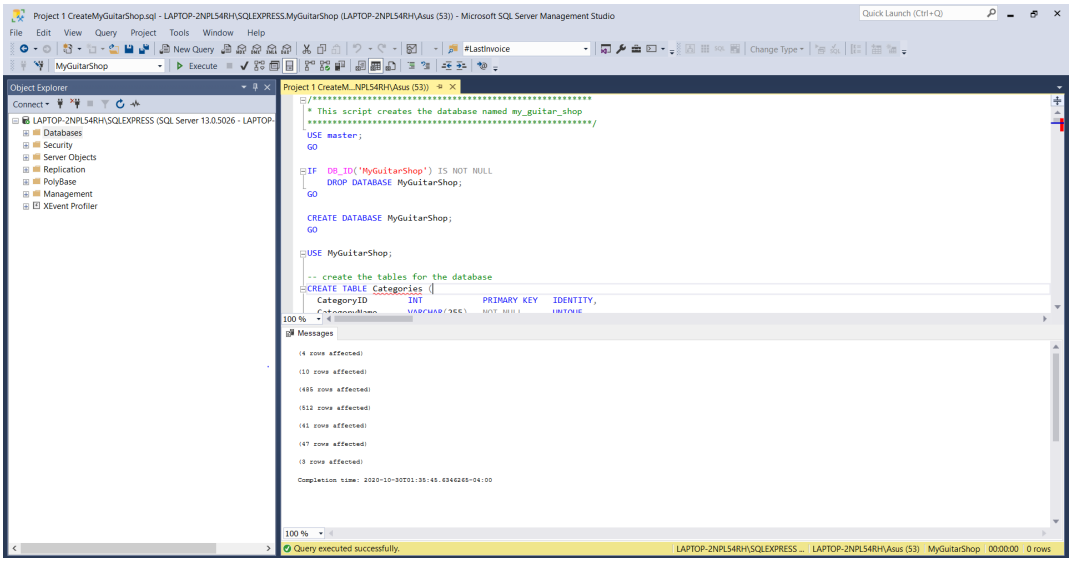
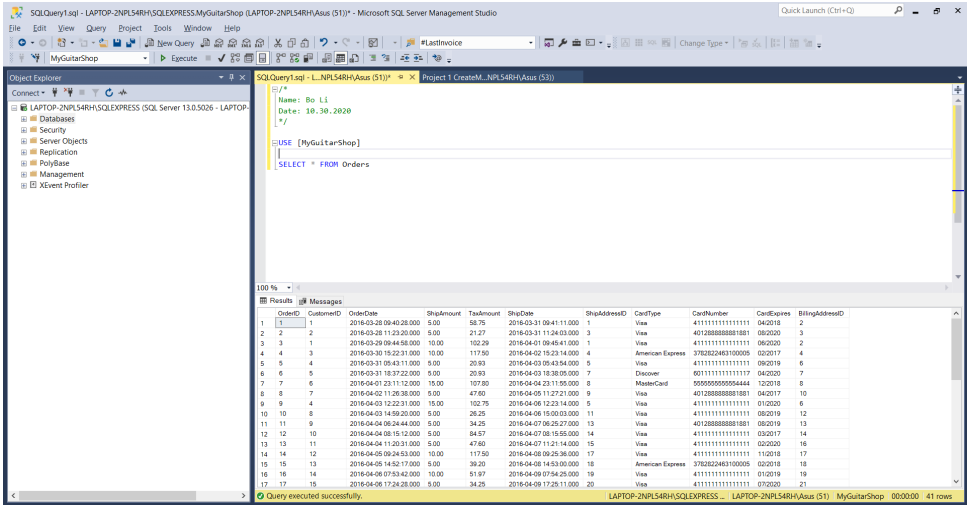
I. My Guitar Shop Database

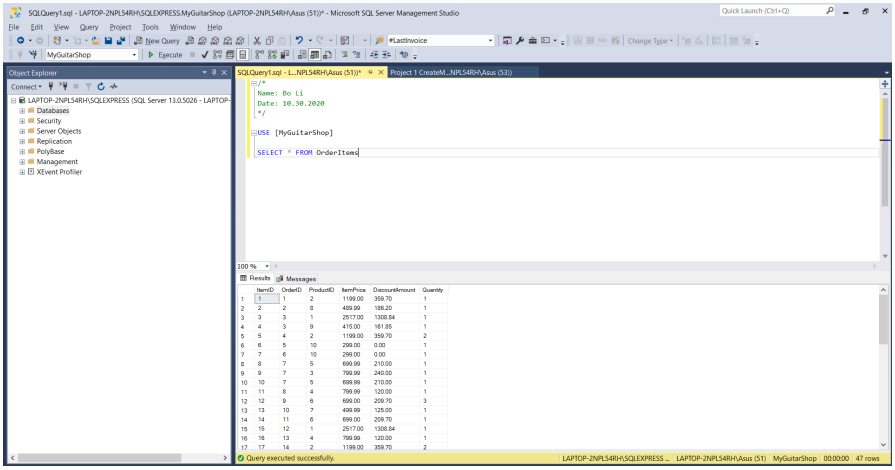
A. Database Setup

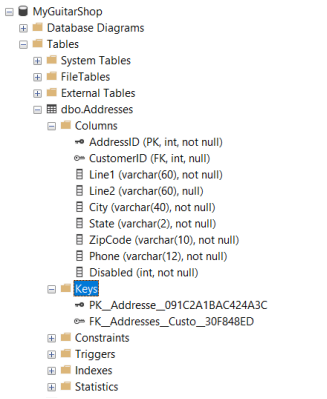
1. (1) Download CreateMyGuitarShop.sql from Project 1 directory on Blackboard and open it in SQL server management studio. Execute the entire script and show the message in the Message tab, indicating the script is executed successfully. A complete screenshot of execution result is required. Your screenshot should show your entire SQL server window. You are not allowed to crop out any part and follow this for all the questions in this project.

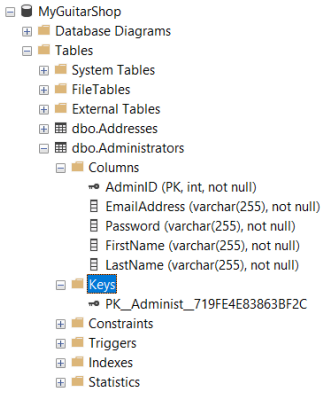


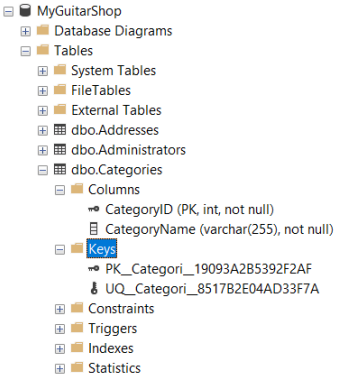
2. (2) Navigate through the database objects and view the column definitions for each table. Open a new Query Editor window. Show details in Orders table and OrdersItems table using SELECT statement.Full screenshots of execution results are required as mentioned.

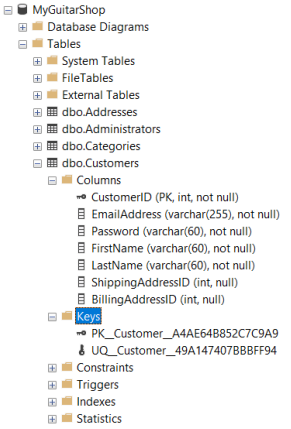


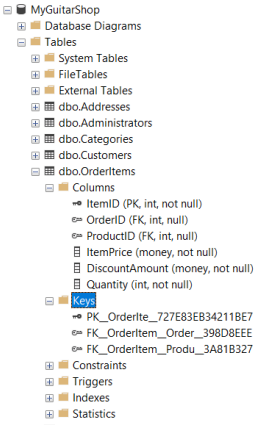

SQLQuery1.sql - LAPTOP-2NPL54RH\SQLEXPRESS.MyGuitarShop (LAPTOP-2NPL54RH\Asus (51))* - Microsoft SQL Server Management Studio
Object Explorer
SQLQuery1.sql - L...NPL54RH\Asus (51))*
/*
Name: Bo Li
Date: 10.30.2020
*/
USE [MyGuitarShop]
SELECT * FROM OrderItems
Results
Messages
Query executed successfully.

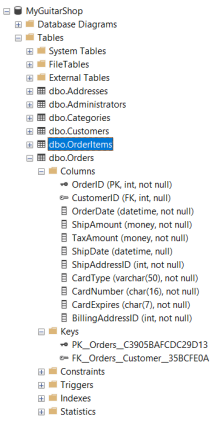

MyGuitarShop
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Addresses
Columns
AddressID (PK, int, not null)
CustomerID (FK, int, not null)
Line1 (varchar(60), not null)
Line2 (varchar(60), null)
City (varchar(40), not null)
State (varchar(2), not null)
ZipCode (varchar(10), not null)
Phone (varchar(12), not null)
Disabled (int, not null)
Keys
PK__Addresse__091C2A1BAC424A3C
FK__Addresses__Custo__30F848ED
Constraints
Triggers
Indexes
Statistics

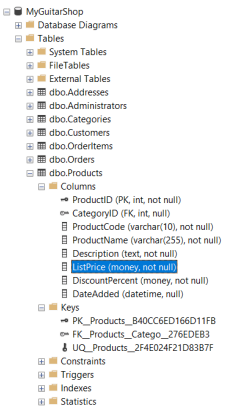

MyGuitarShop
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Addresses
dbo.Administrators
Columns
AdminID (PK, int, not null)
EmailAddress (varchar(255), not null)
Password (varchar(255), not null)
FirstName (varchar(255), not null)
LastName (varchar(255), not null)
Keys
PK__Administ__719FE4E83863BF2C
Constraints
Triggers
Indexes
Statistics


MyGuitarShop
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Addresses
dbo.Administrators
dbo.Categories
Columns
CategoryID (PK, int, not null)
CategoryName (varchar(255), not null)
Keys
PK__Categori__19093A2B5392F2AF
UQ__Categori__8517B2E04AD33F7A
Constraints
Triggers
Indexes
Statistics


MyGuitarShop
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Addresses
dbo.Administrators
dbo.Categories
dbo.Customers
Columns
CustomerID (PK, int, not null)
EmailAddress (varchar(255), not null)
Password (varchar(60), not null)
FirstName (varchar(60), not null)
LastName (varchar(60), not null)
ShippingAddressID (int, null)
BillingAddressID (int, null)
Keys
PK__Customer__A4AE64B852C7C9A9
UQ__Customer__49A147407BBBFF94
Constraints
Triggers
Indexes
Statistics


MyGuitarShop
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Addresses
dbo.Administrators
dbo.Categories
dbo.Customers
dbo.OrderItems
Columns
ItemID (PK, int, not null)
OrderID (FK, int, null)
ProductID (FK, int, null)
ItemPrice (money, not null)
DiscountAmount (money, not null)
Quantity (int, not null)
Keys
PK__OrderIte__727E83EB34211BE7
FK__OrderItem__Order__398D8EEE
FK__OrderItem__Produ__3A81B327
Constraints
Triggers
Indexes
Statistics

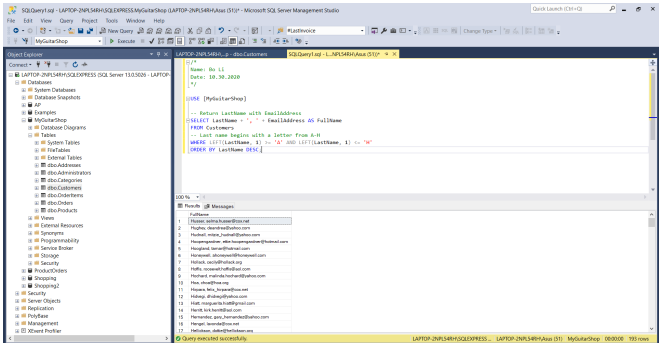

MyGuitarShop
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Addresses
dbo.Administrators
dbo.Categories
dbo.Customers
dbo.OrderItems
dbo.Orders
Columns
OrderID (PK, int, not null)
CustomerID (FK, int, null)
OrderDate (datetime, not null)
ShipAmount (money, not null)
TaxAmount (money, not null)
ShipDate (datetime, null)
ShipAddressID (int, not null)
CardType (varchar(50), not null)
CardNumber (char(16), not null)
CardExpires (char(7), not null)
BillingAddressID (int, not null)
Keys
PK__Orders__C3905BAFCDC29D13
FK__Orders__Customer__35BCFE0A
Constraints
Triggers
Indexes
Statistics


MyGuitarShop
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Addresses
dbo.Administrators
dbo.Categories
dbo.Customers
dbo.OrderItems
dbo.Orders
dbo.Products
Columns
ProductID (PK, int, not null)
CategoryID (FK, int, null)
ProductCode (varchar(10), not null)
ProductName (varchar(255), not null)
Description (text, not null)
ListPrice (money, not null)
DiscountPercent (money, not null)
DateAdded (datetime, null)
Keys
PK__Products__B40CC6ED166D11FB
FK__Products__Catego__276EDEB3
UQ__Products__2F4E024F21D83B7F
Constraints
Triggers
Indexes
Statistics

B. An Introduction to SQL

1. [3] Write a SELECT statement that returns one column from the Customers table named FullName that joins the LastName and EmailAddress columns. Format this column with the last name, a comma, a space, and then Email Address like this: Abdallah, johnetta_abdallah@aol.com Add an ORDER BY clause to this statement that sorts the result set by last name in descending sequence. Return only the contacts whose last name begins with a letter from A to H.

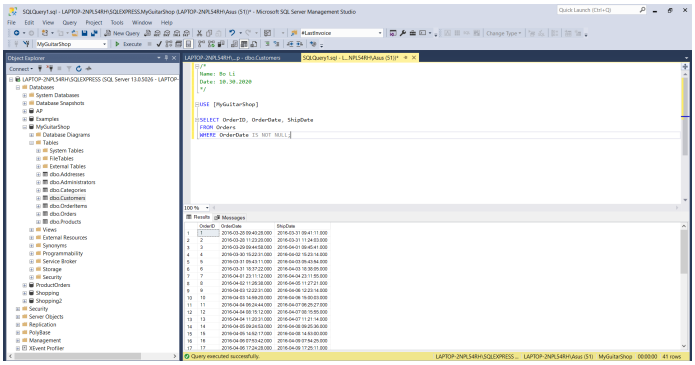
```
USE [MyGuitarShop]

-- Return LastName with EmailAddress
SELECT LastName + ', ' + EmailAddress AS FullName
FROM Customers
-- Last name begins with a letter from A-H
WHERE LEFT(LastName, 1) >= 'A' AND LEFT(LastName, 1) <= 'H'
ORDER BY LastName DESC;
```



2. [3] Write a SELECT statement that returns these columns from the Orders table:
   OrderID The OrderID column
   OrderDate The OrderDate column
   ShipDate The ShipDate column

   Return only the rows where the OrderDate column does not contain a null value.

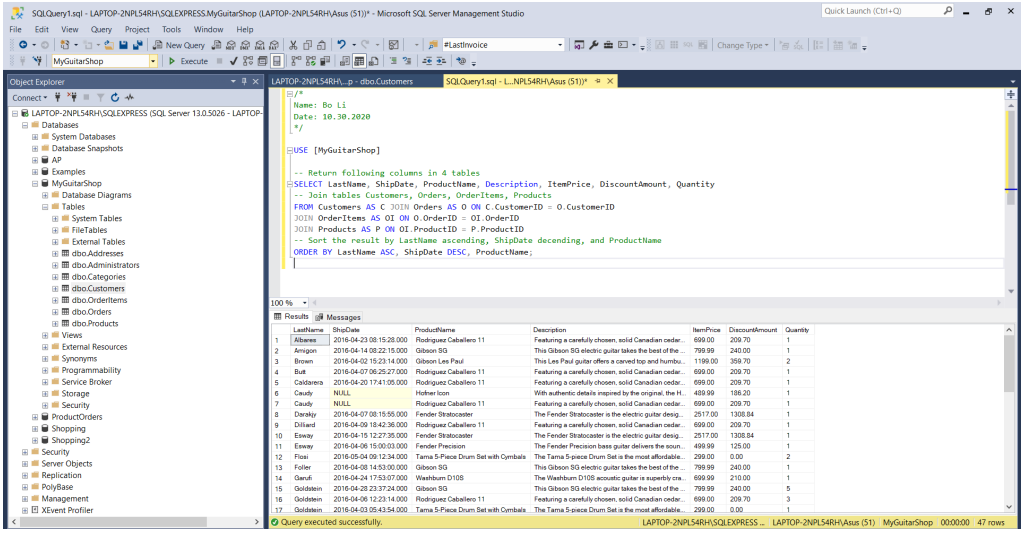
```
USE [MyGuitarShop]

SELECT OrderID, OrderDate, ShipDate
FROM Orders
WHERE OrderDate IS NOT NULL;
```



C. The essential SQL skills

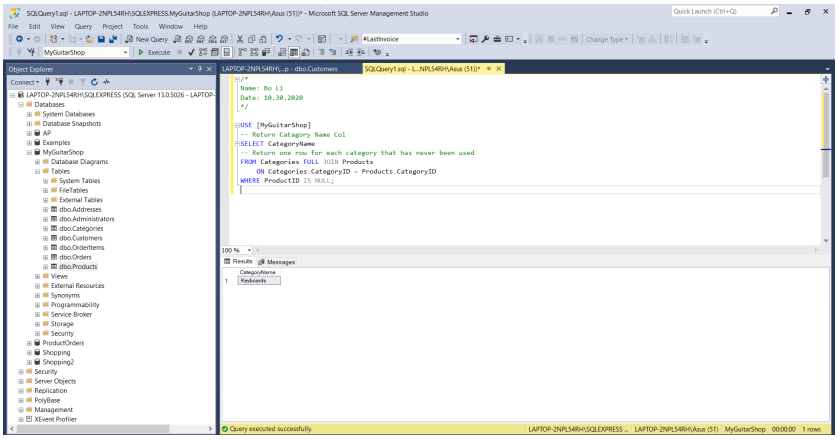
1. [4] Write a SELECT statement that joins the Customers, Orders, OrderItems, and Products tables. This statement should return these columns: LastName, ShipDate, ProductName, Description, ItemPrice, DiscountAmount, and Quantity. Use aliases for the tables. Sort the final result set by LastName in ascending order, and in descending order for ShipDate, and ProductName.

```
USE [MyGuitarShop]

-- Return following columns in 4 tables
SELECT LastName, ShipDate, ProductName, Description, ItemPrice, DiscountAmount,
Quantity
-- Join tables Customers, Orders, OrderItems, Products
FROM Customers AS C JOIN Orders AS O ON C.CustomerID = O.CustomerID
JOIN OrderItems AS OI ON O.OrderID = OI.OrderID
JOIN Products AS P ON OI.ProductID = P.ProductID
-- Sort the result by LastName ascending, ShipDate decending, and ProductName
ORDER BY LastName ASC, ShipDate DESC, ProductName;
```



2. [4] Write a SELECT statement that returns CategoryName column from the Categories table. Return one row for each category that has never been used. (Hint: Use an outer join and only return rows where the ProductID column contains a null value.)

```
USE [MyGuitarShop]
-- Return Catagory Name Col
SELECT CategoryName
-- Return one row for each category that has never been used
FROM Categories FULL JOIN Products
    ON Categories.CategoryID = Products.CategoryID
WHERE ProductID IS NULL;
```



3. [4] Write a SELECT statement that returns one row for each customer that has orders with these columns:

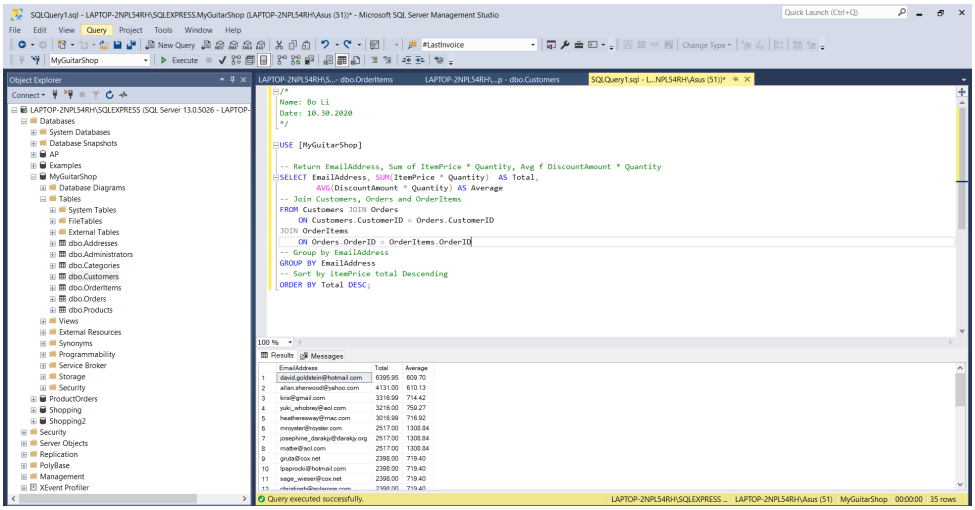
    a) The EmailAddress column from the Customers table

    b) The sum of the ItemPrice in the OrderItems table multiplied by the quantity in the OrderItems table

    c) The average of the DiscountAmount column in the OrderItems table multiplied by the quantity in the OrderItems table.

    Sort the result set in descending sequence by the item price total for each customer.

```
USE [MyGuitarShop]

-- Return EmailAddress, Sum of ItemPrice * Quantity, Avg f DiscountAmount *
Quantity
SELECT EmailAddress, SUM(ItemPrice * Quantity)  AS Total,
          AVG(DiscountAmount * Quantity) AS Average
-- Join Customers, Orders and OrderItems
FROM Customers JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID
JOIN OrderItems
    ON Orders.OrderID = OrderItems.OrderID
-- Group by EmailAddress
GROUP BY EmailAddress
-- Sort by itemPrice total Descending
ORDER BY Total DESC;
```



4. [4] Write a SELECT statement that returns one row for each customer that has orders with these columns:

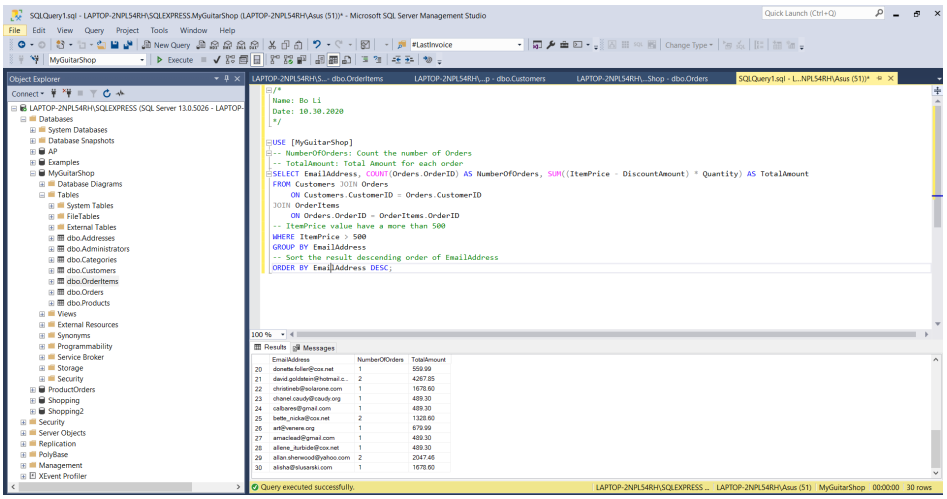
    a) The EmailAddress column from the Customers table

    b) A count of the number of orders

    c) The total amount for each order (Hint: First, subtract the discount amount from the price. Then, multiply by the quantity)

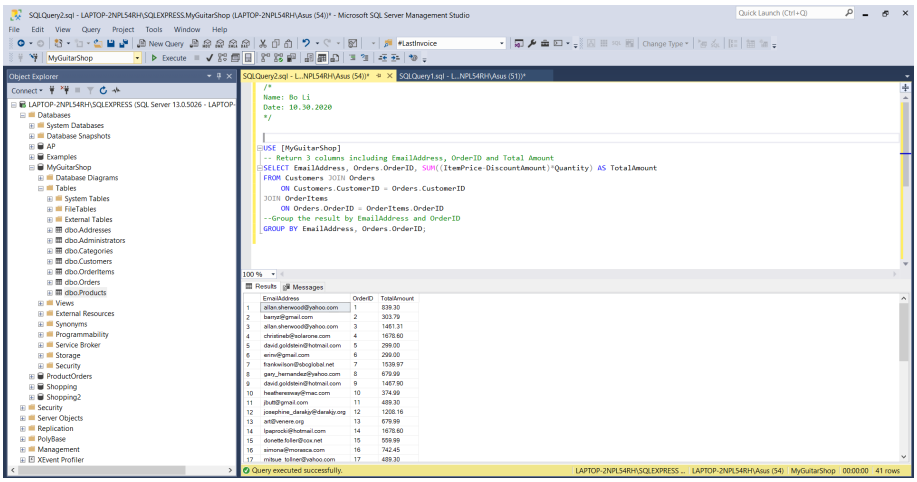
    Return only those rows where items have a more than 500 ItemPrice value. Sort the result set in descending order of EmailAddress column.

```
USE [MyGuitarShop]
-- NumberOfOrders: Count the number of Orders
-- TotalAmount: Total Amount for each order
SELECT EmailAddress, COUNT(Orders.OrderID) AS NumberOfOrders, SUM((ItemPrice -
DiscountAmount) * Quantity) AS TotalAmount
FROM Customers JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID
JOIN OrderItems
    ON Orders.OrderID = OrderItems.OrderID
-- ItemPrice value have a more than 500
WHERE ItemPrice > 500
GROUP BY EmailAddress
-- Sort the result descending order of EmailAddress
ORDER BY EmailAddress DESC;
```



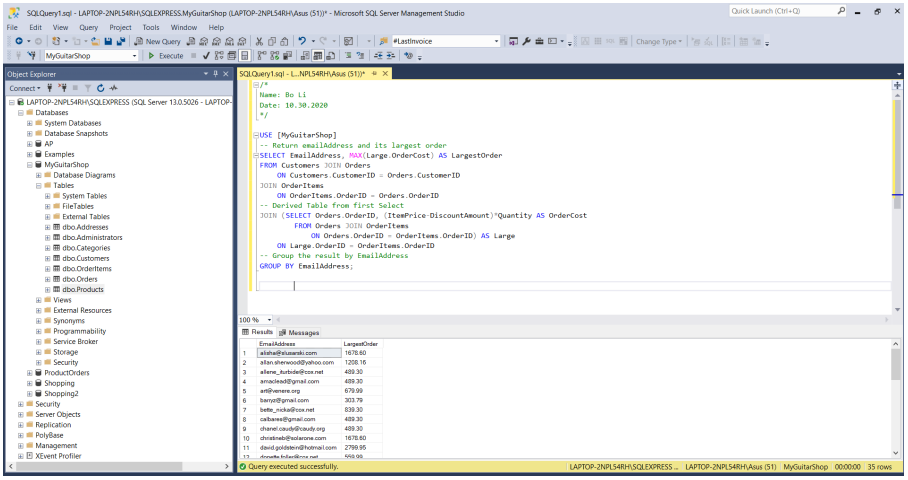
5. [4] (1) Write a SELECT statement that returns three columns: EmailAddress, OrderID, and the order total amount for each customer. To do this, you can group the result set by the EmailAddress and OrderID columns. In addition, you must calculate the order total amount from the columns in the OrderItems table.

```
USE [MyGuitarShop]
-- Return 3 columns including EmailAddress, OrderID and Total Amount
SELECT EmailAddress, Orders.OrderID, SUM((ItemPrice-DiscountAmount)*Quantity)
AS TotalAmount
FROM Customers JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID
JOIN OrderItems
    ON Orders.OrderID = OrderItems.OrderID
--Group the result by EmailAddress and OrderID
GROUP BY EmailAddress, Orders.OrderID;
```



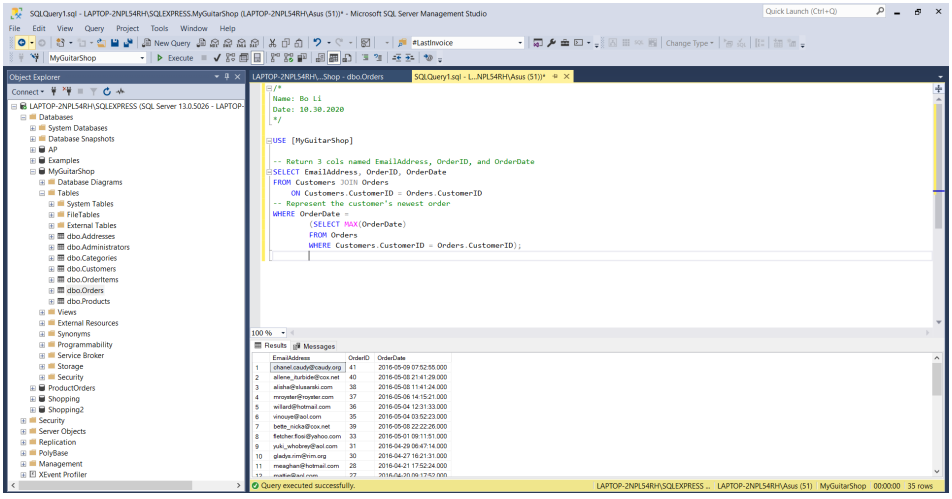
(2) Write a second SELECT statement that uses the first SELECT statement in its FROM clause. The main query should return two columns: the customer's email address and the largest order for that customer. To do this, you can group the result set by the EmailAddress column.

```
USE [MyGuitarShop]
-- Return emailAddress and its largest order
SELECT EmailAddress, MAX(Large.OrderCost) AS LargestOrder
FROM Customers JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID
JOIN OrderItems
    ON OrderItems.OrderID = Orders.OrderID
-- Derived Table from first Select
JOIN (SELECT Orders.OrderID, (ItemPrice-DiscountAmount)*Quantity AS OrderCost
          FROM Orders JOIN OrderItems
                ON Orders.OrderID = OrderItems.OrderID) AS Large
    ON Large.OrderID = OrderItems.OrderID
-- Group the result by EmailAddress
GROUP BY EmailAddress;
```



6. [4] Use a correlated subquery to return one row per customer, representing the customer's newest order (the one with the latest date). Each row should include these three columns: EmailAddress, OrderID, and OrderDate.

```
USE [MyGuitarShop]

-- Return 3 cols named EmailAddress, OrderID, and OrderDate
SELECT EmailAddress, OrderID, OrderDate
FROM Customers JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID
-- Represent the customer's newest order
WHERE OrderDate =
            (SELECT MAX(OrderDate)
            FROM Orders
            WHERE Customers.CustomerID = Orders.CustomerID);
```



7. [4] Write a SELECT statement that returns these columns from the Products table:

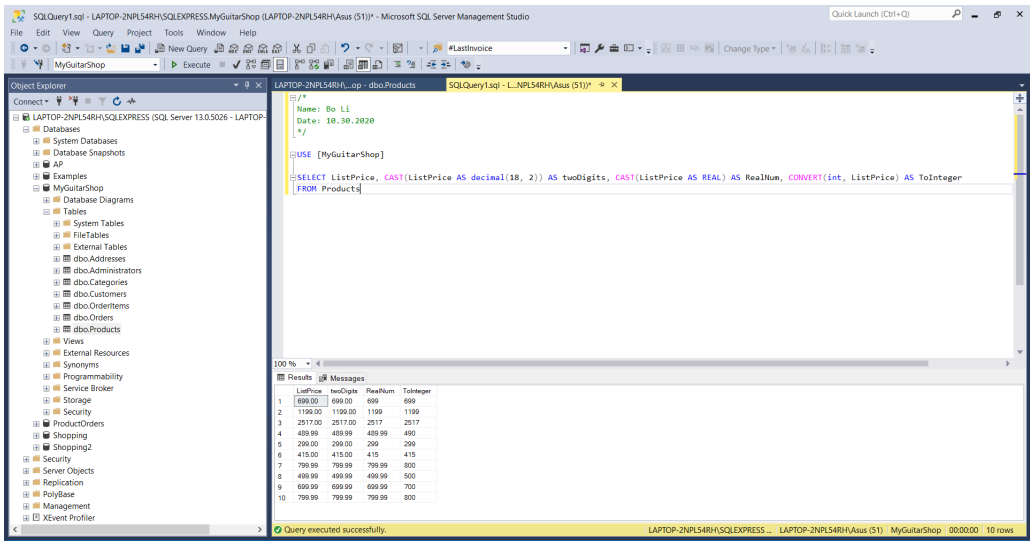
a) The ListPrice column

b) A column that uses the CAST function to return the ListPrice column with 2 digits to the right of the decimal point

c) A column that uses the CAST function to return the ListPrice column as a real number

d) A column that uses the CONVERT function to return the ListPrice column as an integer.

```
USE [MyGuitarShop]

SELECT ListPrice, CAST(ListPrice AS decimal(18, 2)) AS twoDigits,
CAST(ListPrice AS REAL) AS RealNum, CONVERT(int, ListPrice) AS ToInteger
FROM Products
```



8. [4] Write a SELECT statement that returns these columns from the Orders table:

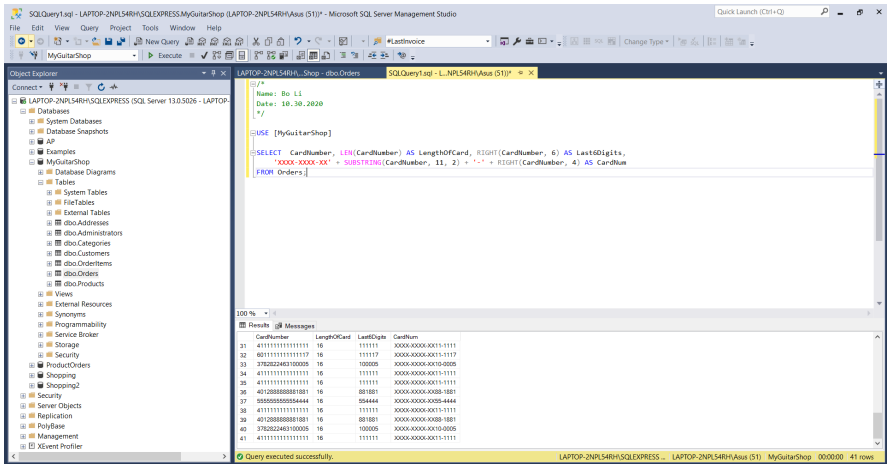
   a) The CardNumber column

   b) The length of the CardNumber column

   c) The last six digits of the CardNumber column

   d) A column that displays the last four digits of the CardNumber column in this format: **XXXX-XXXX-XX12-3456**. In other words, use X's for the first 10 digits of the card number and actual numbers for the last six digits of the number and include dash symbols as specified in the format.

```
USE [MyGuitarShop]

SELECT  CardNumber, LEN(CardNumber) AS LengthOfCard, RIGHT(CardNumber, 6)
AS Last6Digits,
       'XXXX-XXXX-XX' + SUBSTRING(CardNumber, 11, 2) + '-' +
RIGHT(CardNumber, 4) AS CardNum
FROM Orders;
```



9. [4] Write a SELECT statement that returns these columns from the Orders table:

   a) The OrderID column

   b) The OrderDate column

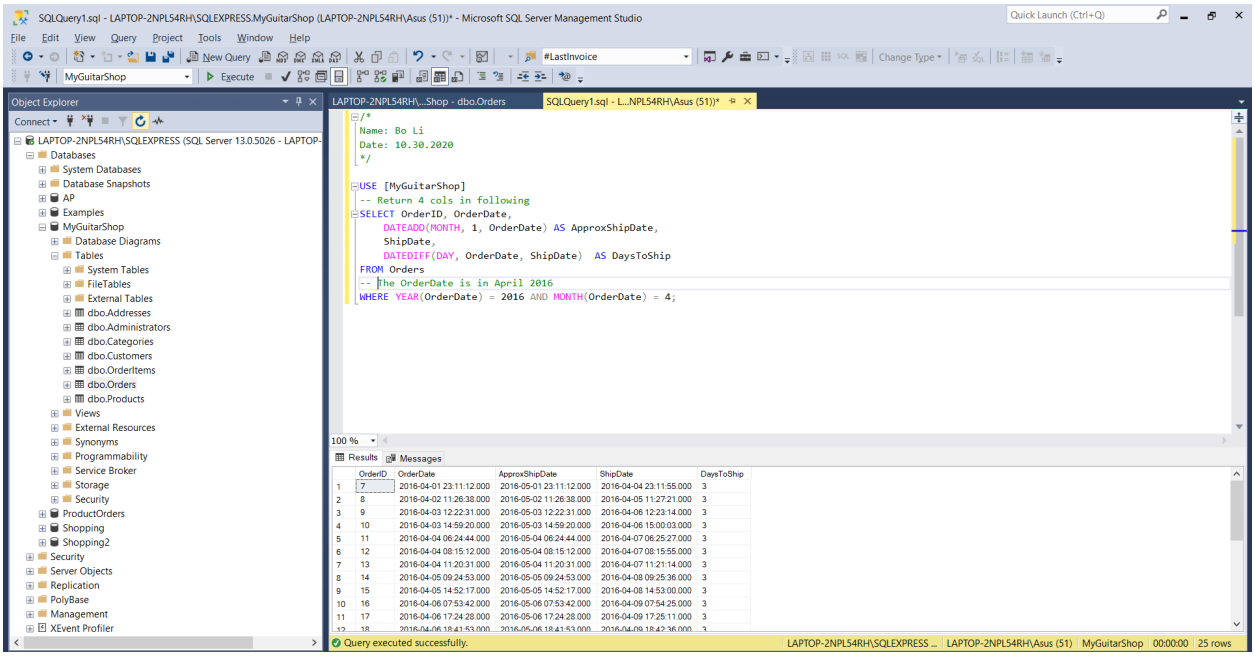
   c) A column named ApproxShipDate that's calculated by adding 1 month to the OrderDate column

   d) The ShipDate column

   e) A column named DaysToShip that shows the number of days between the order date and the ship date

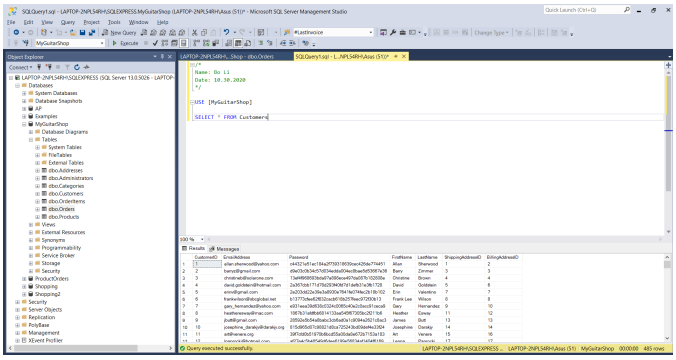
   When you have this working, add a WHERE clause that retrieves just the orders for April 2016.

```
USE [MyGuitarShop]
-- Return 4 cols in following
SELECT OrderID, OrderDate,
    DATEADD(MONTH, 1, OrderDate) AS ApproxShipDate,
    ShipDate,
    DATEDIFF(DAY, OrderDate, ShipDate) AS DaysToShip
FROM Orders
-- The OrderDate is in April 2016
WHERE YEAR(OrderDate) = 2016 AND MONTH(OrderDate) = 4;
```



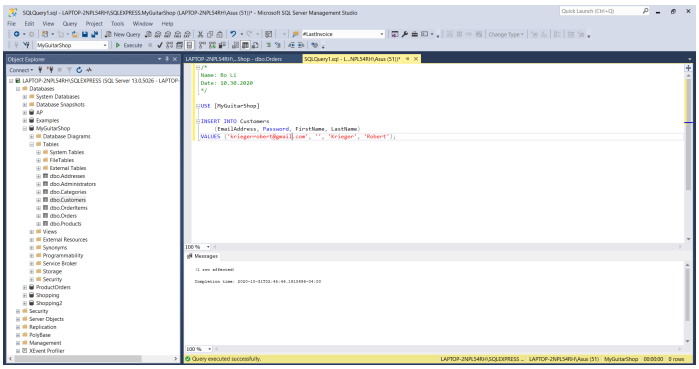
10. [4] Write an INSERT statement that adds this row to the Customers table: EmailAddress: kriegerrobert@gmail.com Password: (empty string) FirstName: Krieger LastName: Robert Use a column list for this statement.

Before:

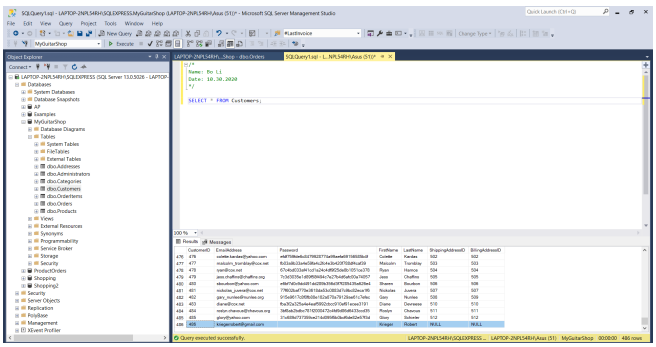


Insert:

```
USE [MyGuitarShop]

INSERT INTO Customers
    (EmailAddress, Password, FirstName, LastName)
VALUES ('kriegerrobert@gmail.com', '', 'Krieger', 'Robert');
```

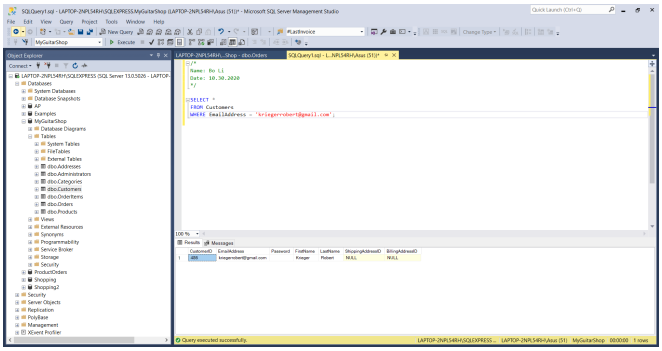


After:



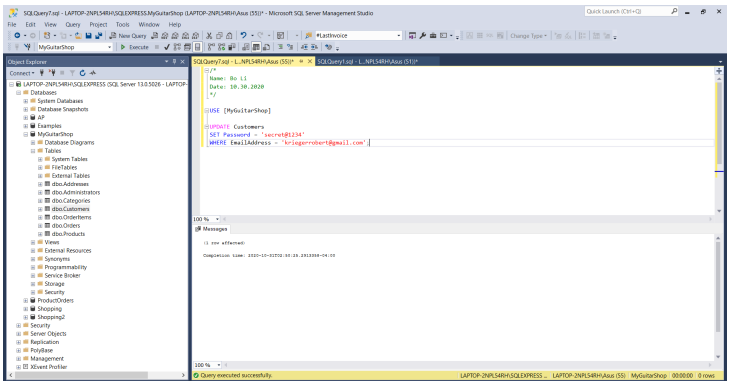
11. [4] Write an UPDATE statement that modifies the Customers table. Change the password column to “secret@1234” for the customer with an email address: kriegerrobert@gmail.com.

Before:

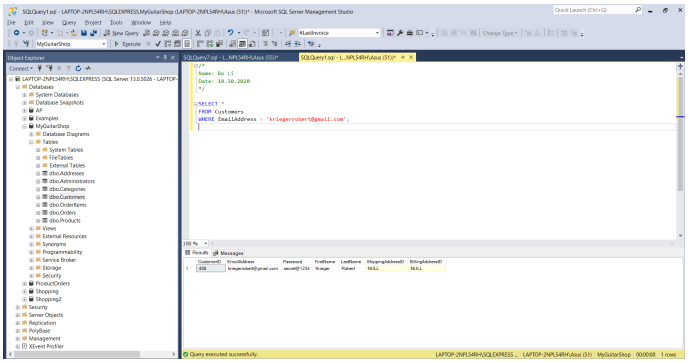


Update:

```
USE [MyGuitarShop]

UPDATE Customers
SET Password = 'secret@1234'
WHERE EmailAddress = 'kriegerrobert@gmail.com';
```



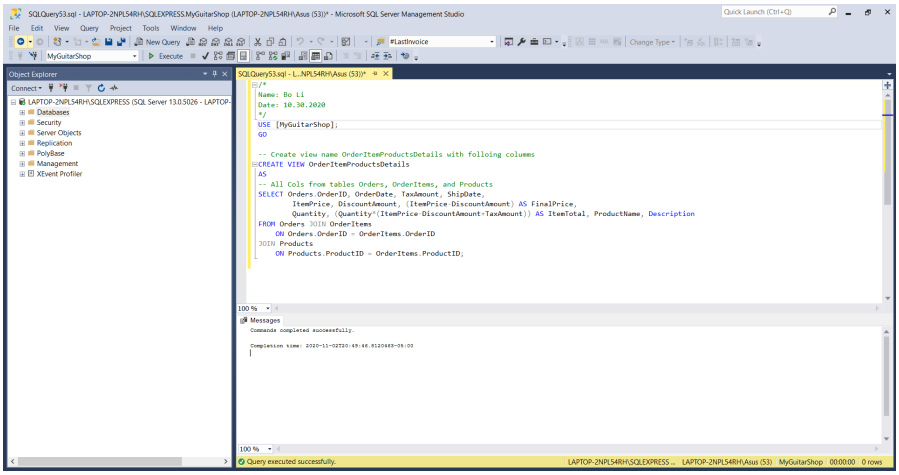
After:



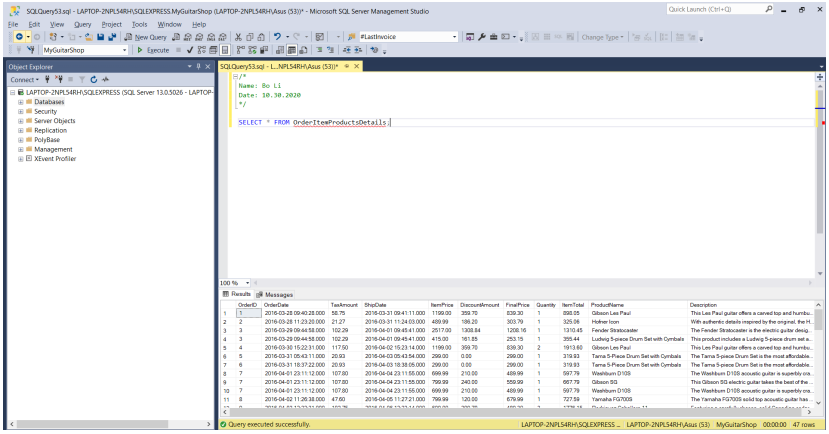
D. Advanced SQL skills

1. [4] Create a view named OrderItemProductsDetails that returns columns from the Orders, OrderItems, and Products tables. a) This view should return these columns from the Orders table: OrderID, OrderDate, TaxAmount, and ShipDate. b) This view should return these columns from the OrderItems table: ItemPrice, DiscountAmount, FinalPrice (the discount amount subtracted from the item price), Quantity, and ItemTotal (the calculated total for the item). c) This view should return the ProductName and Description column from the Products table.

```
USE [MyGuitarShop];
GO
-- Create view name OrderItemProductsDetails with folloing columms
CREATE VIEW OrderItemProductsDetails
AS
-- All Cols from tables Orders, OrderItems, and Products
SELECT Orders.OrderID, OrderDate, TaxAmount, ShipDate,
          ItemPrice, DiscountAmount, (ItemPrice-DiscountAmount) AS FinalPrice,
          Quantity, (Quantity*(ItemPrice-DiscountAmount+TaxAmount)) AS
ItemTotal, ProductName, Description
FROM Orders JOIN OrderItems
    ON Orders.OrderID = OrderItems.OrderID
JOIN Products
    ON Products.ProductID = OrderItems.ProductID;
```

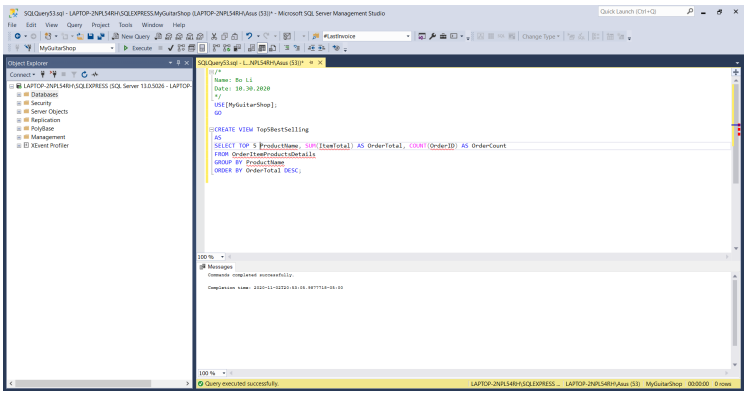


Verification:

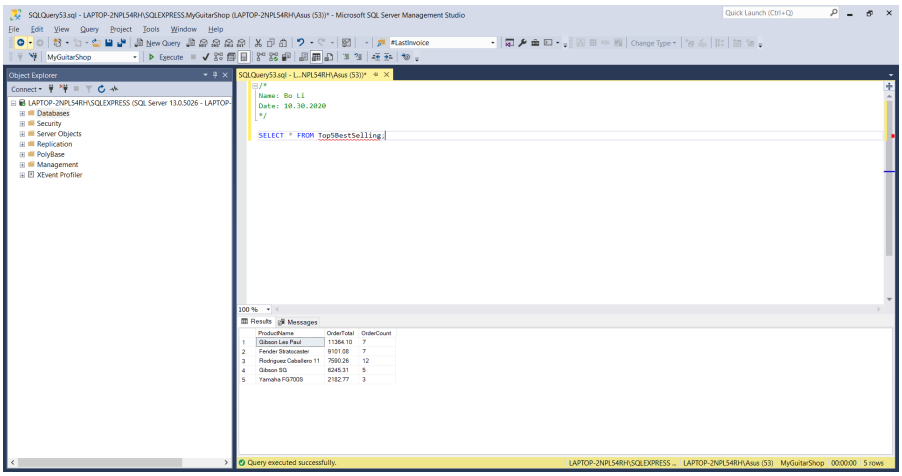


2. [5] Create a view named Top5BestSelling that uses the view you created in Section D Question 1. This view should return some summary information about five best selling products. Each row should include these columns: ProductName, OrderTotal (the total sales for the product) and OrderCount (the number of times the product has been ordered).

```
USE[MyGuitarShop];
GO
CREATE VIEW Top5BestSelling
AS
SELECT TOP 5 ProductName, SUM(ItemTotal) AS OrderTotal, COUNT(OrderID) AS
OrderCount
FROM OrderItemProductsDetails
GROUP BY ProductName
ORDER BY OrderTotal DESC;
```

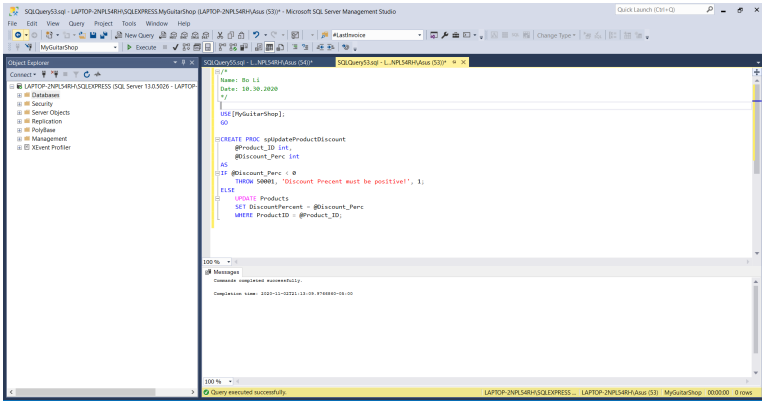


Verification:

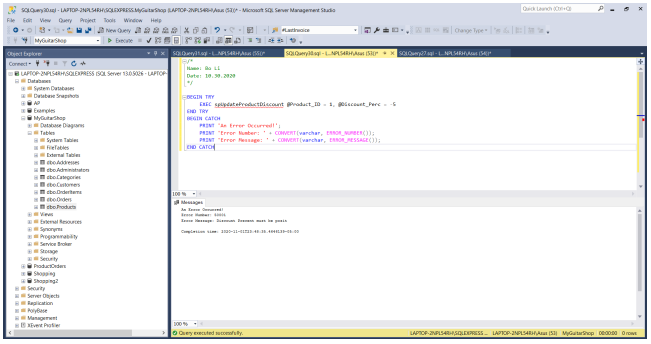


3. [5] Write a script that creates and calls a stored procedure named spUpdateProductDiscount that updates the DiscountPercent column in the Products table. This procedure should have one parameter for the product ID and another for the discount percent. If the value for the DiscountPercent column is a negative number, the stored procedure should raise an error that indicates that the value for this column must be a positive number. Code at least two EXEC statements that test this procedure.

```
USE[MyGuitarShop];
GO

CREATE PROC spUpdateProductDiscount
    @Product_ID int,
    @Discount_Perc int
AS
IF @Discount_Perc < 0
    THROW 50001, 'Discount Precent must be positive!', 1;
ELSE
    UPDATE Products
    SET DiscountPercent = @Discount_Perc
    WHERE ProductID = @Product_ID;
```

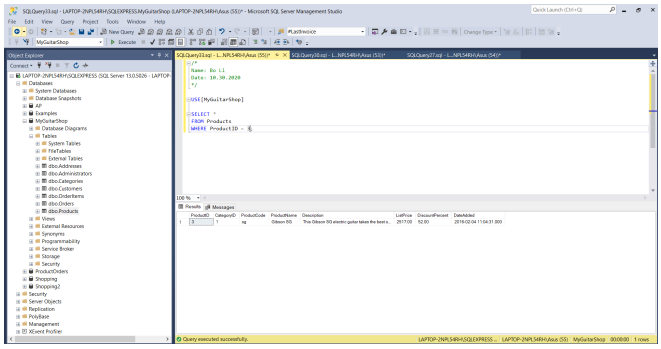


Test 1 (With Error):

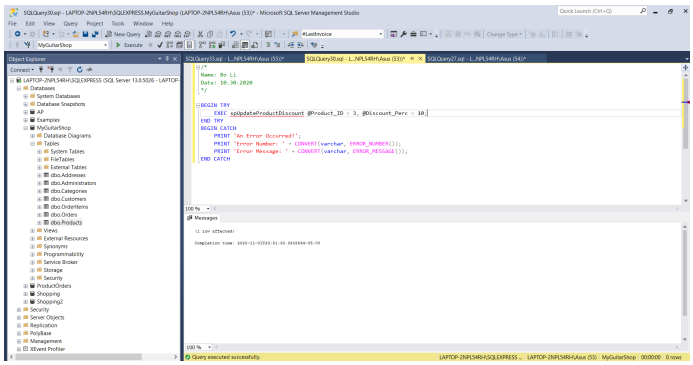


Test 2: (No Error)

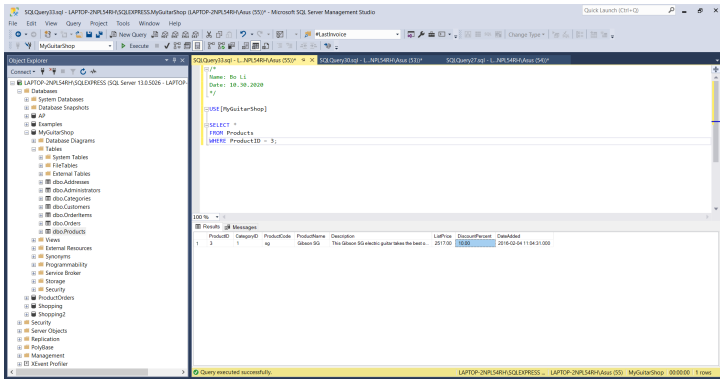
Before:



EXEC:



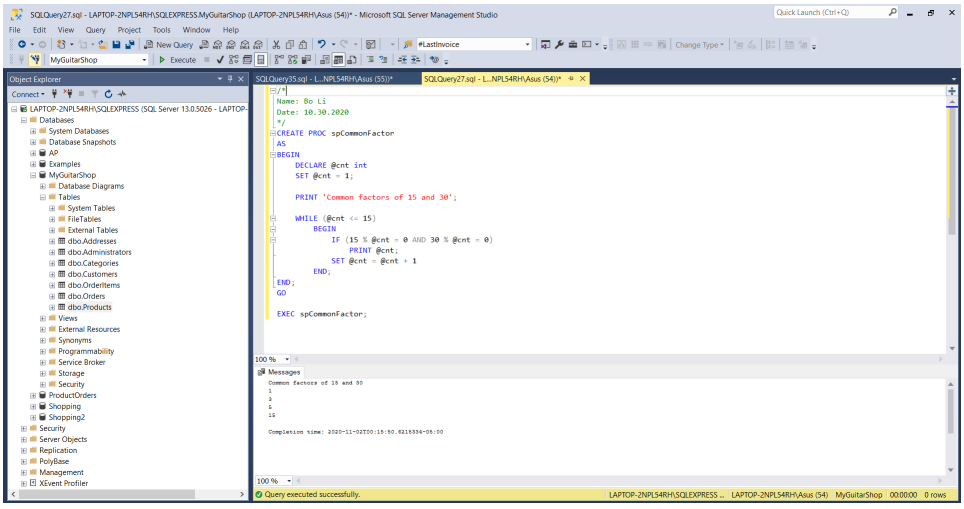
AFTER:



4. [5] Write a script that calculates the common factors between 15 and 30. To find a common factor, you can use the modulo operator (%) to check whether a number can be evenly divided into both numbers. Then, this script should print lines that display the common factors like this:

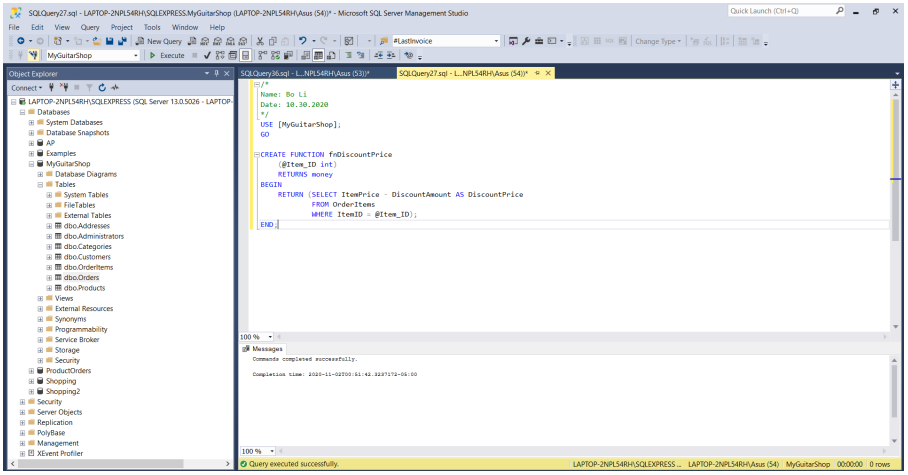
```
Common factors of 15 and 30
1
3
5
15
```

```
CREATE PROC spCommonFactor
AS
BEGIN
	DECLARE @cnt int
	SET @cnt = 1;

	PRINT 'Common factors of 15 and 30';

	WHILE (@cnt <= 15)
		BEGIN
			IF (15 % @cnt = 0 AND 30 % @cnt = 0)
				PRINT @cnt;
		SET @cnt = @cnt + 1
	END;
			END;
GO

EXEC spCommonFactor;
```



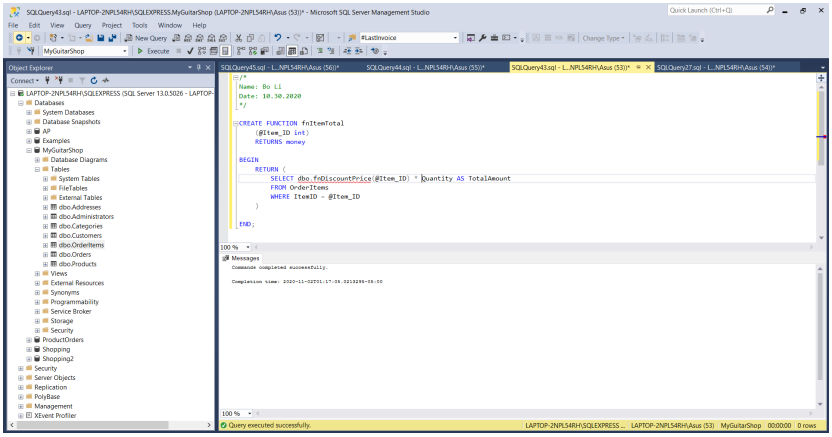
5. [5] (1) Write a script that creates and calls a function named fnDiscountPrice that calculates the discount price of an item in the OrderItems table (discount amount subtracted from item price). To do that, this function should accept one parameter for the item ID, and it should return the value of the discount price for that item.

```
USE [MyGuitarShop];
GO

CREATE FUNCTION fnDiscountPrice
    (@Item_ID int)
    RETURNS money
BEGIN
    RETURN (SELECT ItemPrice - DiscountAmount AS DiscountPrice
            FROM OrderItems
            WHERE ItemID = @Item_ID);
END;
```



(2) Write a script that creates and calls a function named fnItemTotal that calculates the total amount of an item in the OrderItems table (discount price multiplied by quantity). To do that, this function should accept one parameter for the item ID, it should use the DiscountPrice function that you created in (1), and it should return the value of the total for that item.

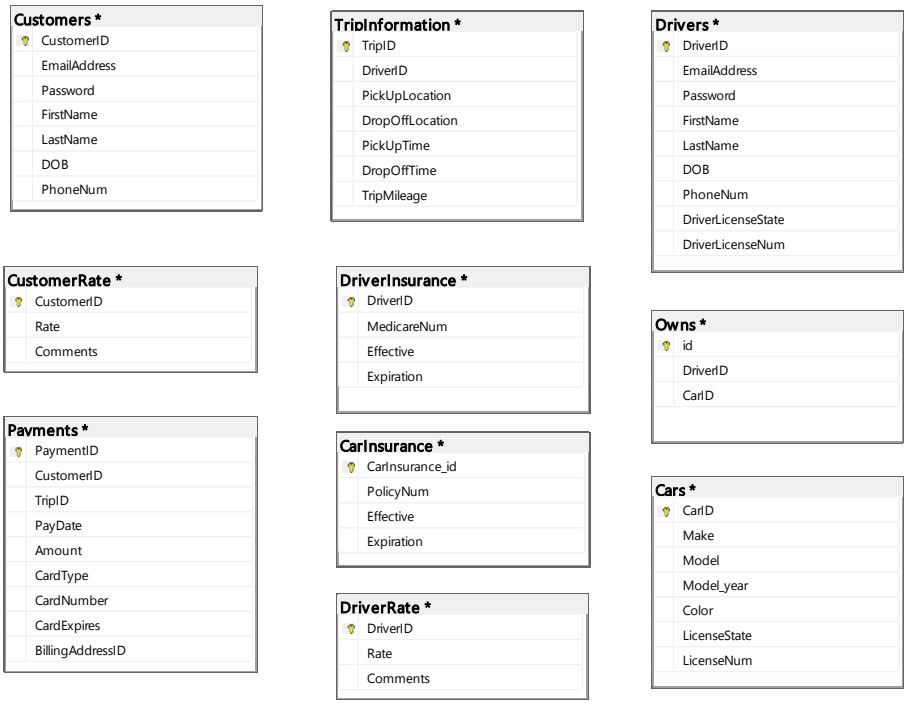
```
CREATE FUNCTION fnItemTotal
    (@Item_ID int)
    RETURNS money

BEGIN
    RETURN (
            SELECT dbo.fnDiscountPrice(@Item_ID) * Quantity AS TotalAmount
            FROM OrderItems
            WHERE ItemID = @Item_ID
    )

END;
```


SQLQuery43.sql - LAPTOP-2NPL54RH\SQLEXPRESS.MyGuitarShop (LAPTOP-2NPL54RH\Asus (53))* - Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
File Edit View Query Project Tools Window Help
New Query
#LastInvoice
Change Type
MyGuitarShop
Execute
Object Explorer
Connect
LAPTOP-2NPL54RH\SQLEXPRESS (SQL Server 13.0.5026 - LAPTOP-
Databases
System Databases
Database Snapshots
AP
Examples
MyGuitarShop
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Addresses
dbo.Administrators
dbo.Categories
dbo.Customers
dbo.OrderItems
dbo.Orders
dbo.Products
Views
External Resources
Synonyms
Programmability
Service Broker
Storage
Security
ProductOrders
Shopping
Shopping2
Security
Server Objects
Replication
PolyBase
Management
XEvent Profiler
SQLQuery45.sql - L...NPL54RH\Asus (56))*
SQLQuery44.sql - L...NPL54RH\Asus (55))*
SQLQuery43.sql - L...NPL54RH\Asus (53))*
SQLQuery27.sql - L...NPL54RH\Asus (54))*
/*
Name: Bo Li
Date: 10.30.2020
*/
CREATE FUNCTION fnItemTotal
(@Item_ID int)
RETURNS money
BEGIN
RETURN (
SELECT dbo.fnDiscountPrice(@Item_ID) * Quantity AS TotalAmount
FROM OrderItems
WHERE ItemID = @Item_ID
)
END;
100 %
Messages
Commands completed successfully.
Completion time: 2020-11-02T01:17:09.3213298-08:00
100 %
Query executed successfully.
LAPTOP-2NPL54RH\SQLEXPRESS ...
LAPTOP-2NPL54RH\Asus (53)
MyGuitarShop
00:00:00
0 rows

II. Database Design

Create a sample database design for Online Cab System and draw one database model for it. It needs to keep track of Customer information, Driver information, login details of both customer and driver, Trip information, Payment details used by customers to pay for a trip, Car details, Rating information given for driver by customer and for customer given by driver and Insurance information(for both car and drivers if applicable). Your design could add more things to the existing requirements, but all the given requirements should be met.

1. **[3] Design the database that makes sense for the problem and select fields that make the most sense. A complete screenshot of your final design model is required.**



2. **[10] Determine the tables, columns, primary keys, nullabilities and show relationships between tables (one -one/one-many/many-many).**

Table	PK	FK
Customers	CustomerID	
Drivers	DriverID	
CustomerRate	CustomerID	CustomerID
DriverRate	DriverID	DriverID
TripInformation	TripID	DriverID
Payments	PaymentID	CustomerID, TripID
Cars	CarID	
Owns	id	DriverID, CarID
DriverInsurance	DriverID	DriverID
CarInsurance	CarInsurance_id	CarInsurance_id

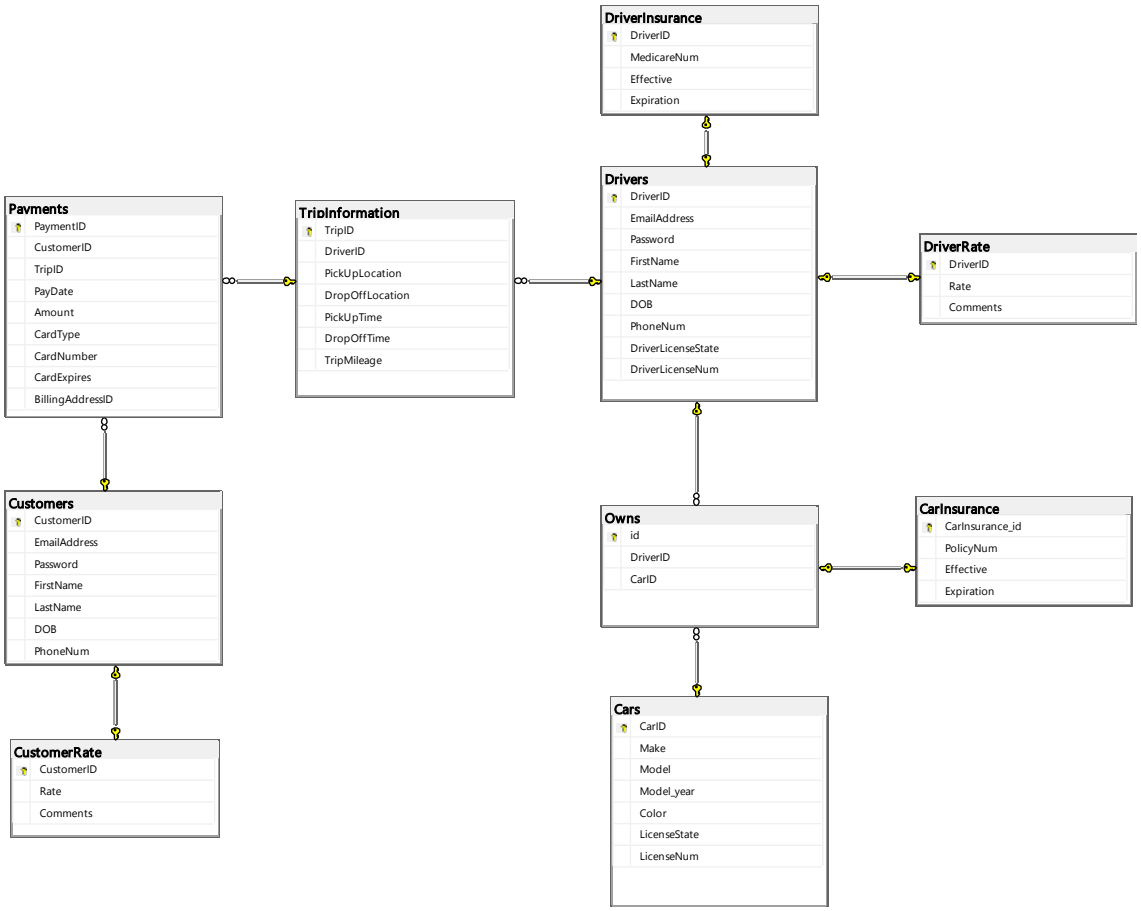
Nullabilities:

Table	Nullabilities Cols
CustomerRate	Comments
DriverRate	Comments

Relationships:

1) Customers - CustomerRate: One-One
2) Drivers – DriverRate: One-One
3) Drivers – DriverInsurance: One – One
4) Customers – Payments: One-Many
5) TripInformation – Payments: One – Many
6) Customers – TripInformation: Many – Many
7) Drivers – TripInformation: One – Many
8) Drivers - Cars: Many – Many
9) Owns – CarInsurance: One – One

3. [5] Normalize your design into 3rd Normal Form. Please use MS Visio, Vertabelo or any similar database design tool.



4. **[2] Explain your design, including relationship between tables.**

In this design, there are some main tables I need to consider creating first, such as Customers, Drivers, TripInformation, Cars, Payments. These tables are the keys of the database to be connected. Secondly Some minor tables, such as rate, insurance, need to be relevant to the major tables using FK -> PK. I also create a table named “Owns”, meaning that driver owns cars, which is relationship between drivers and cars. The following are the relationships among tables in the database.

1) **Customers - CustomerRate**: Each Customer has one overall rating statement, so that other users can view it clearly.
2) **Drivers – DriverRate**: Each Driver has one overall rating statement, so that other users can view it clearly.
3) **Drivers – DriverInsurance**: Each driver has only one insurance for their safety (Alike health insurance). Each insurance only can cover one driver.
4) **Customers – Payments**: Each Customer can make more payments for their orders.
5) **TripInformation – Payments**: Each Trip can be split by one or more payments. E.g. More customers share one trip payment.
6) **Customers – TripInformation**: A Customer can have more trips. More customers can share a trip together.
7) **Drivers – TripInformation**: A driver can have more trips by accepting orders.
8) **Drivers - Cars**: Each Driver can own one or more cars. Each car can be driven by one or more drivers. E.g. the car is sold to other drivers
9) **Owns – CarInsurance**: Each car insurance is only covered by unique car and driver. E.g. Other drivers use this car should put their names on the insurance.

Remark:

In this project, I gained a knowledge with much more deeper understanding of Database system and SQL server. In MyGuitarShop Database, I enhance my skills such as essential SQL skill, Advanced SQL Skill. Based on the exist database, I know how to retrieve data what I want by SELECT. How to create View, Procedures and functions in database. How to use conditions such as If statement in SQL as well. Also, I learned how to design and create a database with MS Visio.