A Database Design for Commodity Retailers

(Designed by Haoxiang Wang, SUID:482006282)
(02/12/2021, at Syracuse University)
(Email: hwang214@syr.edu)

Abstract: This report shows the design of a database model for the target's potential competitor Tarbet. It supports the storage of customer data, store data, warehouse data, supplier data, product data, order data, and the management of operations such as online sales, in-store sales, product restocking, logistics and transportation based on minimum distance principle, and customer returns. In addition, the security and stability of the database are guaranteed by designing multiple stored procedures. Users in different roles call different stored procedures. All users can complete their operations through stored procedures. This prevents the user from directly manipulating the data.

Key Words: Sales database, shopping database, database security, Target competitor

CONTENTS

1. Design

1.1 Introduction

With the development of the Internet, online shopping has become an important way of shopping. Customers can go to physical stores to select products, or they can place orders directly online. The traditional in-store shopping model has been transformed into a composite shopping model that integrates in-store and online. Under this trend, many merchandise retailers have established online and in-store integrated sales systems and Target is the top one. The complexity of the sales system brings great challenges to the design of the database. This report explains the problems that the sales system needs to solve from the aspects of data storage, operation management, data security, etc., and gives a feasible database design.

1.2 Information and E/R Diagram

The information that needs to be stored:

Customer

Customers: CustomerID, FirstName, LastName, UserName, Email, HomePhone, CellPhone, BusinessPhone
ShippingAddressList: CustomerID, AddressID
BillingAddressList: CustomerID, AddressID
CreditCards: CardID, CardNum, OwnerName, ExpireDate
CardList: CustomerID, CardID
WishList: CustomerID, ProductID, Quantity

Product

Products: ProductID, ProductName, Description, UnitPrice, AvgScore
Reviews: ReviewID, ProductID, CustomerID, Text, Score, ReviewDate

Supplier

Suppliers: SupplierID, SupplierName, UserName, AddressID, Phone, Fax, Email, Webpage
ProductDetail: ProductID, SupplierID, AvailableNumber

Warehouse

Warehouses: WarehouseID, UserName, AddressID, Phone, Fax, Email
WarehouseProductList: WarehouseID, ProductID, InStock, OnWay, InReturn

Store

Stores: StoreID, UserName, AddressID, Phone, Fax, Email
StoreProductList: StoreID, ProductID, Price, Quantity

Order

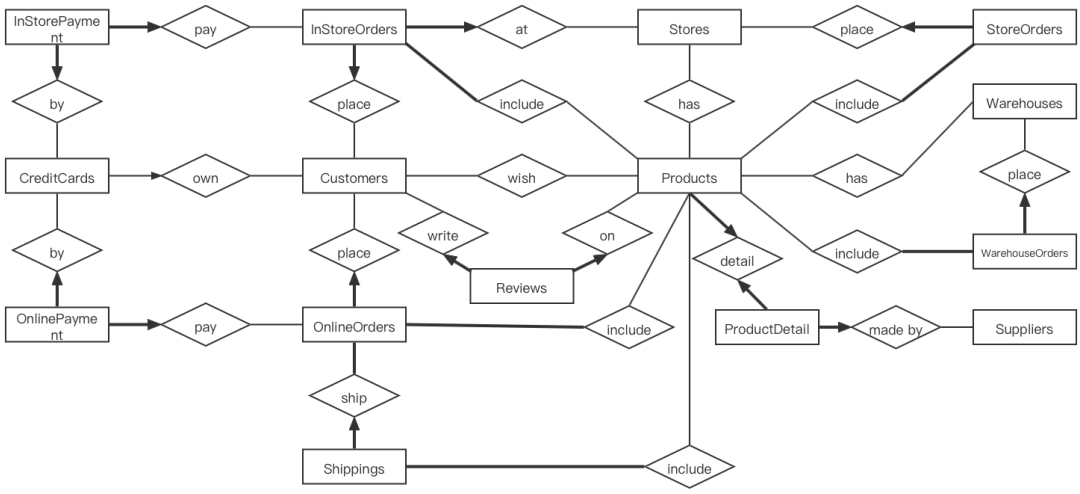
InStoreOrders: OrderID, CustomerID, OrderDate, StoreID, TotalPrice
InStoreOrderToItem: OrderID, ProductID, Quantity, UnitPrice
InStorePayment: PaymentID, OrderID, PaymentTotal, CardNumber, PayDate
OnlineOrders: OrderID, CustomerID, Locked, OrderDate, TotalPrice, PayDueDate, AddressID
OnlineOrderToItem: OrderID, ProductID, Quantity, UnitPrice
OnlinePayment: PaymentID, OrderID, PaymentTotal, CardNumber, PayDate
StoreOrders: OrderID, StoreID, OrderDate, TotalPrice
StoreOrderToItem: OrderID, ProductID, Quantity, UnitPrice
WarehouseOrders: OrderID, WarehouseID, OrderDate, TotalPrice
WarehouseOrderToItem: OrderID, ProductID, Quantity, UnitPrice

Shipping

Shippings: ShippingID, OrderID, ShippingService (USPS, FedEx, UPS), Status (Ready, Shipped, Delivered, Returned), StartAddressID, ArriveAddressID, ShippingFare, StartDate, ExpectedShippingDate, ActualShippingDate
ShippingToItem: ShippingID, ProductID, Quantity

Addresses

Addresses: AddressID, Street, City, State, Zip
ZipList: Zip, Latitude, Longitude



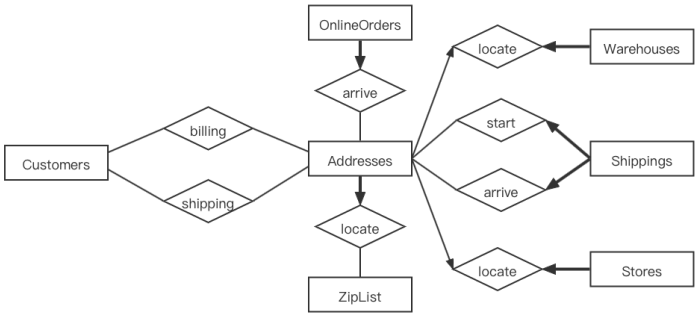


Figure 1. E/R Diagram of Tarbet

As it shows in Figure 1, each address has a Zip code. Each customer can have several billing addresses and shipping addresses. Each online order has one arrival address. Each Shipping has one start address and one arrival address. Each Store has one address. Each warehouse has one address. Each customer can own serval credit cards, online orders, and in-store orders. Each online order can have serval shippings. Each shippings can have serval products. Each store can have serval products and serval store orders. Each warehouse can have several products and serval warehouse orders. Each in-store order / online order / store order / warehouse order can have several products. Each supplier can supply several products, but each product can be supplied by one supplier. Each customer can write several reviews for one product.

1.3 Business Logic and Transactions

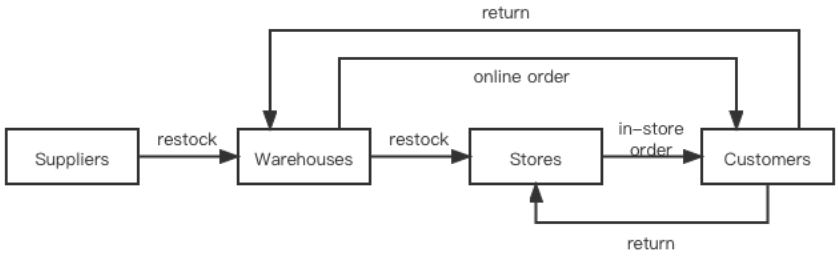


Figure 2. Product Flow

As it shows in Figure 2, customers can buy items by online orders or in-store orders. If customers buy items online, the system needs to schedule shipping for this order. An online order can correspond to several shipping. For each item in the order, the system should schedule the nearest available warehouse. If customers buy items in the store, the quantity of products of that store should be changed. Besides, the store can restock products from the warehouse by placing store orders. The system should schedule the nearest available warehouse for each product. Also, the warehouse can restock products from suppliers by placing warehouse orders.

Customer's Online Orders and Returns

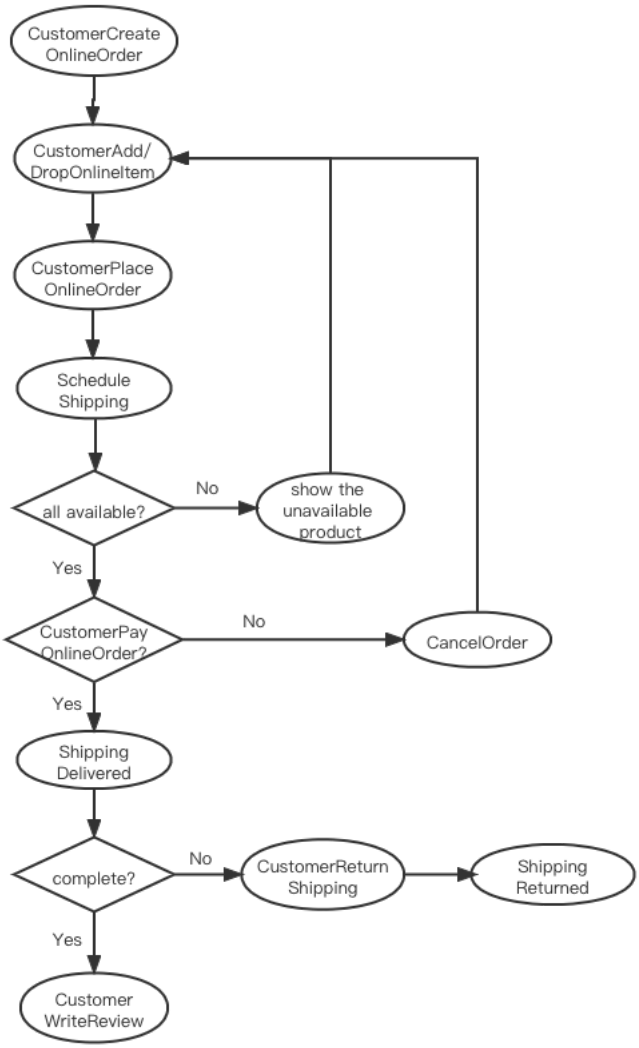


Figure 3. Online Order Flow

CustomerCreateOnlineOrder: Customer creates an order with Locked = 0.

CustomerAddOnlineItem: Customer adds items to his order.

CustomerDropOnlineItem: Customer drops items in his order.

CustomerPlaceOnlineOrder: Customer places his order.
Set Locked = 1, OrderDate = current time, and PayDueDate = OrderDate + 1 day.

ScheduleShipping: Schedule shipping.
If there are some items that are available, show the unavailable items and set Locked = 0. Else, create shippings with status = ready. For any item in the order, decrease InStock and add OnWay of items in its nearest available warehouse.

CustomerPayOnlineOrder: Customer pays his order.

If the customer pays before the due date, set shipping status = shipped. Otherwise, throw the ERROR 50003 “Order expired”.

CancelShipping: Cancel shipping of an order.
If the customer doesn’t pay before the due date, the system will cancel all shippings of this order.

ShippingDelivered: A shipping is delivered.
Set shipping status = delivered. Decrease OnWay of items in the warehouse.

CustomerReturnShipping: Customer returns shipping.
Add InReturn of items of the warehouse.

ShippingReturned: A shipping is returned.
Set shipping status = returned. Decrease InReturn of items of the warehouse. If items are intact, add InStock of the warehouse and return money back to customer.

CustomerWriteReview: Customer writes a review on a product.

Customer’s In-store Orders and Returns

Figure 4. In-Store Order Flow

CustomerCreateInStoreOrder: Customer creates an order.

CustomerAddInStoreItem/CustomerDropInStoreItem: Customer adds items to order.

CustomerPayInStoreOrder: Customer pays his order.
Decrease the quantity of items in this store.

CustomerReturnProduct: Customer returns an item.

If items are intact, add the quantity of this item of the store and return the money back to the customer.

Store/Warehouse Restocks Products

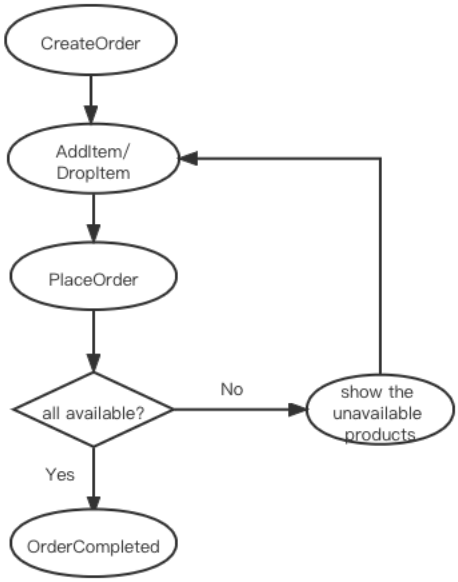


Figure 5. Restocking Flow

StoreCreateOrder / WarehouseCreateOrder: Store / Warehouse creates an order.

StoreAddItem / WarehouseAddItem: Store / Warehouse adds items to its order.

StoreDropItem / WarehouseDropItem: Store / Warehouse drops items in its order.

StorePlaceOrder / WarehousePlaceOrder: Store / Warehouse places its order.
Change the quantity of products of warehouses / suppliers. If there are some unavailable products, roll back the data and show the unavailable products.

StoreOrderCompleted / WarehouseOrderCompleted: Store's / Warehouse's order is completed.
Change the quantity of products in store / warehouse.

The shipping process of store/warehouse restocking is like the shipping process of online orders, so I omit it in my database. Users can add it according to their needs.

1.4 Business Report

In the Tarbet database, there are several roles. Each role has its own requirements and permissions. Customer can view their personal information and orders, edit their personal information, create and edit online or in-store orders, place their orders, and pay their orders. The store can view its products and quantity, create, edit, and place store orders to restock products from warehouses. The warehouse can view its products and the number of InStock, OnWay, InReturn, create, edit, and place warehouse orders to restock products from suppliers. The supplier can view its products and available number, create new products, and restock products. The administrator of Tarbet can change the status of shippings, view and edit all the user tables in the database. The database owner has all permissions in the database. Figure 6 shows the permissions of each role.

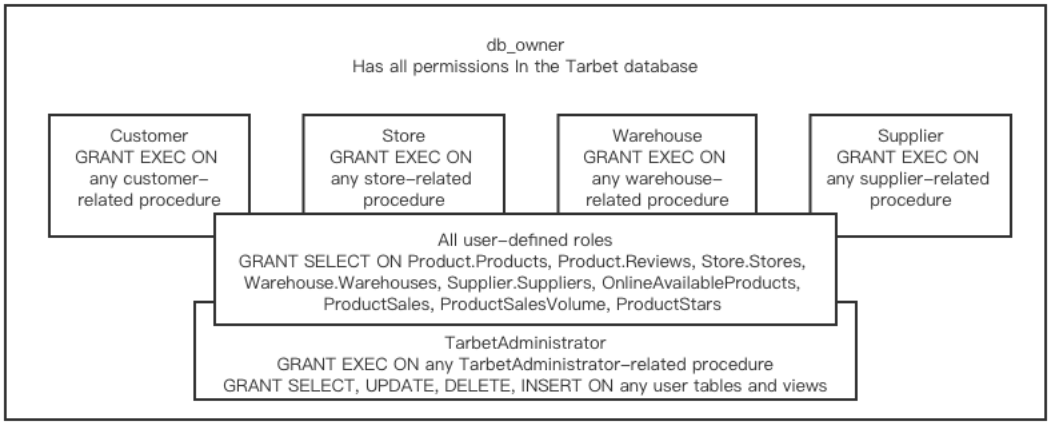


Figure 6. Database Roles and Their Permissions

Please refer to Appendix III Roles and Permissions for the detailed permissions of each role.

2. Implementation

2.1 Normalize Design into 3NF

Normalize tables into the Third Normal Form (3NF) as:

Customers	ShippingAddressList	BillingAddressList	CreditCards
CustomerID	CustomerID	CustomerID	CardID
FirstName	AddressID	AddressID	CardNum
LastName			OwnerName
UserName			ExpireDate
Email			
HomePhone			
CellPhone			
BusinessPhone			

CardList	WishList	Products	Reviews
CustomerID	CustomerID	ProductID	ReviewID
CardID	ProductID	ProductName	ProductID
	Quantity	Description	CustomerID
		UnitPrice	Text
		AvgScore	Score
			ReviewDate

Suppliers	ProductDetail	Warehouses	WarehouseProductList
SupplierID	ProductID	WarehouseID	WarehouseID
SupplierName	SupplierID	UserName	ProductID
UserName	AvailableNumber	AddressID	InStock
AddressID		Phone	OnWay
Phone		Fax	InReturn
Fax		Email	
Email			
Webpage			

Stores	StoreProductList	InStoreOrders	InStoreOrderToItem
StoreID	StoreID	OrderID	OrderID
UserName	ProductID	CustomerID	ProductID
AddressID	Price	OrderDate	Quantity
Phone	Quantity	StoreID	UnitPrice
Fax		TotalPrice	
Email			

InStorePayment	OnlineOrders	OnlineOrderToItem	OnlinePayment
PaymentID	OrderID	OrderID	PaymentID
OrderID	CustomerID	ProductID	OrderID

PaymentTotal CardNumber PayDate	Locked OrderDate TotalPrice PayDueDate AddressID	Quantity UnitPrice	PaymentTotal CardNumber PayDate

StoreOrders	StoreOrderToItem	WarehouseOrders	WarehouseOrderToItem
OrderID	OrderID	OrderID	OrderID
StoreID	ProductID	WarehouseID	ProductID
OrderDate	Quantity	OrderDate	Quantity
TotalPrice	UnitPrice	TotalPrice	UnitPrice

Shippings	ShippingToItem	Addresses	ZipList
ShippingID	ShippingID	AddressID	Zip
OrderID	ProductID	Street	Latitude
ShippingService	Quantity	City	Longitude
Status		State	
StartAddressID		Zip	
ArriveAddressID			
ShippingFare			
StartDate			
ExpectedShippingDate			
ActualShippingDate			

2.2 Create Schemas and Tables

Create schedules, tables, and their constraints in the Tarbet database:

```
/****** Object:  Database Target2    ******/
CREATE DATABASE Tarbet
GO

USE Tarbet
GO

/****** Object:  SCHEMA Customer    ******/
CREATE SCHEMA Customer;
GO

/****** Object:  SCHEMA Address    ******/
CREATE SCHEMA Address;
GO

/****** Object:  SCHEMA Product    ******/
```

```
CREATE SCHEMA Product;
GO

/****** Object:  SCHEMA Order_    ******/
CREATE SCHEMA Order_;
GO

/****** Object:  SCHEMA Shipping    ******/
CREATE SCHEMA Shipping;
GO

/****** Object:  SCHEMA Store    ******/
CREATE SCHEMA Store;
GO

/****** Object:  SCHEMA Warehouse    ******/
CREATE SCHEMA Warehouse;
GO

/****** Object:  SCHEMA Supplier    ******/
CREATE SCHEMA Supplier;
GO

/****** Object:  Table ZipList    ******/
CREATE TABLE Address.ZipList(
    Zip char(5) NOT NULL,
    Latitude float NOT NULL,
    Longitude float NOT NULL,
    PRIMARY KEY (Zip),
    CONSTRAINT CHK_Zip CHECK (LEN(Zip) = 5),
    CONSTRAINT CHK_LatLng CHECK ((Latitude BETWEEN -90 AND 90) AND (Longitude
BETWEEN -180 AND 180))
)
GO

/****** Object:  Table Addresses    ******/
CREATE TABLE Address.Addresses(
    AddressID int IDENTITY(1, 1) NOT NULL,
    Street varchar(100) NOT NULL,
    City varchar(20) NOT NULL,
    State varchar(20) NOT NULL,
    Zip char(5) NOT NULL,
    PRIMARY KEY (AddressID),
    FOREIGN KEY (Zip) REFERENCES Address.ZipList(Zip)
```

```
)
GO

/****** Object:  Table Customers    ******/
CREATE TABLE Customer.Customers(
    CustomerID int IDENTITY(1, 1) NOT NULL,
    FirstName varchar(50) NOT NULL,
    LastName varchar(50) NOT NULL,
    UserName varchar(50) NOT NULL UNIQUE,
    Email varchar(100),
    HomePhone varchar(20),
    CellPhone varchar(20),
    BusinessPhone varchar(20),
    PRIMARY KEY (CustomerID)
)
GO

/****** Object:  Table ShippingAddressList    ******/
CREATE TABLE Customer.ShippingAddressList(
    CustomerID int NOT NULL,
    AddressID int NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customer.Customers(CustomerID),
    FOREIGN KEY (AddressID) REFERENCES Address.Addresses(AddressID)
)
GO

/****** Object:  Table BillingAddressList    ******/
CREATE TABLE Customer.BillingAddressList(
    CustomerID int NOT NULL,
    AddressID int NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customer.Customers(CustomerID),
    FOREIGN KEY (AddressID) REFERENCES Address.Addresses(AddressID)
)
GO

/****** Object:  Table CreditCards    ******/
CREATE TABLE Customer.CreditCards(
    CardID int IDENTITY(1, 1) NOT NULL,
    CardNum varchar(20) NOT NULL,
    OwnerName varchar(100),
    ExpireDate date,
    PRIMARY KEY (CardID)
)
GO
```

```
/****** Object:  Table CardList    ******/
CREATE TABLE Customer.CardList(
    CustomerID int NOT NULL,
    CardID int NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customer.Customers(CustomerID),
    FOREIGN KEY (CardID) REFERENCES Customer.CreditCards(CardID)
)
GO

/****** Object:  Table Products    ******/
CREATE TABLE Product.Products(
    ProductID int IDENTITY(1, 1) NOT NULL,
    ProductName varchar(50) NOT NULL,
    Description varchar(1000),
    UnitPrice money NOT NULL,
    AvgScore float,
    PRIMARY KEY (ProductID)
)
GO

/****** Object:  Table Reviews    ******/
CREATE TABLE Product.Reviews(
    ReviewID int IDENTITY(1, 1) NOT NULL,
    ProductID int NOT NULL,
    CustomerID int NOT NULL,
    Text varchar(1000),
    Score float NOT NULL,
    ReviewDate datetime NOT NULL,
    PRIMARY KEY (ReviewID),
    FOREIGN KEY (ProductID) REFERENCES Product.Products(ProductID),
    FOREIGN KEY (CustomerID) REFERENCES Customer.Customers(CustomerID),
    CONSTRAINT CHK_Score CHECK (Score BETWEEN 0.0 AND 5.0)
)
GO

/****** Object:  Table WishList    ******/
CREATE TABLE Customer.WishList(
    CustomerID int NOT NULL,
    ProductID int NOT NULL,
    Quantity int NOT NULL DEFAULT 1,
    FOREIGN KEY (CustomerID) REFERENCES Customer.Customers(CustomerID),
    FOREIGN KEY (ProductID) REFERENCES Product.Products(ProductID),
    CONSTRAINT CHK_WishQuantity CHECK (Quantity >= 0)
```

```
)
GO


/****** Object:  Table Suppliers    ******/
CREATE TABLE Supplier.Suppliers(
    SupplierID int IDENTITY(1, 1) NOT NULL,
    SupplierName varchar(50) NOT NULL,
    UserName varchar(50) NOT NULL UNIQUE,
    AddressID int NOT NULL UNIQUE,
    Phone varchar(20),
    Fax varchar(20),
    Email varchar(100),
    Webpage varchar(100),
    PRIMARY KEY (SupplierID),
    FOREIGN KEY (AddressID) REFERENCES Address.Addresses(AddressID)
)
GO

/****** Object:  Table ProductDetail    ******/
CREATE TABLE Supplier.ProductDetail(
    ProductID int NOT NULL,
    SupplierID int NOT NULL,
    AvailableNumber int NOT NULL DEFAULT 0,
    PRIMARY KEY (ProductID),
    FOREIGN KEY (ProductID) REFERENCES Product.Products(ProductID),
    FOREIGN KEY (SupplierID) REFERENCES Supplier.Suppliers(SupplierID),
    CONSTRAINT CHK_SupplierAvailable CHECK (AvailableNumber >= 0)
)
GO

/****** Object:  Table Warehouses    ******/
CREATE TABLE Warehouse.Warehouses(
    WarehouseID int IDENTITY(1, 1) NOT NULL,
    UserName varchar(50) NOT NULL UNIQUE,
    AddressID int NOT NULL UNIQUE,
    Phone varchar(20),
    Fax varchar(20),
    Email varchar(100),
    PRIMARY KEY (WarehouseID),
    FOREIGN KEY (AddressID) REFERENCES Address.Addresses(AddressID)
)
GO

/****** Object:  Table WarehouseProductList    ******/
```

```
CREATE TABLE Warehouse.WarehouseProductList(
   WarehouseID int NOT NULL,
   ProductID int NOT NULL,
   InStock int NOT NULL DEFAULT 0,
   OnWay int NOT NULL DEFAULT 0,
   InReturn int NOT NULL DEFAULT 0,
   PRIMARY KEY (WarehouseID, ProductID),
   FOREIGN KEY (WarehouseID) REFERENCES Warehouse.Warehouses(WarehouseID),
   FOREIGN KEY (ProductID) REFERENCES Product.Products(ProductID),
   CONSTRAINT CHK_WarehouseAvailable CHECK (InStock >= 0 AND OnWay >=0 AND
InReturn >=0)
)
GO

/****** Object:  Table Stores    ******/
CREATE TABLE Store.Stores(
   StoreID int IDENTITY(1, 1) NOT NULL,
   UserName varchar(50) NOT NULL UNIQUE,
   AddressID int NOT NULL UNIQUE,
   Phone varchar(20),
   Fax varchar(20),
   Email varchar(100),
   PRIMARY KEY (StoreID),
   FOREIGN KEY (AddressID) REFERENCES Address.Addresses(AddressID)
)
GO

/****** Object:  Table StoreProductList    ******/
CREATE TABLE Store.StoreProductList(
   StoreID int NOT NULL,
   ProductID int NOT NULL,
   Price money NOT NULL,
   Quantity int NOT NULL,
   PRIMARY KEY (StoreID, ProductID),
   FOREIGN KEY (StoreID) REFERENCES Store.Stores(StoreID),
   FOREIGN KEY (ProductID) REFERENCES Product.Products(ProductID),
   CONSTRAINT CHK_StoreQuantity CHECK (Quantity >= 0)
)
GO

/****** Object:  Table InStoreOrders    ******/
CREATE TABLE Order_.InStoreOrders(
   OrderID int IDENTITY(1, 1) NOT NULL,
   CustomerID int NOT NULL,
```

```
    OrderDate datetime NOT NULL,
    StoreID int NOT NULL,
    TotalPrice money NOT NULL DEFAULT 0,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (CustomerID) REFERENCES Customer.Customers(CustomerID),
    FOREIGN KEY (StoreID) REFERENCES Store.Stores(StoreID)
)
GO

/****** Object:  Table InStoreOrderToItem    ******/
CREATE TABLE Order_.InStoreOrderToItem(
    OrderID int NOT NULL,
    ProductID int NOT NULL,
    Quantity int NOT NULL,
    UnitPrice money NOT NULL,
    FOREIGN KEY (OrderID) REFERENCES Order_.InStoreOrders(OrderID) ON DELETE
CASCADE,
    FOREIGN KEY (ProductID) REFERENCES Product.Products(ProductID),
    CONSTRAINT CHK_InStoreOrderQuantity CHECK (Quantity >= 0)
)
GO

/****** Object:  Table InStorePayment    ******/
CREATE TABLE Order_.InStorePayment(
    PaymentID int IDENTITY(1, 1) NOT NULL,
    OrderID int NOT NULL,
    PaymentTotal money NOT NULL,
    CardNumber varchar(20) NOT NULL,
    PayDate datetime NOT NULL,
    PRIMARY KEY (PaymentID),
    FOREIGN KEY (OrderID) REFERENCES Order_.InStoreOrders(OrderID),
)
GO

/****** Object:  Table OnlineOrders    ******/
CREATE TABLE Order_.OnlineOrders(
    OrderID int IDENTITY(1, 1) NOT NULL,
    CustomerID int NOT NULL,
    Locked bit NOT NULL DEFAULT 0,
    OrderDate datetime,
    TotalPrice money NOT NULL DEFAULT 0,
    PayDueDate datetime,
    AddressID int,
    PRIMARY KEY (OrderID),
```

```
    FOREIGN KEY (CustomerID) REFERENCES Customer.Customers(CustomerID),
    FOREIGN KEY (AddressID) REFERENCES Address.Addresses(AddressID),
    CONSTRAINT CHK_PayDueDate CHECK (PayDueDate > OrderDate)
)
GO

/****** Object:  Table OnlineOrderToItem    ******/
CREATE TABLE Order_.OnlineOrderToItem(
    OrderID int NOT NULL,
    ProductID int NOT NULL,
    Quantity int NOT NULL,
    UnitPrice money NOT NULL,
    FOREIGN KEY (OrderID) REFERENCES Order_.OnlineOrders(OrderID) ON DELETE
CASCADE,
    FOREIGN KEY (ProductID) REFERENCES Product.Products(ProductID),
    CONSTRAINT CHK_OnlineOrderQuantity CHECK (Quantity >= 0)
)
GO

/****** Object:  Table OnlinePayment    ******/
CREATE TABLE Order_.OnlinePayment(
    PaymentID int IDENTITY(1, 1) NOT NULL,
    OrderID int NOT NULL,
    PaymentTotal money NOT NULL,
    CardNumber varchar(20) NOT NULL,
    PayDate datetime NOT NULL,
    PRIMARY KEY (PaymentID),
    FOREIGN KEY (OrderID) REFERENCES Order_.OnlineOrders(OrderID),
)
GO

/****** Object:  Table Shippings    ******/
CREATE TABLE Shipping.Shippings(
    ShippingID int IDENTITY(1, 1) NOT NULL,
    OrderID int NOT NULL,
    ShippingService varchar(20) NOT NULL,
    Status varchar(10) NOT NULL,
    StartAddressID int NOT NULL,
    ArriveAddressID int NOT NULL,
    ShippingFare money NOT NULL DEFAULT 0,
    StartDate datetime,
    ExpectedShippingDate datetime,
    ActualShippingDate datetime,
    PRIMARY KEY (ShippingID),
```

```
    FOREIGN KEY (OrderID) REFERENCES Order_.OnlineOrders(OrderID),
    FOREIGN KEY (StartAddressID) REFERENCES Address.Addresses(AddressID),
    FOREIGN KEY (ArriveAddressID) REFERENCES Address.Addresses(AddressID),
    CONSTRAINT CHK_ShippingService CHECK (ShippingService IN ('USPS',
'FedEx', 'UPS')),
    CONSTRAINT CHK_Status CHECK (Status IN ('Ready', 'Shipped', 'Delivered',
'Returned')),
)
GO

/****** Object:  Table ShippingToItem    ******/
CREATE TABLE Shipping.ShippingToItem(
    ShippingID int NOT NULL,
    ProductID int NOT NULL,
    Quantity int NOT NULL,
    PRIMARY KEY (ShippingID, ProductID),
    FOREIGN KEY (ShippingID) REFERENCES Shipping.Shippings(ShippingID),
    FOREIGN KEY (ProductID) REFERENCES Product.Products(ProductID),
    CONSTRAINT CHK_ShippingQuantity CHECK (Quantity >= 0)
)
GO

/****** Object:  Table StoreOrders    ******/
CREATE TABLE Order_.StoreOrders(
    OrderID int IDENTITY(1, 1) NOT NULL,
    StoreID int NOT NULL,
    OrderDate datetime,
    TotalPrice money NOT NULL DEFAULT 0,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (StoreID) REFERENCES Store.Stores(StoreID)
)
GO

/****** Object:  Table StoreOrderToItem    ******/
CREATE TABLE Order_.StoreOrderToItem(
    OrderID int NOT NULL,
    ProductID int NOT NULL,
    Quantity int NOT NULL,
    UnitPrice money NOT NULL,
    FOREIGN KEY (OrderID) REFERENCES Order_.StoreOrders(OrderID) ON DELETE
CASCADE,
    FOREIGN KEY (ProductID) REFERENCES Product.Products(ProductID),
    CONSTRAINT CHK_StoreOrderQuantity CHECK (Quantity >= 0)
)
```

```
GO

/****** Object:  Table WarehouseOrders    ******/
CREATE TABLE Order_.WarehouseOrders(
    OrderID int IDENTITY(1, 1) NOT NULL,
    WarehouseID int NOT NULL,
    OrderDate datetime,
    TotalPrice money NOT NULL DEFAULT 0,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (WarehouseID) REFERENCES Warehouse.Warehouses(WarehouseID)
)
GO

/****** Object:  Table WarehouseOrderToItem    ******/
CREATE TABLE Order_.WarehouseOrderToItem(
    OrderID int NOT NULL,
    ProductID int NOT NULL,
    Quantity int NOT NULL,
    UnitPrice money NOT NULL,
    FOREIGN KEY (OrderID) REFERENCES Order_.WarehouseOrders(OrderID) ON
DELETE CASCADE,
    FOREIGN KEY (ProductID) REFERENCES Product.Products(ProductID),
    CONSTRAINT CHK_WarehouseOrderQuantity CHECK (Quantity >= 0)
)
GO
```

2.3 Create Views

View OnlineAvailableProducts contains all the products that customer can buy online:

```
CREATE VIEW OnlineAvailableProducts AS
    SELECT DISTINCT ProductID
    FROM Warehouse.WarehouseProductList
    WHERE InStock > 0;
GO
```

View ProductSalesVolume contains the total online and in-store sales volume for each product.

```
CREATE VIEW ProductSalesVolume AS
    SELECT ISNULL (OnlineProductSaleVolume.ProductID,
InStoreProductSaleVolume.ProductID) AS ProductID,
        ISNULL (OnlineSaleVolume, 0) + ISNULL (InStoreSaleVolume, 0) AS
SalesVolume
    FROM (SELECT ProductID, SUM(UnitPrice * Quantity) AS OnlineSaleVolume
```

```
            FROM Order_.OnlineOrderToItem
            GROUP BY ProductID) AS OnlineProductSaleVolume
        FULL OUTER JOIN (
            SELECT ProductID, SUM(UnitPrice * Quantity) AS InStoreSaleVolume
            FROM Order_.InStoreOrderToItem
            GROUP BY ProductID) AS InStoreProductSaleVolume
        ON OnlineProductSaleVolume.ProductID =
InStoreProductSaleVolume.ProductID;
GO
```

View ProductStars gives stars for products based on their average score.

```
CREATE VIEW ProductStars AS
    SELECT ProductID, '5-stars Product' AS Grade
    FROM Product.Products
    WHERE AvgScore >= 4.7
UNION
    SELECT ProductID, '4-stars Product' AS Grade
    FROM Product.Products
    WHERE AvgScore < 4.7 AND AvgScore >= 4.4
UNION
    SELECT ProductID, '3-stars Product' AS Grade
    FROM Product.Products
    WHERE AvgScore < 4.4 AND AvgScore >= 3.0
GO
```

View ProductSales contains the total online and in-store sales for each product.

```
CREATE VIEW ProductSales AS
    SELECT ISNULL (OnlineProductSalesQuantity.ProductID,
InStoreProductSalesQuantity.ProductID) AS ProductID,
        ISNULL (OnlineSalesQuantity, 0) + ISNULL (InStoreSalesQuantity, 0) AS
Sales
    FROM (SELECT ProductID, SUM(Quantity) AS OnlineSalesQuantity
            FROM Order_.OnlineOrderToItem
            GROUP BY ProductID) AS OnlineProductSalesQuantity
        FULL OUTER JOIN (
            SELECT ProductID, SUM(Quantity) AS InStoreSalesQuantity
            FROM Order_.InStoreOrderToItem
            GROUP BY ProductID) AS InStoreProductSalesQuantity
        ON OnlineProductSalesQuantity.ProductID =
InStoreProductSalesQuantity.ProductID;
GO
```

2.4 Create Functions

Function fnDistance(FirstAddressID, SecondAddressID) returns the distance(km) between two addresses.

```
CREATE FUNCTION fnDistance
   (@FirstAddressID int, @SecondAddressID int)
   RETURNS float
BEGIN
   DECLARE @Zip1 char(5) = (
      SELECT Zip
      FROM Address.Addresses
      WHERE Address.Addresses.AddressID = @FirstAddressID);
   DECLARE @Zip2 char(5) = (
      SELECT Zip
      FROM Address.Addresses
      WHERE Address.Addresses.AddressID = @SecondAddressID);
   DECLARE @LAT1 float = (
      SELECT Latitude
      FROM Address.ZipList
      WHERE Address.ZipList.Zip = @Zip1);
   DECLARE @LONG1 float = (
      SELECT Longitude
      FROM Address.ZipList
      WHERE Address.ZipList.Zip = @Zip1);
   DECLARE @LAT2 float = (
      SELECT Latitude
      FROM Address.ZipList
      WHERE Address.ZipList.Zip = @Zip2);
   DECLARE @LONG2 float = (
      SELECT Longitude
      FROM Address.ZipList
      WHERE Address.ZipList.Zip = @Zip2);
   RETURN 111.045 * DEGREES(ACOS(COS(RADIANS(@LAT1))
              * COS(RADIANS(@LAT2))
              * COS(RADIANS(@LONG1) - RADIANS(@LONG2))
              + SIN(RADIANS(@LAT1))
              * SIN(RADIANS(@LAT2))));
END;
GO
```

Function fnWarehouseRankByDistance(AddressID) returns the rank of warehouses ordered by the distance between warehouses and arrival address.

```
CREATE FUNCTION fnWarehouseRankByDistance
   (@AddressID int)
```

```
    RETURNS TABLE
RETURN (
    SELECT ROW_NUMBER() OVER(ORDER BY dbo.fnDistance(@AddressID, AddressID))
AS Rank, WarehouseID
    FROM Warehouse.Warehouses);
GO
```

Function fnFindWarehouseByAddress(AddressID) returns the warehouse that is located at this address.

```
CREATE FUNCTION fnFindWarehouseByAddress
    (@AddressID int)
    RETURNS int
BEGIN
    RETURN (
        SELECT WarehouseID
        FROM Warehouse.Warehouses
        WHERE Warehouse.Warehouses.AddressID = @AddressID);
END;
GO
```

Function fnCheckAvailability(OrderID) returns the unavailable items in the online order.

```
CREATE FUNCTION fnCheckAvailability
    (@OrderID int)
    RETURNS TABLE
RETURN (
    SELECT ProductID
    FROM Order_.OnlineOrderToItem
    WHERE OrderID = @OrderID AND ProductID NOT IN (SELECT ProductID FROM
OnlineAvailableProducts));
GO
```

2.5 Create Stored Procedures

Stored procedure spCustomerCreateOnlineOrder creates a new online order for this customer and returns the OrderID.

```
CREATE PROC spCustomerCreateOnlineOrder
    @OrderID int OUTPUT,
    @CustomerID int,
    @AddressID int
AS
-- DECLARE @CustomerID int = dbo.fnCurrentCustomerID();
INSERT INTO Order_.OnlineOrders (CustomerID, AddressID)
VALUES (@CustomerID, @AddressID);
```

```
SET @OrderID = @@IDENTITY;
GO
```

Stored procedure spCustomerAddOnlineItem adds items for an online order.

```
CREATE PROC spCustomerAddOnlineItem
    @OrderID int,
    @ProductID int,
    @Quantity int
AS
-- DECLARE @CustomerID int = dbo.fnCurrentCustomerID();
IF NOT EXISTS (SELECT * FROM Order_.OnlineOrders WHERE OrderID = @OrderID)
    THROW 50001, 'Invalid OrderID!', 1;
-- IF (SELECT CustomerID FROM Order_.OnlineOrders WHERE OrderID = @OrderID)
<> @CustomerID
--     THROW 50008, 'Insufficient permissions!', 1;
IF NOT EXISTS (SELECT * FROM Product.Products WHERE ProductID = @ProductID)
    THROW 50007, 'Invalid ProductID!', 1;
IF (SELECT Locked FROM Order_.OnlineOrders WHERE OrderID = @OrderID) = 1
    THROW 50002, 'Invalid operation! This order is locked.', 1;
IF EXISTS (SELECT * FROM Order_.OnlineOrderToItem WHERE OrderID = @OrderID
AND ProductID = @ProductID)
    UPDATE Order_.OnlineOrderToItem
    SET Quantity = Quantity + @Quantity
    WHERE OrderID = @OrderID AND ProductID = @ProductID;
ELSE
BEGIN
    DECLARE @UnitPrice money = (SELECT UnitPrice FROM Product.Products WHERE
ProductID = @ProductID);
    INSERT INTO Order_.OnlineOrderToItem (OrderID, ProductID, Quantity,
UnitPrice)
    VALUES (@OrderID, @ProductID, @Quantity, @UnitPrice);
END;
GO
```

Stored procedure spCustomerDropOnlineItem drops items for an online order.

```
IF OBJECT_ID('spCustomerDropOnlineItem') IS NOT NULL
    DROP PROC spCustomerDropOnlineItem
GO
CREATE PROC spCustomerDropOnlineItem
    @OrderID int,
    @ProductID int,
    @Quantity int
AS
-- DECLARE @CustomerID int = dbo.fnCurrentCustomerID();
```

```
IF NOT EXISTS (SELECT * FROM Order_.OnlineOrders WHERE OrderID = @OrderID)
    THROW 50001, 'Invalid OrderID', 1;
-- IF (SELECT CustomerID FROM Order_.OnlineOrders WHERE OrderID = @OrderID)
<> @CustomerID
--     THROW 50008, 'Insufficient permissions!', 1;
IF NOT EXISTS (SELECT * FROM Product.Products WHERE ProductID = @ProductID)
    THROW 50007, 'Invalid ProductID!', 1;
IF (SELECT Locked FROM Order_.OnlineOrders WHERE OrderID = @OrderID) = 1
    THROW 50002, 'Invalid operation! This order is locked.', 1;
IF EXISTS (SELECT * FROM Order_.OnlineOrderToItem WHERE OrderID = @OrderID
AND ProductID = @ProductID)
BEGIN
    IF (SELECT Quantity FROM Order_.OnlineOrderToItem WHERE OrderID =
@OrderID AND ProductID = @ProductID) < @Quantity
        UPDATE Order_.OnlineOrderToItem
        SET Quantity = 0
        WHERE OrderID = @OrderID AND ProductID = @ProductID;
    ELSE
        UPDATE Order_.OnlineOrderToItem
        SET Quantity = Quantity - @Quantity
        WHERE OrderID = @OrderID AND ProductID = @ProductID;
END
```

Stored procedure spCustomerPlaceOnlineOrder places an online order and locks it.

```
CREATE PROC spCustomerPlaceOnlineOrder
    @OrderID int
AS
-- DECLARE @CustomerID int = dbo.fnCurrentCustomerID();
IF NOT EXISTS (SELECT * FROM Order_.OnlineOrders WHERE OrderID = @OrderID)
    THROW 50001, 'Invalid OrderID!', 1;
-- IF (SELECT CustomerID FROM Order_.OnlineOrders WHERE OrderID = @OrderID)
<> @CustomerID
--     THROW 50008, 'Insufficient permissions!', 1;
IF (SELECT Locked FROM Order_.OnlineOrders WHERE OrderID = @OrderID) = 1
    THROW 50002, 'Invalid operation! This order is locked.', 1;
UPDATE Order_.OnlineOrders
SET Locked = 1
WHERE OrderID = @OrderID;
UPDATE Order_.OnlineOrders
SET OrderDate = GETDATE()
WHERE OrderID = @OrderID;
UPDATE Order_.OnlineOrders
SET PayDueDate = DATEADD(day, 1, GETDATE())
```

```
WHERE OrderID = @OrderID;
GO
```

Stored procedure spCustomerWriteReview adds a review for a product. Then the score of this product will be updated by trigger.

```
CREATE PROC spCustomerWriteReview
   @CustomerID int,
   @ProductID int,
   @Text varchar(1000),
   @Score float
AS
-- DECLARE @CustomerID int = dbo.fnCurrentCustomerID();
IF NOT EXISTS (SELECT * FROM Product.Products WHERE ProductID = @ProductID)
   THROW 50007, 'Invalid ProductID!', 1;
INSERT INTO Product.Reviews (ProductID, CustomerID, Text, Score, ReviewDate)
VALUES (@ProductID, @CustomerID, @Text, @Score, GETDATE());
GO
```

2.6 Create Triggers

Trigger Addresses_INSERT_UPDATE uppers the letters of the address after it is inserted or updated.

```
CREATE TRIGGER Addresses_INSERT_UPDATE
   ON Address.Addresses
   AFTER INSERT, UPDATE
AS
UPDATE Address.Addresses
SET Street = UPPER(Street)
WHERE AddressID IN (SELECT AddressID FROM Inserted);
UPDATE Address.Addresses
SET City = UPPER(City)
WHERE AddressID IN (SELECT AddressID FROM Inserted);
UPDATE Address.Addresses
SET State = UPPER(State)
WHERE AddressID IN (SELECT AddressID FROM Inserted);
GO
```

Trigger OnlineOrderToItem_INSERT_UPDATE and OnlineOrderToItem_DELETE update the TotalPrice of the order after customer edit this order.

```
CREATE TRIGGER OnlineOrderToItem_INSERT_UPDATE
   ON Order_.OnlineOrderToItem
   AFTER INSERT, UPDATE
AS
```

```
UPDATE Order_.OnlineOrders
SET TotalPrice = (
    SELECT SUM(UnitPrice * Quantity)
    FROM Order_.OnlineOrderToItem
    WHERE Order_.OnlineOrderToItem.OrderID = Order_.OnlineOrders.OrderID
)
WHERE OrderID IN (SELECT OrderID FROM Inserted);
GO

CREATE TRIGGER OnlineOrderToItem_DELETE
    ON Order_.OnlineOrderToItem
    AFTER DELETE
AS
UPDATE Order_.OnlineOrders
SET TotalPrice = (
    SELECT SUM(UnitPrice * Quantity)
    FROM Order_.OnlineOrderToItem
    WHERE Order_.OnlineOrderToItem.OrderID = Order_.OnlineOrders.OrderID
)
WHERE OrderID IN (SELECT OrderID FROM Deleted);
GO
```

Trigger Reviews_INSEERT updates the AvgScore of the product after a customer writes a review for it.

```
IF OBJECT_ID('Reviews_INSERT') IS NOT NULL
    DROP TRIGGER Reviews_INSERT
GO
CREATE TRIGGER Reviews_INSERT
    ON Product.Reviews
    AFTER INSERT
AS
UPDATE Product.Products
SET AvgScore = (
    SELECT AVG(Score)
    FROM Product.Reviews
    WHERE Product.Reviews.ProductID = Product.Products.ProductID
)
WHERE ProductID IN (SELECT ProductID FROM Inserted);
GO
```

Trigger OnlineOrders_DELETE replaces DELETE when customer try to delete online orders. It deletes the orders that are not locked in Deleted. If there are some orders that are locked, it will throw an error.

```
CREATE TRIGGER OnlineOrders_DELETE
```

```
    ON Order_.OnlineOrders
    INSTEAD OF DELETE
AS
DELETE Order_.OnlineOrders
WHERE OrderID IN (SELECT OrderID FROM Deleted WHERE Locked = 0);
IF (SELECT COUNT(*) FROM deleted WHERE Locked = 1) > 0
    THROW 50002, 'Invalid operation! This order is locked.', 1;
GO
```

2.7 Write Transactions

The transaction in spScheduleShipping schedules shippings with status = Ready for an order. If it can't schedule all the items in this order, it will rollback transaction.

```
CREATE PROC spScheduleShipping
    @OrderID int
AS
DECLARE @UnavailableProductID int;
DECLARE @ErrorMessage varchar(100);
DECLARE @ShippingAddressID int;
DECLARE @WarehouseCount int;
DECLARE @CurrentWarehouseID int;
DECLARE @CurrentProductID int;
DECLARE @CurrentWarehouseHave int;
DECLARE @CurrentProductNeed int;
DECLARE @HasShipping bit;
DECLARE @CurrentShippingID int;
DECLARE @UnscheduledProducts TABLE
    (ID int,
    ProductID int,
    Quantity int);
DECLARE @WarehouseRank TABLE
    (Rank int,
    WarehouseID int);
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
IF NOT EXISTS (SELECT * FROM Order_.OnlineOrders WHERE OrderID = @OrderID)
    THROW 50001, 'Invalid OrderID!', 1;
INSERT @UnscheduledProducts
SELECT ROW_NUMBER() OVER(ORDER BY ProductID) AS ID, ProductID, Quantity
FROM Order_.OnlineOrderToItem
WHERE OrderID = @OrderID;
SET @ShippingAddressID = (SELECT AddressID FROM Order_.OnlineOrders WHERE
OrderID = @OrderID);
INSERT @WarehouseRank
```

```
SELECT *
FROM dbo.fnWarehouseRankByDistance(@ShippingAddressID);
SET @WarehouseCount = (SELECT COUNT(*) FROM @WarehouseRank);
BEGIN TRAN
    DECLARE @i int = 1;
    -- Schedule warehouses for items from near to far
    WHILE @i <= @WarehouseCount AND (SELECT SUM(Quantity) FROM
@UnscheduledProducts) > 0
    BEGIN
        SET @HasShipping = 0;
        SET @CurrentWarehouseID = (SELECT WarehouseID FROM @WarehouseRank
WHERE Rank = @i);
        DECLARE @j int = 1;
        WHILE @j <= (SELECT COUNT(*) FROM @UnscheduledProducts)
        BEGIN
            SET @CurrentProductID = (SELECT ProductID FROM
@UnscheduledProducts WHERE ID = @j);
            SET @CurrentWarehouseHave = (
                SELECT InStock
                FROM Warehouse.WarehouseProductList
                WHERE WarehouseID = @CurrentWarehouseID AND ProductID =
@CurrentProductID);
            SET @CurrentProductNeed = (
                SELECT Quantity
                FROM @UnscheduledProducts
                WHERE ProductID = @CurrentProductID);
            IF @CurrentProductNeed > 0 AND @CurrentWarehouseHave > 0
            BEGIN
                IF @HasShipping = 0
                BEGIN
                    DECLARE @StartAddressID int = (SELECT AddressID FROM
Warehouse.Warehouses WHERE WarehouseID = @CurrentWarehouseID);
                    DECLARE @ArriveAddressID int = (SELECT AddressID FROM
Order_.OnlineOrders WHERE OrderID = @OrderID);
                    INSERT INTO Shipping.Shippings (OrderID, Status,
StartAddressID, ArriveAddressID, StartDate, ShippingService, ShippingFare)
                    VALUES (@OrderID, 'Ready', @StartAddressID,
@ArriveAddressID, GETDATE(), 'USPS', dbo.fnDistance(@StartAddressID,
@ArriveAddressID) / 1000 + 5);
                    SET @HasShipping = 1;
                    SET @CurrentShippingID = @@IDENTITY;
                END
                IF @CurrentProductNeed > @CurrentWarehouseHave
                    SET @CurrentProductNeed = @CurrentWarehouseHave;
```

```
            UPDATE Warehouse.WarehouseProductList
            SET InStock = InStock - @CurrentProductNeed
            WHERE WarehouseID = @CurrentWarehouseID AND ProductID =
@CurrentProductID;
            UPDATE Warehouse.WarehouseProductList
            SET OnWay = OnWay + @CurrentProductNeed
            WHERE WarehouseID = @CurrentWarehouseID AND ProductID =
@CurrentProductID;
            UPDATE @UnscheduledProducts
            SET Quantity = Quantity - @CurrentProductNeed
            WHERE ID = @CurrentProductID;
            INSERT INTO Shipping.ShippingToItem (ShippingID, ProductID,
Quantity)
            VALUES (@CurrentShippingID, @CurrentProductID,
@CurrentProductNeed);
         END
         SET @j = @j + 1;
      END
      SET @i = @i + 1;
   END
   IF (SELECT SUM(Quantity) FROM @UnscheduledProducts) > 0
      GOTO PROBLEM;
COMMIT TRAN
PROBLEM:
   IF (SELECT SUM(Quantity) FROM @UnscheduledProducts) > 0
   BEGIN
      ROLLBACK TRAN;
      UPDATE Order_.OnlineOrders
      SET Locked = 0
      WHERE OrderID = @OrderID;
      SET @UnavailableProductID = (SELECT TOP 1 ProductID FROM
@UnscheduledProducts WHERE Quantity > 0);
      SET @ErrorMessage = 'Product ' + CONVERT(varchar(10),
@UnavailableProductID)  + ' is unavailable!';
      THROW 50004, @ErrorMessage, 1;
   END
GO
```

The transaction in spCustomerPayOnlineOrder inserts a row in OnlinePayment and sets all related shippings' status = Shipped. If customer pays it after the due date, it will rollback transaction.

```
CREATE PROC spCustomerPayOnlineOrder
   @OrderID int,
   @PaymentTotal money,
```

```
    @CardNumber varchar(20)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
-- DECLARE @CustomerID int = dbo.fnCurrentCustomerID();
IF NOT EXISTS (SELECT * FROM Order_.OnlineOrders WHERE OrderID = @OrderID)
    THROW 50001, 'Invalid OrderID!', 1;
-- IF (SELECT CustomerID FROM Order_.OnlineOrders WHERE OrderID = @OrderID)
<> @CustomerID
--     THROW 50008, 'Insufficient permissions!', 1;
IF (SELECT Locked FROM Order_.OnlineOrders WHERE OrderID = @OrderID) = 0
    THROW 50005, 'Invalid operation! Please place order before pay it.', 1;
BEGIN TRAN
    INSERT INTO Order_.OnlinePayment (OrderID, PaymentTotal, CardNumber,
PayDate)
    VALUES (@OrderID, @PaymentTotal, @CardNumber, GETDATE());
    IF (SELECT SUM(PaymentTotal) FROM Order_.OnlinePayment WHERE OrderID =
@OrderID) >=
        (SELECT TotalPrice FROM Order_.OnlineOrders WHERE OrderID = @OrderID)
    BEGIN
        UPDATE Shipping.Shippings
        SET Status = 'Shipped'
        WHERE OrderID = @OrderID;
    END
    IF (SELECT PayDueDate FROM Order_.OnlineOrders WHERE OrderID = @OrderID)
< GETDATE()
        GOTO PROBLEM1;
COMMIT TRAN
PROBLEM1:
    IF (SELECT PayDueDate FROM Order_.OnlineOrders WHERE OrderID = @OrderID)
< GETDATE()
    BEGIN
        THROW 50003, 'Order expired!', 1;
        ROLLBACK TRAN;
    END
GO
```

The transaction in spShippingDelivered sets shipping's status = Delivered and updates the OnWay column in WarehouseProductList for the products in this shipping.

```
CREATE PROC spShippingDelivered
    @ShippingID int
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
-- DECLARE @CustomerID int = dbo.fnCurrentCustomerID();
```

```
IF NOT EXISTS (SELECT * FROM Shipping.Shippings WHERE ShippingID =
@ShippingID)
    THROW 50006, 'Invalid ShippingID!', 1;
-- IF (SELECT CustomerID FROM Order_.OnlineOrders WHERE OrderID = (SELECT
OrderID FROM Shipping.Shippings WHERE ShippingID = @ShippingID)) <>
@CustomerID
--    THROW 50008, 'Insufficient permissions!', 1;
BEGIN TRAN
    UPDATE Shipping.Shippings
    SET Status = 'Delivered'
    WHERE ShippingID = @ShippingID;
    UPDATE Warehouse.WarehouseProductList
    SET OnWay = OnWay - (
        SELECT Quantity
        FROM Shipping.ShippingToItem
        WHERE ShippingID = @ShippingID AND ProductID =
Warehouse.WarehouseProductList.ProductID)
    WHERE WarehouseID = (
        SELECT WarehouseID
        FROM Warehouse.Warehouses
        WHERE AddressID = (
            SELECT StartAddressID
            FROM Shipping.Shippings
            WHERE ShippingID = @ShippingID))
        AND ProductID IN (
            SELECT ProductID
            From Shipping.ShippingToItem
            WHERE ShippingID = @ShippingID);
COMMIT TRAN
GO
```

The transaction in spShippingReturned sets shipping's status = Returned and updates the InReturn column in WarehouseProductList for the products in this shipping. If Intact = 1, it updates the InStock column in WarehouseProductList and refunds money to customer.

```
CREATE PROC spShippingReturned
    @ShippingID int,
    @Intact bit
AS
DECLARE @WarehouseID int = (SELECT WarehouseID FROM Warehouse.Warehouses
WHERE AddressID = (
    SELECT StartAddressID
    FROM Shipping.Shippings
    WHERE ShippingID = @ShippingID));
```

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
IF NOT EXISTS (SELECT * FROM Shipping.Shippings WHERE ShippingID =
@ShippingID)
    THROW 50006, 'Invalid ShippingID!', 1;
BEGIN TRAN
    UPDATE Shipping.Shippings
    SET Status = 'Returned'
    WHERE ShippingID = @ShippingID;
    UPDATE Warehouse.WarehouseProductList
    SET InReturn = InReturn - (
        SELECT Quantity
        FROM Shipping.ShippingToItem
        WHERE ShippingID = @ShippingID AND ProductID =
Warehouse.WarehouseProductList.ProductID)
    WHERE WarehouseID = @WarehouseID AND ProductID IN (
        SELECT ProductID
        FROM Shipping.ShippingToItem
        WHERE ShippingID = @ShippingID);
    IF @Intact = 1
    BEGIN
        UPDATE Warehouse.WarehouseProductList
        SET InStock = InStock + (
            SELECT Quantity
            FROM Shipping.ShippingToItem
            WHERE ShippingID = @ShippingID AND ProductID =
Warehouse.WarehouseProductList.ProductID)
        WHERE WarehouseID = @WarehouseID AND ProductID IN (
            SELECT ProductID
            FROM Shipping.ShippingToItem
            WHERE ShippingID = @ShippingID);
        DECLARE @OrderID int = (SELECT OrderID FROM Shipping.Shippings WHERE
ShippingID = @ShippingID);
        DECLARE @TotalPrice money = (
              SELECT SUM(Quantity * UnitPrice)
              FROM Shipping.ShippingToItem JOIN Product.Products
                  ON Shipping.ShippingToItem.ProductID =
Product.Products.ProductID
              WHERE ShippingID = @ShippingID);
        INSERT INTO Order_.OnlinePayment (OrderID, PaymentTotal, CardNumber,
PayDate)
        VALUES (@OrderID, -@TotalPrice, (SELECT TOP 1 CardNumber FROM
Order_.OnlinePayment WHERE OrderID = @OrderID), GETDATE());
    END
COMMIT TRAN
```

```
GO
```

2.8 Write Scripts (Security Related)

Create database roles Visitor, Customer, Store, Warehouse, Supplier, TarbetAdministrator and grant their permissions:

```
USE Tarbet
GO

CREATE ROLE Visitor;
GRANT SELECT ON Product.Products TO Visitor;
GRANT SELECT ON Product.Reviews TO Visitor;
GRANT SELECT ON Store.Stores TO Visitor;
GRANT SELECT ON Warehouse.Warehouses TO Visitor;
GRANT SELECT ON Supplier.Suppliers TO Visitor;
GRANT SELECT ON OnlineAvailableProducts TO Visitor;
GRANT SELECT ON ProductSales TO Visitor;
GRANT SELECT ON ProductSalesVolume TO Visitor;
GRANT SELECT ON ProductStars TO Visitor;
GO

CREATE ROLE Customer;
ALTER ROLE Visitor ADD MEMBER Customer;
-- GRANT EXEC ON spCustomerCreateAccount TO Customer;
-- GRANT EXEC ON spCustomerShowBillingAddress TO Customer;
GRANT EXEC ON spCustomerAddBillingAddress TO Customer;
-- GRANT EXEC ON spCustomerDropBillingAddress TO Customer;
-- GRANT EXEC ON spCustomerShowShippingAddress TO Customer;
GRANT EXEC ON spCustomerAddShippingAddress TO Customer;
-- GRANT EXEC ON spCustomerDropShippingAddress TO Customer;
-- GRANT EXEC ON spCustomerShowCardList TO Customer;
-- GRANT EXEC ON spCustomerAddCreditCard TO Customer;
-- GRANT EXEC ON spCustomerDropCreditCard TO Customer;
GRANT EXEC ON spCustomerCreateOnlineOrder TO Customer;
-- GRANT EXEC ON spCustomerShowOnlineOrderList TO Customer;
-- GRANT EXEC ON spCustomerShowOnlineOrderList TO Customer;
GRANT EXEC ON spCustomerAddOnlineItem TO Customer;
GRANT EXEC ON spCustomerDropOnlineItem TO Customer;
GRANT EXEC ON spCustomerPlaceOnlineOrder TO Customer;
GRANT EXEC ON spCustomerPayOnlineOrder TO Customer;
GRANT EXEC ON spCustomerWriteReview TO Customer;
GRANT EXEC ON spCustomerCreateInStoreOrder TO Customer;
-- GRANT EXEC ON spCustomerShowInStoreOrderList TO Customer;
```

```
GRANT EXEC ON spCustomerAddInStoreItem TO Customer;
GRANT EXEC ON spCustomerDropInStoreItem TO Customer;
-- GRANT EXEC ON spCustomerPayInStoreOrder TO Customer;
-- GRANT EXEC ON spCustomerReturnProduct TO Customer;
GRANT EXEC ON spCustomerReturnShipping TO Customer;
GO

CREATE ROLE Store;
ALTER ROLE Visitor ADD MEMBER Store;
GRANT EXEC ON spStoreCreateOrder TO Store;
GRANT EXEC ON spStoreAddItemInOrder TO Store;
GRANT EXEC ON spStoreDropItemInOrder TO Store;
GRANT EXEC ON spStorePlaceOrder TO Store;
-- GRANT EXEC ON spStoreShowOrderList TO Store;
-- GRANT EXEC ON spStoreShowProductList TO Store;
GO

CREATE ROLE Warehouse;
ALTER ROLE Visitor ADD MEMBER Warehouse;
GRANT EXEC ON spWarehouseCreateOrder TO Warehouse;
-- GRANT EXEC ON spWarehouseAddItemInOrder TO Warehouse;
-- GRANT EXEC ON spWarehouseDropItemInOrder TO Warehouse;
-- GRANT EXEC ON spWarehousePlaceOrder TO Warehouse;
-- GRANT EXEC ON spWarehouseShowOrderList TO Warehouse;
-- GRANT EXEC ON spWarehouseShowProductList TO Warehouse;
GO

CREATE ROLE Supplier;
ALTER ROLE Visitor ADD MEMBER Supplier;
-- GRANT EXEC ON spSupplierCreateProduct TO Supplier;
GRANT EXEC ON spSupplierRestockProduct TO Supplier;
-- GRANT EXEC ON spSupplierShowProductList TO Supplier;
GO

CREATE ROLE TarbetAdministrator;
ALTER ROLE db_datareader ADD MEMBER TarbetAdministrator;
ALTER ROLE db_datawriter ADD MEMBER TarbetAdministrator;
GRANT EXEC ON spScheduleShipping TO TarbetAdministrator;
GRANT EXEC ON spShippingDelivered TO TarbetAdministrator;
GRANT EXEC ON spShippingReturned TO TarbetAdministrator;
GRANT EXEC ON spStoreOrderCompleted TO TarbetAdministrator;
-- GRANT EXEC ON spWarehouseOrderCompleted TO TarbetAdministrator;
-- GRANT EXEC ON spAlterProductPrice TO TarbetAdministrator;
GO
```

Create five logins:

```
USE master
GO
CREATE LOGIN JamesTaylor WITH PASSWORD = 'TarbetCustomer#1',
   DEFAULT_DATABASE = Tarbet;
CREATE LOGIN Store1 WITH PASSWORD = 'TarbetStore#1',
   DEFAULT_DATABASE = Tarbet;
CREATE LOGIN Warehouse1 WITH PASSWORD = 'TarbetWarehouse#1',
   DEFAULT_DATABASE = Tarbet;
CREATE LOGIN Starbucks WITH PASSWORD = 'TarbetSupplier#1',
   DEFAULT_DATABASE = Tarbet;
CREATE LOGIN Administrator1 WITH PASSWORD = 'TarbetAdministrator#1',
   DEFAULT_DATABASE = Tarbet;
GO
```

Create five users and assign their roles:

```
USE Tarbet
GO
CREATE USER JamesTaylor FOR LOGIN JamesTaylor;
ALTER ROLE Customer ADD MEMBER JamesTaylor;
CREATE USER Store1 FOR LOGIN Store1;
ALTER ROLE Store ADD MEMBER Store1;
CREATE USER Warehouse1 FOR LOGIN Warehouse1;
ALTER ROLE Warehouse ADD MEMBER Warehouse1;
CREATE USER Starbucks FOR LOGIN Starbucks;
ALTER ROLE Supplier ADD MEMBER Starbucks;
CREATE USER Administrator1 FOR LOGIN Administrator1;
ALTER ROLE TarbetAdministrator ADD MEMBER Administrator1;
GO
```

Show database roles and their permissions and members.

```
USE Tarbet
GO

/** Logins and corresponding Usernames **/
SELECT l.name AS Logins, u.name AS UserNames
FROM sys.syslogins AS l JOIN sys.sysusers AS u
   ON l.sid = u.sid;

/** User-defined database roles and their permissions **/
SELECT Princ.name, Princ.type_desc, Perm.permission_name, Perm.state_desc,
Perm.class_desc, OBJECT_NAME(Perm.major_id)
```

```
FROM sys.database_principals AS Princ LEFT JOIN sys.database_permissions AS
Perm
       ON Perm.grantee_principal_id = Princ.principal_id
WHERE Princ.name IN ('Visitor', 'Customer', 'Store', 'Warehouse',
'Supplier', 'TarbetAdministrator');

/** Database roles and their members **/
SELECT Princ1.name AS DatabaseRole, Princ2.name AS DatabaseRoleMember
FROM sys.database_role_members AS RoleMembers
   RIGHT OUTER JOIN sys.database_principals AS Princ1
       ON RoleMembers.role_principal_id = Princ1.principal_id
   LEFT OUTER JOIN sys.database_principals AS Princ2
       ON RoleMembers.member_principal_id = Princ2.principal_id
WHERE Princ1.type = 'R' AND Princ1.name IN ('Visitor', 'Customer', 'Store',
'Warehouse', 'Supplier', 'TarbetAdministrator', 'db_datareader',
'db_datawriter', 'db_owner');
```

2.9 Potential Integrity and Security Issues

To keep the integrity of the database. I set REFERENCE constraints between related table, some of them are ON DELETE CASCADE. For example, when an order is deleted, it will trigger a trigger. If it is unlocked (the customer hasn't placed it), all the related items will be deleted too. Otherwise, it will cause an error. To avoid security issues, users cannot edit tables directly. They can only edit their own data by executing stored procedures. The stored procedure will check their permissions by the fnCurrentCustomerID / fnCurrentStoreID / fnCurrentWarehouseID / fnCurrentSupplierID function, which can return the customerID / StoreID / WarehouseID / SupplierID based on the CURRENT_USER.

3. Testing

3.1 Initialize Data

Initialize ZipList table based on the data from http://www.census.gov/geo/maps-data/data/gazetteer.html.

```
INSERT INTO Address.ZipList (Zip, Latitude, Longitude) VALUES
   ('00601', 18.180555, -66.749961),
   ('00602', 18.361945, -67.175597),
   …… Omit 33140 rows
   ('99927', 56.239062,-133.457924),
   ('99929', 56.370751,-131.693301);
```

Initialize Address table as:

```
SET IDENTITY_INSERT Address.Addresses ON
INSERT INTO Address.Addresses (AddressID, Street, City, State, Zip) VALUES
   (1, '302 Marshall St', 'Syracuse', 'NY', '13210'),
   (2, '120 L St', 'Boston', 'MA', '02127'),
   (3, '1501 NW 56th St', 'Seattle', 'WA', '98107'),
   (4, '4375 Payne Ave', 'San Jose', 'CA', '95117'),
   (5, '8770 Dell Center Dr', 'Liverpool', 'NY', '13090'),
   (6, '5399 W Genesee St', 'Camillus', 'NY', '13031'),
   (7, '6438 Basile Rowe', 'Syracuse', 'NY', '13057'),
   (8, '12620 SE 41st Pl', 'Bellevue', 'WA', '98006'),
   (9, '3816 Smith Ave', 'Everett', 'WA', '98201'),
   (10, '1425 N Rabe Ave UNIT 102', 'Fresno', 'CA', '93727'),
   (11, '320 4th Ave N', 'Franklin', 'TN', '37064'),
   (12, '2474 State Rte 21', 'Canandaigua', 'NY', '14424'),
   (13, '1 Coca Cola Pl SE', 'Atlanta', 'GA', '30313'),
   (14, '2401 Utah Ave S', 'Seattle', 'WA', '98134'),
   (15, 'One Apple Park Way', 'Cupertino', 'CA', '95014'),
   (16, '1 Dell Way', 'Round Rock', 'TX', '78681');
SET IDENTITY_INSERT Address.Addresses OFF
```

Initialize Customers table as:

```
SET IDENTITY_INSERT Customer.Customers ON
INSERT INTO Customer.Customers (CustomerID, FirstName, LastName, UserName)
VALUES
   (1, 'James', 'Taylor', 'JamesTaylor'),
   (2, 'Robert', 'Evans', 'RobertEvans'),
   (3, 'John', 'Smith', 'JohnSmith'),
   (4, 'William', 'Davies', 'WilliamDavies');
SET IDENTITY_INSERT Customer.Customers OFF
```

Initialize Stores table as:

```
SET IDENTITY_INSERT Store.Stores ON
INSERT INTO Store.Stores (StoreID, UserName, AddressID) VALUES
   (1, 'Store1', 5),
   (2, 'Store2', 6),
   (3, 'Store3', 7),
   (4, 'Store4', 8);
SET IDENTITY_INSERT Store.Stores OFF
```

Initialize Warehouses table as:

```
SET IDENTITY_INSERT Warehouse.Warehouses ON
INSERT INTO Warehouse.Warehouses (WarehouseID, UserName, AddressID) VALUES
   (1, 'Warehouse1', 9),
   (2, 'Warehouse2', 10),
   (3, 'Warehouse3', 11),
   (4, 'Warehouse4', 12);
SET IDENTITY_INSERT Warehouse.Warehouses OFF
```

Initialize Suppliers table as:

```
SET IDENTITY_INSERT Supplier.Suppliers ON
INSERT INTO Supplier.Suppliers (SupplierID, SupplierName, UserName,
AddressID) VALUES
   (1, 'Coco-Cola Company', 'Coco-Cola', 13),
   (2, 'Starbucks Coffee Company', 'Starbucks', 14),
   (3, 'Apple Inc.', 'Apple', 15),
   (4, 'Dell Inc.', 'Dell', 16);
SET IDENTITY_INSERT Supplier.Suppliers OFF
```

Initialize Products table as:

```
SET IDENTITY_INSERT Product.Products ON
INSERT INTO Product.Products (ProductID, ProductName, UnitPrice) VALUES
   (1, 'Coco-Cola Coke 18 pack', 34.99),
   (2, 'Starbucks Frappuccino Coffee Drink', 2.99),
   (3, 'Apple iPhone 13 (256GB, Starlight)', 929.00),
   (4, 'Dell Inspiron 24 5400', 979.99);
SET IDENTITY_INSERT Product.Products OFF
```

Initialize ProductDetail table as:

```
INSERT INTO Supplier.ProductDetail (ProductID, SupplierID, AvailableNumber)
VALUES
   (1, 1, 10000),
   (2, 2, 50000),
   (3, 3, 3000),
   (4, 4, 1000);
```

Initialize StoreProductList table as:

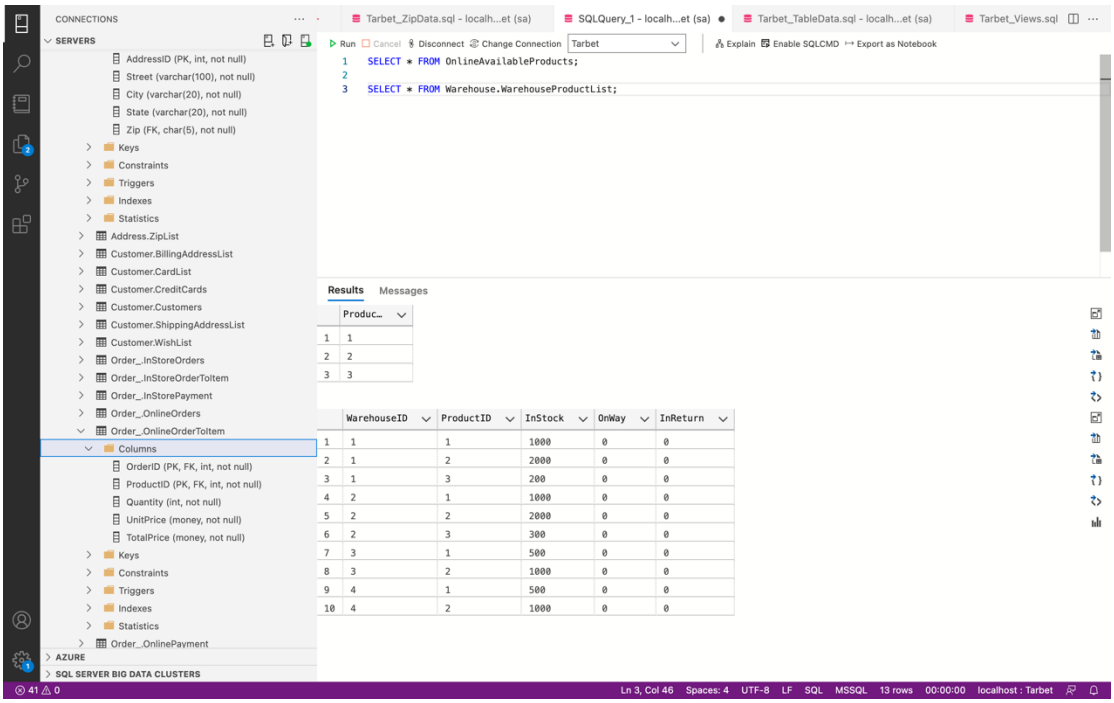
```
INSERT INTO Store.StoreProductList (ProductID, StoreID, Price, Quantity)
VALUES
   (1, 1, 34.99, 100),
   (1, 2, 2.99, 200),
   (1, 3, 929.00, 10),
   (1, 4, 979.99, 5);
GO
```

Initialize WarehouseProductList table as:

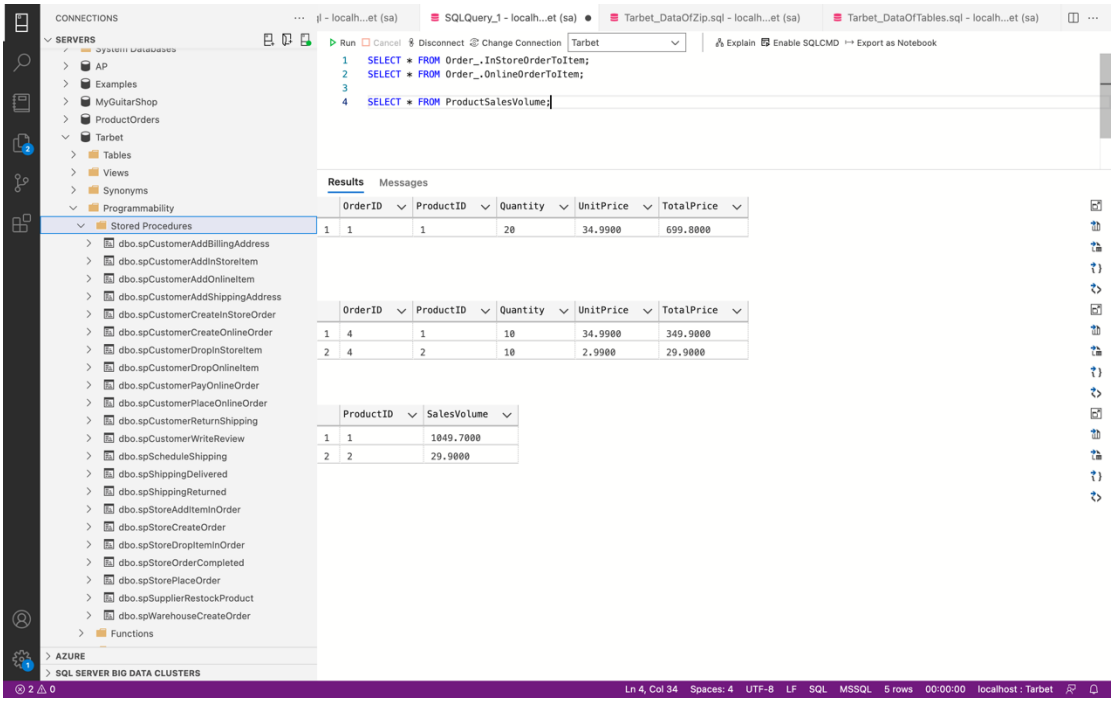
```
INSERT INTO Warehouse.WarehouseProductList (WarehouseID, ProductID, InStock)
VALUES
   (1, 1, 1000),
   (1, 2, 2000),
   (1, 3, 200),
   (2, 1, 1000),
   (2, 2, 2000),
   (2, 3, 300),
   (3, 1, 500),
   (3, 2, 1000),
   (4, 1, 500),
   (4, 2, 1000);
```

3.2 Test Views

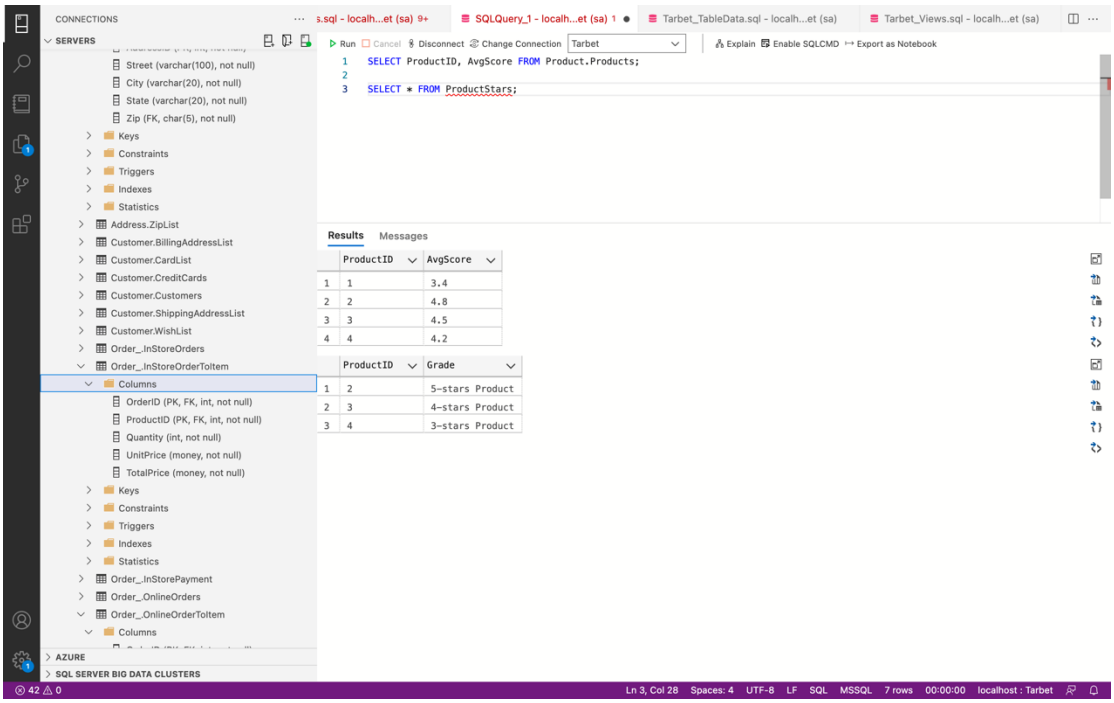
Test View OnlineAvailableProducts: Product 1, Product 2, and Product 3 are available, so this view is correct.



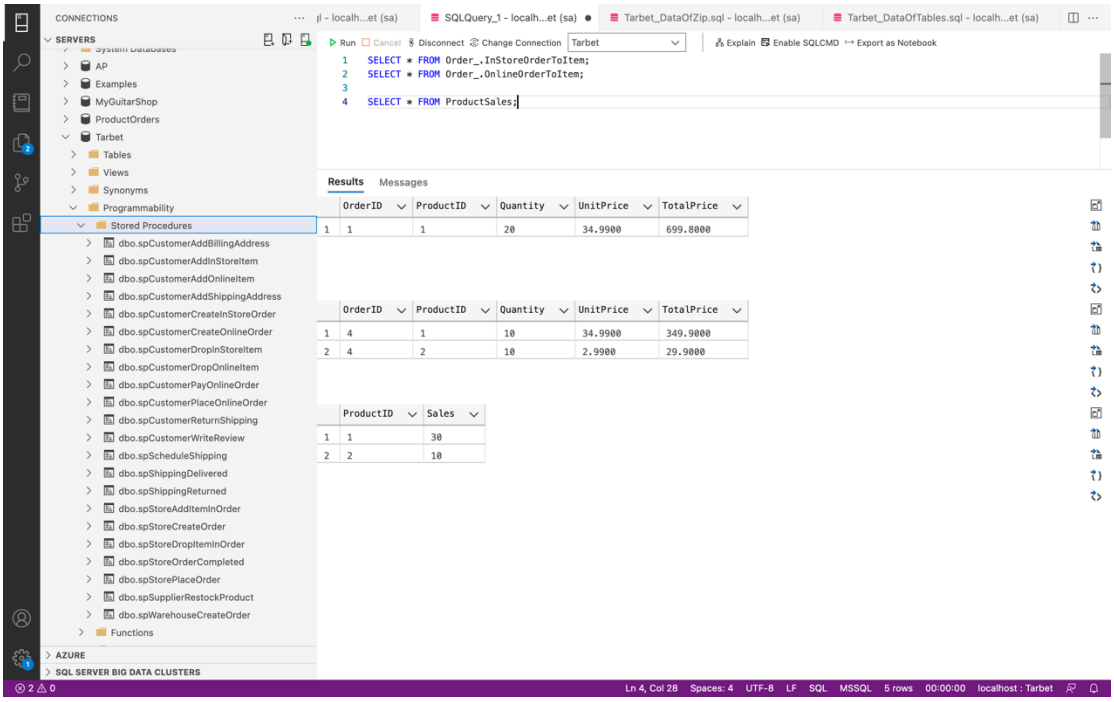
Test View ProductSalesVolume: the result shows that this view calculates the sum of the total online sales volume and the total in-store sales volume of each product correctly.



Test View ProductStars: the result shows that this view is correct.

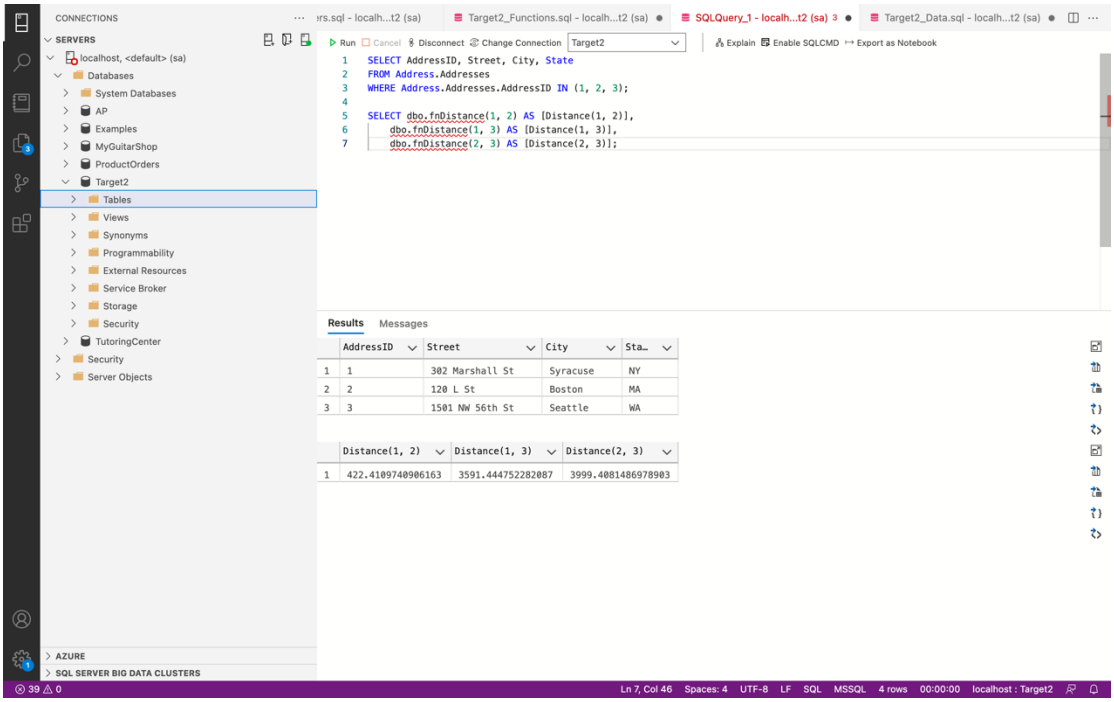


Test View ProductSales: the result shows that this view calculates the sum of the total online sales and the total in-store sales of each product correctly.

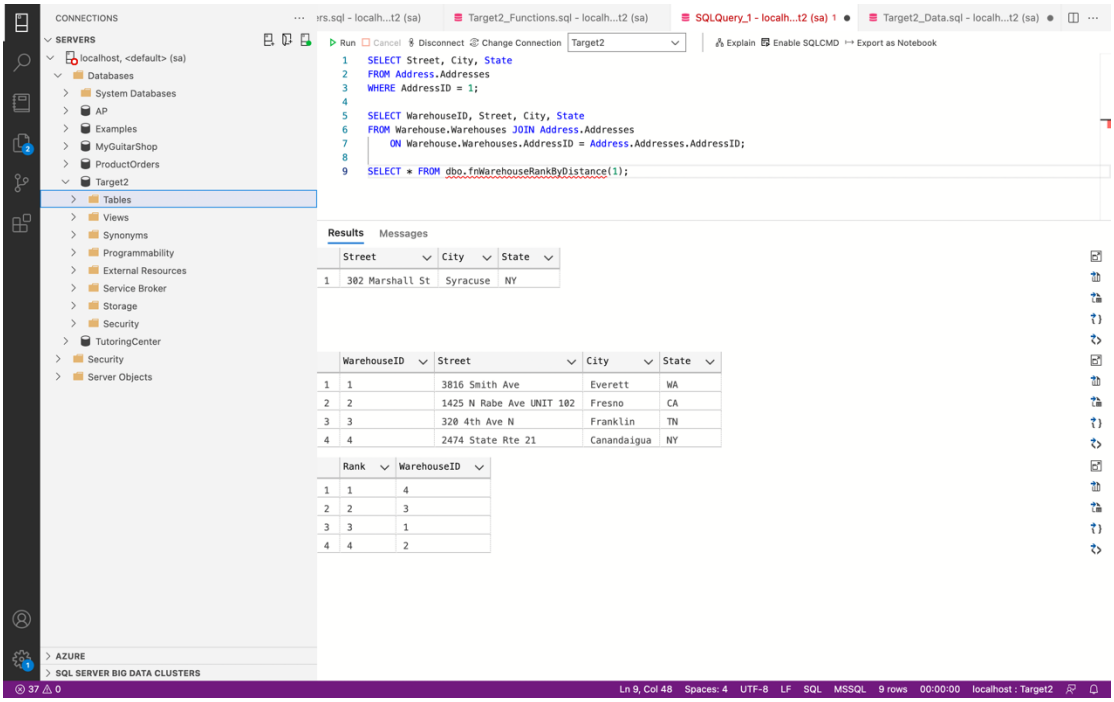


3.3 Test Functions

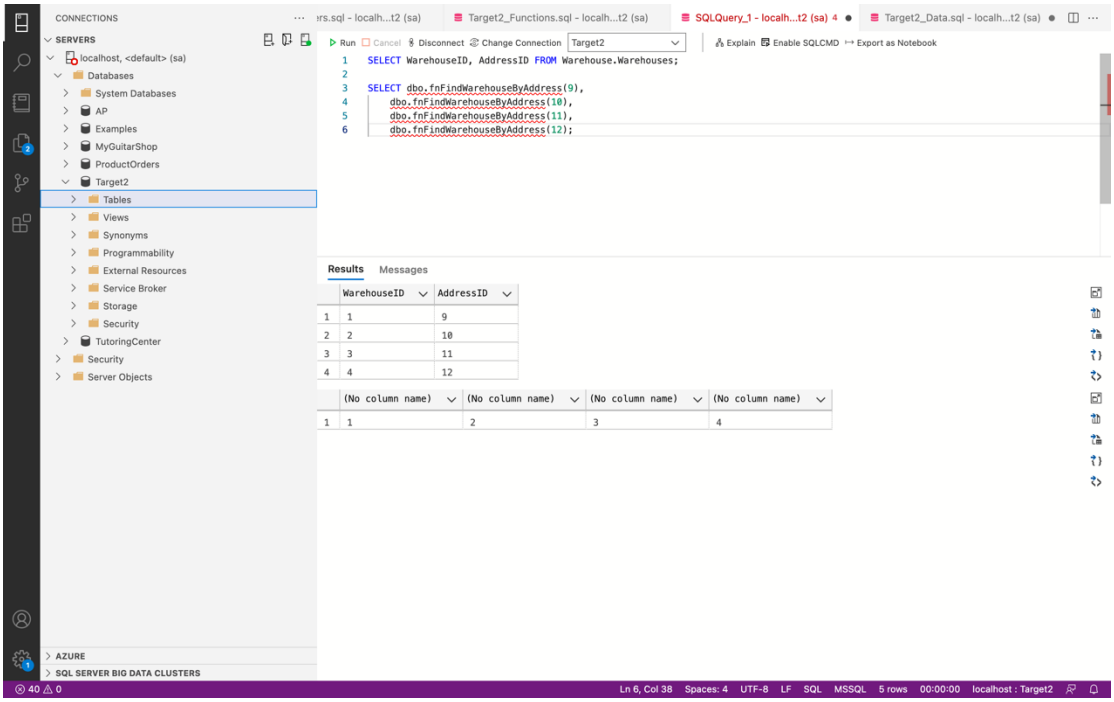
Test fnDistance function by calculating the distance between Address 1, Address 2, and Address 3. The result from this function is equal to the result from Google Earth. It shows that this function is correct.



Test fnWarehouseRankByDistance function by selecting the warehouse from near to far. The nearest warehouse to Address 1 is Warehouse 4, and the furthest warehouse to Address 1 is Warehouse 2. The result shows that this function is correct.



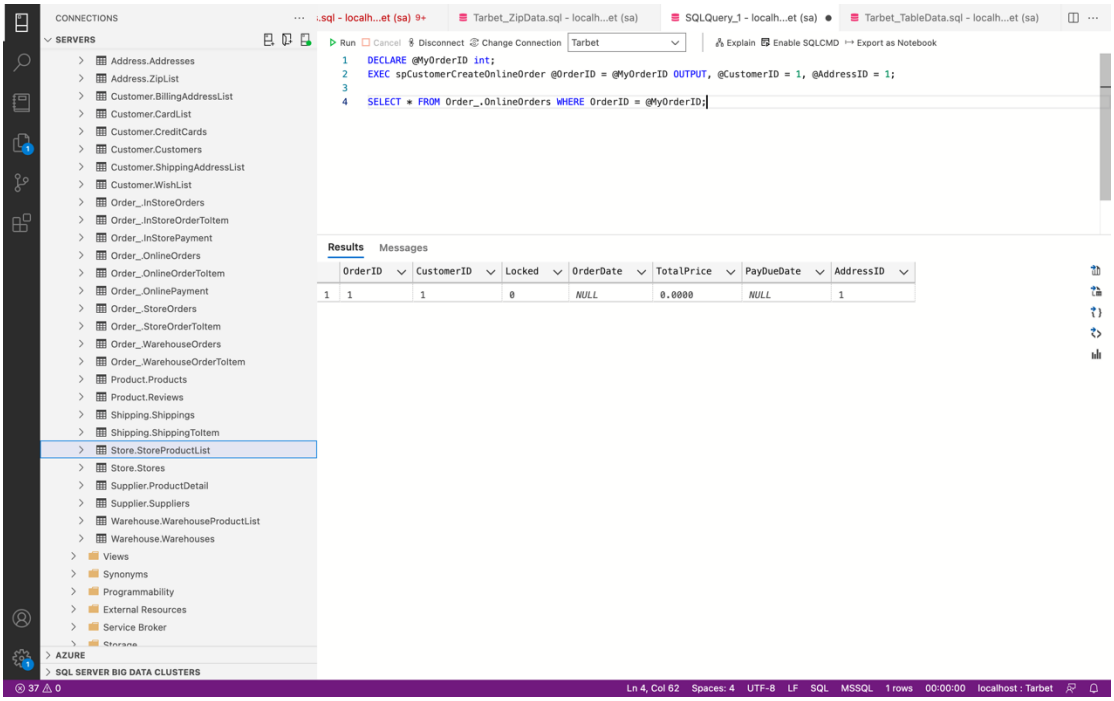
Test fnFindWarehouseID function by finding the WarehouseID of several AddressID. Warehouse 1 is located at Address 9, Warehouse 2 is located at Address 10, Warehouse 3 is located at Address 11, Warehouse 4 is located at Address 12. The result shows that this function is correct.



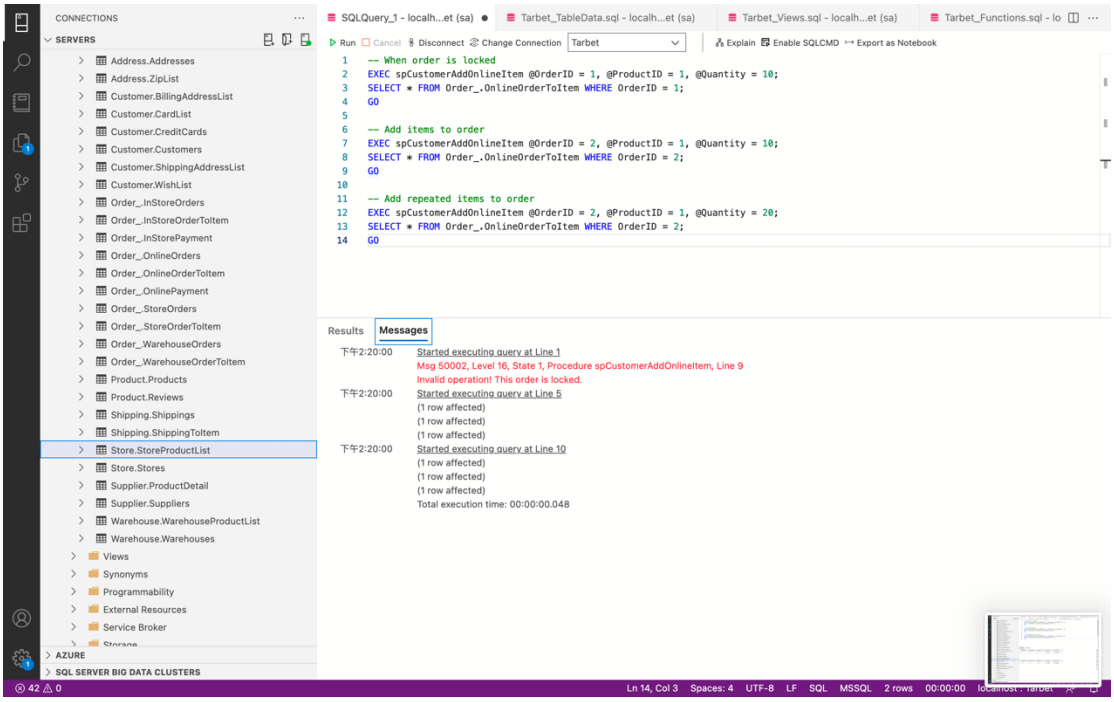
Test fnCheckAvailability function by find the unavailable items in Order 1. The result shows that this function is correct.

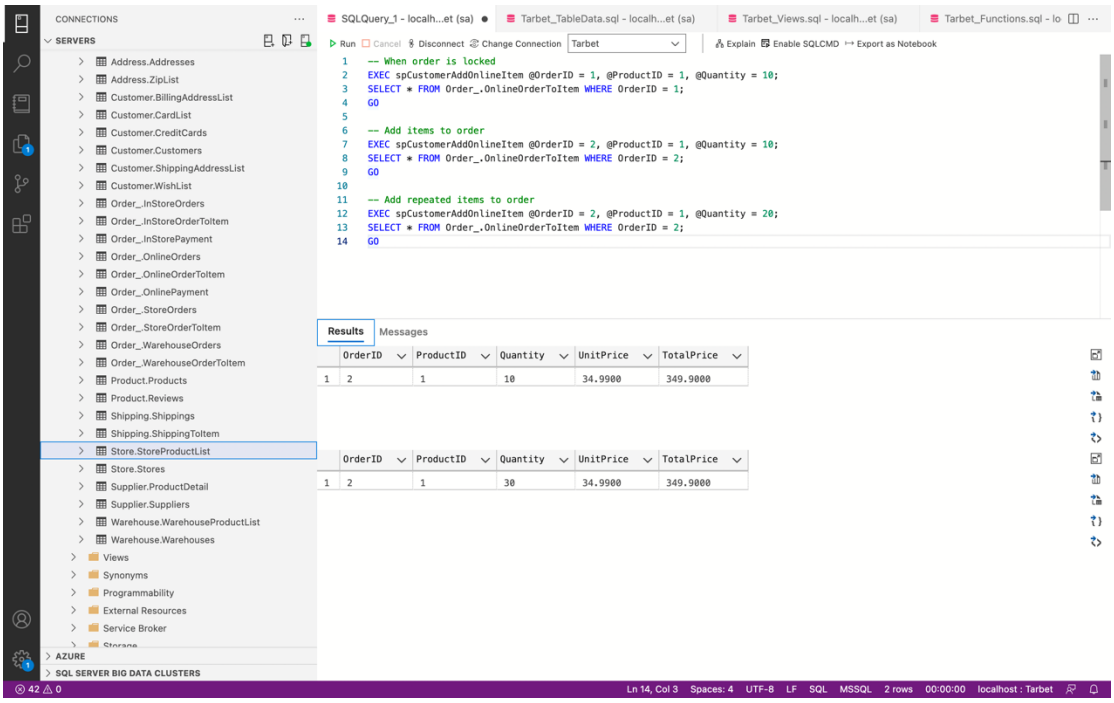
3.4 Test Stored Procedures

Test Stored procedure spCustomerCreateOnlineOrder by creating an order and testing its existence. The result shows that this stored procedure is correct.

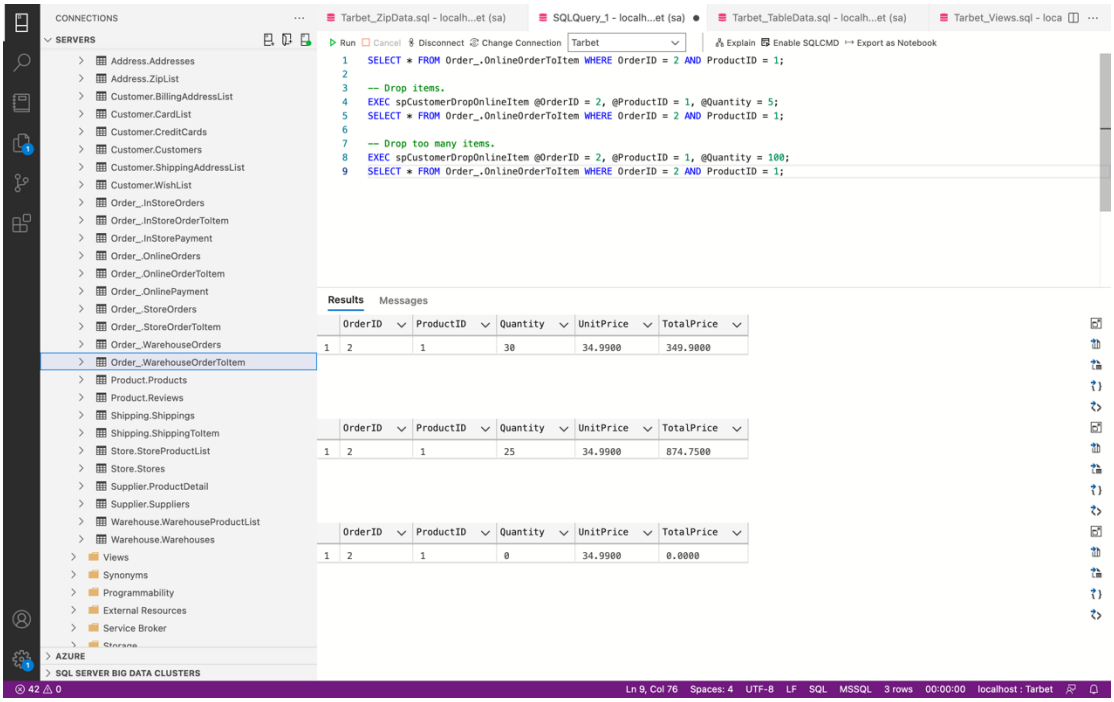


Test stored procedure spCustomerAddOnlineItem by adding items for a locked order and an unlocked order. The result shows that this stored procedure is correct.

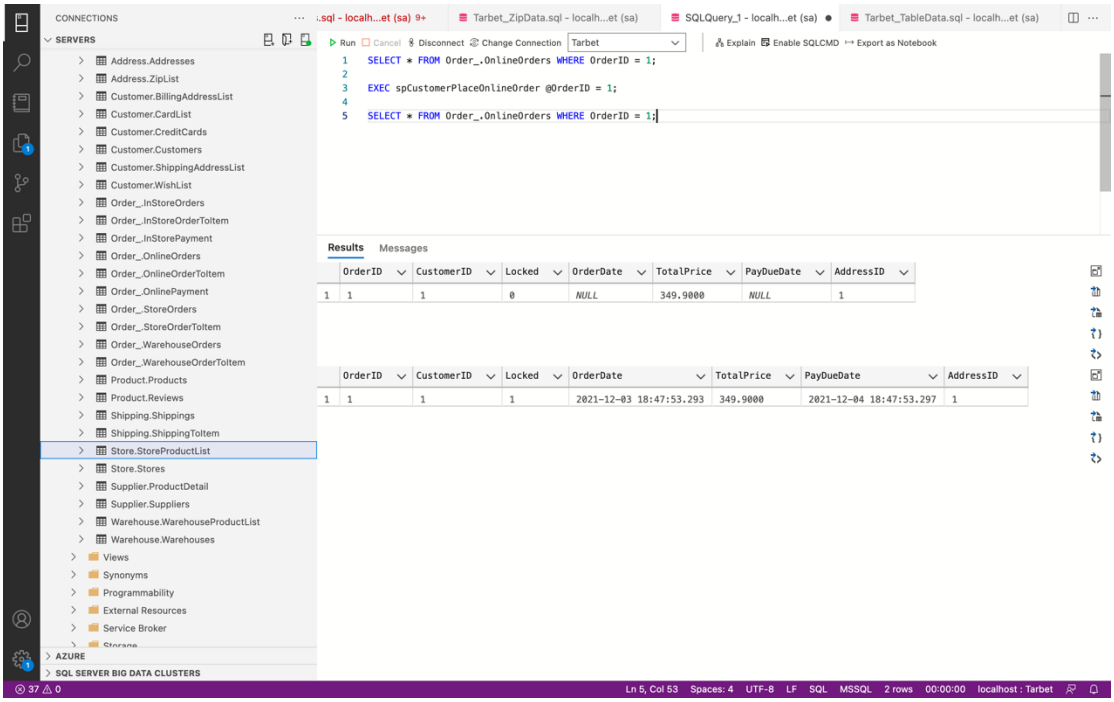


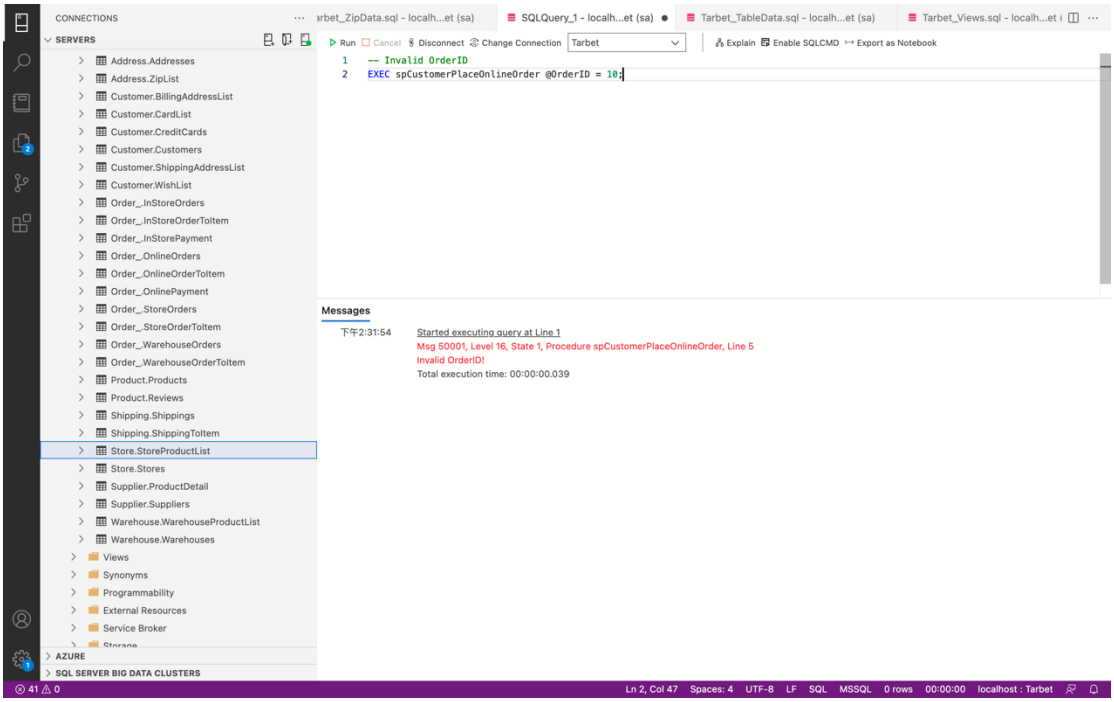


Test stored procedure spCustomerDropOnlineItem by dropping items and dropping too many items. The result shows that this stored procedure is correct.

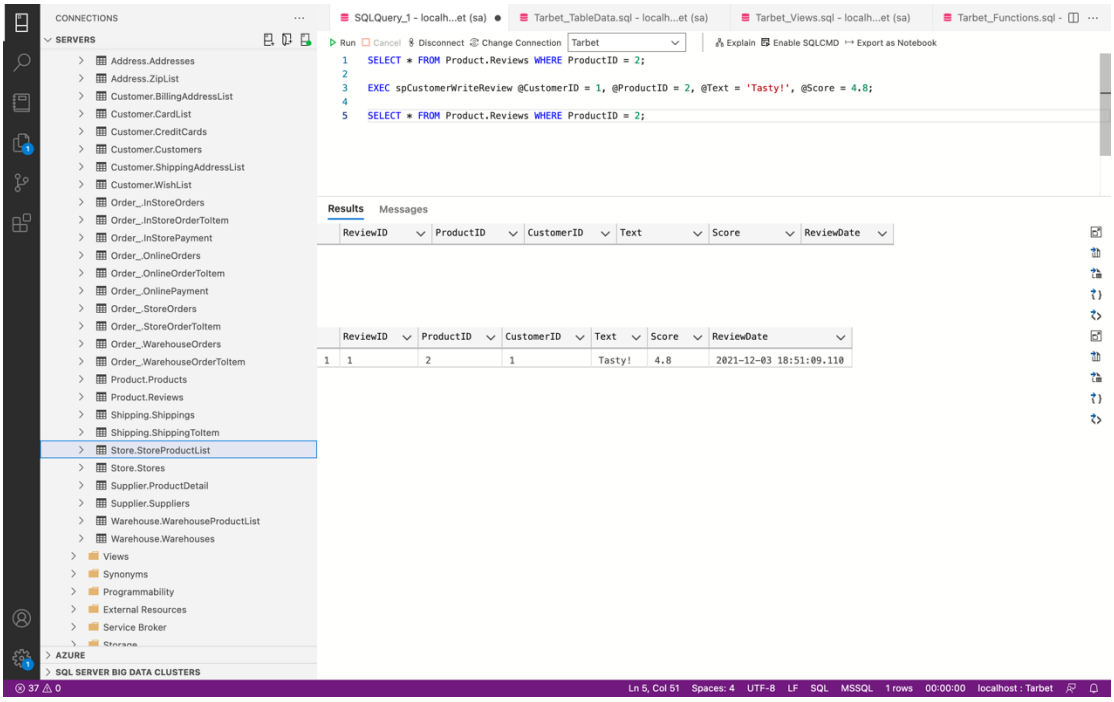


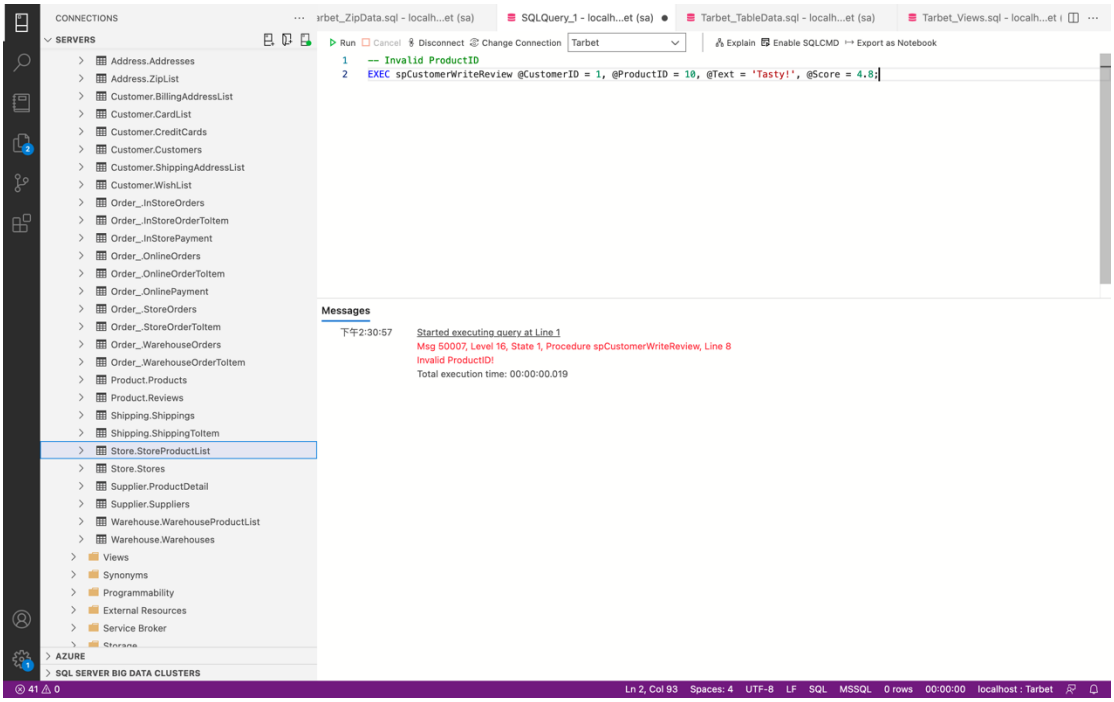
Test stored procedure spCustomerPlaceOnlineOrder by placing an online order and checking its Locked, OrderDate and PayDueDate column. The result shows that Locked, OrderDate and PayDueDate have been set to correct value.





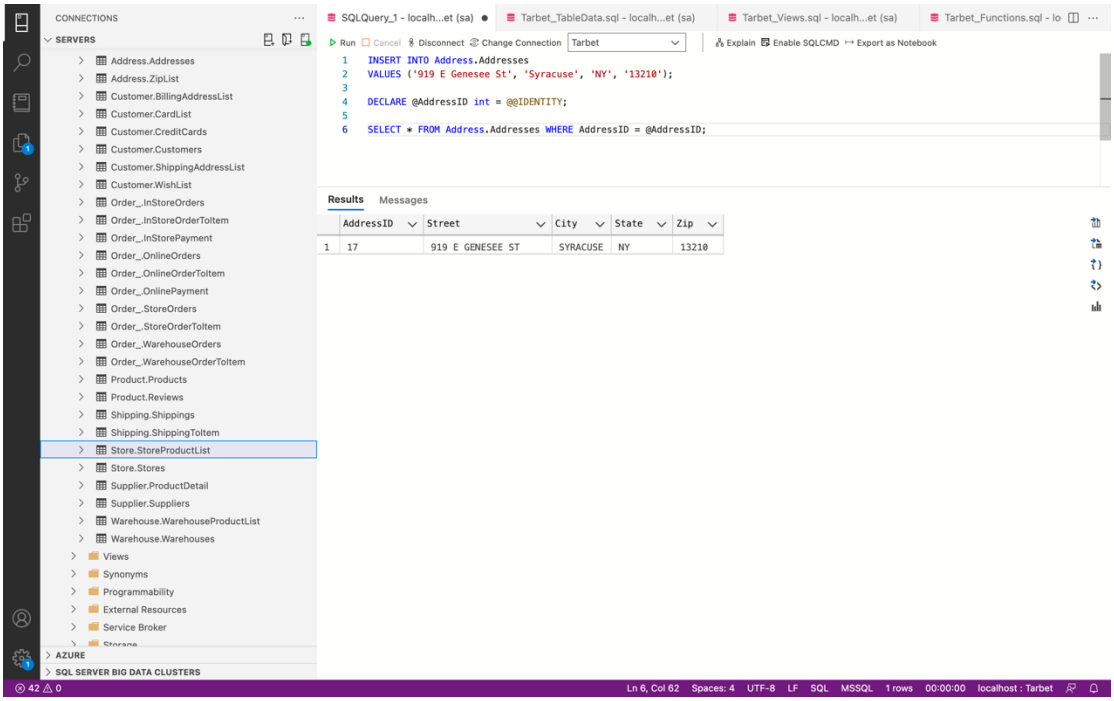
Test stored procedure spCustomerWriteReview by adding a review for a product. The result shows that this stored procedure is correct.



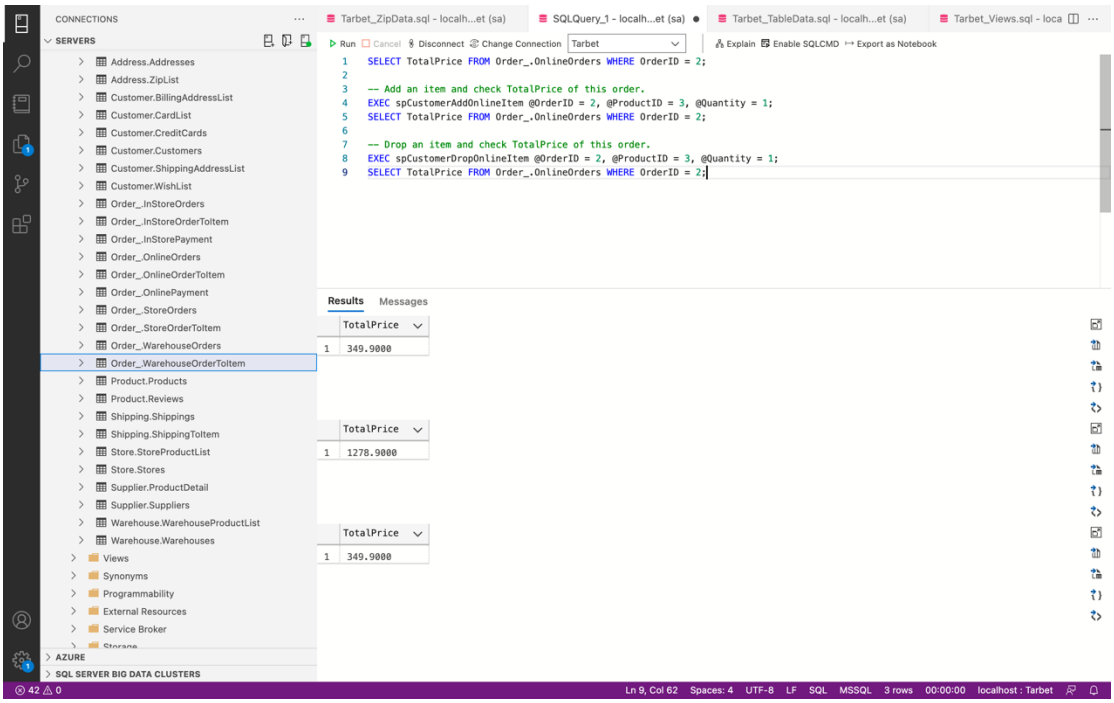


3.5 Test Triggers

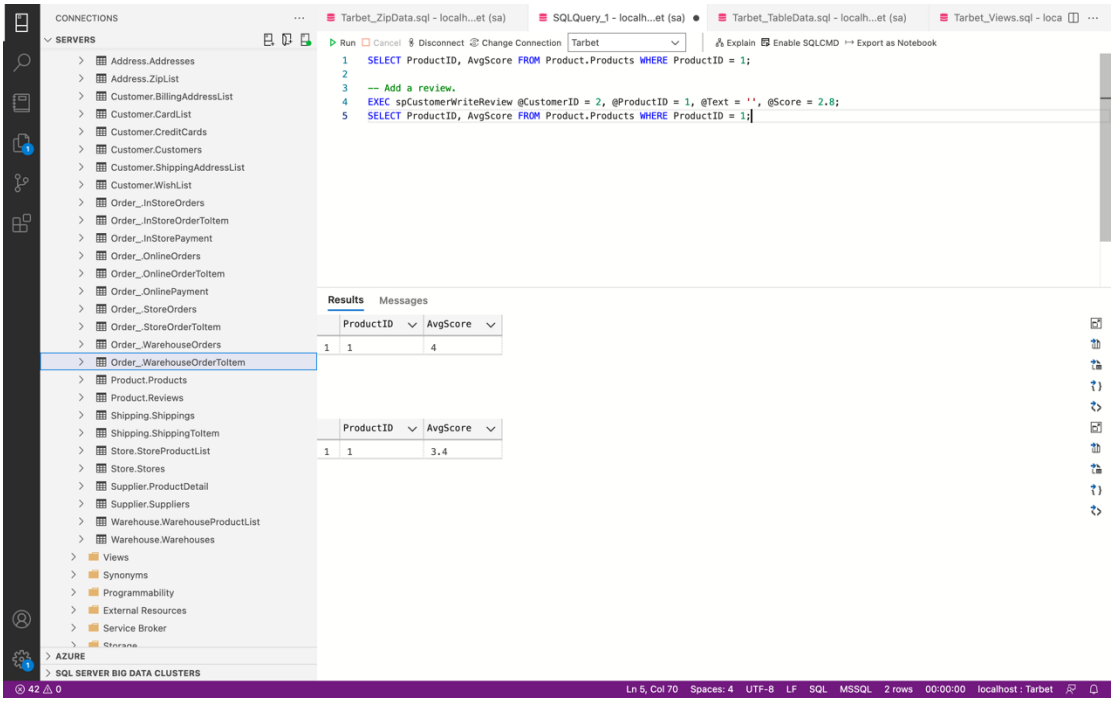
Test trigger Addresses_INSERT_UPDATE by inserting an address that contains lower case letters. It shows that this address is turned into upper case letters.



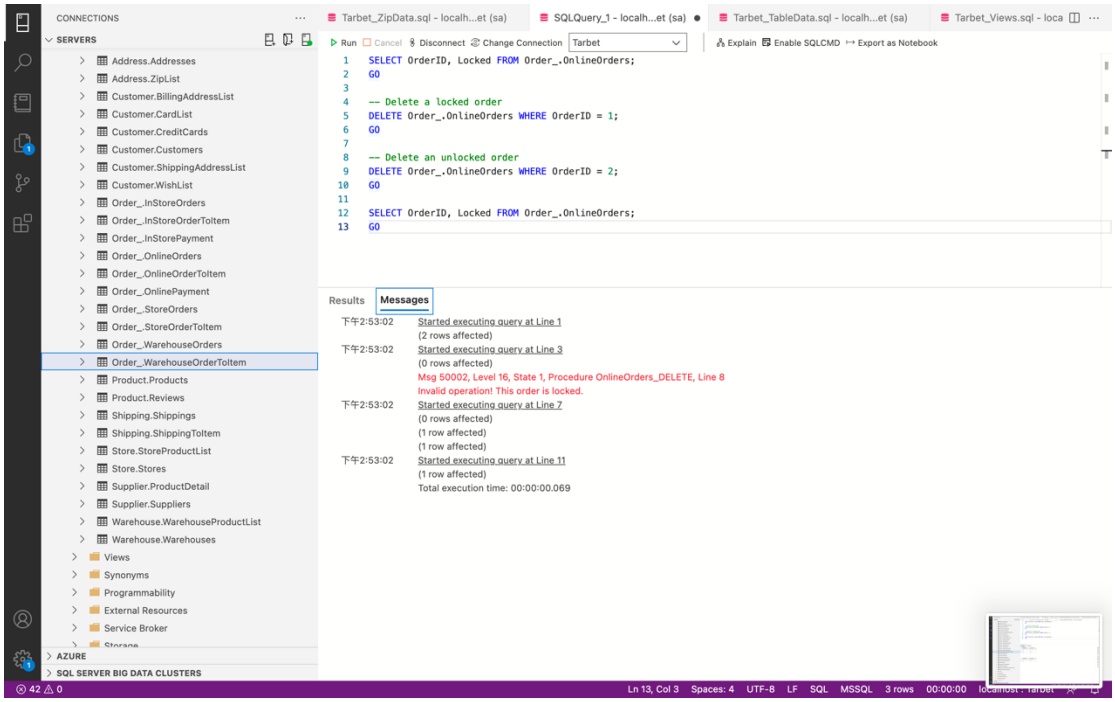
Test trigger OnlineOrderToItem_INSERT_UPDATE and OnlineOrderToItem_DELETE by adding or dropping items in an order. It shows that the TotalPrice of order is updated correctly.

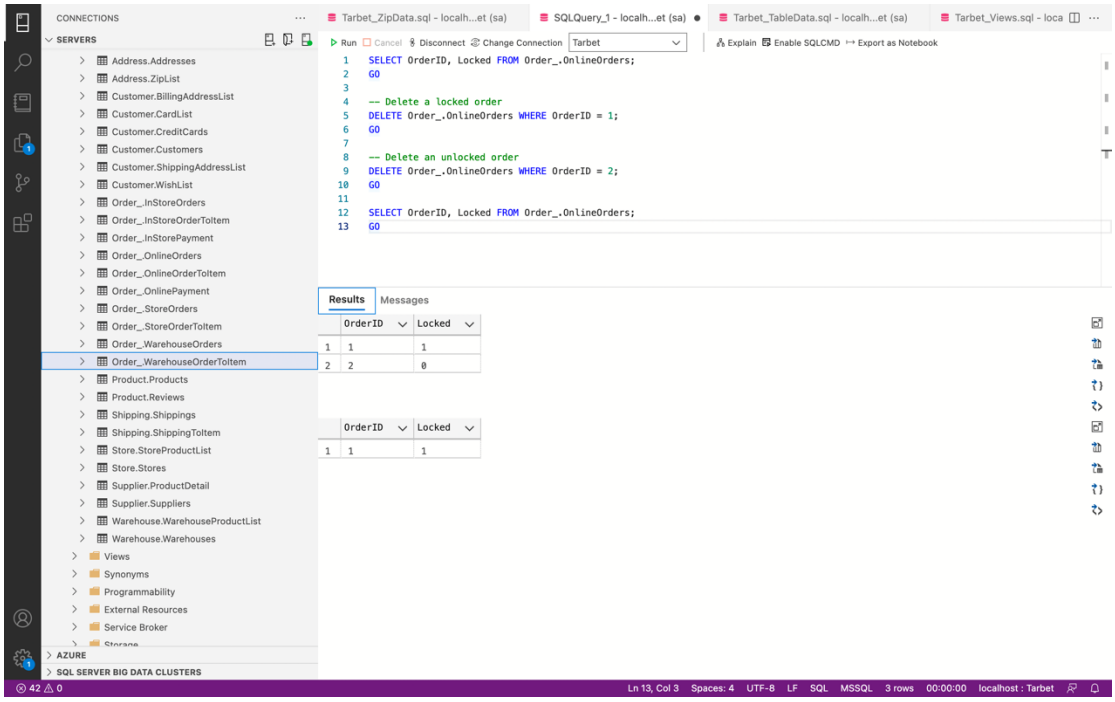


Test trigger Reviews_INSEERT by checking the AvgScore of the product after a customer writes a new review for it. The result shows that the AvgScore is updated correctly.



Test trigger OnlineOrders_DELETE by deleting a locked online order and an unlocked online order. The result shows that Order 1 keeps and Order 2 is deleted because Order 1 is locked and Order 2 is unlocked.

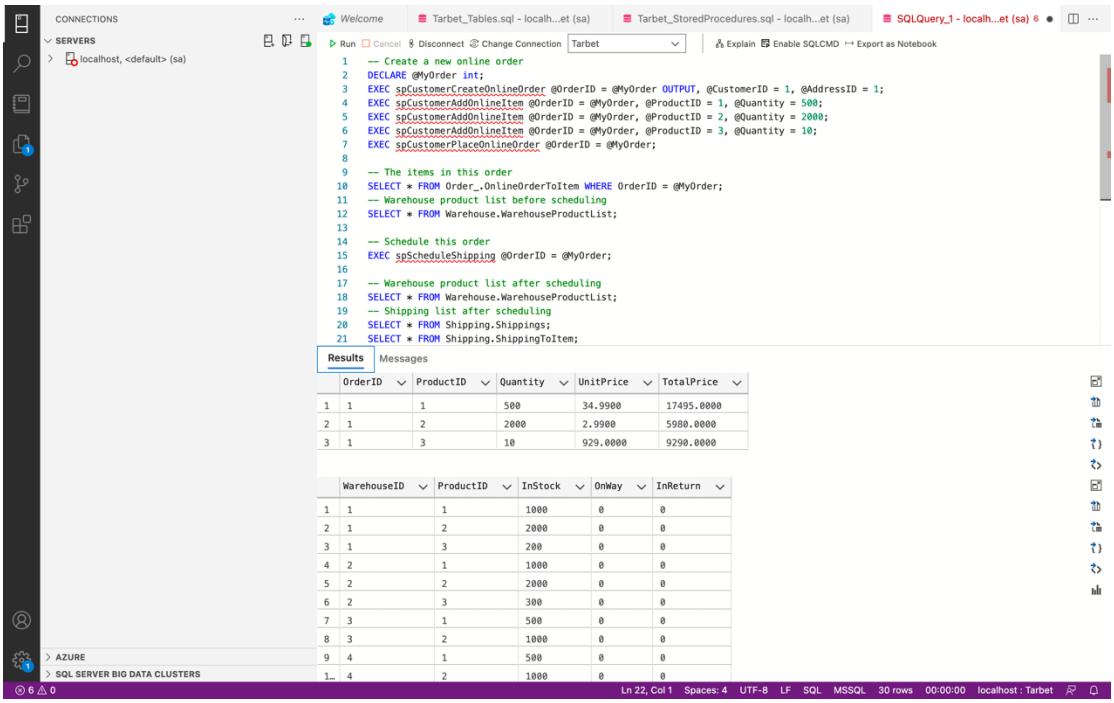




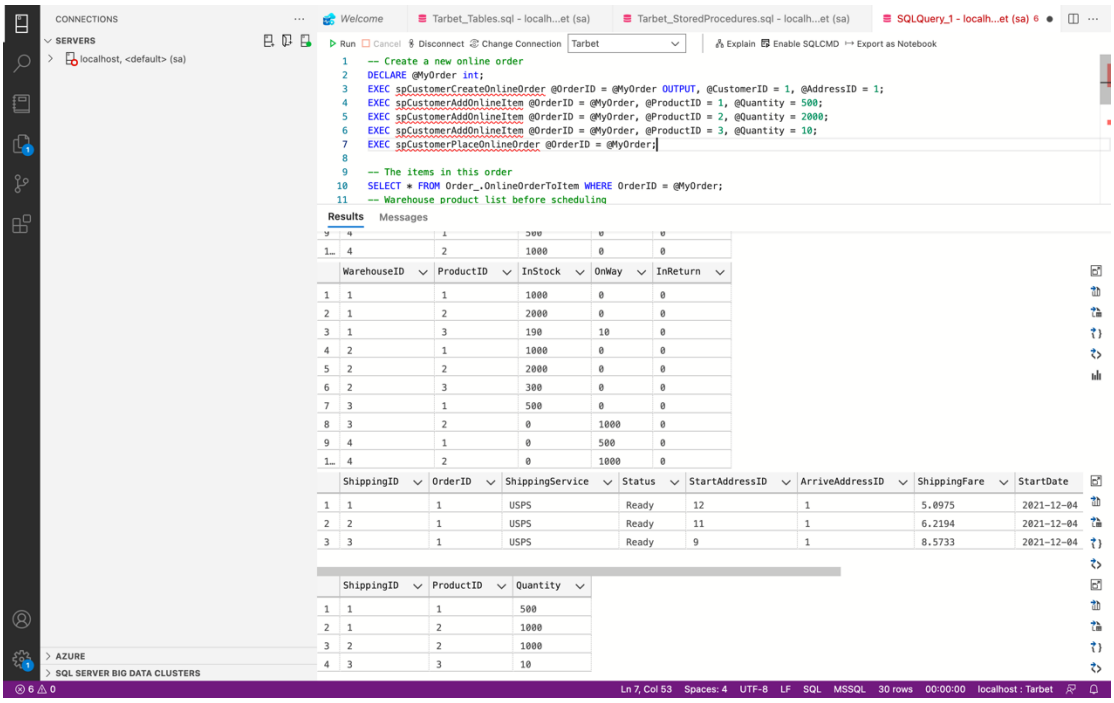
3.6 Test Transactions

Test the transaction in spScheduleShipping by calling spScheduleShipping for two orders. The first order contains many products, and these products need to be scheduled from several warehouses. The transaction schedules products from near to far, and the result is correct.

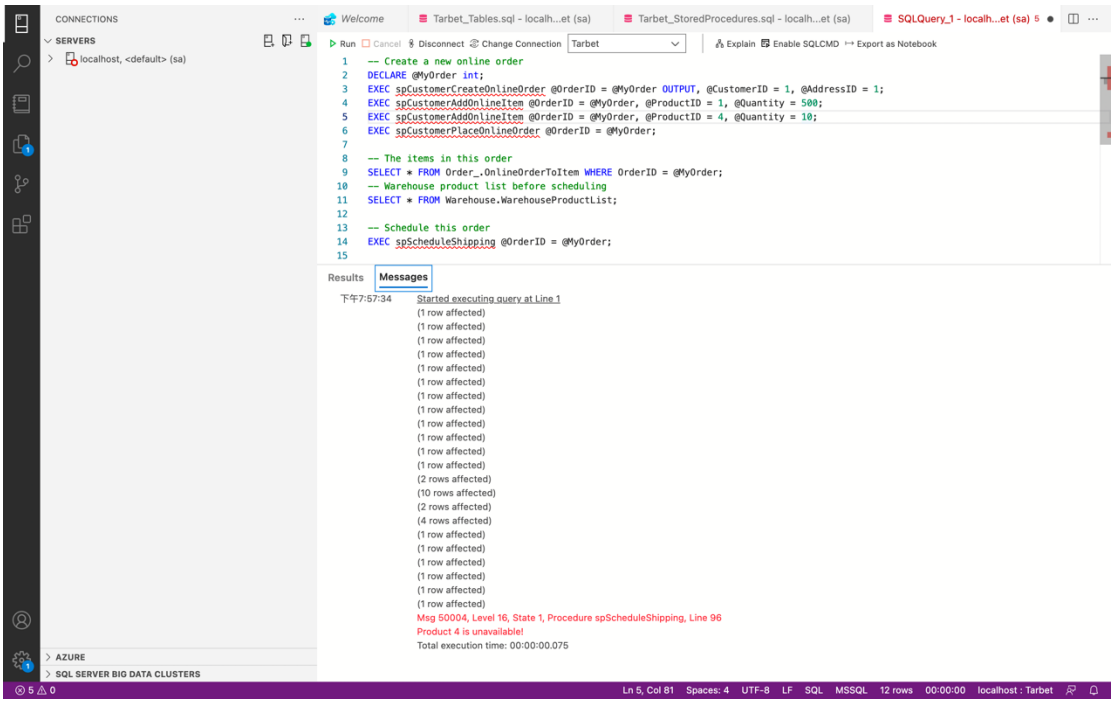
The OrderToItem table and WarehouseProductList table before scheduling.



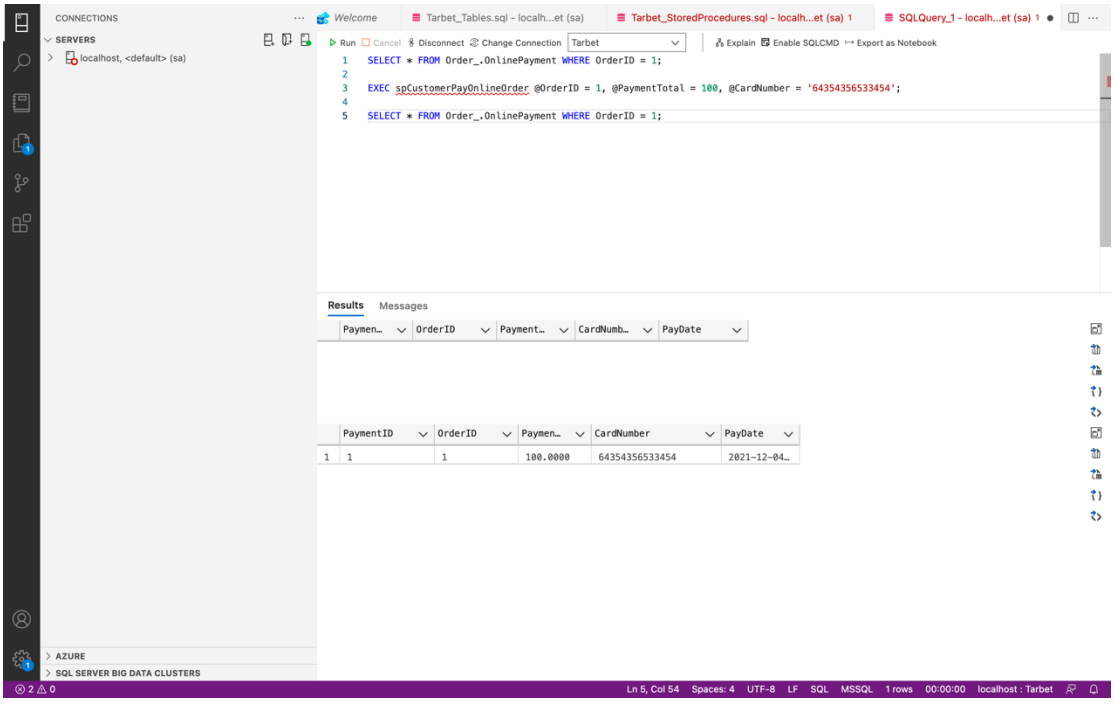
The WarehouseProductList table, Shippings table, and ShippingToItem table after scheduling.



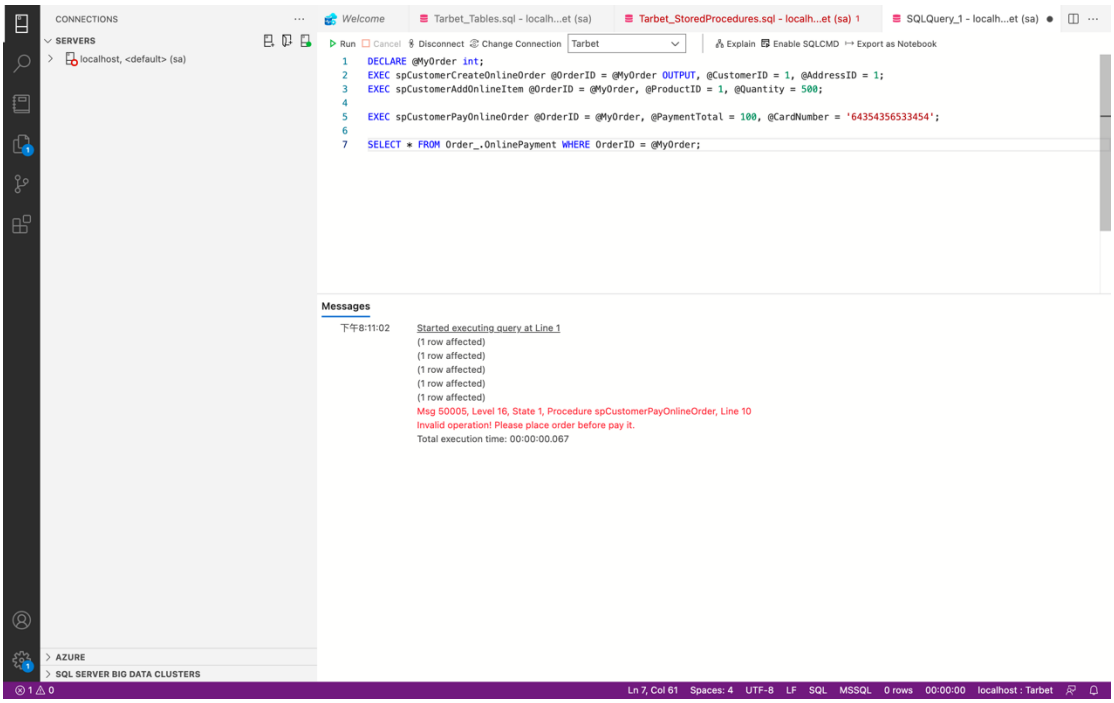
The second order contains an unavailable product in all warehouses. The transaction rollback and throw an error.



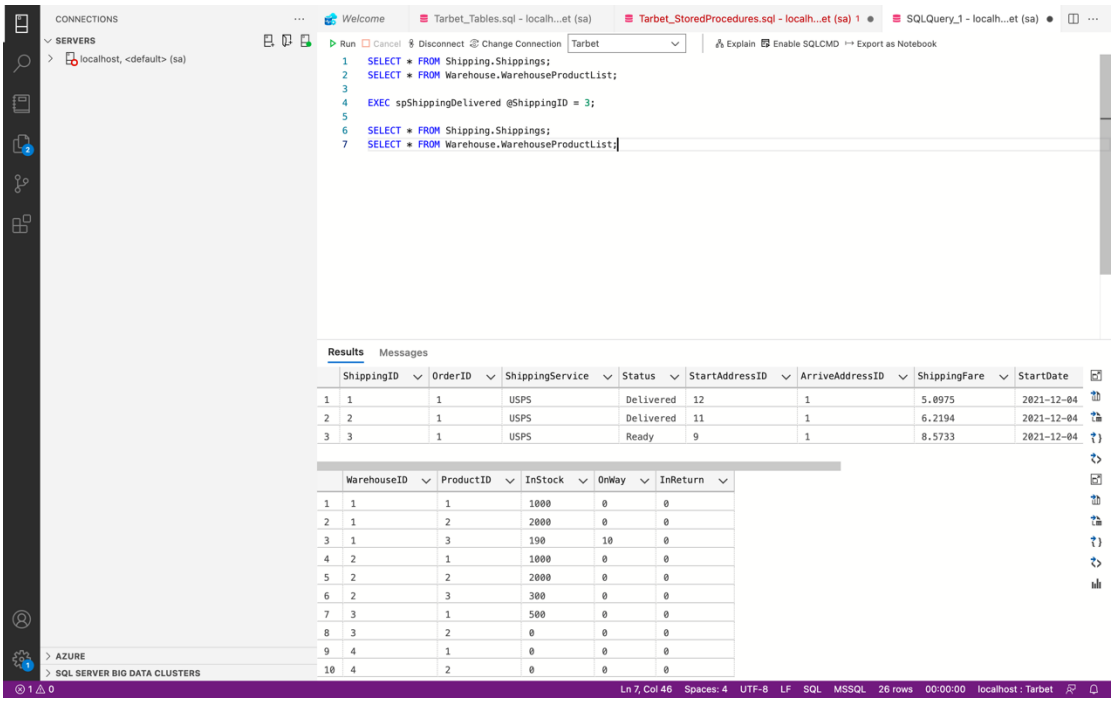
Test the transaction in spCustomerPayOnlineOrder by calling spCustomerPayOnlineOrder for two orders. The first order has been placed. The result shows that the customer pays it successfully.



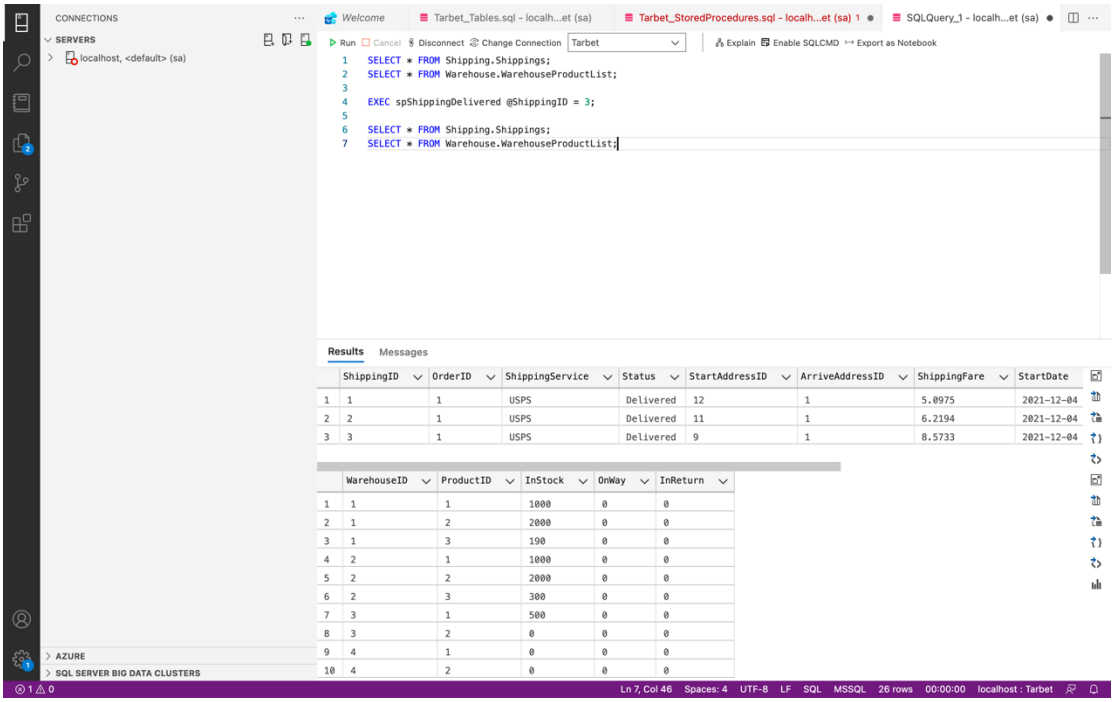
The second order hasn't been placed, so it causes an error.



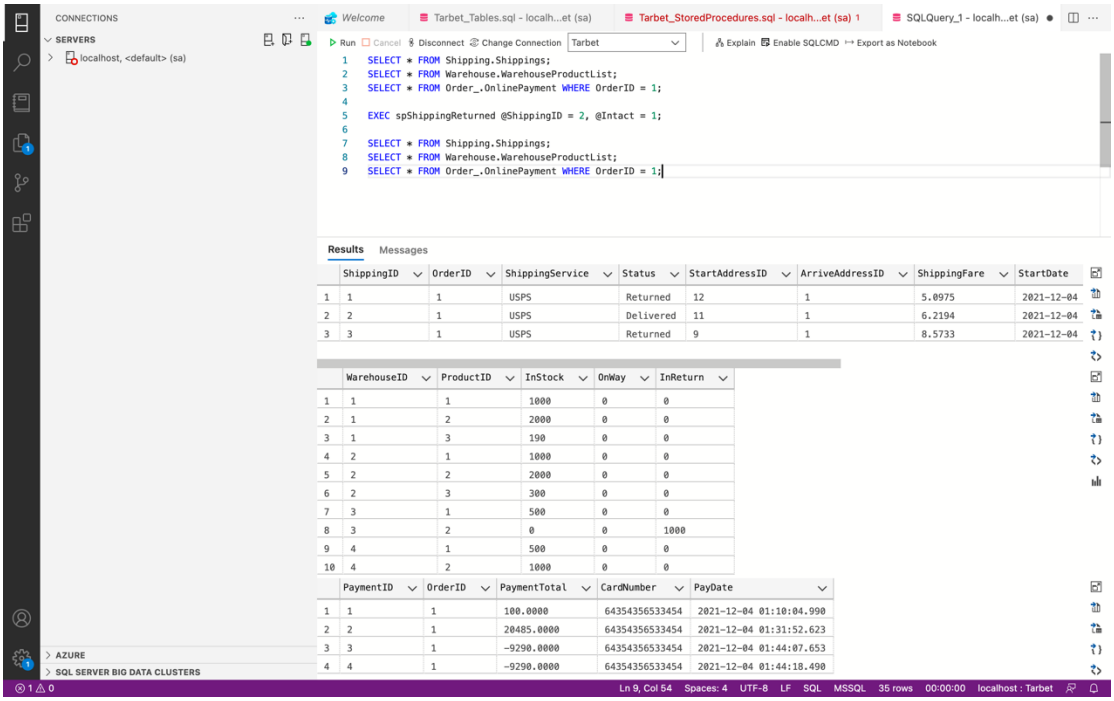
Test the transaction in spShippingDelievered by calling spShippingDelievered. These two screenshots show that the status of Shipping 3 and the OnWay column of WarehouseProductList are updated correctly.
The Shippings table and WarehouseProductList table before delivered.



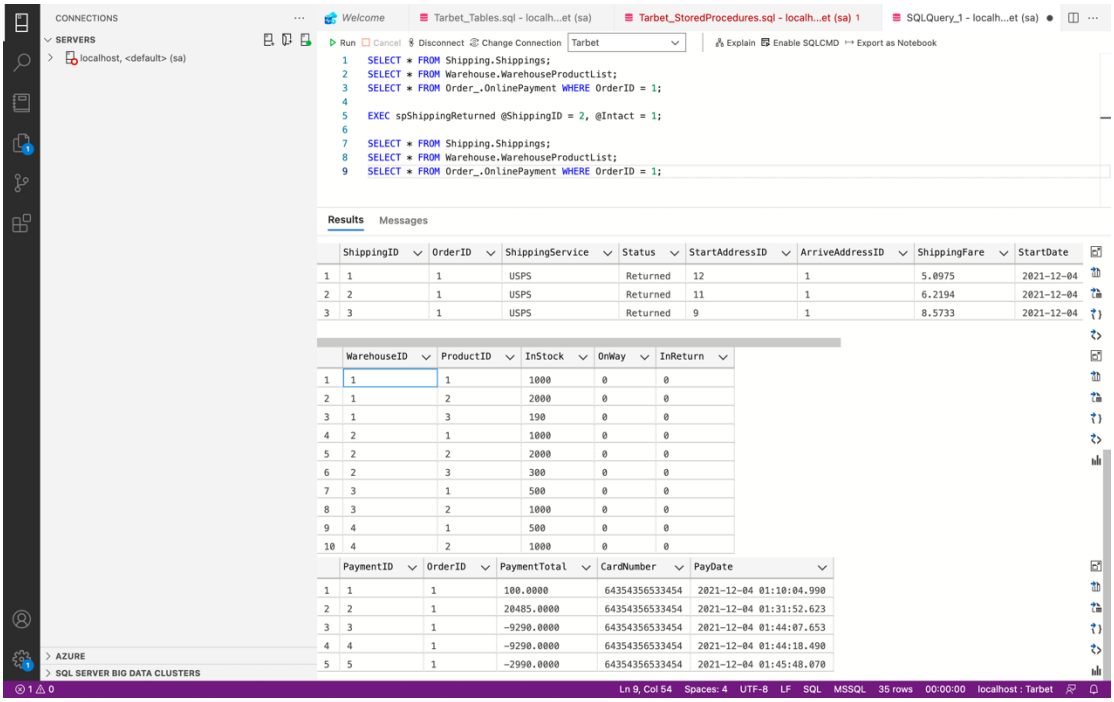
The Shippings table and WarehouseProductList table after delievered.



Test the transaction in spShippingReturned by calling spShippingReturned. These two screenshots show that Shipping 2 is returned. The InStock column and the InReturn column are updated correctly. The customer gets cashback with amount = $2990.
The Shippings table, WarehouseProductList table, and OnlinePayment table before shipping is returned.

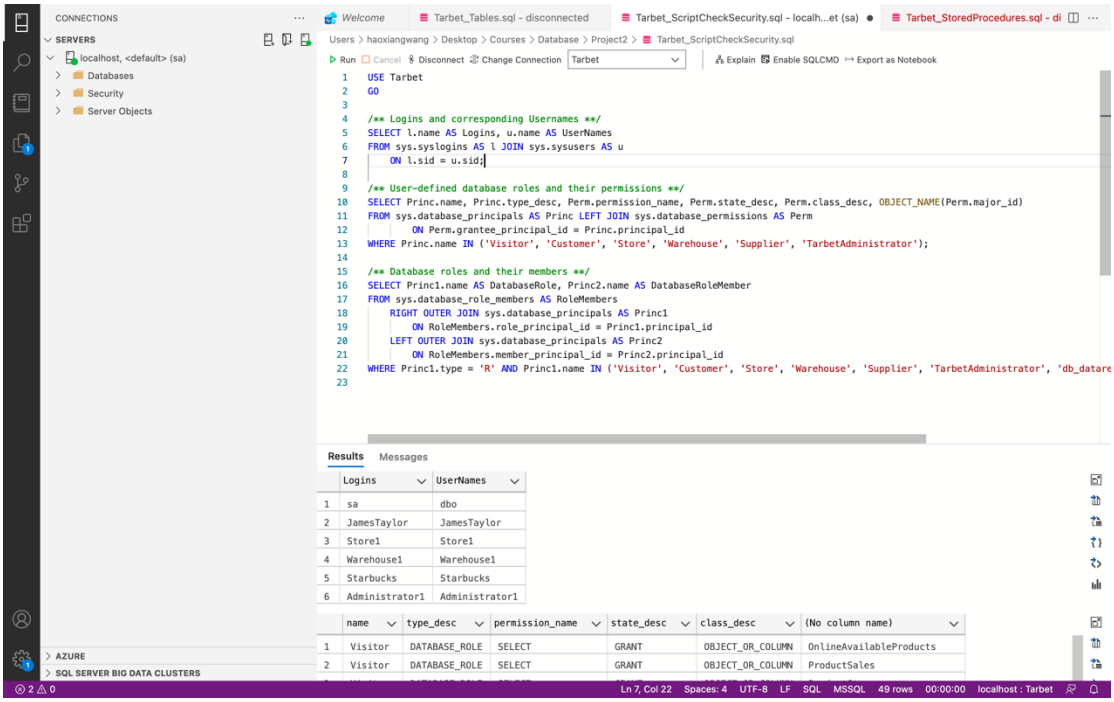


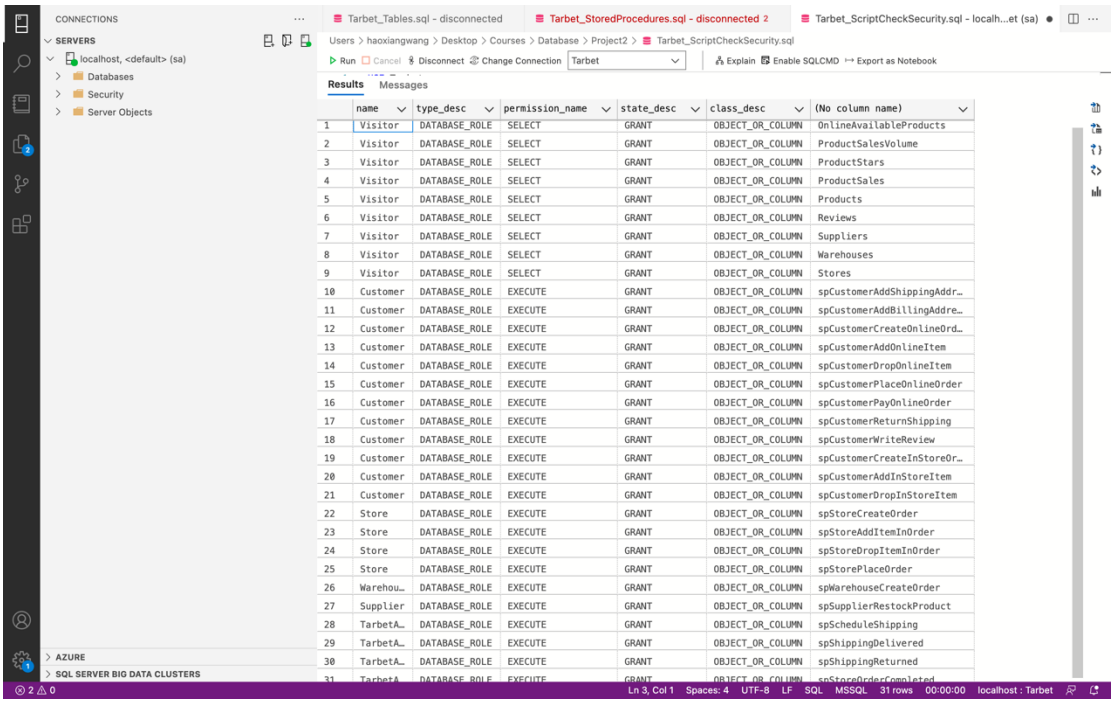
The Shippings table, WarehouseProductList table, and OnlinePayment table after shipping is returned.

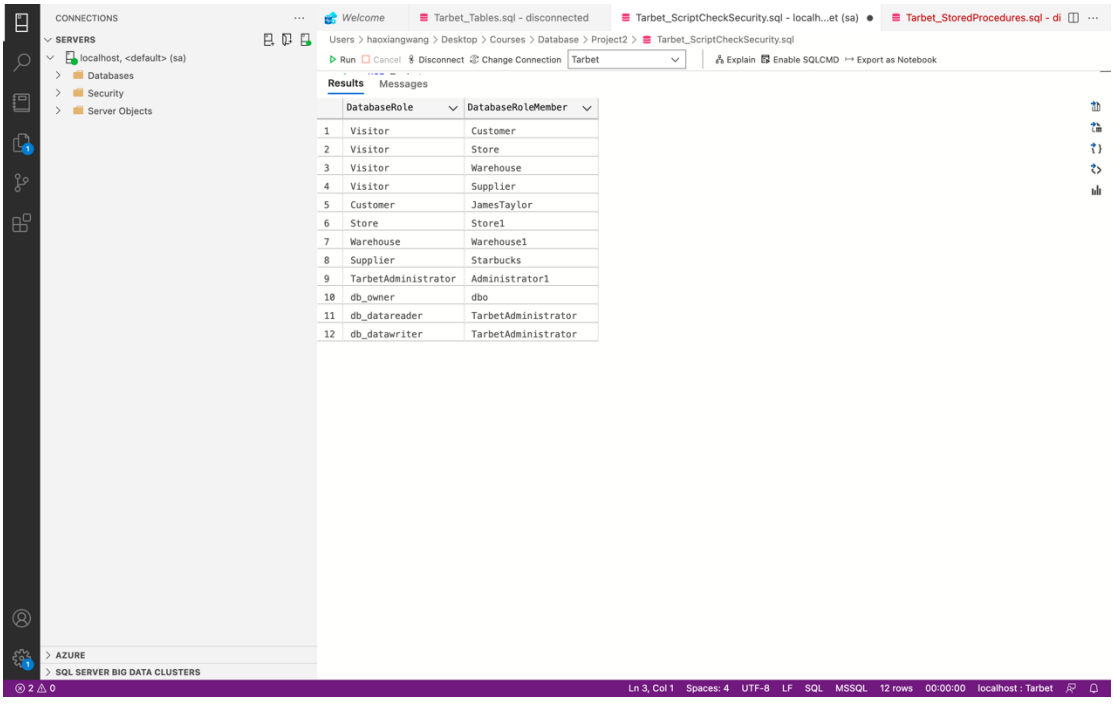


3.7 Test Scripts

Test scripts by showing database roles' permissions and members of Tarbet database. The following results show that my scripts grant permissions to database roles correctly, and all users are assigned to their database roles.




CONNECTIONS
SERVERS
localhost, <default> (sa)
Databases
Security
Server Objects
AZURE
SQL SERVER BIG DATA CLUSTERS
Tarbet_Tables.sql - disconnected
Tarbet_StoredProcedures.sql - disconnected 2
Tarbet_ScriptCheckSecurity.sql - localh...et (sa)
Users > haoxiangwang > Desktop > Courses > Database > Project2 > Tarbet_ScriptCheckSecurity.sql
Run Cancel Disconnect Change Connection Tarbet
Explain Enable SQLCMD Export as Notebook
Results Messages
name type_desc permission_name state_desc class_desc (No column name)
1 Visitor DATABASE_ROLE SELECT GRANT OBJECT_OR_COLUMN OnlineAvailableProducts
2 Visitor DATABASE_ROLE SELECT GRANT OBJECT_OR_COLUMN ProductSalesVolume
3 Visitor DATABASE_ROLE SELECT GRANT OBJECT_OR_COLUMN ProductStars
4 Visitor DATABASE_ROLE SELECT GRANT OBJECT_OR_COLUMN ProductSales
5 Visitor DATABASE_ROLE SELECT GRANT OBJECT_OR_COLUMN Products
6 Visitor DATABASE_ROLE SELECT GRANT OBJECT_OR_COLUMN Reviews
7 Visitor DATABASE_ROLE SELECT GRANT OBJECT_OR_COLUMN Suppliers
8 Visitor DATABASE_ROLE SELECT GRANT OBJECT_OR_COLUMN Warehouses
9 Visitor DATABASE_ROLE SELECT GRANT OBJECT_OR_COLUMN Stores
10 Customer DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spCustomerAddShippingAddr…
11 Customer DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spCustomerAddBillingAddre…
12 Customer DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spCustomerCreateOnlineOrd…
13 Customer DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spCustomerAddOnlineItem
14 Customer DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spCustomerDropOnlineItem
15 Customer DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spCustomerPlaceOnlineOrder
16 Customer DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spCustomerPayOnlineOrder
17 Customer DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spCustomerReturnShipping
18 Customer DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spCustomerWriteReview
19 Customer DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spCustomerCreateInStoreOr…
20 Customer DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spCustomerAddInStoreItem
21 Customer DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spCustomerDropInStoreItem
22 Store DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spStoreCreateOrder
23 Store DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spStoreAddItemInOrder
24 Store DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spStoreDropItemInOrder
25 Store DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spStorePlaceOrder
26 Warehou… DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spWarehouseCreateOrder
27 Supplier DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spSupplierRestockProduct
28 TarbetA… DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spScheduleShipping
29 TarbetA… DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spShippingDelivered
30 TarbetA… DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spShippingReturned
31 TarbetA… DATABASE_ROLE EXECUTE GRANT OBJECT_OR_COLUMN spStoreOrderCompleted
2 0
Ln 3, Col 1 Spaces: 4 UTF-8 LF SQL MSSQL 31 rows 00:00:00 localhost : Tarbet


CONNECTIONS
SERVERS
localhost, <default> (sa)
Databases
Security
Server Objects
AZURE
SQL SERVER BIG DATA CLUSTERS
Welcome
Tarbet_Tables.sql - disconnected
Tarbet_ScriptCheckSecurity.sql - localh...et (sa)
Tarbet_StoredProcedures.sql - di
Users > haoxiangwang > Desktop > Courses > Database > Project2 > Tarbet_ScriptCheckSecurity.sql
Run Cancel Disconnect Change Connection Tarbet
Explain Enable SQLCMD Export as Notebook
Results Messages
DatabaseRole DatabaseRoleMember
1 Visitor Customer
2 Visitor Store
3 Visitor Warehouse
4 Visitor Supplier
5 Customer JamesTaylor
6 Store Store1
7 Warehouse Warehouse1
8 Supplier Starbucks
9 TarbetAdministrator Administrator1
10 db_owner dbo
11 db_datareader TarbetAdministrator
12 db_datawriter TarbetAdministrator
2 0
Ln 3, Col 1 Spaces: 4 UTF-8 LF SQL MSSQL 12 rows 00:00:00 localhost : Tarbet

4. Conclusions

In this report, I design and implement the Tarbet database for Target's competitor. This database supports the storage of customer data, store data, warehouse data, supplier data, product data, different types of order data, and the management of operations such as online sales, in-store sales, product restocking, logistics and transportation, and customer returns. It consists of 9 schemas, 28 tables, 4 views, 4 triggers, 4 functions, more than 40 stored procedures and transactions, 7 database roles with different permissions, and 4 scripts to create logins, roles, users, and grant permissions. Besides, it prevents user from directly manipulating the data by restricting the user to only call the stored procedures, which ensures the security of the data.

During the design process, I encountered and solved many problems. One of the problems is how to schedule shipping at the least cost. Firstly, I found the latitude and longitude data of Zip codes on the official website. So, I can calculate the distance between two addresses. Then, I schedule shipping for warehouses in the order from near to far. If the nearest warehouse can't schedule all the items in the order, I will continue to find the required items in the next warehouse.

Another problem is to make the database design in line with common sense. For example, I set ON DELETE CASCADE when OrderToItem references Orders on the OrderID column. If someone deletes an order, the order's detail will be deleted automatically. However, after the customer places their order and the system schedules shipping for this order, this order can't be deleted anymore, so I add the Locked column in Orders. When an order is deleted, it will trigger a trigger that checks if this order is locked and deletes it when it is not locked.

The third problem is to solve data security problems. I designed stored procedures for different users. Users can't edit data directly. They can only submit changes by executing stored procedures. The stored procedures can check the user's authority based on his role and username. For example, a customer can only create orders for himself and edit the orders created by him only.

Appendix

I. Codes

All the codes can be found here:
https://github.com/haoxiangwang123/A-Target-Competitor
If you can't visit it, please contact me (hwang214@syr.edu) to get the access.

II. Error Types

Error Code	Information
50001	Invalid OrderID!
50002	Invalid operation! This order is locked.
50003	Order expired!
50004	Product [ProductID] is unavailable!
50005	Invalid operation! Please place order before paying it.
50006	Invalid ShippingID!
50007	Invalid ProductID!
50008	Insufficient permissions!

III. Roles and Permissions

Role	Permission	Object
Visitor (Permissions here are available to all roles)	SELECT	Product.Products Product.Reviews Store.Stores Warehouse.Warehouses Supplier.Suppliers OnlineAvailableProducts ProductSales ProductSalesVolume ProductStars
Customer	EXEC	spCustomerCreateAccount* spCustomerShowBillingAddress* spCustomerAddBillingAddress spCustomerDropBillingAddress* spCustomerShowShippingAddress* spCustomerAddShippingAddress spCustomerDropShippingAddress* spCustomerShowCardList* spCustomerAddCreditCard* spCustomerDropCreditCard* spCustomerCreateOnlineOrder spCustomerShowOnlineOrderList*

		spCustomerAddOnlineItem spCustomerDropOnlineItem spCustomerPlaceOnlineOrder spCustomerPayOnlineOrder spCustomerWriteReview spCustomerCreateInStoreOrder spCustomerShowInStoreOrderList* spCustomerAddInStoreItem spCustomerDropInStoreItem spCustomerPayInStoreOrder* spCustomerReturnProduct* spCustomerReturnShipping
Store	EXEC	spStoreCreateOrder spStoreAddItemInOrder spStoreDropItemInOrder spStorePlaceOrder spStoreShowOrderList* spStoreShowProductList*
Warehouse	EXEC	spWarehouseCreateOrder spWarehouseAddItemInOrder* spWarehouseDropItemInOrder* spWarehousePlaceOrder* spWarehouseShowOrderList* spWarehouseShowProductList*
Supplier	EXEC	spSupplierCreateProduct* spSupplierRestockProduct spSupplierShowProductList*
TarbetAdministrator	EXEC	spScheduleShipping spShippingDelivered spShippingReturned spStoreOrderCompleted spWarehouseOrderCompleted* spAlterProductPrice*
	SELECT UPDATE INSERT DELETE	All user tables and views
db_owner	All permissions	All objects

(*): This object has not yet been implemented.