**Advanced SQL Exercises for Online Retail Store**

**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.

**Solution :**

**Objective :** The goal of this exercise is to use ROW\_NUMBER(), RANK(), and DENSE\_RANK() functions along with PARTITION BY and ORDER BY clauses to find the top 3 most expensive products in each category**.**

**Database and Table Creation Code :**

CREATE DATABASE RetailStore;

GO

USE RetailStore;

GO

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(100),

Price DECIMAL(10,2)

);

**Data Insertion Code :**

INSERT INTO Products VALUES

(1, 'Laptop', 'Electronics', 80000),

(2, 'Smartphone', 'Electronics', 60000),

(3, 'Tablet', 'Electronics', 30000),

(4, 'Camera', 'Electronics', 60000),

(5, 'Shirt', 'Clothing', 2000),

(6, 'Jeans', 'Clothing', 2500),

(7, 'Jacket', 'Clothing', 5000),

(8, 'Shoes', 'Clothing', 2500),

(9, 'Blender', 'Home Appliance', 4000),

(10, 'Microwave', 'Home Appliance', 7000),

(11, 'Toaster', 'Home Appliance', 4000),

(12, 'Fridge', 'Home Appliance', 12000);

**Queries and Results**

**1. ROW\_NUMBER() Query**

SELECT

Category,

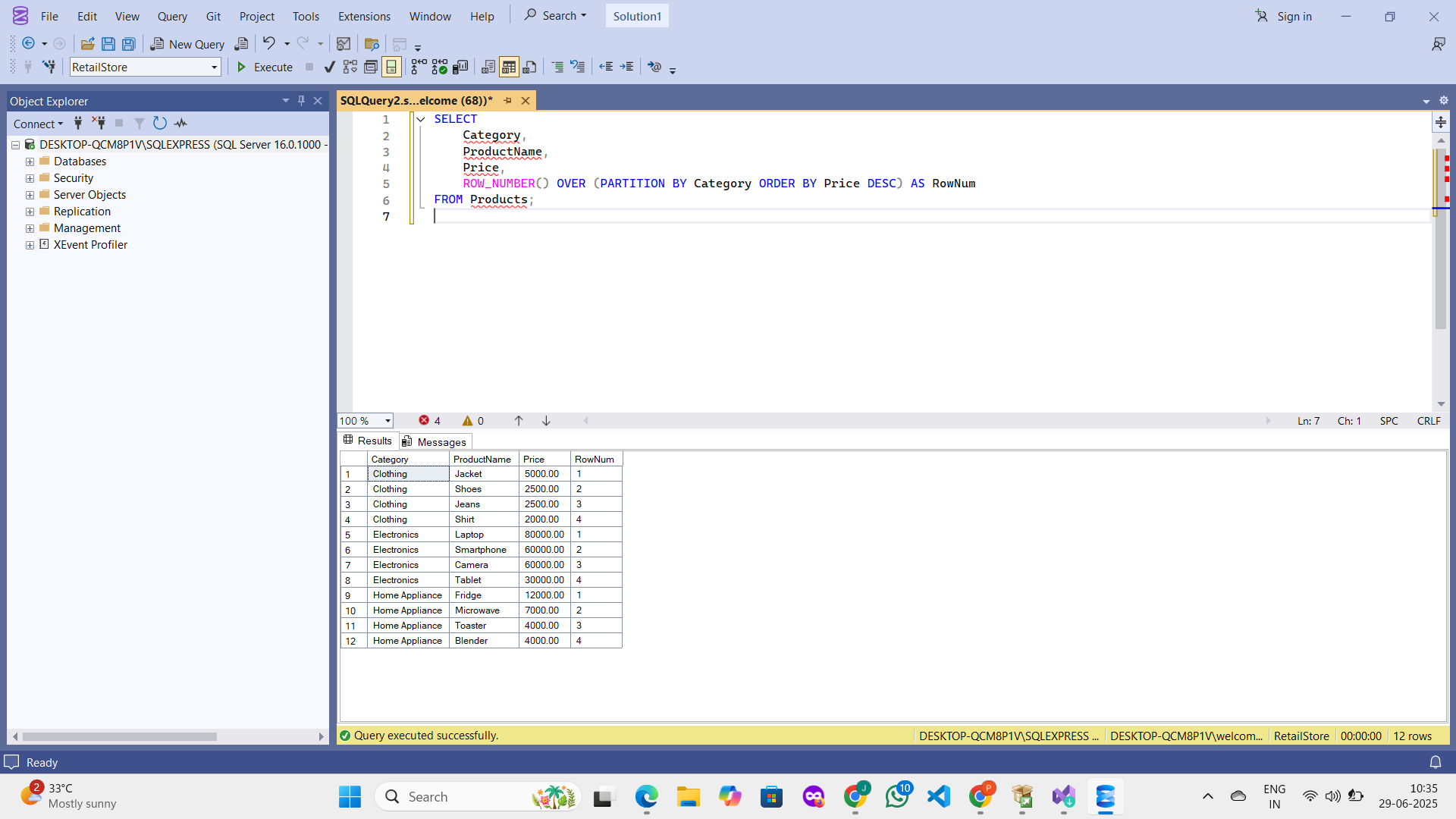
ProductName,

Price,

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

FROM Products;

**OUTPUT :**

****

**2. RANK() Query**

SELECT

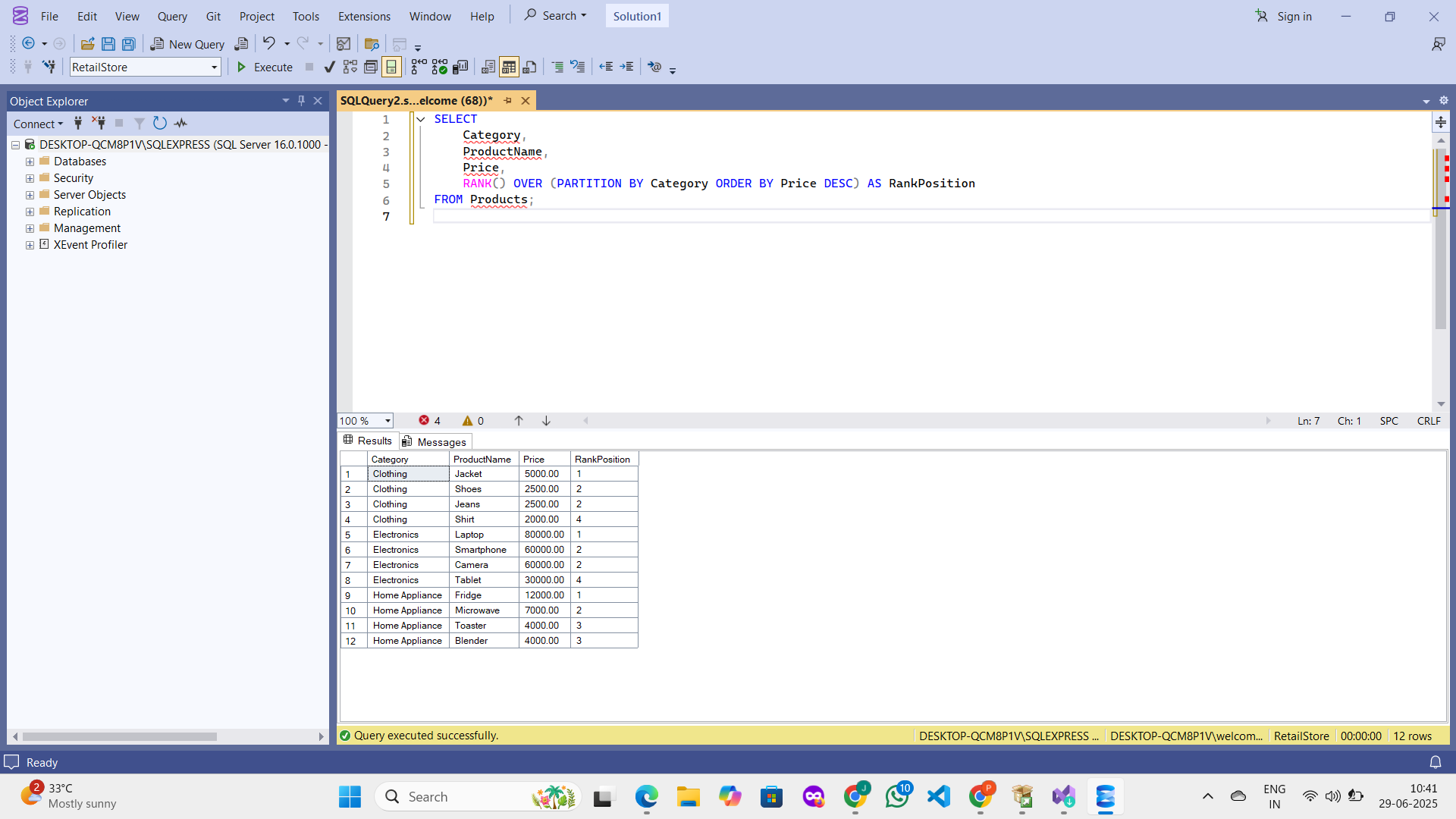
Category,

ProductName,

Price,

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

FROM Products;  
**OUTPUT :**



**🔹 3. DENSE\_RANK() Query**

SELECT

Category,

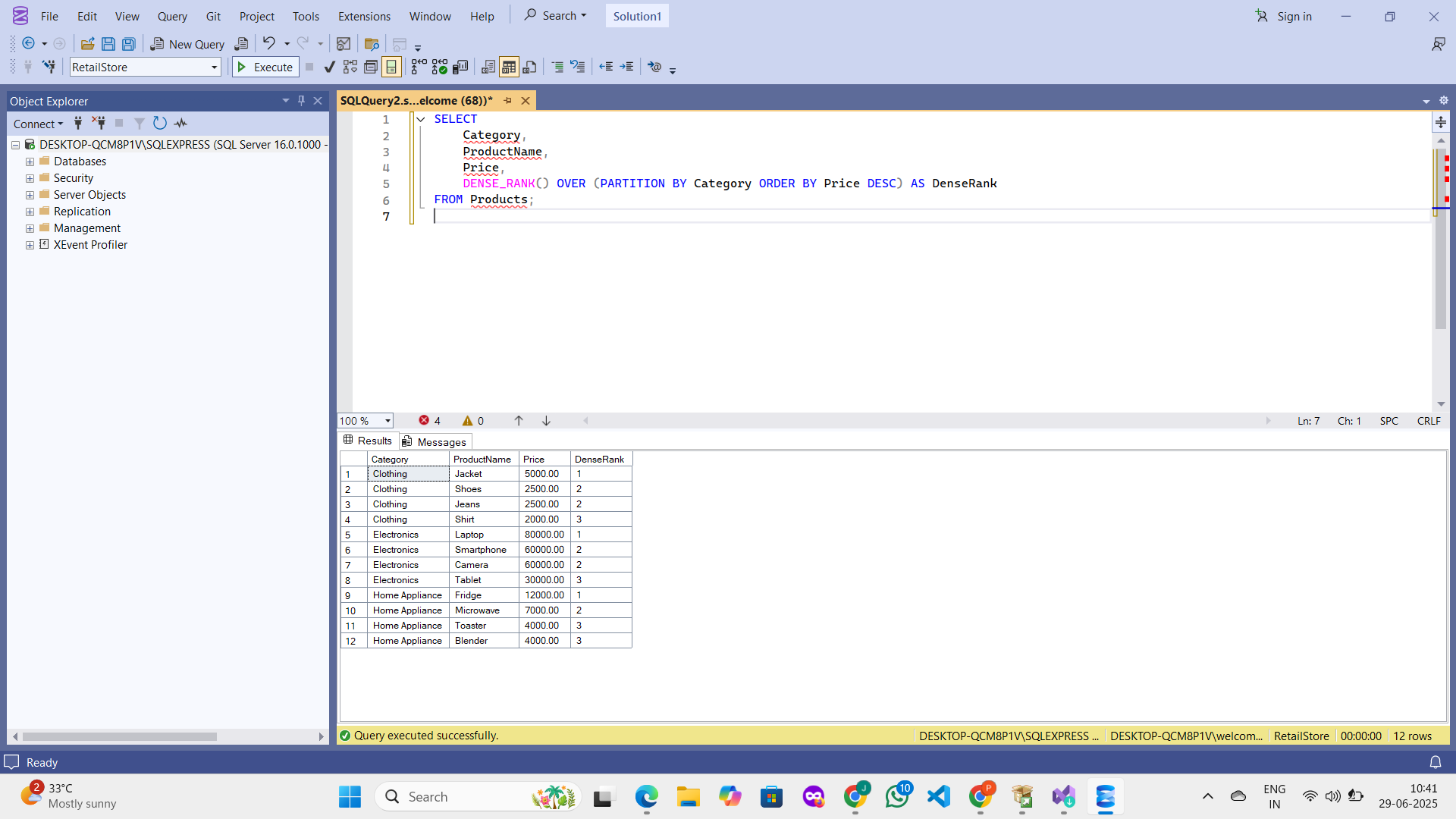
ProductName,

Price,

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

FROM Products;

**OUTPUT :**

****

**Final Query – Top 3 Products per Category**

SELECT \*

FROM (

SELECT

Category,

ProductName,

Price,

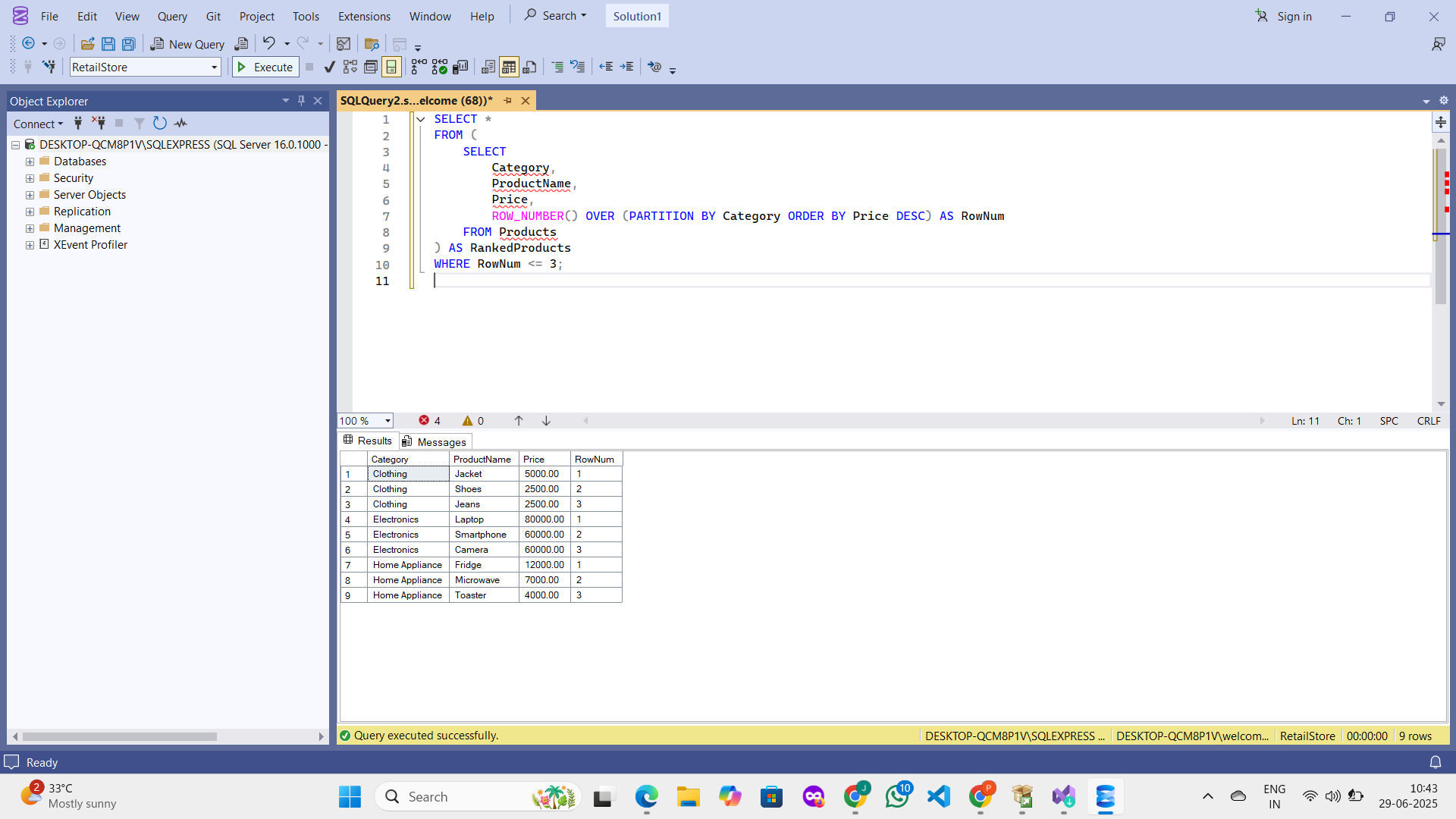
ROW\_NUMBER() OVER (PARTITION BY Category ORDER BY Price DESC) AS RowNum

FROM Products

) AS RankedProducts

WHERE RowNum <= 3;

**OUTPUT :**

****

**Exercise 2: Aggregation with GROUPING SETS, CUBE, and ROLLUP Goal: Analyze sales data across multiple dimensions.**

**Scenario: Generate a report showing total quantity sold by Region and Category using GROUPING SETS, ROLLUP, and Steps.**

**SOLUTION :**

**Objective :**

The goal of this exercise is to use SQL Server’s powerful aggregation techniques (GROUPING SETS, ROLLUP, and CUBE) to analyze sales data across multiple dimensions — specifically Region and Category.

**Table Creation Scripts :**

USE RetailStore;

GO

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(100),

Region VARCHAR(100)

);

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 Queries**

-- Insert Customers

INSERT INTO Customers VALUES

(1, 'John', 'North'),

(2, 'Asha', 'South'),

(3, 'Li', 'East'),

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

-- Insert Orders

INSERT INTO Orders VALUES

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

(102, 2, '2024-02-15'),

(103, 3, '2024-03-20'),

(104, 4, '2024-04-10');

-- Insert OrderDetails

INSERT INTO OrderDetails VALUES

(1, 101, 1, 2),

(2, 101, 5, 3),

(3, 102, 2, 1),

(4, 103, 7, 4),

(5, 104, 10, 2);

**Join Query for Verification**

SELECT

C.Region,

P.Category,

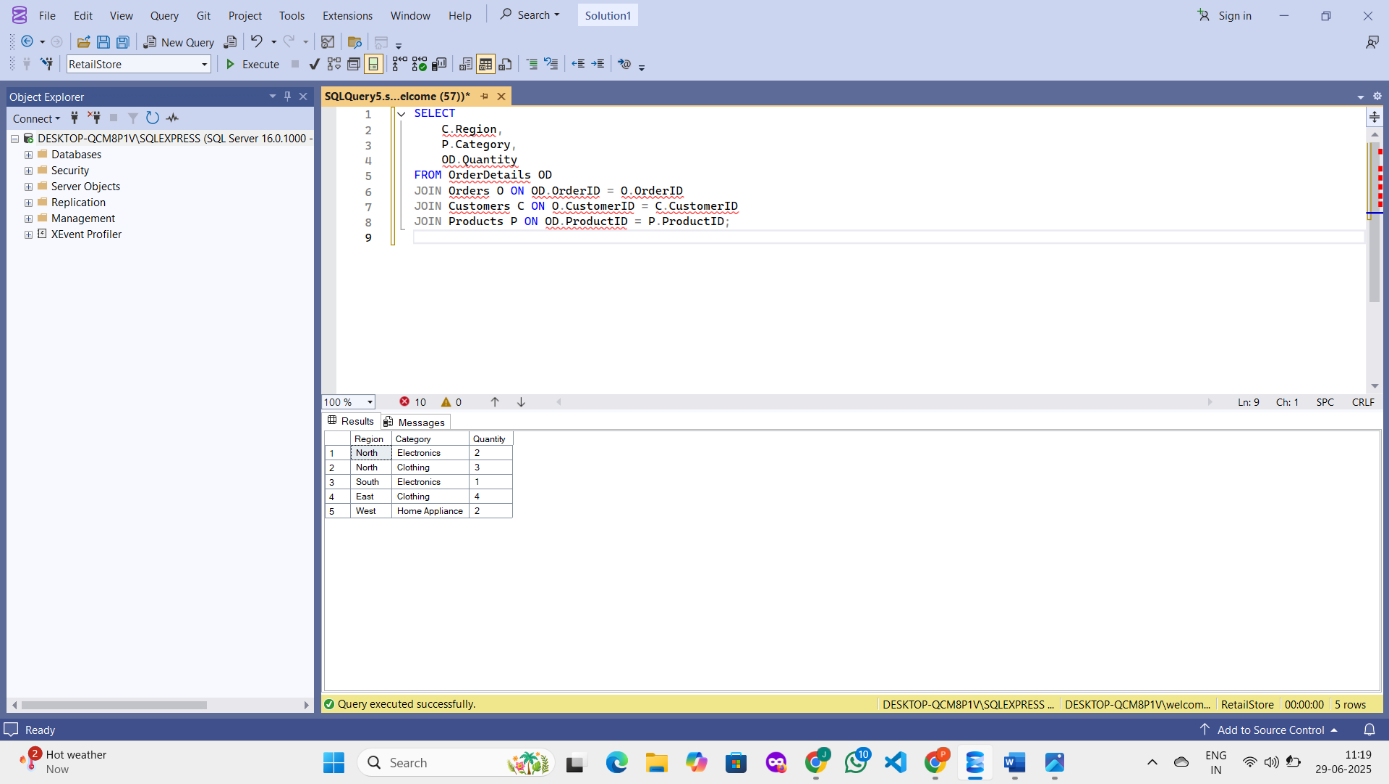
OD.Quantity

FROM OrderDetails OD

JOIN Orders O ON OD.OrderID = O.OrderID

JOIN Customers C ON O.CustomerID = C.CustomerID

JOIN Products P ON OD.ProductID = P.ProductID;

**OUTPUT :  
**

**Aggregation Queries & Results**

**GROUPING SETS**

SELECT

C.Region,

P.Category,

SUM(OD.Quantity) AS TotalQuantity

FROM OrderDetails OD

JOIN Orders O ON OD.OrderID = O.OrderID

JOIN Customers C ON O.CustomerID = C.CustomerID

JOIN Products P ON OD.ProductID = P.ProductID

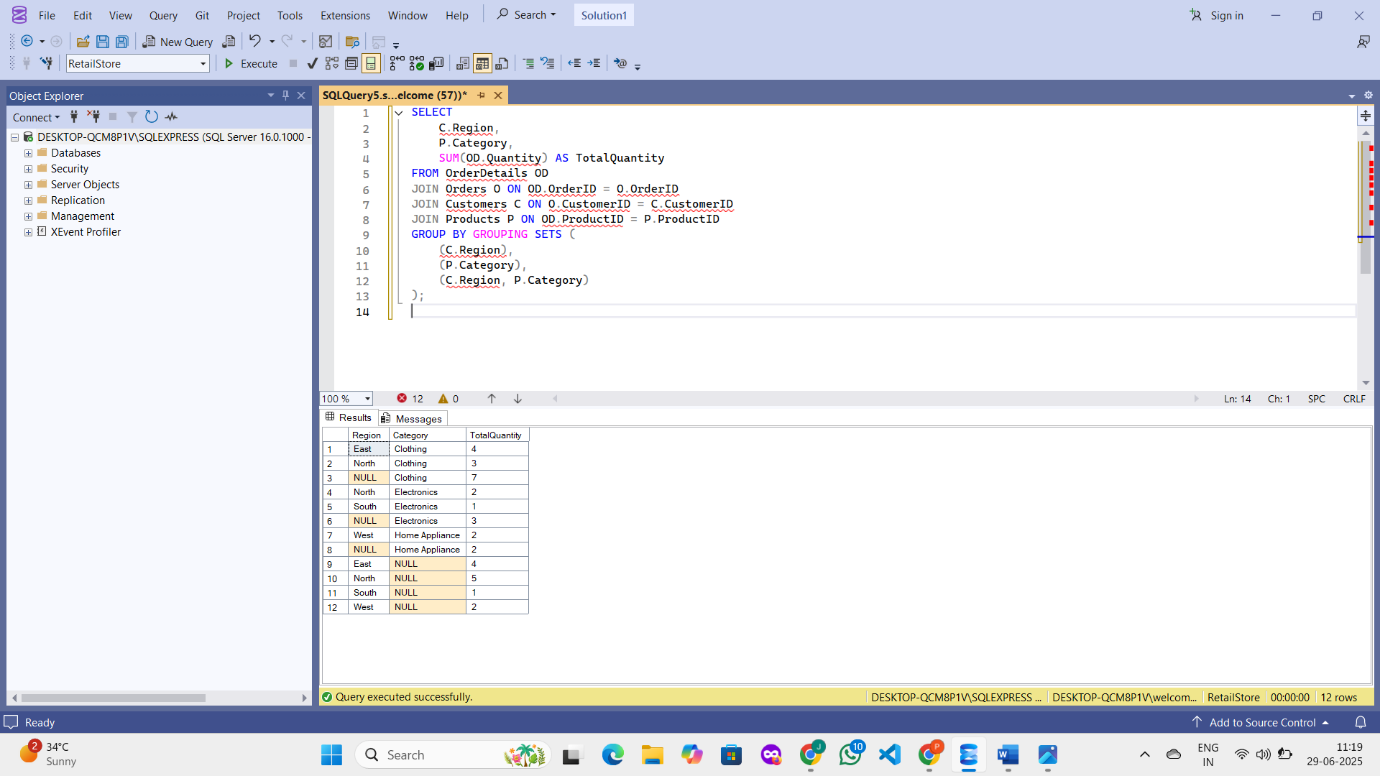
GROUP BY GROUPING SETS (

(C.Region),

(P.Category),

(C.Region, P.Category)

);

**OUTPUT :**

**ROLLUP**

SELECT

C.Region,

P.Category,

SUM(OD.Quantity) AS TotalQuantity

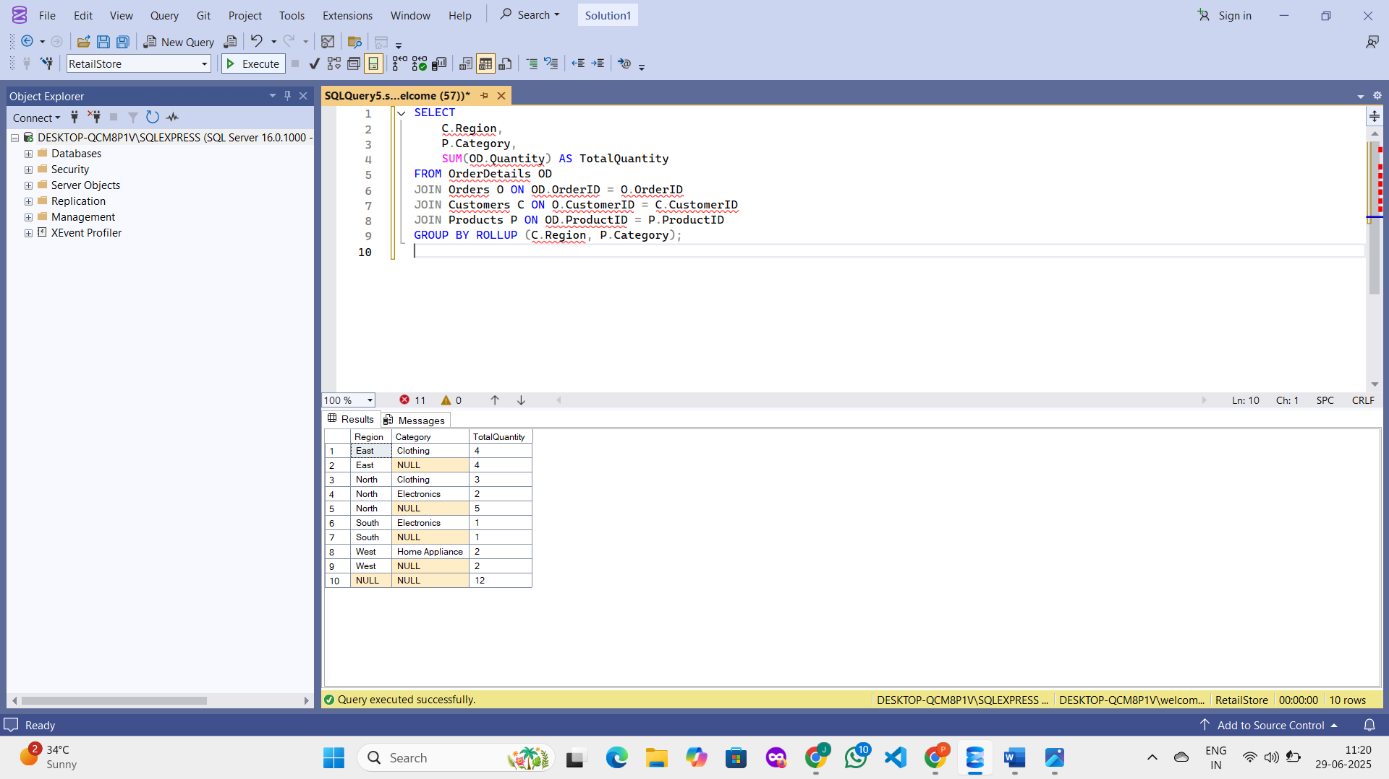
FROM OrderDetails OD

JOIN Orders O ON OD.OrderID = O.OrderID

JOIN Customers C ON O.CustomerID = C.CustomerID

JOIN Products P ON OD.ProductID = P.ProductID

GROUP BY ROLLUP (C.Region, P.Category);

**OUTPUT :  
**

**CUBE**

SELECT

C.Region,

P.Category,

SUM(OD.Quantity) AS TotalQuantity

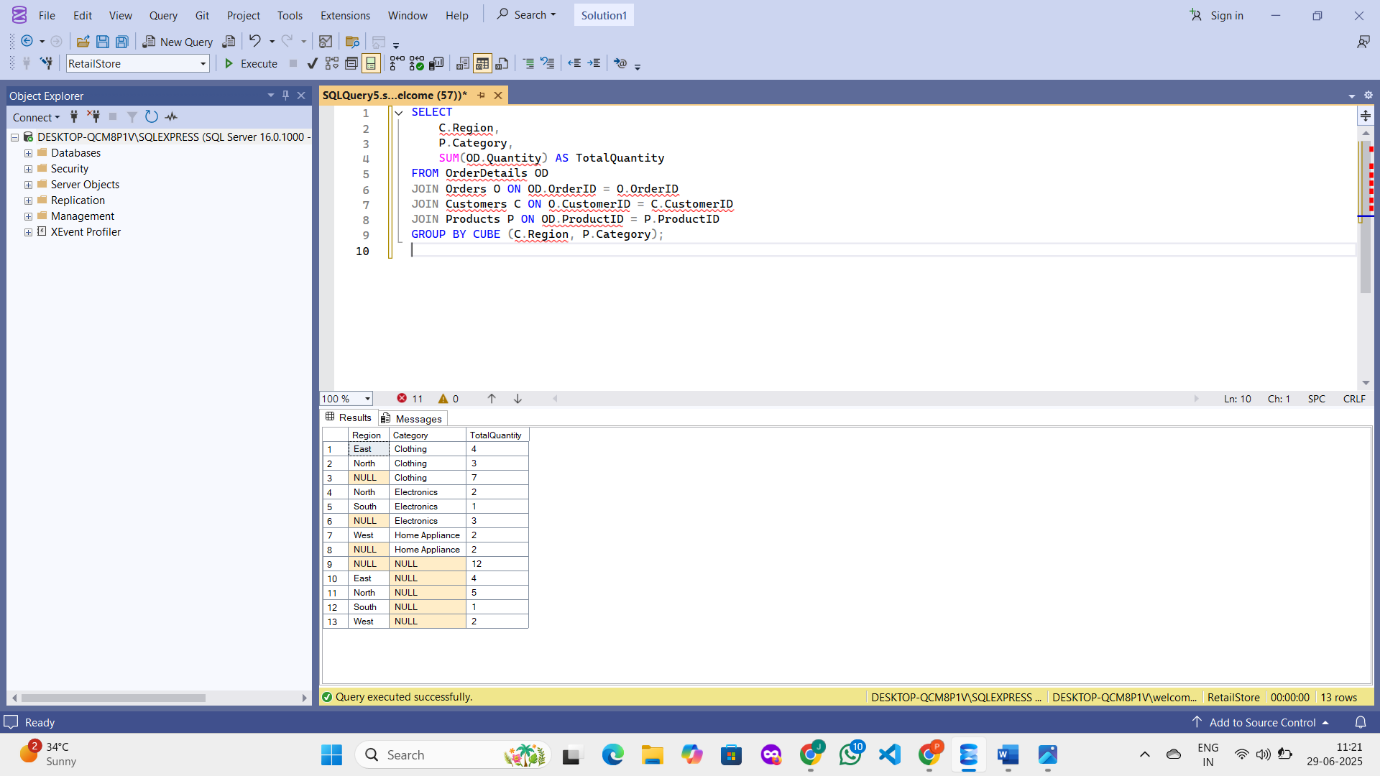
FROM OrderDetails OD

JOIN Orders O ON OD.OrderID = O.OrderID

JOIN Customers C ON O.CustomerID = C.CustomerID

JOIN Products P ON OD.ProductID = P.ProductID

GROUP BY CUBE (C.Region, P.Category);

**OUTPUT :  
**

**Exercise 3: CTEs and MERGE**

**Goal: Use WITH, CTEs, Recursive CTEs, and MERGE. Scenario.  
 a) Create a recursive CTE to generate a calendar table.**

**b) Use a MERGE statement to update or insert product prices from a staging table.**

**SOLUTION :**

**Objective :**

The objective of this exercise is to practice using advanced SQL constructs including Common Table Expressions (CTEs) and the MERGE statement.

**Generate Calendar Using Recursive CTE**

WITH CalendarCTE (DateValue) AS (

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

UNION ALL

SELECT DATEADD(DAY, 1, DateValue)

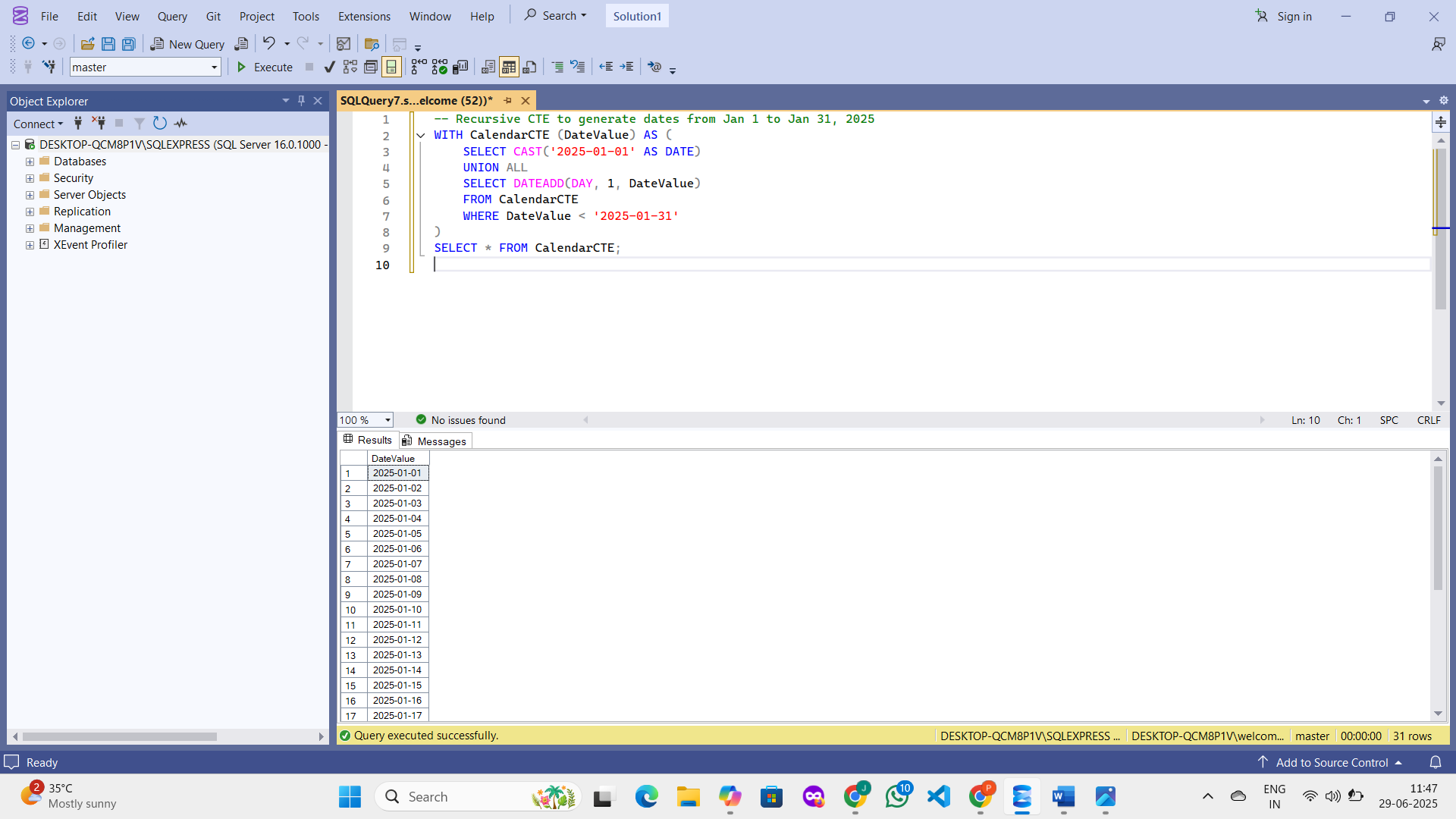
FROM CalendarCTE

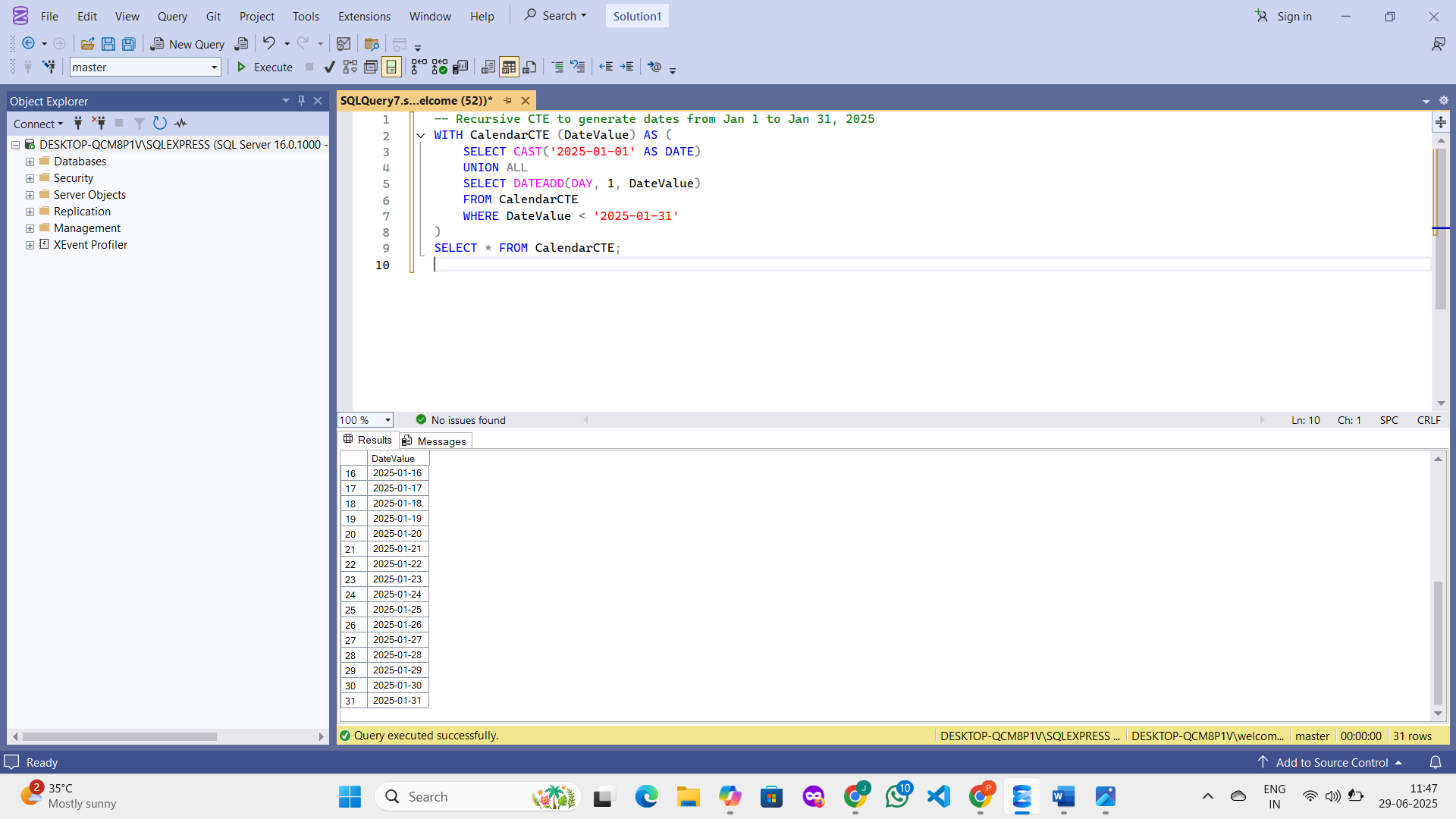
WHERE DateValue < '2025-01-31'

)

SELECT \* FROM CalendarCTE;

**OUTPUT :**





**Create StagingProducts Table with Updated Prices**

CREATE TABLE StagingProducts (

ProductID INT,

ProductName VARCHAR(100),

Category VARCHAR(100),

Price DECIMAL(10,2)

);

INSERT INTO StagingProducts VALUES

(1, 'Laptop', 'Electronics', 85000), -- Existing: updated price

(2, 'Smartphone', 'Electronics', 62000), -- Existing: updated price

(13, 'Air Conditioner', 'Home Appliance', 30000); -- New product

**Create Staging Table :**

CREATE TABLE StagingProducts (

ProductID INT,

ProductName VARCHAR(100),

Category VARCHAR(100),

Price DECIMAL(10,2)

);

INSERT INTO StagingProducts VALUES

(1, 'Laptop', 'Electronics', 85000),

(2, 'Smartphone', 'Electronics', 62000),

(13, 'Air Conditioner', 'Home Appliance', 30000);

**MERGE Statement :**

sql

CopyEdit

MERGE INTO Products AS Target

USING StagingProducts AS Source

ON Target.ProductID = Source.ProductID

WHEN MATCHED THEN

UPDATE SET

Target.ProductName = Source.ProductName,

Target.Category = Source.Category,

Target.Price = Source.Price

WHEN NOT MATCHED THEN

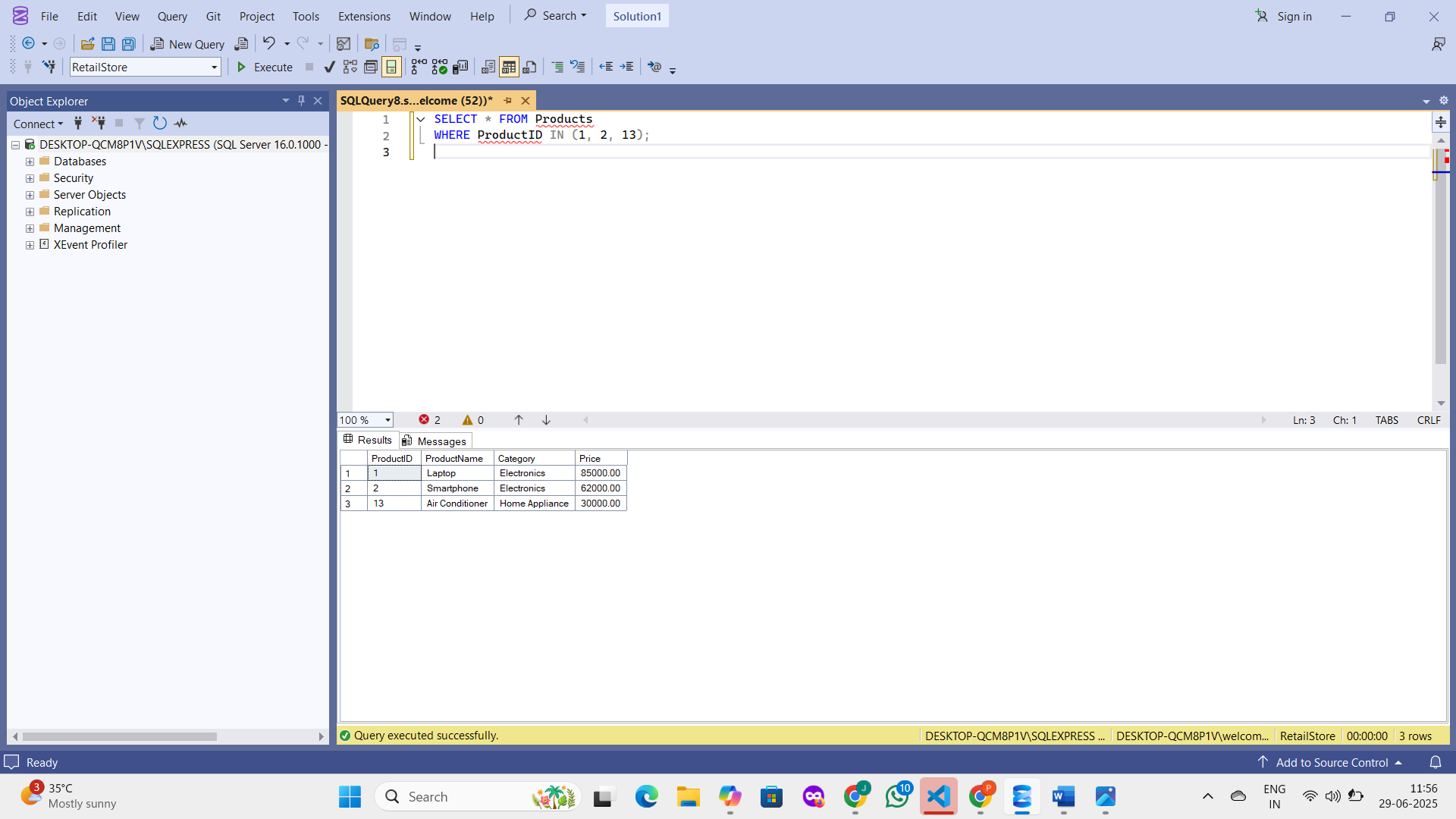
INSERT (ProductID, ProductName, Category, Price)

VALUES (Source.ProductID, Source.ProductName, Source.Category, Source.Price);

**Final Result Check**

SELECT \* FROM Products

WHERE ProductID IN (1, 2, 13);

**OUTPUT :**

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

**Scenario: Show monthly sales quantity per product in a pivoted format, and then unpivot it back.**

**SOLUTION :**

**Objective**

The goal is to analyze monthly product sales and transform the data:

* From row-based format to column-based format using PIVOT
* Then back to row-based format using UNPIVOT

**Use RetailStore Database**

USE RetailStore;

GO

**Aggregate Sales by Product and Month**

DROP TABLE IF EXISTS ProductMonthlySales;

GO

SELECT

P.ProductName,

FORMAT(O.OrderDate, 'MMM') AS SalesMonth,

SUM(OD.Quantity) AS TotalQuantity

INTO ProductMonthlySales

FROM OrderDetails OD

JOIN Orders O ON OD.OrderID = O.OrderID

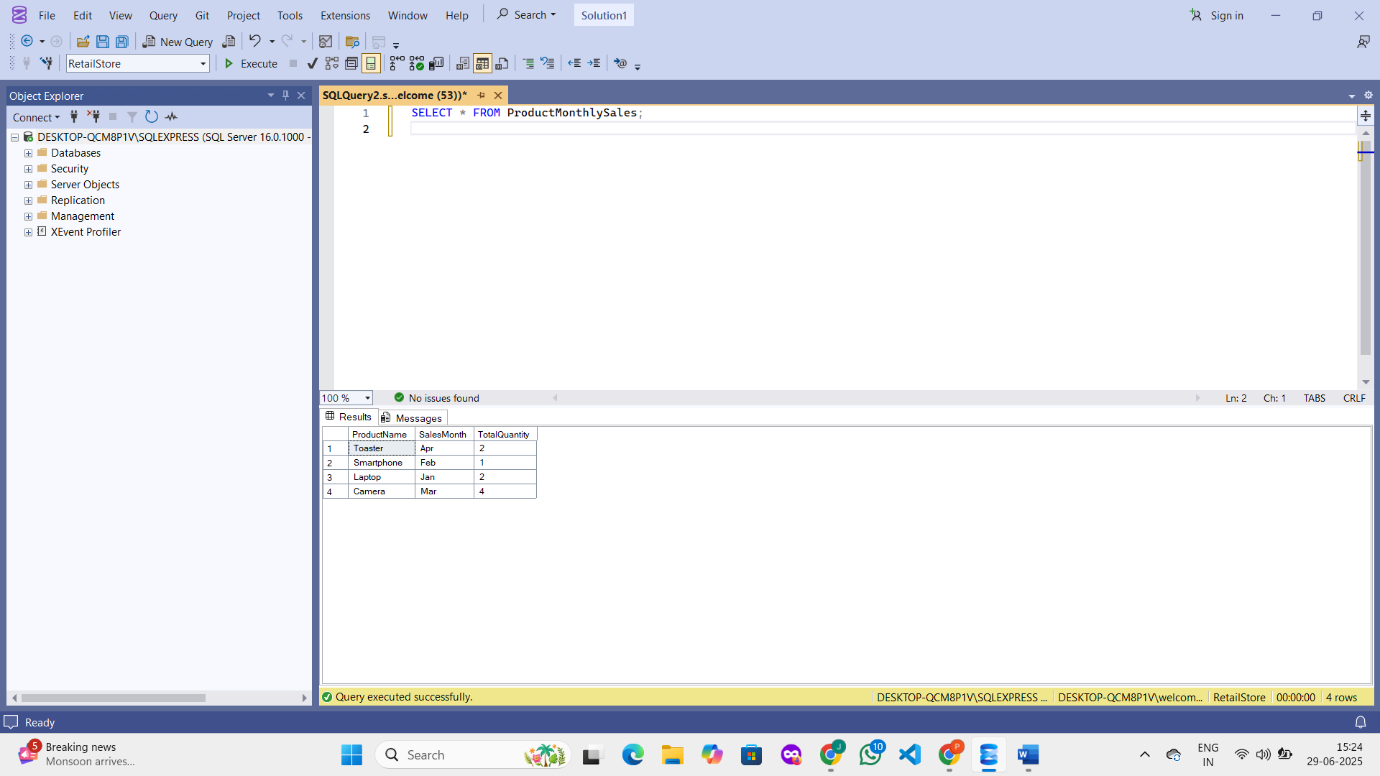
JOIN Products P ON OD.ProductID = P.ProductID

GROUP BY P.ProductName, FORMAT(O.OrderDate, 'MMM');

**Verify Aggregated Table**

SELECT \* FROM ProductMonthlySales;

**OUTPUT :**



**Apply PIVOT**

SELECT \*

FROM (

SELECT ProductName, SalesMonth, TotalQuantity

FROM ProductMonthlySales

) AS SourceTable

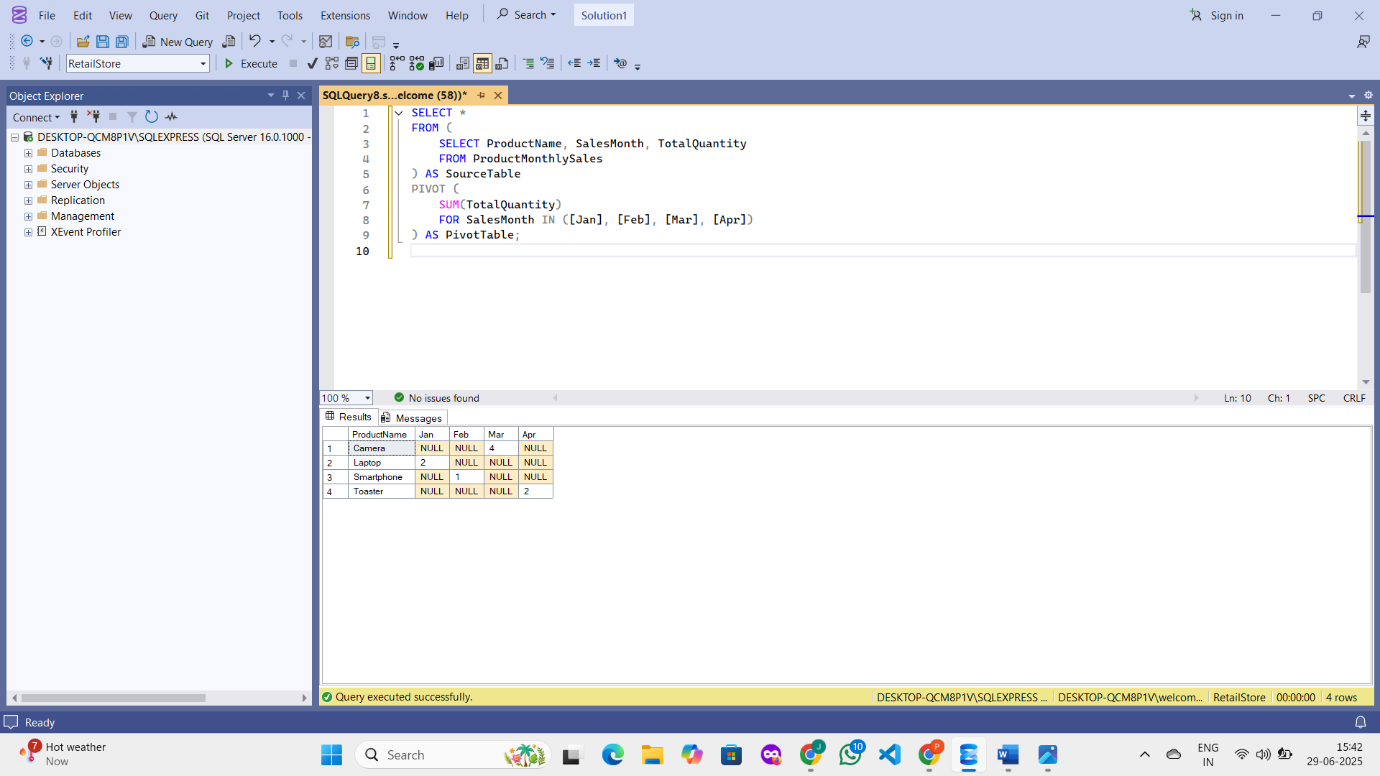
PIVOT (

SUM(TotalQuantity)

FOR SalesMonth IN ([Jan], [Feb], [Mar], [Apr])

) AS PivotTable;

**OUTPUT :**



**Apply UNPIVOT**

SELECT ProductName, SalesMonth, TotalQuantity

FROM (

SELECT \*

FROM (

SELECT ProductName, SalesMonth, TotalQuantity

FROM ProductMonthlySales

) AS SourceTable

PIVOT (

SUM(TotalQuantity)

FOR SalesMonth IN ([Jan], [Feb], [Mar], [Apr])

) AS PivotTable

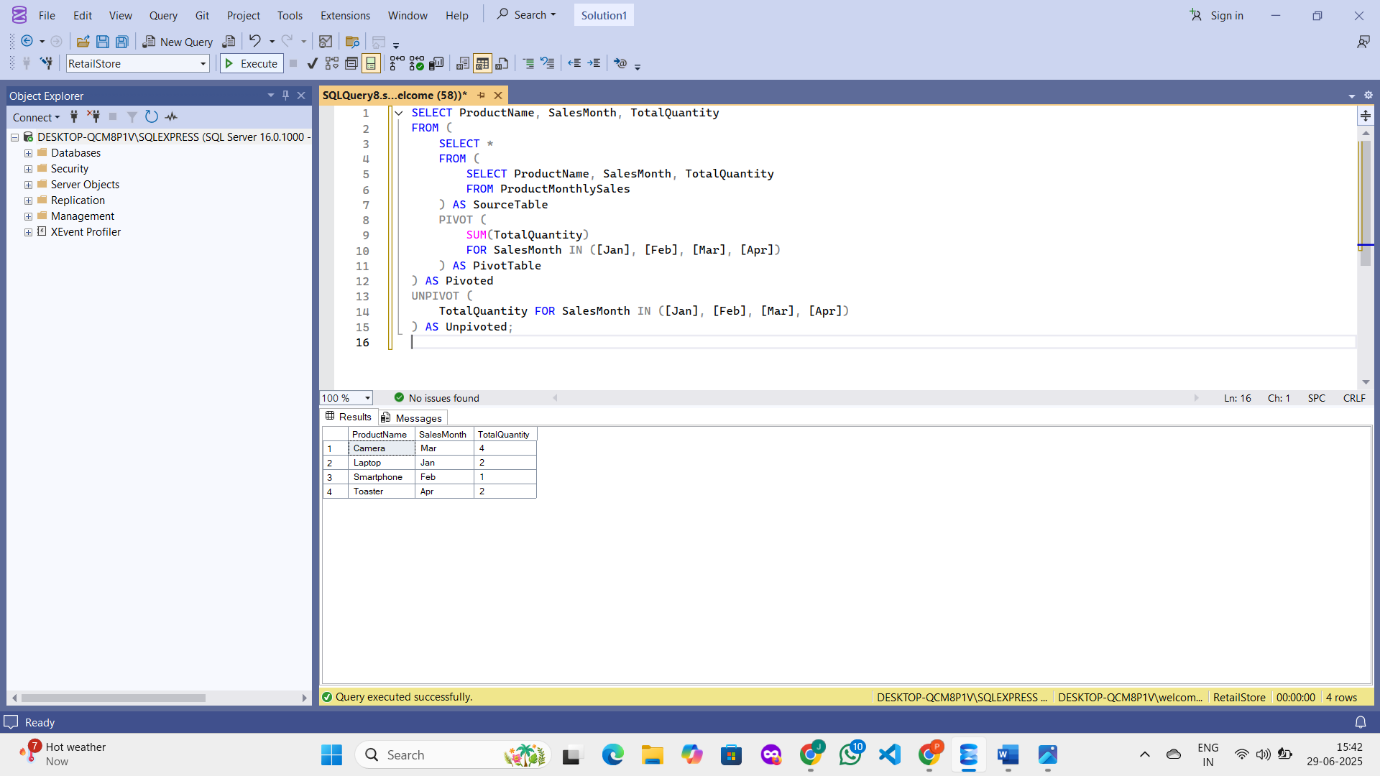
) AS Pivoted

UNPIVOT (

TotalQuantity FOR SalesMonth IN ([Jan], [Feb], [Mar], [Apr])

) AS Unpivoted;

**OUTPUT :**



**Exercise 5: Using CTE to Simplify a Query Goal: Use Common Table Expressions (CTEs) to simplify complex queries. Scenario: The business wants to find all customers who have placed more than 3 orders in total**.

**SOLUTION** :  
**Create the Database and Use It**

CREATE DATABASE RetailDB;

GO

USE RetailDB;

**Create Tables**

**Customers Table**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName NVARCHAR(100)

);

**Orders Table**

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

**Insert Sample Data**

**Insert Data into Customers**

INSERT INTO Customers (CustomerID, CustomerName)

VALUES

(1, 'Asha'),

(2, 'Bala'),

(3, 'Chetan'),

(4, 'Divya'),

(5, 'Esha');

**Insert Data into Orders**

INSERT INTO Orders (OrderID, CustomerID, OrderDate)

VALUES

(101, 1, '2023-01-01'),

(102, 1, '2023-01-05'),

(103, 1, '2023-01-10'),

(104, 1, '2023-01-15'),

(105, 2, '2023-02-01'),

(106, 2, '2023-02-05'),

(107, 3, '2023-03-01'),

(108, 3, '2023-03-05'),

(109, 3, '2023-03-10'),

(110, 3, '2023-03-15'),

(111, 3, '2023-03-20');

**Query Using CTE**

WITH CustomerOrderCounts AS (

SELECT

CustomerID,

COUNT(OrderID) AS OrderCount

FROM Orders

GROUP BY CustomerID

)

SELECT

c.CustomerID,

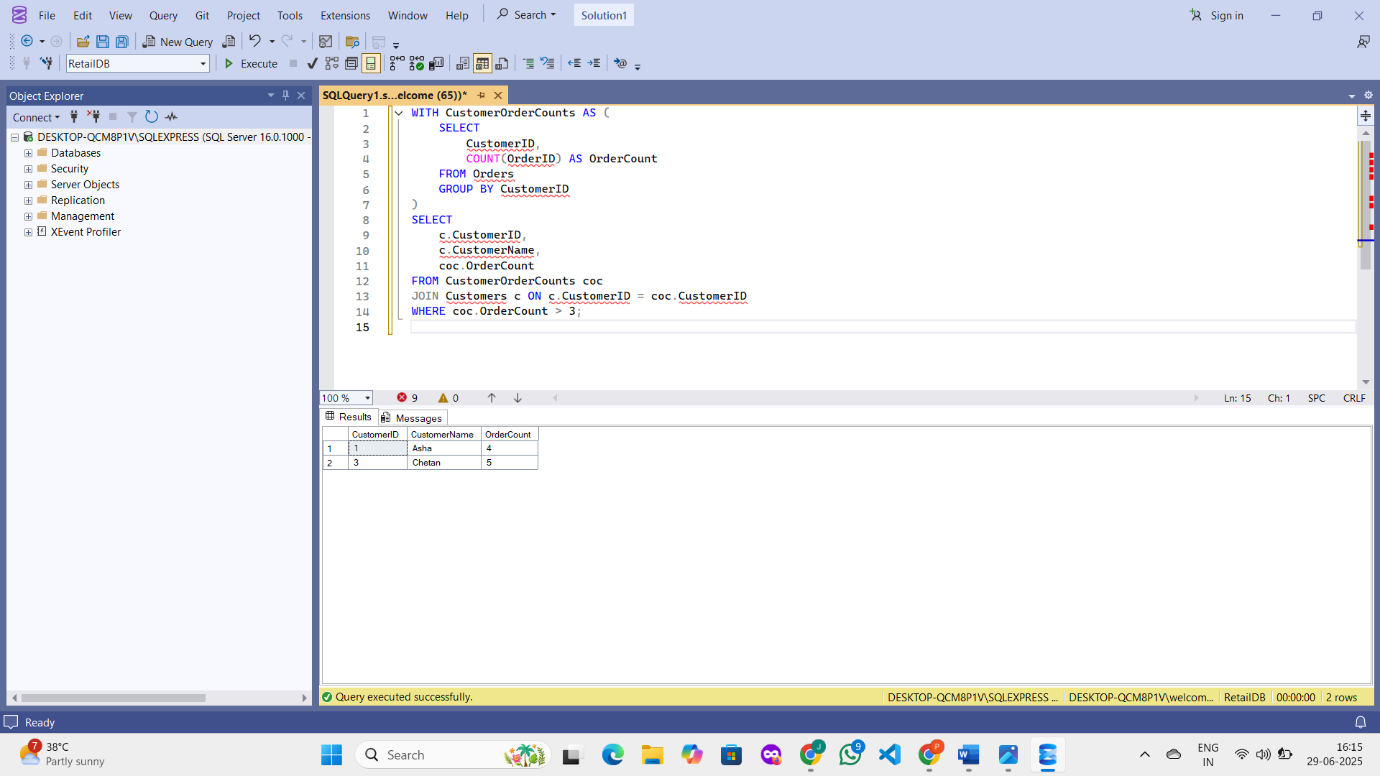
c.CustomerName,

coc.OrderCount

FROM CustomerOrderCounts coc

JOIN Customers c ON c.CustomerID = coc.CustomerID

WHERE coc.OrderCount > 3;

**OUTPUT :  
**