**Supply Chain Database Management**

The project is focused on creating a system, for managing a Supply Chain Database. Each table in this database holds information that's crucial for effectively handling different aspects of a supply chain ecosystem.

1. Suppliers Table: This table stores details about suppliers, including their contact information, location, and company details. It plays a role in maintaining a database of suppliers to ensure communication and sourcing.

2. Categories Table: Here different categories of products are stored. These categories help in organizing and classifying types of products making it easier for users to manage inventory and navigate through them.

3. Products Table: In this table we can find information about each product, such as its name, supplier details, category classification, pricing, and stock availability. This table forms the core of the inventory management system as it tracks products along with their availability and other related details.

4. OrderDetails Table: This table contains information about items within an order. Things like quantity purchased or ordered per item, unit price per item or any discounts applicable to them. It serves as a breakdown of each order placed which assists in fulfilling orders while also enabling financial tracking.

5. Customers Table: This table stores information, about customers, including their contact details and location. It plays a role in managing customer relationships tracking purchases and providing services.

6. Shippers Table: The Shippers Table contains data about shipping companies or entities for delivering orders. It includes their contact information. Helps in managing shipping services.

7. Orders Table: The Orders Table holds information regarding orders, such as the order date, shipping details and customer IDs. It serves as a repository for order related data facilitating order processing, tracking and management.

8. Transactions Table: The Transactions Table keeps records of transactions that include details about the products sold customers involved, quantities, pricing, and any applicable discounts. It aids in tracking understanding sales patterns accurately and analysing customer purchasing behaviour.

The objective of the Supply Chain Database Management project is to create an efficient database system specifically designed for supply chain management. This database will contain information, for monitoring and optimizing various aspects of the supply chain process. A system, like this is beneficial as it optimizes operations enhances efficiency and ensures a movement of goods and services, across the supply chain.

Reason for Selection:

I chose this project to streamline the supply chain operations by centralizing data on suppliers, products orders, and customers. Having a database system enables us to better manage inventory track orders effectively and gain an understanding of transactional details. This centralized system promotes efficiency in our processes while facilitating decision making and the analysis of supply chain performance, for improvement.

Database Table Design:

-- Create the Prasa573 database

CREATE DATABASE Prasa573

USE Prasa573

-- Create Suppliers Table

CREATE TABLE Suppliers (

SupplierID INT PRIMARY KEY,

CompanyName VARCHAR(100),

ContactName VARCHAR(50),

Address VARCHAR(100),

City VARCHAR(50),

Region VARCHAR(50),

Country VARCHAR(50),

Phone VARCHAR(20)

);

SELECT \* FROM Suppliers

-- Create table Categories

CREATE TABLE Categories (

CategoryID INT PRIMARY KEY,

CategoryName VARCHAR(100),

Description VARCHAR(255)

);

SELECT \* FROM Categories

-- Create Products Table

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

SupplierID INT, -- Foreign key referencing Suppliers table

CategoryID INT, -- Foreign key referencing Categories table

QuantityPerUnit VARCHAR(50),

UnitPrice DECIMAL(10, 2),

UnitsInStock INT,

UnitsOnOrder INT,

ReorderLevel INT,

Discontinued VARCHAR (10),

FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID),

FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)

);

SELECT \* FROM Products

-- Create table orderDetails

CREATE TABLE OrderDetails (

OrderID INT PRIMARY KEY,

ProductID INT,

UnitPrice DECIMAL(10, 2),

Quantity INT,

Discount DECIMAL(5, 2),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

SELECT \* FROM OrderDetails

--Create Table Customers

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CompanyName VARCHAR(100),

ContactName VARCHAR(50),

City VARCHAR(50),

Region VARCHAR(50),

Country VARCHAR(50),

Phone VARCHAR(20)

);

SELECT \* FROM Customers

-- Create Table Shippers

CREATE TABLE Shippers (

ShipperID INT PRIMARY KEY,

CompanyName VARCHAR(100),

Phone VARCHAR(20)

);

SELECT \* FROM Shippers

--Create Table Orders

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT, -- Foreign key referencing Customers table

OrderDate DATE,

RequiredDate DATE,

ShippedDate DATE,

ShipVia INT,

ShipAddress VARCHAR(100),

ShipCity VARCHAR(50),

ShipPostalCode VARCHAR(20),

ShipCountry VARCHAR(50),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

SELECT \* FROM Orders

--Create Table Transactions

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY,

OrderID INT,

ProductID INT,

CustomerID INT,

ShipperID INT,

Quantity INT,

TotalPrice DECIMAL(10, 2),

DiscountApplied DECIMAL(5, 2),

TransactionDate DATE,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID),

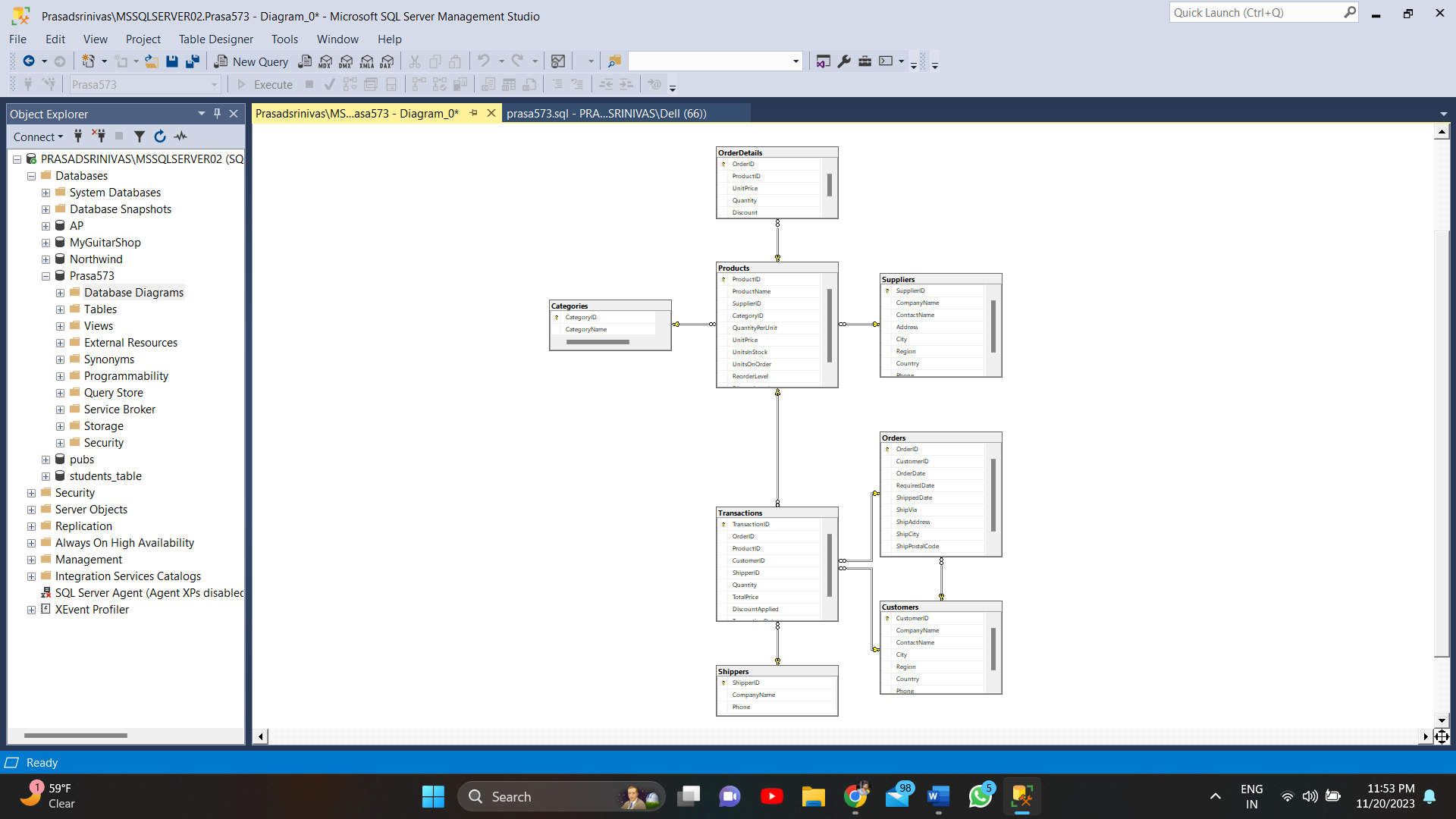
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

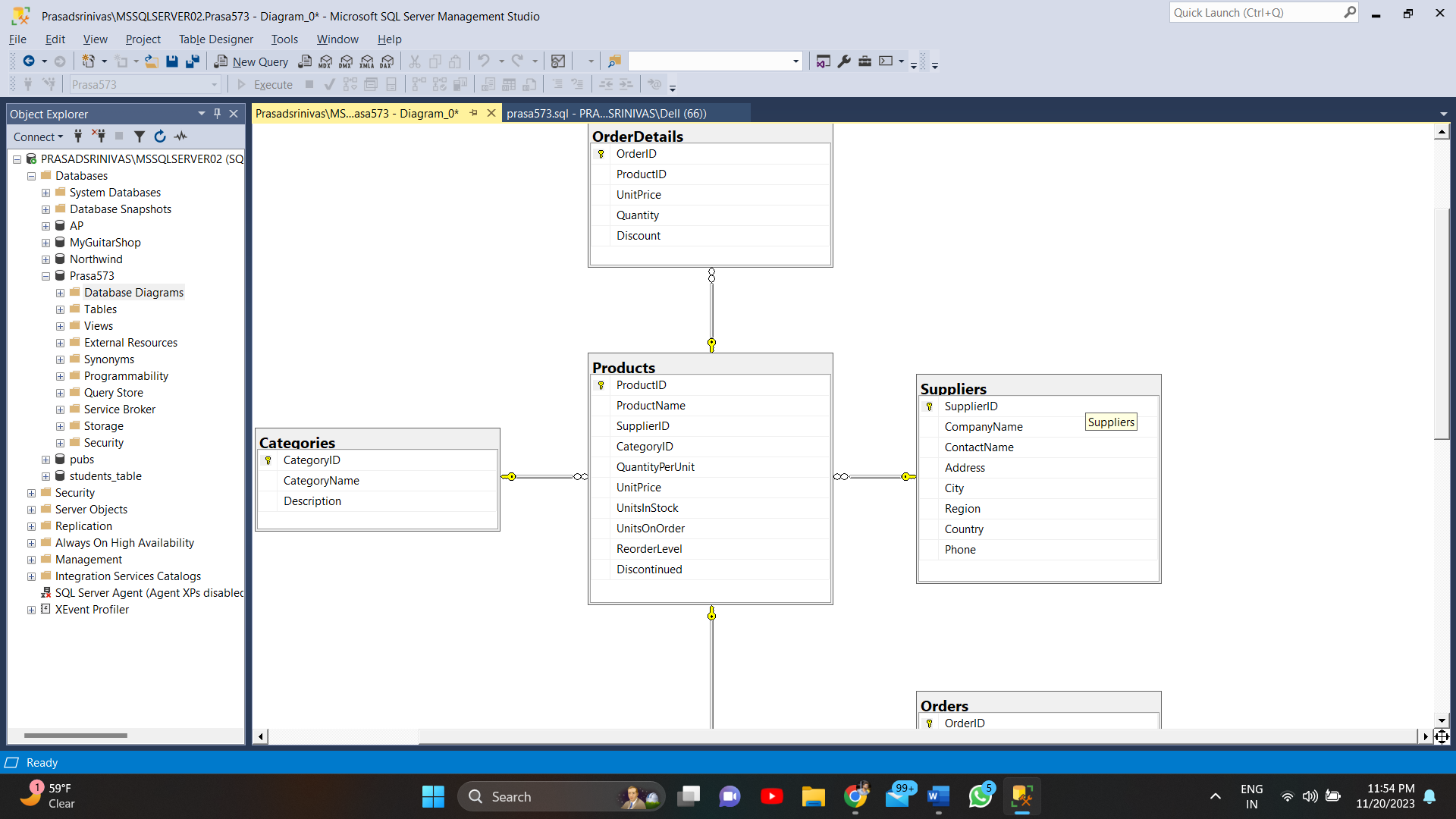
FOREIGN KEY (ShipperID) REFERENCES Shippers(ShipperID)

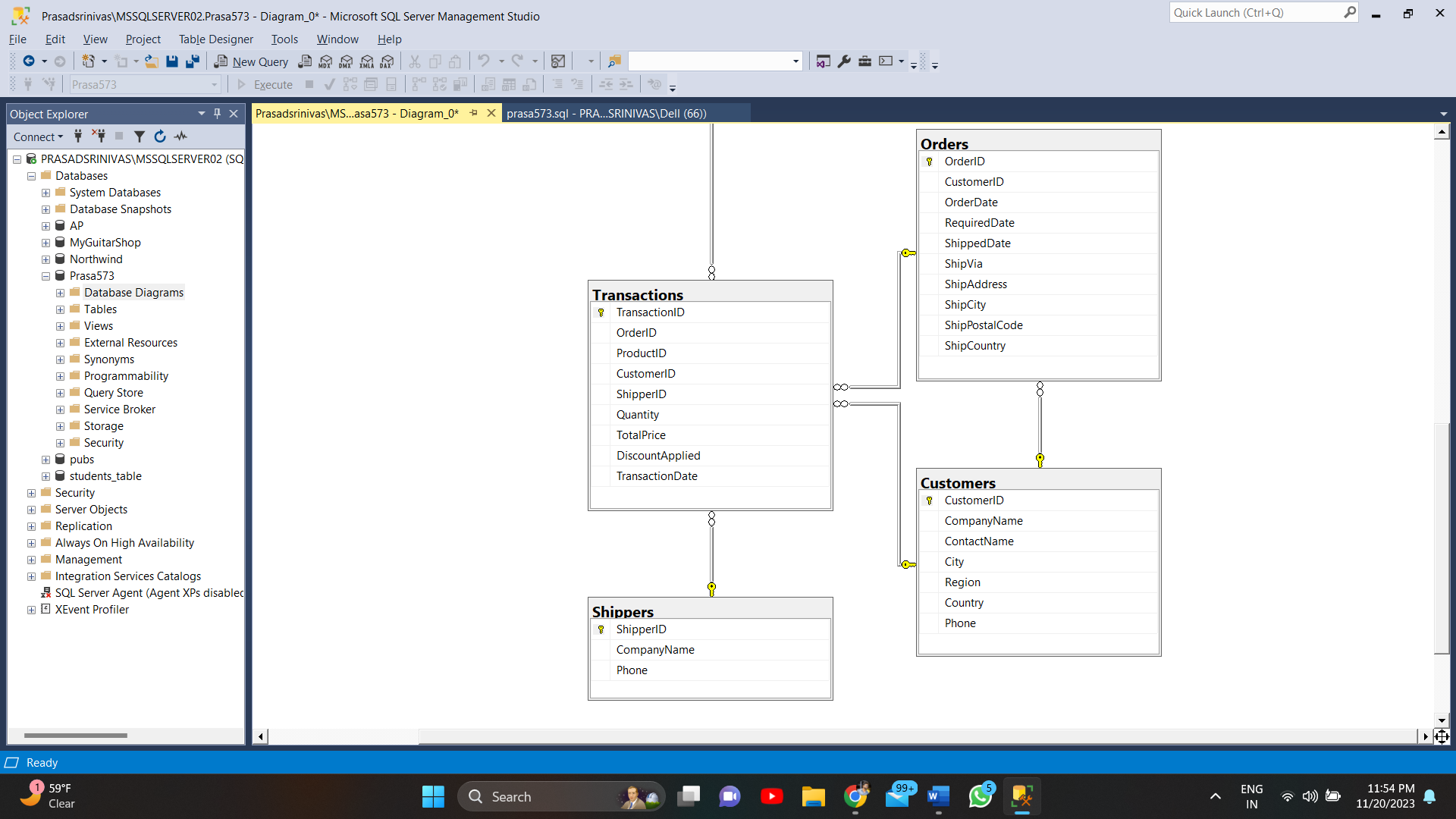
);

SELECT \* FROM Transactions

ERD Diagram:







Populating tables:

-- Insert data into the Suppliers table

INSERT INTO Suppliers (SupplierID, CompanyName, ContactName, Address, City, Region, Country, Phone)

VALUES

(1, 'ABC Corporation', 'John Doe', '123 Main St', 'Anytown', 'North Region', 'Atlantis', '123-456-7890'),

(2, 'XYZ Industries', 'Jane Smith', '456 Elm St', 'Otherville', 'East Region', 'Lemuria', '987-654-3210'),

(3, 'Smithson Ltd.', 'Alice Johnson', '789 Oak St', 'Sometown', 'West Region', 'Eldorado', '111-222-3333'),

(4, 'Johnson & Sons', 'Bob Wilson', '321 Pine St', 'Yourtown', 'South Region', 'Avalon', '444-555-6666'),

(5, 'Maple Enterprises', 'Emily Brown', '654 Maple St', 'Hometown', 'Central Region', 'Shangri-La', '777-888-9999'),

(6, 'Cedar Co.', 'David Lee', '987 Cedar St', 'Theirtown', 'Coastal Region', 'Hyperborea', '222-333-4444'),

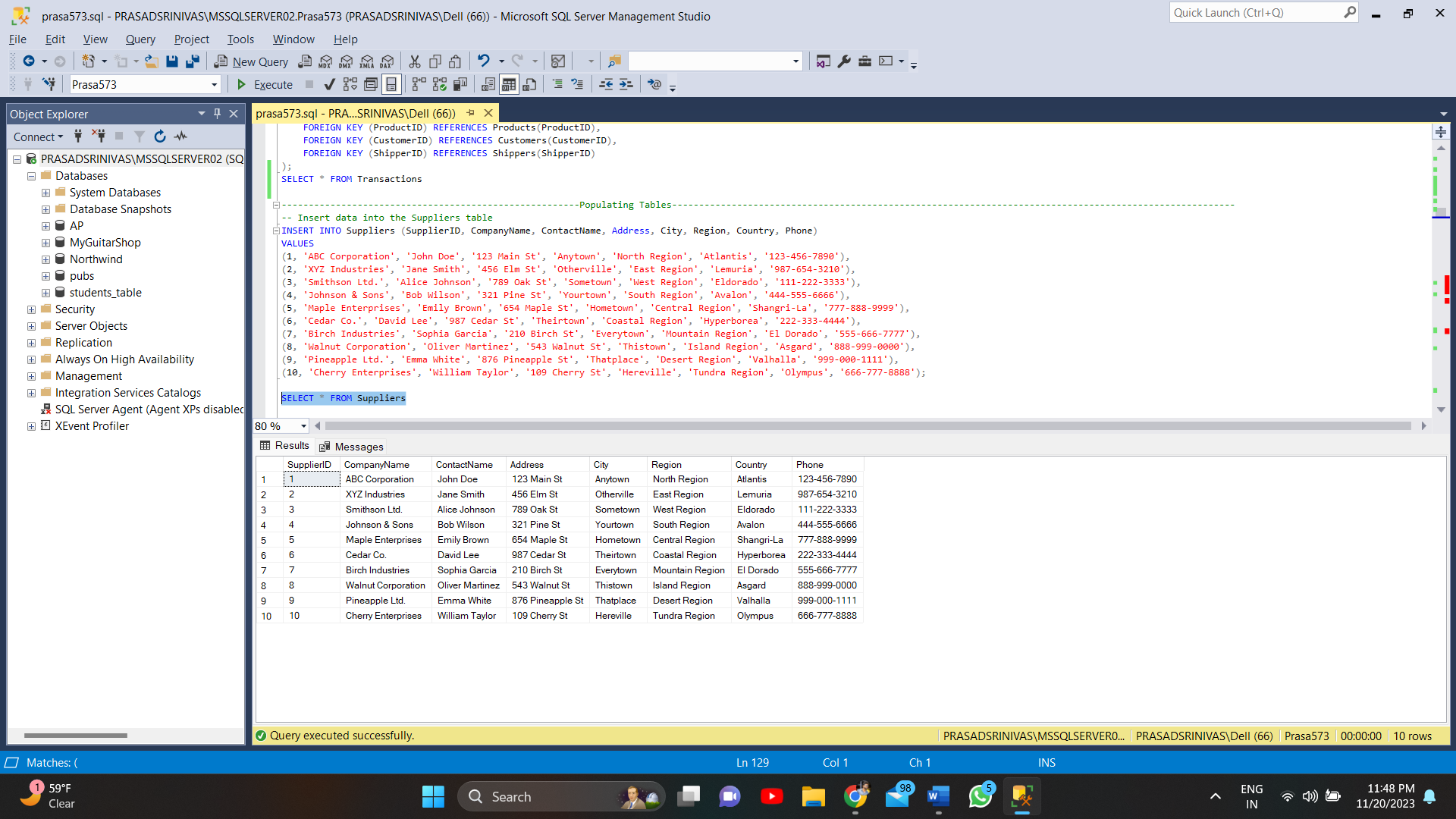
(7, 'Birch Industries', 'Sophia Garcia', '210 Birch St', 'Everytown', 'Mountain Region', 'El Dorado', '555-666-7777'),

(8, 'Walnut Corporation', 'Oliver Martinez', '543 Walnut St', 'Thistown', 'Island Region', 'Asgard', '888-999-0000'),

(9, 'Pineapple Ltd.', 'Emma White', '876 Pineapple St', 'Thatplace', 'Desert Region', 'Valhalla', '999-000-1111'),

(10, 'Cherry Enterprises', 'William Taylor', '109 Cherry St', 'Hereville', 'Tundra Region', 'Olympus', '666-777-8888');

SELECT \* FROM Suppliers



-- Insert data into the categories table

INSERT INTO Categories (CategoryID, CategoryName, Description)

VALUES

(1, 'Electronics', 'Products related to electronic devices and components'),

(2, 'Clothing', 'Clothing items for various ages and genders'),

(3, 'Home and Garden', 'Items for home decor and gardening'),

(4, 'Books', 'Different genres of books and publications'),

(5, 'Sports and Outdoors', 'Sporting goods and outdoor equipment'),

(6, 'Toys and Games', 'Various toys and games for all ages'),

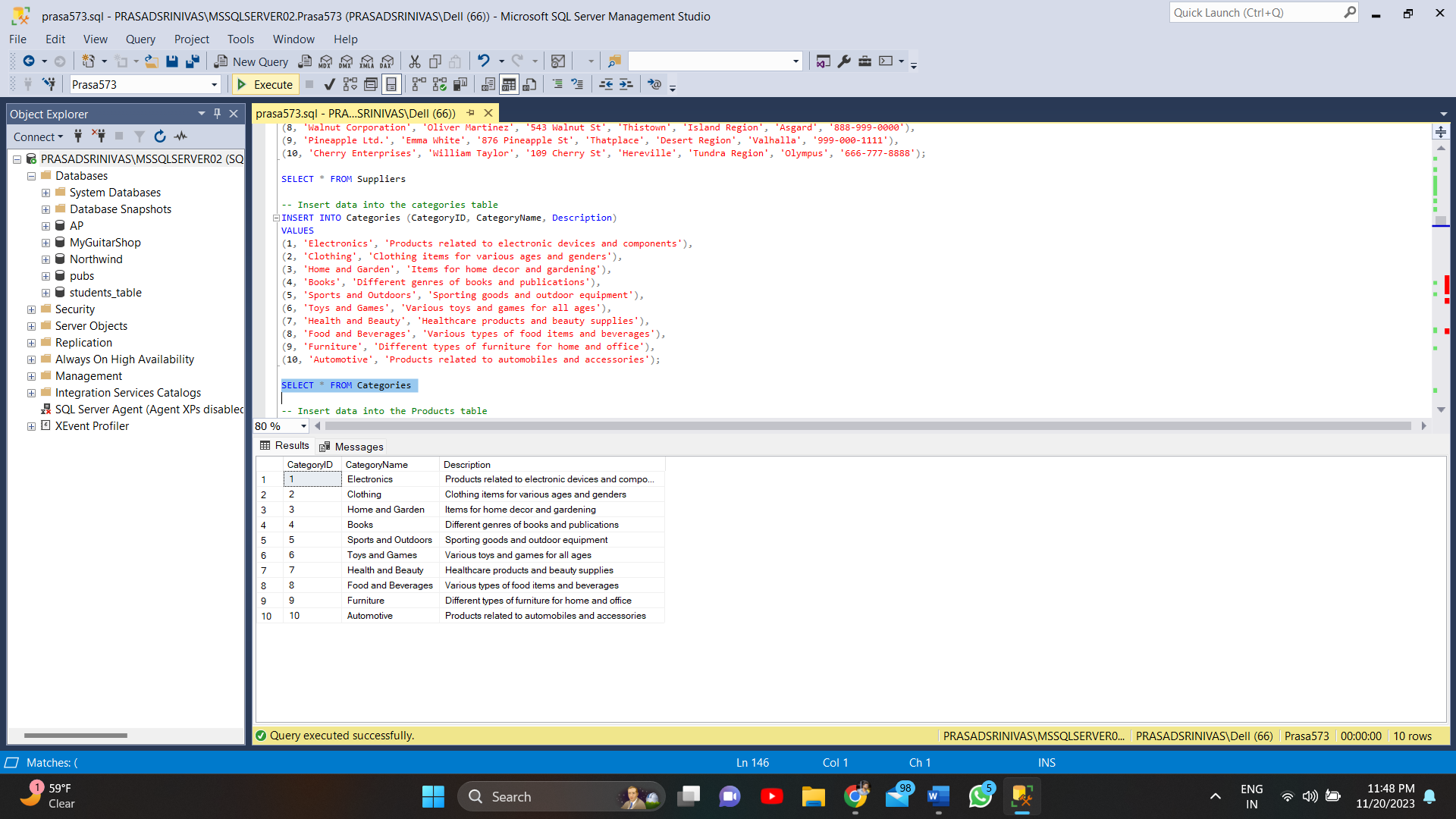
(7, 'Health and Beauty', 'Healthcare products and beauty supplies'),

(8, 'Food and Beverages', 'Various types of food items and beverages'),

(9, 'Furniture', 'Different types of furniture for home and office'),

(10, 'Automotive', 'Products related to automobiles and accessories');

SELECT \* FROM Categories



-- Insert data into the Products table

INSERT INTO Products (ProductID, ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued)

VALUES

(1, 'Smartphone', 1, 1, '1 unit', 599.99, 50, 10, 20, 'No'),

(2, 'T-Shirt', 2, 2, '1 unit', 19.99, 100, 20, 30, 'No'),

(3, 'Gardening Tools Set', 3, 3, '1 set', 49.99, 30, 5, 10, 'No'),

(4, 'Fiction Novel', 4, 4, '1 unit', 12.99, 80, 15, 25, 'No'),

(5, 'Soccer Ball', 5, 5, '1 unit', 29.99, 40, 8, 15, 'No'),

(6, 'Board Game', 6, 6, '1 unit', 39.99, 60, 12, 20, 'No'),

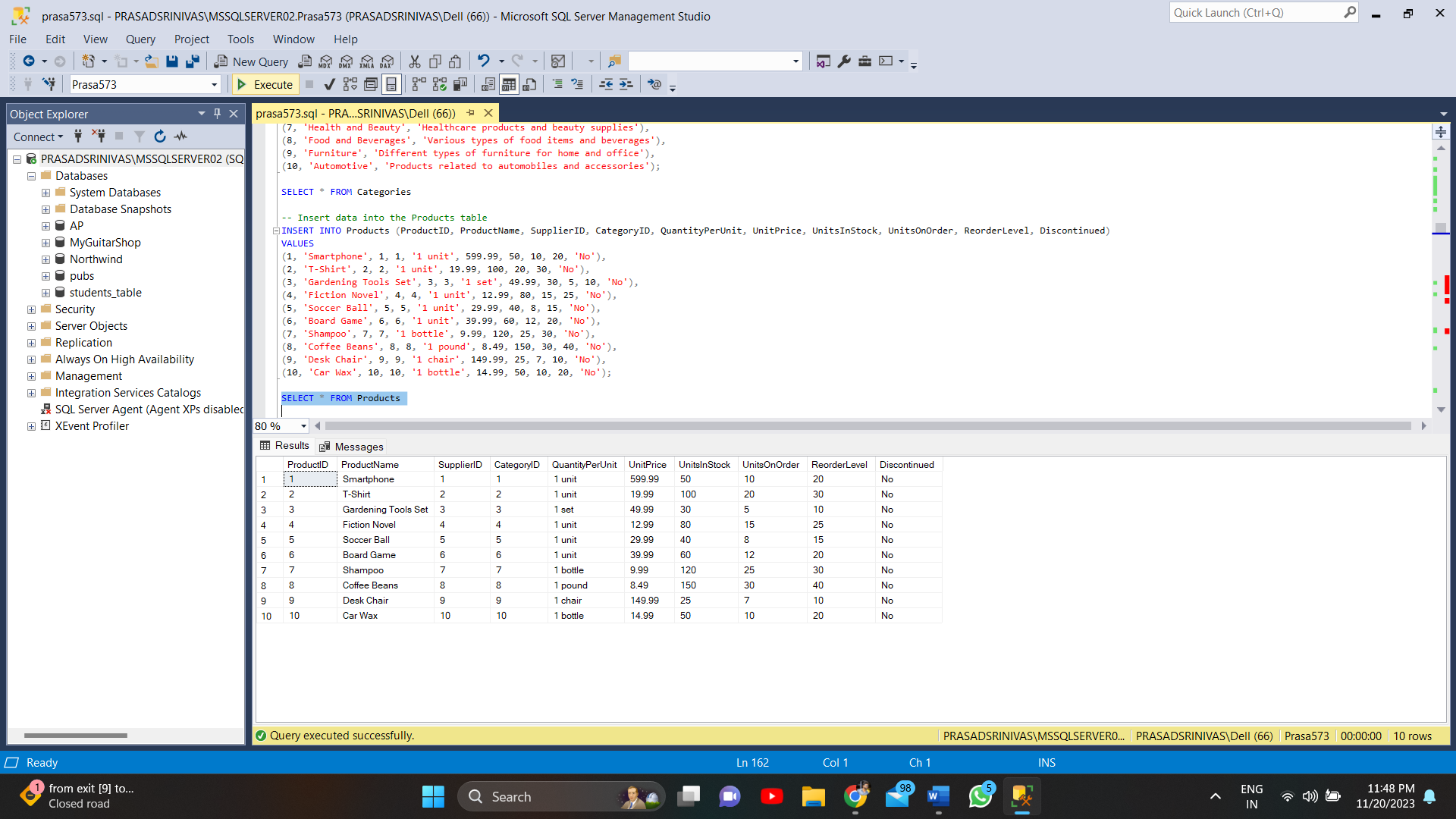
(7, 'Shampoo', 7, 7, '1 bottle', 9.99, 120, 25, 30, 'No'),

(8, 'Coffee Beans', 8, 8, '1 pound', 8.49, 150, 30, 40, 'No'),

(9, 'Desk Chair', 9, 9, '1 chair', 149.99, 25, 7, 10, 'No'),

(10, 'Car Wax', 10, 10, '1 bottle', 14.99, 50, 10, 20, 'No');

SELECT \* FROM Products



-- Insert data into the OrderDeatils table

INSERT INTO OrderDetails (OrderID, ProductID, UnitPrice, Quantity, Discount)

VALUES

(1, 1, 599.99, 2, 0.05),

(2, 3, 49.99, 1, 0.1),

(3, 6, 39.99, 3, 0.15),

(4, 8, 8.49, 5, 0.2),

(5, 2, 19.99, 4, 0.1),

(6, 4, 12.99, 2, 0.05),

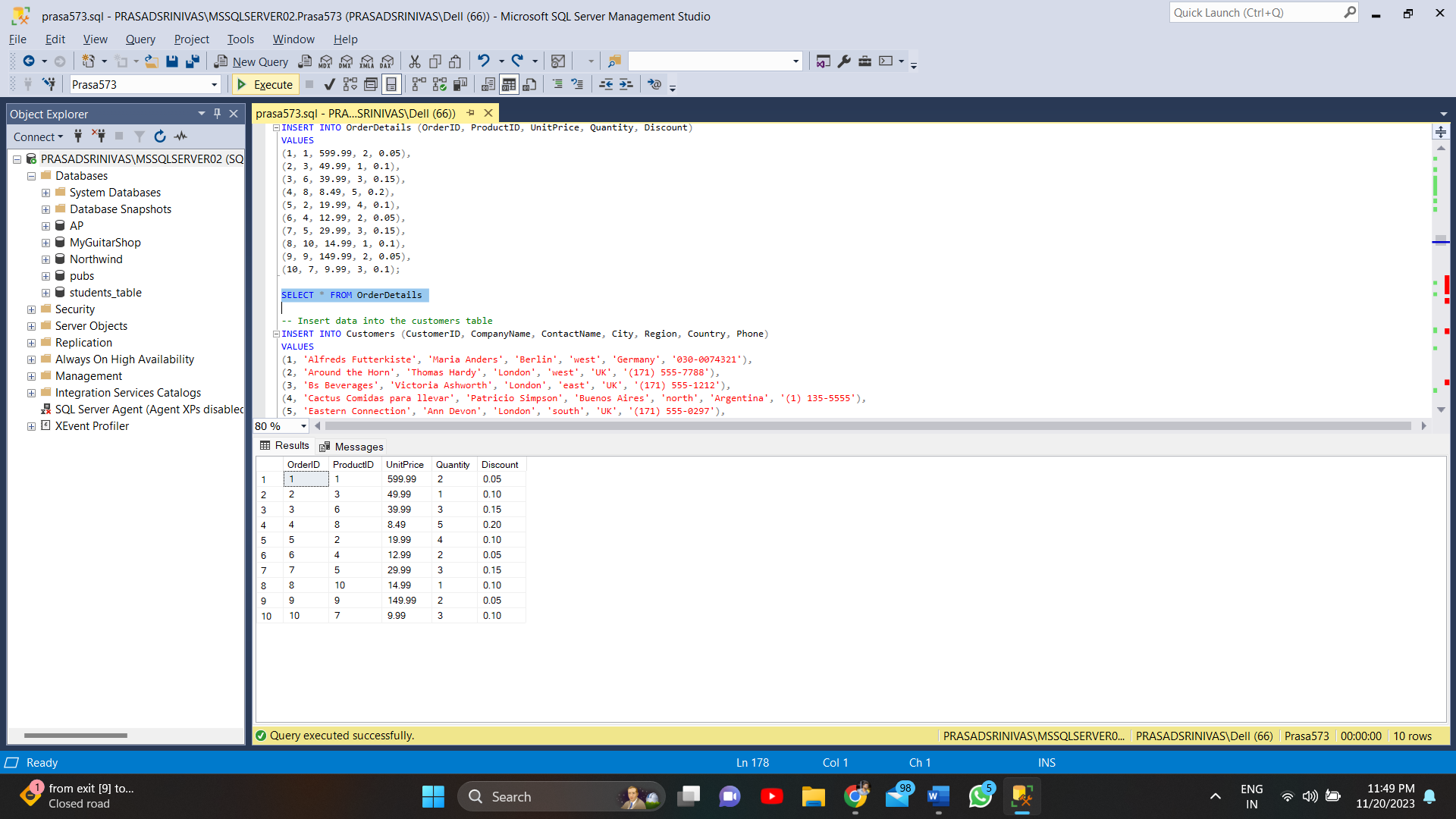
(7, 5, 29.99, 3, 0.15),

(8, 10, 14.99, 1, 0.1),

(9, 9, 149.99, 2, 0.05),

(10, 7, 9.99, 3, 0.1);

SELECT \* FROM OrderDetails



-- Insert data into the customers table

INSERT INTO Customers (CustomerID, CompanyName, ContactName, City, Region, Country, Phone)

VALUES

(1, 'Alfreds Futterkiste', 'Maria Anders', 'Berlin', 'west', 'Germany', '030-0074321'),

(2, 'Around the Horn', 'Thomas Hardy', 'London', 'west', 'UK', '(171) 555-7788'),

(3, 'Bs Beverages', 'Victoria Ashworth', 'London', 'east', 'UK', '(171) 555-1212'),

(4, 'Cactus Comidas para llevar', 'Patricio Simpson', 'Buenos Aires', 'north', 'Argentina', '(1) 135-5555'),

(5, 'Eastern Connection', 'Ann Devon', 'London', 'south', 'UK', '(171) 555-0297'),

(6, 'Frankenversand', 'Peter Franken', 'München', 'west', 'Germany', '089-0877310'),

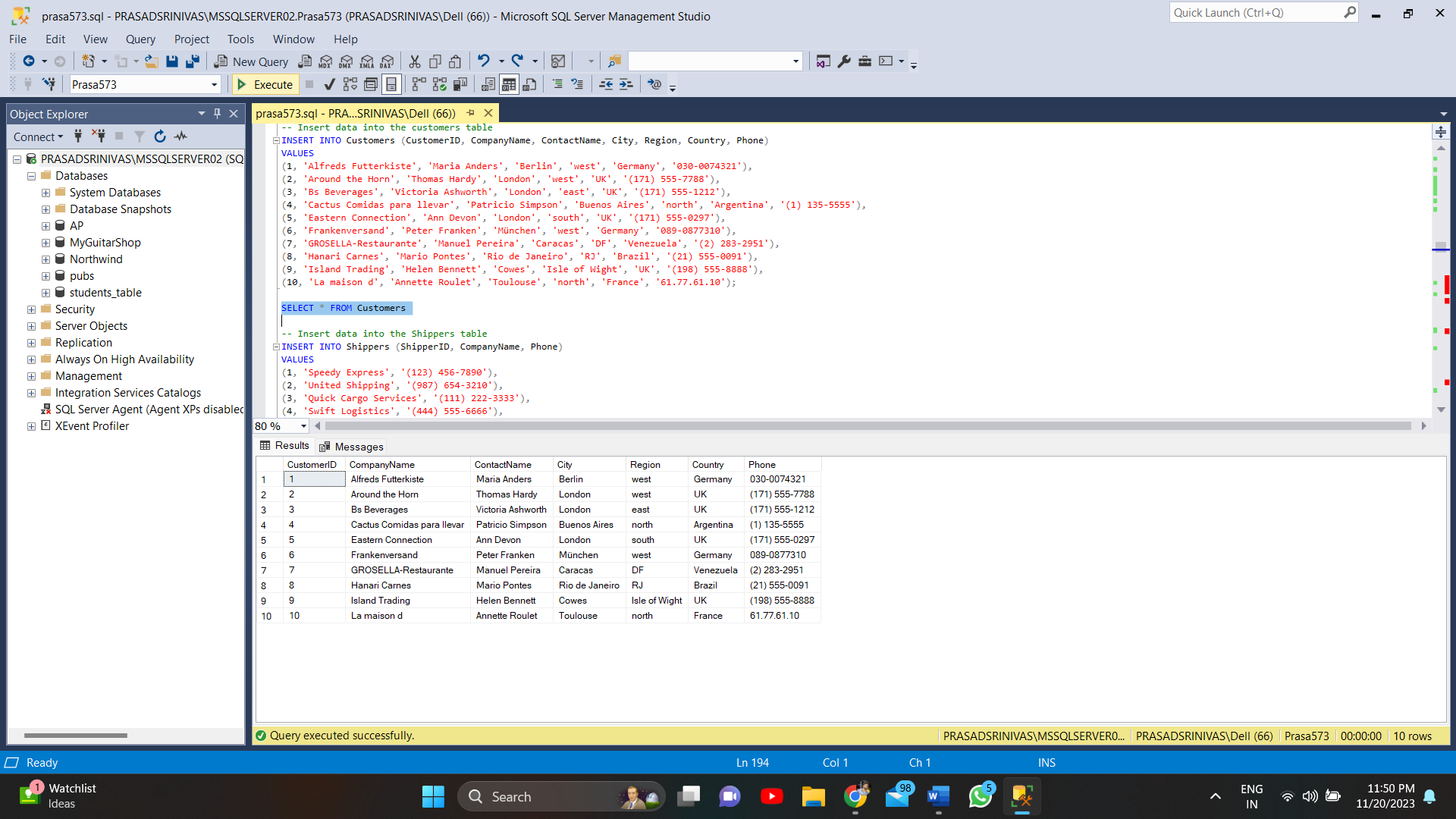
(7, 'GROSELLA-Restaurante', 'Manuel Pereira', 'Caracas', 'DF', 'Venezuela', '(2) 283-2951'),

(8, 'Hanari Carnes', 'Mario Pontes', 'Rio de Janeiro', 'RJ', 'Brazil', '(21) 555-0091'),

(9, 'Island Trading', 'Helen Bennett', 'Cowes', 'Isle of Wight', 'UK', '(198) 555-8888'),

(10, 'La maison d', 'Annette Roulet', 'Toulouse', 'north', 'France', '61.77.61.10');

SELECT \* FROM Customers



-- Insert data into the Shippers table

INSERT INTO Shippers (ShipperID, CompanyName, Phone)

VALUES

(1, 'Speedy Express', '(123) 456-7890'),

(2, 'United Shipping', '(987) 654-3210'),

(3, 'Quick Cargo Services', '(111) 222-3333'),

(4, 'Swift Logistics', '(444) 555-6666'),

(5, 'FastTrack Couriers', '(777) 888-9999'),

(6, 'Rapid Delivery', '(222) 333-4444'),

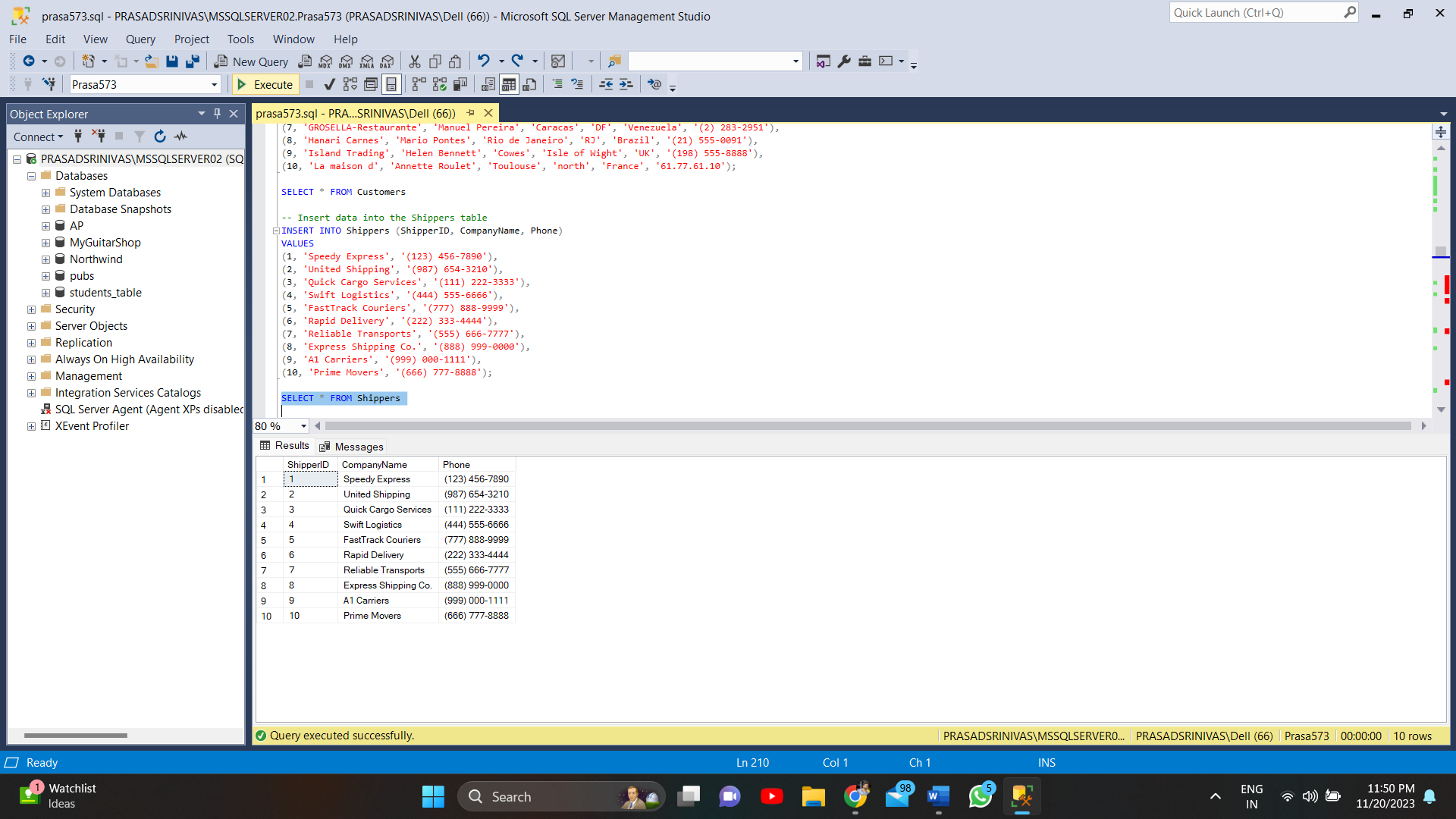
(7, 'Reliable Transports', '(555) 666-7777'),

(8, 'Express Shipping Co.', '(888) 999-0000'),

(9, 'A1 Carriers', '(999) 000-1111'),

(10, 'Prime Movers', '(666) 777-8888');

SELECT \* FROM Shippers



-- Insert data into the orders table

INSERT INTO Orders (OrderID, CustomerID, OrderDate, RequiredDate, ShippedDate, ShipVia, ShipAddress, ShipCity, ShipPostalCode, ShipCountry)

VALUES

(1, 1, '2023-01-15', '2023-01-20', '2023-01-18', 1, '123 Main St', 'Anytown', '12345', 'United States'),

(2, 3, '2023-02-10', '2023-02-15', '2023-02-12', 2, '456 Elm St', 'Otherville', '67890', 'Canada'),

(3, 5, '2023-03-05', '2023-03-10', '2023-03-08', 1, '789 Oak St', 'Sometown', '23456', 'Australia'),

(4, 2, '2023-04-12', '2023-04-17', '2023-04-15', 3, '321 Pine St', 'Yourtown', '78901', 'Germany'),

(5, 4, '2023-05-20', '2023-05-25', '2023-05-23', 2, '654 Maple St', 'Hometown', '34567', 'France'),

(6, 6, '2023-06-18', '2023-06-23', '2023-06-21', 1, '987 Cedar St', 'Theirtown', '89012', 'Japan'),

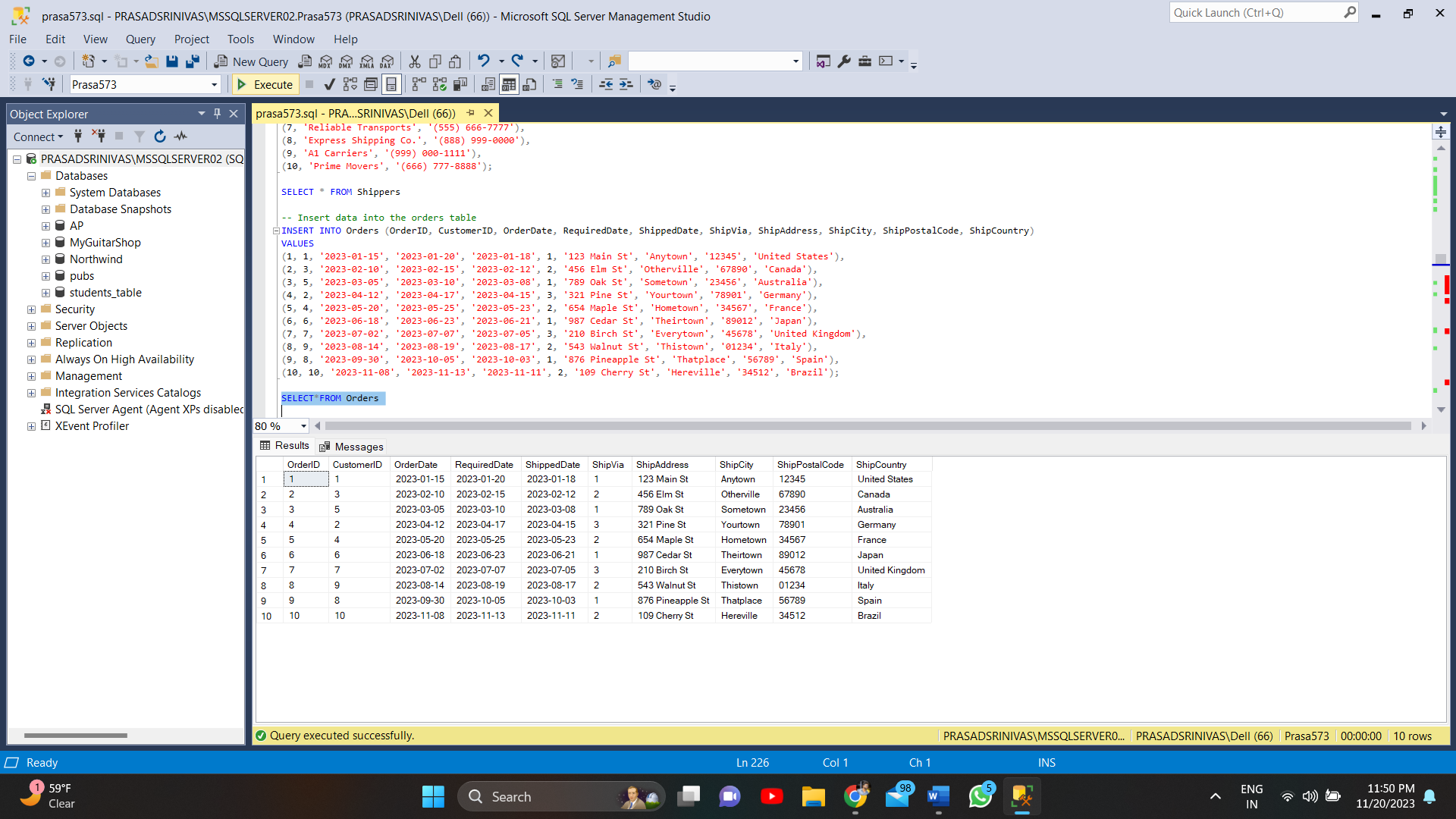
(7, 7, '2023-07-02', '2023-07-07', '2023-07-05', 3, '210 Birch St', 'Everytown', '45678', 'United Kingdom'),

(8, 9, '2023-08-14', '2023-08-19', '2023-08-17', 2, '543 Walnut St', 'Thistown', '01234', 'Italy'),

(9, 8, '2023-09-30', '2023-10-05', '2023-10-03', 1, '876 Pineapple St', 'Thatplace', '56789', 'Spain'),

(10, 10, '2023-11-08', '2023-11-13', '2023-11-11', 2, '109 Cherry St', 'Hereville', '34512', 'Brazil');

SELECT\*FROM Orders



-- Insert data into the Transactions table

INSERT INTO Transactions (TransactionID, OrderID, ProductID, CustomerID, ShipperID, Quantity, TotalPrice, DiscountApplied, TransactionDate)

VALUES

(1, 1, 1, 1, 1, 2, 1199.98, 0.05, '2023-01-18'),

(2, 2, 3, 3, 2, 1, 44.99, 0.1, '2023-02-12'),

(3, 3, 6, 5, 1, 3, 101.97, 0.15, '2023-03-08'),

(4, 4, 8, 2, 3, 5, 33.72, 0.2, '2023-04-15'),

(5, 5, 2, 4, 2, 4, 75.96, 0.1, '2023-05-23'),

(6, 6, 4, 6, 1, 2, 25.98, 0.05, '2023-06-21'),

(7, 7, 5, 7, 3, 3, 77.97, 0.15, '2023-07-05'),

(8, 8, 10, 9, 2, 1, 13.49, 0.1, '2023-08-17'),

(9, 9, 9, 8, 1, 2, 299.98, 0.05, '2023-10-03'),

(10, 10, 7, 10, 2, 3, 29.97, 0.1, '2023-11-11'),

(11, 1, 1, 1, 1, 3, 1799.97, 0.05, '2023-01-18'),

(12, 2, 3, 3, 2, 2, 99.98, 0.1, '2023-02-12'),

(13, 3, 6, 5, 1, 1, 39.99, 0.15, '2023-03-08'),

(14, 4, 8, 2, 3, 3, 25.47, 0.2, '2023-04-15'),

(15, 5, 2, 4, 2, 2, 39.98, 0.1, '2023-05-23'),

(16, 6, 4, 6, 1, 1, 12.99, 0.05, '2023-06-21'),

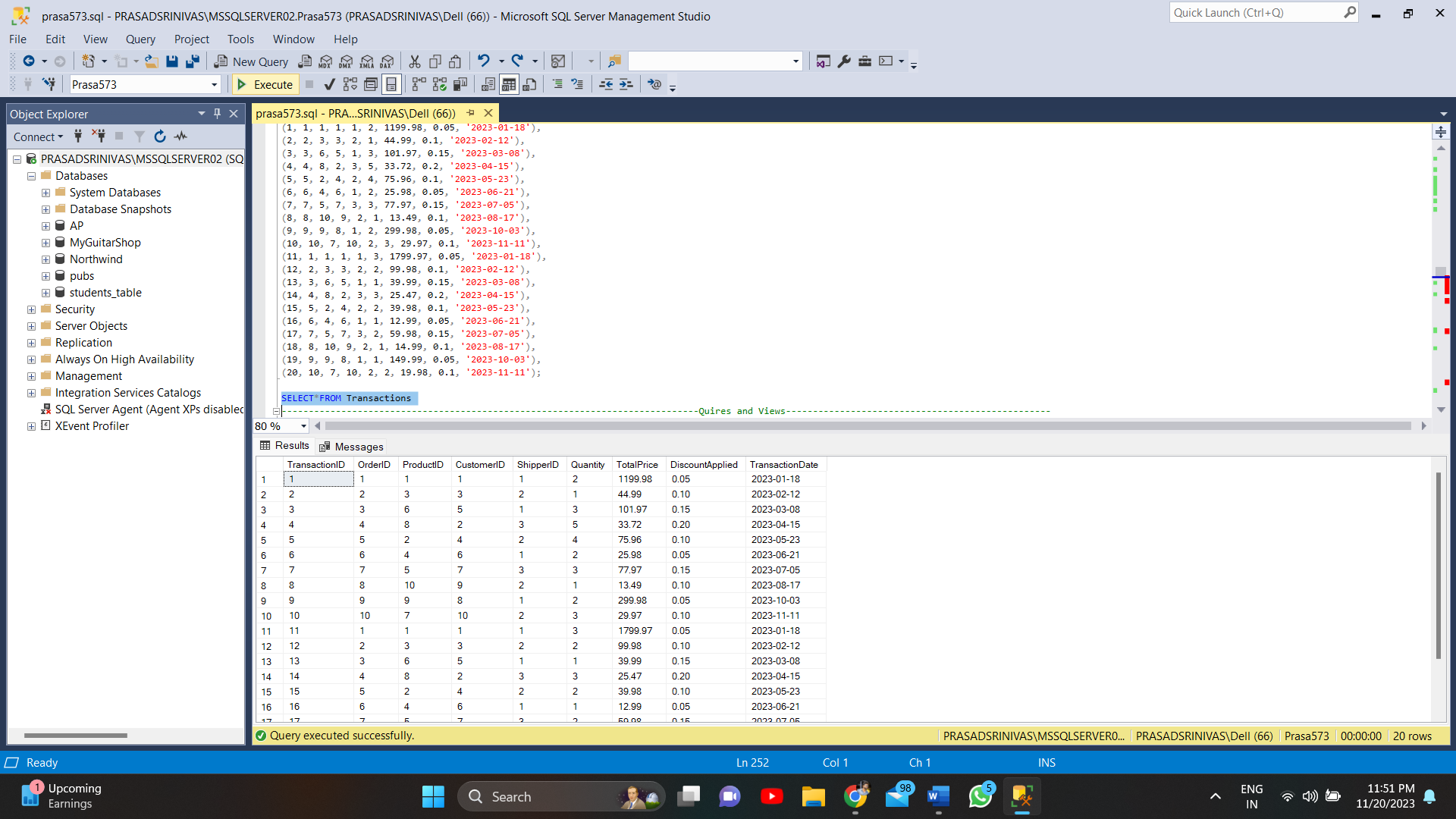
(17, 7, 5, 7, 3, 2, 59.98, 0.15, '2023-07-05'),

(18, 8, 10, 9, 2, 1, 14.99, 0.1, '2023-08-17'),

(19, 9, 9, 8, 1, 1, 149.99, 0.05, '2023-10-03'),

(20, 10, 7, 10, 2, 2, 19.98, 0.1, '2023-11-11');

SELECT\*FROM Transactions



**Optimizing Insights: Converting Queries into Views**

-- Creates a view 'TopSellers' to display the top 5 selling products in 2023

CREATE VIEW TopSellers AS

SELECT TOP 5 ProductID, SUM(Quantity) AS TotalSold

FROM Transactions

WHERE Year(TransactionDate) = 2023

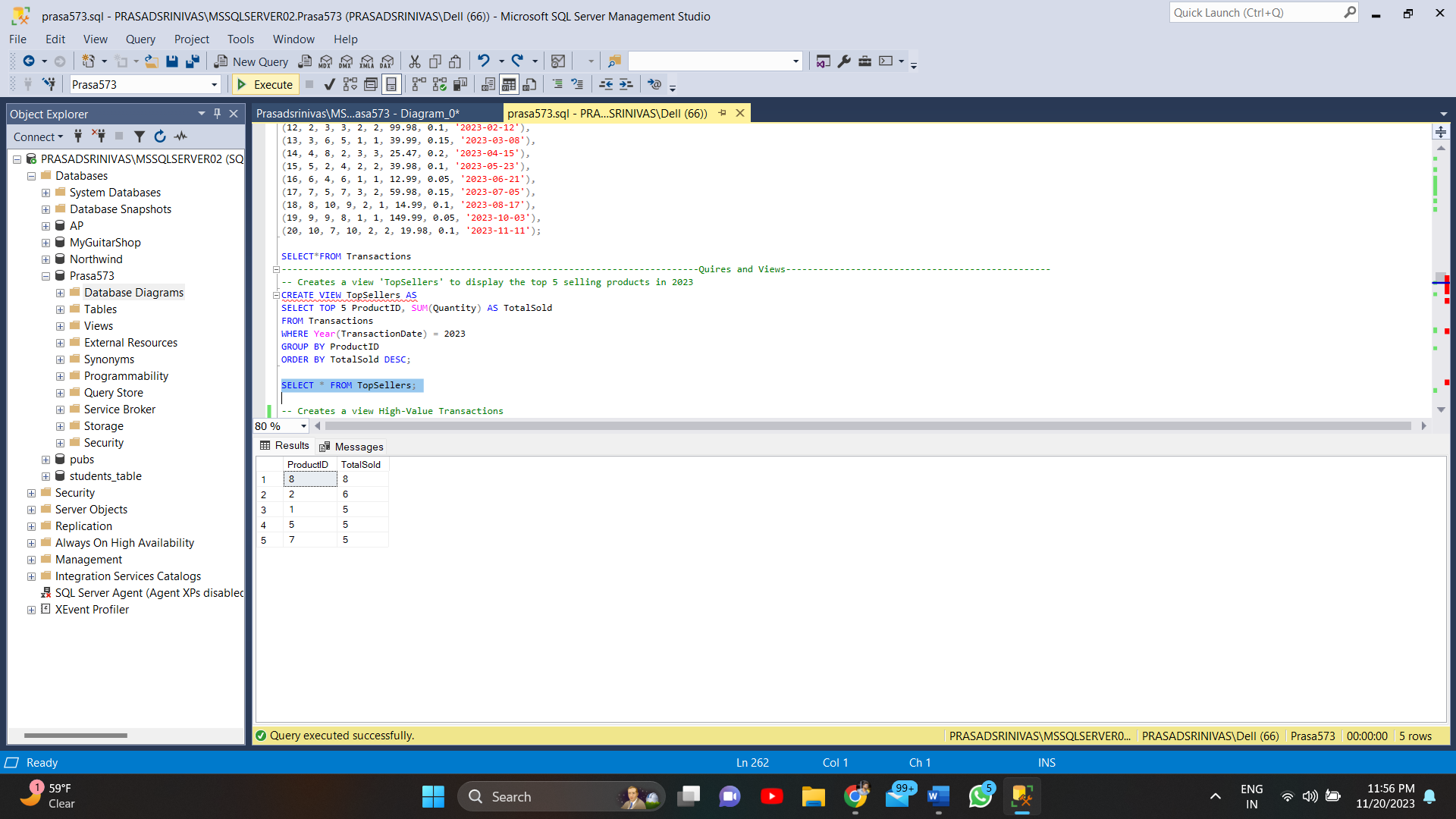
GROUP BY ProductID

ORDER BY TotalSold DESC;

SELECT \* FROM TopSellers;

-- Drop the view TopSellers

DROP VIEW TopSellers;



Developing a Top Sellers View would be incredibly beneficial, for business analysis and decision making to create a view that showcases the 5 selling products in 2023 (TopSellers). This view offers an overview of which products are performing well in terms of sales. This valuable information can guide inventory management, marketing strategies and decisions related to restocking products. Understanding which items are the sellers can help focus efforts or adjust pricing strategies to maximize the success of popular products.

-- Creates a view High-Value Transactions

CREATE VIEW HighValueTransactions AS

SELECT \*

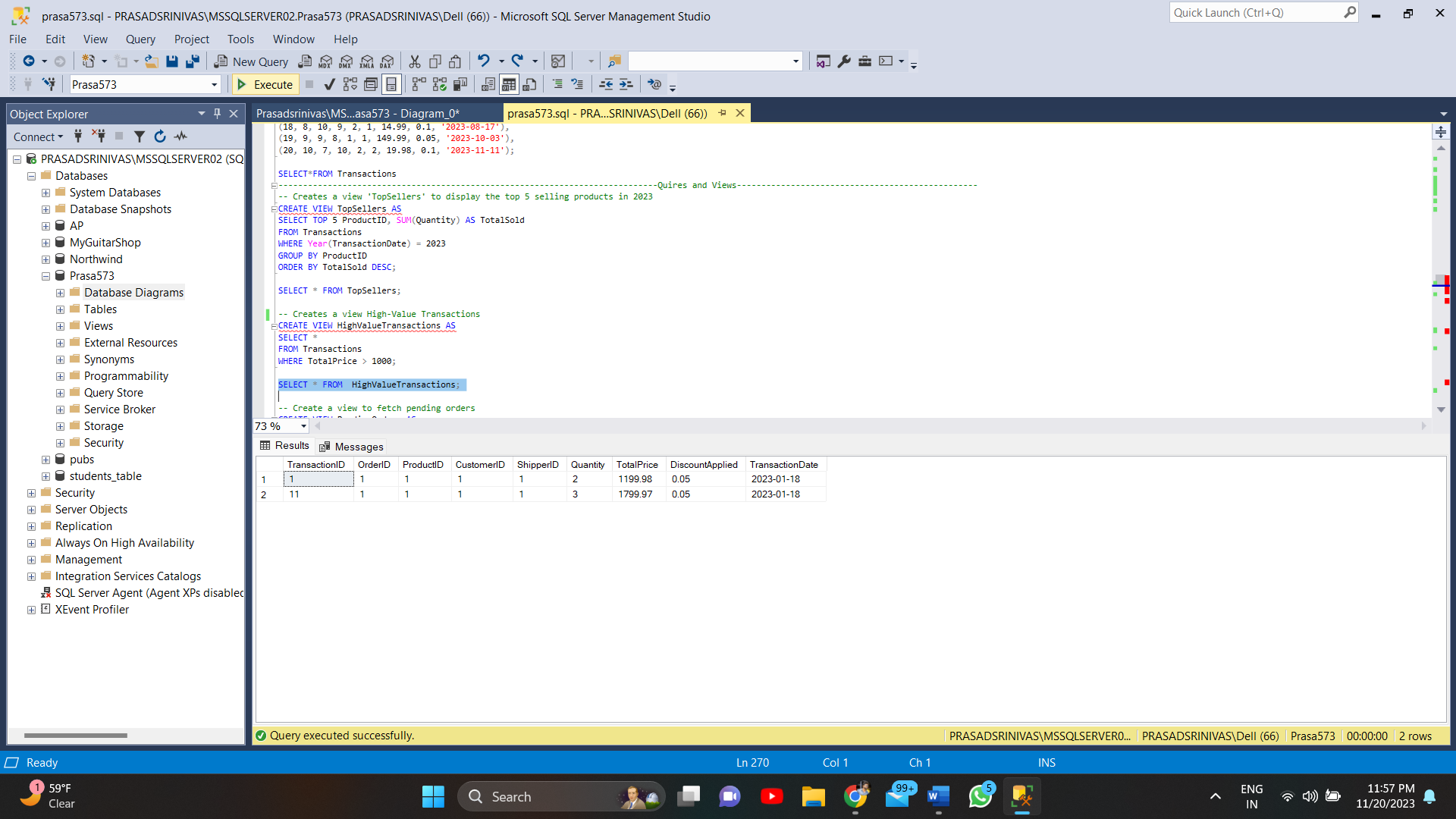
FROM Transactions

WHERE TotalPrice > 1000;

SELECT \* FROM HighValueTransactions;

-- Drop the view HighValueTransactions

DROP VIEW HighValueTransactions;



Creating the HighValueTransactions view which filters transactions with a price exceeding $1000 provides insights into significant revenue generating sales. This view proves useful for reporting identifying customers or products that contribute to high value sales and analyzing trends in expensive purchases. It helps gain an understanding of the segment of transactions that significantly impact business revenue.

-- Create a view to fetch pending orders

CREATE VIEW PendingOrders AS

SELECT OrderID, OrderDate, ShippedDate

FROM Orders

WHERE ShippedDate IS NULL;

-- Update ShippedDate for a specific OrderID

UPDATE Orders

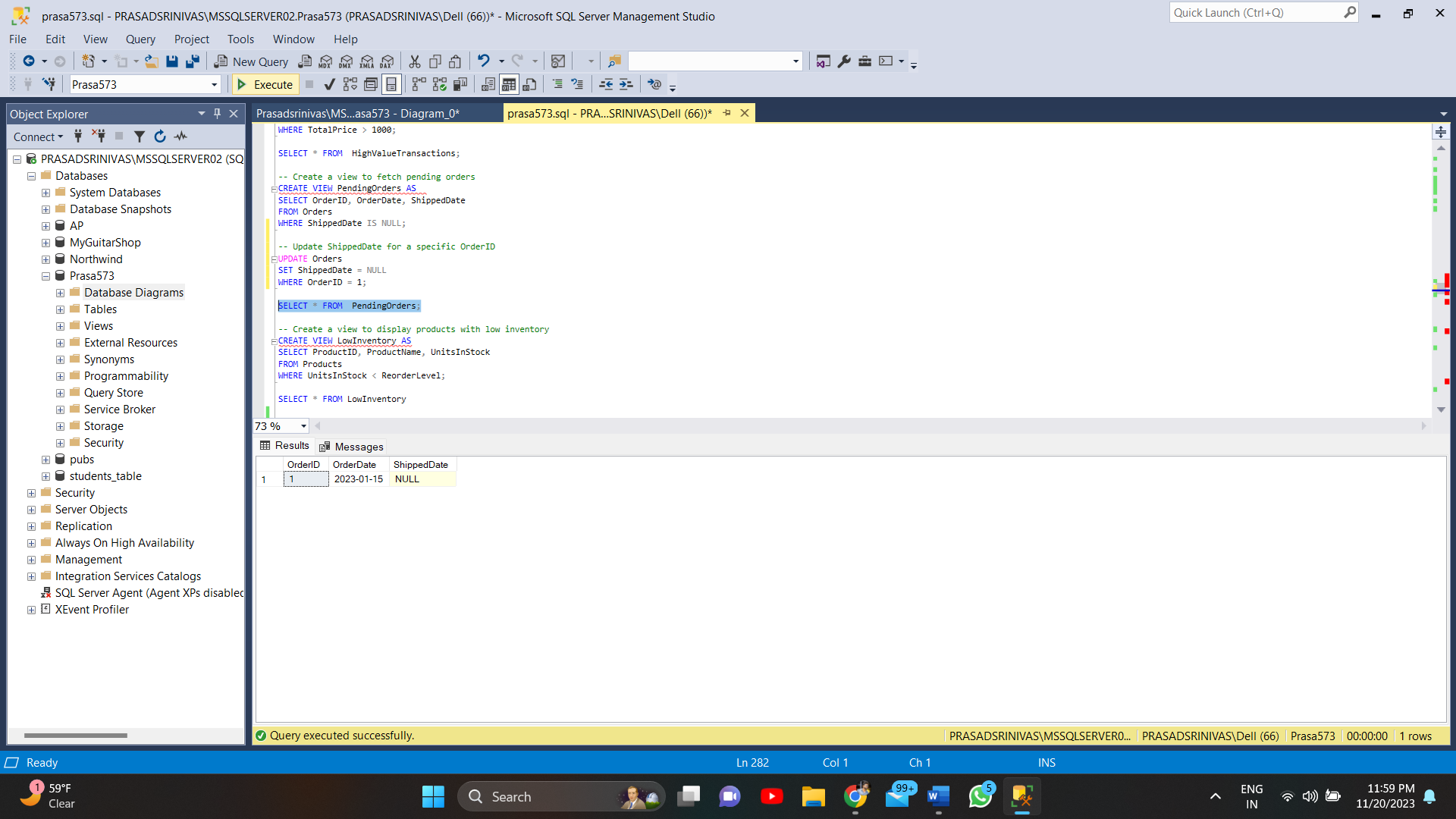
SET ShippedDate = NULL

WHERE OrderID = 1;

SELECT \* FROM PendingOrders;

-- Drop the view PendingOrders

DROP VIEW PendingOrders;



The PendingOrders view is vital for logistics and customer service as it displays orders that haven't been shipped yet. It aids in monitoring pending deliveries, identifying delays and ensuring shipment processing. This view assists, in keeping track of orders requiring attention—an aspect of maintaining customer satisfaction by providing updates on order statuses.

These different views provide insights, into subsets of data that are relevant to various aspects of running a business. They offer access to information helping with decision making, performance analysis and operational management.

-- Create a view to display products with low inventory

CREATE VIEW LowInventory AS

SELECT ProductID, ProductName, UnitsInStock

FROM Products

WHERE UnitsInStock < ReorderLevel;

-- Update UnitsInStock in the Products table

UPDATE Products

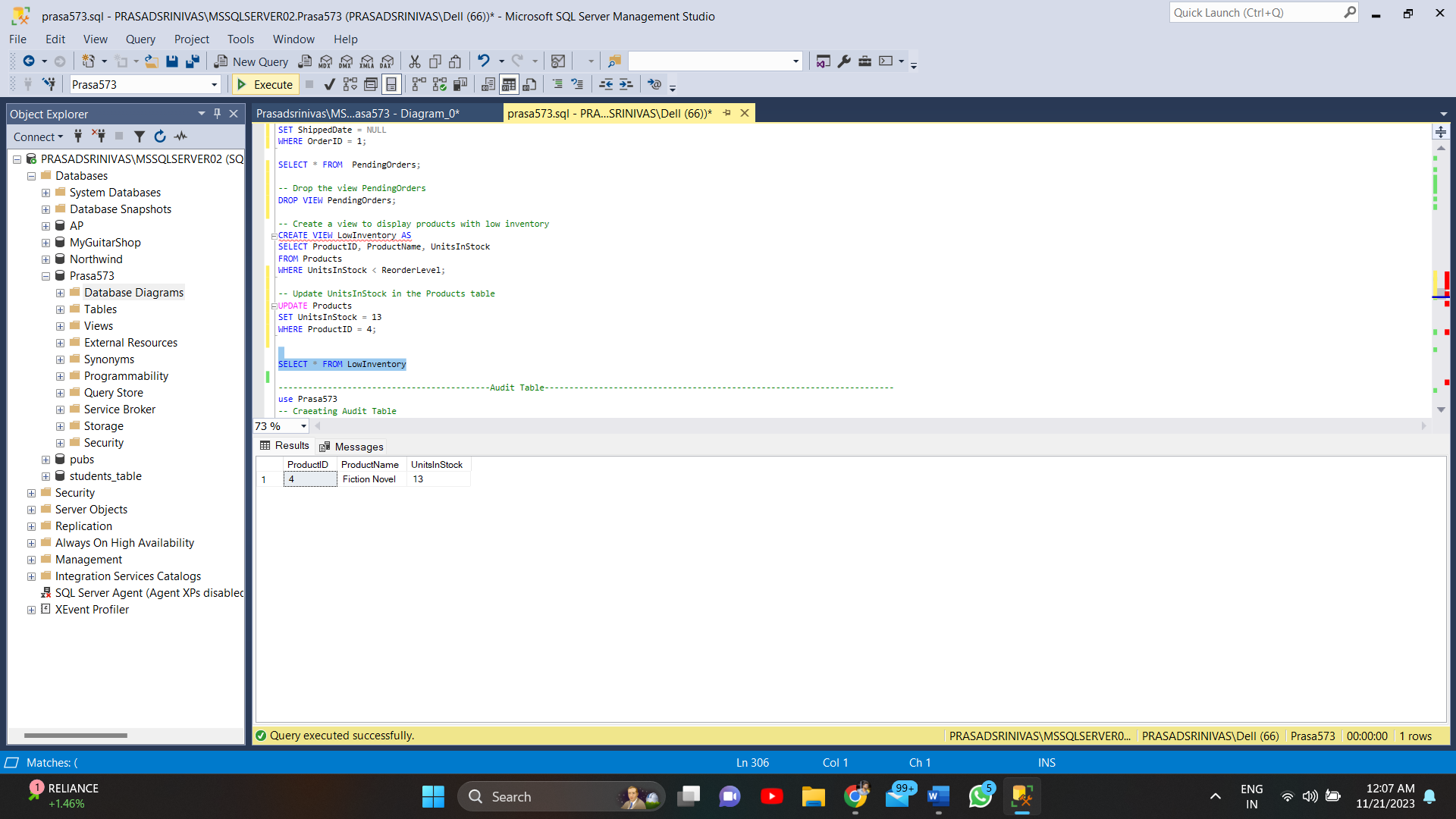
SET UnitsInStock = 13

WHERE ProductID = 4;

SELECT \* FROM LowInventory

-- Drop the view LowInventory

DROP VIEW LowInventory;



Creating a view called "LowInventory" that displays products with stock levels is incredibly beneficial for inventory management and proactive restocking strategies. This view allows immediate visibility into items that have availability potentially preventing situations where stock runs out or delays in fulfilling orders occur. By identifying products that're below their reorder levels it enables reordering or production to ensure that popular items don't unexpectedly go out of stock. This approach helps maintain inventory levels and meets customer demand seamlessly without any interruptions. Additionally monitoring inventory items can also help identify trends or patterns in sales that may require adjustments, in stocking strategies or promotions.

**AUDIT TABLE:**

use Prasa573

-- Craeating Audit Table

CREATE TABLE CategoriesAudit (

AuditID int IDENTITY(1,1) PRIMARY KEY,

CategoryID int,

CategoryName varchar(100),

Description varchar(255),

ChangeType varchar(10),

ChangedOn datetime DEFAULT GETDATE()

);

1. Insert trigger:

-- Craeating Insert Trigger

CREATE TRIGGER tr\_Categories\_Insert

ON Categories

AFTER INSERT

AS

BEGIN

INSERT INTO CategoriesAudit

(CategoryID, CategoryName, Description, ChangeType)

SELECT

i.CategoryID, i.CategoryName, i.Description, 'INSERT'

FROM inserted i;

END

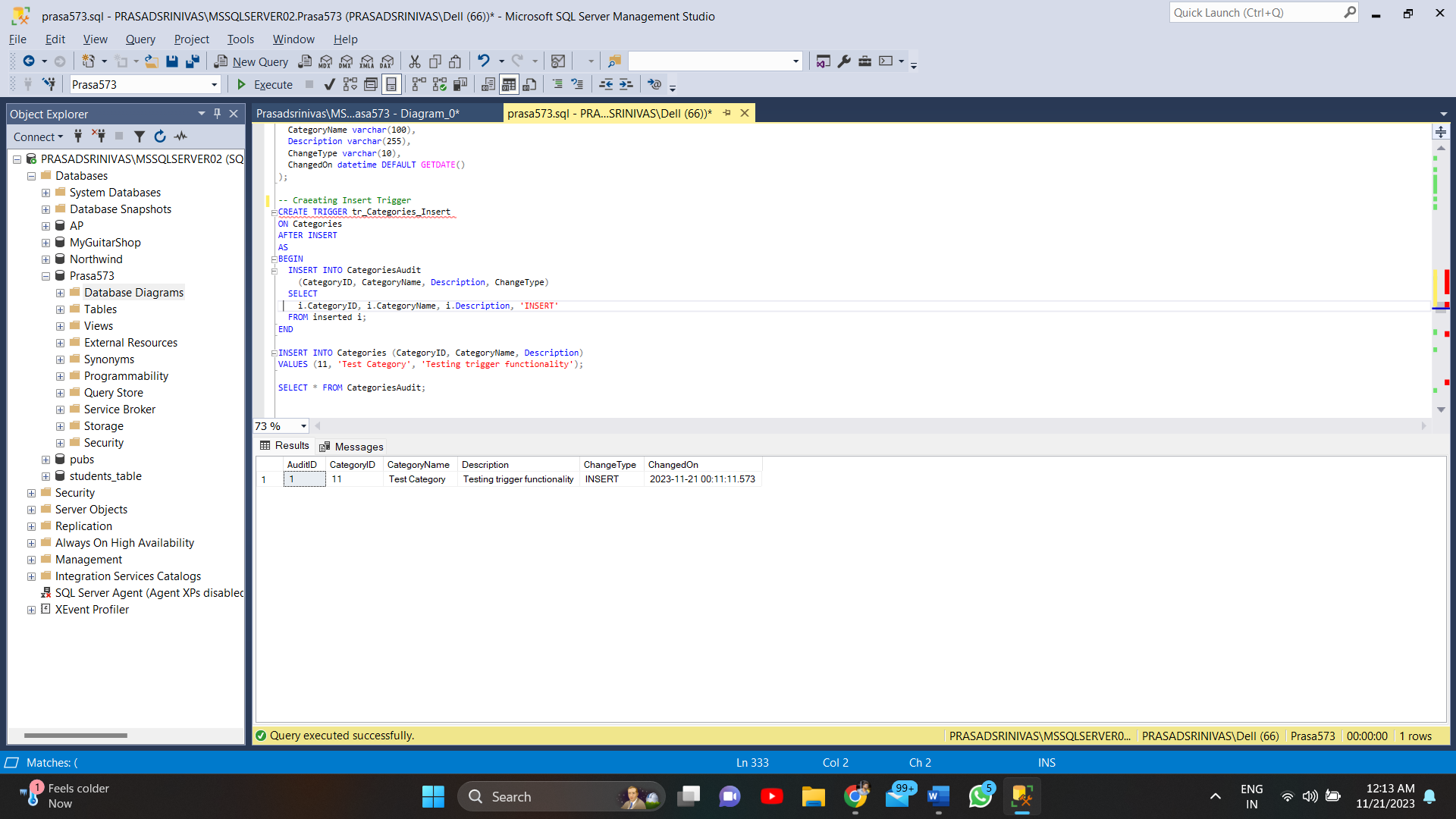
INSERT INTO Categories (CategoryID, CategoryName, Description)

VALUES (11, 'Test Category', 'Testing trigger functionality');

SELECT \* FROM CategoriesAudit;

-- Drop the trigger tr\_Categories\_Insert

DROP TRIGGER tr\_Categories\_Insert



1. Update Trigger:

-- Craeating Update Tigger

CREATE TRIGGER tr\_Categories\_Update

ON Categories

AFTER UPDATE

AS

BEGIN

INSERT INTO CategoriesAudit

(CategoryID, CategoryName, Description, ChangeType)

SELECT

i.CategoryID, i.CategoryName, i.Description, 'UPDATE'

FROM inserted i;

END

-- Update a category in the Categories table

UPDATE Categories

SET Description = 'New Description'

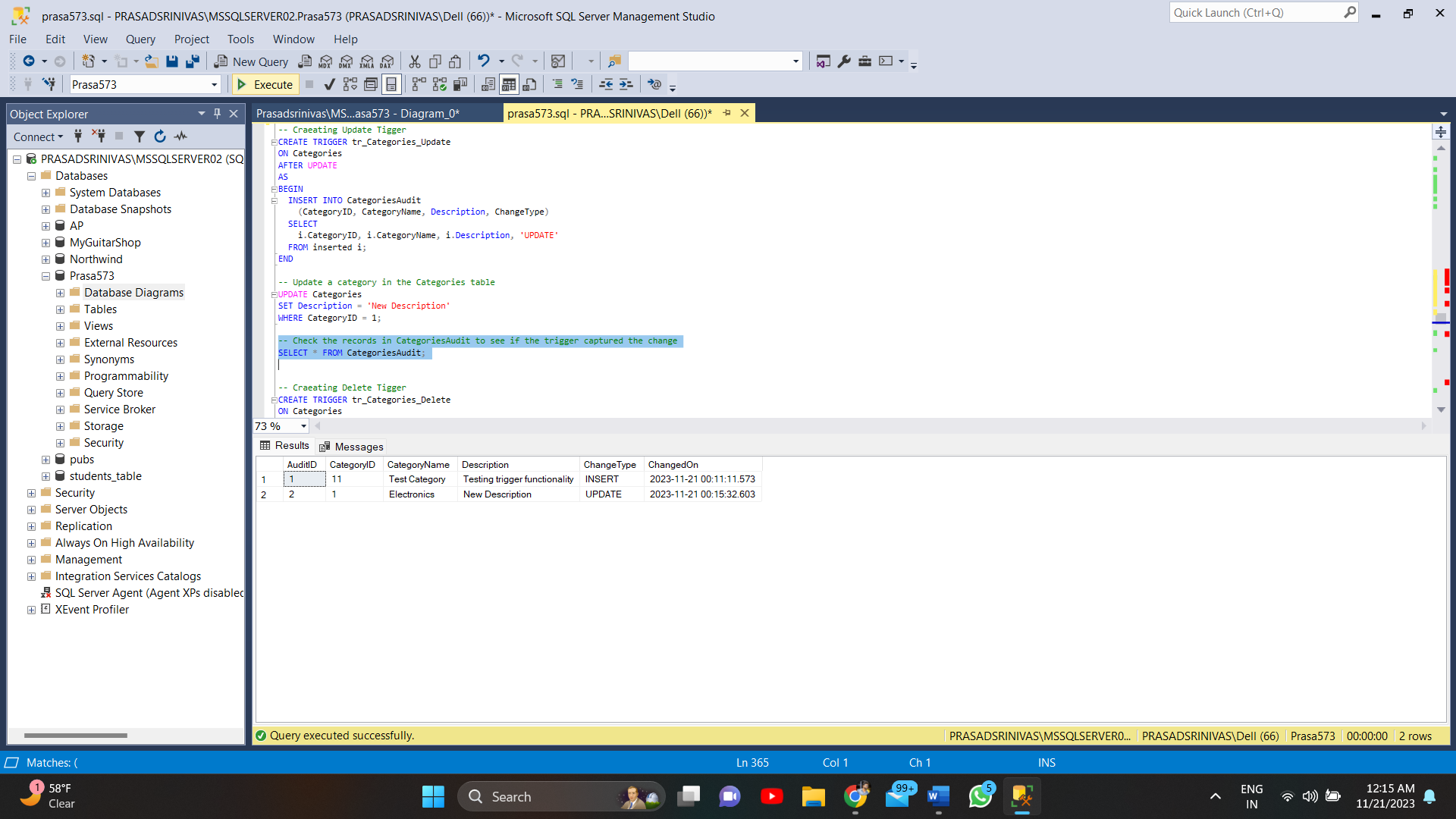
WHERE CategoryID = 1;

-- Check the records in CategoriesAudit to see if the trigger captured the change

SELECT \* FROM CategoriesAudit;

-- Drop the trigger tr\_Categories\_Update

DROP TRIGGER tr\_Categories\_Update;



1. Delete Trigger :

-- Creating Delete Tigger

CREATE TRIGGER tr\_Categories\_Delete

ON Categories

AFTER DELETE

AS

BEGIN

INSERT INTO CategoriesAudit

(CategoryID, CategoryName, Description, ChangeType)

SELECT

d.CategoryID, d.CategoryName, d.Description, 'DELETE'

FROM deleted d;

END

-- create a delete operation on Categories table

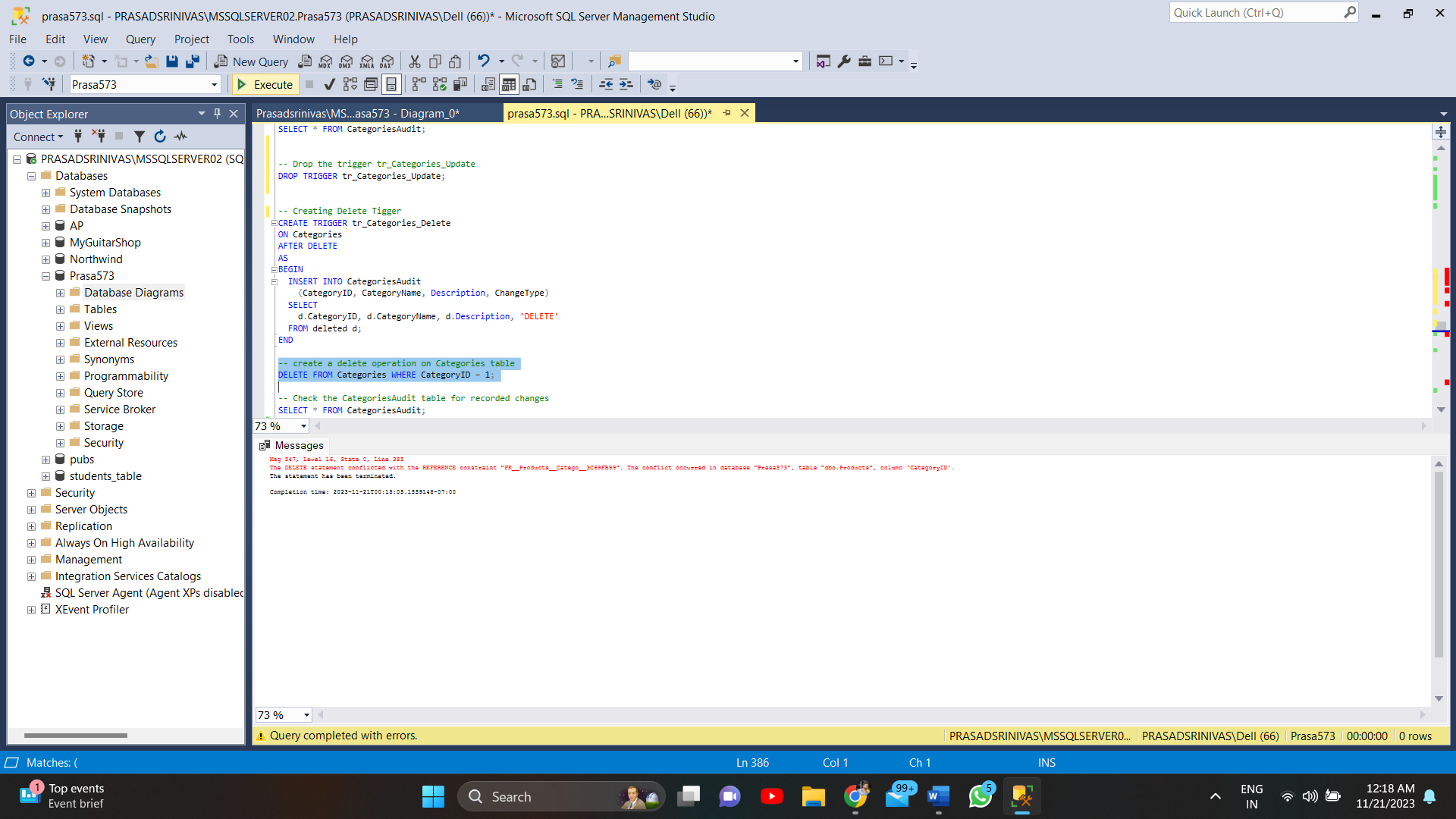
DELETE FROM Categories WHERE CategoryID = 1;

-- Check the CategoriesAudit table for recorded changes

SELECT \* FROM CategoriesAudit;

-- Drop the trigger tr\_Categories\_Delete

DROP TRIGGER tr\_Categories\_Delete;



Unable to implement DELETE trigger: The interdependent relationships between tables (Categories, Products, Orders, etc.) prevent the use of a delete trigger on the Categories table.

**Stored Procedure:**

CREATE PROCEDURE GetTopCustomers

AS

BEGIN

SELECT TOP 5 c.CustomerID, c.CompanyName, SUM(t.TotalPrice) AS TotalRevenue

FROM Customers c

JOIN Transactions t ON c.CustomerID = t.CustomerID

GROUP BY c.CustomerID, c.CompanyName

ORDER BY TotalRevenue DESC

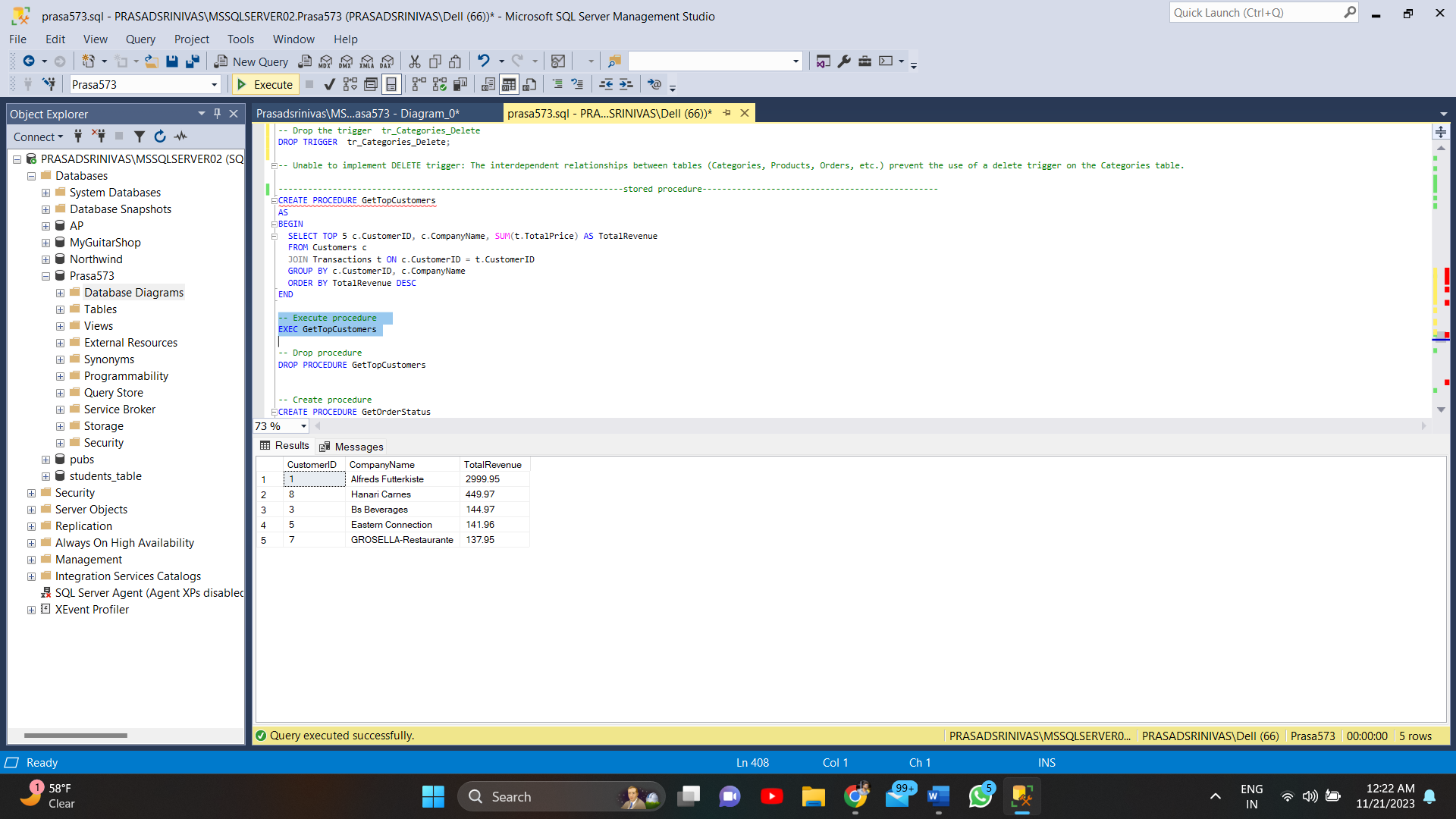
END

-- Execute procedure

EXEC GetTopCustomers

-- Drop procedure

DROP PROCEDURE GetTopCustomers



-- Create procedure

CREATE PROCEDURE GetOrderStatus

@OrderID int

AS

BEGIN

SELECT

o.OrderID,

CASE

WHEN ShippedDate IS NULL THEN 'Pending'

ELSE 'Shipped'

END AS OrderStatus

FROM Orders o

WHERE o.OrderID = @OrderID

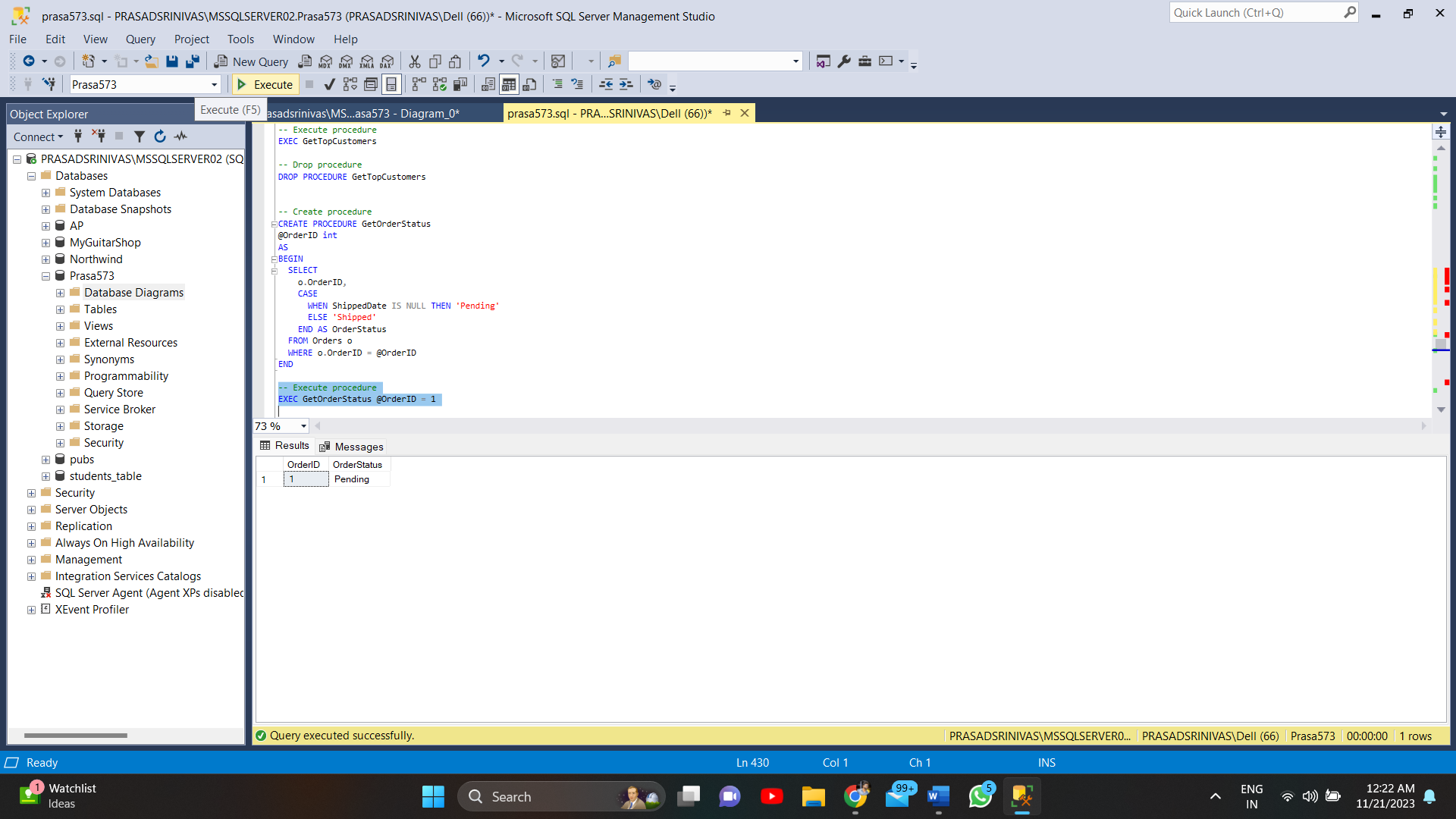
END

-- Execute procedure

EXEC GetOrderStatus @OrderID = 1

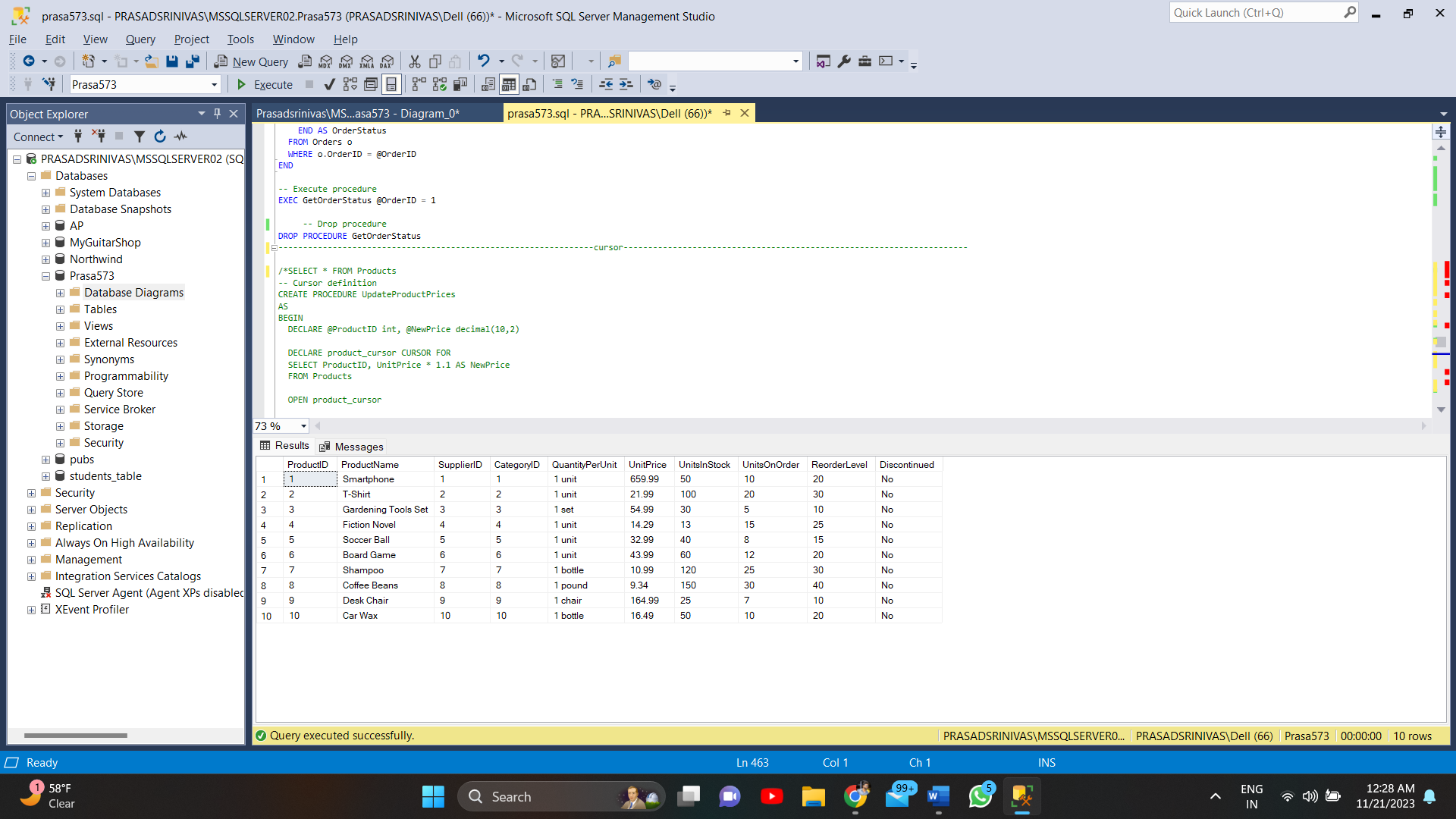
-- Drop procedure

DROP PROCEDURE GetOrderStatus



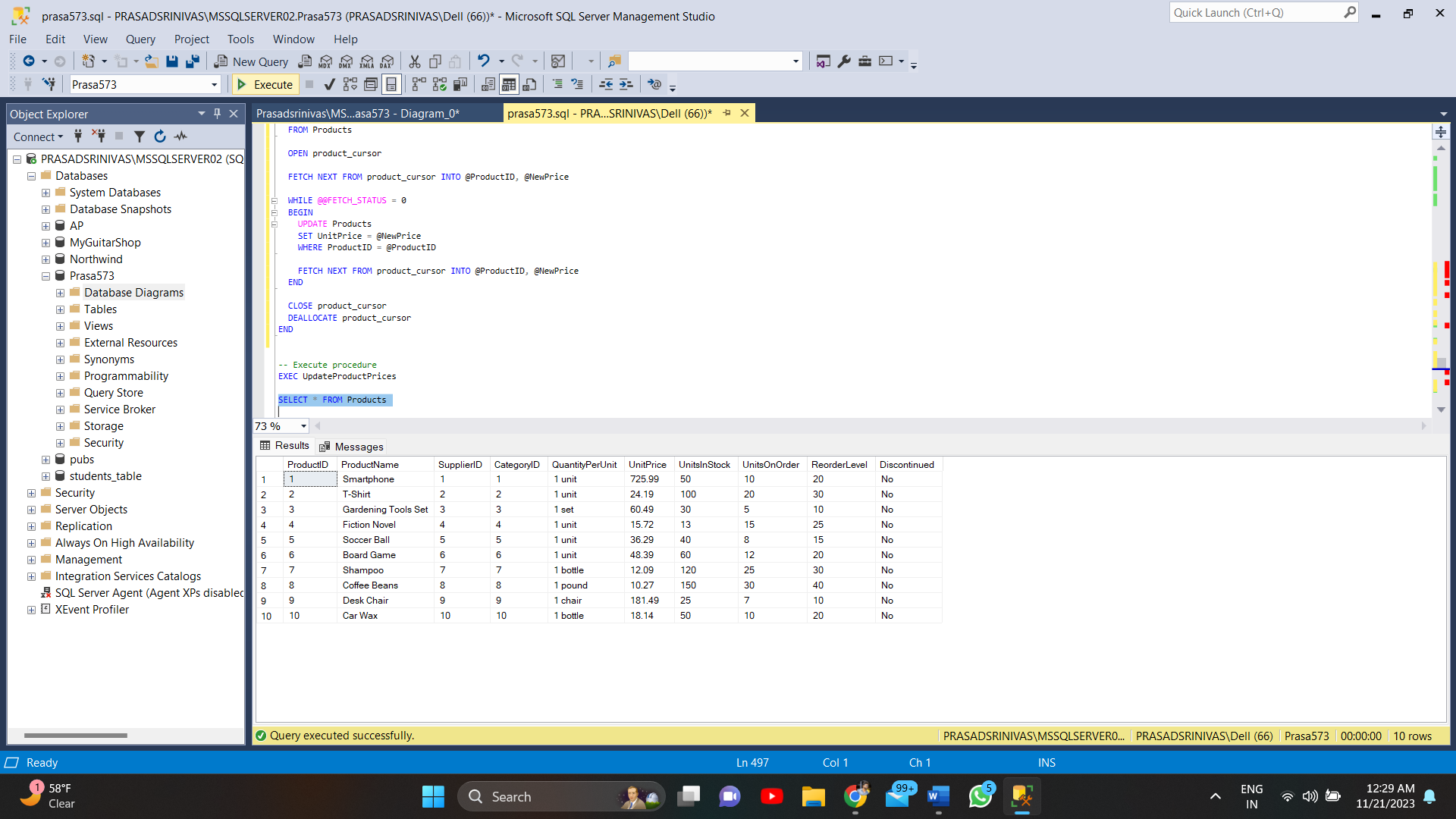
**Cursor:**

**Before:**





**AFTER:**



SELECT \* FROM Products

GO

CREATE PROCEDURE UpdateProductPrices

AS

BEGIN

DECLARE @ProductID int, @NewPrice decimal(10,2)

DECLARE product\_cursor CURSOR FOR

SELECT ProductID, UnitPrice \* 1.1 AS NewPrice

FROM Products

OPEN product\_cursor

FETCH NEXT FROM product\_cursor INTO @ProductID, @NewPrice

WHILE @@FETCH\_STATUS = 0

BEGIN

UPDATE Products

SET UnitPrice = @NewPrice

WHERE ProductID = @ProductID

FETCH NEXT FROM product\_cursor INTO @ProductID, @NewPrice

END

CLOSE product\_cursor

DEALLOCATE product\_cursor

END

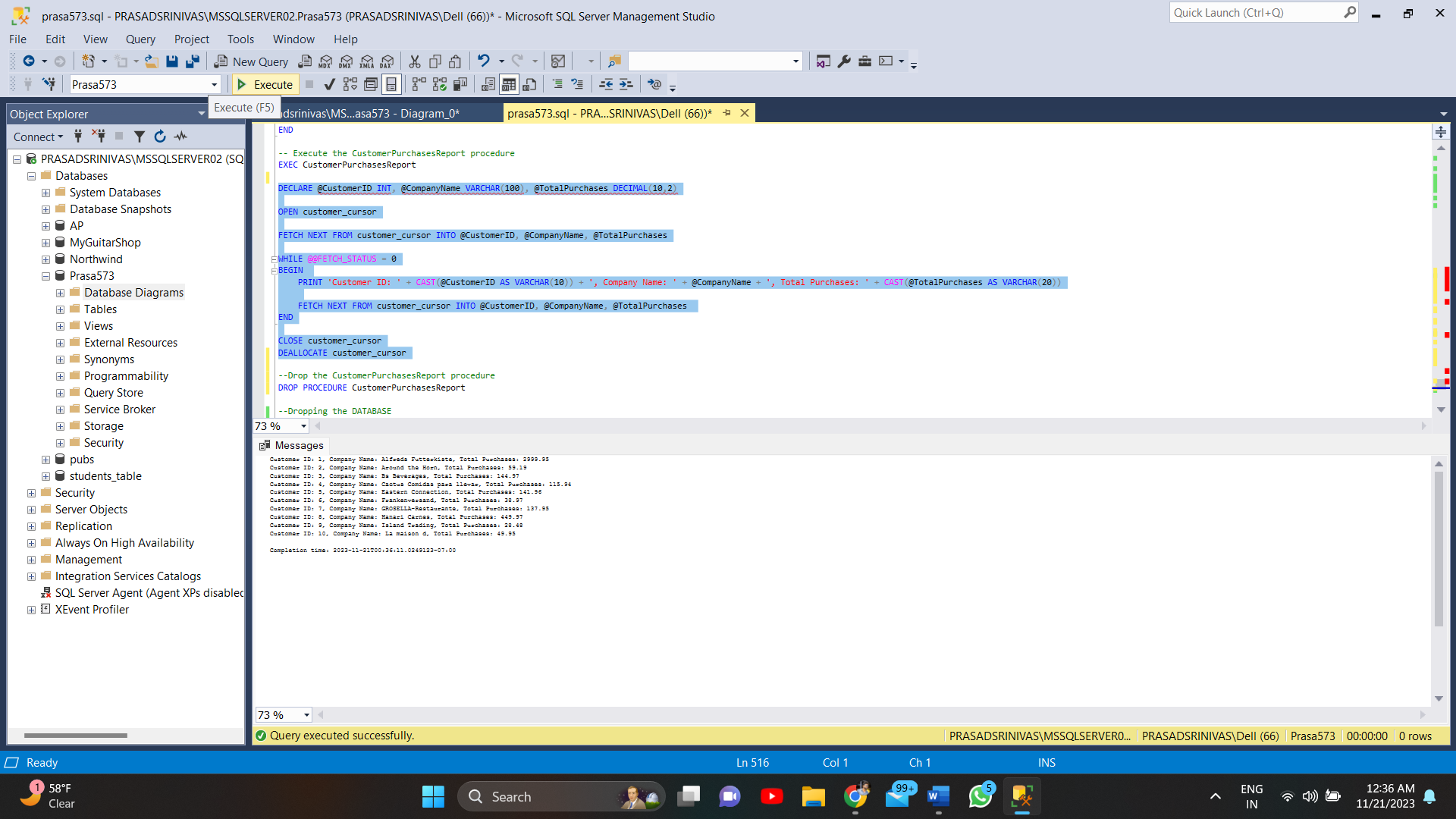
-- Execute procedure

EXEC UpdateProductPrices

SELECT \* FROM Products

-- Drop procedure

DROP PROCEDURE UpdateProductPrices



-- Creating proceudre customerPurchaseReport

CREATE PROCEDURE CustomerPurchasesReport

AS

BEGIN

DECLARE @CustomerID int, @CompanyName varchar(100), @TotalPurchases decimal(10,2)

-- Check if the cursor exists in the current session

IF CURSOR\_STATUS('local', 'customer\_cursor') >= 0

BEGIN

-- If the cursor exists, deallocate it

DEALLOCATE customer\_cursor;

END

-- Define the cursor

DECLARE customer\_cursor CURSOR FOR

SELECT c.CustomerID, c.CompanyName, SUM(t.TotalPrice) AS TotalPurchases

FROM Customers c

JOIN Transactions t ON c.CustomerID = t.CustomerID

GROUP BY c.CustomerID, c.CompanyName

END

-- Execute the CustomerPurchasesReport procedure

EXEC CustomerPurchasesReport

DECLARE @CustomerID INT, @CompanyName VARCHAR(100), @TotalPurchases DECIMAL(10,2)

OPEN customer\_cursor

FETCH NEXT FROM customer\_cursor INTO @CustomerID, @CompanyName, @TotalPurchases

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT 'Customer ID: ' + CAST(@CustomerID AS VARCHAR(10)) + ', Company Name: ' + @CompanyName + ', Total Purchases: ' + CAST(@TotalPurchases AS VARCHAR(20))

FETCH NEXT FROM customer\_cursor INTO @CustomerID, @CompanyName, @TotalPurchases

END

CLOSE customer\_cursor

DEALLOCATE customer\_cursor

--Drop the CustomerPurchasesReport procedure

DROP PROCEDURE CustomerPurchasesReport