A TARGET COMPETITOR

Prof. Ercanli

Ercanli

Seyed Alireza Zarrin Mehr

Syracuse University

Introduction to Database Management Systems

Abstract:

in this project, we designed a database for a chain store that offers online and in-store sales. This design is based on the following business model:

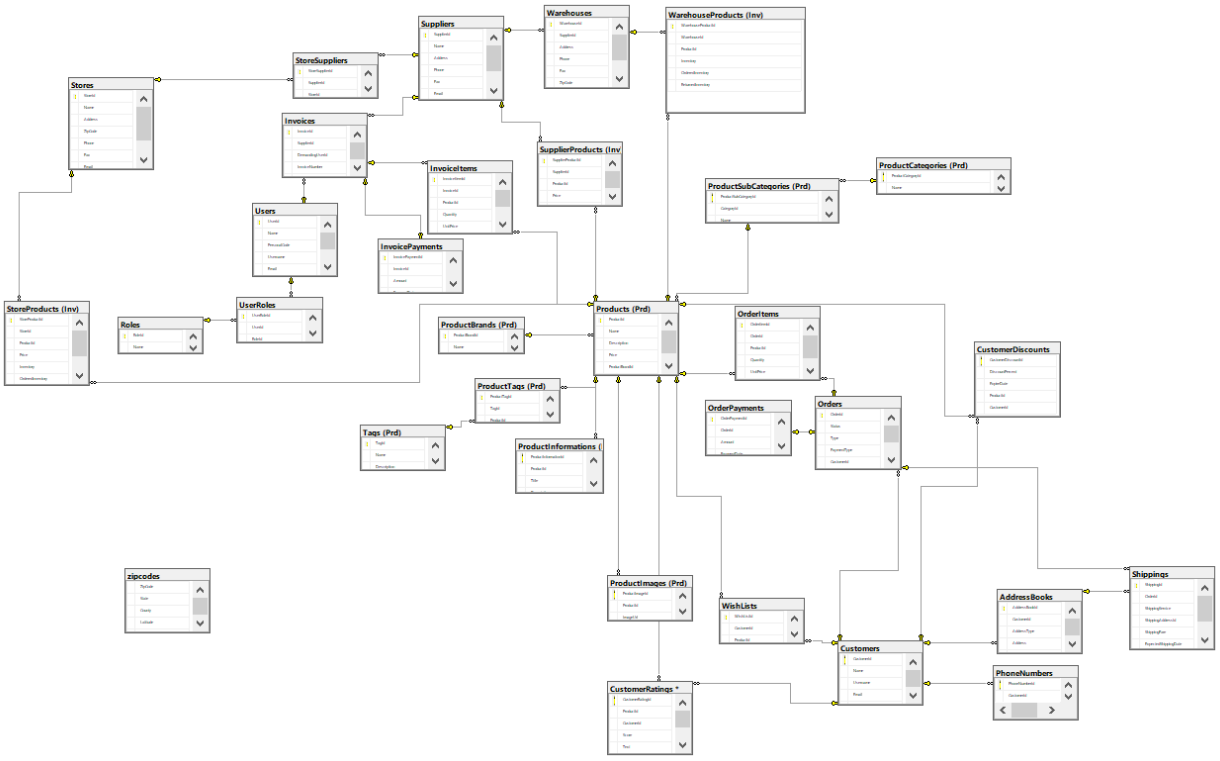
- Stores can place orders to deliver products from warehouses to stores for in-store purchases.
- Each online order can contain multiple products from one or more store warehouses and online sales will get supplied from the nearest store's product inventory.
- Multiple warehouses may have the same product.
- Each online order has a single shipping address.
- Suppliers can have multiple products in various warehouses.

Design Introduction:

The database has got tested with real data and is ready to be implemented in any chain store business. Various real-world scenarios are considered which will be introduced later in the testing section of this report. Following are some of the features of the design:

- To decrease the shipping costs, the system finds the nearest store location to the customers and the order will get shipped from there.
- If the first store does not have enough inventory to supply the order, the system will find another store (the second store) to supply the rest of the order items.
- Product quantity in stores and warehouses gets updated based on transactions.

E/R diagram:



Design considerations, table relationship

Stores-store products: one to many, each store has multiple products.
Store suppliers-stores: many to many
Store-store product: one to many
Product brands-Products: one to many
Products-product tags: many to many

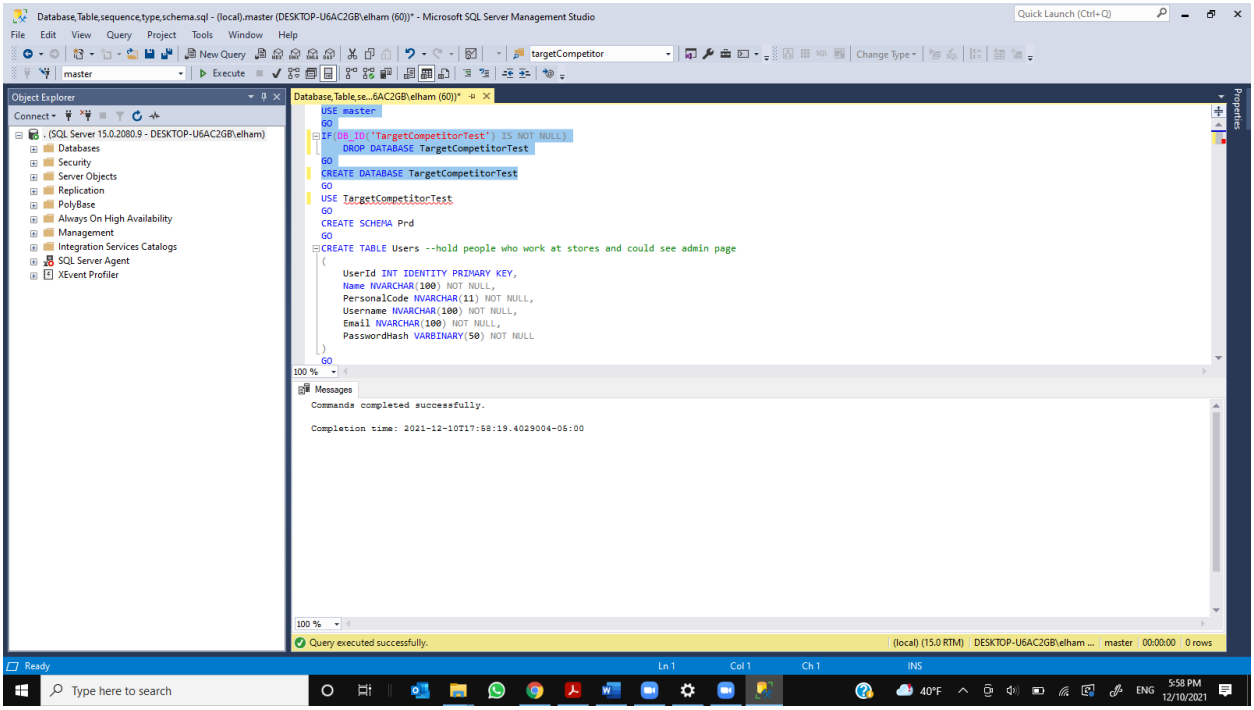
Tags-product tags: one to many
Products-product information: one to many, each product may have multiple information (e.g., a laptop has information for ram and hard, etc.)
Product-product images: one to many, each product may have multiple images.
Product-customers: many to many with the wish list linking table
Product-customers: many to many with the customer ratings linking table
Customer-phone numbers: one to many, each customer may have various phones of different types.
Customer-address books: many to many, each customer may have multiple addresses (billing, shipping, etc.)
Order-shipping: each order has one shipping and each shipping is for one order
Shipping-address books: one to many, there can be multiple shipping that goes to the same address.
Customers-customer discount: one to one, each customer can have a certain amount of coupon on a product
Customer-products: many to many with customer discount linking table, each product may have a different amount of discount for different customers, and each customer may get different discounts on different products.
Product- product subcategories: each category may have different products.
Product subcategories- product categories: each category has various subcategories.
Products-warehouses: many to many, with the warehouse product linking table.
Supplier product-products: one to one
Supplier- supplier products: one to many: each supplier has multiple products.

- Warehouses deliver the products to the stores and when an online order is placed the order will get shipped from the stores to decrease the shipping costs and a fast delivery. Each supplier has various products in different warehouses

Implementation; SQL source codes with comments:

- ✓ All the objects and items in this database has got named using CamelCase naming standard.
- ✓ The design has got normalized into its 3rd normal form

```
USE master
GO
IF(DB_ID('TargetCompetitorTest') IS NOT NULL)
        DROP DATABASE TargetCompetitorTest
GO
CREATE DATABASE TargetCompetitorTest
```

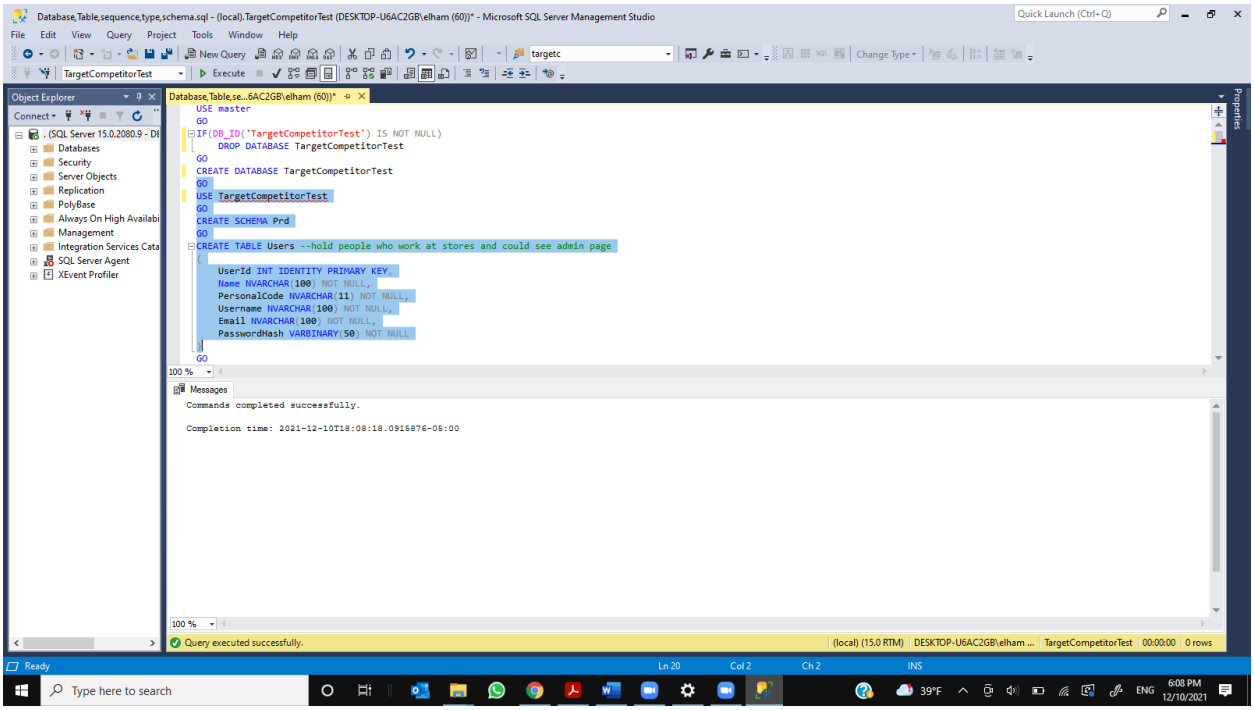


Comment: here we created the database

Tables with constraints

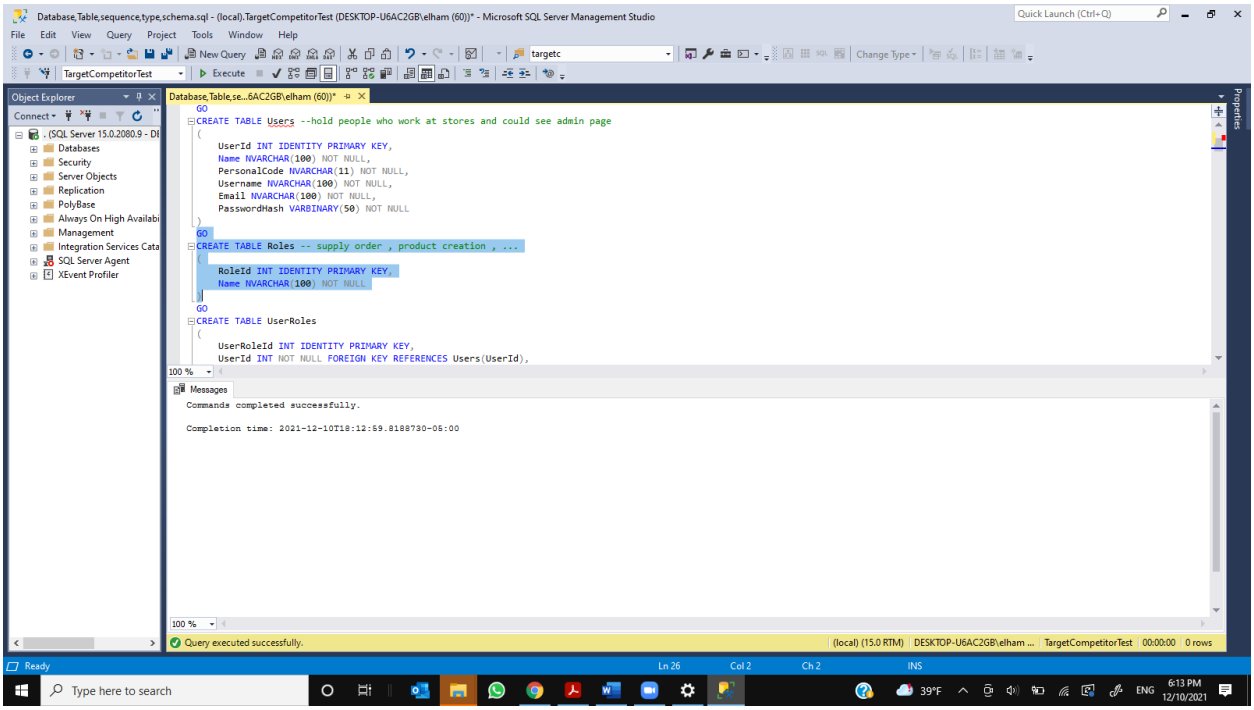
- ✓ Schema and tables have got created with columns, primary keys, foreign keys, proper data types, nullabilities, and necessary relationships and constraints

```
GO
USE TargetCompetitorTest
GO
CREATE SCHEMA Prd
GO
CREATE TABLE Users --hold people who work at stores and could see admin page
(
        UserId INT IDENTITY PRIMARY KEY,
        Name NVARCHAR(100) NOT NULL,
        PersonalCode NVARCHAR(11) NOT NULL,
        Username NVARCHAR(100) NOT NULL,
        Email NVARCHAR(100) NOT NULL,
        PasswordHash VARBINARY(50) NOT NULL
)
```



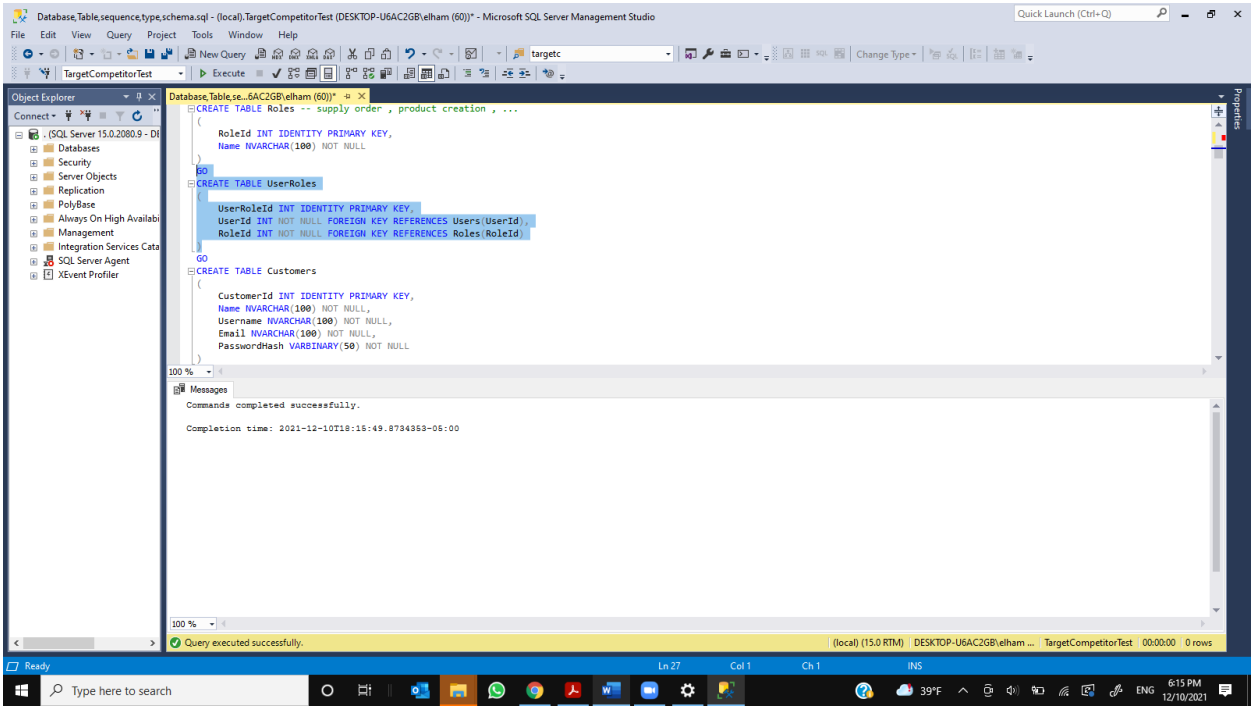
Comment: here we created the created the users table for people who work at stores and could see admin page.

```
GO
CREATE TABLE Roles -- supply order , product creation , ...
(
        RoleId INT IDENTITY PRIMARY KEY,
        Name NVARCHAR(100) NOT NULL
)
```



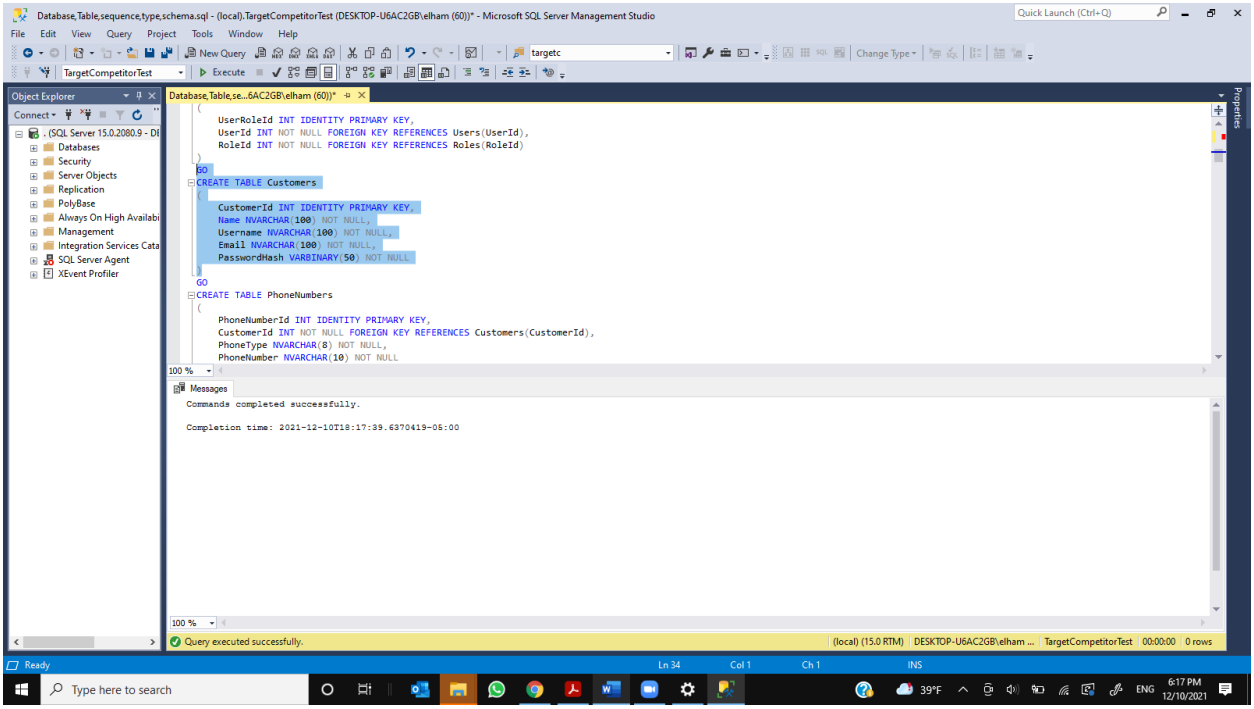
Comment: here we created Roles table.

```
GO
CREATE TABLE UserRoles
(
        UserRoleId INT IDENTITY PRIMARY KEY,
        UserId INT NOT NULL FOREIGN KEY REFERENCES Users(UserId),
        RoleId INT NOT NULL FOREIGN KEY REFERENCES Roles(RoleId)
)
```



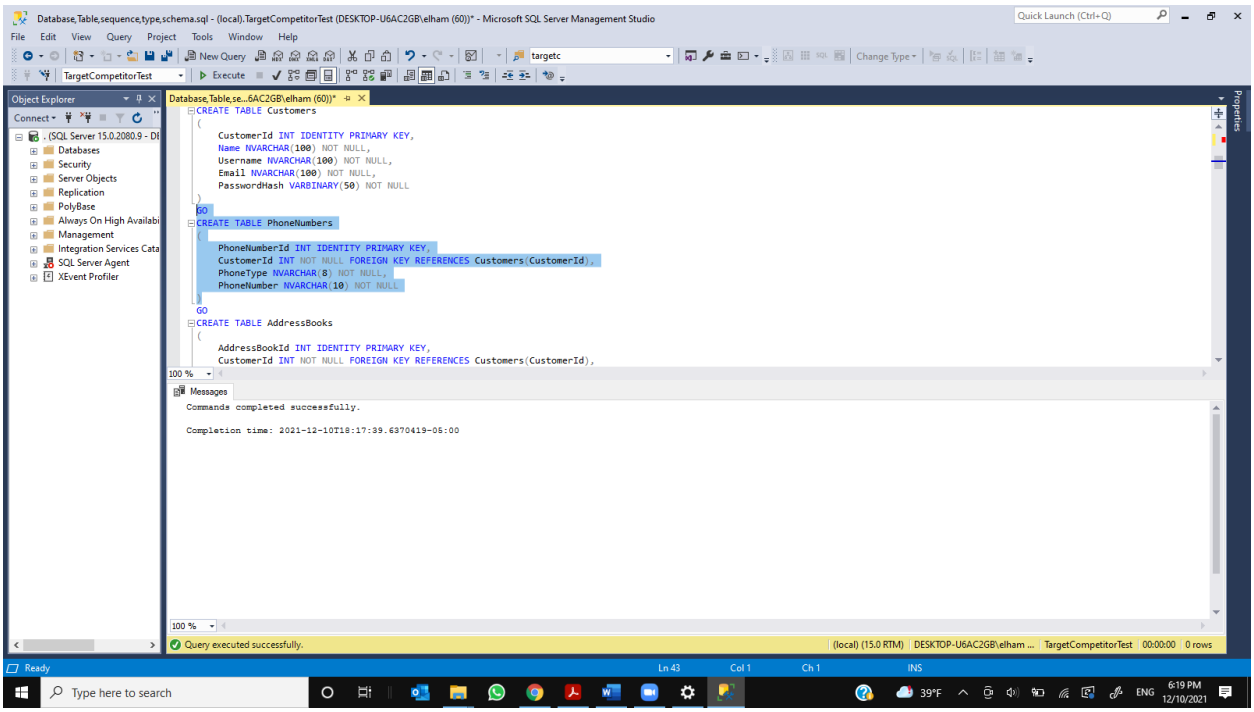
Comment: here we created the user roles table.

```
GO
CREATE TABLE Customers
(
        CustomerId INT IDENTITY PRIMARY KEY,
        Name NVARCHAR(100) NOT NULL,
        Username NVARCHAR(100) NOT NULL,
        Email NVARCHAR(100) NOT NULL,
        PasswordHash VARBINARY(50) NOT NULL
)
```



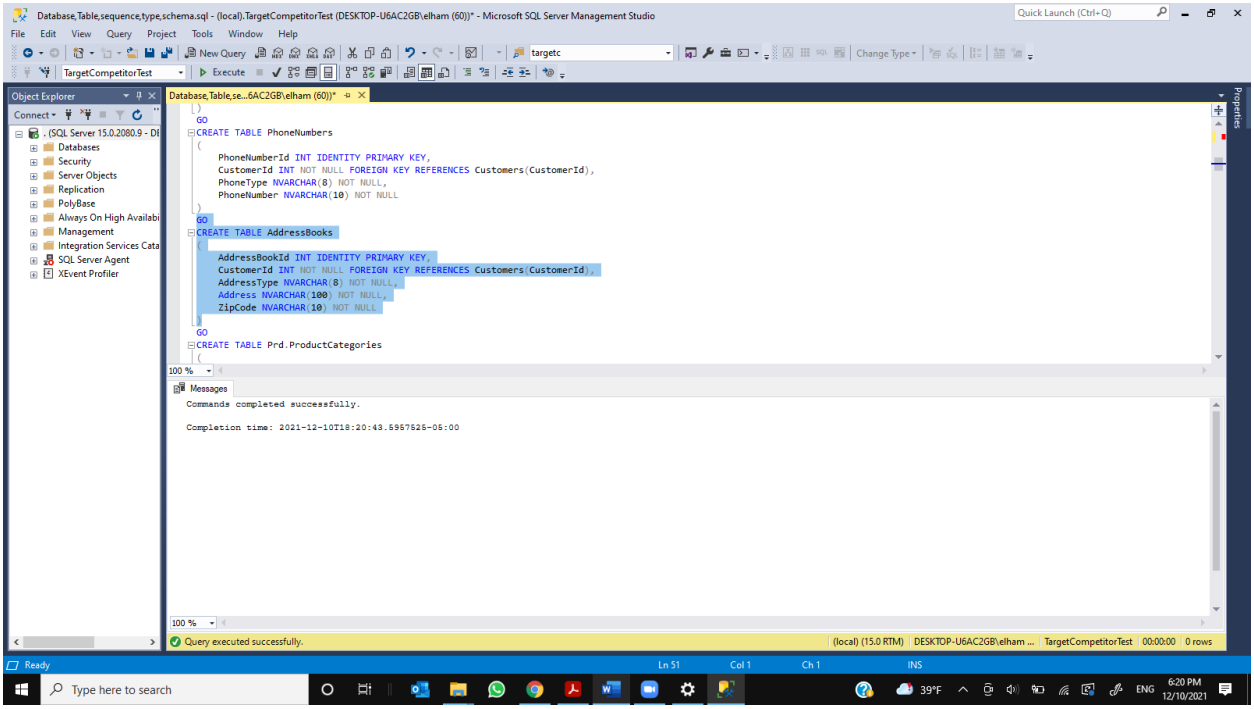
Comment: here we created the customer table.

```
GO
CREATE TABLE PhoneNumbers
(
        PhoneNumberId INT IDENTITY PRIMARY KEY,
        CustomerId INT NOT NULL FOREIGN KEY REFERENCES Customers(CustomerId),
        PhoneType NVARCHAR(8) NOT NULL,
        PhoneNumber NVARCHAR(10) NOT NULL
)
```



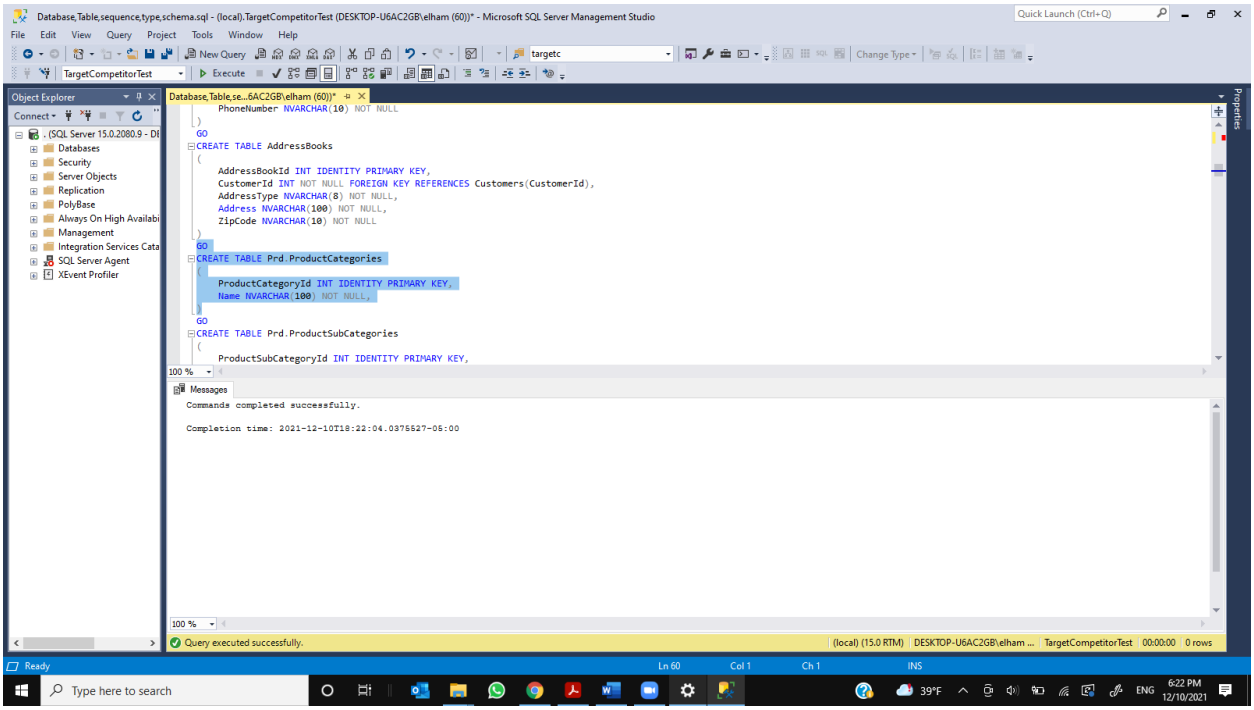
Comment: here we created the phone number table.

```
GO
CREATE TABLE AddressBooks
(
        AddressBookId INT IDENTITY PRIMARY KEY,
        CustomerId INT NOT NULL FOREIGN KEY REFERENCES Customers(CustomerId),
        AddressType NVARCHAR(8) NOT NULL,
        Address NVARCHAR(100) NOT NULL,
        ZipCode NVARCHAR(10) NOT NULL
)
```



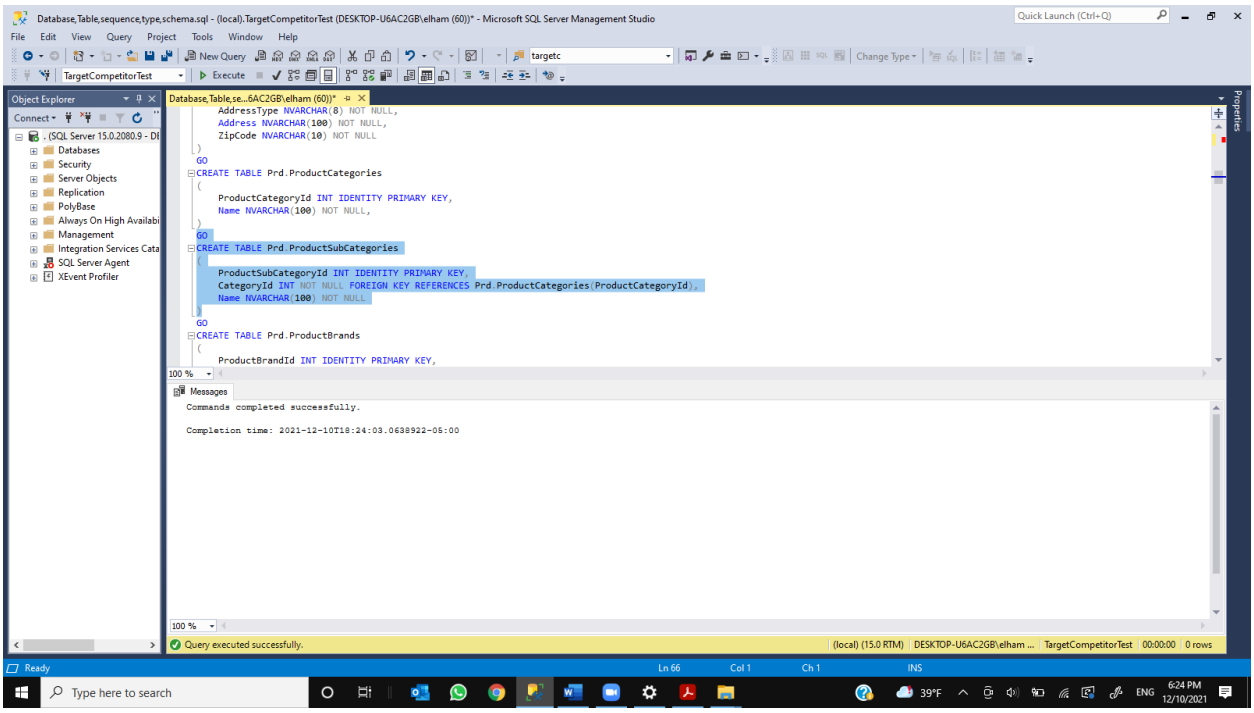
Comment: here we created the address book table.

```
GO
CREATE TABLE Prd.ProductCategories
(
        ProductCategoryId INT IDENTITY PRIMARY KEY,
        Name NVARCHAR(100) NOT NULL,
)
```



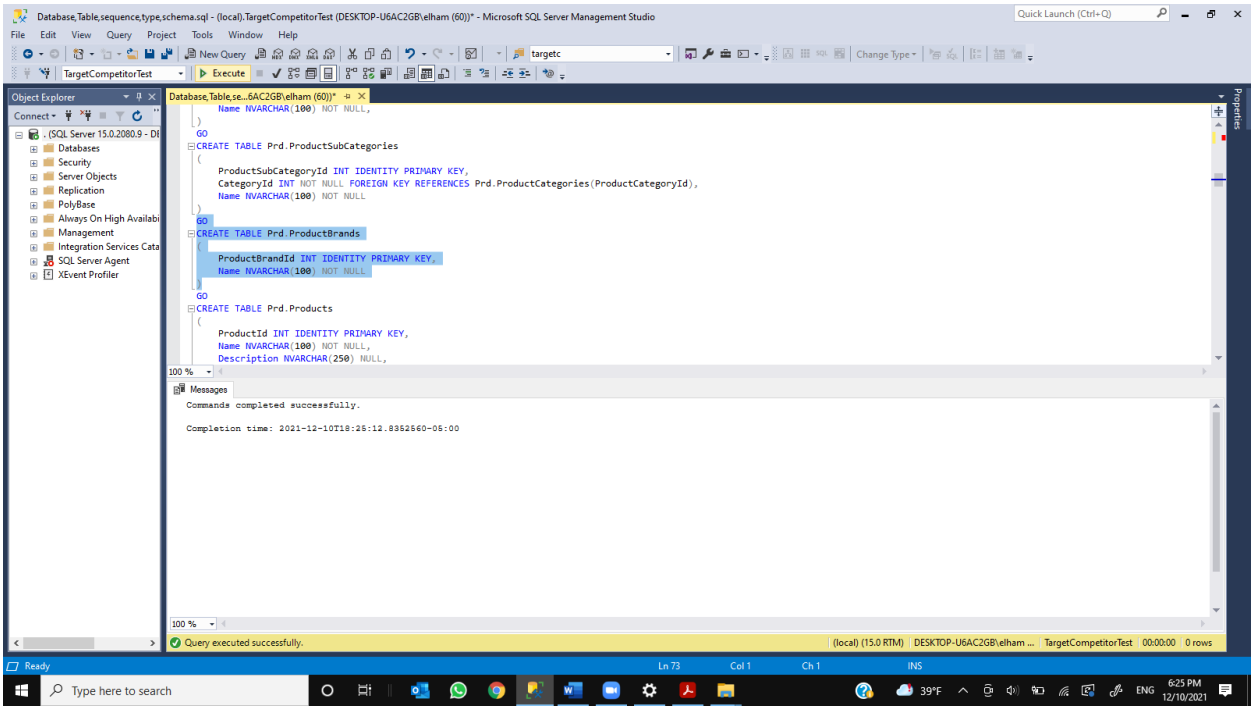
Comment: here we created the product categories table.

```
GO
CREATE TABLE Prd.ProductSubCategories
(
        ProductSubCategoryId INT IDENTITY PRIMARY KEY,
        CategoryId INT NOT NULL FOREIGN KEY REFERENCES
Prd.ProductCategories(ProductCategoryId),
        Name NVARCHAR(100) NOT NULL
)
```



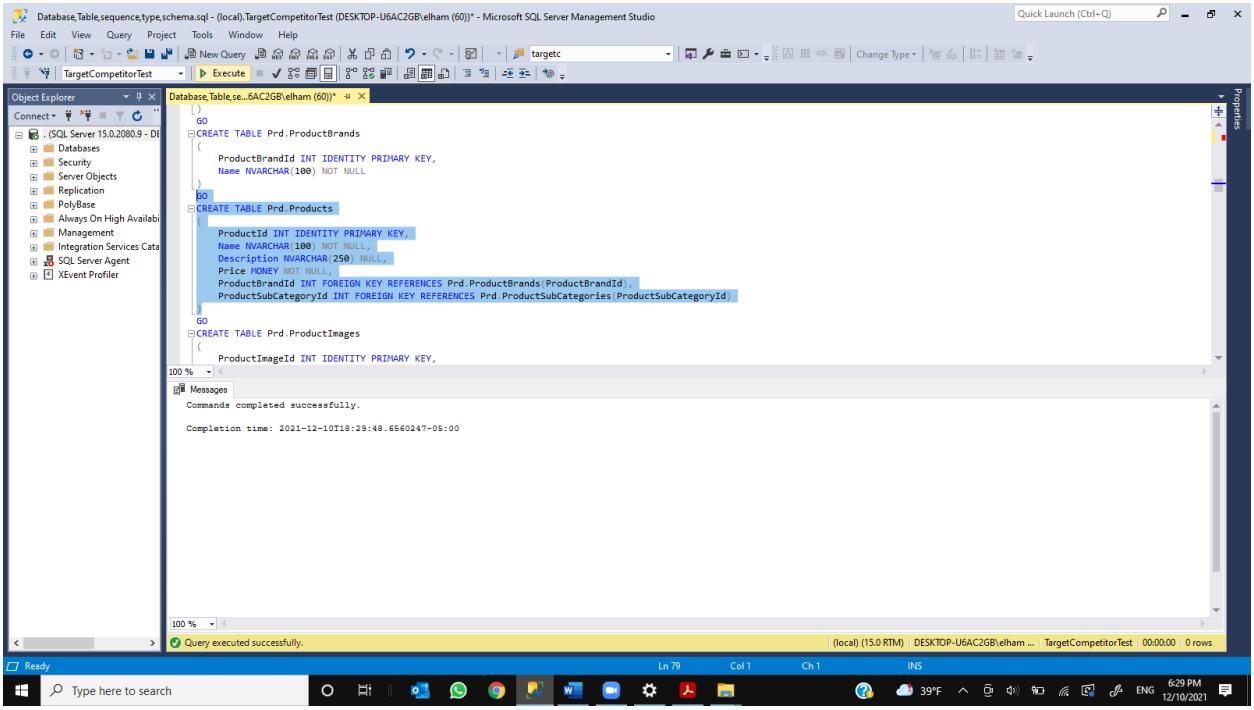
Comment: here we created the product subcategories table.

```
GO
CREATE TABLE Prd.ProductBrands
(
        ProductBrandId INT IDENTITY PRIMARY KEY,
        Name NVARCHAR(100) NOT NULL
)
```



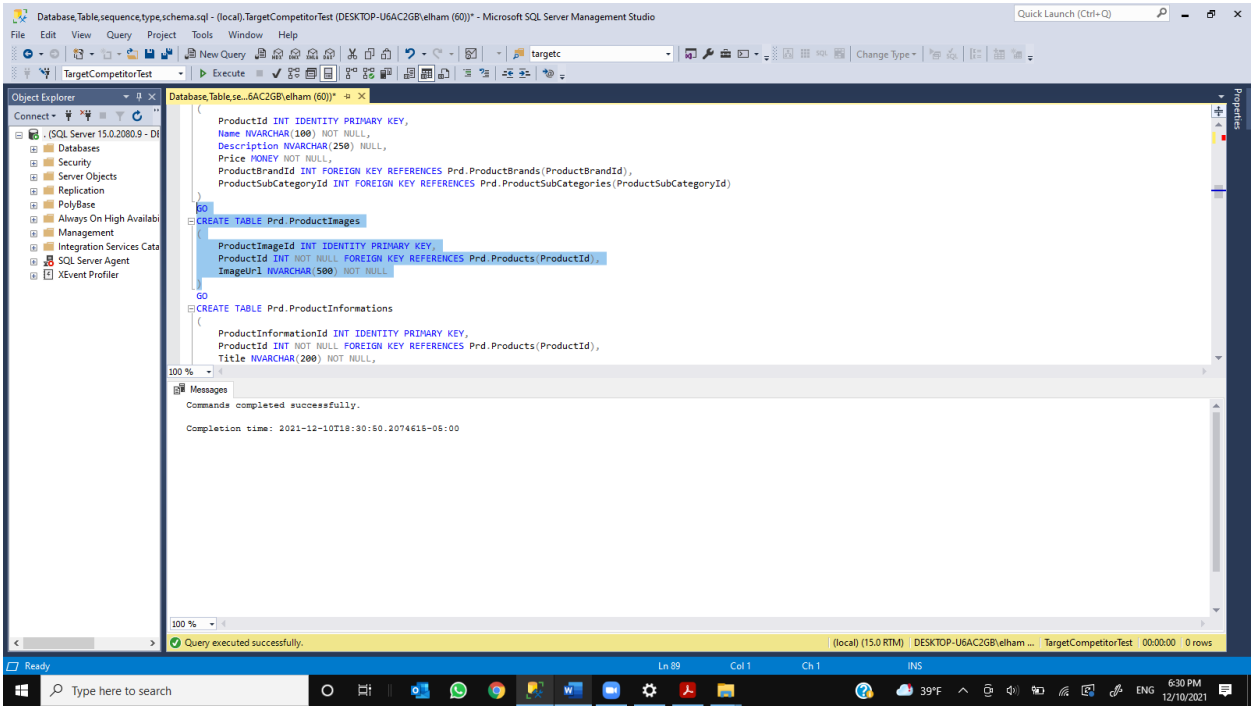
Comment: here we created the product brands table.

```
GO
CREATE TABLE Prd.Products
(
        ProductId INT IDENTITY PRIMARY KEY,
        Name NVARCHAR(100) NOT NULL,
        Description NVARCHAR(250) NULL,
        Price MONEY NOT NULL,
        ProductBrandId INT FOREIGN KEY REFERENCES Prd.ProductBrands(ProductBrandId),
        ProductSubCategoryId INT FOREIGN KEY REFERENCES
Prd.ProductSubCategories(ProductSubCategoryId)
)
```



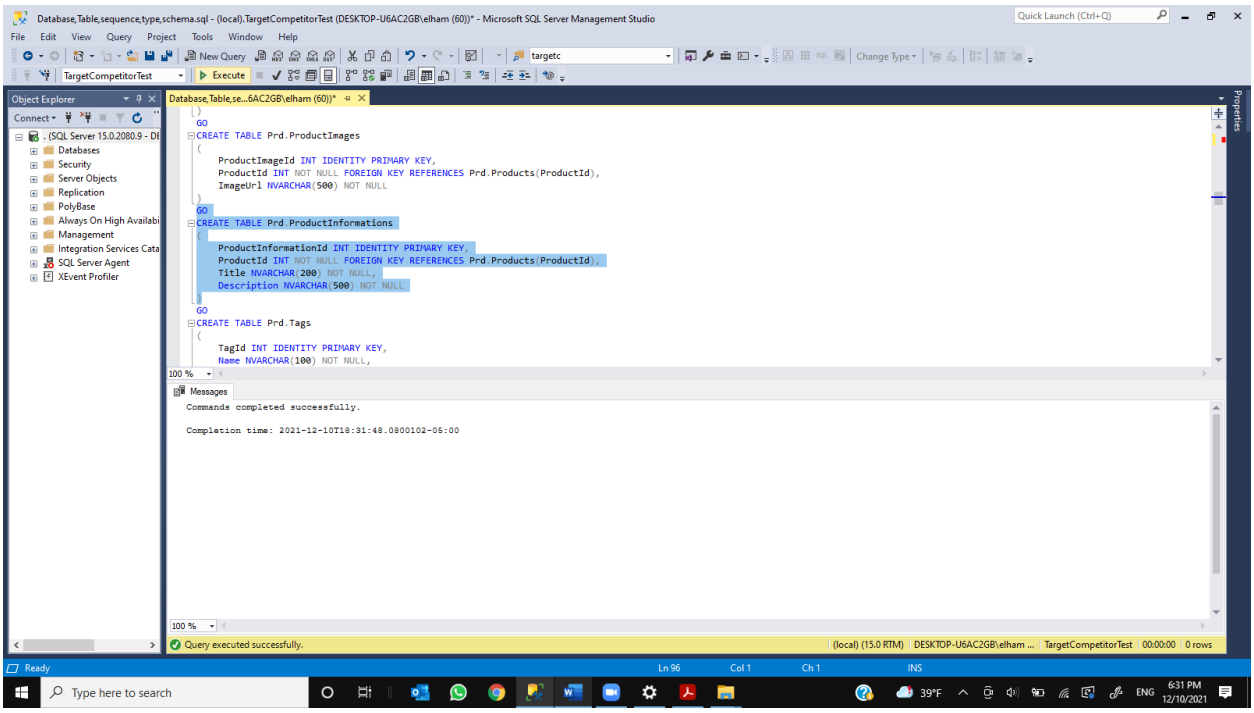
Comment: here we created the products table.

```
GO
CREATE TABLE Prd.ProductImages
(
	ProductImageId INT IDENTITY PRIMARY KEY,
	ProductId INT NOT NULL FOREIGN KEY REFERENCES Prd.Products(ProductId),
	ImageUrl NVARCHAR(500) NOT NULL
)
```



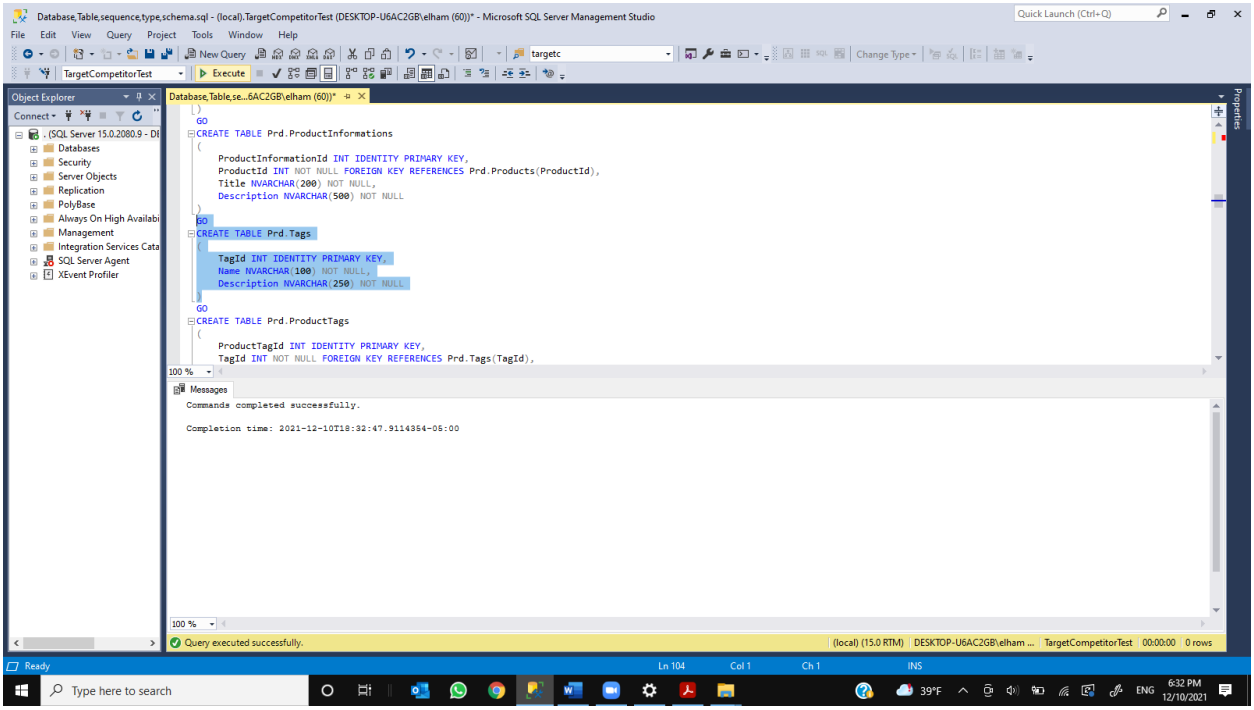
Comment: here we created the product images table.

```
GO
CREATE TABLE Prd.ProductInformations
(
        ProductInformationId INT IDENTITY PRIMARY KEY,
        ProductId INT NOT NULL FOREIGN KEY REFERENCES Prd.Products(ProductId),
        Title NVARCHAR(200) NOT NULL,
        Description NVARCHAR(500) NOT NULL
)
```



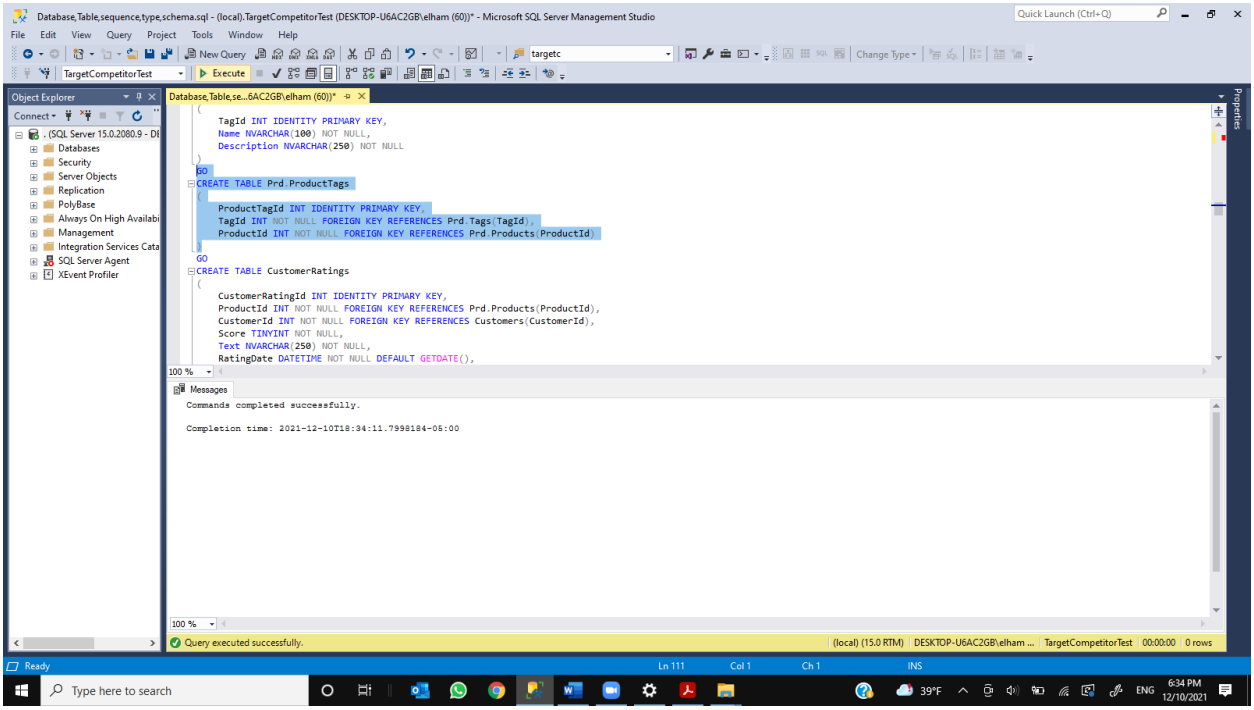
Comment: here we created the product information table.

```
GO
CREATE TABLE Prd.Tags
(
        TagId INT IDENTITY PRIMARY KEY,
        Name NVARCHAR(100) NOT NULL,
        Description NVARCHAR(250) NOT NULL
)
```



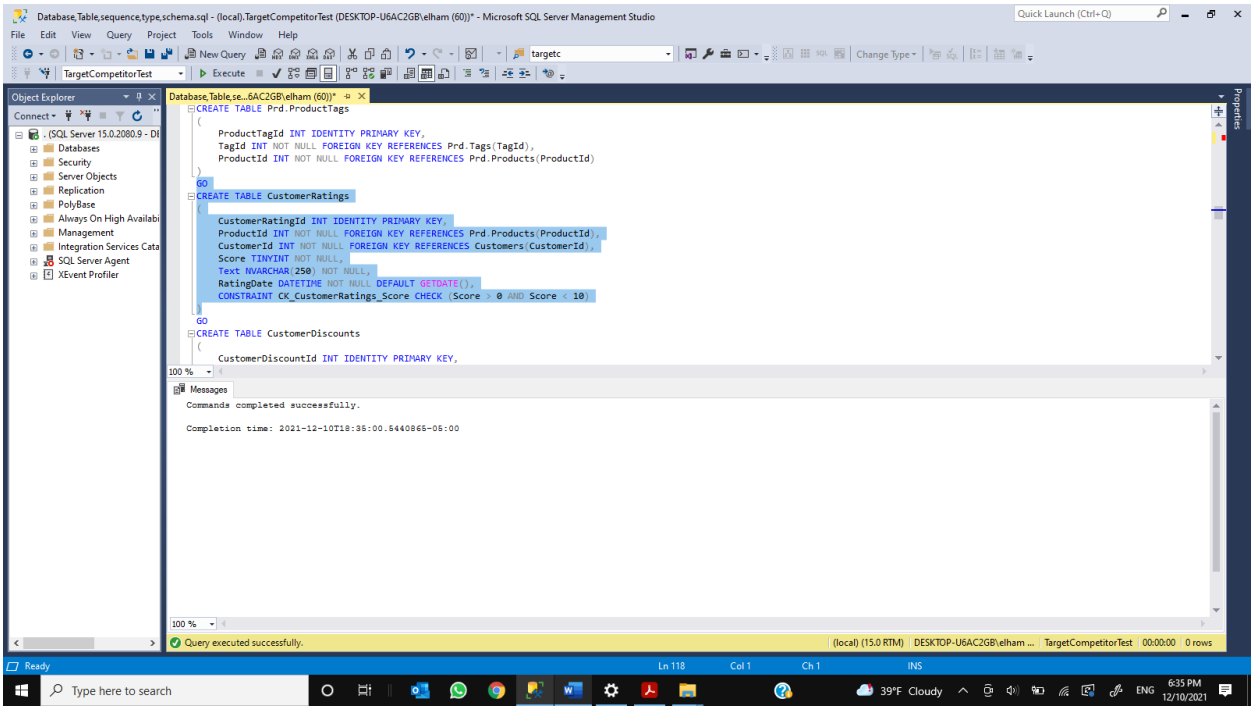
Comment: here we created the tags table.

```
GO
CREATE TABLE Prd.ProductTags
(
        ProductTagId INT IDENTITY PRIMARY KEY,
        TagId INT NOT NULL FOREIGN KEY REFERENCES Prd.Tags(TagId),
        ProductId INT NOT NULL FOREIGN KEY REFERENCES Prd.Products(ProductId)
)
```



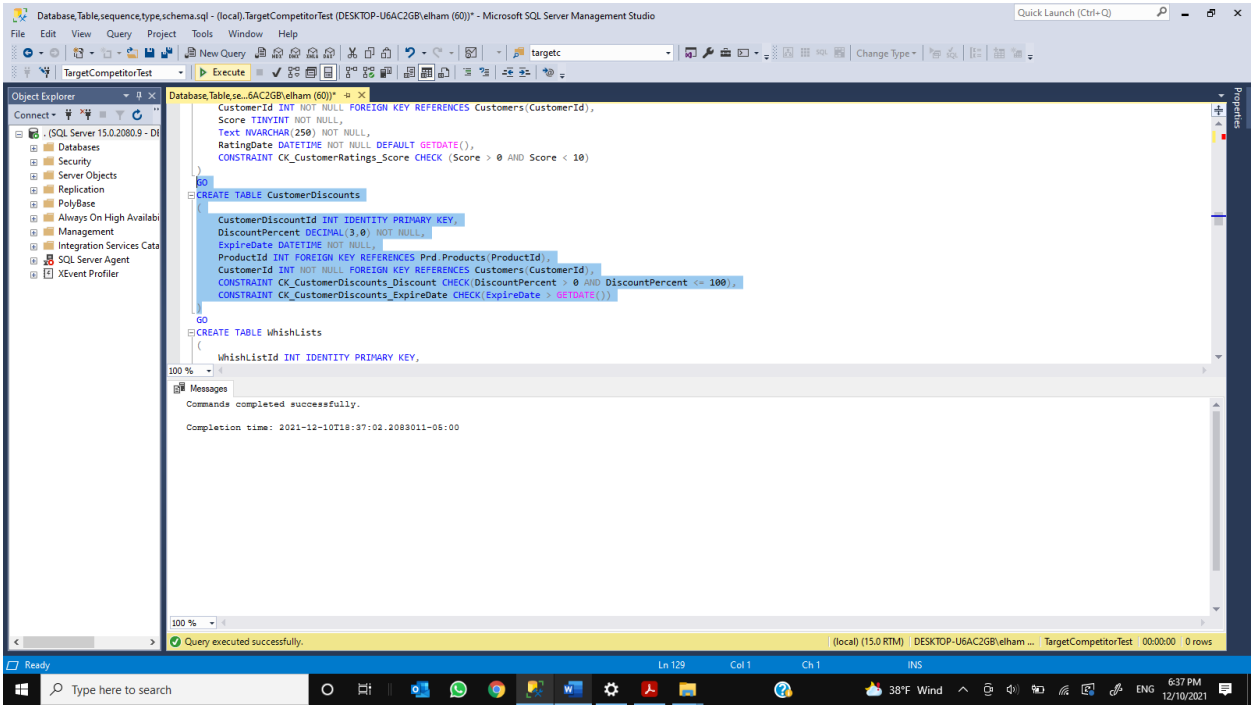
Comment: here we created the product tags linking table.

```
GO
CREATE TABLE CustomerRatings
(
        CustomerRatingId INT IDENTITY PRIMARY KEY,
        ProductId INT NOT NULL FOREIGN KEY REFERENCES Prd.Products(ProductId),
        CustomerId INT NOT NULL FOREIGN KEY REFERENCES Customers(CustomerId),
        Score DECIMAL(4,2) NOT NULL,
        Text NVARCHAR(250) NOT NULL,
        RatingDate DATETIME NOT NULL DEFAULT GETDATE(),
        CONSTRAINT CK_CustomerRatings_Score CHECK (Score > 0 AND Score < 10)
)
```



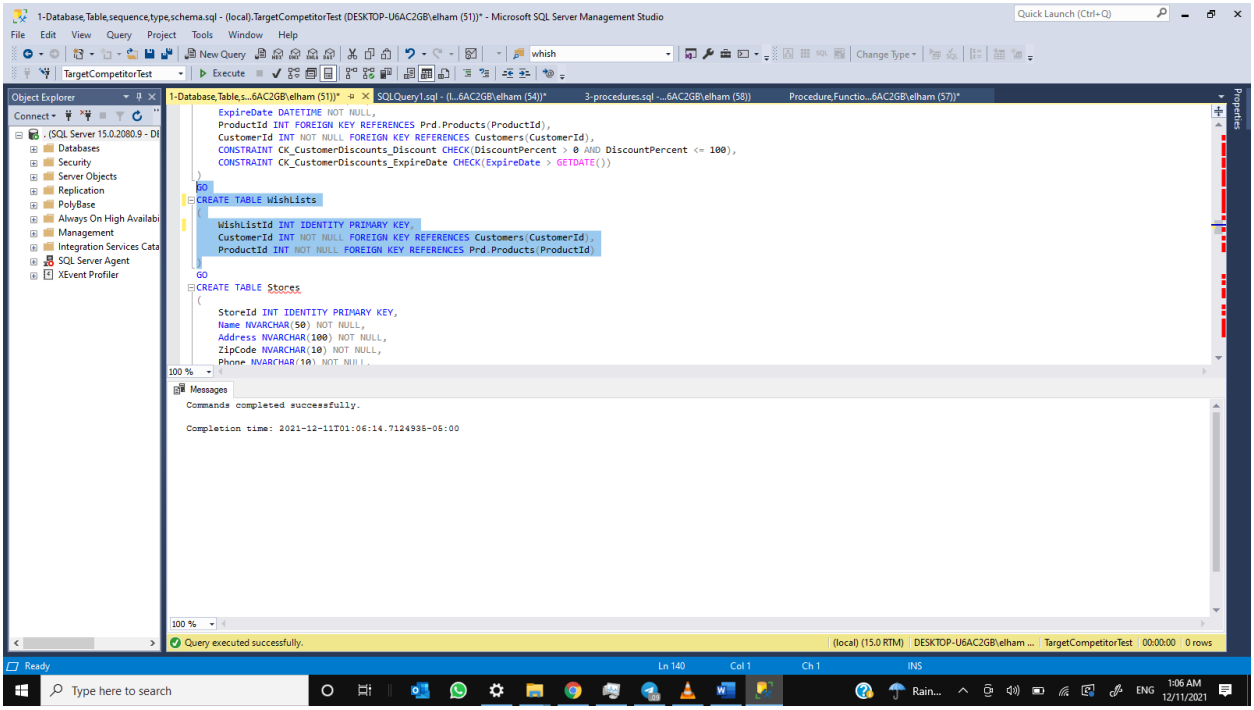
Comment: here we created the customer ratings table.

```
GO
CREATE TABLE CustomerDiscounts
(
	CustomerDiscountId INT IDENTITY PRIMARY KEY,
	DiscountPercent DECIMAL(3,0) NOT NULL,
	ExpireDate DATETIME NOT NULL,
	ProductId INT FOREIGN KEY REFERENCES Prd.Products(ProductId),
	CustomerId INT NOT NULL FOREIGN KEY REFERENCES Customers(CustomerId),
	CONSTRAINT CK_CustomerDiscounts_Discount CHECK(DiscountPercent > 0 AND DiscountPercent
<= 100),
	CONSTRAINT CK_CustomerDiscounts_ExpireDate CHECK(ExpireDate > GETDATE())
)
```



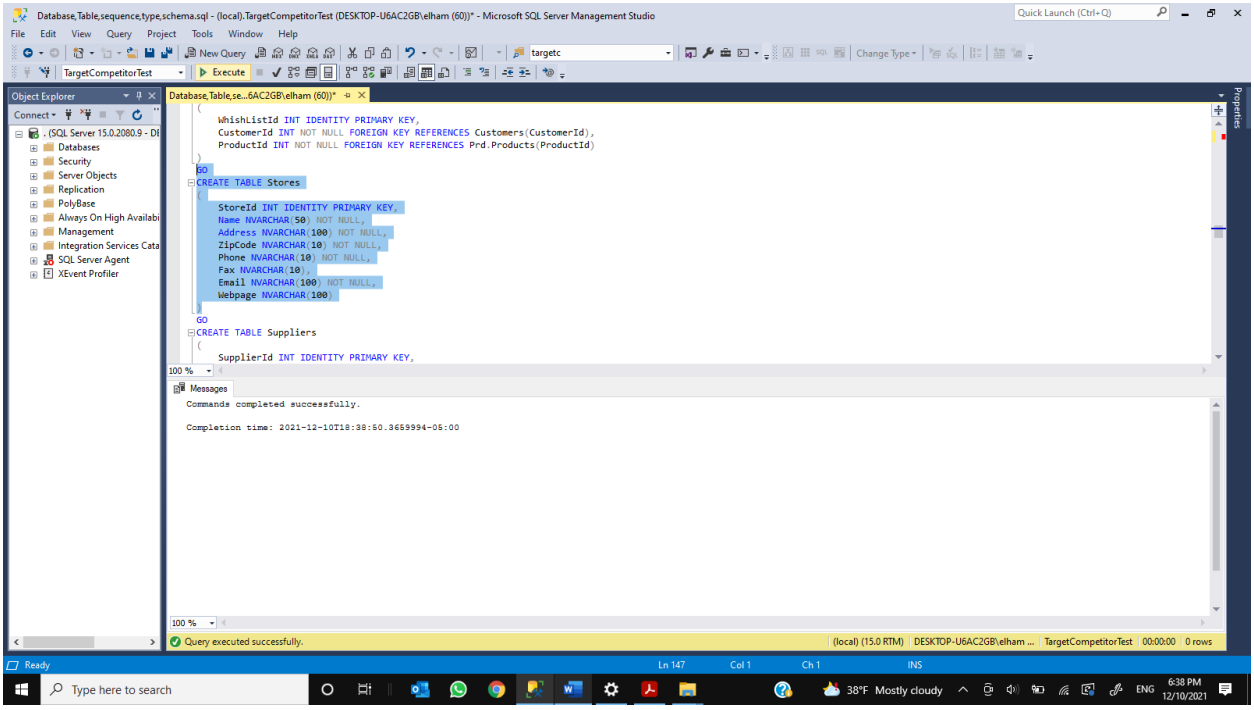
Comment: here we created the customer discount table.

```
GO
CREATE TABLE WishLists
(
        WishListId INT IDENTITY PRIMARY KEY,
        CustomerId INT NOT NULL FOREIGN KEY REFERENCES Customers(CustomerId),
        ProductId INT NOT NULL FOREIGN KEY REFERENCES Prd.Products(ProductId)
)
```



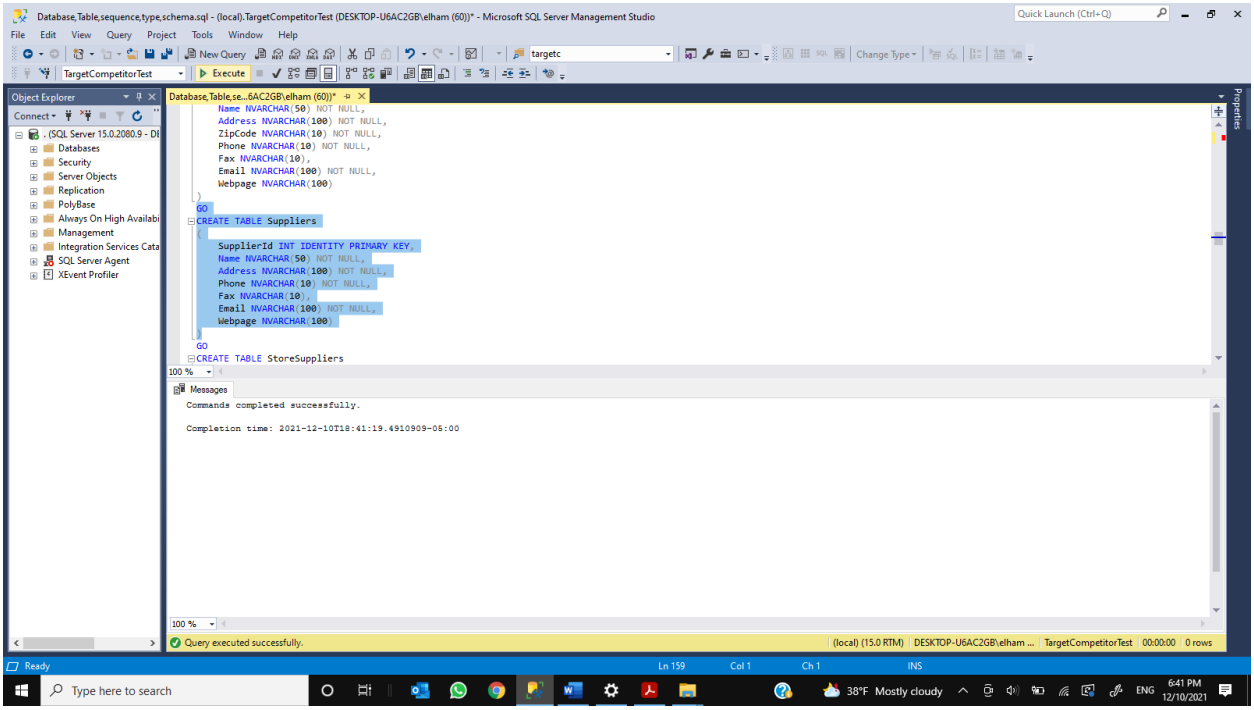
Comment: here we created the wish lists table.

```
GO
CREATE TABLE Stores
(
        StoreId INT IDENTITY PRIMARY KEY,
        Name NVARCHAR(50) NOT NULL,
        Address NVARCHAR(100) NOT NULL,
        ZipCode NVARCHAR(10) NOT NULL,
        Phone NVARCHAR(10) NOT NULL,
        Fax NVARCHAR(10),
        Email NVARCHAR(100) NOT NULL,
        Webpage NVARCHAR(100)
)
```



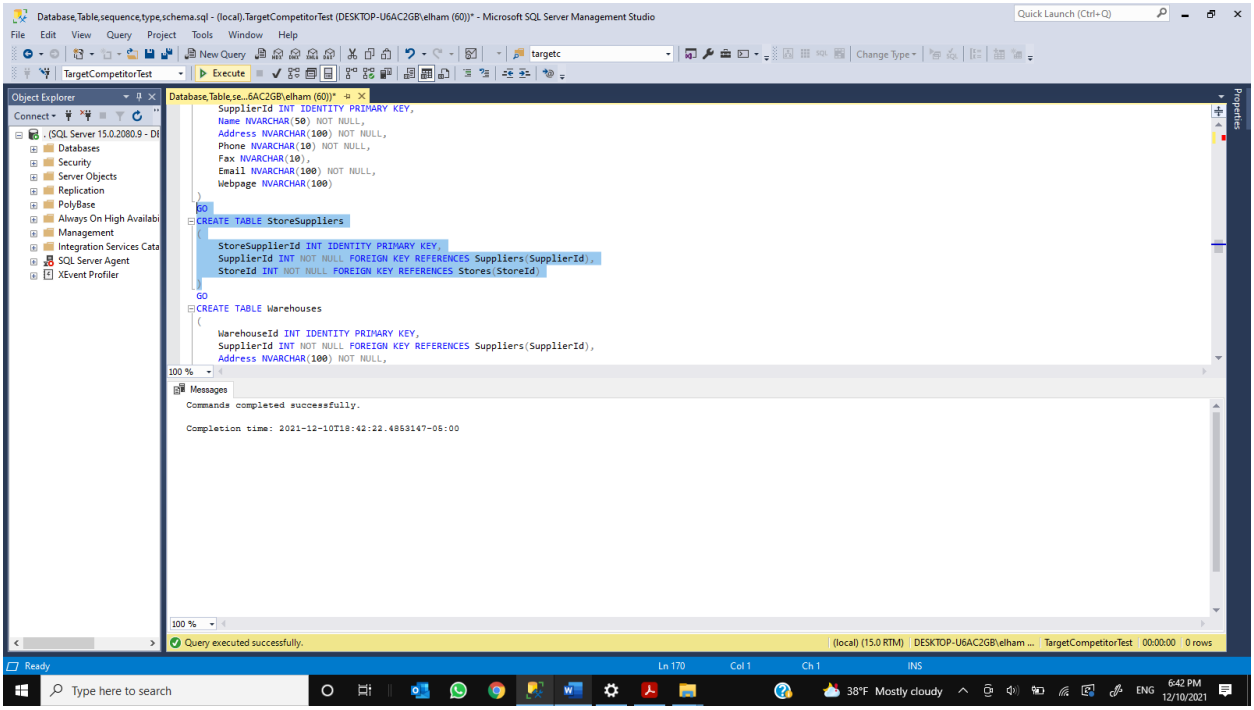
Comment: here we created the stores table.

```
GO
CREATE TABLE Suppliers
(
        SupplierId INT IDENTITY PRIMARY KEY,
        Name NVARCHAR(50) NOT NULL,
        Address NVARCHAR(100) NOT NULL,
        Phone NVARCHAR(10) NOT NULL,
        Fax NVARCHAR(10),
        Email NVARCHAR(100) NOT NULL,
        Webpage NVARCHAR(100)
)
```



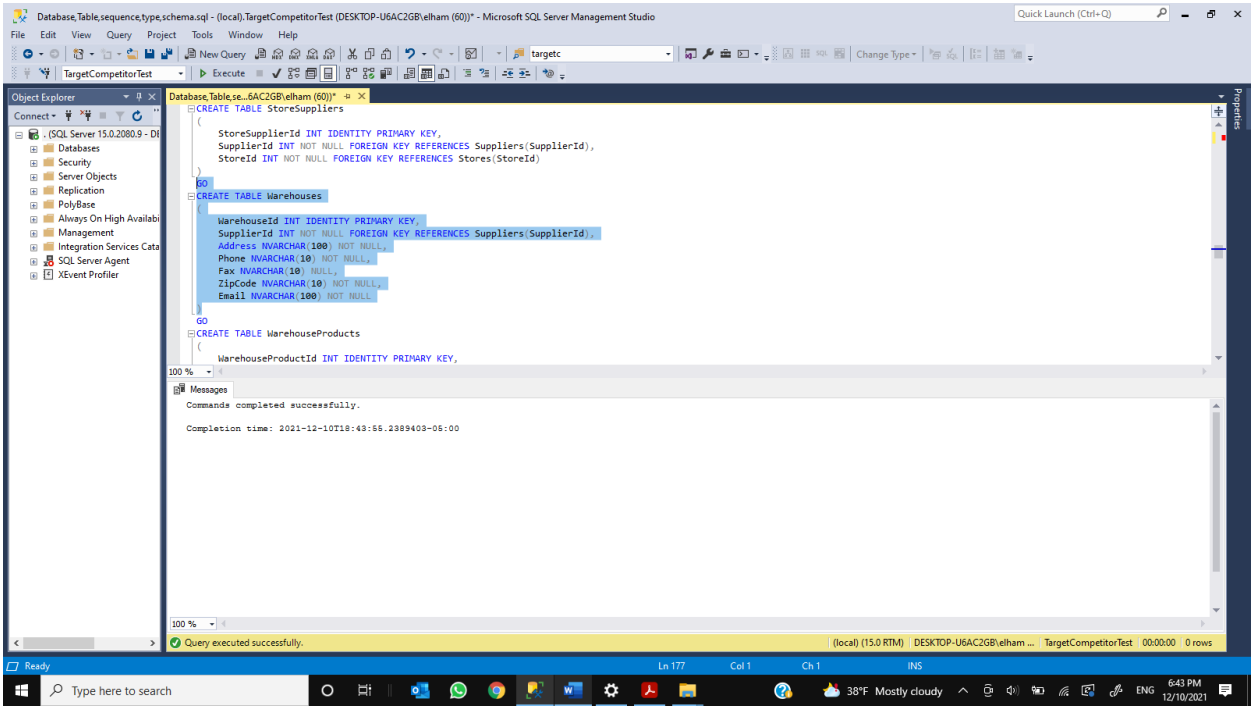
Comment: here we created the supplier's table.

```
GO
CREATE TABLE StoreSuppliers
(
        StoreSupplierId INT IDENTITY PRIMARY KEY,
        SupplierId INT NOT NULL FOREIGN KEY REFERENCES Suppliers(SupplierId),
        StoreId INT NOT NULL FOREIGN KEY REFERENCES Stores(StoreId)
)
```



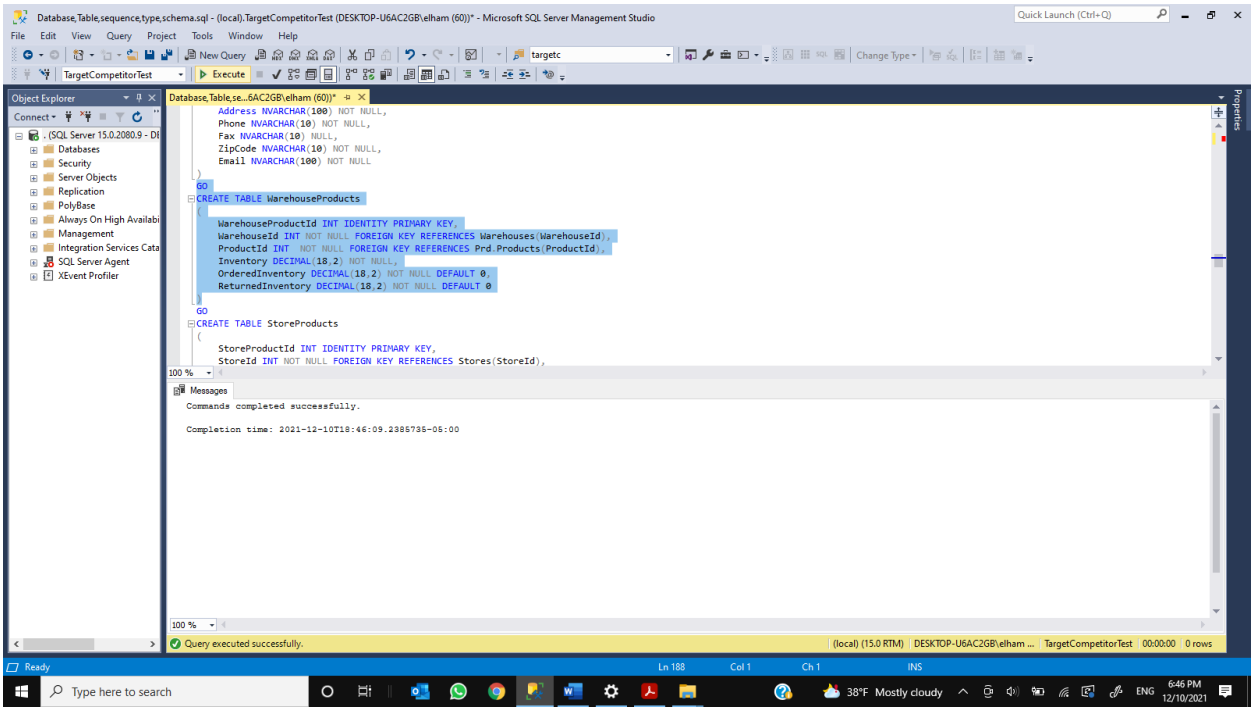
Comment: here we created the store-suppliers linking table.

```
GO
CREATE TABLE Warehouses
(
        WarehouseId INT IDENTITY PRIMARY KEY,
        SupplierId INT NOT NULL FOREIGN KEY REFERENCES Suppliers(SupplierId),
        Address NVARCHAR(100) NOT NULL,
        Phone NVARCHAR(10) NOT NULL,
        Fax NVARCHAR(10) NULL,
        ZipCode NVARCHAR(10) NOT NULL,
        Email NVARCHAR(100) NOT NULL
)
```



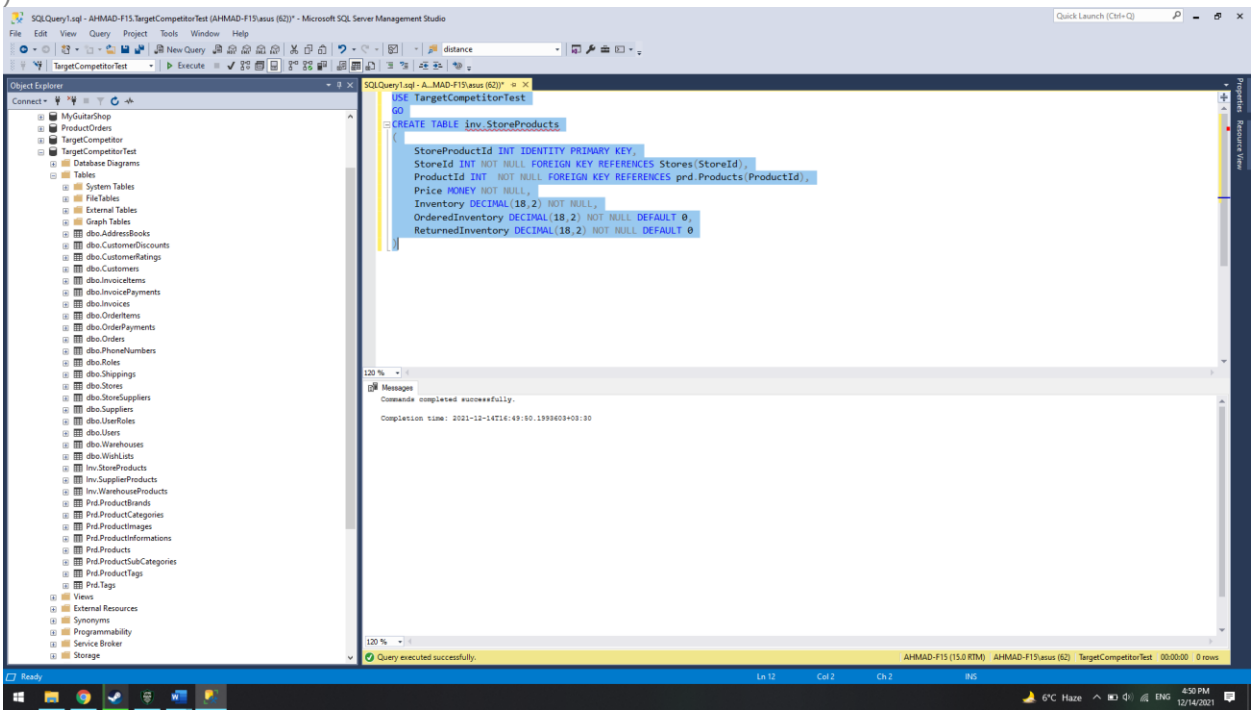
Comment: here we created the warehouses table.

```
GO
CREATE TABLE WarehouseProducts
(
        WarehouseProductId INT IDENTITY PRIMARY KEY,
        WarehouseId INT NOT NULL FOREIGN KEY REFERENCES Warehouses(WarehouseId),
        ProductId INT  NOT NULL FOREIGN KEY REFERENCES Prd.Products(ProductId),
        Inventory DECIMAL(18,2) NOT NULL,
        OrderedInventory DECIMAL(18,2) NOT NULL DEFAULT 0,
        ReturnedInventory DECIMAL(18,2) NOT NULL DEFAULT 0
)
```



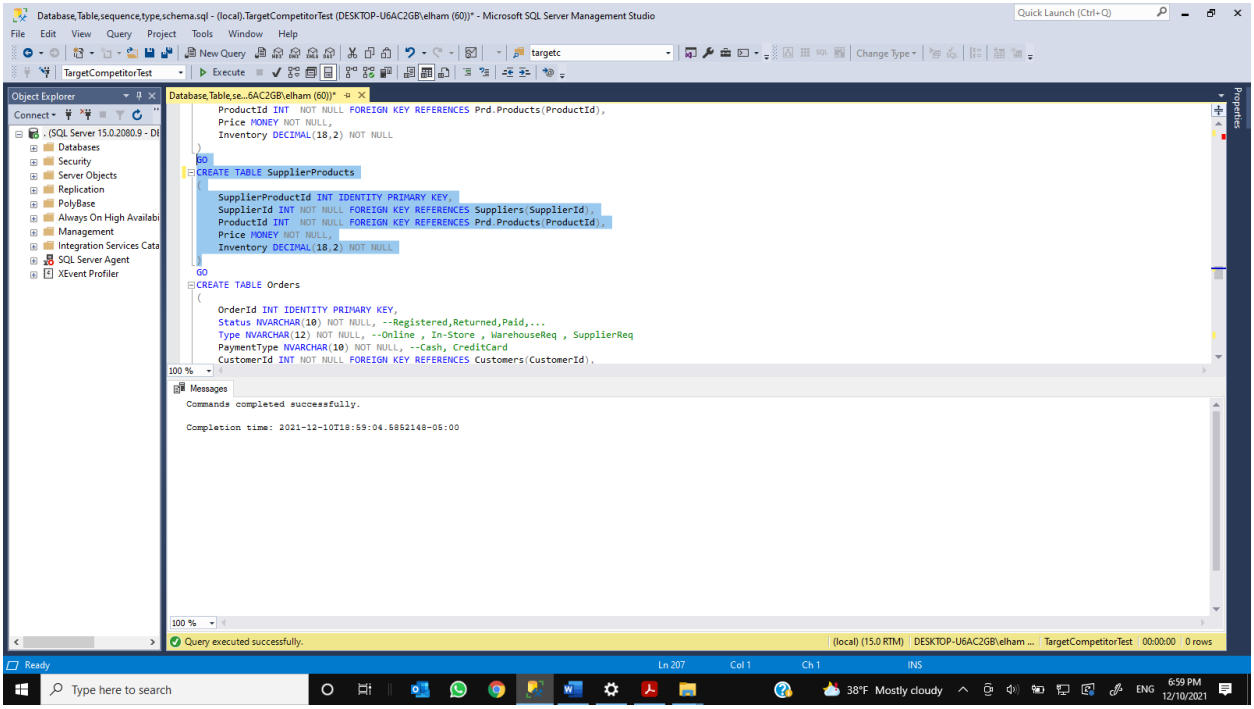
Comment: here we created the warehouse-products linking table. There are columns to show the inventory, ordered inventory, and returned inventory.

```
USE TargetCompetitorTest
GO
CREATE TABLE inv.StoreProducts
(
	StoreProductId INT IDENTITY PRIMARY KEY,
	StoreId INT NOT NULL FOREIGN KEY REFERENCES Stores(StoreId),
	ProductId INT  NOT NULL FOREIGN KEY REFERENCES prd.Products(ProductId),
	Price MONEY NOT NULL,
	Inventory DECIMAL(18,2) NOT NULL,
	OrderedInventory DECIMAL(18,2) NOT NULL DEFAULT 0,
	ReturnedInventory DECIMAL(18,2) NOT NULL DEFAULT 0
)
```



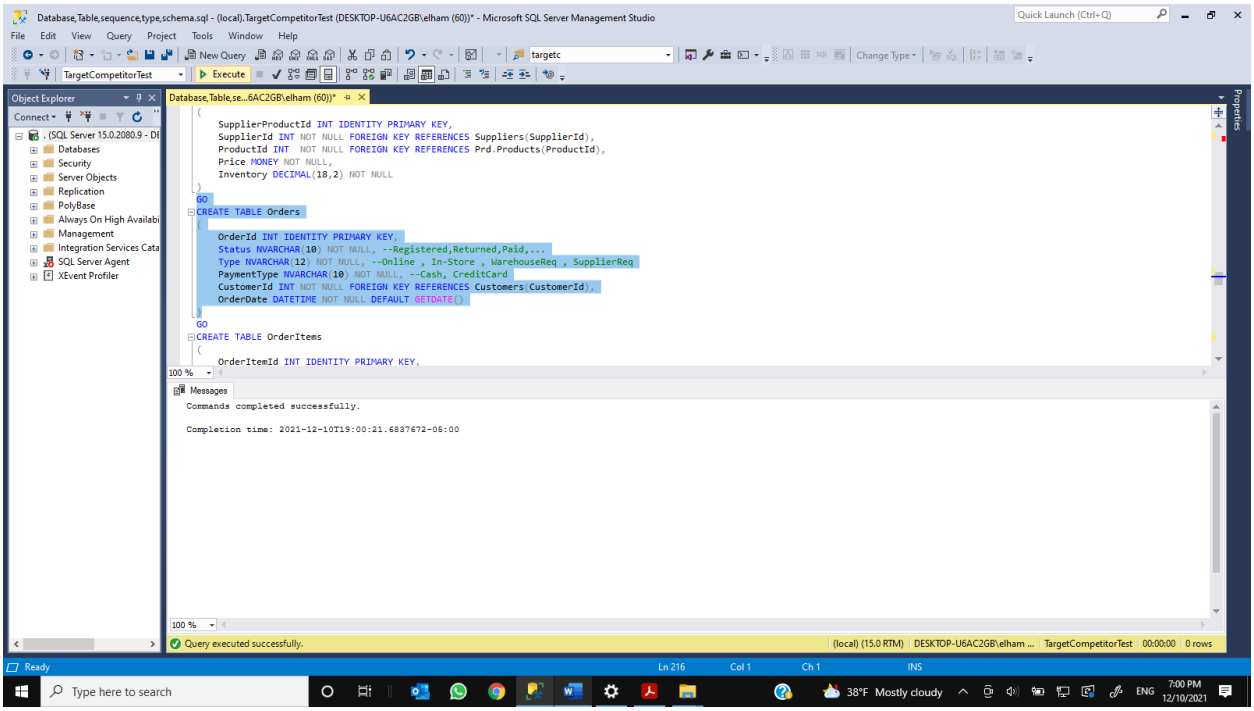
Comment: here we created the store-products linking table.

```
GO
CREATE TABLE SupplierProducts
(
        SupplierProductId INT IDENTITY PRIMARY KEY,
        SupplierId INT NOT NULL FOREIGN KEY REFERENCES Suppliers(SupplierId),
        ProductId INT  NOT NULL FOREIGN KEY REFERENCES Prd.Products(ProductId),
        Price MONEY NOT NULL,
        Inventory DECIMAL(18,2) NOT NULL
)
```



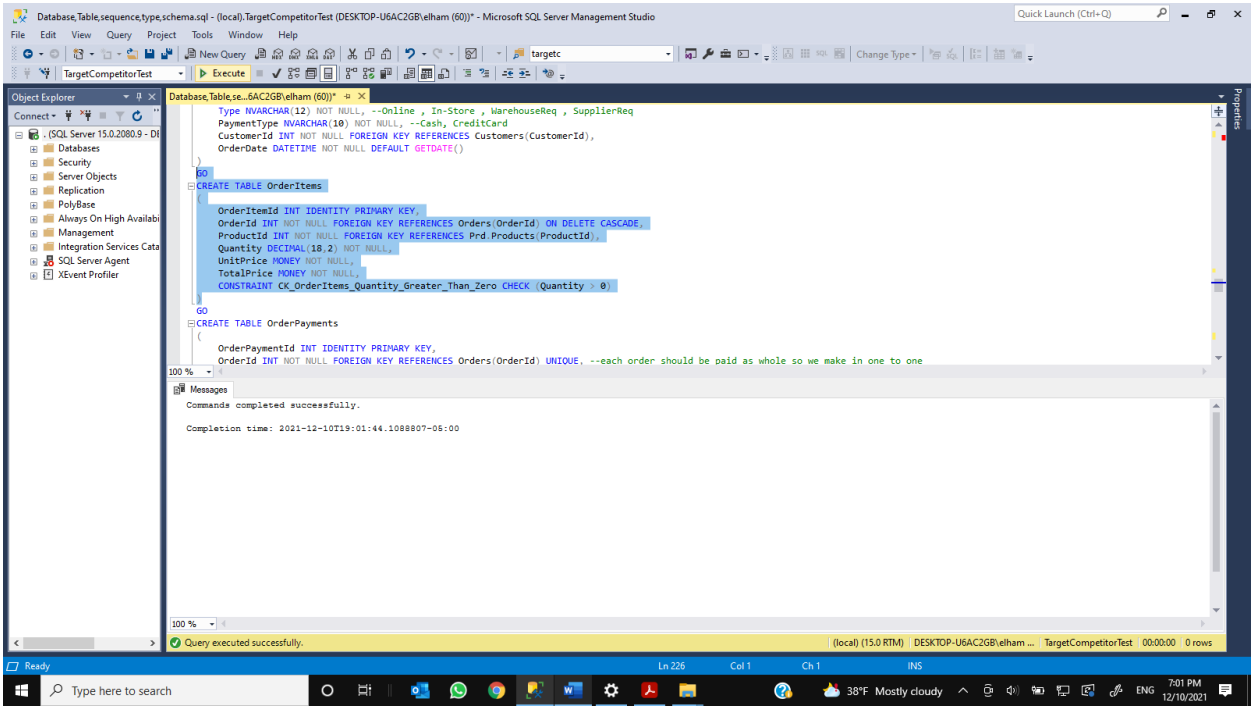
Comment: here we created the supplier-products linking table.

```
GO
CREATE TABLE Orders
(
        OrderId INT IDENTITY PRIMARY KEY,
        Status NVARCHAR(10) NOT NULL, --Registered,Returned,Paid,...
        Type NVARCHAR(12) NOT NULL, --Online , In-Store , WarehouseReq , SupplierReq
        PaymentType NVARCHAR(10) NOT NULL, --Cash, CreditCard
        CustomerId INT NOT NULL FOREIGN KEY REFERENCES Customers(CustomerId),
        OrderDate DATETIME NOT NULL DEFAULT GETDATE()
)
```



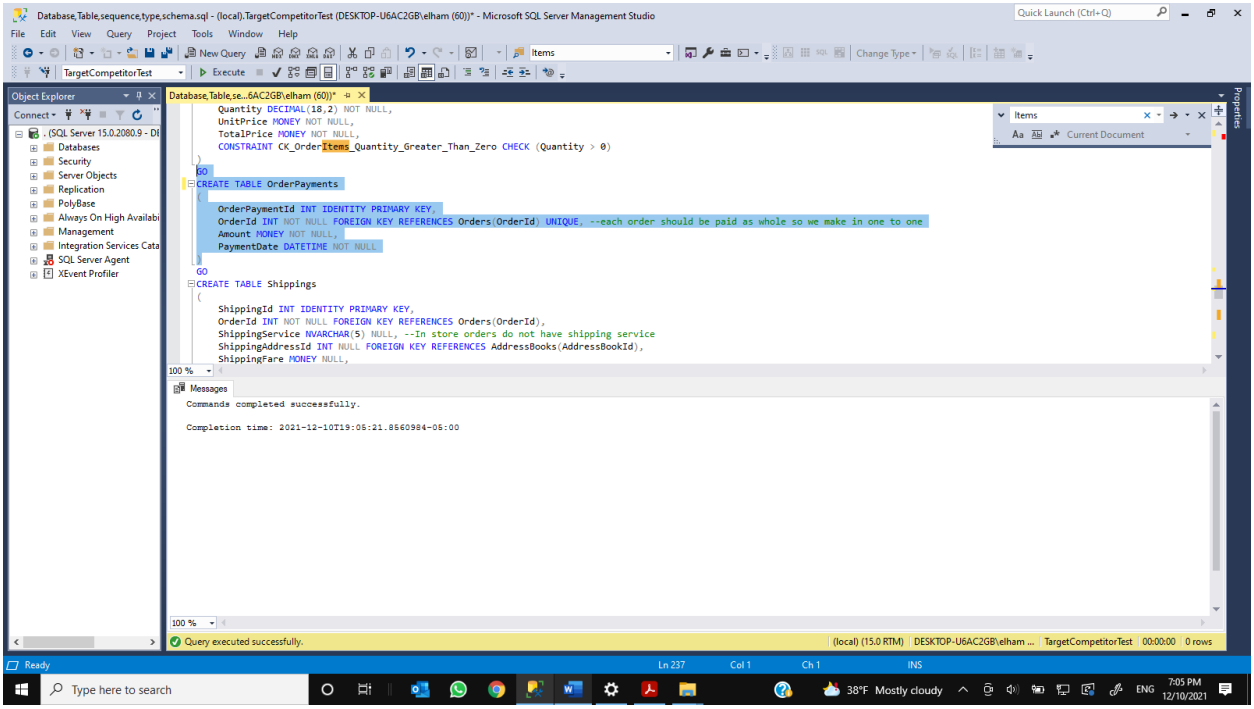
Comment: here we created the orders table.

```
GO
CREATE TABLE OrderItems
(
        OrderItemId INT IDENTITY PRIMARY KEY,
        OrderId INT NOT NULL FOREIGN KEY REFERENCES Orders(OrderId) ON DELETE CASCADE,
        ProductId INT NOT NULL FOREIGN KEY REFERENCES Prd.Products(ProductId),
        Quantity DECIMAL(18,2) NOT NULL,
        UnitPrice MONEY NOT NULL,
        TotalPrice MONEY NOT NULL,
        CONSTRAINT CK_OrderItems_Quantity_Greater_Than_Zero CHECK (Quantity > 0)
)
```



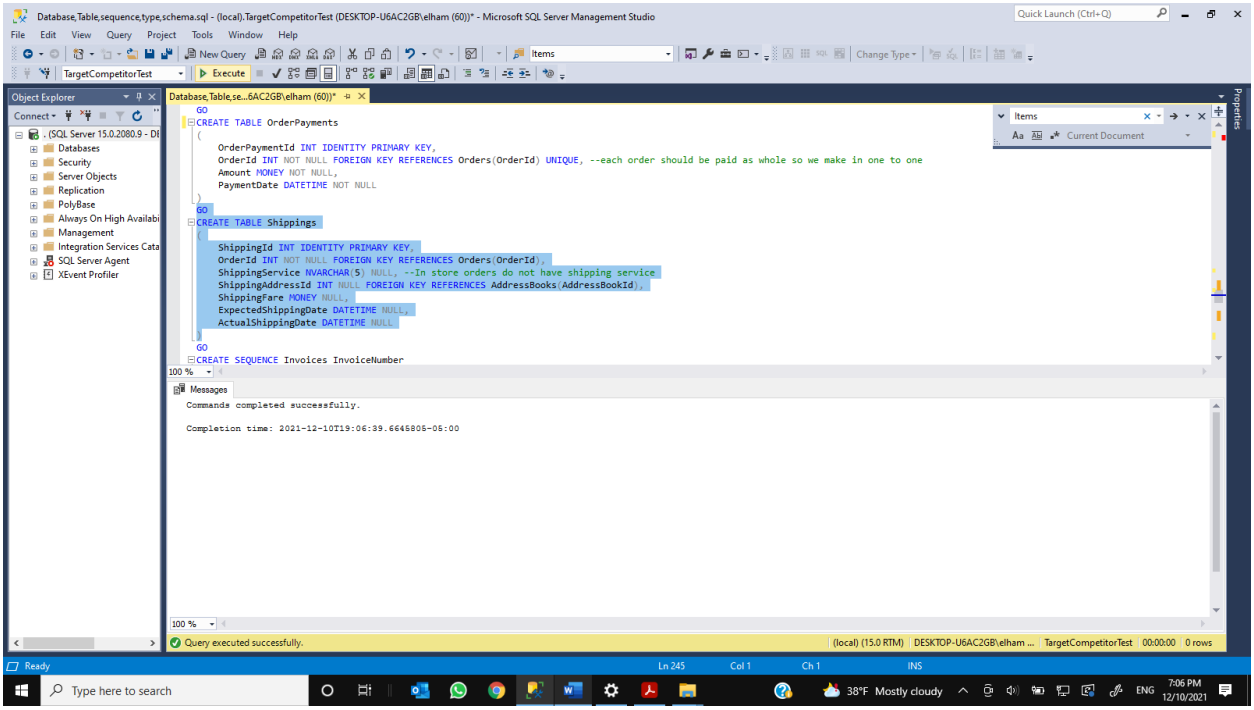
Comment: here we created the order-items linking table.

```
GO
CREATE TABLE OrderPayments
(
        OrderPaymentId INT IDENTITY PRIMARY KEY,
        OrderId INT NOT NULL FOREIGN KEY REFERENCES Orders(OrderId) UNIQUE, --each order
should be paid as whole so we make in one to one
        Amount MONEY NOT NULL,
        PaymentDate DATETIME NOT NULL
)
```



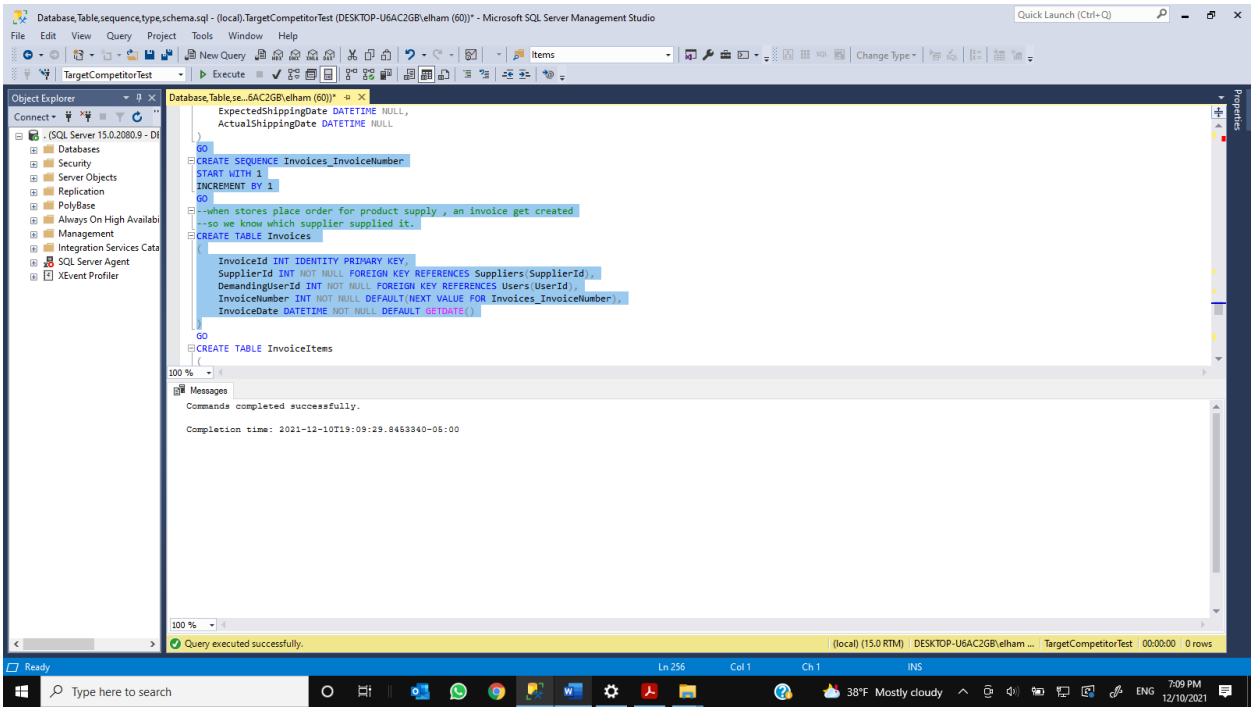
Comment: here we created the order payment table.

```
GO
CREATE TABLE Shippings
(
        ShippingId INT IDENTITY PRIMARY KEY,
        OrderId INT NOT NULL FOREIGN KEY REFERENCES Orders(OrderId),
        ShippingService NVARCHAR(5) NULL, --In store orders do not have shipping service
        ShippingAddressId INT NULL FOREIGN KEY REFERENCES AddressBooks(AddressBookId),
        ShippingFare MONEY NULL,
        ExpectedShippingDate DATETIME NULL,
        ActualShippingDate DATETIME NULL
)
```



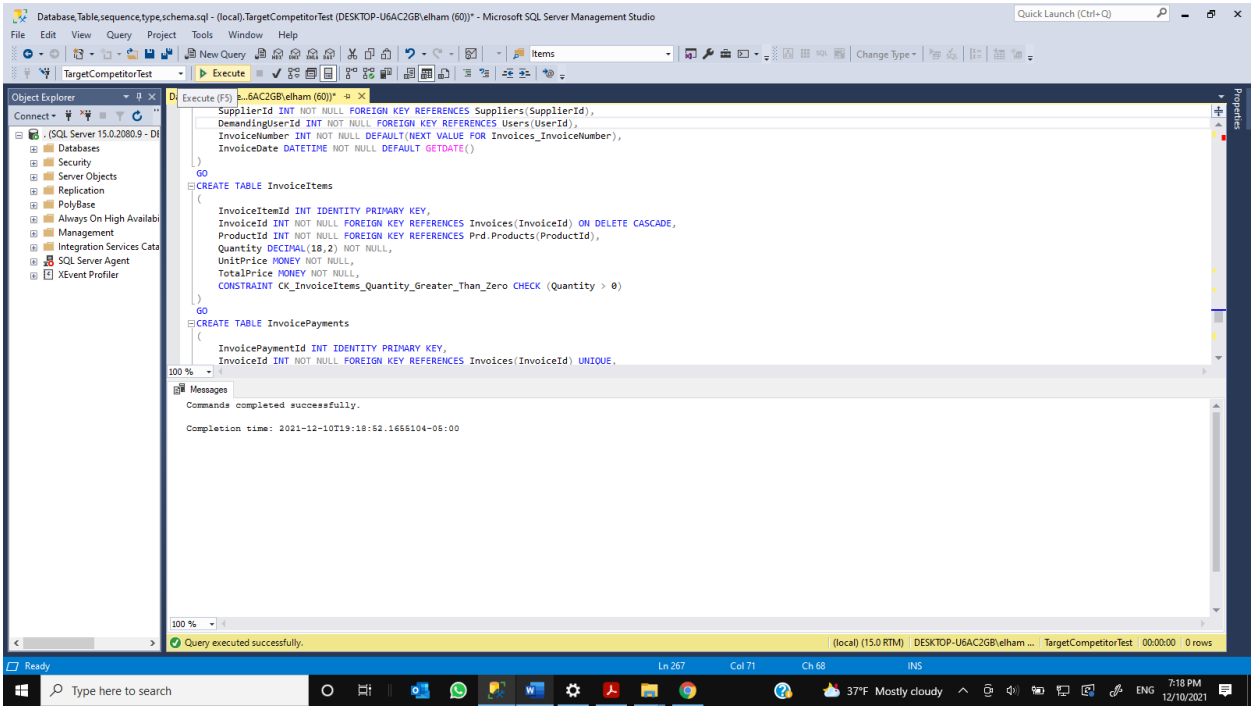
Comment: here we created the shipping table.

```
GO
CREATE SEQUENCE Invoices_InvoiceNumber
START WITH 1
INCREMENT BY 1
GO
--when stores place order for product supply , an invoice get created
--so we know which supplier supplied it.
CREATE TABLE Invoices
(
        InvoiceId INT IDENTITY PRIMARY KEY,
        SupplierId INT NOT NULL FOREIGN KEY REFERENCES Suppliers(SupplierId),
        DemandingUserId INT NOT NULL FOREIGN KEY REFERENCES Users(UserId),
        InvoiceNumber INT NOT NULL DEFAULT(NEXT VALUE FOR Invoices_InvoiceNumber),
        InvoiceDate DATETIME NOT NULL DEFAULT GETDATE()
)
```



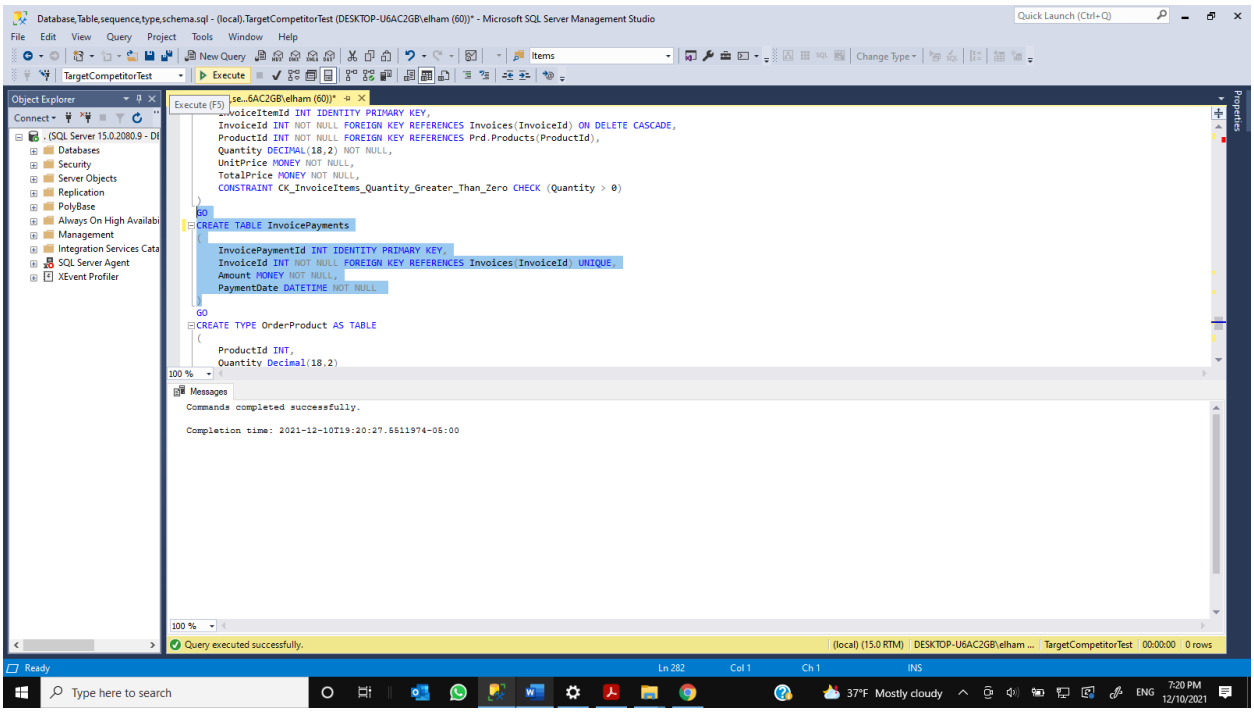
Comment: Here we created the invoices table. We also created a sequence to create numbers for the times when stores place an order for product supply. (An invoice gets created to inform us about the supplier who supplied it).

```
GO
CREATE TABLE InvoiceItems
(
        InvoiceItemId INT IDENTITY PRIMARY KEY,
        InvoiceId INT NOT NULL FOREIGN KEY REFERENCES Invoices(InvoiceId) ON DELETE CASCADE,
        ProductId INT NOT NULL FOREIGN KEY REFERENCES Prd.Products(ProductId),
        Quantity DECIMAL(18,2) NOT NULL,
        UnitPrice MONEY NOT NULL,
        TotalPrice MONEY NOT NULL,
        CONSTRAINT CK_InvoiceItems_Quantity_Greater_Than_Zero CHECK (Quantity > 0)
)
```



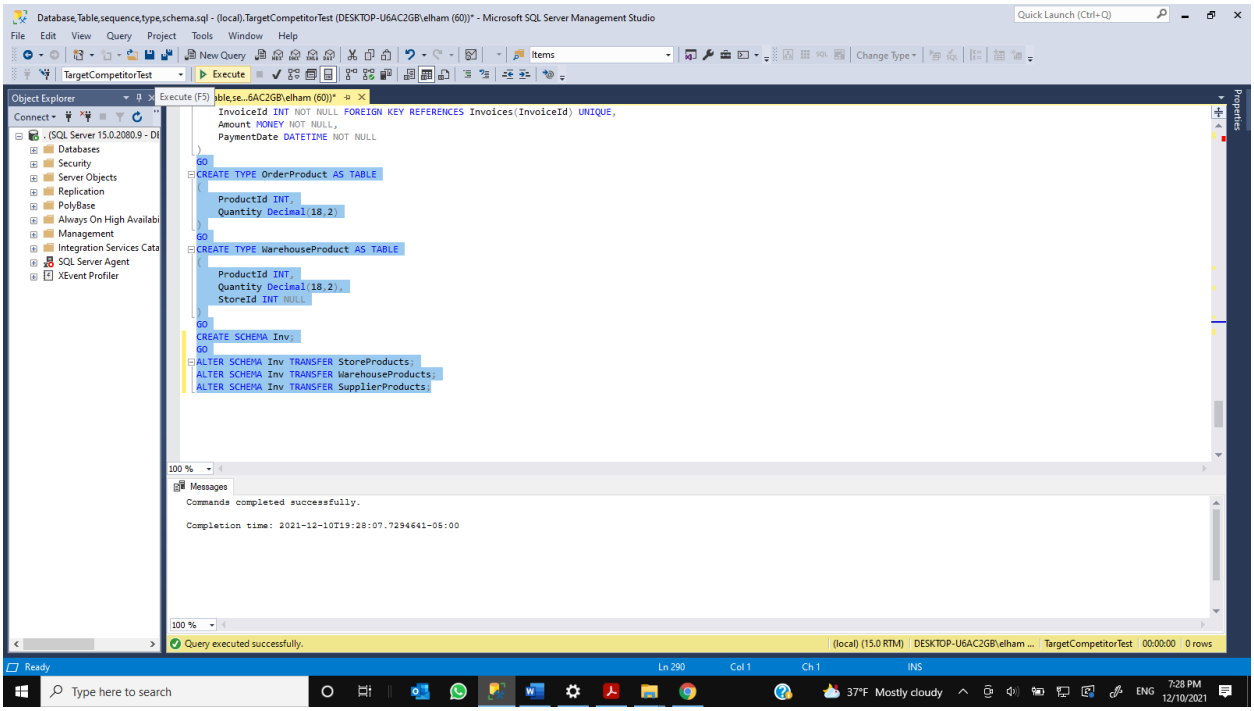
Comment: here we created the invoice items table.

```
GO
CREATE TABLE InvoicePayments
(
        InvoicePaymentId INT IDENTITY PRIMARY KEY,
        InvoiceId INT NOT NULL FOREIGN KEY REFERENCES Invoices(InvoiceId) UNIQUE,
        Amount MONEY NOT NULL,
        PaymentDate DATETIME NOT NULL
)
```



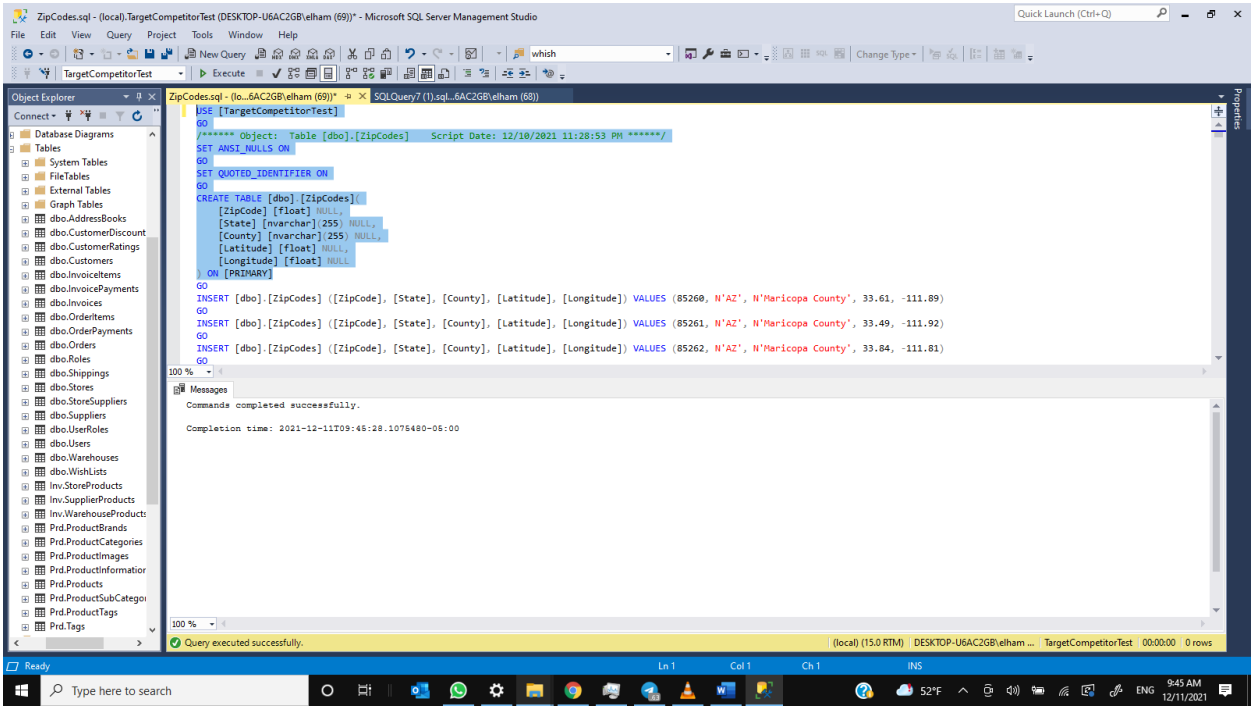
Comment: here we created the invoice payment table.

```
GO
CREATE TYPE OrderProduct AS TABLE
(
        ProductId INT,
        Quantity Decimal(18,2)
)
GO
CREATE TYPE WarehouseProduct AS TABLE
(
        ProductId INT,
        Quantity Decimal(18,2),
        StoreId INT NULL
)
GO
CREATE SCHEMA Inv;
GO
ALTER SCHEMA Inv TRANSFER StoreProducts;
ALTER SCHEMA Inv TRANSFER WarehouseProducts;
ALTER SCHEMA Inv TRANSFER SupplierProducts;
```



Comment: here we created the types and schemas.

```
USE [TargetCompetitorTest]
GO
CREATE TABLE [dbo].[ZipCodes](
	[ZipCode] [float] NULL,
	[State] [nvarchar](255) NULL,
	[County] [nvarchar](255) NULL,
	[Latitude] [float] NULL,
	[Longitude] [float] NULL
) ON [PRIMARY]
```

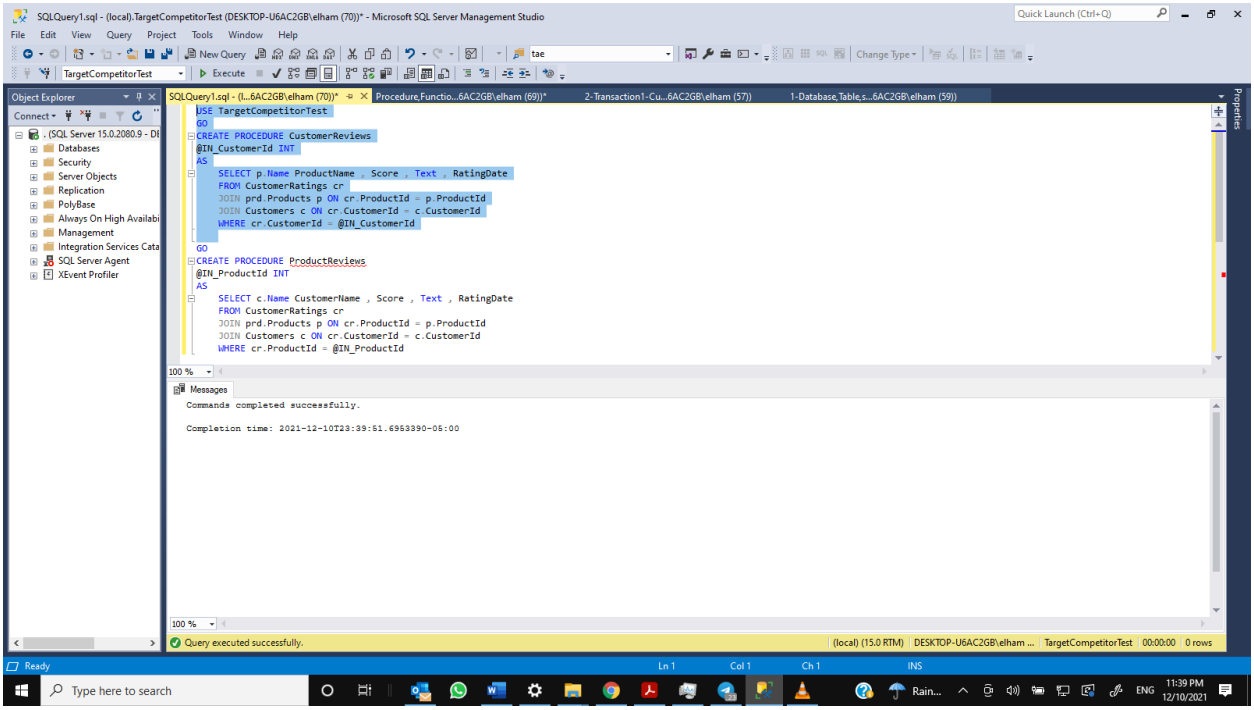


Comment: here we created the zip codes table to insert all the zip codes and the longitude and latitude to be able to calculate the distances.

Transactions & Stored procedures:

1)

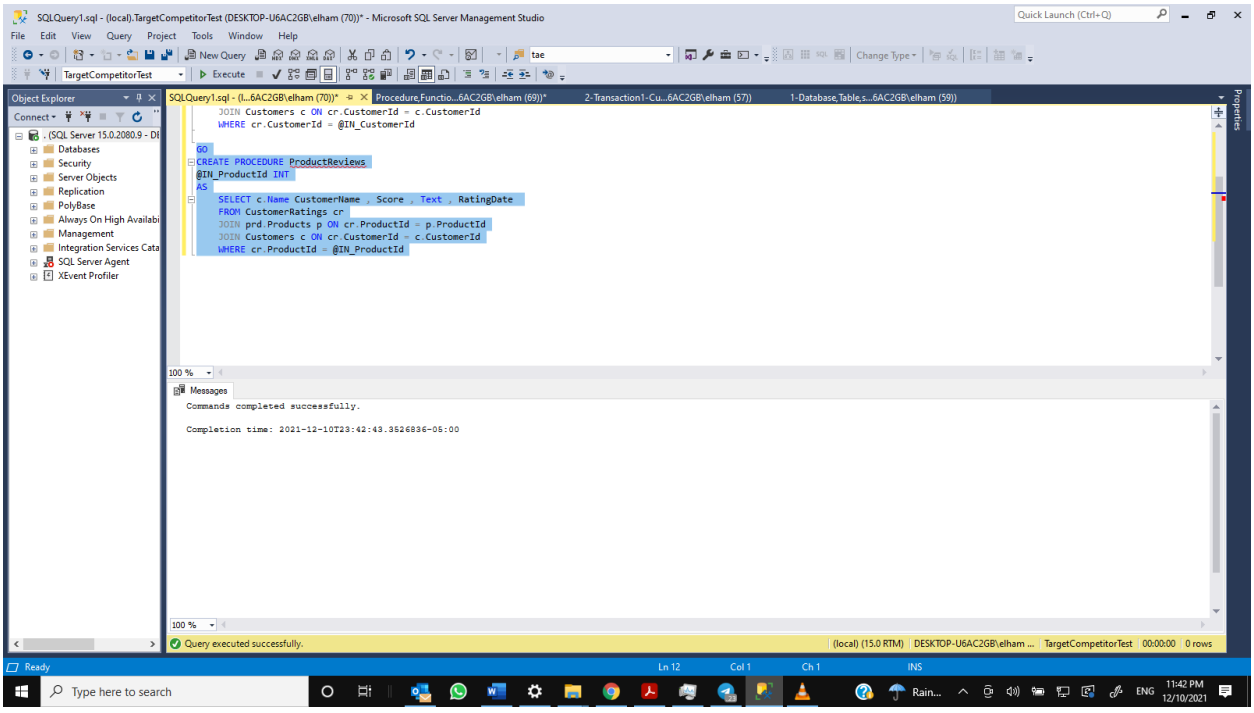
```
USE TargetCompetitorTest
GO
CREATE PROCEDURE CustomerReviews
@IN_CustomerId INT
AS
        SELECT p.Name ProductName , Score , Text , RatingDate
        FROM CustomerRatings cr
        JOIN prd.Products p ON cr.ProductId = p.ProductId
        JOIN Customers c ON cr.CustomerId = c.CustomerId
        WHERE cr.CustomerId = @IN_CustomerId
GO
```



Comment: here we created a stored procedure to show reviews for a specific customer.

2)

```
GO
CREATE PROCEDURE ProductReviews
@IN_ProductId INT
AS
        SELECT c.Name CustomerName , Score , Text , RatingDate
        FROM CustomerRatings cr
        JOIN prd.Products p ON cr.ProductId = p.ProductId
        JOIN Customers c ON cr.CustomerId = c.CustomerId
        WHERE cr.ProductId = @IN_ProductId
```



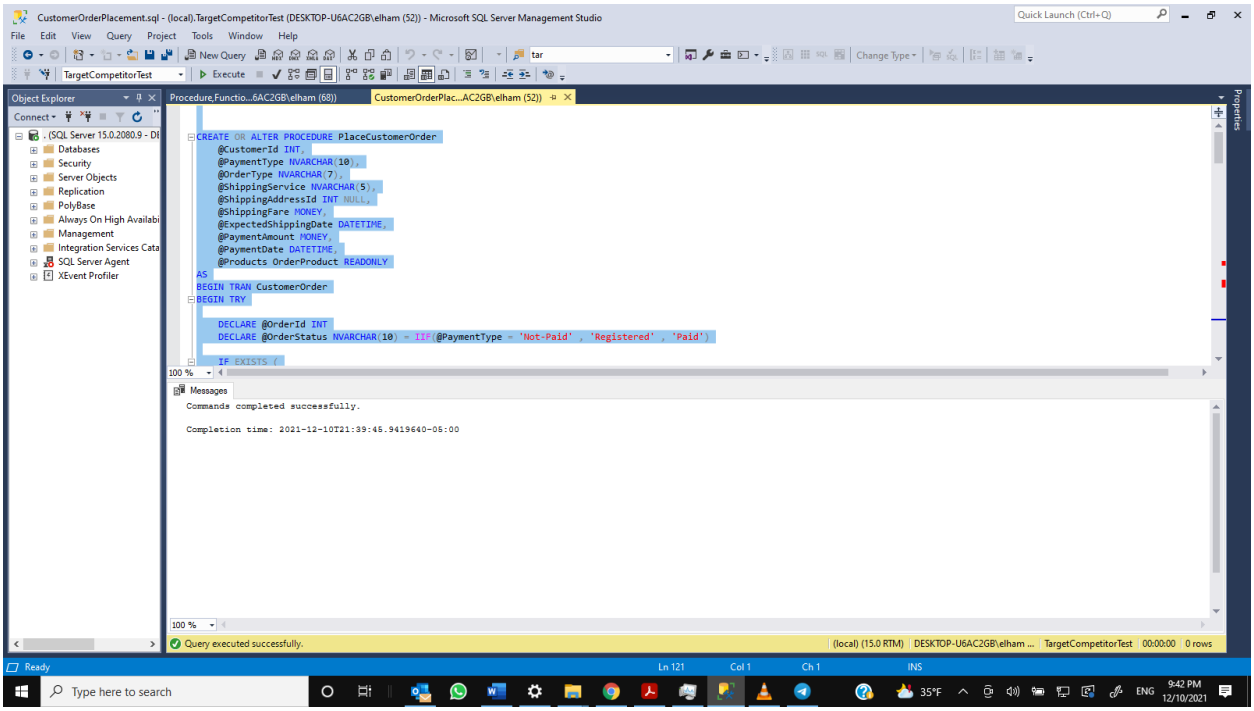
Comment: here we created a stored procedure to show product reviews.

3-4)

```
CREATE OR ALTER PROCEDURE PlaceCustomerOrder
	@CustomerId INT,
	@PaymentType NVARCHAR(10),
	@OrderType NVARCHAR(7),
	@ShippingService NVARCHAR(5),
	@ShippingAddressId INT NULL,
	@ShippingFare MONEY,
	@ExpectedShippingDate DATETIME,
	@PaymentAmount MONEY,
	@PaymentDate DATETIME,
	@Products OrderProduct READONLY
AS
BEGIN TRAN CustomerOrder
BEGIN TRY

	DECLARE @OrderId INT
	DECLARE @OrderStatus NVARCHAR(10) = IIF(@PaymentType = 'Not-Paid' , 'Registered' , 'Paid')

	IF EXISTS (
		SELECT * FROM @Products p1
		LEFT JOIN prd.Products p2 ON p1.ProductId = p2.ProductId
		WHERE p2.ProductId IS NULL
	)
		THROW 50001 , 'Invalid products ids detected' , 1;

	IF EXISTS (
		SELECT * FROM @Products p1
		LEFT JOIN prd.Products p2 ON p1.ProductId = p2.ProductId
		WHERE p1.Quantity = 0
	)
		THROW 50002 , 'Products quantity cannot be zero' , 1;

	IF @OrderType = 'Online' AND (@ShippingAddressId IS NULL OR @ShippingService IS NULL)
		THROW 50003 , 'Online orders should have shipping address and service specified' , 1;

	DECLARE @NotEnough NVARCHAR(MAX) = 'Not enough product inventory ' + (SELECT
STRING_AGG(inv , ', ') FROM (

	SELECT CAST(p.ProductId AS VARCHAR) + ' Inventory : ' + CAST(SUM(sp.Inventory) AS
VARCHAR) inv
		FROM inv.StoreProducts sp
		JOIN @Products p ON sp.ProductId = p.ProductId
		GROUP BY p.ProductId , p.Quantity
		HAVING SUM(sp.Inventory) < p.Quantity
	)a)

	IF @NotEnough IS NOT NULL
		THROW 50004 , @NotEnough  , 1;

	INSERT Orders
	VALUES(@OrderStatus , @OrderType , @PaymentType , @CustomerId , GETDATE())

	SET @OrderId = SCOPE_IDENTITY()

	INSERT OrderItems
	SELECT		@OrderId,
			p2.ProductId,
```

```
                        p1.Quantity,
                        IIF(cd.DiscountPercent IS NOT NULL,
                                p2.Price - p2.Price * cd.DiscountPercent / 100 ,
                                p2.Price),
                        IIF(cd.DiscountPercent IS NOT NULL,
                                p2.Price - p2.Price * cd.DiscountPercent / 100 ,
                                p2.Price) * p1.Quantity
        FROM @Products p1
        JOIN prd.Products p2 ON p1.ProductId = p2.ProductId
        LEFT JOIN CustomerDiscounts cd ON  p2.ProductId = cd.ProductId AND cd.CustomerId =
@CustomerId AND cd.ExpireDate < GETDATE()

        INSERT OrderPayments
        VALUES (@OrderId , @PaymentAmount , @PaymentDate)

        IF @OrderType = 'Online'
        BEGIN
                INSERT Shippings
                VALUES (@OrderId , @ShippingService , @ShippingAddressId , @ShippingFare,
@ExpectedShippingDate , NULL)
        END

        ;WITH StoreInventoryUpdate
        AS
        (
                SELECT
                        p.ProductId,
                        sp.StoreId,
                        sp.StoreProductId,
                        dbo.DistanceOfTwoPoint(sz.Latitude , sz.Longitude , az.Latitude , az.Longitude)
Distance,
                        p.Quantity CustomerRequestedQuantity,
                        sp.Inventory,
                        a.AddressBookId,
                        ISNULL(p.Quantity - SUM(sp.Inventory)
                                        OVER(
                                                PARTITION BY P.ProductId
                                                ORDER BY dbo.DistanceOfTwoPoint(sz.Latitude ,
sz.Longitude , az.Latitude , az.Longitude)
                                                ROWS BETWEEN UNBOUNDED PRECEDING AND 1
PRECEDING) , IIF(p.Quantity > sp.Inventory , sp.Inventory , p.Quantity)) RemainedRequestedInventory
                FROM @Products p
                JOIN inv.StoreProducts sp ON p.ProductId = sp.ProductId
                JOIN Stores s ON sp.StoreId = s.StoreId
                JOIN ZipCodes sz ON s.ZipCode = sz.ZipCode
                JOIN AddressBooks a ON 1 = 1
                JOIN ZipCodes az ON az.ZipCode = a.ZipCode
                WHERE a.AddressBookId = @ShippingAddressId
        )

                UPDATE StoreInventoryUpdate
                SET
                Inventory = IIF(CustomerRequestedQuantity < Inventory ,
                                                Inventory - CustomerRequestedQuantity ,
                                                IIF(RemainedRequestedInventory >= Inventory , 0 ,
Inventory - RemainedRequestedInventory)
                                        )
        WHERE RemainedRequestedInventory >= 0
        DELETE inv.StoreProducts
```

```
        WHERE Inventory = 0

        COMMIT TRAN CustomerOrder
END TRY
BEGIN CATCH
        ROLLBACK TRAN CustomerOrder
        PRINT 'Error occured'
        PRINT 'ErroNumber : ' + CAST(ERROR_NUMBER() AS NVARCHAR(MAX))
        PRINT 'Message : ' + ERROR_MESSAGE()
END CATCH
```



Comments: this procedure/transaction performs the following tasks

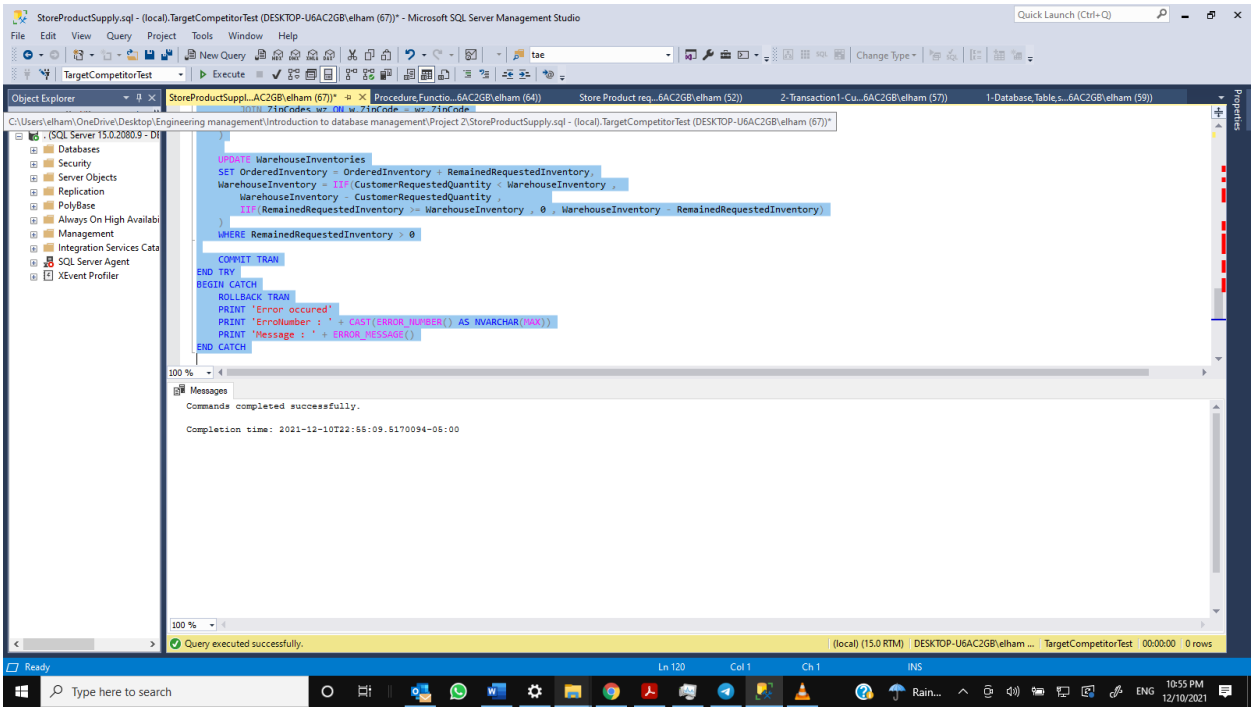
- ✓ Detection of various errors like invalid product id, products with a zero quantity, orders with no shipping address, and shipping services.
- ✓ Inserting orders, the related dates, payments, and the shipping information
- ✓ Calculation of the price and the applied discounts if there are any.
- ✓ Updating the inventory of stores (by using a CTE)
- ✓ Deleting the products which have no inventory value.

5-6)

```
USE TargetCompetitorTest
GO
CREATE OR ALTER PROCEDURE DeliverProductToStoreFromWarehouse
@PaymentType NVARCHAR(10),
@PaymentAmount MONEY,
@PaymentDate DATETIME,
@SupplierId INT,
@DemandingUserId INT,
@Products WarehouseProduct READONLY
AS
BEGIN TRAN
BEGIN TRY
	DECLARE @InvoiceId INT


	IF EXISTS (
		SELECT * FROM @Products p1
		LEFT JOIN Products p2 ON p1.ProductId = p2.ProductId
		WHERE p2.ProductId IS NULL
	)
		THROW 50001 , 'Invalid products ids detected' , 1;

	IF EXISTS (
		SELECT * FROM @Products p1
		LEFT JOIN Products p2 ON p1.ProductId = p2.ProductId
		WHERE p1.Quantity = 0
	)
		THROW 50002 , 'Products quantity cannot be zero' , 1;

	DECLARE @NotEnough NVARCHAR(MAX) = 'Not enough product inventory ' + (SELECT
STRING_AGG(inv , ', ') FROM (

	SELECT CAST(p.ProductId AS VARCHAR) + ' Inventory : ' + CAST(SUM(wp.Inventory) AS
VARCHAR) inv
		FROM WarehouseProducts wp
		JOIN @Products p ON wp.ProductId = p.ProductId
		JOIN Warehouses w ON wp.WarehouseId = w.WarehouseId
		WHERE w.SupplierId = @SupplierId
		GROUP BY p.ProductId , p.Quantity
		HAVING SUM(wp.Inventory) < p.Quantity
	)a)

	IF @NotEnough IS NOT NULL
		THROW 50004 , @NotEnough  , 1;

	INSERT Invoices (SupplierId , DemandingUserId, InvoiceDate)
	VALUES(@SupplierId , @DemandingUserId , GETDATE())

	SET @InvoiceId = SCOPE_IDENTITY()

	INSERT InvoiceItems
	SELECT		@InvoiceId,
			p1.ProductId,
			p1.Quantity,
			p2.Price,
			p1.Quantity * p2.Price
	FROM @Products p1
```

```
		JOIN Products p2 ON p1.ProductId = p2.ProductId

		INSERT InvoicePayments
		VALUES(@InvoiceId , @PaymentAmount , @PaymentDate)

		MERGE INTO StoreProducts sp
		USING(
			SELECT ip1.ProductId , ip1.Quantity , ip1.StoreId , ip2.Price
			FROM @Products ip1
			JOIN Products ip2 ON ip1.ProductId = ip2.ProductId
		) p (ProductId , Quantity , StoreId, Price)
		ON sp.ProductId = p.ProductId AND sp.StoreId = p.StoreId
		WHEN MATCHED THEN
			UPDATE SET sp.Inventory = sp.Inventory + p.Quantity
		WHEN NOT MATCHED THEN
			INSERT(StoreId , ProductId , Price , Inventory)
			VALUES(p.StoreId , p.ProductId , p.Price , p.Quantity);

		;WITH WarehouseInventories
		AS(
			SELECT		p.ProductId,
					s.StoreId,
					sp.StoreProductId,
					dbo.DistanceOfTwoPoint(sz.Latitude , sz.Longitude , wz.Latitude ,
wz.Longitude) Distance,
					p.Quantity CustomerRequestedQuantity,
					wp.Inventory WarehouseInventory,
					wp.OrderedInventory,
					ISNULL(p.Quantity - SUM(wp.Inventory)
							OVER(
								PARTITION BY P.ProductId
								ORDER BY dbo.DistanceOfTwoPoint(sz.Latitude ,
sz.Longitude , wz.Latitude , wz.Longitude)
								ROWS BETWEEN UNBOUNDED PRECEDING
AND 1 PRECEDING) , IIF(p.Quantity > wp.Inventory , wp.Inventory , p.Quantity))
RemainedRequestedInventory
			FROM @Products p
			JOIN WarehouseProducts wp ON p.ProductId = wp.ProductId AND wp.Inventory != 0
			JOIN Stores s ON p.StoreId = s.StoreId
			JOIN StoreSuppliers ss ON s.StoreId = ss.StoreId
			JOIN Warehouses w ON wp.WarehouseId = w.WarehouseId AND w.SupplierId =
ss.SupplierId
			LEFT JOIN StoreProducts sp ON sp.ProductId = wp.ProductId AND sp.StoreId = p.StoreId
			JOIN ZipCodes wz ON w.ZipCode = wz.ZipCode
			JOIN ZipCodes sz ON sz.ZipCode = s.ZipCode
		)

		UPDATE WarehouseInventories
		SET OrderedInventory = OrderedInventory + RemainedRequestedInventory,
		WarehouseInventory = IIF(CustomerRequestedQuantity < WarehouseInventory ,
			WarehouseInventory - CustomerRequestedQuantity ,
			IIF(RemainedRequestedInventory >= WarehouseInventory , 0 , WarehouseInventory -
RemainedRequestedInventory)
		)
		WHERE RemainedRequestedInventory > 0

		COMMIT TRAN
END TRY
BEGIN CATCH
```

```
        ROLLBACK TRAN
        PRINT 'Error occured'
        PRINT 'ErroNumber : ' + CAST(ERROR_NUMBER() AS NVARCHAR(MAX))
        PRINT 'Message : ' + ERROR_MESSAGE()
END CATCH
```

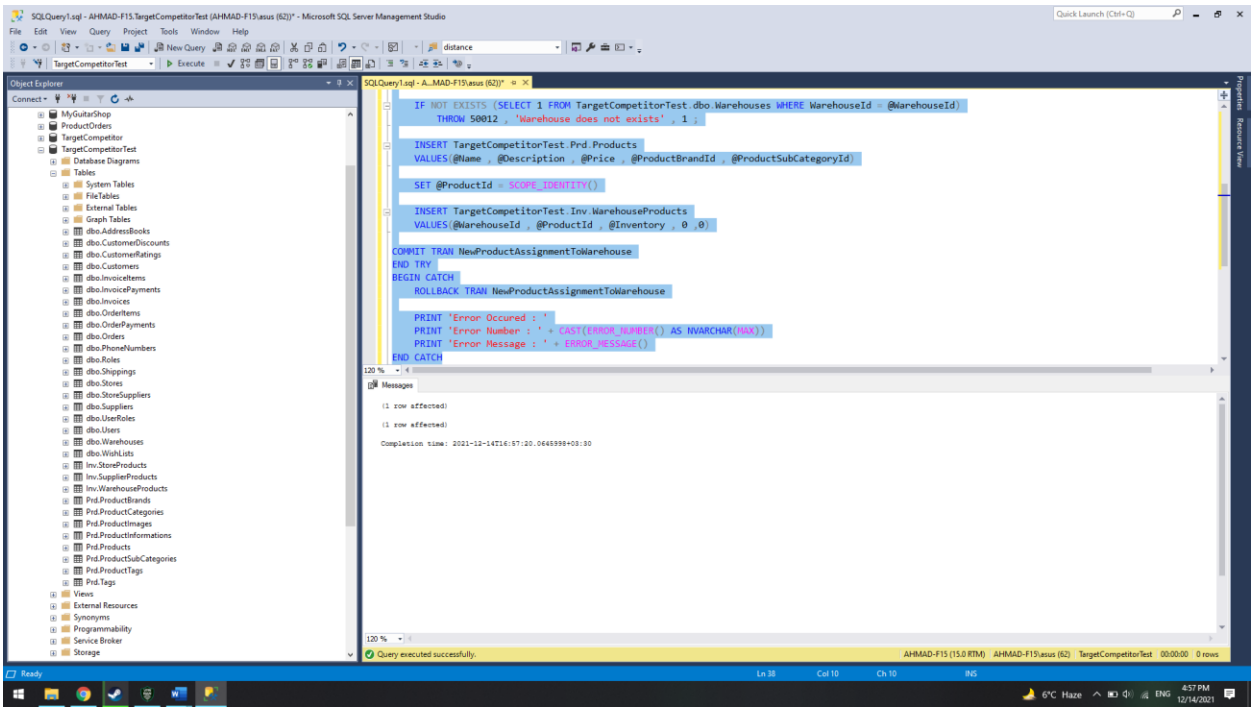


Comment: This transaction/procedure performs the following tasks.

- ✓ Inserting data into invoices like supplier information, user information and invoice date.
- ✓ Updating store product if there is existing inventory.
- ✓ Inserting store product if there is no existing inventory
- ✓ Reducing inventory of warehouse which supplied the product.

7)

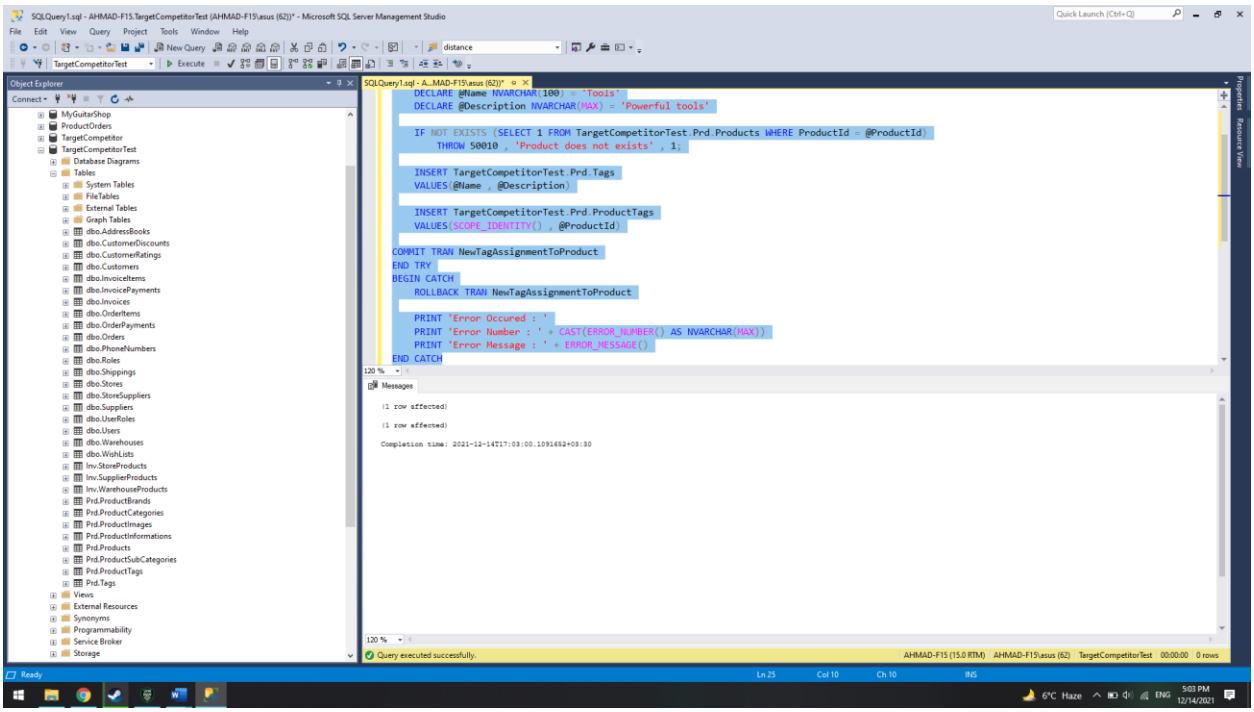
```
BEGIN TRY
BEGIN TRAN NewProductAssignmentToWarehouse

	DECLARE @ProductBrandId INT = 3
	DECLARE @ProductSubCategoryId INT = 40
	DECLARE @Name NVARCHAR(100) = 'Cat food'
	DECLARE @Description NVARCHAR(250) = NULL
	DECLARE @Price MONEY = 50
	DECLARE @WarehouseId INT = 1
	DECLARE @Inventory INT = 10
	DECLARE @ProductId INT

	IF NOT EXISTS (SELECT 1 FROM TargetCompetitorTest.Prd.ProductBrands WHERE
ProductBrandId = @ProductBrandId)
		THROW 50010 , 'Product brand does not exists' , 1;

	IF NOT EXISTS (SELECT 1 FROM TargetCompetitorTest.Prd.ProductSubCategories WHERE
ProductSubCategoryId = @ProductSubCategoryId)
		THROW 50011 , 'Product sub cateogry does not exists' , 1;

	IF NOT EXISTS (SELECT 1 FROM TargetCompetitorTest.dbo.Warehouses WHERE WarehouseId =
@WarehouseId)
		THROW 50012 , 'Warehouse does not exists' , 1 ;

	INSERT TargetCompetitorTest.Prd.Products
	VALUES(@Name , @Description , @Price , @ProductBrandId , @ProductSubCategoryId)

	SET @ProductId = SCOPE_IDENTITY()

	INSERT TargetCompetitorTest.Inv.WarehouseProducts
	VALUES(@WarehouseId , @ProductId , @Inventory , 0 ,0)

COMMIT TRAN NewProductAssignmentToWarehouse
END TRY
BEGIN CATCH
	ROLLBACK TRAN NewProductAssignmentToWarehouse

	PRINT 'Error Occured : '
	PRINT 'Error Number : ' + CAST(ERROR_NUMBER() AS NVARCHAR(MAX))
	PRINT 'Error Message : ' + ERROR_MESSAGE()
END CATCH
```



Comment: here we created a transaction that adds a new product to warehouse.

8)

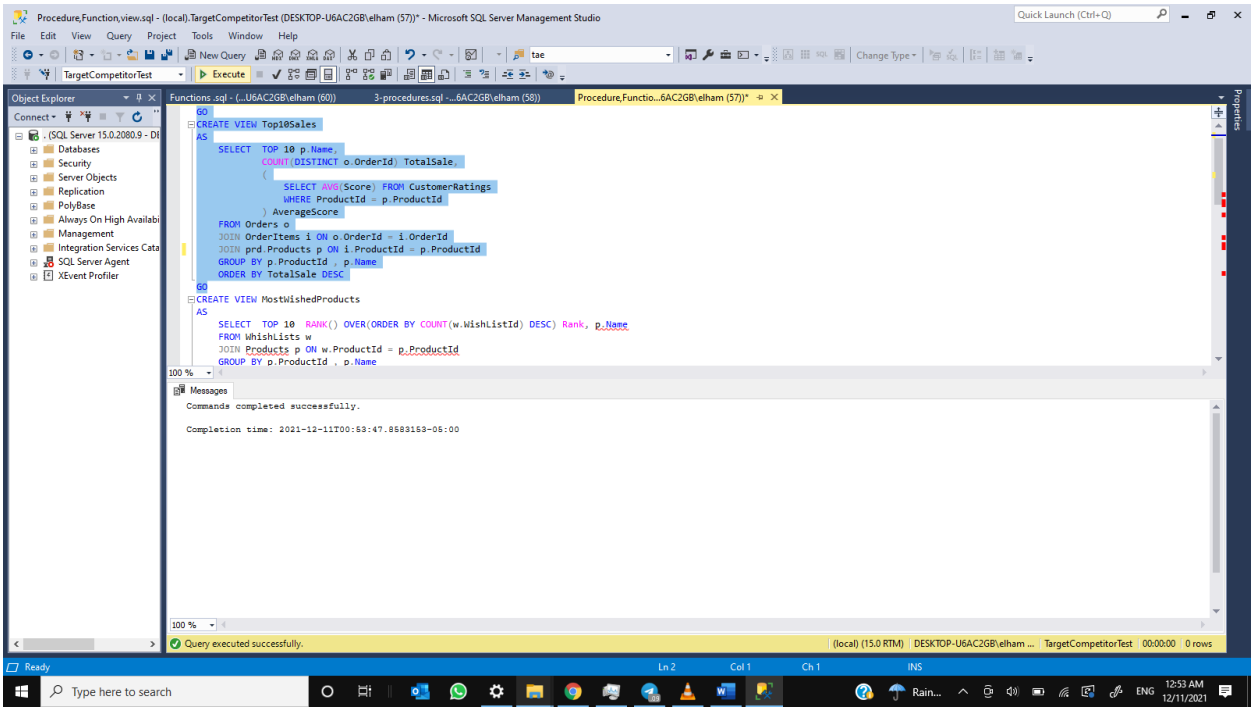
```
BEGIN TRY
BEGIN TRAN NewTagAssignmentToProduct

        DECLARE @ProductId INT = 104
        DECLARE @Name NVARCHAR(100) = 'Tools'
        DECLARE @Description NVARCHAR(MAX) = 'Powerful tools'

        IF NOT EXISTS (SELECT 1 FROM TargetCompetitorTest.Prd.Products WHERE ProductId =
@ProductId)
                THROW 50010 , 'Product does not exists' , 1;

        INSERT TargetCompetitorTest.Prd.Tags
        VALUES(@Name , @Description)

        INSERT TargetCompetitorTest.Prd.ProductTags
        VALUES(SCOPE_IDENTITY() , @ProductId)

COMMIT TRAN NewTagAssignmentToProduct
END TRY
BEGIN CATCH
        ROLLBACK TRAN NewTagAssignmentToProduct

        PRINT 'Error Occured : '
        PRINT 'Error Number : ' + CAST(ERROR_NUMBER() AS NVARCHAR(MAX))
        PRINT 'Error Message : ' + ERROR_MESSAGE()
END CATCH
```



Comment: This transaction assign new tags to products.

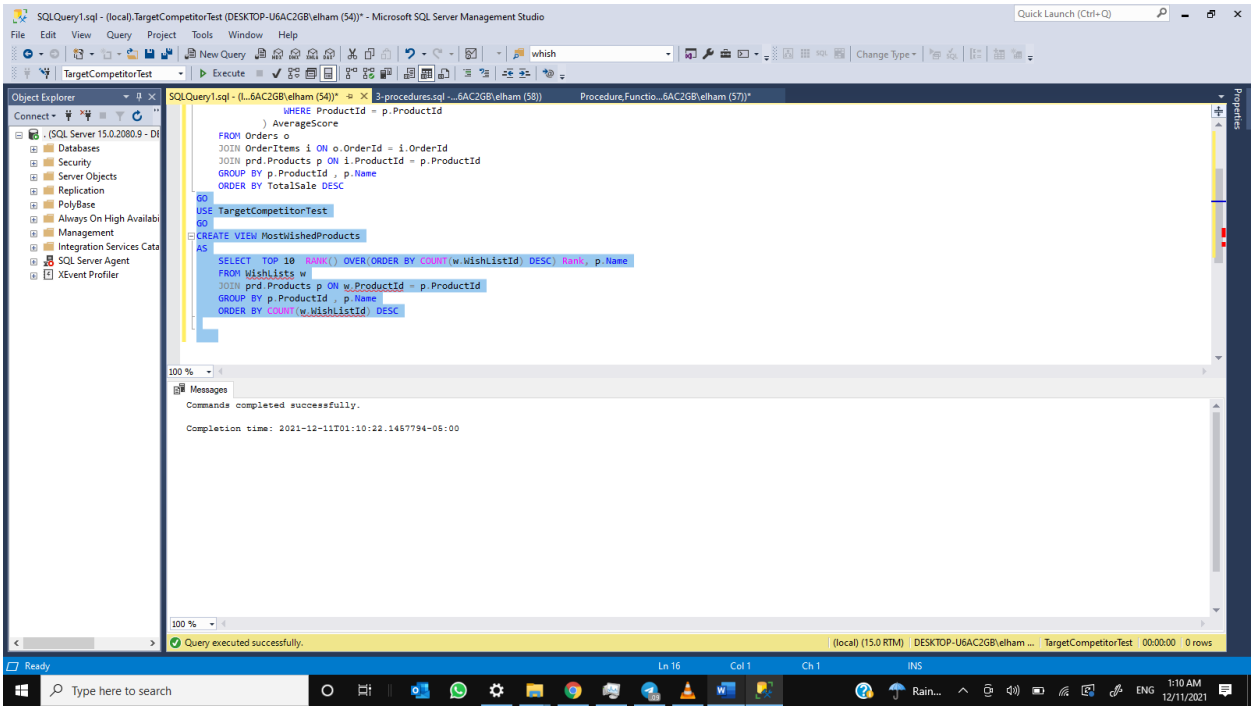
Views :

```
USE TargetCompetitorTest
GO
CREATE VIEW Top10Sales
AS
	SELECT		TOP 10 p.Name,
			COUNT(DISTINCT o.OrderId) TotalSale,
			(
				SELECT AVG(Score) FROM CustomerRatings
				WHERE ProductId = p.ProductId
			) AverageScore
		FROM Orders o
		JOIN OrderItems i ON o.OrderId = i.OrderId
		JOIN prd.Products p ON i.ProductId = p.ProductId
		GROUP BY p.ProductId , p.Name
		ORDER BY TotalSale DESC
```



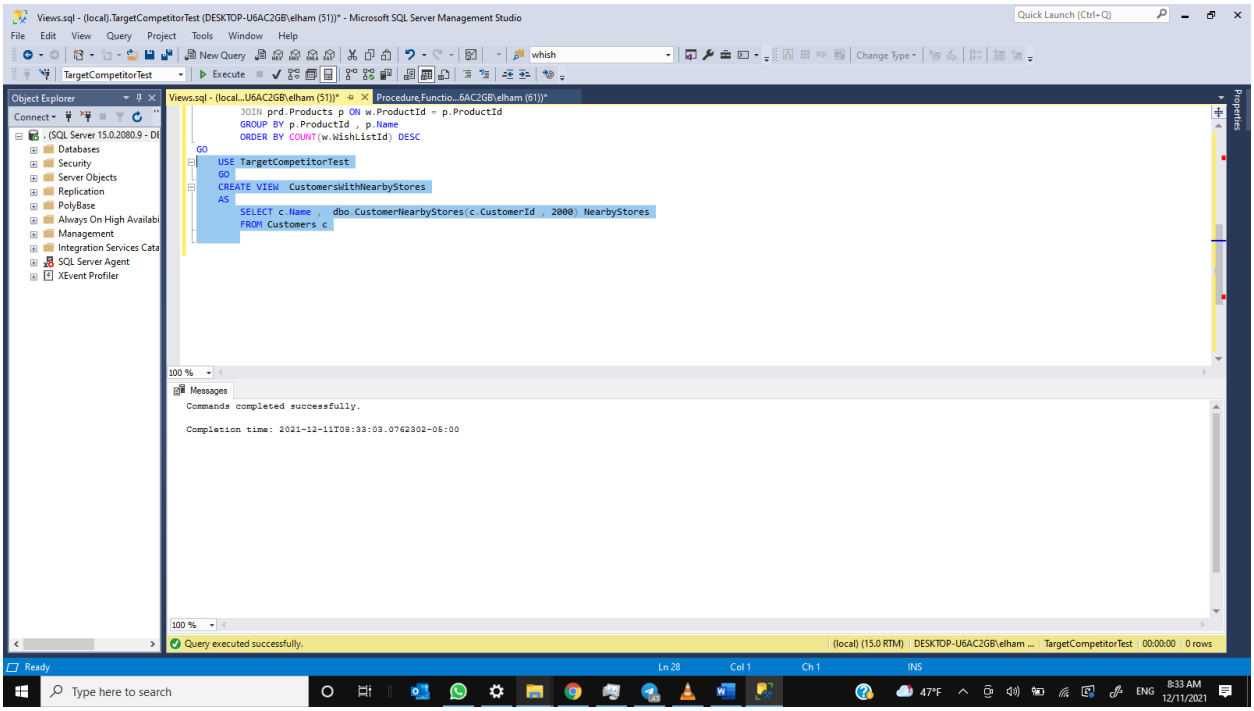
Comment: This view shows top 10 product regarding their total sales.

```
GO
USE TargetCompetitorTest
GO
CREATE VIEW MostWishedProducts
AS
        SELECT      TOP 10  RANK() OVER(ORDER BY COUNT(w.WishListId) DESC) Rank, p.Name
        FROM WishLists w
        JOIN prd.Products p ON w.ProductId = p.ProductId
        GROUP BY p.ProductId , p.Name
        ORDER BY COUNT(w.WishListId) DESC
```



Comment: This view shows the most wished products.

```
USE TargetCompetitorTest
GO
CREATE VIEW  CustomersWithNearbyStores
AS
        SELECT c.Name ,  dbo.CustomerNearbyStores(c.CustomerId , 2000) NearbyStores
        FROM Customers c
```

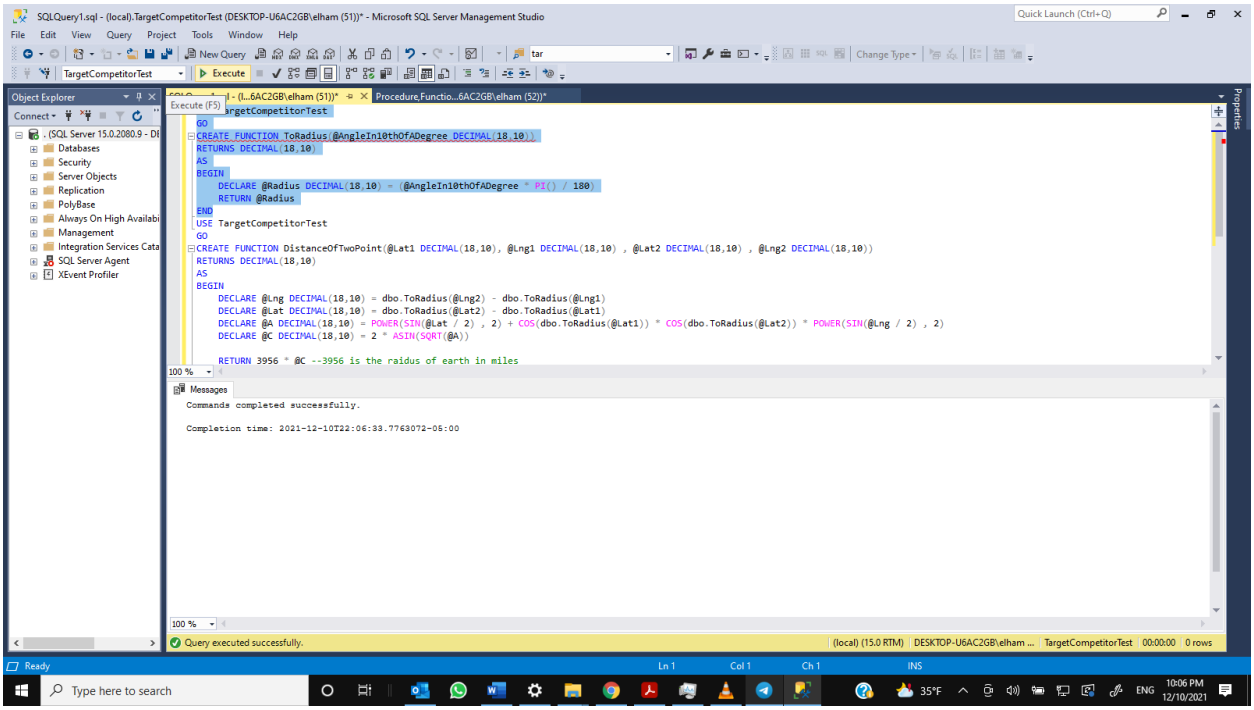


Comment: this view use CustomerNearbyStores to show the stores near the customers

Functions :

1)

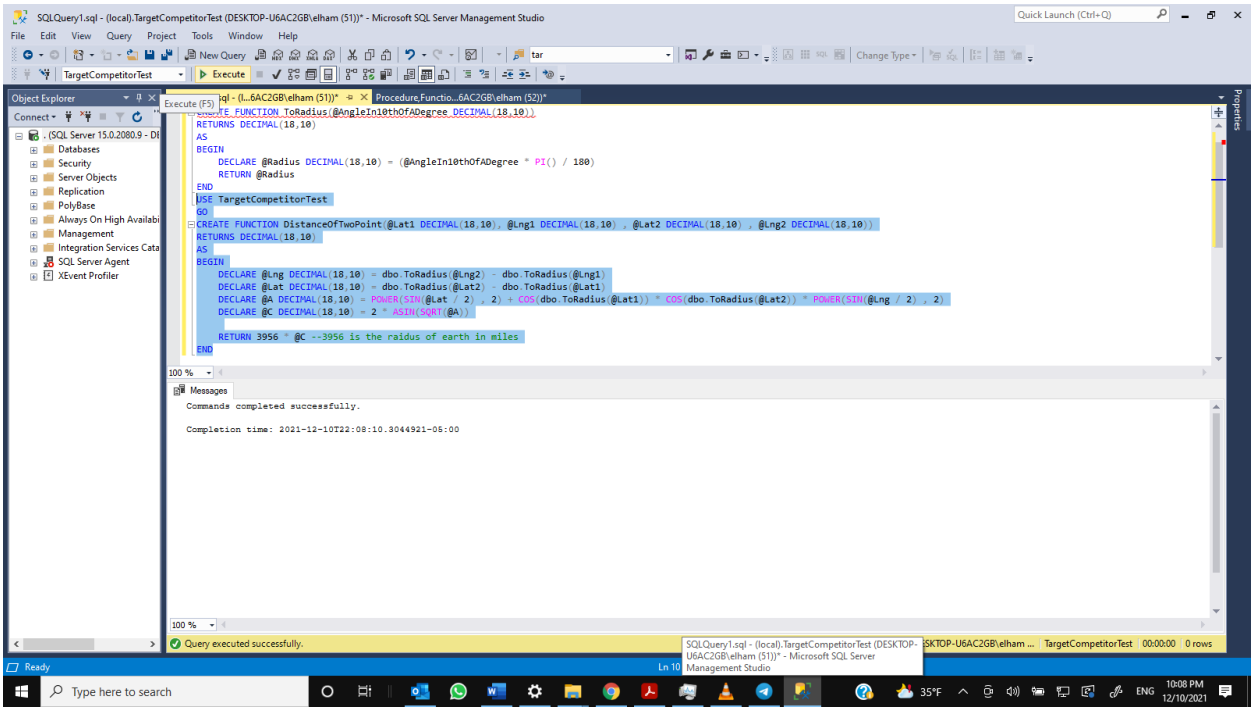
```
USE TargetCompetitorTest
GO
CREATE FUNCTION ToRadius(@AngleIn10thOfADegree DECIMAL(18,10))
RETURNS DECIMAL(18,10)
AS
BEGIN
        DECLARE @Radius DECIMAL(18,10) = (@AngleIn10thOfADegree * PI() / 180)
        RETURN @Radius
END
```



Comment: the output of this function will be used to calculate the distance between two points on earth.

2)

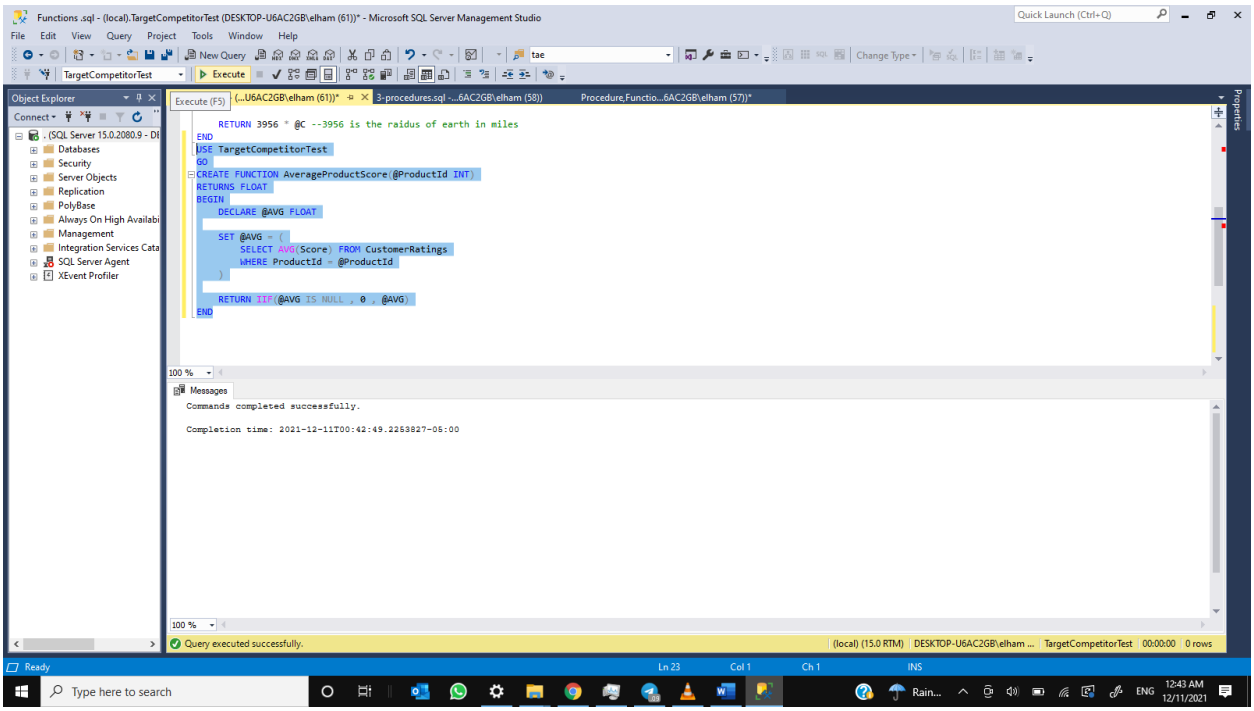
```
USE TargetCompetitorTest
GO
CREATE FUNCTION DistanceOfTwoPoint(@Lat1 DECIMAL(18,10), @Lng1 DECIMAL(18,10) , @Lat2
DECIMAL(18,10) , @Lng2 DECIMAL(18,10))
RETURNS DECIMAL(18,10)
AS
BEGIN
        DECLARE @Lng DECIMAL(18,10) = dbo.ToRadius(@Lng2) - dbo.ToRadius(@Lng1)
        DECLARE @Lat DECIMAL(18,10) = dbo.ToRadius(@Lat2) - dbo.ToRadius(@Lat1)
        DECLARE @A DECIMAL(18,10) = POWER(SIN(@Lat / 2) , 2) + COS(dbo.ToRadius(@Lat1)) *
COS(dbo.ToRadius(@Lat2)) * POWER(SIN(@Lng / 2) , 2)
        DECLARE @C DECIMAL(18,10) = 2 * ASIN(SQRT(@A))

        RETURN 3956 * @C --3956 is the raidus of earth in miles
END
```



Comment: This function calculates the distance of two points which will be useful to find the nearest store to ship from.

3)

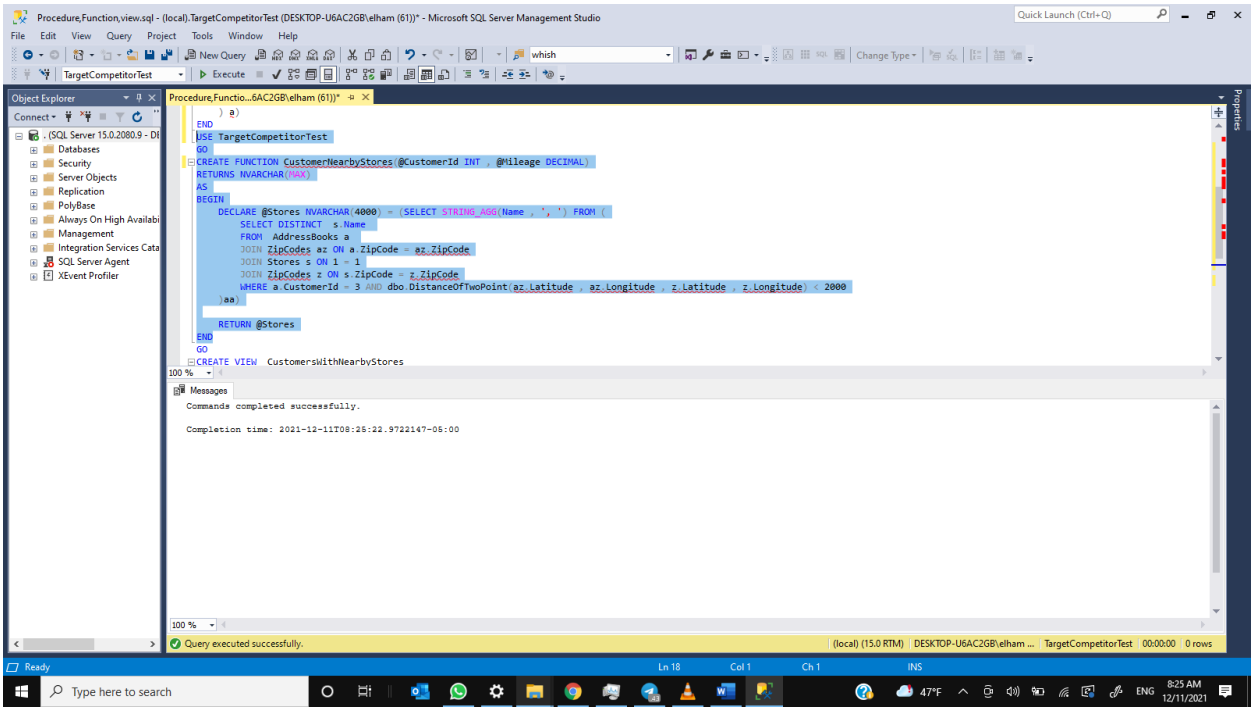
```
USE TargetCompetitorTest
GO
CREATE FUNCTION AverageProductScore(@ProductId INT)
RETURNS DECIMAL(4,2)
BEGIN
	DECLARE @AVG DECIMAL(4,2)

	SET @AVG = (
		SELECT AVG(Score) FROM CustomerRatings
		WHERE ProductId = @ProductId
	)

	RETURN IIF(@AVG IS NULL , 0 , @AVG)
END
```



Comment: This function calculates the average product scores.

4)

```
USE TargetCompetitorTest
GO
CREATE FUNCTION CustomerNearbyStores(@CustomerId INT , @Mileage DECIMAL)
RETURNS NVARCHAR(MAX)
AS
BEGIN
	DECLARE @Stores NVARCHAR(4000) = (SELECT STRING_AGG(Name , ', ') FROM (
		SELECT DISTINCT  s.Name
		FROM  AddressBooks a
		JOIN ZipCodes az ON a.ZipCode = az.ZipCode
		JOIN Stores s ON 1 = 1
		JOIN ZipCodes z ON s.ZipCode = z.ZipCode
		WHERE a.CustomerId = 3 AND dbo.DistanceOfTwoPoint(az.Latitude , az.Longitude ,
z.Latitude , z.Longitude) < 2000
	)aa)

	RETURN @Stores
END
```

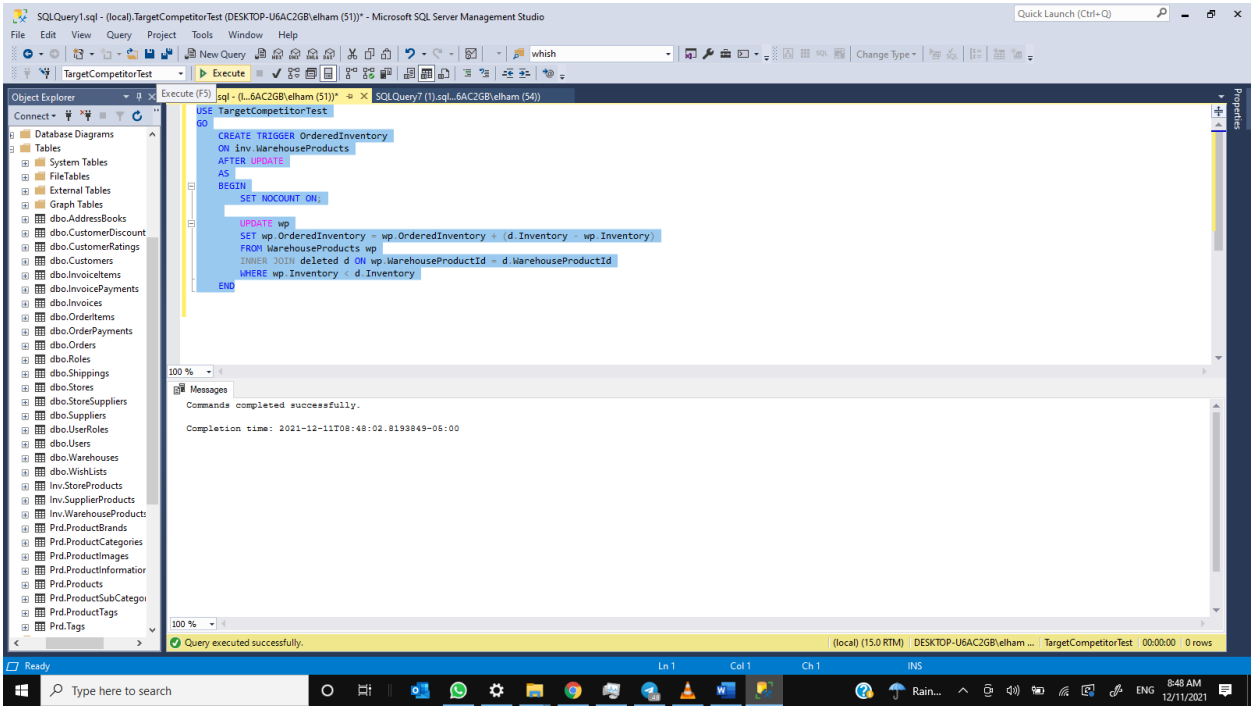


Comment: output of this function will be used in one of the views to show stores near customers.

Triggers:

1)

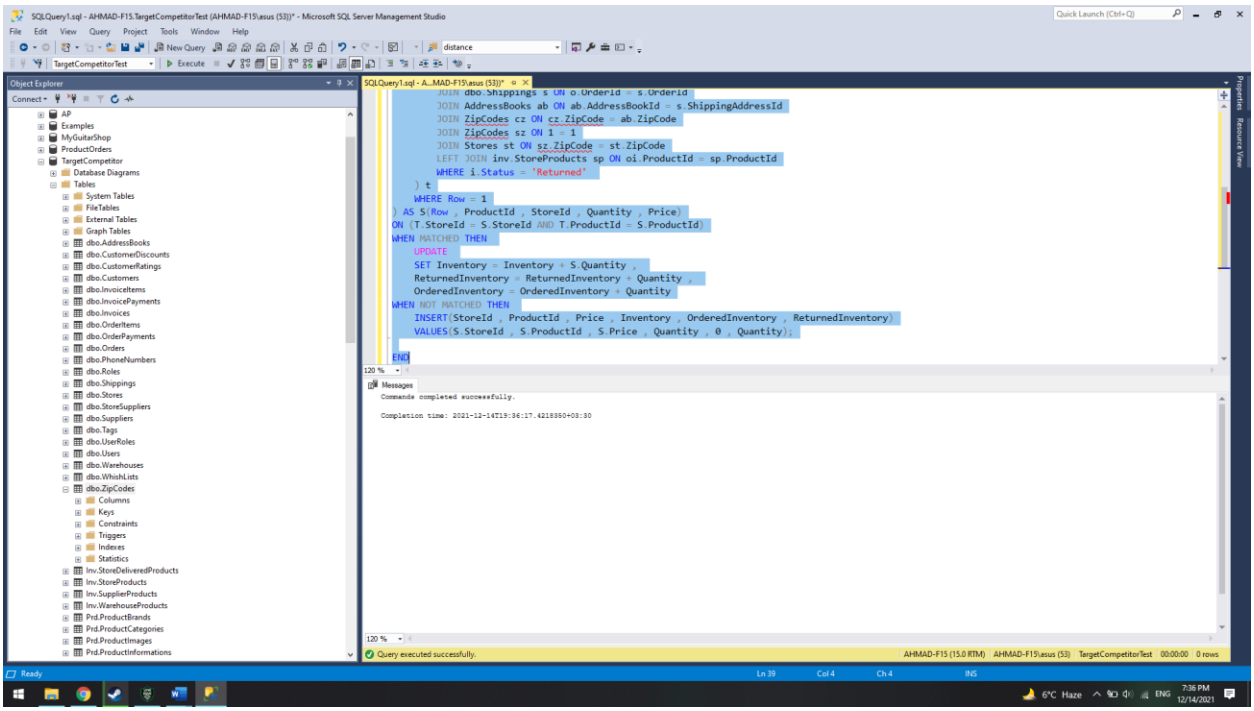
```
USE TargetCompetitorTest
GO
	CREATE TRIGGER  OrderedInventory
	ON inv.WarehouseProducts
	AFTER UPDATE
	AS
	BEGIN
		SET NOCOUNT ON;

		UPDATE wp
		SET wp.OrderedInventory = wp.OrderedInventory + (d.Inventory - wp.Inventory)
		FROM WarehouseProducts wp
		INNER JOIN deleted d ON wp.WarehouseProductId = d.WarehouseProductId
		WHERE wp.Inventory < d.Inventory
	END
```



Comment: This trigger updates the warehouse product table column named ordered inventory. This will be useful for the times one of the products them has been ordered

2)

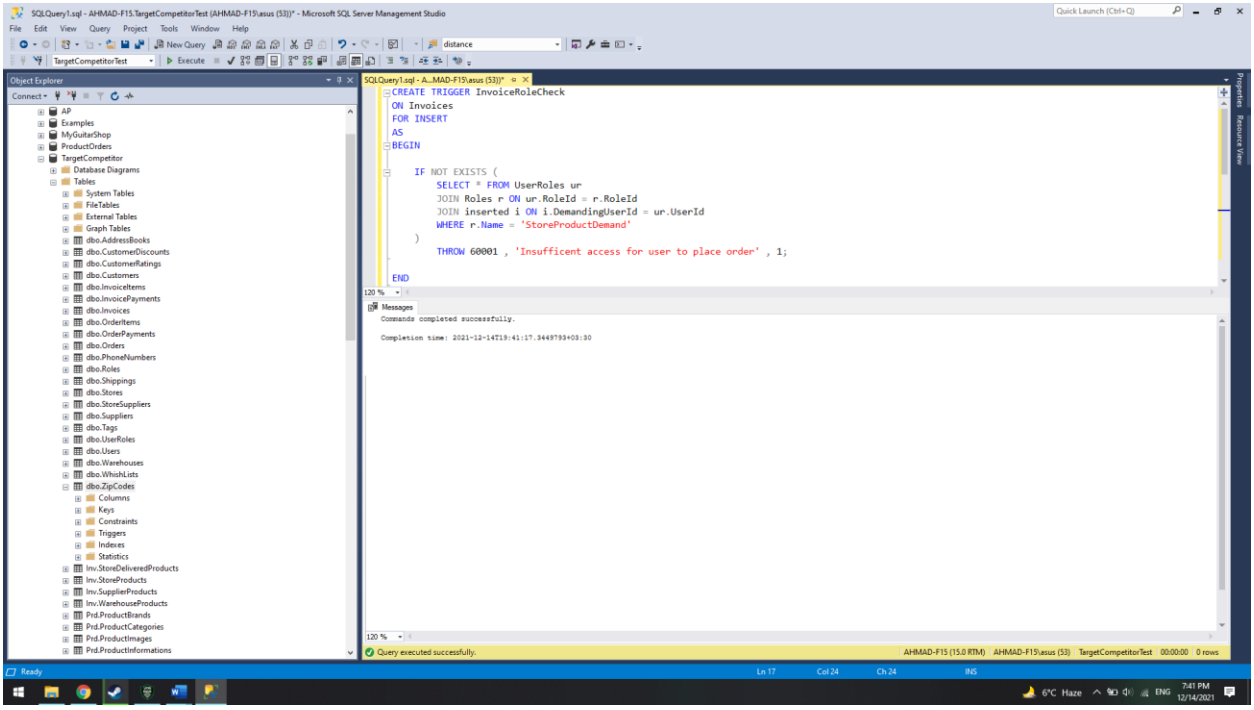
```
USE TargetCompetitorTest
GO
CREATE TRIGGER ReturnedOrderInventoryFix
ON TargetCompetitorTest.dbo.Orders
AFTER UPDATE
AS
BEGIN

MERGE inv.StoreProducts AS T
USING (
	SELECT Row ,ProductId,StoreId , Quantity , UnitPrice FROM (
		SELECT ROW_NUMBER() OVER(
			PARTITION BY oi.ProductId
			ORDER BY dbo.DistanceOfTwoPoint(cz.Latitude , cz.Longitude , sz.Latitude ,
sz.Longitude)
		) Row , oi.ProductId , st.StoreId , oi.Quantity , oi.UnitPrice
		FROM dbo.Orders o
		JOIN Inserted i ON o.OrderId = i.OrderId
		JOIN dbo.OrderItems oi ON o.OrderId = oi.OrderId
		JOIN dbo.Shippings s ON o.OrderId = s.OrderId
		JOIN AddressBooks ab ON ab.AddressBookId = s.ShippingAddressId
		JOIN ZipCodes cz ON cz.ZipCode = ab.ZipCode
		JOIN ZipCodes sz ON 1 = 1
		JOIN Stores st ON sz.ZipCode = st.ZipCode
		LEFT JOIN inv.StoreProducts sp ON oi.ProductId = sp.ProductId
		WHERE i.Status = 'Returned'
	) t
	WHERE Row = 1
) AS S(Row , ProductId , StoreId , Quantity , Price)
ON (T.StoreId = S.StoreId AND T.ProductId = S.ProductId)
WHEN MATCHED THEN
	UPDATE
	SET Inventory = Inventory + S.Quantity ,
	ReturnedInventory = ReturnedInventory + Quantity ,
	OrderedInventory = OrderedInventory + Quantity
WHEN NOT MATCHED THEN
	INSERT(StoreId , ProductId , Price , Inventory , OrderedInventory , ReturnedInventory)
	VALUES(S.StoreId , S.ProductId , S.Price , Quantity , 0 , Quantity);

END
```



Comment: Here we created a trigger that updates the inventory when a product has got returned. This trigger finds the nearest store and increases its inventory

3)

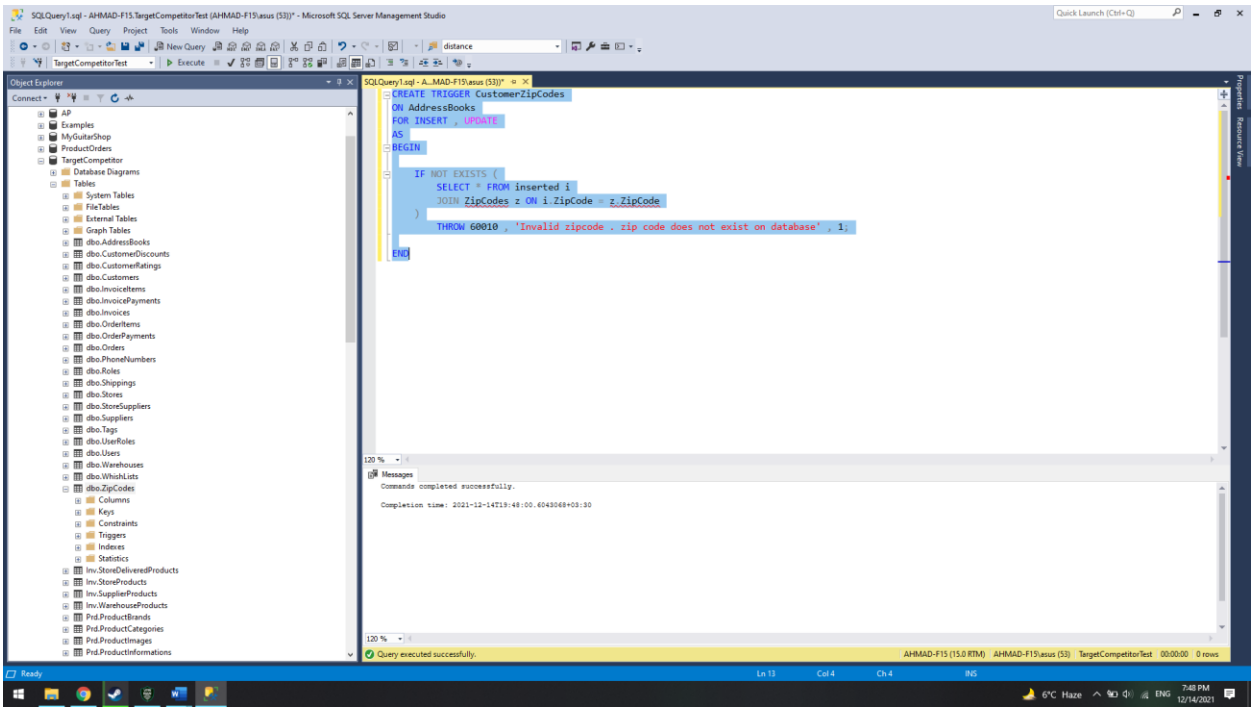
```
CREATE TRIGGER  InvoiceRoleCheck
ON Invoices
FOR INSERT
AS
BEGIN

        IF NOT EXISTS (
                SELECT * FROM  UserRoles ur
                JOIN Roles r ON ur.RoleId = r.RoleId
                JOIN inserted i ON i.DemandingUserId = ur.UserId
                WHERE r.Name = 'StoreProductDemand'
        )
                THROW 60001 , 'Insufficent access for user to place order' , 1;

END
```



Comment: This trigger checks the if the user is authorized to insert data into invoices and throws an error when there is no enough access.

4)

```
CREATE TRIGGER CustomerZipCodes
ON AddressBooks
FOR INSERT , UPDATE
AS
BEGIN

        IF NOT EXISTS (
                SELECT * FROM inserted i
                JOIN ZipCodes z ON i.ZipCode = z.ZipCode
        )
                THROW 60010 , 'Invalid zipcode . zip code does not exist on database' , 1;

END
```

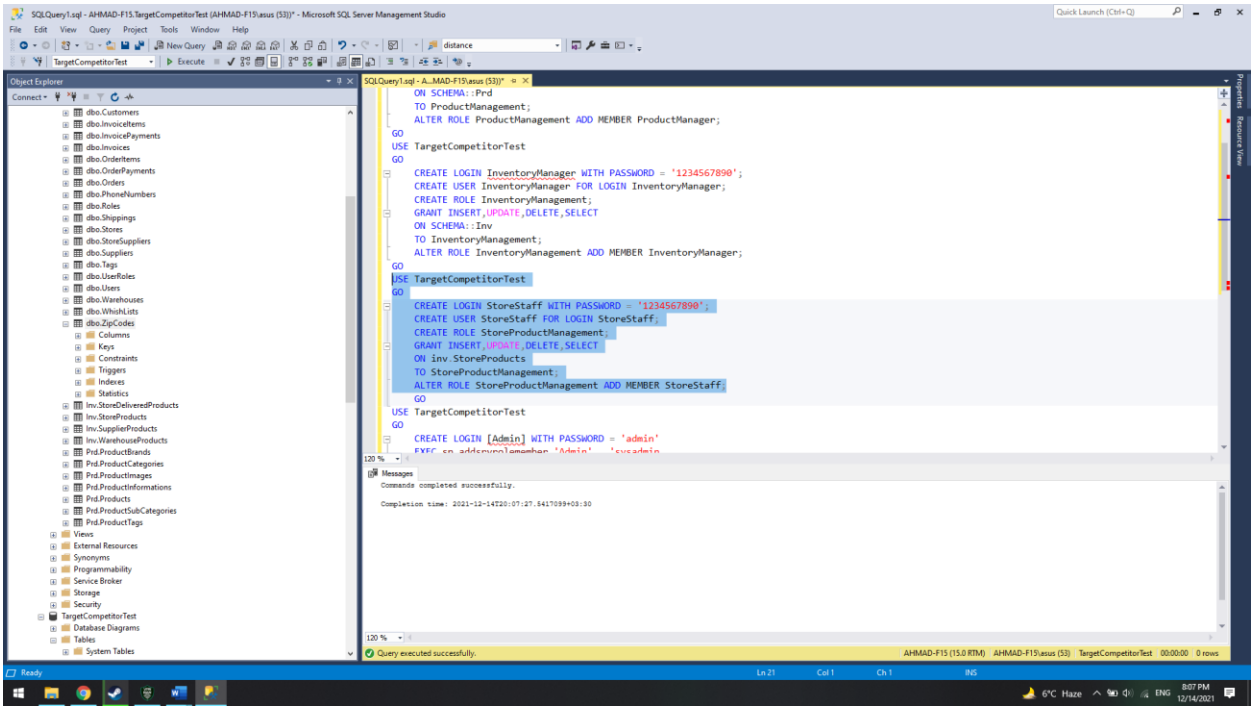


Comment: This trigger fires and then throws an error if an invalid zip code is entered into database.

Scripts to create users with various security levels, passwords, and roles

1)

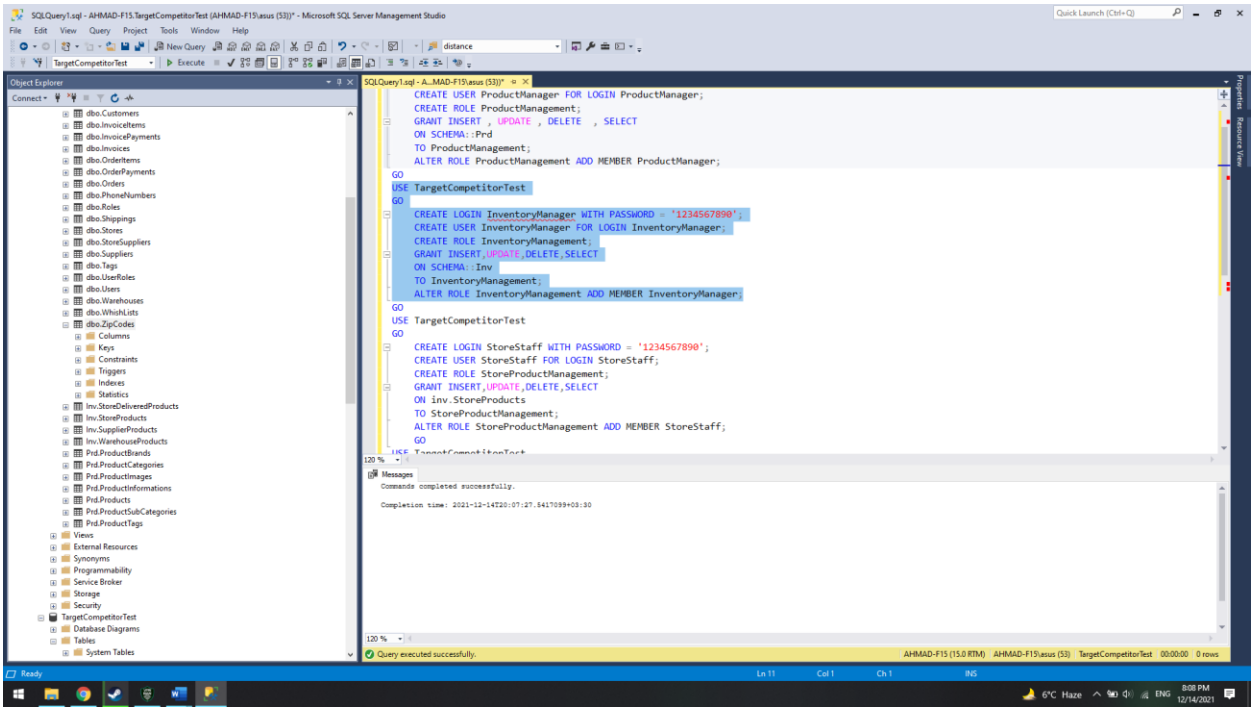
```
USE TargetCompetitorTest
GO
        CREATE LOGIN StoreStaff WITH PASSWORD = '1234567890';
        CREATE USER StoreStaff FOR LOGIN StoreStaff;
        CREATE ROLE StoreProductManagement;
        GRANT INSERT,UPDATE,DELETE,SELECT
        ON inv.StoreProducts
        TO StoreProductManagement;
        ALTER ROLE StoreProductManagement ADD MEMBER StoreStaff;
```



Comment: Here we wrote a script that allows Store staff to perform DML on store products table.

2)

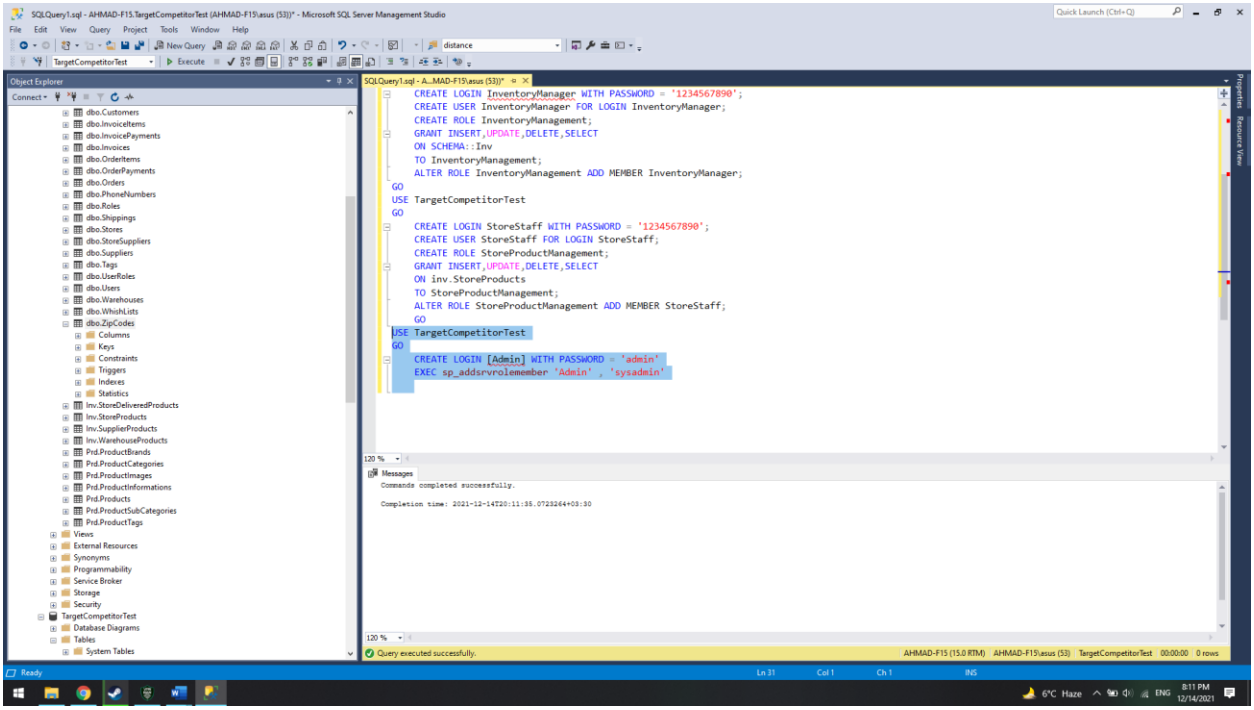
```
USE TargetCompetitorTest
GO
        CREATE LOGIN InventoryManager WITH PASSWORD = '1234567890';
        CREATE USER InventoryManager FOR LOGIN InventoryManager;
        CREATE ROLE InventoryManagement;
        GRANT INSERT,UPDATE,DELETE,SELECT
        ON SCHEMA::Inv
        TO InventoryManagement;
        ALTER ROLE InventoryManagement ADD MEMBER InventoryManager;
```



Comment: Here we wrote a script that allows inventory manager to perform DML on INV schema.

3)

```
USE TargetCompetitorTest
GO
        CREATE LOGIN [Admin] WITH PASSWORD = 'admin'
        EXEC sp_addsrvrolemember 'Admin' , 'sysadmin'
```

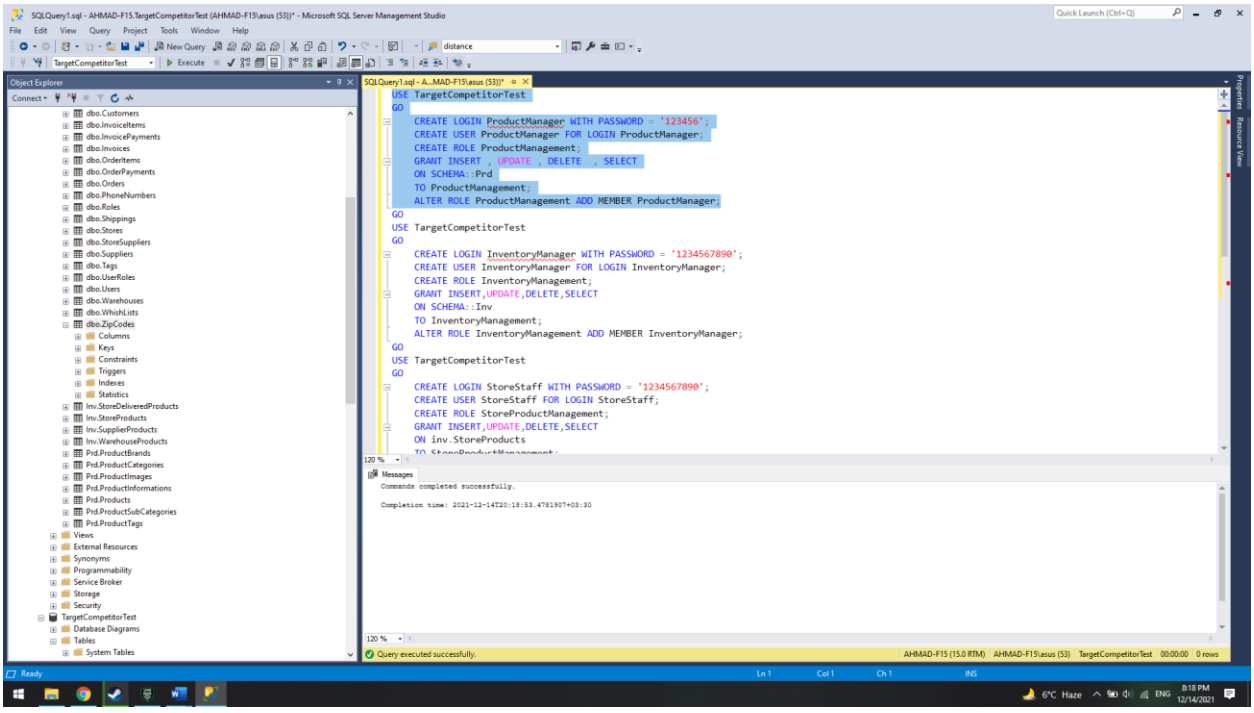


Comment: Here we wrote a script that gives full access to sysadmin

.

4)

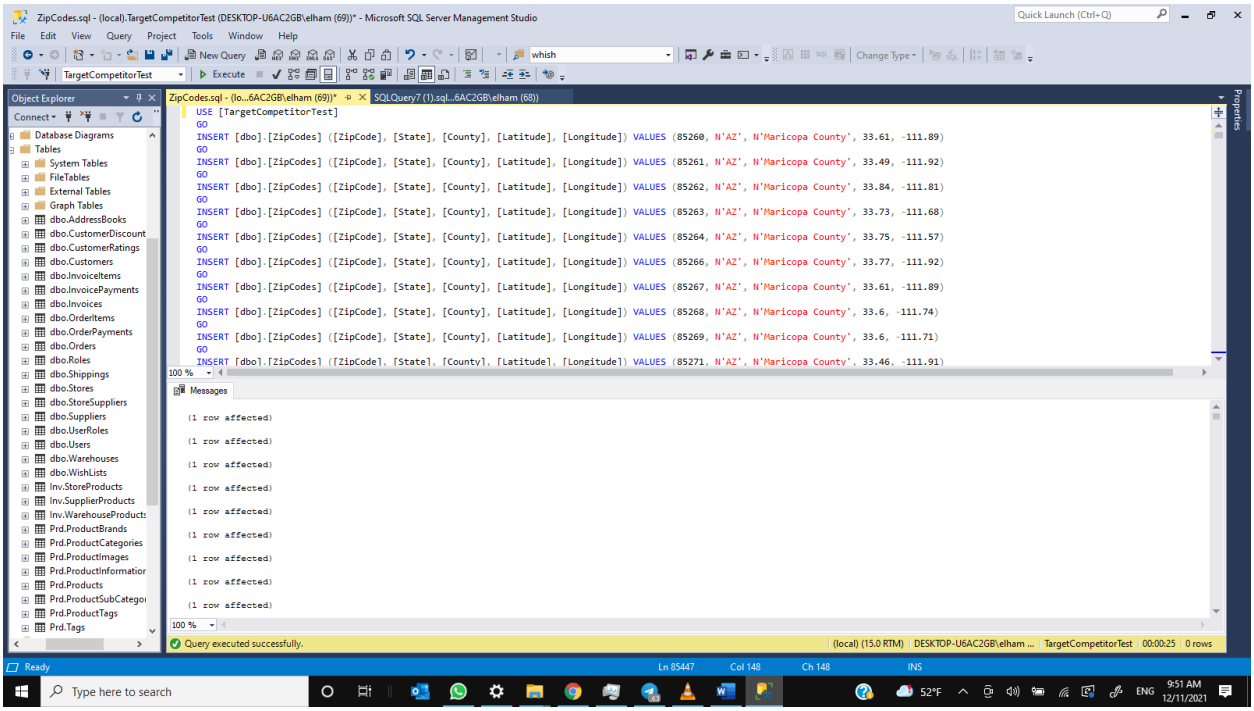
```
USE TargetCompetitorTest
GO
        CREATE LOGIN ProductManager WITH PASSWORD = '123456';
        CREATE USER ProductManager FOR LOGIN ProductManager;
        CREATE ROLE ProductManagement;
        GRANT INSERT , UPDATE , DELETE  , SELECT
        ON SCHEMA::Prd
        TO ProductManagement;
        ALTER ROLE ProductManagement ADD MEMBER ProductManager;
```



Comment: Here we wrote a script that allows product manager to perform DML on PRD schema.

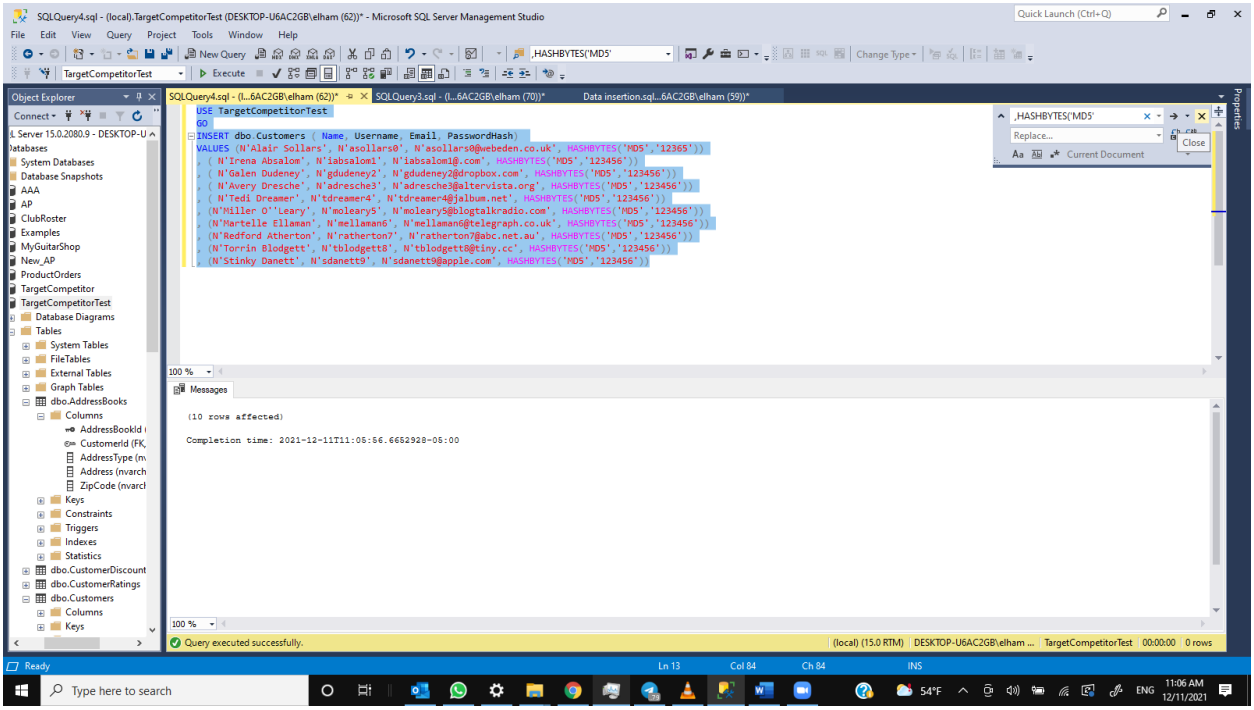
Database population with test data:

In this section, we insert data to be able to start testing the database. we start by inserting all the zip codes of all the regions in the country.



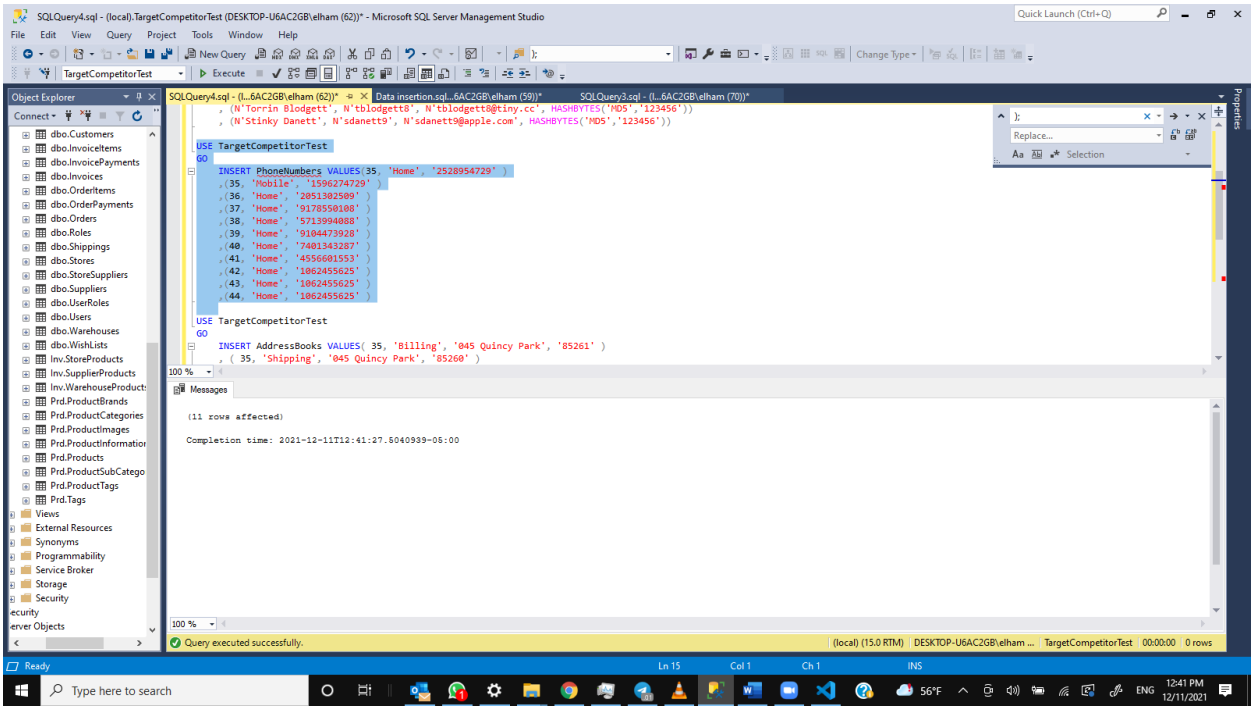
Comment: you can find the codes in appendix section (part1)

```
USE TargetCompetitorTest
GO
        INSERT dbo.Customers ( Name, Username, Email, PasswordHash)
        VALUES (N'Alair Sollars', N'asollars0', N'asollars0@webeden.co.uk', HASHBYTES('MD5','12365'))
        , ( N'Irena Absalom', N'iabsalom1', N'iabsalom1@.com', HASHBYTES('MD5','123456'))
        , ( N'Galen Dudeney', N'gdudeney2', N'gdudeney2@dropbox.com', HASHBYTES('MD5','123456'))
        , ( N'Avery Dresche', N'adresche3', N'adresche3@altervista.org', HASHBYTES('MD5','123456'))
        , ( N'Tedi Dreamer', N'tdreamer4', N'tdreamer4@jalbum.net', HASHBYTES('MD5','123456'))
        , (N'Miller O''Leary', N'moleary5', N'moleary5@blogtalkradio.com', HASHBYTES('MD5','123456'))
        , (N'Martelle Ellaman', N'mellaman6', N'mellaman6@telegraph.co.uk', HASHBYTES('MD5','123456'))
        , (N'Redford Atherton', N'ratherton7', N'ratherton7@abc.net.au', HASHBYTES('MD5','123456'))
        , (N'Torrin Blodgett', N'tblodgett8', N'tblodgett8@tiny.cc', HASHBYTES('MD5','123456'))
        , (N'Stinky Danett', N'sdanett9', N'sdanett9@apple.com', HASHBYTES('MD5','123456'))
```



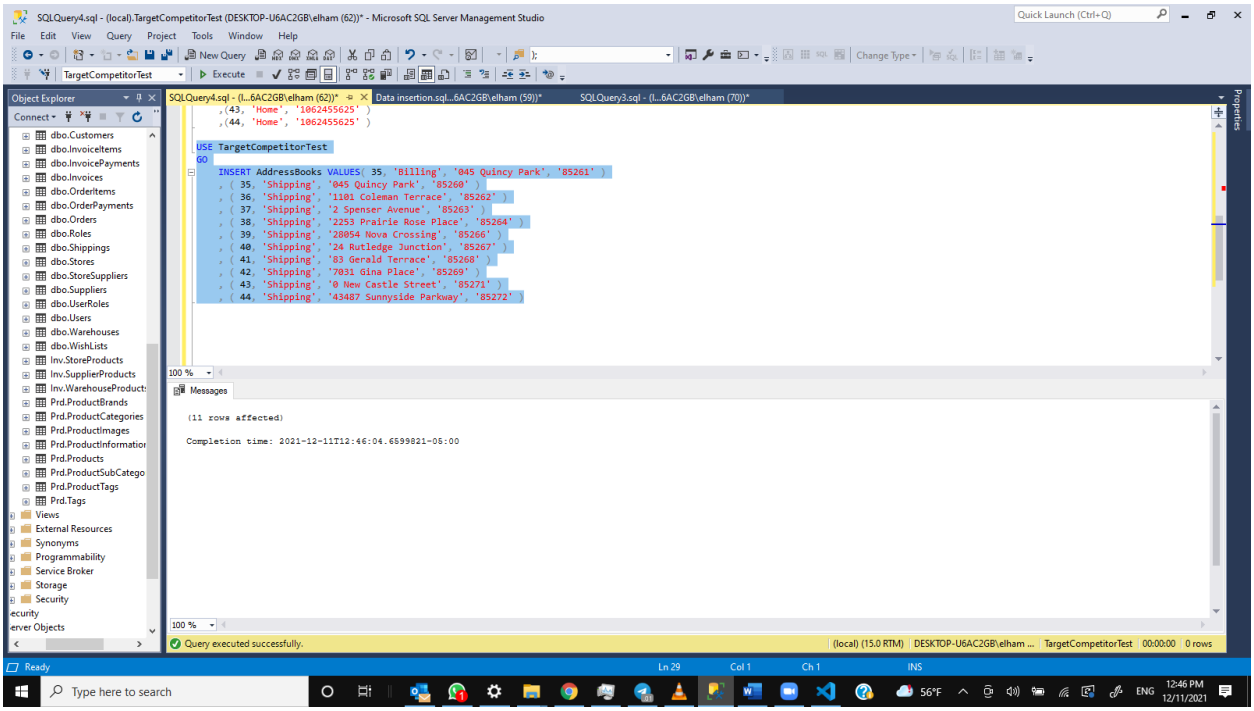
Comment: here we inserted name, username, email and password for ten customers.

```
USE TargetCompetitorTest
GO
        INSERT PhoneNumbers VALUES(35, 'Home', '2528954729' )
        ,(35, 'Mobile', '1596274729' )
        ,(36, 'Home', '2051302509' )
        ,(37, 'Home', '9178550108' )
        ,(38, 'Home', '5713994088' )
        ,(39, 'Home', '9104473928' )
        ,(40, 'Home', '7401343287' )
        ,(41, 'Home', '4556601553' )
        ,(42, 'Home', '1062455625' )
        ,(43, 'Home', '1062455625' )
        ,(44, 'Home', '1062455625' )
```



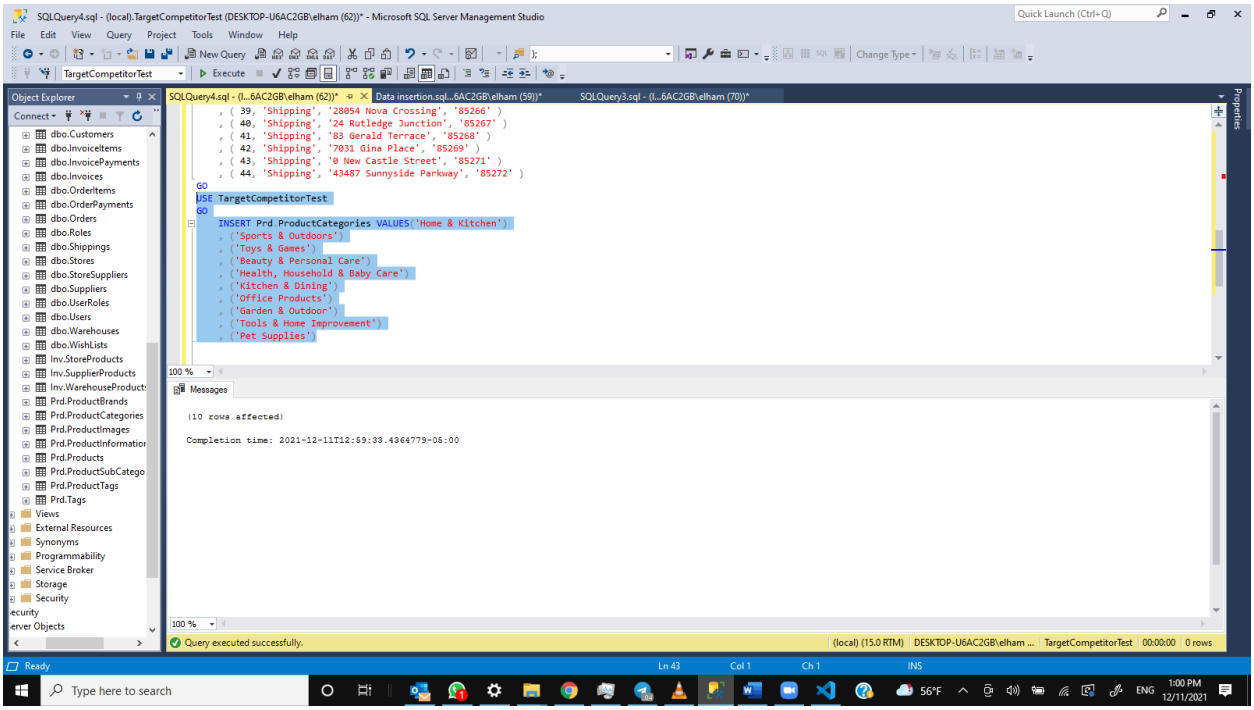
Comment: phone numbers have got inserted for each customer. As can be seen, each customer can have various phone numbers with different types (home, work, mobile)

```
USE TargetCompetitorTest
GO
        INSERT AddressBooks VALUES( 35, 'Billing', '045 Quincy Park', '85261' )
      , ( 35, 'Shipping', '045 Quincy Park', '85260' )
      , ( 36, 'Shipping', '1101 Coleman Terrace', '85262' )
      , ( 37, 'Shipping', '2 Spenser Avenue', '85263' )
      , ( 38, 'Shipping', '2253 Prairie Rose Place', '85264' )
      , ( 39, 'Shipping', '28054 Nova Crossing', '85266' )
      , ( 40, 'Shipping', '24 Rutledge Junction', '85267' )
      , ( 41, 'Shipping', '83 Gerald Terrace', '85268' )
      , ( 42, 'Shipping', '7031 Gina Place', '85269' )
      , ( 43, 'Shipping', '0 New Castle Street', '85271' )
      , ( 44, 'Shipping', '43487 Sunnyside Parkway', '85272' )
```



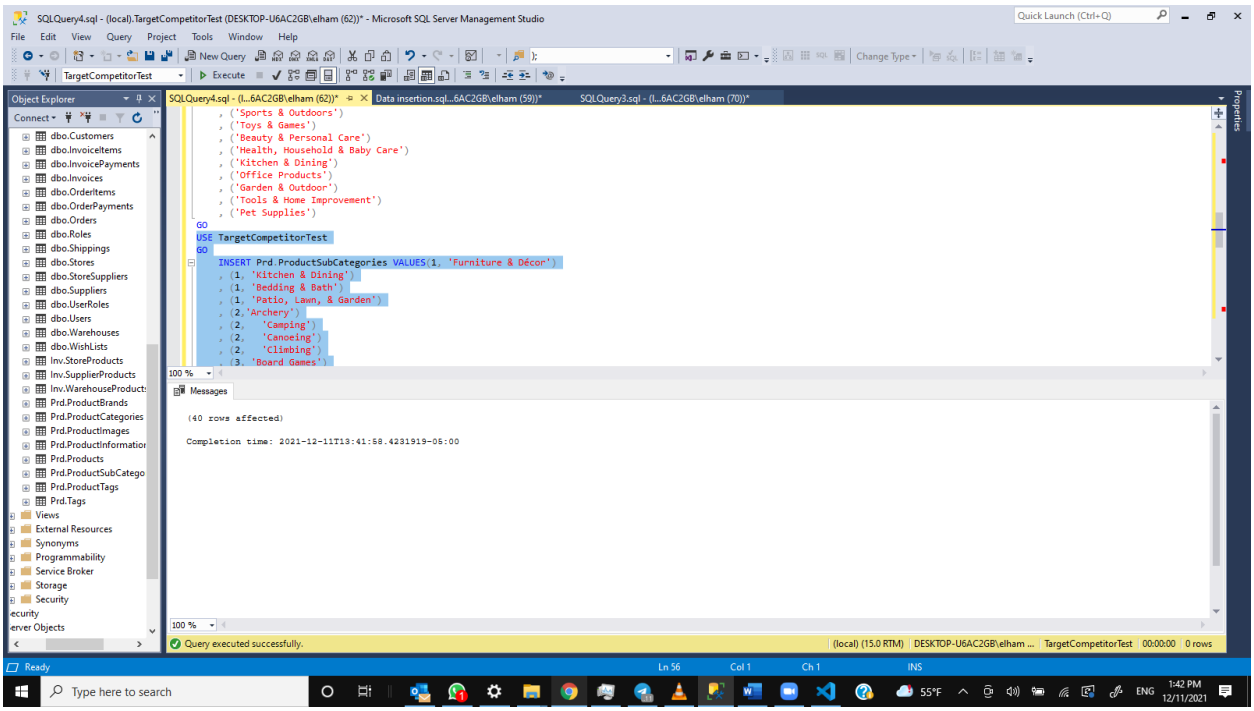
Comment: Addresses have got inserted for each customer. As can be seen, each customer can have various addresses with different types (billing, shipping)

```
USE TargetCompetitorTest
GO
        INSERT Prd.ProductCategories VALUES('Home & Kitchen')
        , ('Sports & Outdoors')
        , ('Toys & Games')
        , ('Beauty & Personal Care')
        , ('Health, Household & Baby Care')
        , ('Kitchen & Dining')
        , ('Office Products')
        , ('Garden & Outdoor')
        , ('Tools & Home Improvement')
        , ('Pet Supplies')
```



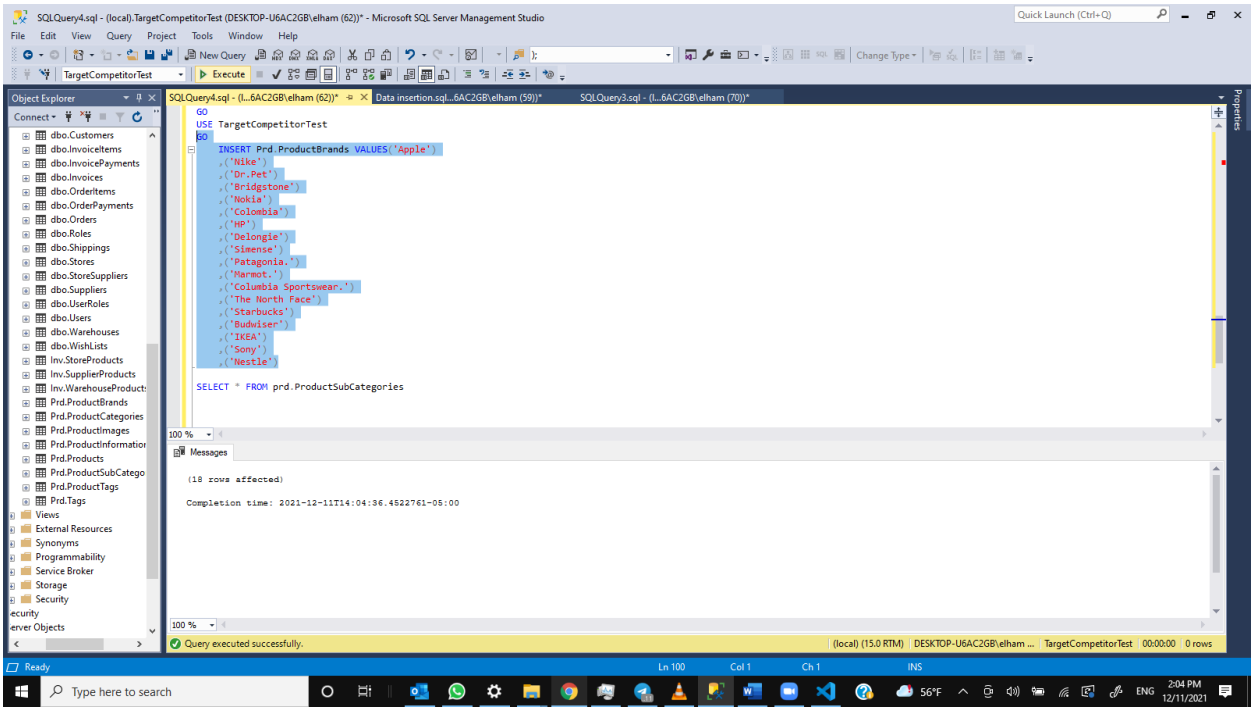
Comment: here we identified product categories. Next step would be defining product's subcategories.

```
USE TargetCompetitorTest
GO
        INSERT Prd.ProductSubCategories VALUES(1, 'Furniture & Décor')
        , (1, 'Kitchen & Dining')
        , (1, 'Bedding & Bath')
        , (1, 'Patio, Lawn, & Garden')
        , (2,'Archery')
        , (2,       'Camping')
        , (2,       'Canoeing')
        , (2,       'Climbing')
        , (3, 'Board Games')
        , (3,'Toy Foam Blasters & Guns')
        , (3,'Electronic Learning & Education Toys')
        , (3,'Kids" Electronics')
        , (4,'Facial Skin Care Products')
        , (4,'Body Skin Care Products')
        , (4,'Eye Treatment Products')
        , (4,'Bath & Bathing Accessories')
        , (5,'Household Supplies')
        , (5,'Health Care Products')
        , (5,'Baby Care Products')
        , (5,'Industrial & Scientific')
        , (6,'Dining Room Furniture')
        , (6,'Kitchen Utensils & Gadgets')
        , (6,'Kitchen Small Appliances')
        , (6,'Dining & Entertaining')
        , (7,'Desk Accessories')
        , (7,'Workspace Organizers')
        , (7,'Paper & Printable Media')
        , (7,'Filing Products')
        , (8,'Outdoor Benches')
        , (8,'Planter Raised Beds')
        , (8,'Deck Boxes')
        , (8,'Patio Conversation Sets')
        , (9,'Power Tools')
        , (9,'Hand Tools')
        , (9,'Power Tool Parts & Accessories')
        , (9,'Tool Boxes')
        , (10,'Dog Housebreaking Supplies')
        , (10,'Dog Beds & Furniture')
        , (10,'Dog Carriers & Travel Products')
        , (10,'Dog Bowls & Dishes')
```



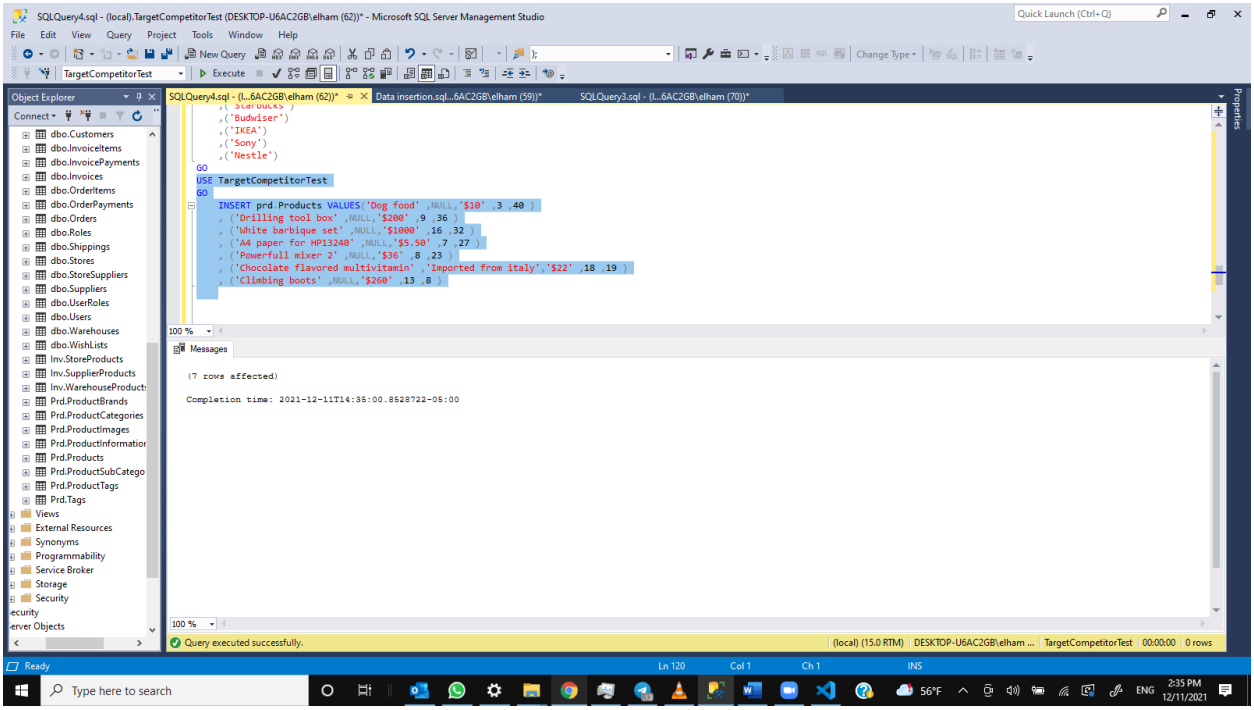
Comment: Here we identified four subcategories for each category.

```
GO
        INSERT Prd.ProductBrands VALUES('Apple')
        ,('Nike')
        ,('Dr.Pet')
        ,('Bridgstone')
        ,('Nokia')
        ,('Colombia')
        ,('HP')
        ,('Delongie')
        ,('Simense')
        ,('Patagonia.')
        ,('Marmot.')
        ,('Columbia Sportswear.')
        ,('The North Face')
        ,('Starbucks')
        ,('Budwiser')
        ,('IKEA')
        ,('Sony')
        ,('Nestle')
```



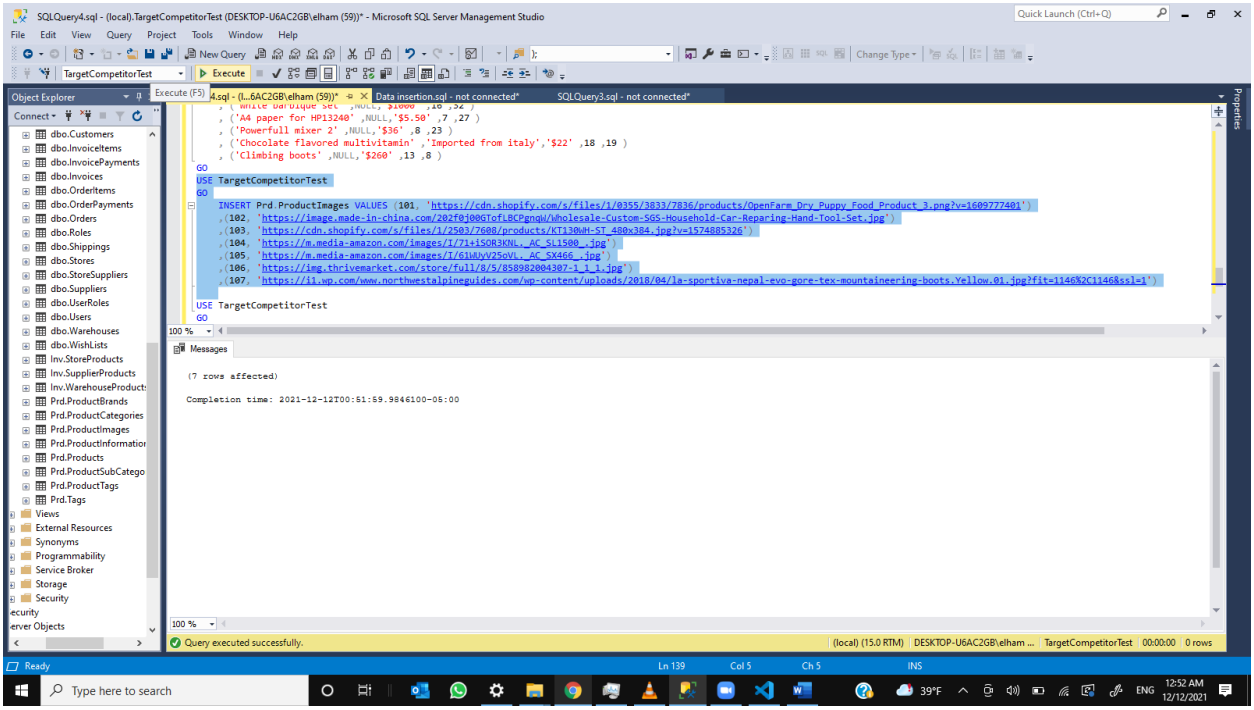
Comment: Some brands got inserter to product brands table.

```
USE TargetCompetitorTest
GO
        INSERT prd.Products VALUES('Dog food' ,NULL,'$10' ,3 ,40 )
        , ('Drilling tool box' ,NULL,'$200' ,9 ,36 )
        , ('White barbique set' ,NULL,'$1000' ,16 ,32 )
        , ('A4 paper for HP13240' ,NULL,'$5.50' ,7 ,27 )
        , ('Powerfull mixer 2' ,NULL,'$36' ,8 ,23 )
        , ('Chocolate flavored multivitamin' ,'Imported from italy','$22' ,18 ,19 )
        , ('Climbing boots' ,NULL,'$260' ,13 ,8 )
```



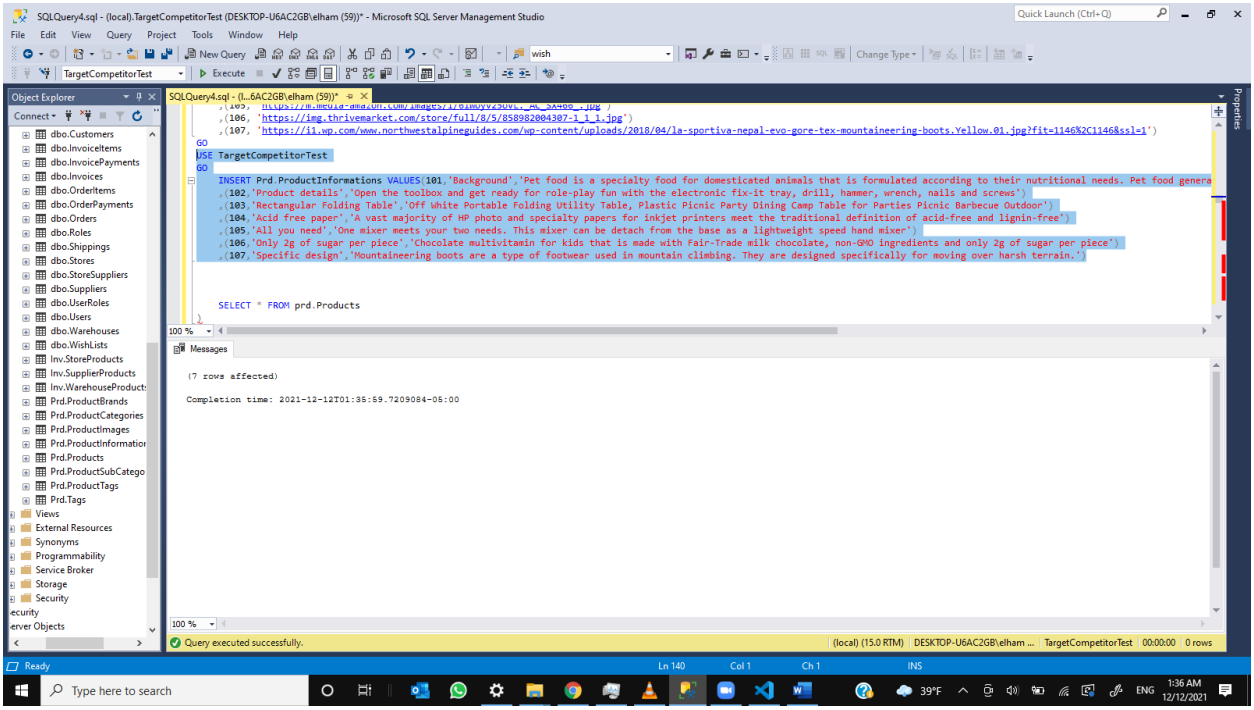
Comment: here real-world products has got entered and the subcategory and brand has got set.

```
USE TargetCompetitorTest
GO
	INSERT Prd.ProductImages VALUES (101,
'https://cdn.shopify.com/s/files/1/0355/3833/7836/products/OpenFarm_Dry_Puppy_Food_Product_3.png?v=1609777401')
	,(102, 'https://image.made-in-china.com/202f0j00GTofLBCPgnqW/Wholesale-Custom-SGS-Household-Car-Reparing-Hand-Tool-Set.jpg')
	,(103, 'https://cdn.shopify.com/s/files/1/2503/7608/products/KT130WH-ST_480x384.jpg?v=1574885326')
	,(104, 'https://m.media-amazon.com/images/I/71+iSOR3KNL._AC_SL1500_.jpg')
	,(105, 'https://m.media-amazon.com/images/I/61WUyV25oVL._AC_SX466_.jpg')
	,(106, 'https://img.thrivemarket.com/store/full/8/5/858982004307-1_1_1.jpg')
	,(107, 'https://i1.wp.com/www.northwestalpineguides.com/wp-content/uploads/2018/04/la-sportiva-nepal-evo-gore-tex-mountaineering-boots.Yellow.01.jpg?fit=1146%2C1146&ssl=1')
```

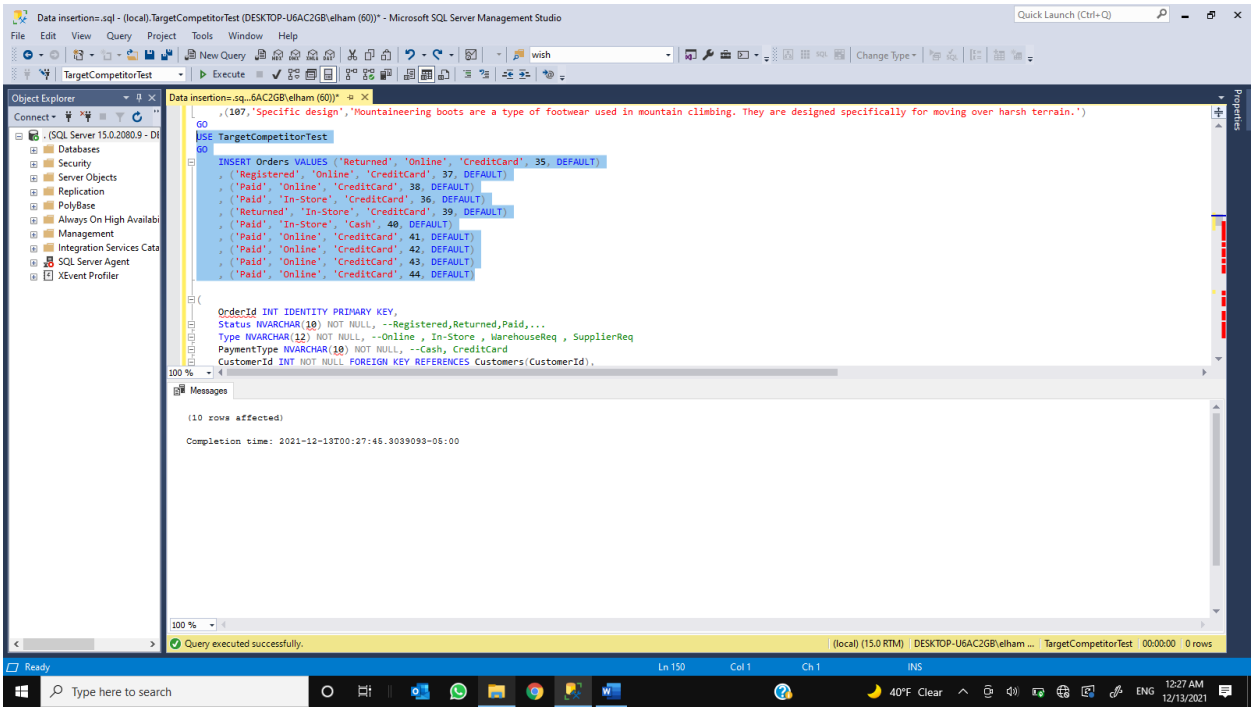


Comment: here we created the product images table which contains a link to the product's image.

```
USE TargetCompetitorTest
GO
        INSERT Prd.ProductInformations VALUES(101,'Background','Pet food is a specialty food for domesticated animals that is formulated according to their nutritional needs. Pet food generally consists of meat, meat byproducts, cereals, grain, vitamins, and minerals.')
        ,(102,'Product details','Open the toolbox and get ready for role-play fun with the electronic fix-it tray, drill, hammer, wrench, nails and screws')
        ,(103,'Rectangular Folding Table','Off White Portable Folding Utility Table, Plastic Picnic Party Dining Camp Table for Parties Picnic Barbecue Outdoor')
        ,(104,'Acid free paper','A vast majority of HP photo and specialty papers for inkjet printers meet the traditional definition of acid-free and lignin-free')
        ,(105,'All you need','One mixer meets your two needs. This mixer can be detach from the base as a lightweight speed hand mixer')
        ,(106,'Only 2g of sugar per piece','Chocolate multivitamin for kids that is made with Fair-Trade milk chocolate, non-GMO ingredients and only 2g of sugar per piece')
        ,(107,'Specific design','Mountaineering boots are a type of footwear used in mountain climbing. They are designed specifically for moving over harsh terrain.')
```

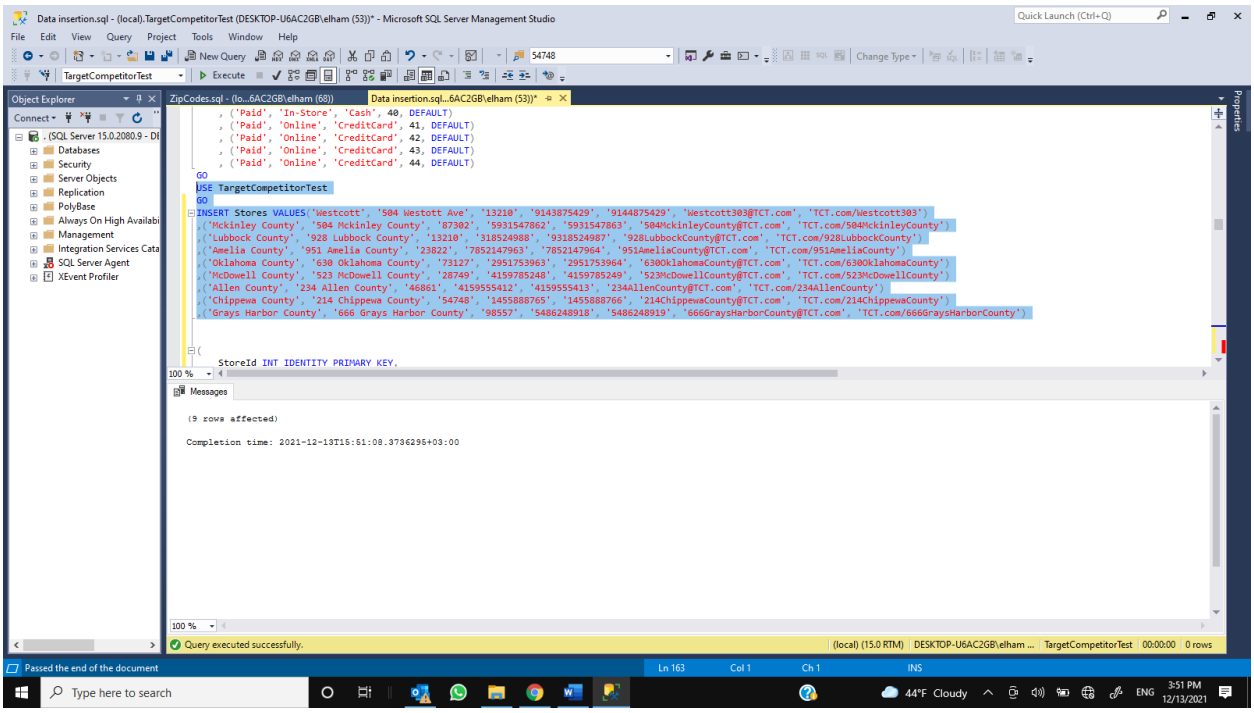


```
USE TargetCompetitorTest
GO
        INSERT Orders VALUES ('Returned', 'Online', 'CreditCard', 35, DEFAULT)
        , ('Registered', 'Online', 'CreditCard', 37, DEFAULT)
        , ('Paid', 'Online', 'CreditCard', 38, DEFAULT)
        , ('Paid', 'In-Store', 'CreditCard', 36, DEFAULT)
        , ('Returned', 'In-Store', 'CreditCard', 39, DEFAULT)
        , ('Paid', 'In-Store', 'Cash', 40, DEFAULT)
        , ('Paid', 'Online', 'CreditCard', 41, DEFAULT)
        , ('Paid', 'Online', 'CreditCard', 42, DEFAULT)
        , ('Paid', 'Online', 'CreditCard', 43, DEFAULT)
        , ('Paid', 'Online', 'CreditCard', 44, DEFAULT)
```



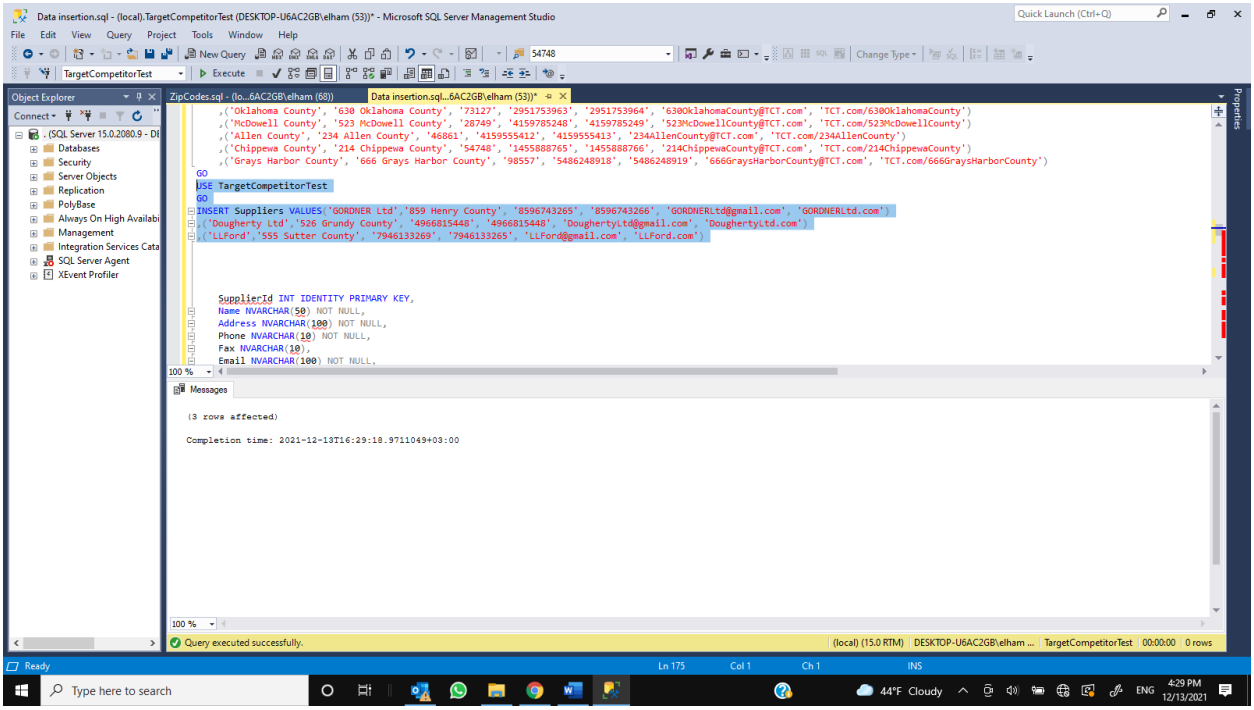
Comment: Here we entered some orders.

```
USE TargetCompetitorTest
GO
INSERT Stores VALUES('Westcott', '504 Westott Ave', '13210', '9143875429', '9144875429',
'Westcott303@TCT.com', 'TCT.com/Westcott303')
,('Mckinley County', '504 Mckinley County', '87302', '5931547862', '5931547863',
'504MckinleyCounty@TCT.com', 'TCT.com/504MckinleyCounty')
,('Lubbock County', '928 Lubbock County', '13210', '318524988', '9318524987',
'928LubbockCounty@TCT.com', 'TCT.com/928LubbockCounty')
,('Amelia County', '951 Amelia County', '23822', '7852147963', '7852147964', '951AmeliaCounty@TCT.com',
'TCT.com/951AmeliaCounty')
,('Oklahoma County', '630 Oklahoma County', '73127', '2951753963', '2951753964',
'630OklahomaCounty@TCT.com', 'TCT.com/630OklahomaCounty')
,('McDowell County', '523 McDowell County', '28749', '4159785248', '4159785249',
'523McDowellCounty@TCT.com', 'TCT.com/523McDowellCounty')
,('Allen County', '234 Allen County', '46861', '4159555412', '4159555413', '234AllenCounty@TCT.com',
'TCT.com/234AllenCounty')
,('Chippewa County', '214 Chippewa County', '54748', '1455888765', '1455888766',
'214ChippewaCounty@TCT.com', 'TCT.com/214ChippewaCounty')
,('Grays Harbor County', '666 Grays Harbor County', '98557', '5486248918', '5486248919',
'666GraysHarborCounty@TCT.com', 'TCT.com/666GraysHarborCounty')
```



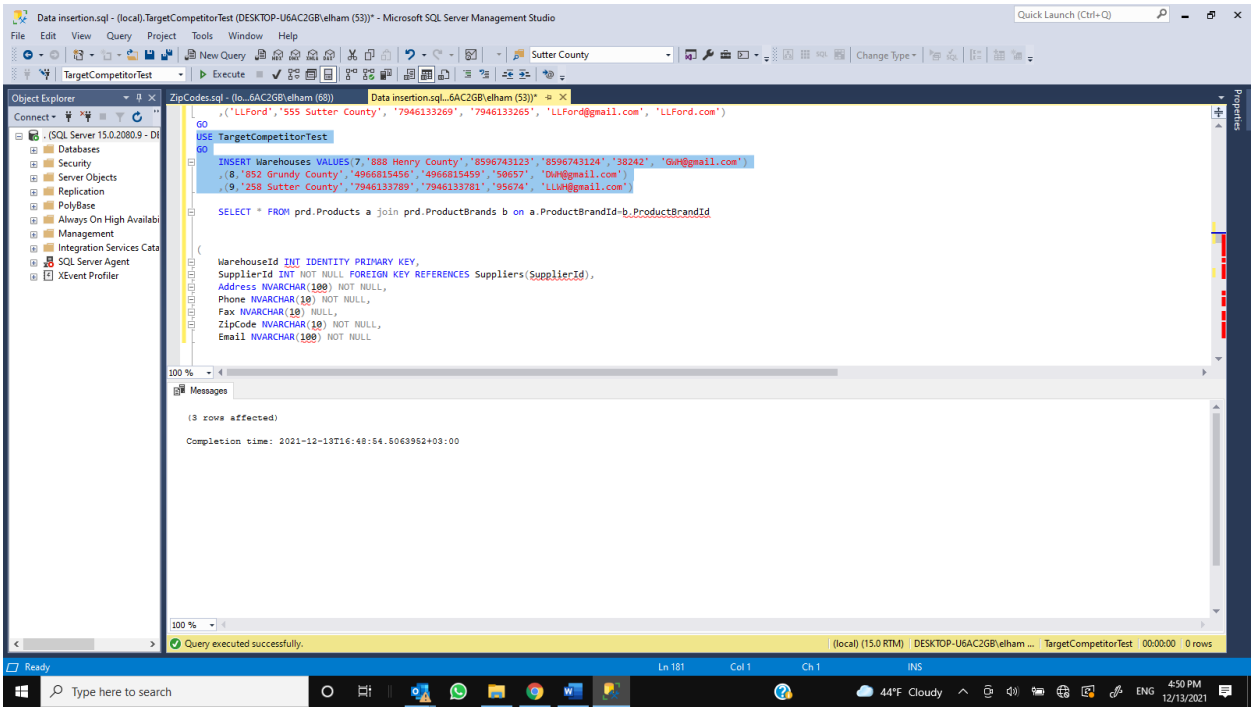
Comment: The information of stores has got entered in the store table.

```
USE TargetCompetitorTest
GO
INSERT Suppliers VALUES('GORDNER Ltd','859 Henry County', '8596743265', '8596743266',
'GORDNERLtd@gmail.com', 'GORDNERLtd.com')
,('Dougherty Ltd','526 Grundy County', '4966815448', '4966815448', 'DoughertyLtd@gmail.com',
'DoughertyLtd.com')
,('LLFord','555 Sutter County', '7946133269', '7946133265', 'LLFord@gmail.com', 'LLFord.com')
```



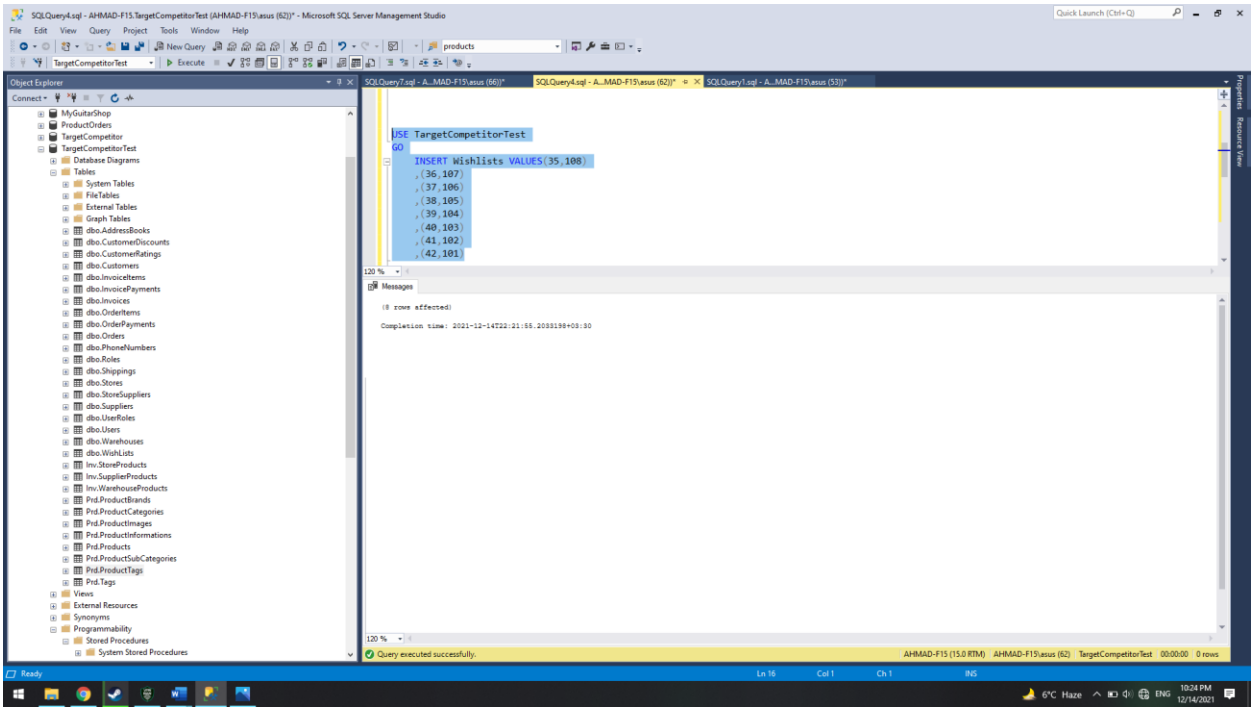
Comment: Here we entered our main three suppliers.

```
USE TargetCompetitorTest
GO
        INSERT Warehouses VALUES(7,'888 Henry County','8596743123','8596743124','38242',
'GWH@gmail.com')
        ,(8,'852 Grundy County','4966815456','4966815459','50657', 'DWH@gmail.com')
        ,(9,'258 Sutter County','7946133789','7946133781','95674', 'LLWH@gmail.com')
```



Comment: Here we entered our main three warehouses.

```
USE TargetCompetitorTest
GO
        INSERT Wishlists VALUES(35,108)
        ,(36,107)
        ,(37,106)
        ,(38,105)
        ,(39,104)
        ,(40,103)
        ,(41,102)
        ,(42,101)
```

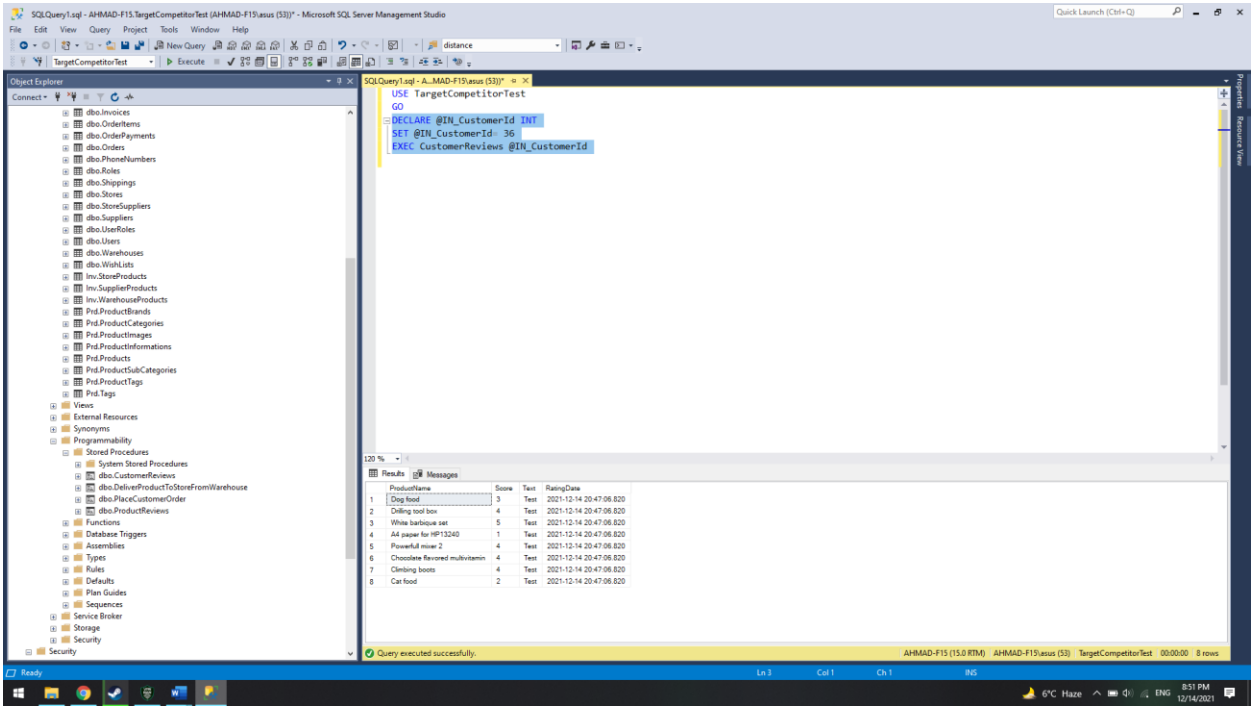


Comment: here we updated the Wish lists table data.

Testing, database's reliability demonstration and logic:

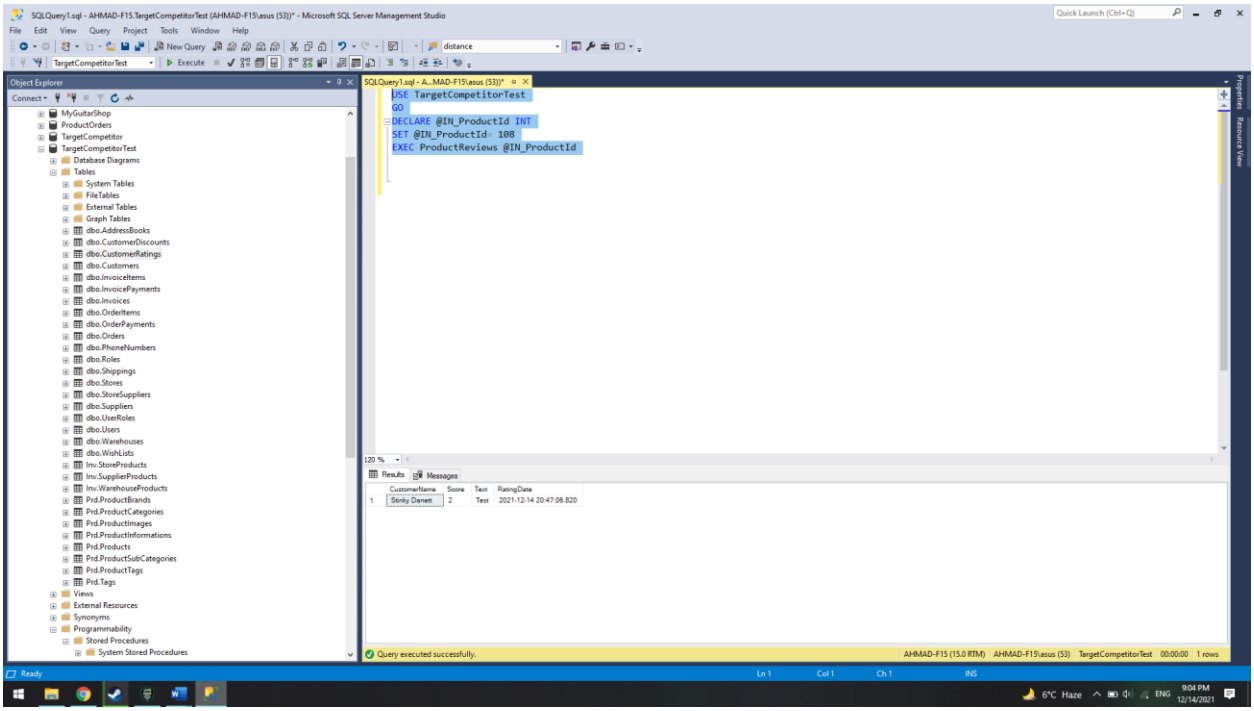
Transaction and stored procedures Testing:

```
DECLARE @IN_CustomerId INT
SET @IN_CustomerId = 36
EXEC CustomerReviews @IN_CustomerId
```



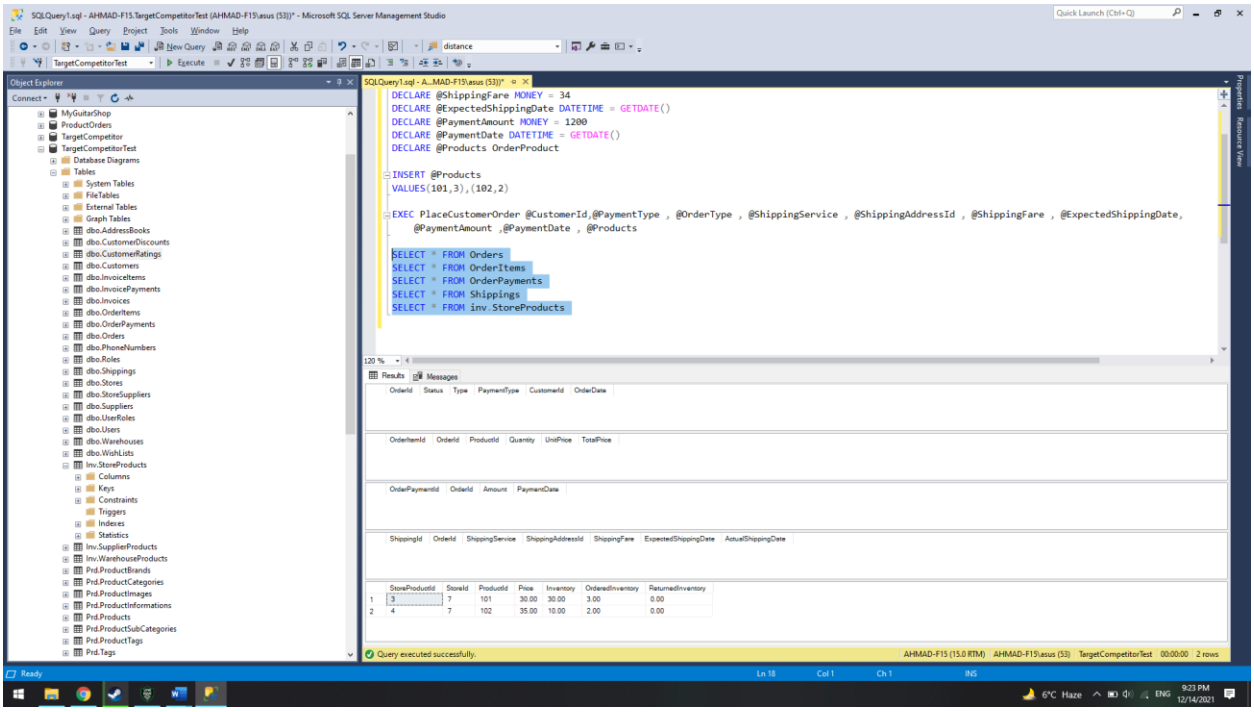
Comment: by running procedures we see that "White barbeque set" has the highest score of 5 and may be considered for promotion.

```
USE TargetCompetitorTest
GO
DECLARE @IN_ProductId INT
SET @IN_ProductId= 108
EXEC ProductReviews @IN_ProductId
```



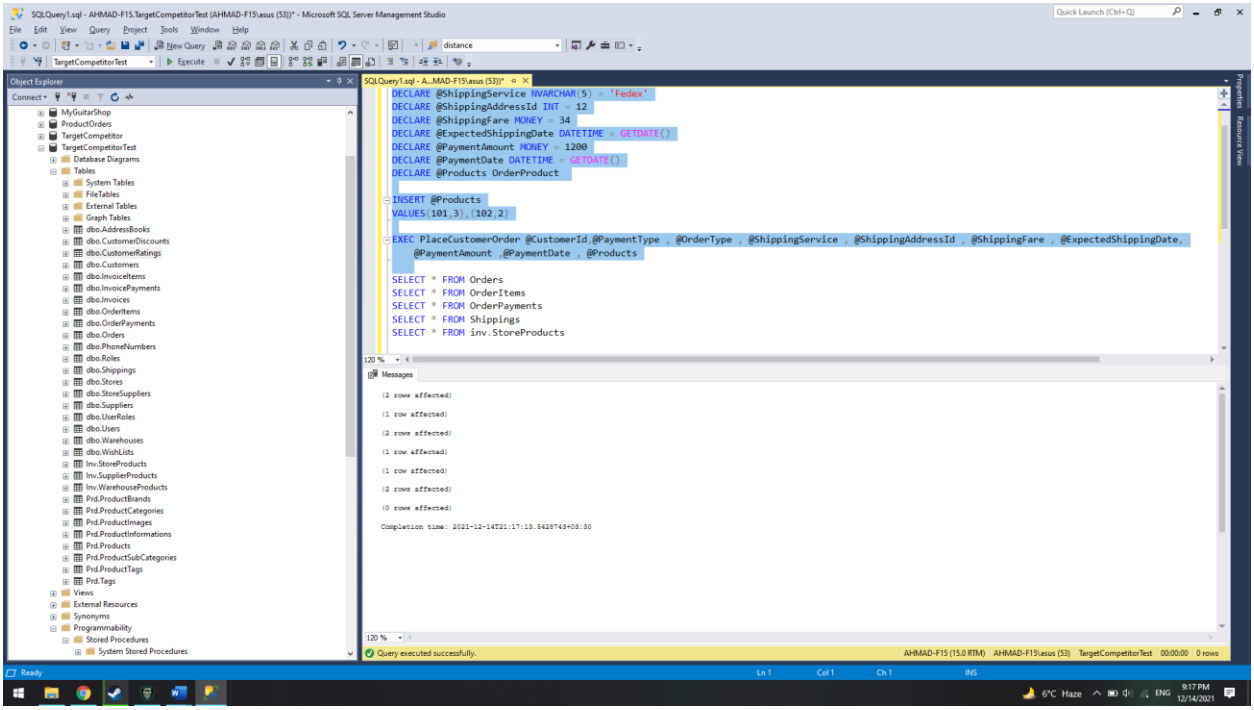
Comment: by running procedures we see that the product 108 has one review from Stinky Danett who gave a score of 2 to the product.

```
SELECT * FROM Orders
SELECT * FROM OrderItems
SELECT * FROM OrderPayments
SELECT * FROM Shippings
SELECT * FROM inv.StoreProducts
```



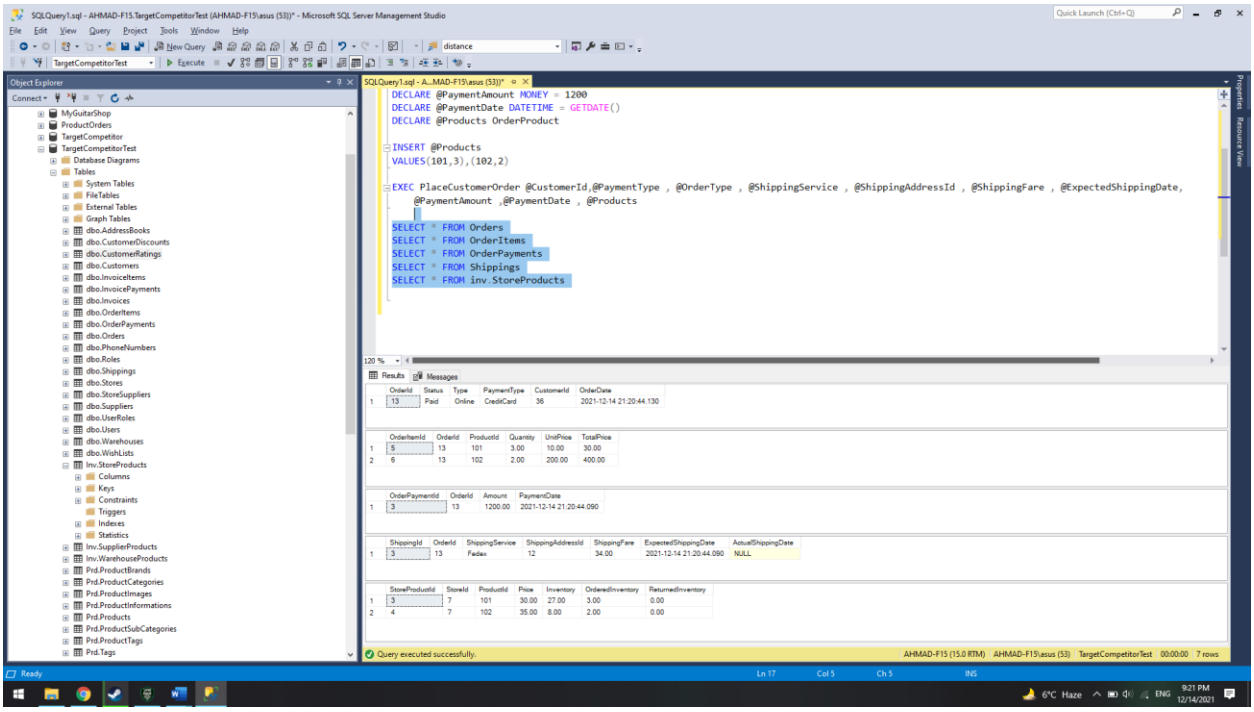
Comment: before running the procedure & transaction there is no orders in the database.

```
DECLARE @CustomerId INT = 36
DECLARE @PaymentType NVARCHAR(10) = 'CreditCard'
DECLARE @OrderType NVARCHAR(7) = 'Online'
DECLARE @ShippingService NVARCHAR(5) = 'Fedex'
DECLARE @ShippingAddressId INT = 12
DECLARE @ShippingFare MONEY = 34
DECLARE @ExpectedShippingDate DATETIME = GETDATE()
DECLARE @PaymentAmount MONEY = 1200
DECLARE @PaymentDate DATETIME = GETDATE()
DECLARE @Products OrderProduct

INSERT @Products
VALUES(101,3),(102,2)

EXEC PlaceCustomerOrder @CustomerId,@PaymentType , @OrderType , @ShippingService ,
@ShippingAddressId , @ShippingFare , @ExpectedShippingDate,
            @PaymentAmount ,@PaymentDate , @Products
```



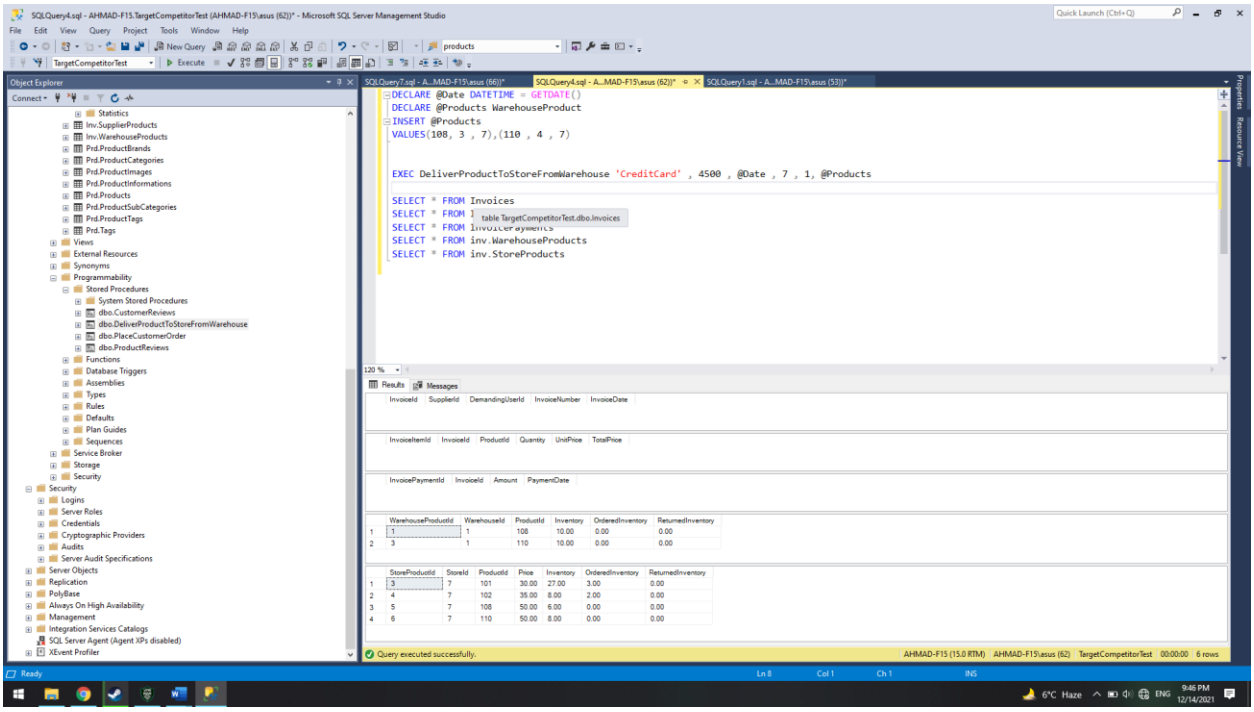
Comment: by running the script we get the following results.

```
SELECT * FROM Orders
SELECT * FROM OrderItems
SELECT * FROM OrderPayments
SELECT * FROM Shippings
SELECT * FROM inv.StoreProducts
```



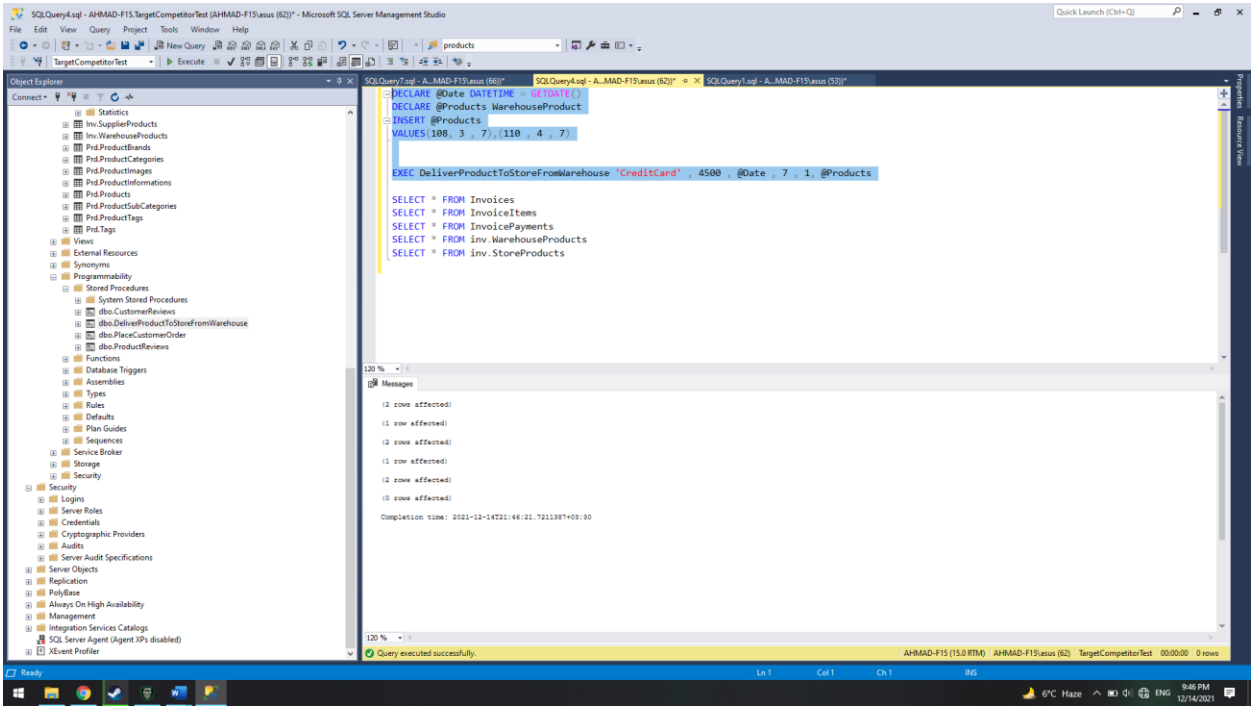
Comment: here we can see the ordered inventory decreased.

```
SELECT * FROM Invoices
SELECT * FROM InvoiceItems
SELECT * FROM InvoicePayments
SELECT * FROM inv.WarehouseProducts
SELECT * FROM inv.StoreProducts
```



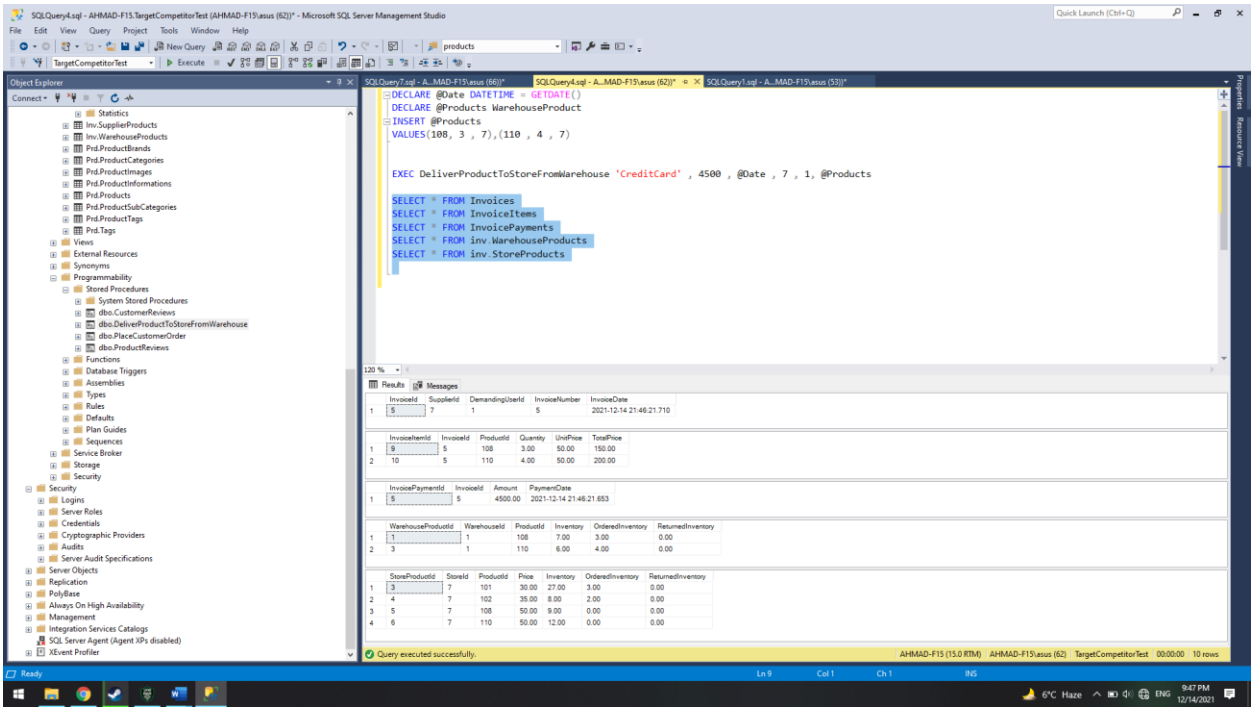
Comment: before running the procedure and transaction we get the above result.

```
DECLARE @Date DATETIME = GETDATE()
DECLARE @Products WarehouseProduct
INSERT @Products
VALUES(108, 3 , 7),(110 , 4 , 7)

EXEC DeliverProductToStoreFromWarehouse 'CreditCard' , 4500 , @Date , 7 , 1, @Products
```



Comment: here we ran the script and we got the following result.

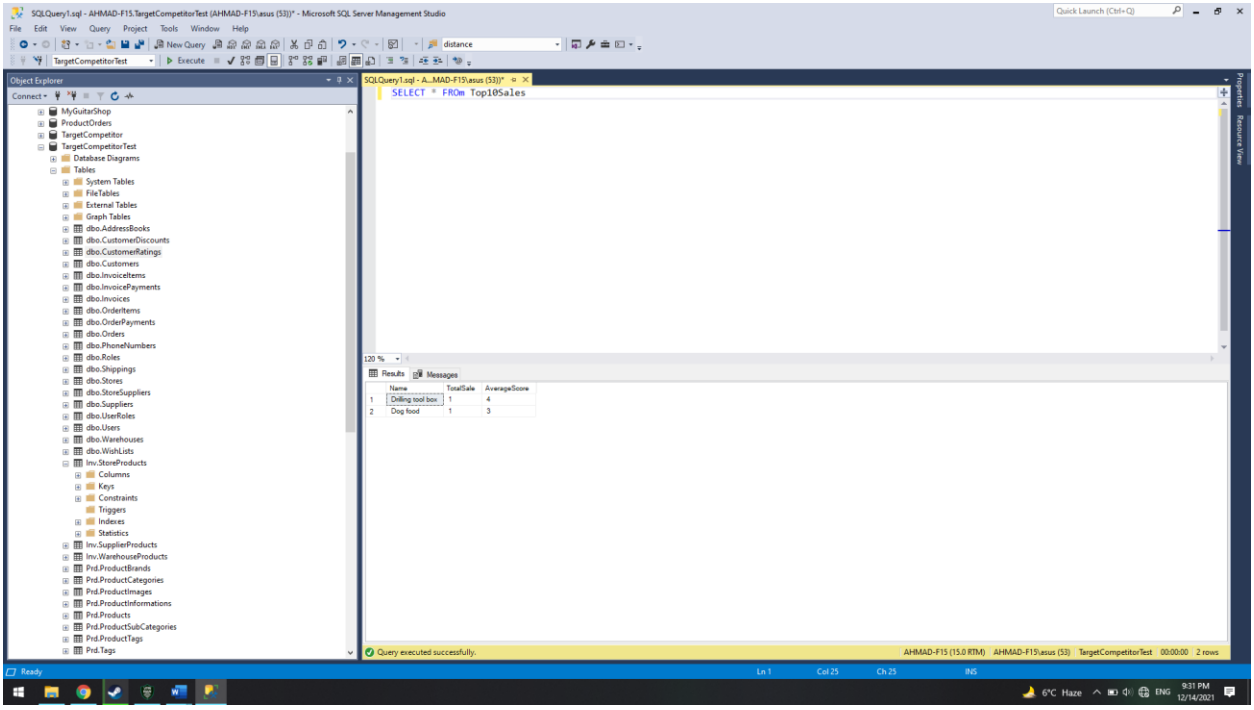
```
SELECT * FROM Invoices
SELECT * FROM InvoiceItems
SELECT * FROM InvoicePayments
SELECT * FROM inv.WarehouseProducts
SELECT * FROM inv.StoreProducts
```



Comment: after running the script we see that inventory and ordered inventory has got updated.

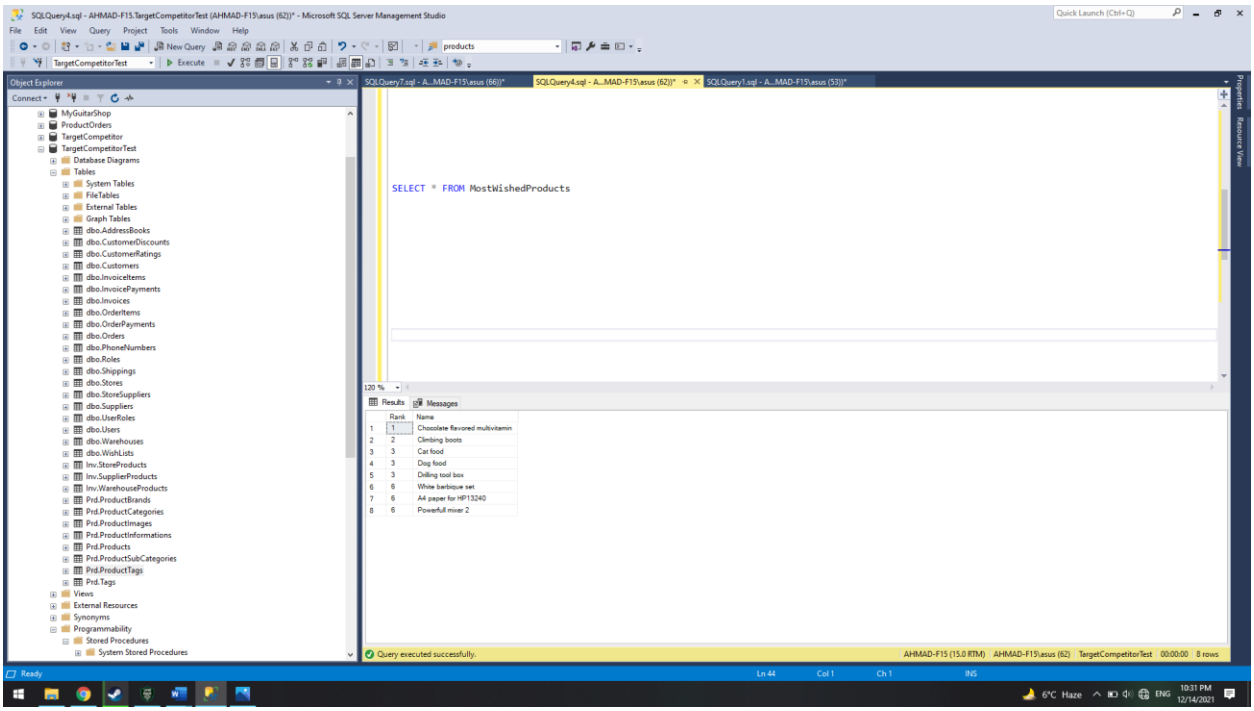
View Testing:

```
SELECT * FROM Top10Sales
```



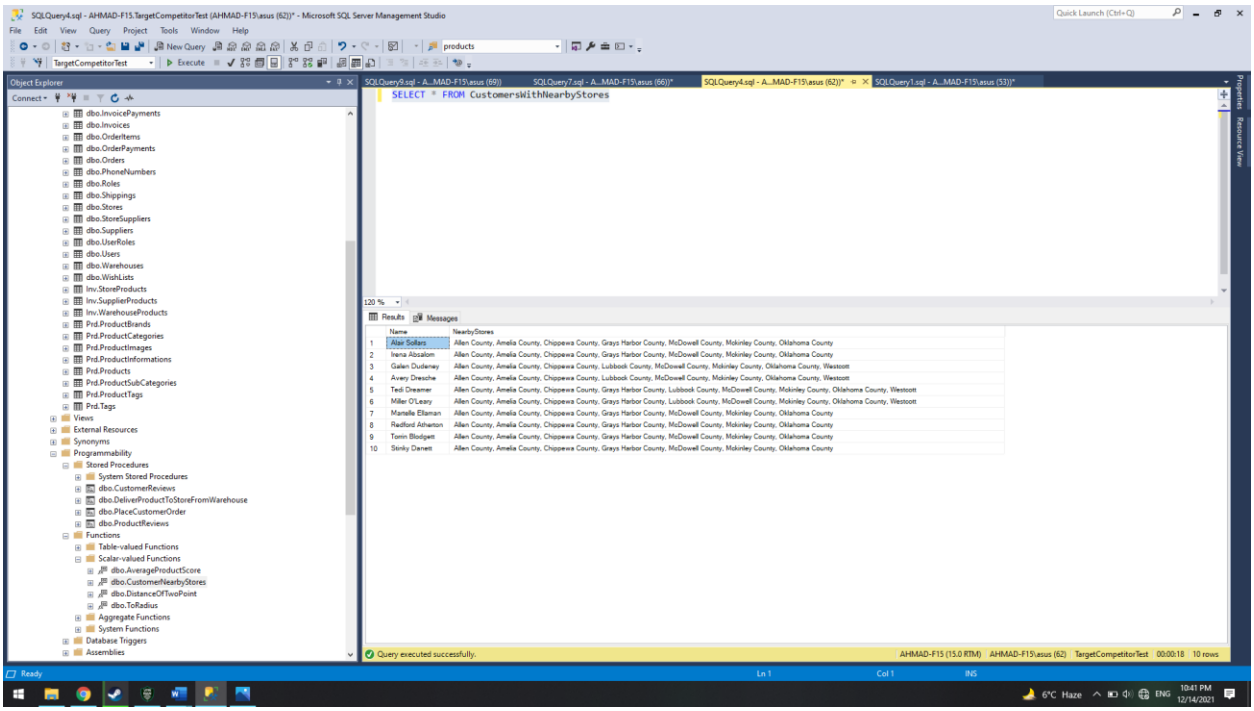
Comment: here we can see the top10 sales.

```
SELECT * FROM MostWishedProducts
```



Comment: here we can see the most wished products.

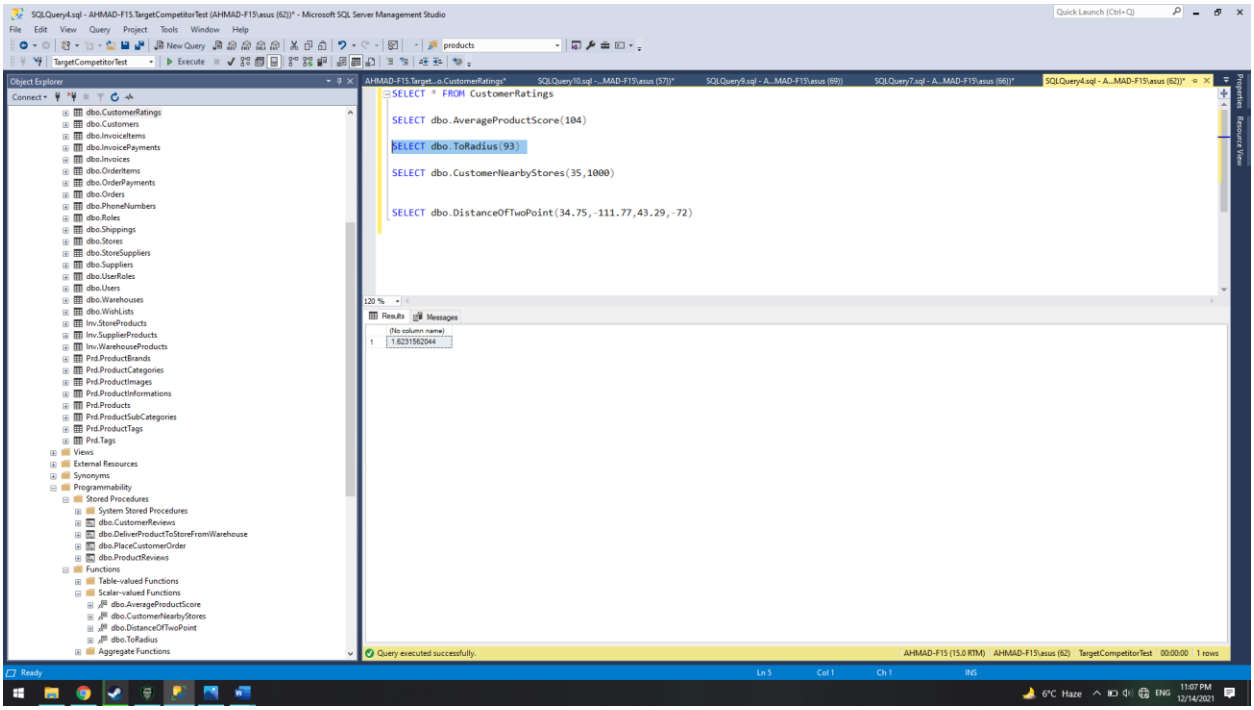
```
SELECT * FROM CustomersWithNearbyStores
```



Comment: here we get the nearby stores for each customer.

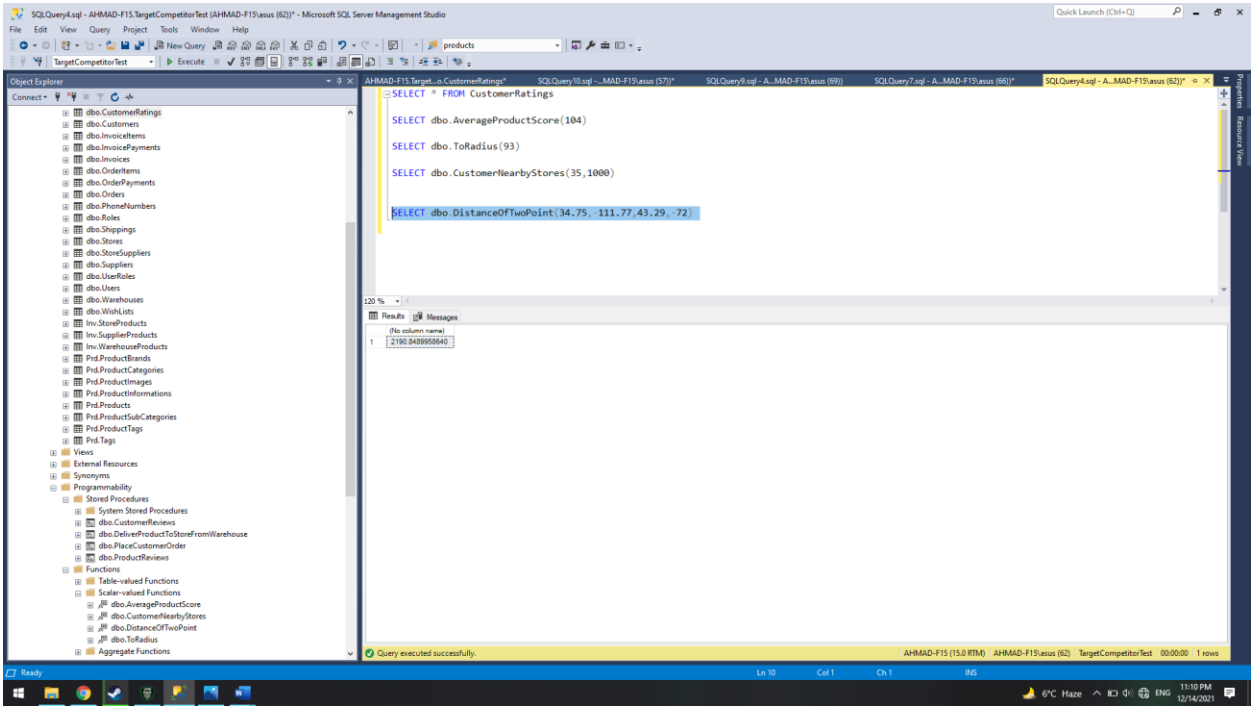
Function Testing:

```
SELECT dbo.ToRadius(93)
```



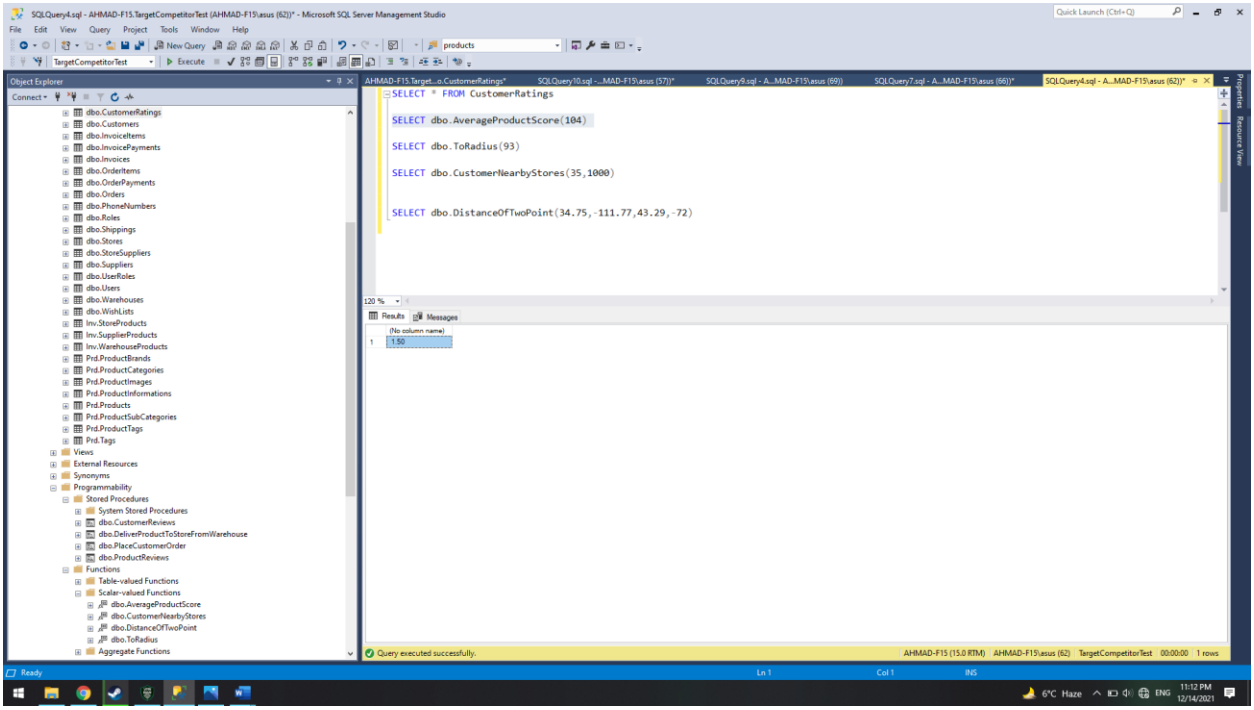
Comment: by entering the 93 we get 1.6231562044. the output of this function will be used to calculate the distance between two points on earth.

```
SELECT dbo.DistanceOfTwoPoint(34.75,-111.77,43.29,-72)
```



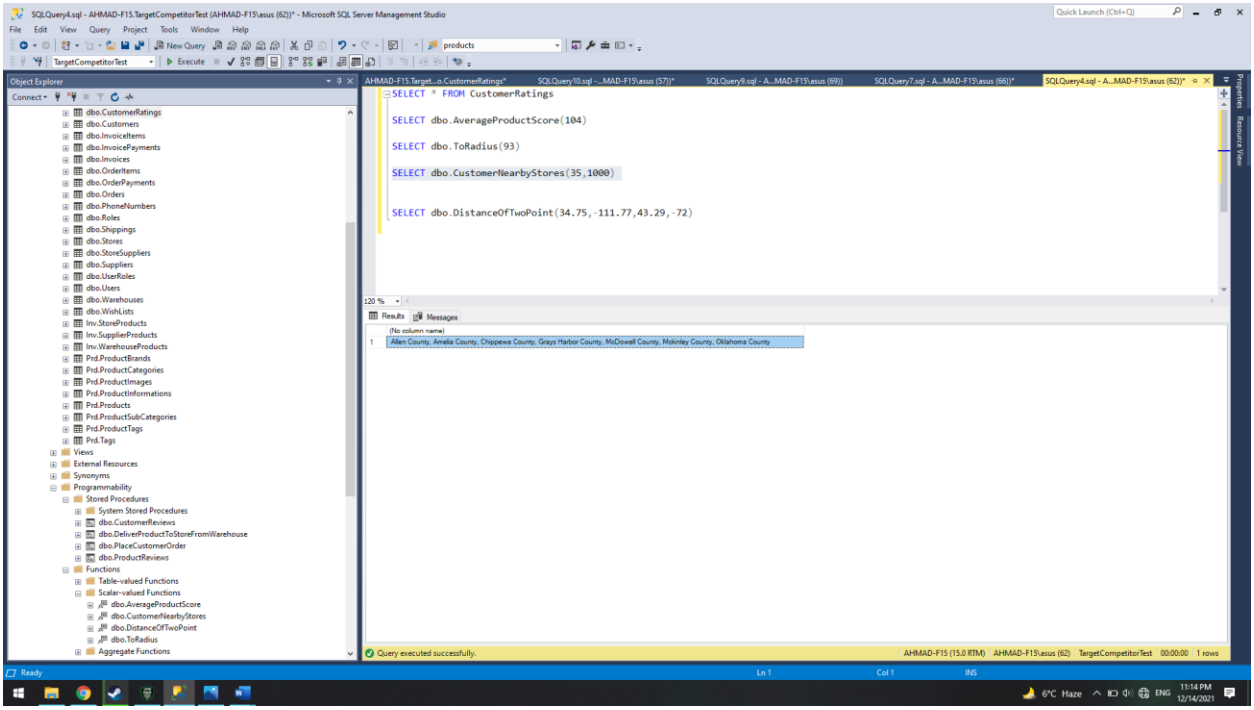
Comment: by entering `(34.75,-111.77,43.29,-72)` we get 2190.84.

```
SELECT dbo.AverageProductScore(104)
```



Comment: by entering 104 as product ID we get the average score of 1.5

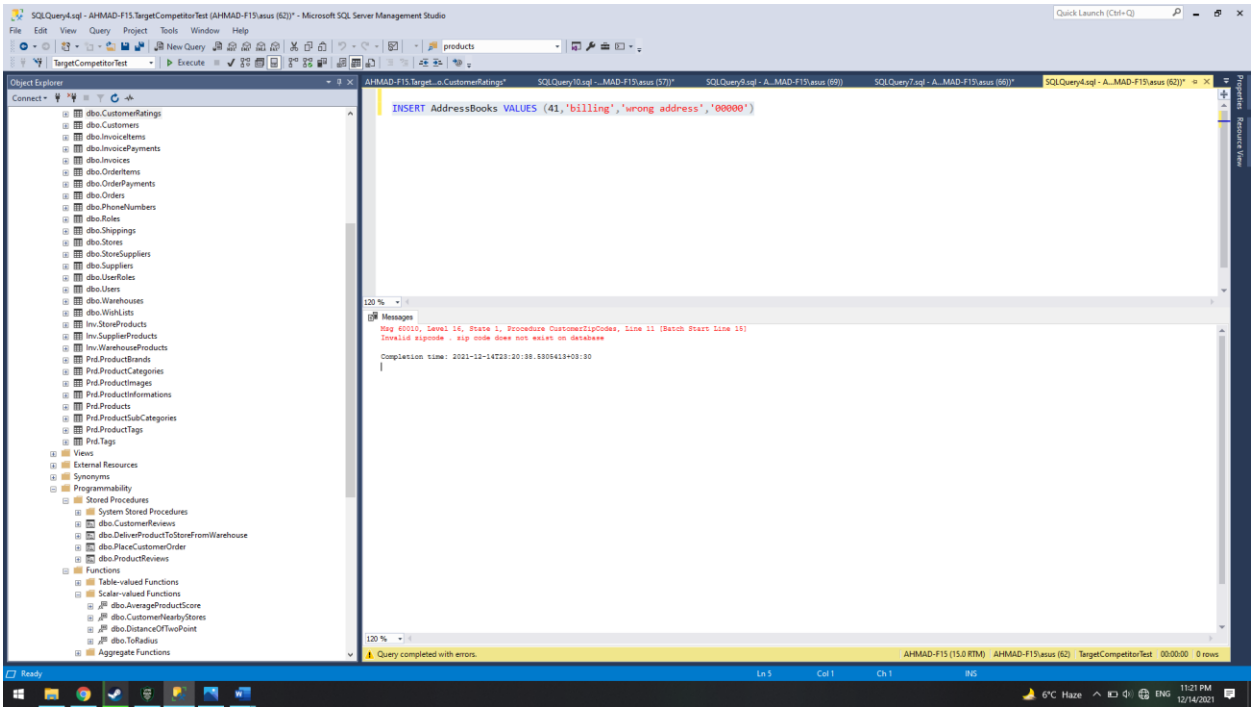
```
SELECT dbo.CustomerNearbyStores(35,1000)
```



Comment: by entering the customer ID we get the stores nearby customers.

Trigger Testing:

```
INSERT AddressBooks VALUES (41,'billing','wrong address','00000')
```



Comment: the trigger cached a wrong zip code and throwed “Invalid zipcode . zip code does not exist on database”

Conclusions:

creating a database for a sale business seems overwhelming but is a truly interesting task. How to create tables and organize them is the infrastructure which I really paid attention to. For the second step I taught about the real word requirement of such business and made a business plan and created and tested all the scripts which may be useful in these types of businesses.
This project was really helpful in reviewing all the materials we have learned during the semester

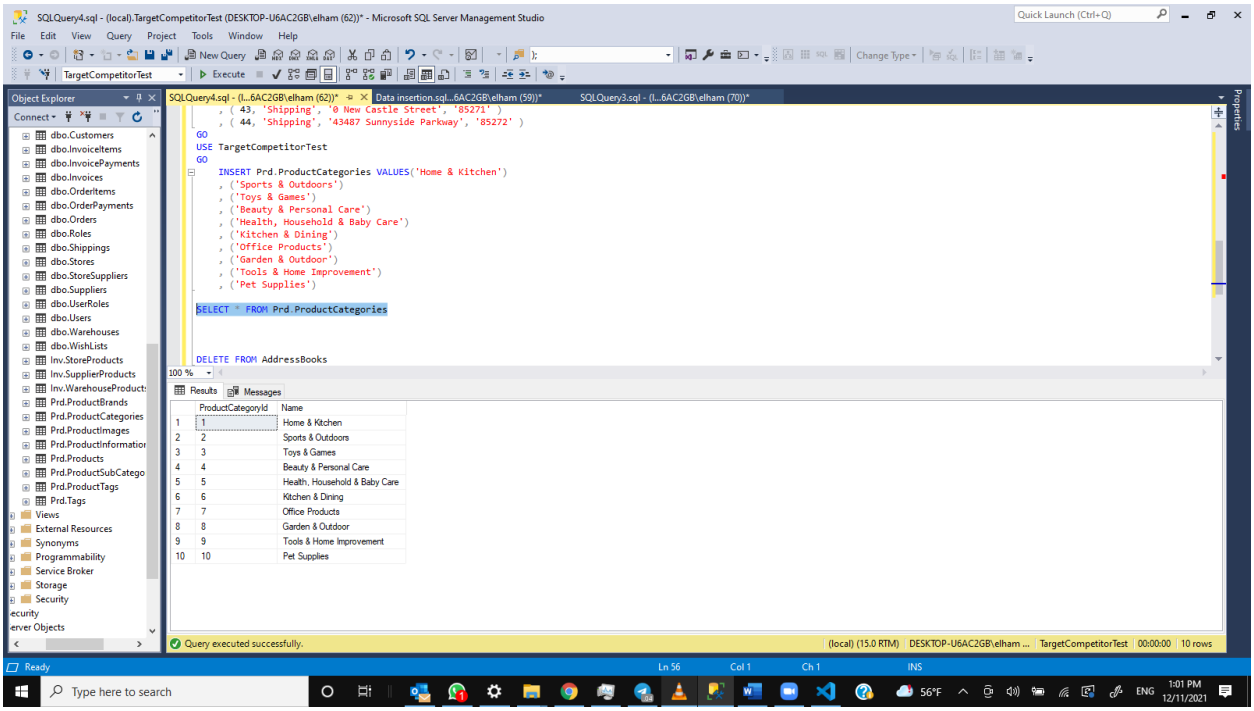
Appendix: screenshots from SQL Server for codes, testing, etc.

Part1:

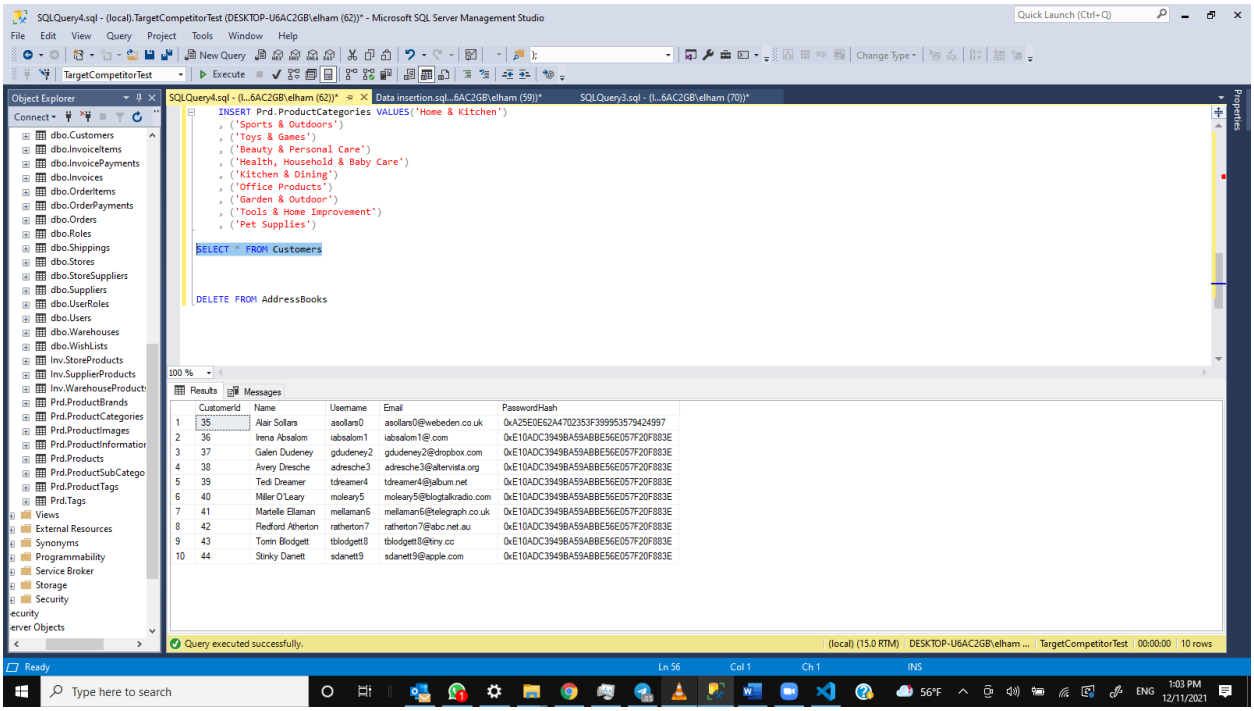
```
USE [TargetCompetitorTest]
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85260, N'AZ',
N'Maricopa County', 33.61, -111.89)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85261, N'AZ',
N'Maricopa County', 33.49, -111.92)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85262, N'AZ',
N'Maricopa County', 33.84, -111.81)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85263, N'AZ',
N'Maricopa County', 33.73, -111.68)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85264, N'AZ',
N'Maricopa County', 33.75, -111.57)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85266, N'AZ',
N'Maricopa County', 33.77, -111.92)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85267, N'AZ',
N'Maricopa County', 33.61, -111.89)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85268, N'AZ',
N'Maricopa County', 33.6, -111.74)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85269, N'AZ',
N'Maricopa County', 33.6, -111.71)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85271, N'AZ',
N'Maricopa County', 33.46, -111.91)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85272, N'AZ', N'Pinal
County', 32.85, -111.97)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85273, N'AZ', N'Pinal
County', 33.28, -111.11)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85274, N'AZ',
N'Maricopa County', 33.38, -111.87)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85275, N'AZ',
N'Maricopa County', 33.45, -111.76)
GO
INSERT [dbo].[ZipCodes] ([ZipCode], [State], [County], [Latitude], [Longitude]) VALUES (85277, N'AZ',
N'Maricopa County', 33.45, -111.72)
GO
.
.
.
```

You can find all the codes in the files attached to this report
Part 2: table verification

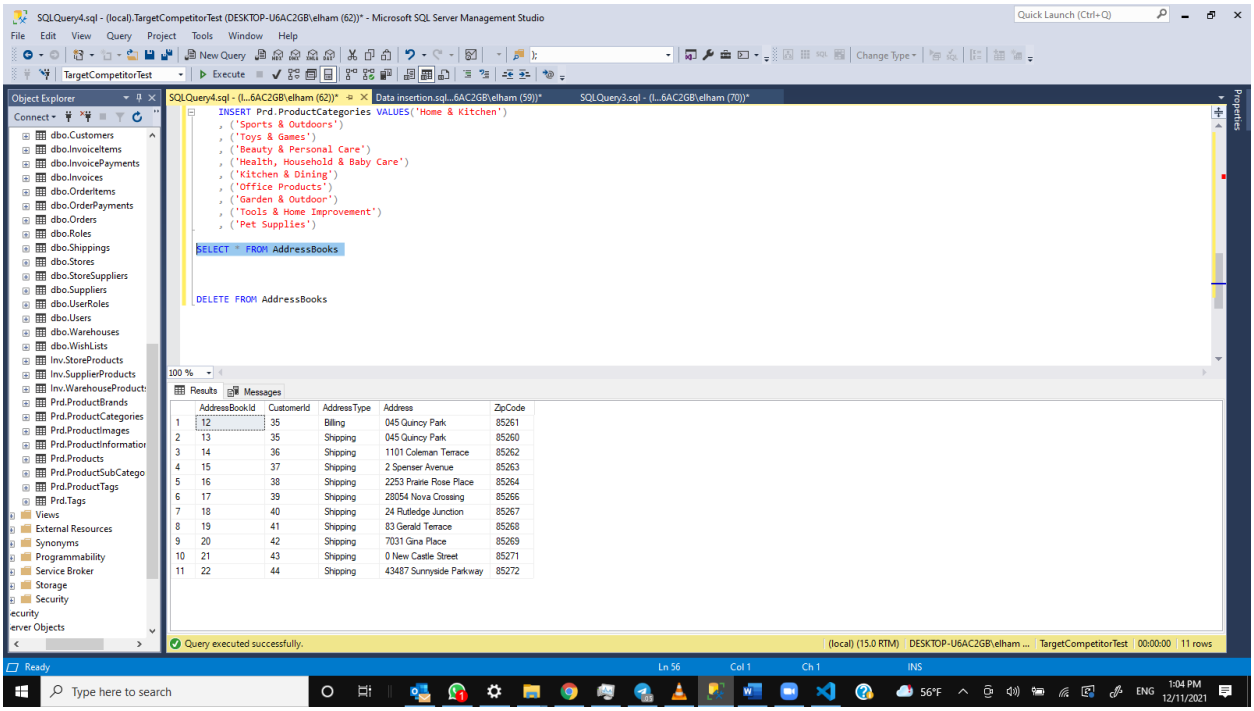
SELECT * FROM Prd.ProductCategories



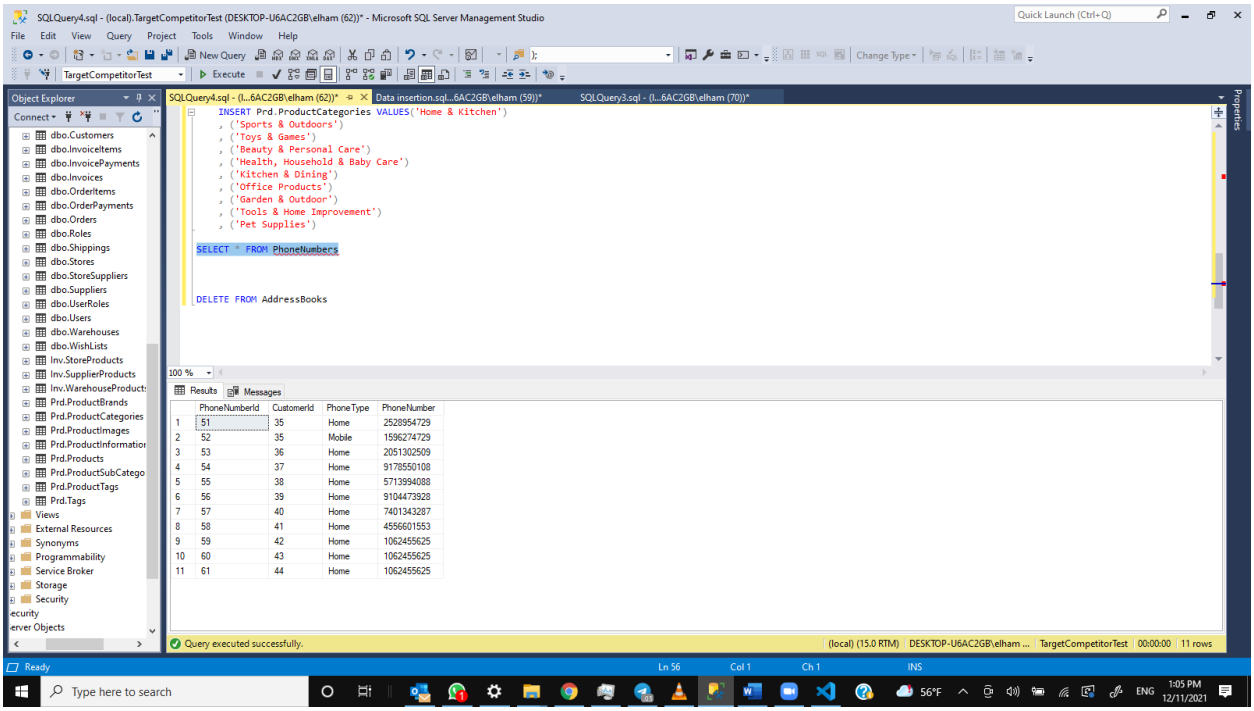
SELECT * FROM Customers



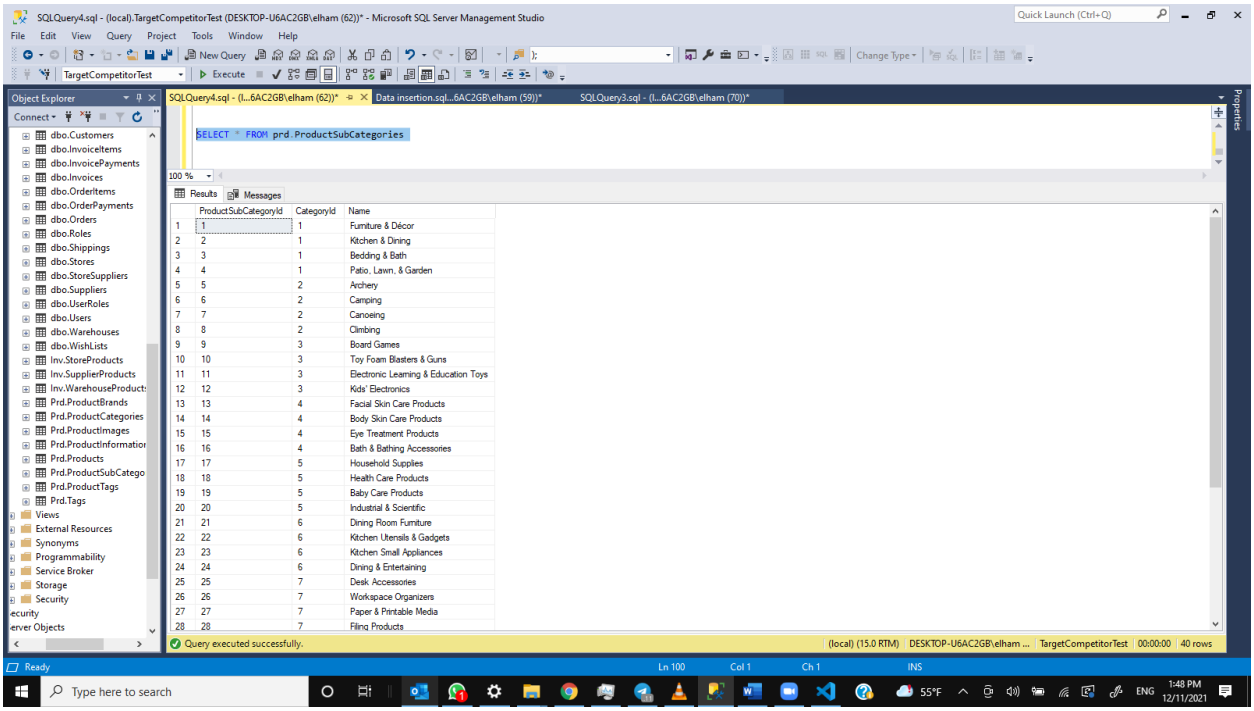
SELECT * FROM AddressBooks



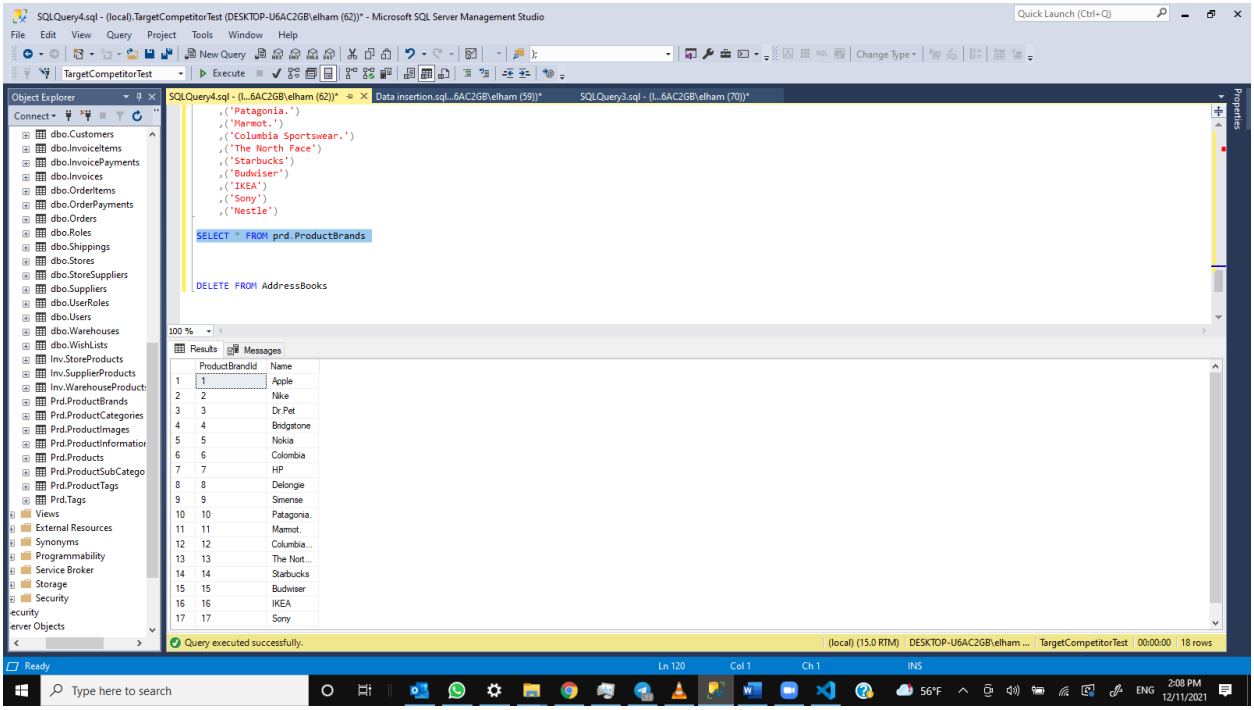
SELECT * FROM PhoneNumbers



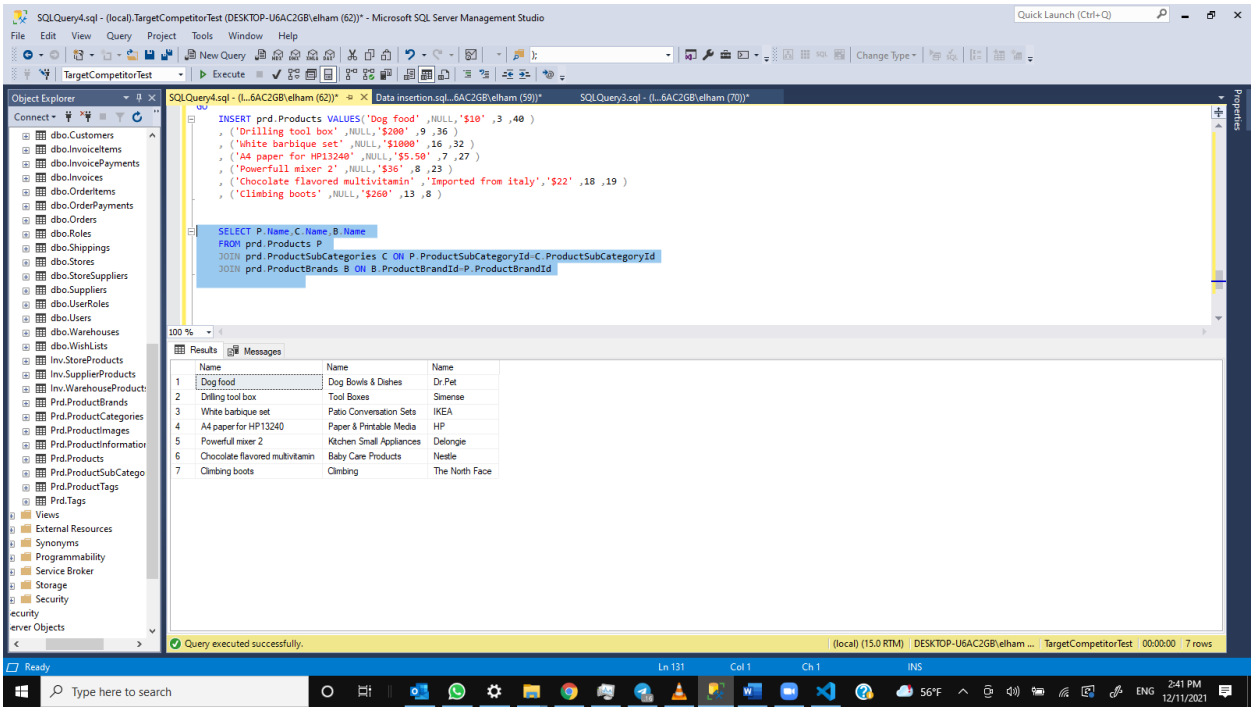
```
SELECT * FROM prd.ProductSubCategories
```



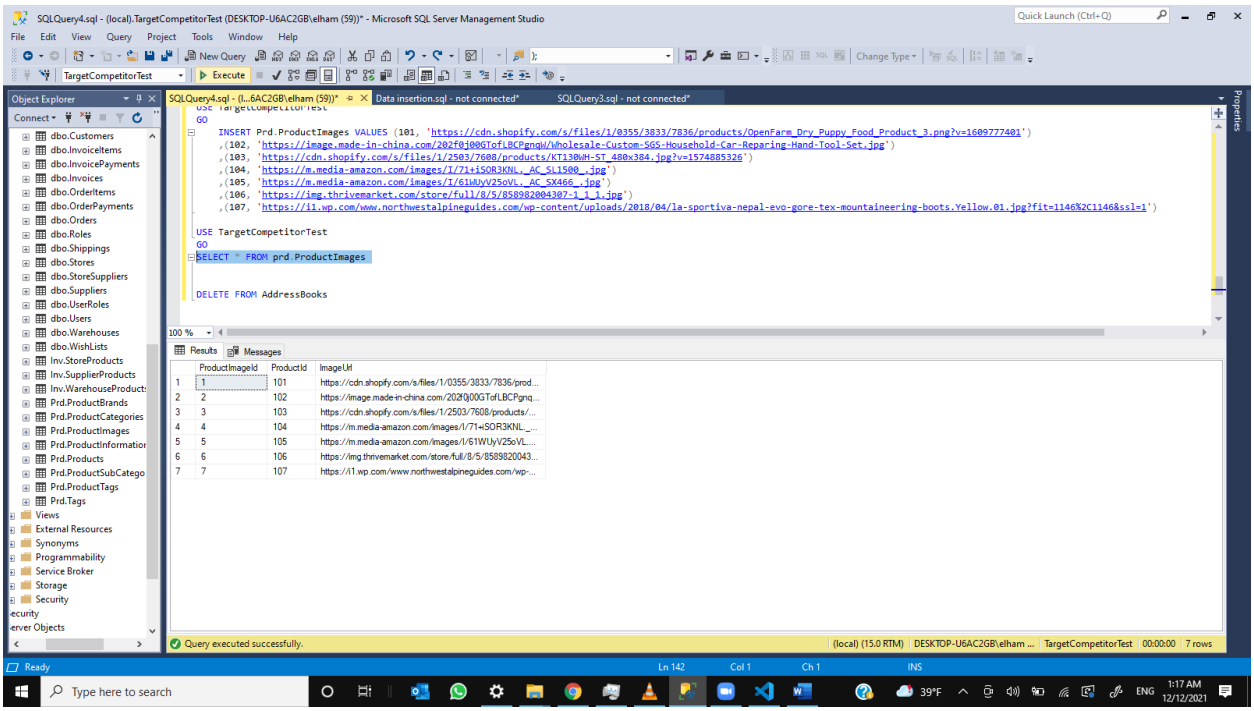

```
SELECT * FROM prd.ProductBrands
```



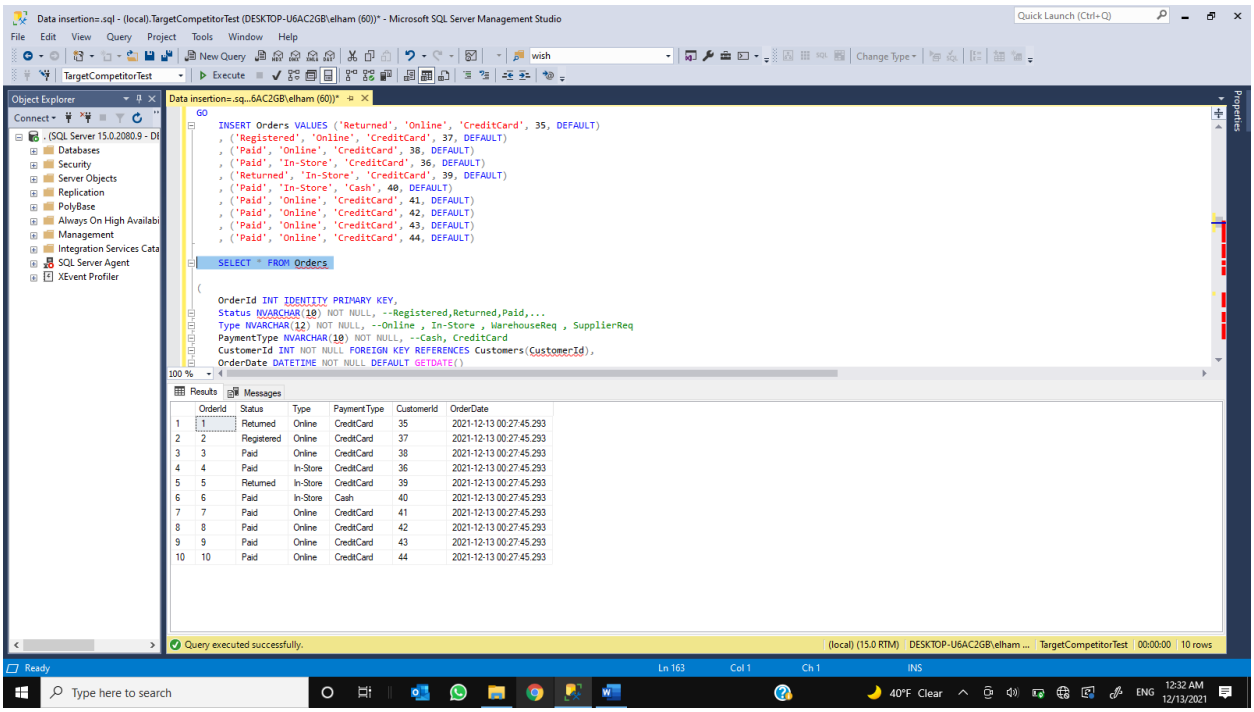
```
SELECT P.Name,C.Name,B.Name
FROM prd.Products P
JOIN prd.ProductSubCategories C ON P.ProductSubCategoryId=C.ProductSubCategoryId
JOIN prd.ProductBrands B ON B.ProductBrandId=P.ProductBrandId
```



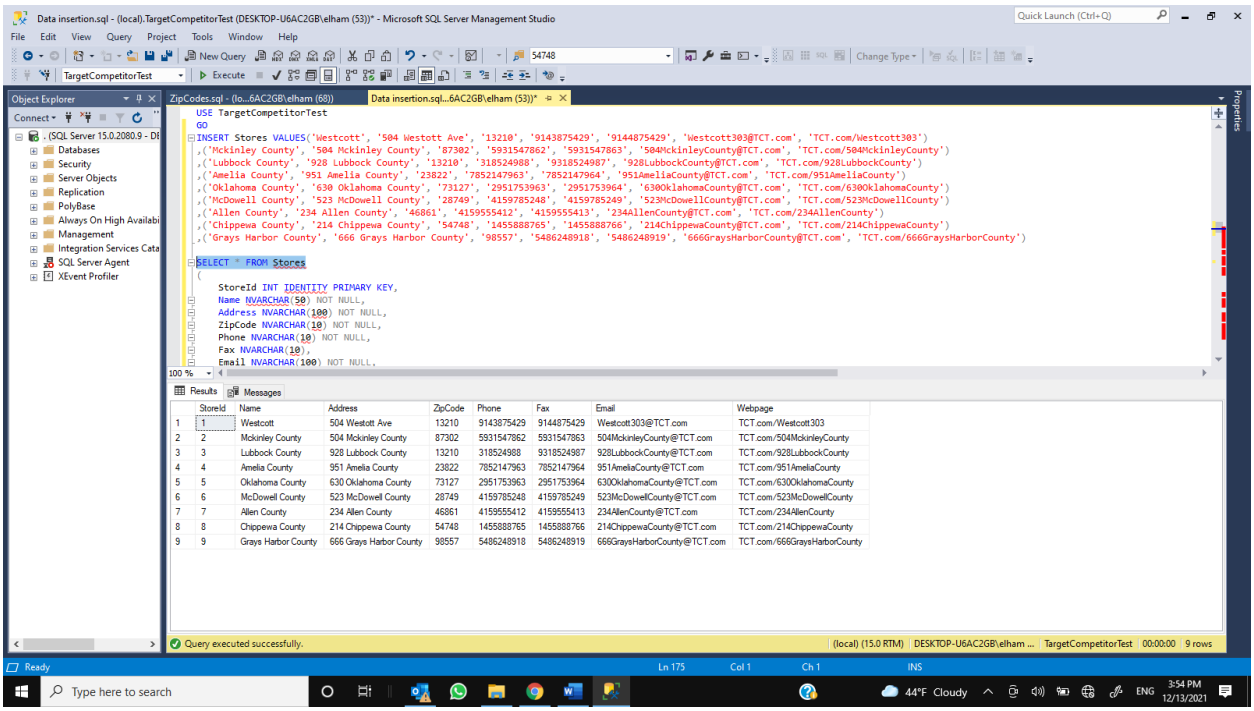

```
SELECT * FROM prd.ProductImages
```



SELECT * FROM Orders

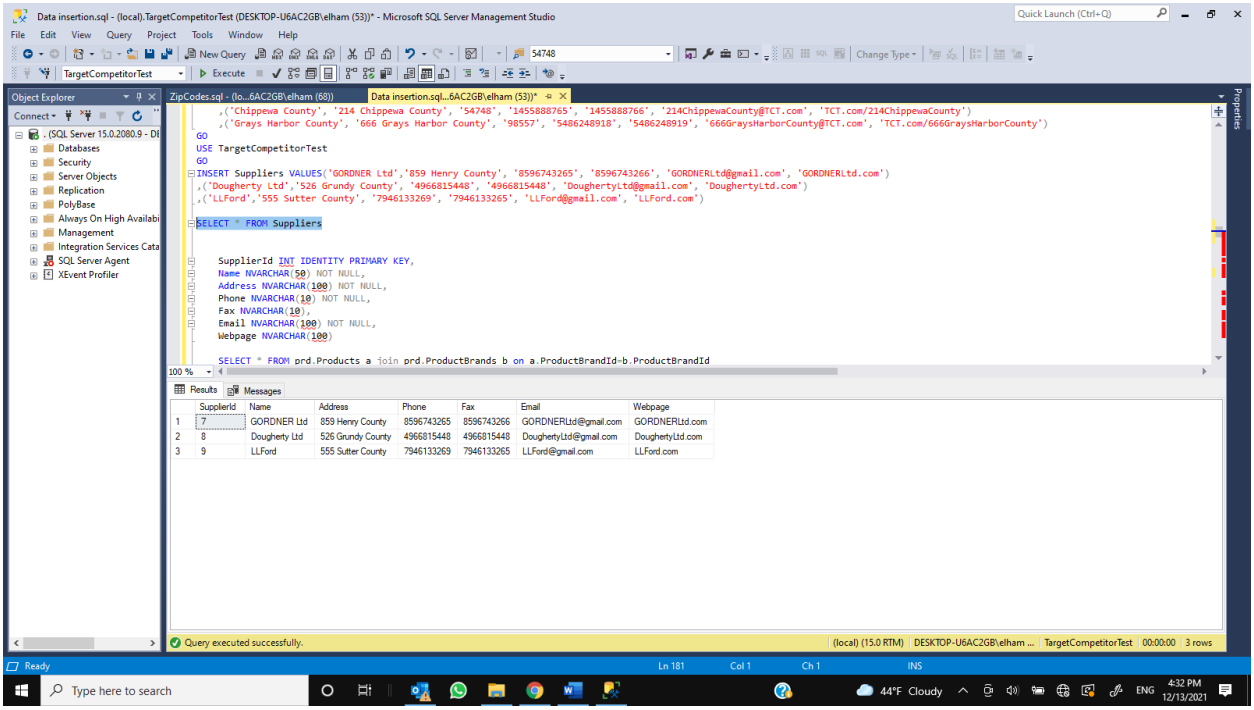


SELECT * FROM Stores



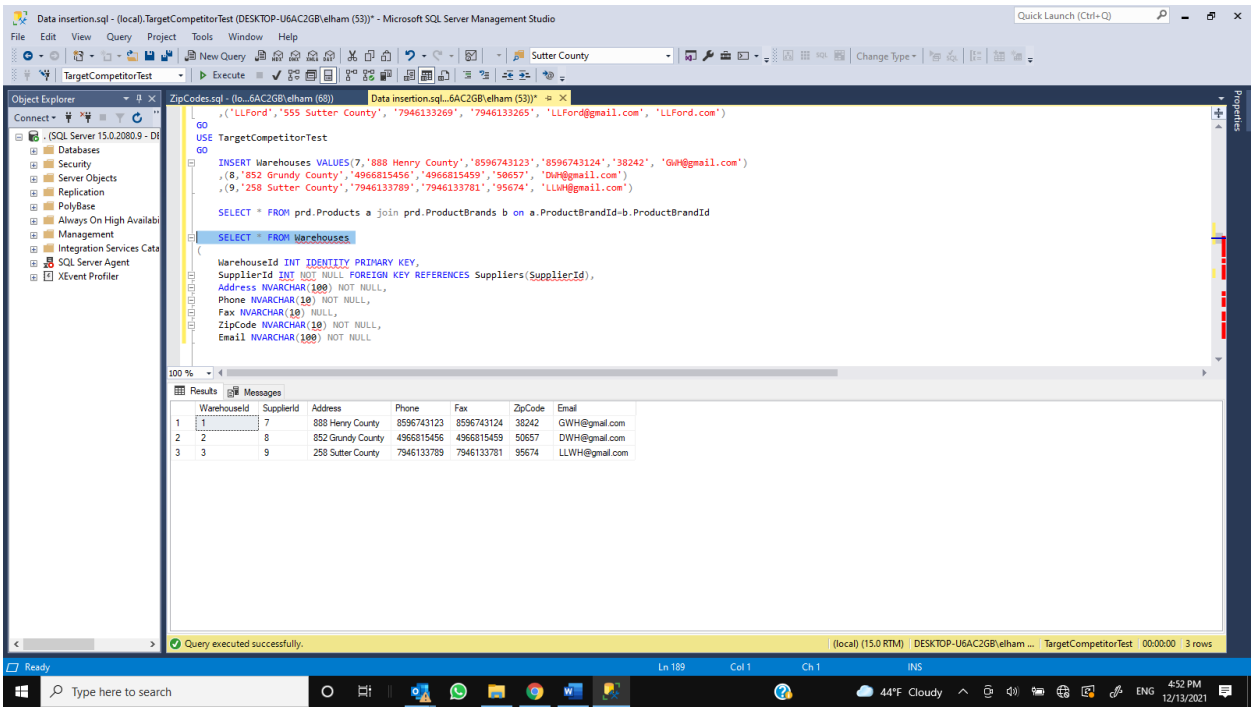
	StoreId	Name	Address	ZipCode	Phone	Fax	Email	Webpage
1	1	Westcott	504 Westott Ave	13210	9143875429	9144875429	Westcott303@TCT.com	TCT.com/Westcott303
2	2	Mckinley County	504 Mckinley County	87302	5931547862	5931547863	504MckinleyCounty@TCT.com	TCT.com/504MckinleyCounty
3	3	Lubbock County	928 Lubbock County	13210	318524988	9318524987	928LubbockCounty@TCT.com	TCT.com/928LubbockCounty
4	4	Amelia County	951 Amelia County	23822	7852147963	7852147964	951AmeliaCounty@TCT.com	TCT.com/951AmeliaCounty
5	5	Oklahoma County	630 Oklahoma County	73127	2951753963	2951753964	630OklahomaCounty@TCT.com	TCT.com/630OklahomaCounty
6	6	McDowell County	523 McDowell County	28749	4159785248	4159785249	523McDowellCounty@TCT.com	TCT.com/523McDowellCounty
7	7	Allen County	234 Allen County	46861	4159555412	4159555413	234AllenCounty@TCT.com	TCT.com/234AllenCounty
8	8	Chippewa County	214 Chippewa County	54748	1455888765	1455888766	214ChippewaCounty@TCT.com	TCT.com/214ChippewaCounty
9	9	Grays Harbor County	666 Grays Harbor County	98557	5486248918	5486248919	666GraysHarborCounty@TCT.com	TCT.com/666GraysHarborCounty

SELECT * FROM Suppliers



	SupplierId	Name	Address	Phone	Fax	Email	Webpage
1	7	GORDNER Ltd	859 Henry County	8596743265	8596743266	GORDNERLtd@gmail.com	GORDNERLtd.com
2	8	Dougherty Ltd	526 Grundy County	4966815448	4966815448	DoughertyLtd@gmail.com	DoughertyLtd.com
3	9	LLFord	555 Sutter County	7946133269	7946133265	LLFord@gmail.com	LLFord.com

SELECT * FROM Warehouses


Data insertion.sql - (local).TargetCompetitorTest (DESKTOP-U6AC2GB\elham (53))* - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query
Sutter County
Change Type
TargetCompetitorTest
Execute
Object Explorer
Connect
(SQL Server 15.0.2080.9 - DE
Databases
Security
Server Objects
Replication
PolyBase
Always On High Availabi
Management
Integration Services Cata
SQL Server Agent
XEvent Profiler
ZipCodes.sql - (lo...6AC2GB\elham (68))
Data insertion.sql...6AC2GB\elham (53))*
,('LLFord','555 Sutter County', '7946133269', '7946133265', 'LLFord@gmail.com', 'LLFord.com')
GO
USE TargetCompetitorTest
GO
INSERT Warehouses VALUES(7,'888 Henry County','8596743123','8596743124','38242', 'GWH@gmail.com')
,(8,'852 Grundy County','4966815456','4966815459','50657', 'DWH@gmail.com')
,(9,'258 Sutter County','7946133789','7946133781','95674', 'LLWH@gmail.com')
SELECT * FROM prd.Products a join prd.ProductBrands b on a.ProductBrandId=b.ProductBrandId
SELECT * FROM Warehouses
(
WarehouseId INT IDENTITY PRIMARY KEY,
SupplierId INT NOT NULL FOREIGN KEY REFERENCES Suppliers(SupplierId),
Address NVARCHAR(100) NOT NULL,
Phone NVARCHAR(10) NOT NULL,
Fax NVARCHAR(10) NULL,
ZipCode NVARCHAR(10) NOT NULL,
Email NVARCHAR(100) NOT NULL
100 %
Results
Messages
WarehouseId SupplierId Address Phone Fax ZipCode Email
1 1 7 888 Henry County 8596743123 8596743124 38242 GWH@gmail.com
2 2 8 852 Grundy County 4966815456 4966815459 50657 DWH@gmail.com
3 3 9 258 Sutter County 7946133789 7946133781 95674 LLWH@gmail.com
Query executed successfully.
(local) (15.0 RTM) DESKTOP-U6AC2GB\elham ... TargetCompetitorTest 00:00:00 3 rows
Ready
Ln 189 Col 1 Ch 1 INS
Type here to search
44°F Cloudy
ENG
4:52 PM
12/13/2021