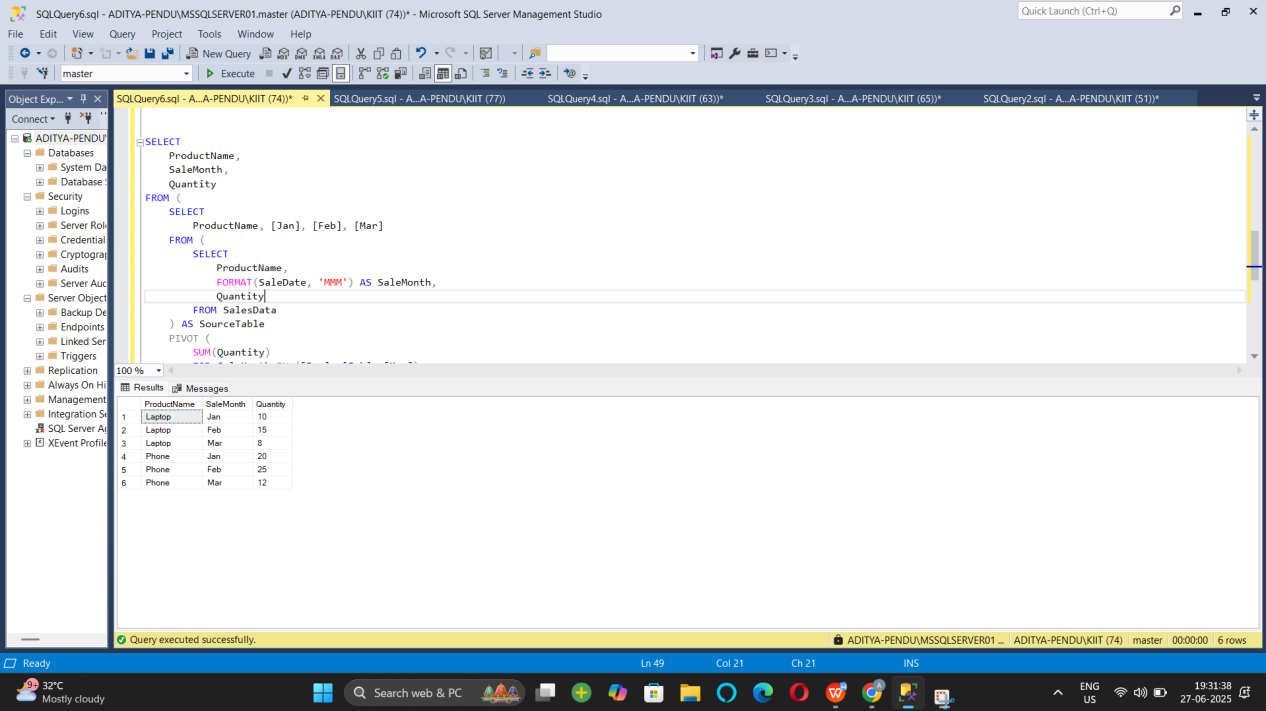
) AS PivotedTable

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,

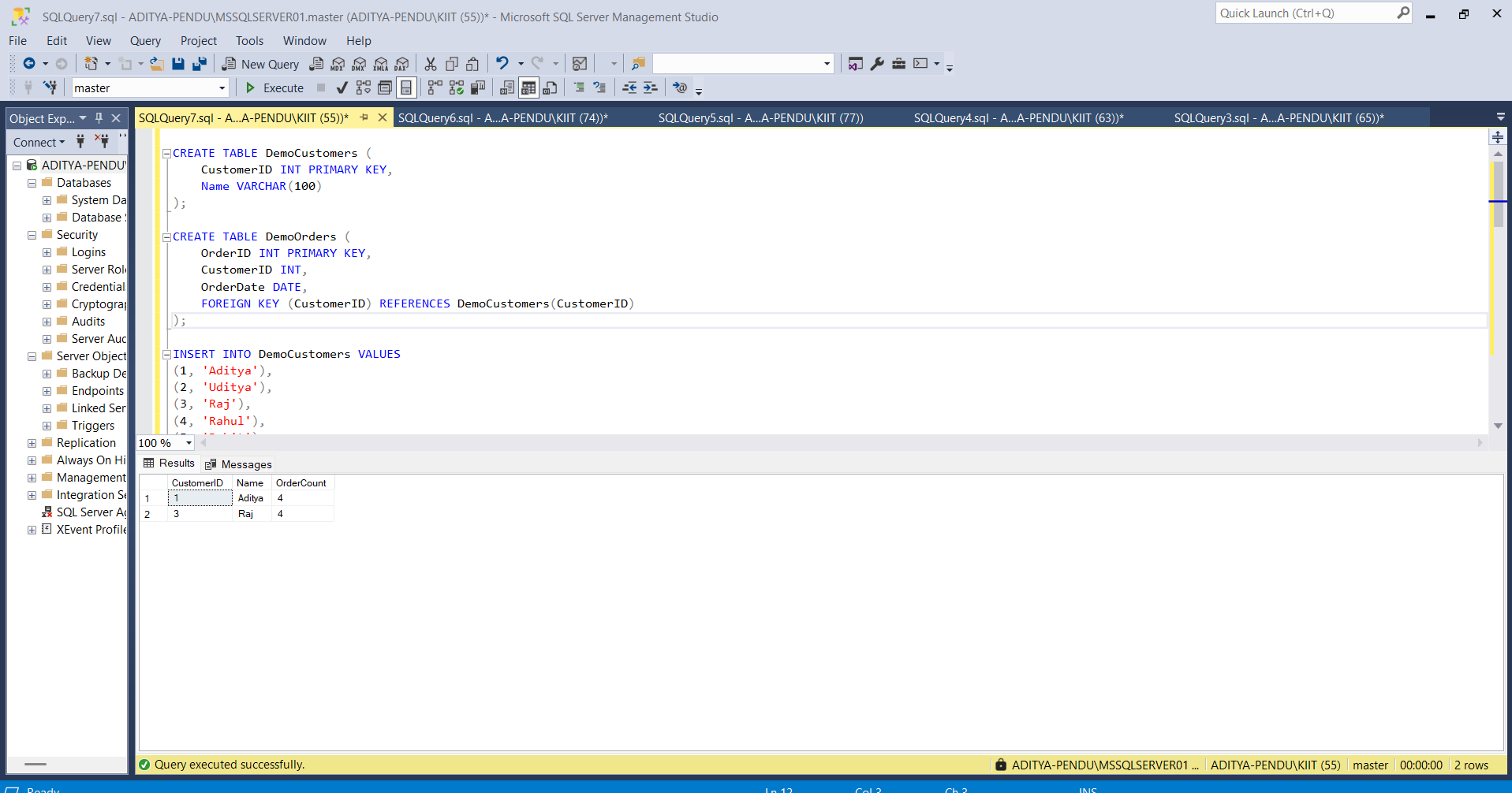
coc.OrderCount

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),

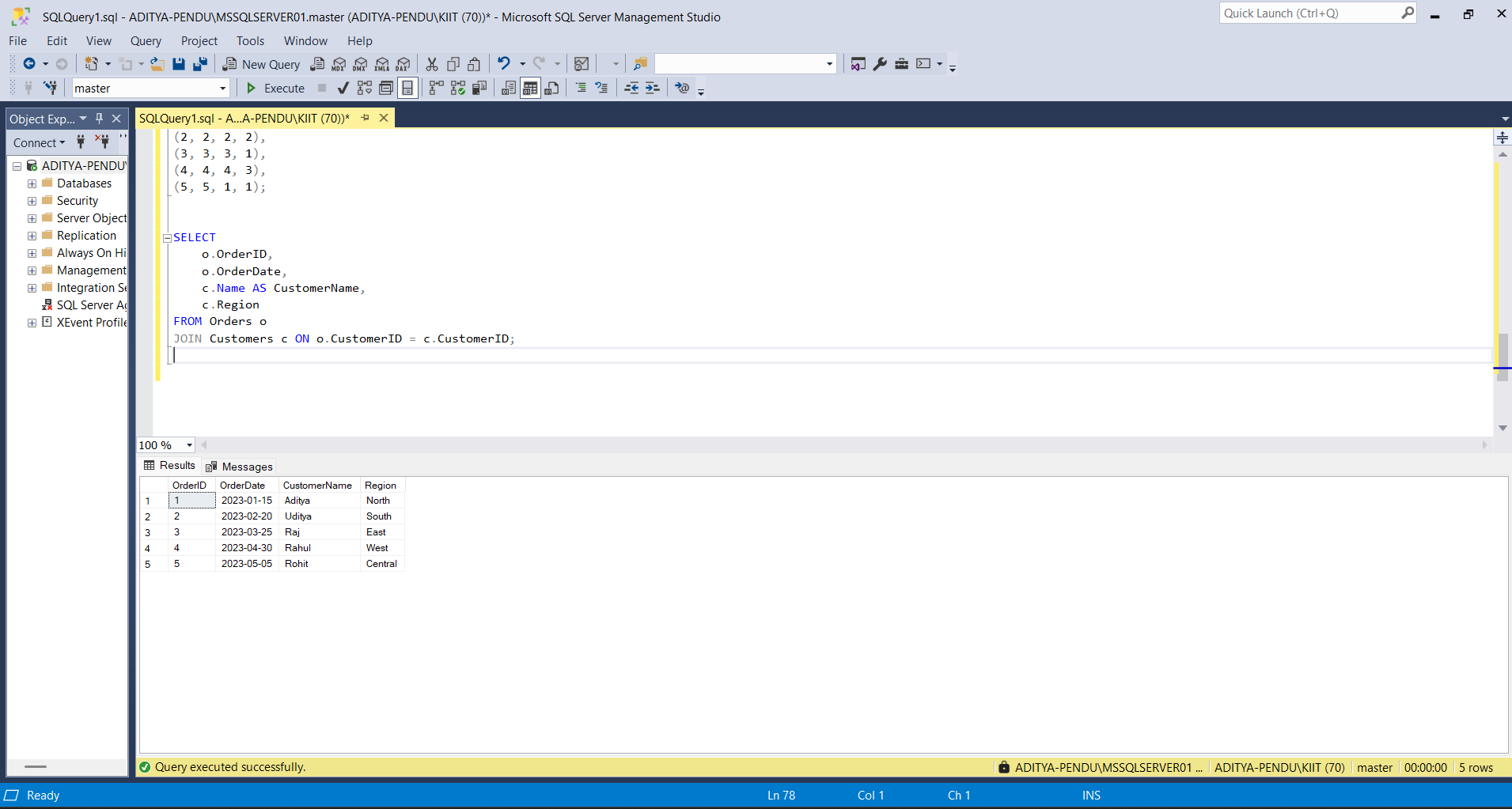
(2, 2, 2, 2),

(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

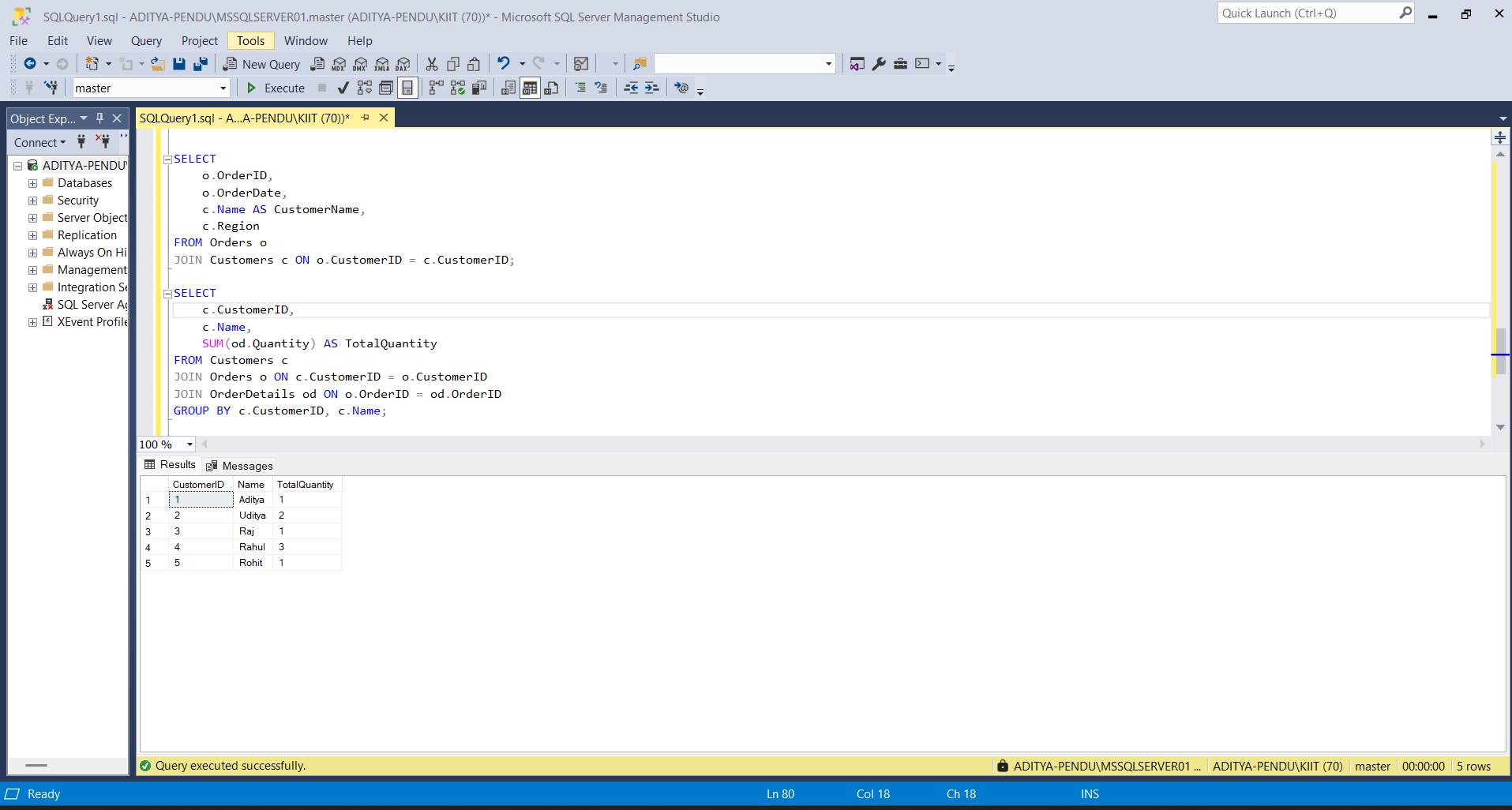
FROM Customers c

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

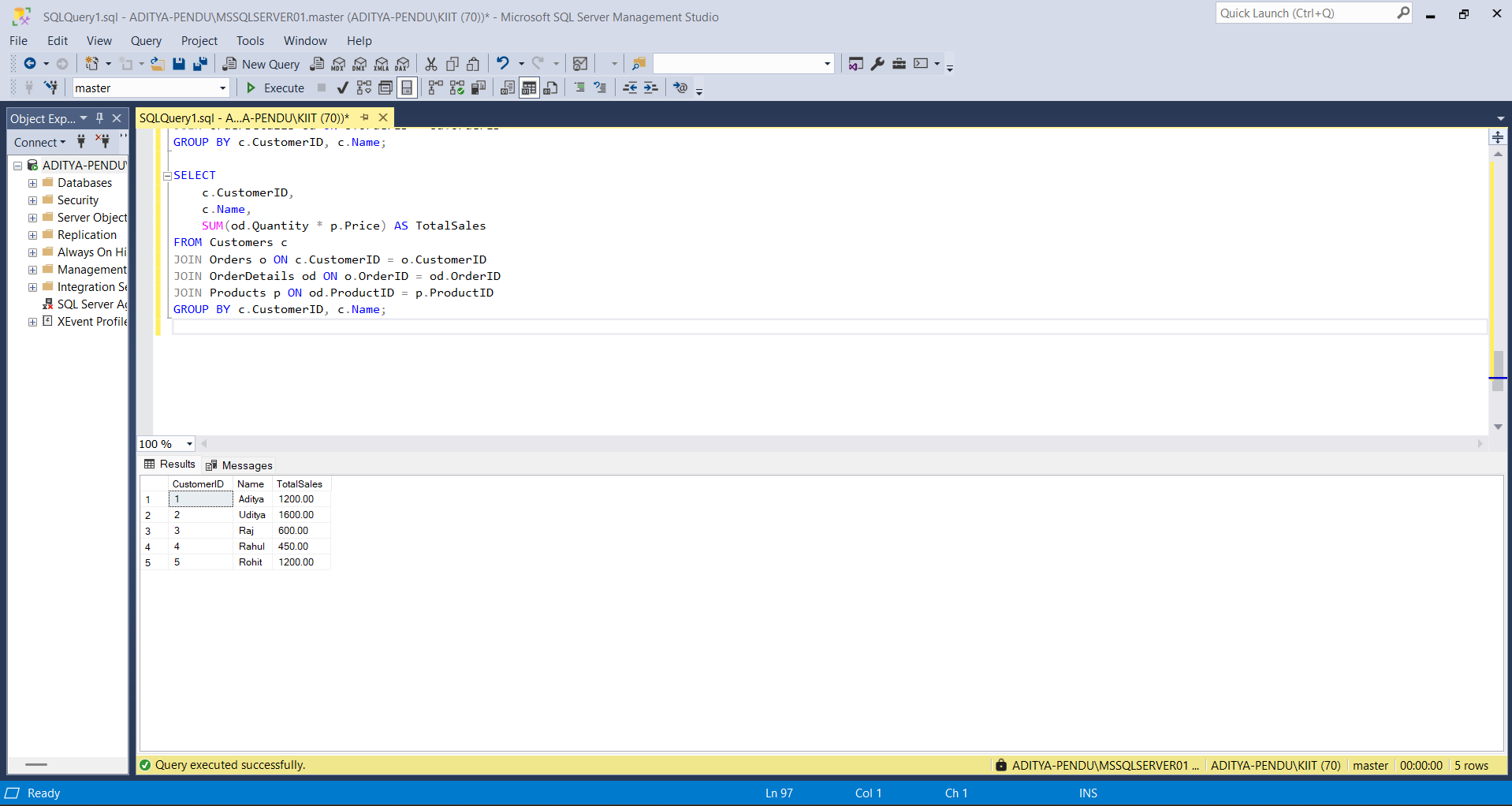
JOIN Orders o ON c.CustomerID = o.CustomerID

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,

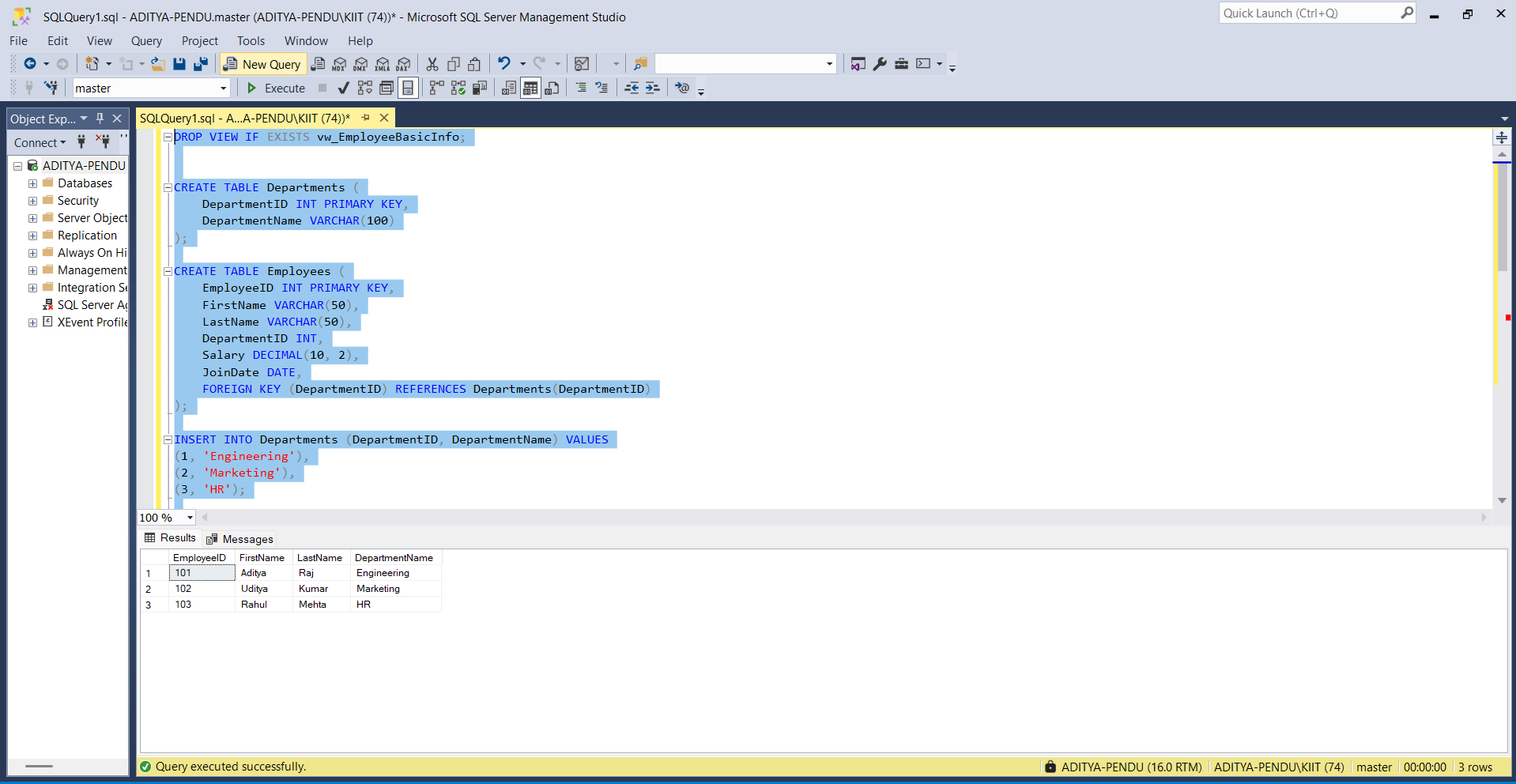
e.LastName,

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,

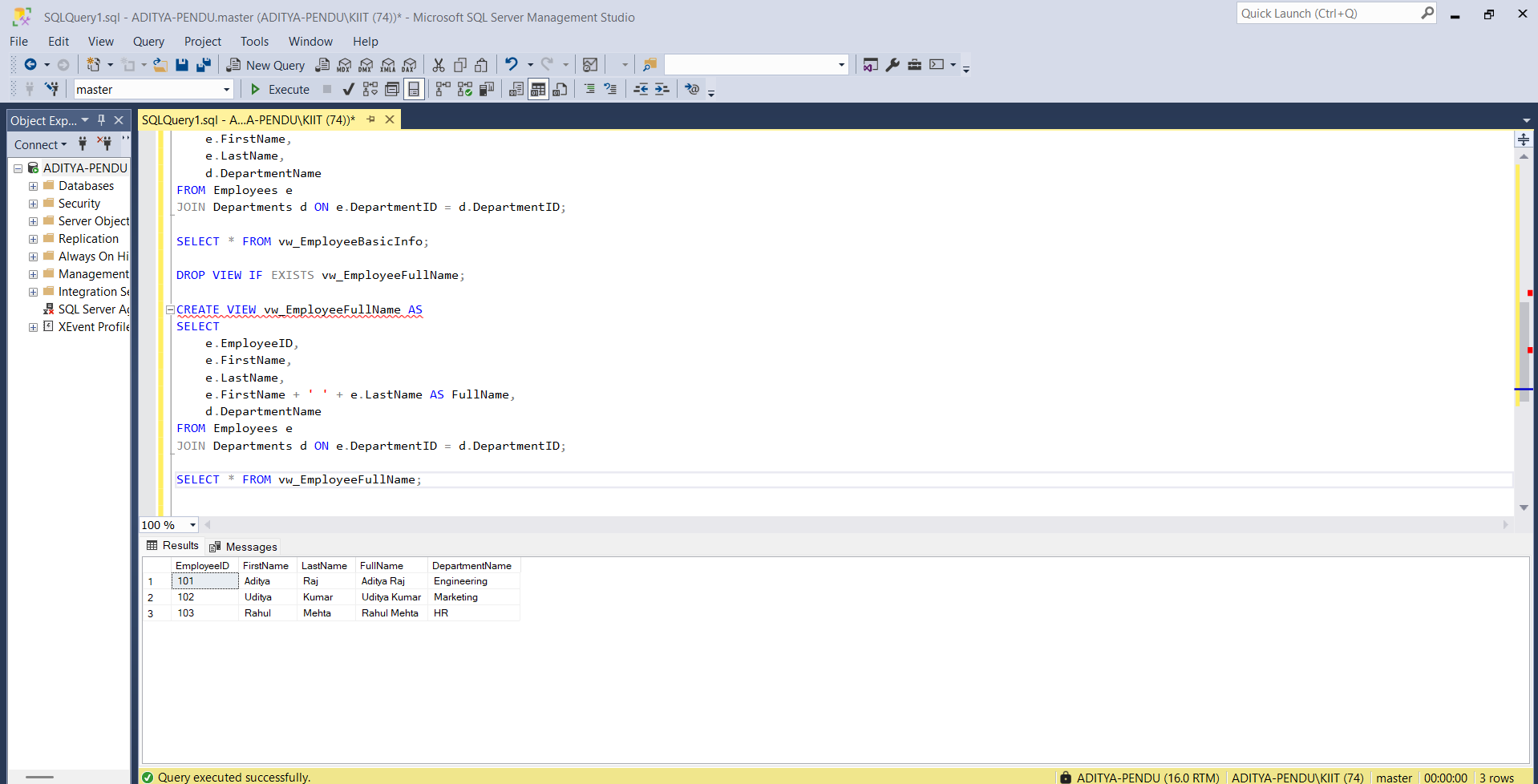
d.DepartmentName

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,

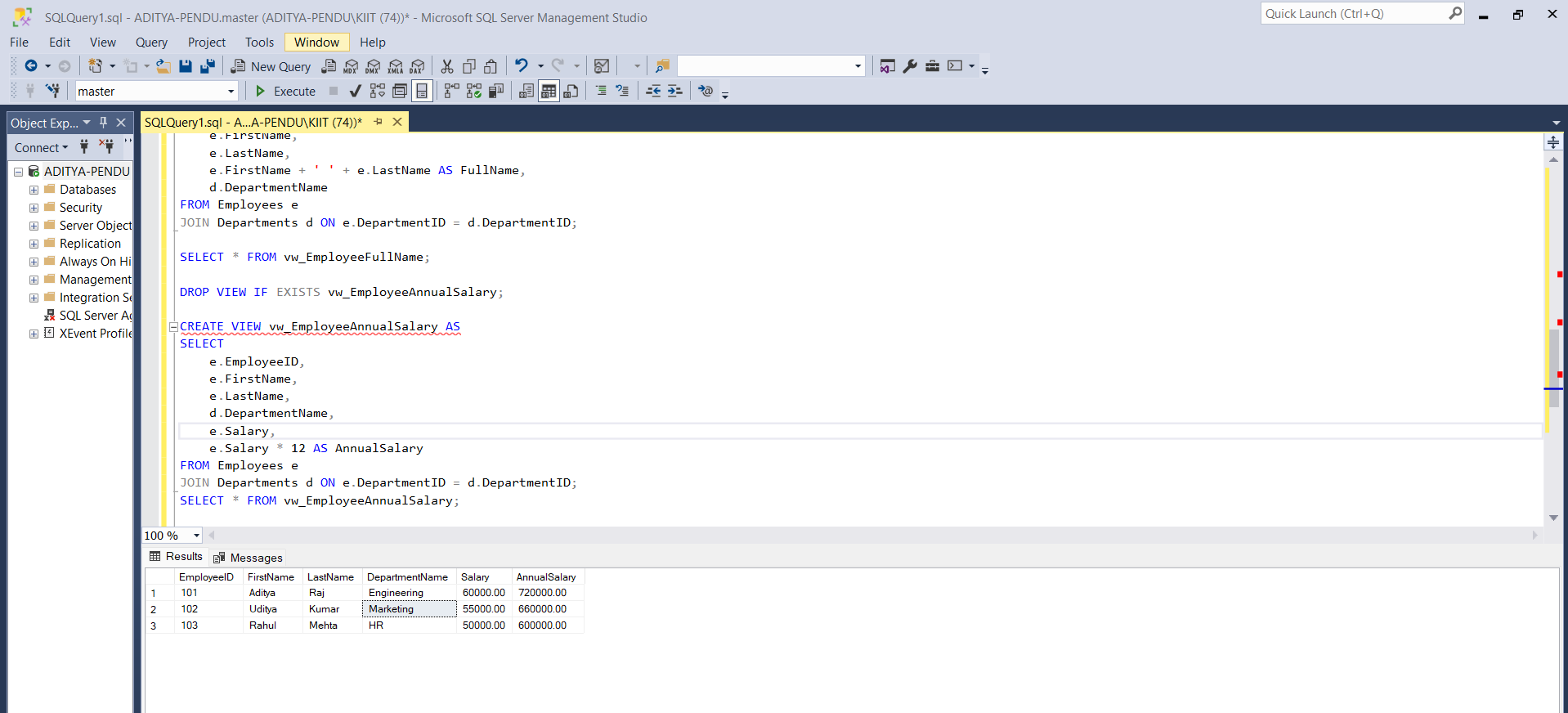
e.Salary \* 12 AS AnnualSalary

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,

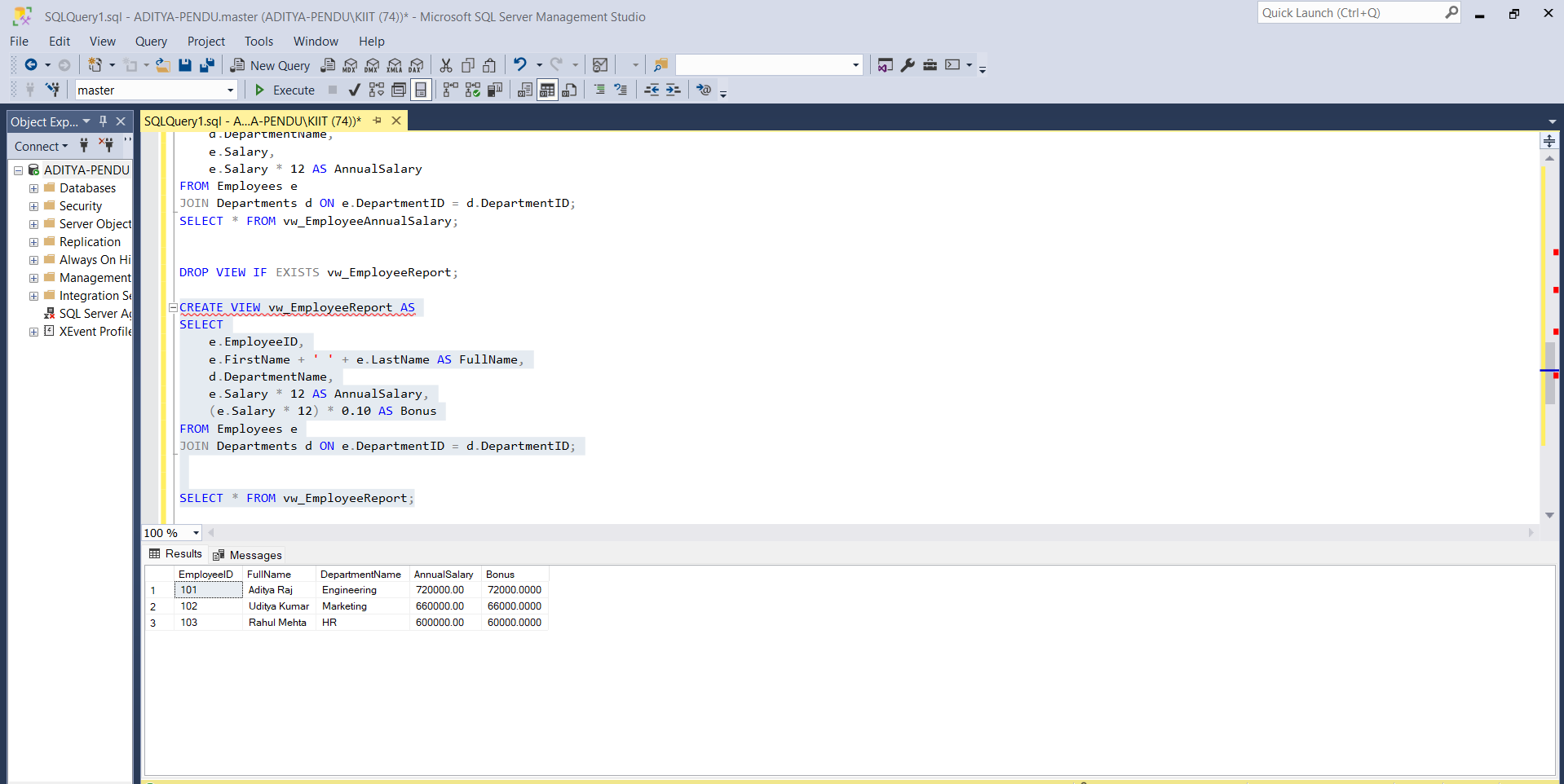
(e.Salary \* 12) \* 0.10 AS Bonus

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

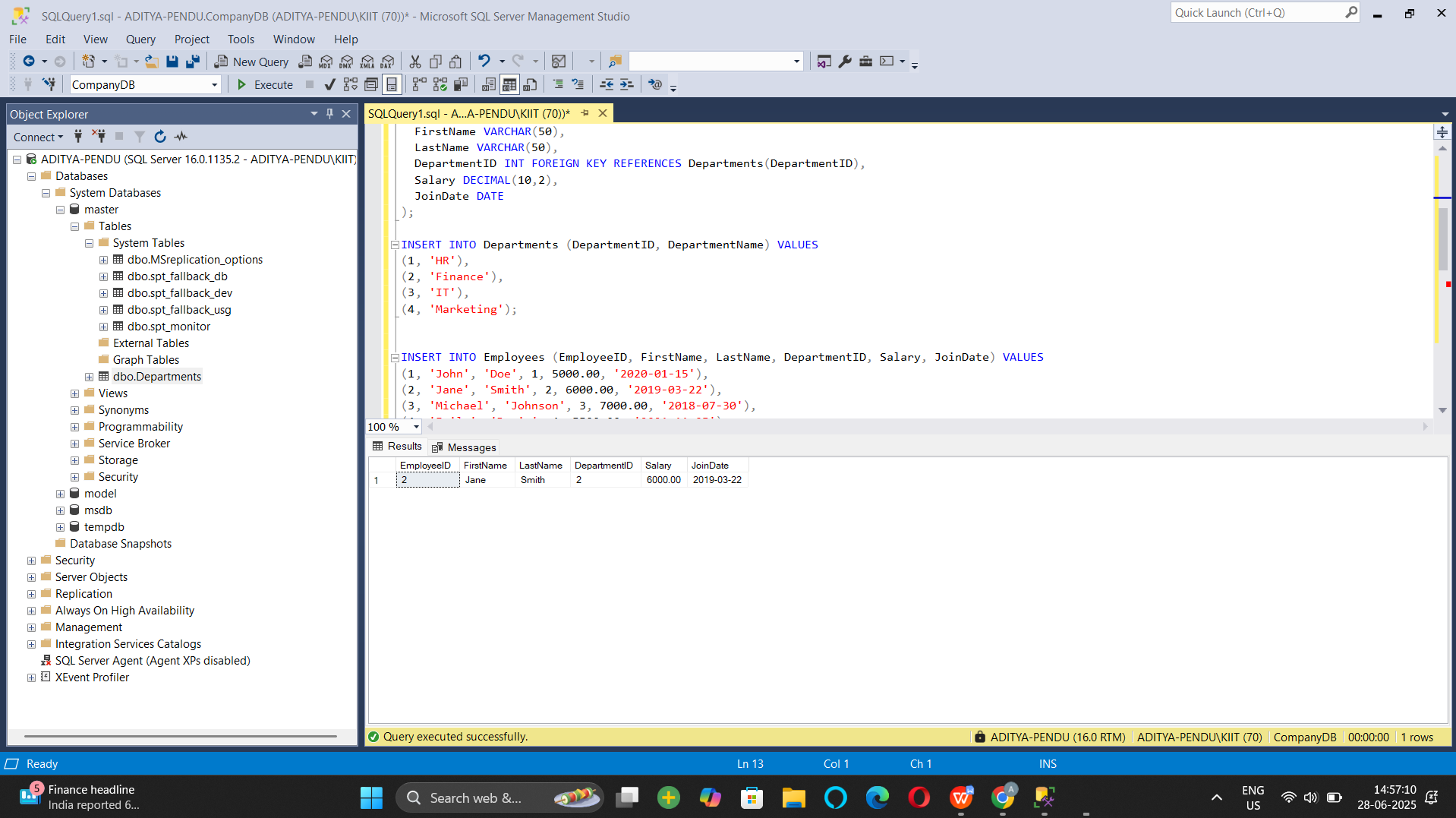
FROM Employees

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',

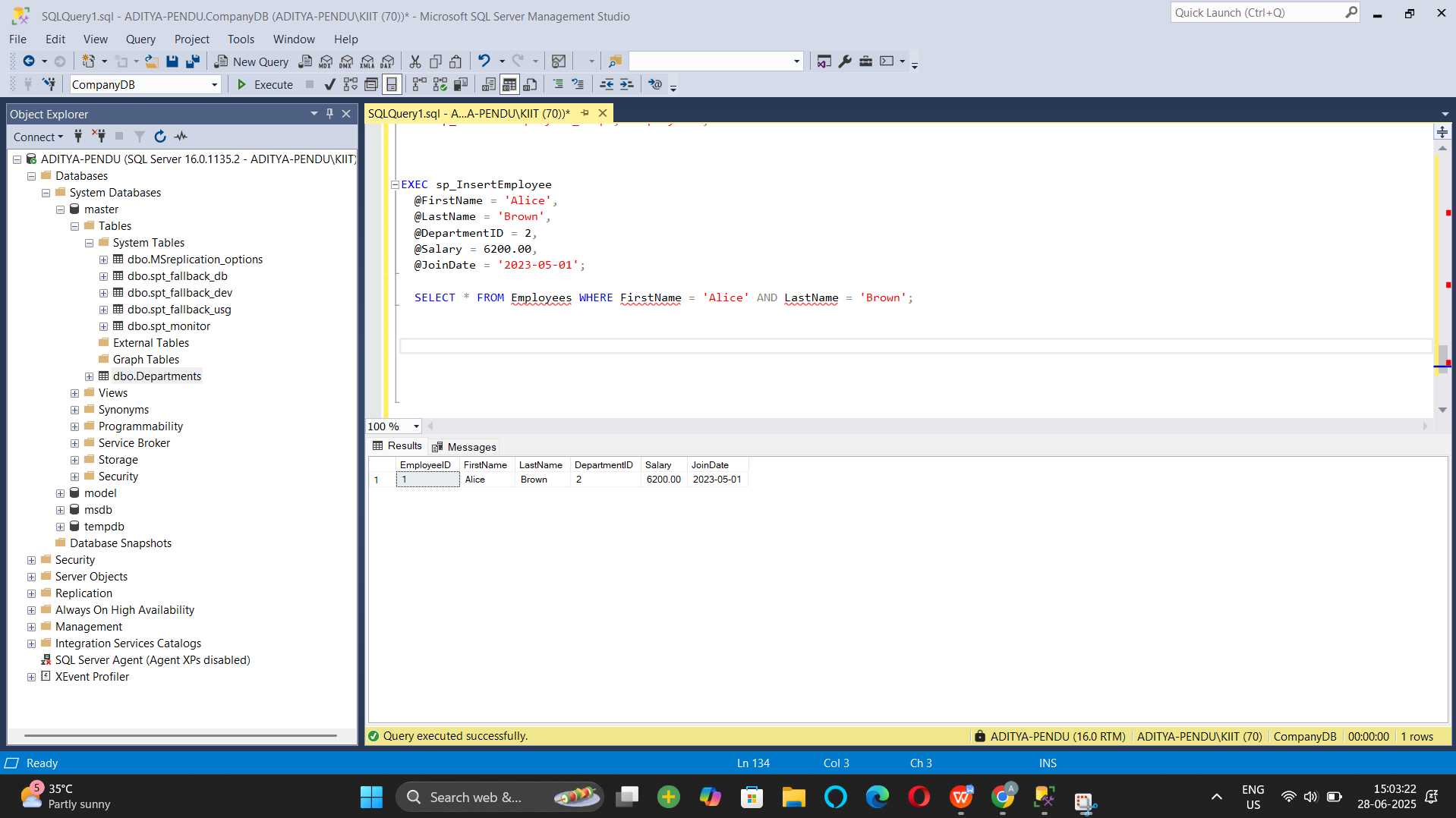
@DepartmentID = 2,

@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',

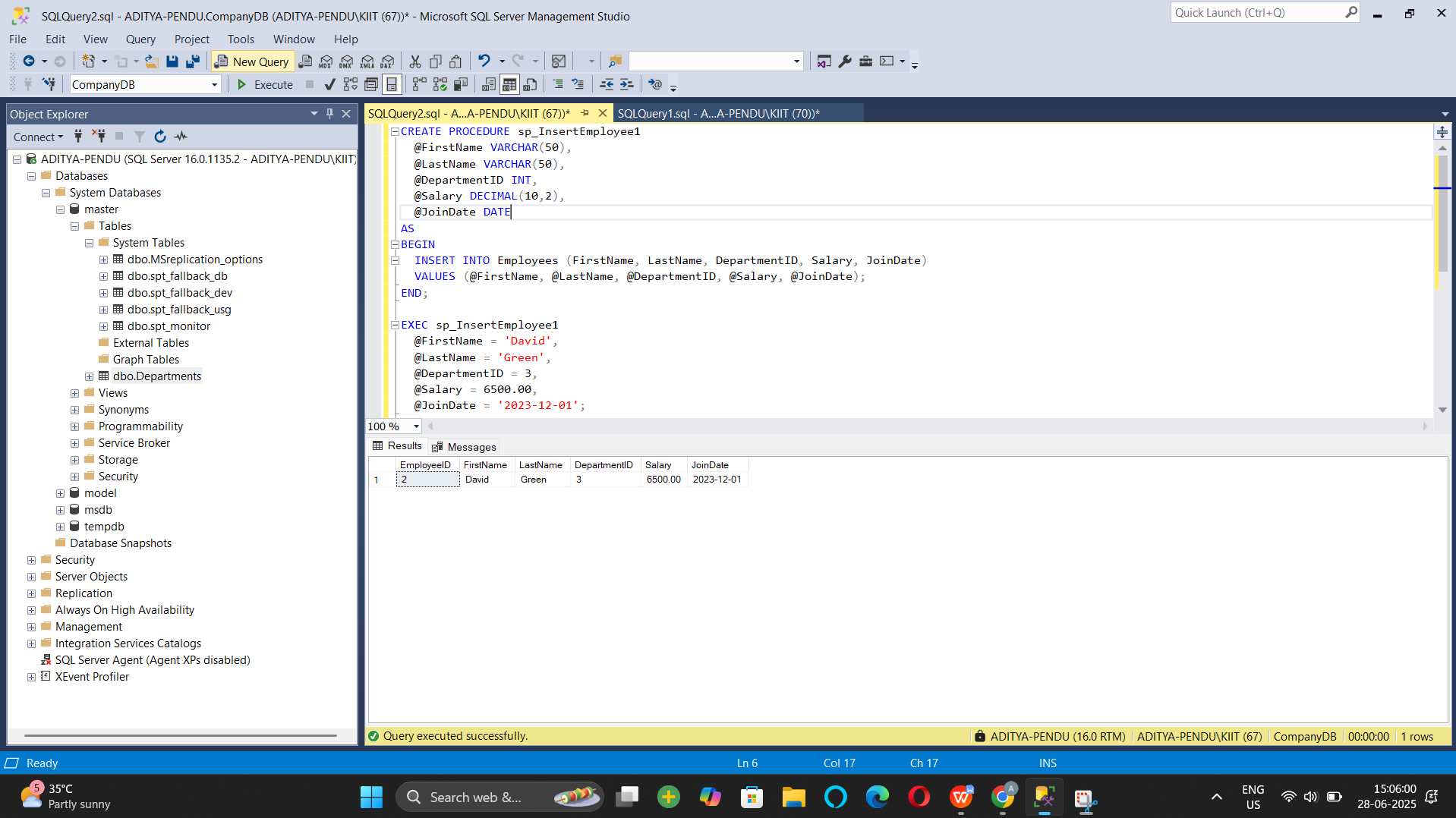
@DepartmentID = 3,

@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 :**



**Exercise 4: SQL EXERCISE - Functions :-**

Question 1: Create a Scalar Function

IF OBJECT\_ID('dbo.fn\_CalculateAnnualSalary', 'FN') IS NOT NULL DROP FUNCTION dbo.fn\_CalculateAnnualSalary;

IF OBJECT\_ID('dbo.fn\_GetEmployeesByDepartment', 'IF') IS NOT NULL DROP FUNCTION dbo.fn\_GetEmployeesByDepartment;

IF OBJECT\_ID('dbo.fn\_CalculateBonus', 'FN') IS NOT NULL DROP FUNCTION dbo.fn\_CalculateBonus;

IF OBJECT\_ID('Employees', 'U') IS NOT NULL DROP TABLE Employees;

IF OBJECT\_ID('Departments', 'U') IS NOT NULL DROP TABLE Departments;

GO

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

GO

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

);

GO

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'IT'),

(3, 'Finance');

GO

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, 'Bob', 'Johnson', 3, 5500.00, '2021-07-01');

GO

CREATE FUNCTION fn\_CalculateAnnualSalary (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 12;

END;

GO

SELECT

EmployeeID,

FirstName,

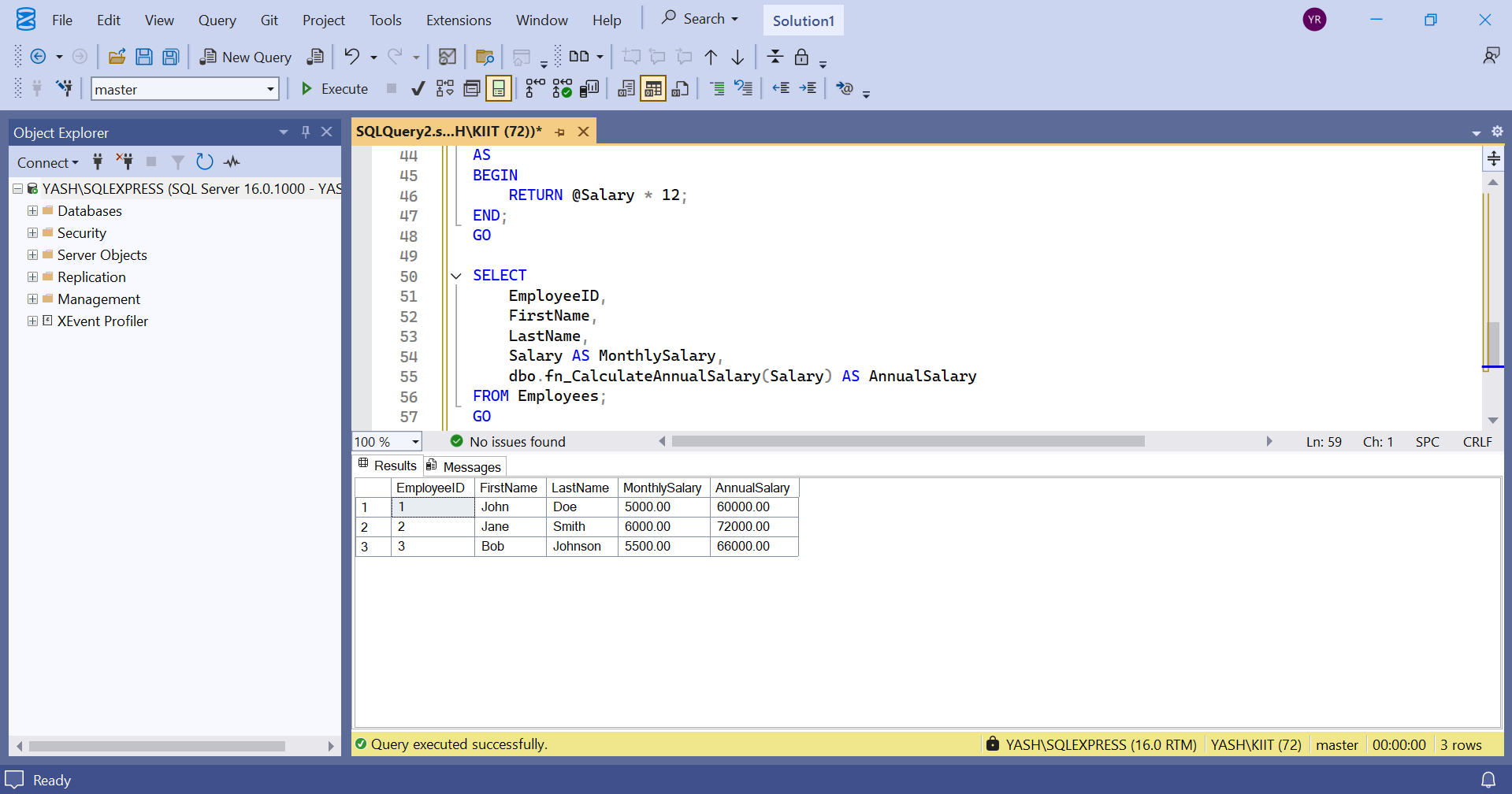
LastName,

Salary AS MonthlySalary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees;

GO



1. Create a Table-Valued Function:-

CREATE FUNCTION fn\_GetEmployeesByDepartment (@DeptID INT)

RETURNS TABLE

AS

RETURN

SELECT

EmployeeID,

FirstName,

LastName,

DepartmentID,

Salary,

JoinDate

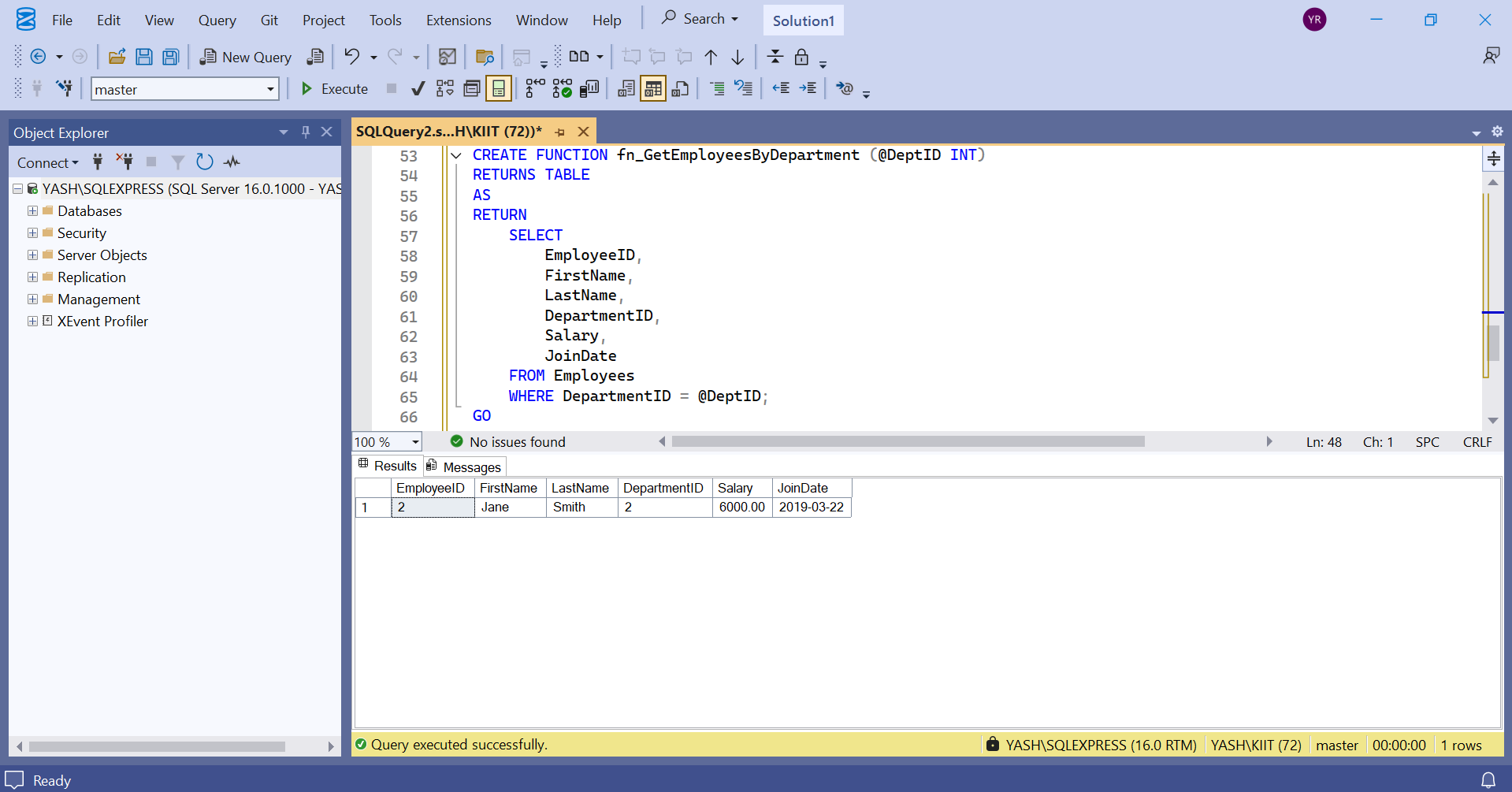
FROM Employees

WHERE DepartmentID = @DeptID;

GO

SELECT \* FROM fn\_GetEmployeesByDepartment(2);

GO



7: Return Data from a Scalar Function:-

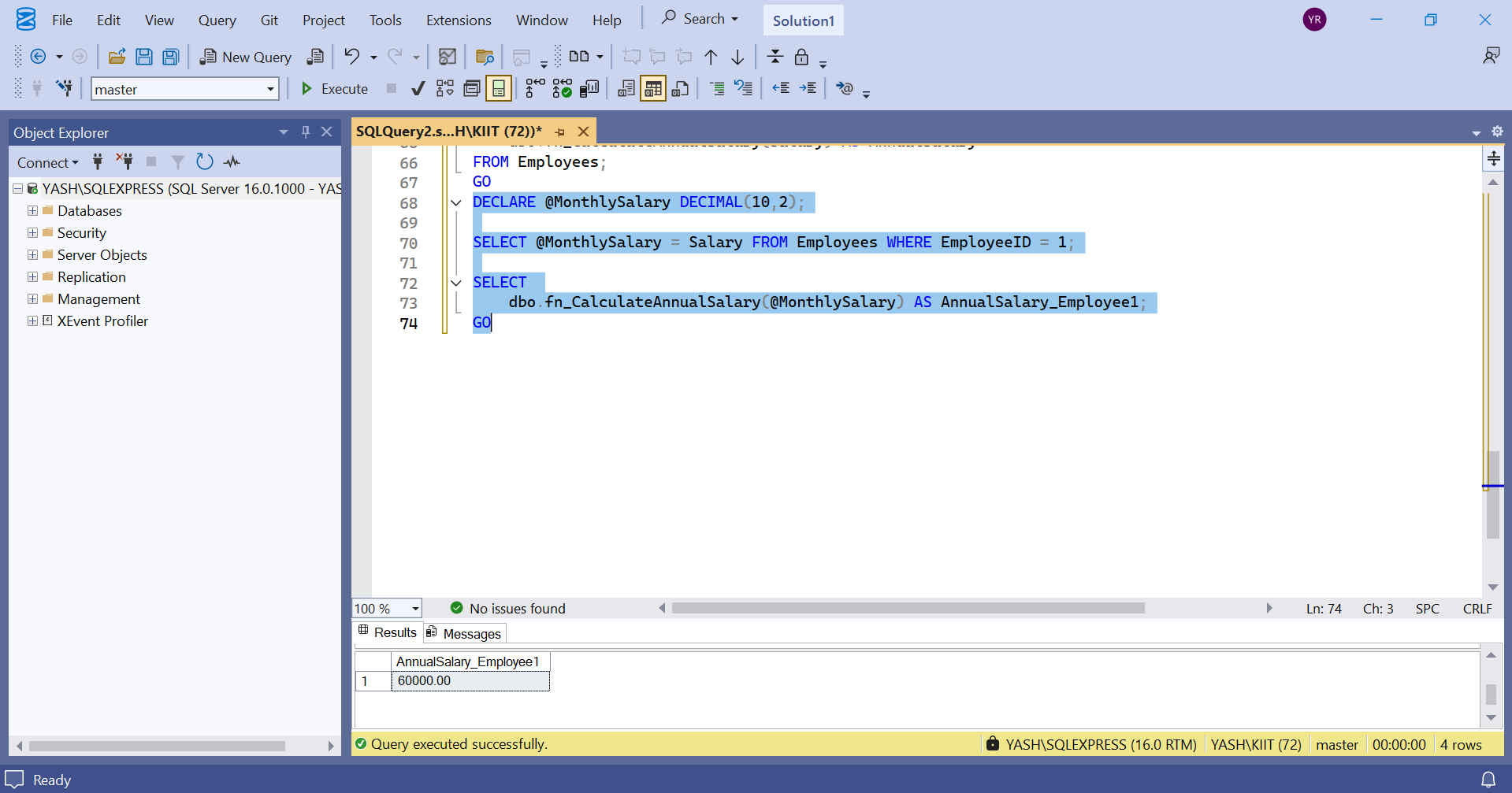
DECLARE @MonthlySalary DECIMAL(10,2);

SELECT @MonthlySalary = Salary FROM Employees WHERE EmployeeID = 1;

SELECT

dbo.fn\_CalculateAnnualSalary(@MonthlySalary) AS AnnualSalary\_Employee1;

GO



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; using CalcLibrary;

namespace Nunit

{

[TestFixture]

public class CalculatorTests

{

private Calculator calc;

[SetUp]

public void SetUp()

{

calc = new Calculator(); }

[TearDown]

public void TearDown()

{

}

[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()

{

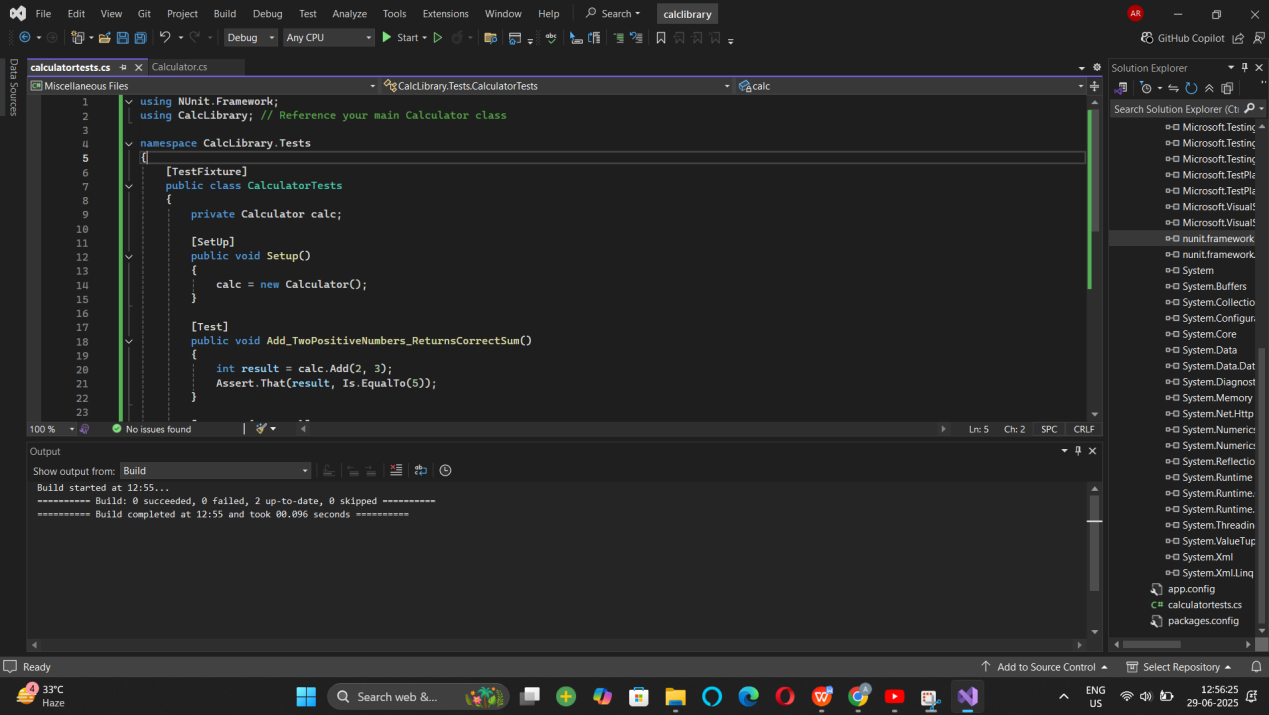
Assert.Fail("This is just a placeholder test.");

}

}

}

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()

{

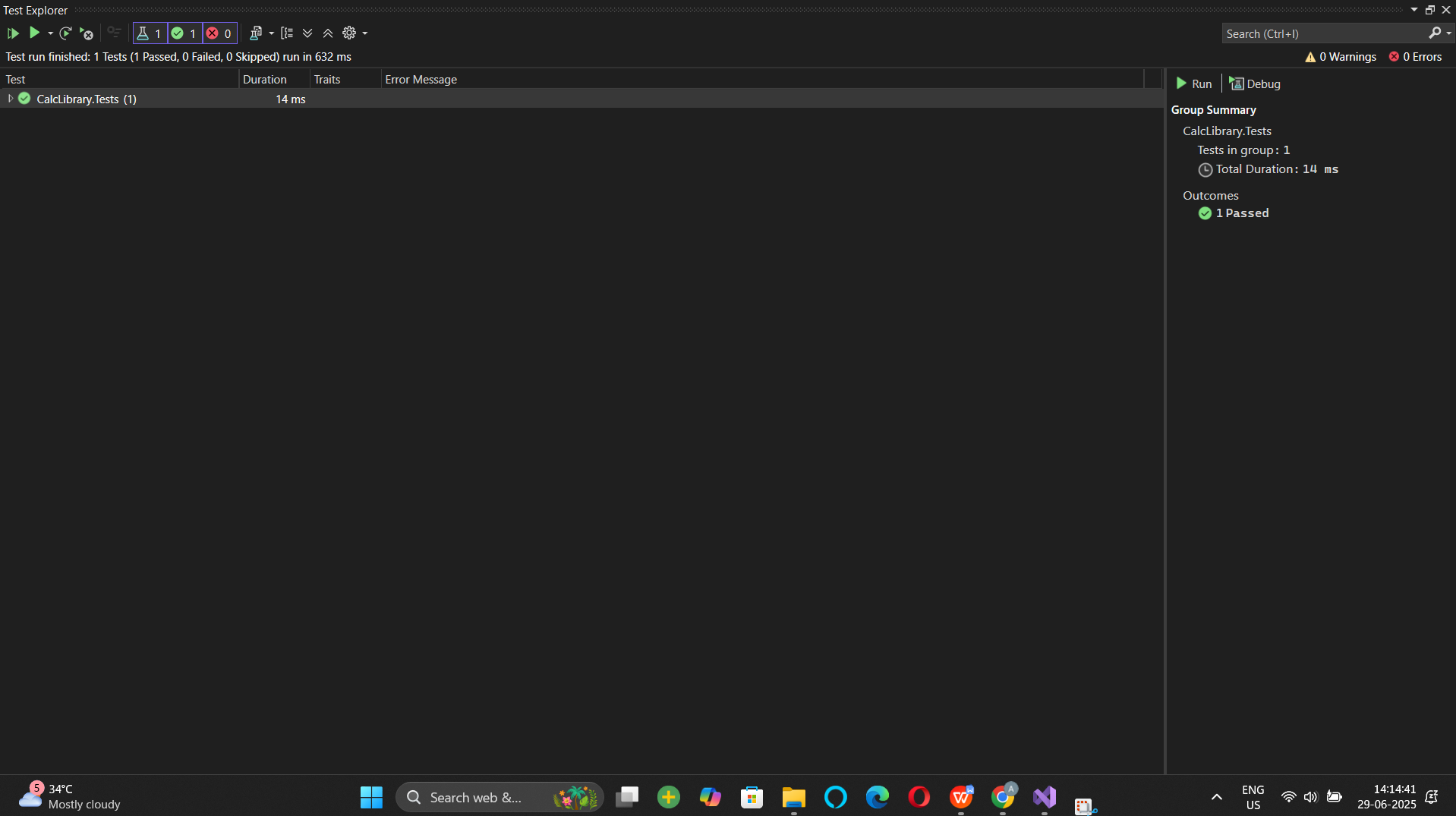
result = 0;

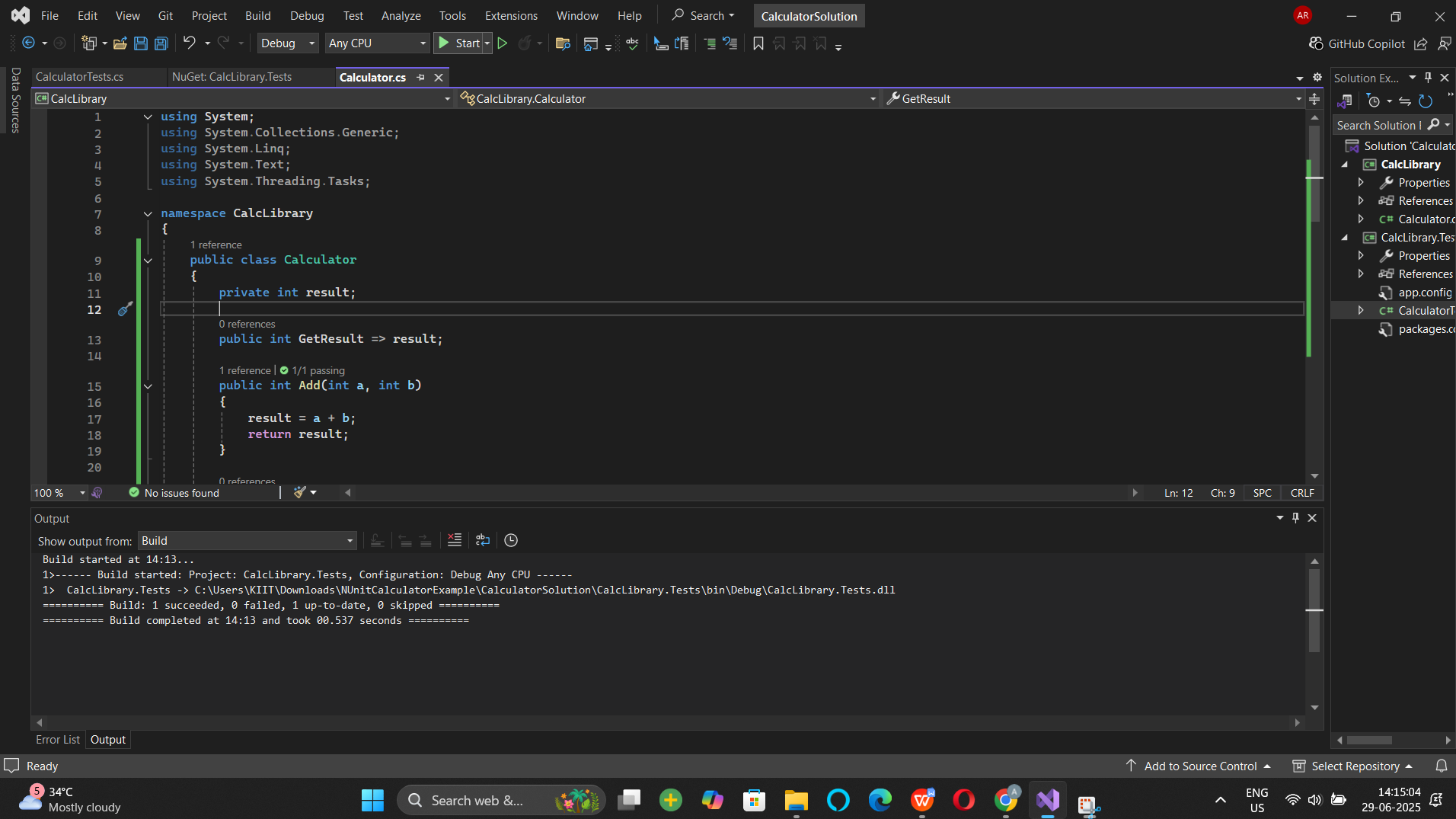
}

}

}

Output :





Moq handson:

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 Moq;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CustomerCommLib.Tests

{

public class CustomerCommTests

{

public void SendMailToCustomer\_ShouldReturnTrue\_WhenMailIsSentSuccessfully()

{

var mockMailSender = new Mock<IMailSender>();

mockMailSender.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>())).Returns(true);

var customerComm = new CustomerComm(mockMailSender.Object);

bool result = customerComm.SendMailToCustomer();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CustomerCommLib

{

public class CustomerComm

{

IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

string to = "cust123@abc.com";

string message = "Some Message";

return \_mailSender.SendMail(to, message);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CustomerCommLib

{

public interface IMailSender

{

bool SendMail(string toAddress, string message);

}

}

