

Introduction to Database Management Systems

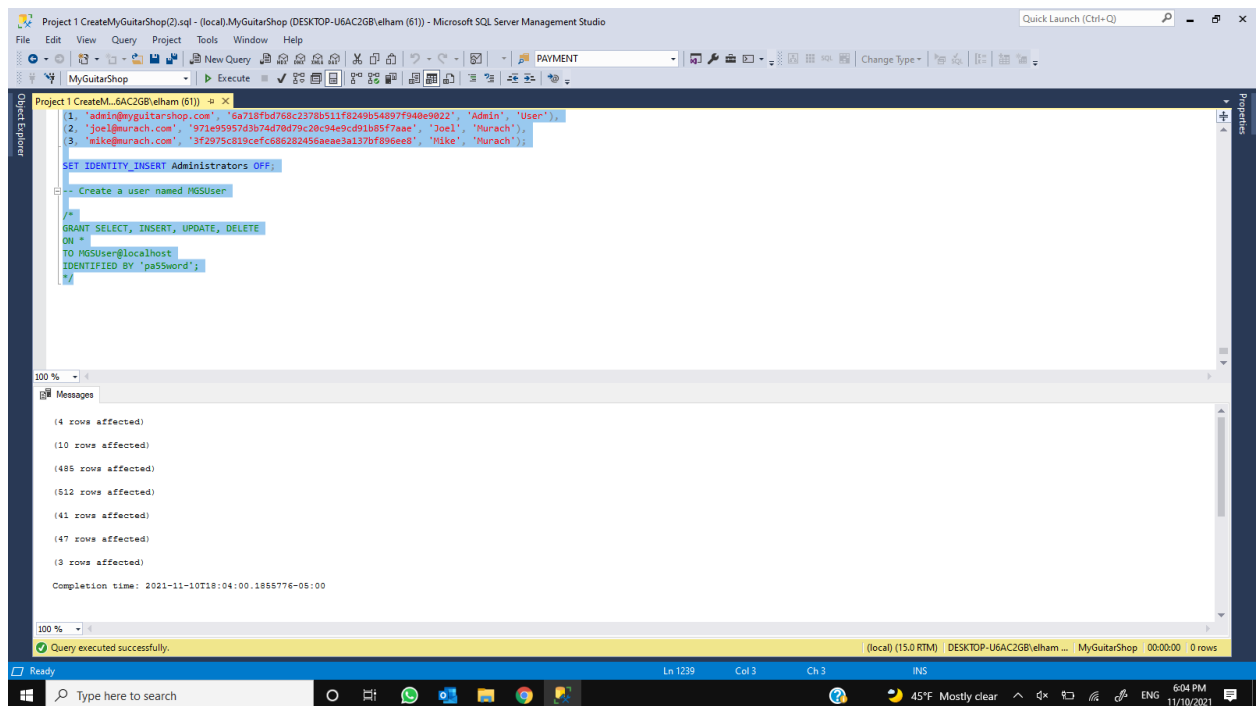
PROJECT 1

Seyed Alireza Zarrin Mehr

I. My Guitar Shop Database In part I, you will use SQL Server Management Studio to create the MyGuitarShop database, to review the tables in the MyGuitarShop database, and to enter SQL statements and run them against this database.

A. Database Setup [2 pts.] 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



Remark: After downloading and running the scripts database is all set.

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

USE MyGuitarShop
GO
SELECT * FROM Orders

Project 1 CreateMyGuitarShop(2).sql - (local)MyGuitarShop (DESKTOP-U6AC2GB\elham (55)) - Microsoft SQL Server Management Studio

FileEditViewQueryProjectToolsWindowHelp

Quick Launch (Ctrl+Q)

MyGuitarShop

Execute

Object Explorer

Connect -

SQL Server 15.0.2080.9 - DESKTOP-U6AC2GB\elham (55)

Databases

System Databases

Database Snapshots

AP

Examples

MyGuitarShop

Database Diagrams

Tables

System Tables

FileTables

External Tables

Graph Tables

dbo.Addresses

dbo.AccountInformation

dbo.Categories

dbo.Customers

dbo.OrderItems

dbo.Orders

dbo.Products

Views

External Resources

Synonyms

Programmability

Service Broker

Storage

Security

ProductOrders

Security

Server Objects

Replication

PolyBase

Always On High Availability

Management

Integration Services Catalogs

SQL Server Agent

Project 1 CreateMyGuitarShop (DESKTOP-U6AC2GB\elham (55))

USE MyGuitarShop

GO

SELECT * FROM Orders

100 %

Results

Messages

OrderID	CustomerID	OrderDate	ShipAmount	TaxAmount	ShipDate	ShipAddressID	CardType	CardNumber	CardExpires	BillingAddressID	
23	23	20	2016-04-14 07:59:31.000	5.00	34.25	2016-04-17 08:00:14.000	27	Visa	4012888888881881	03/2019	27
24	24	21	2016-04-17 17:40:22.000	5.00	34.25	2016-04-20 17:41:05.000	28	Visa	4111111111111111	04/2018	28
25	25	22	2016-04-20 08:23:32.000	10.00	117.50	2016-04-23 08:24:15.000	29	Visa	4111111111111111	09/2018	29
26	26	23	2016-04-20 08:14:45.000	5.00	0.00	2016-04-23 08:15:28.000	30	American Express	3782822463100005	08/2017	30
27	27	24	2016-04-20 09:17:52.000	5.00	84.57	2016-04-23 09:18:35.000	31	Visa	4111111111111111	02/2018	31
28	28	25	2016-04-21 17:52:34.000	5.00	34.30	2016-04-24 17:53:07.000	32	Visa	4111111111111111	08/2019	32
29	29	4	2016-04-25 23:36:41.000	25.00	196.00	2016-04-28 23:37:24.000	5	Visa	4012888888881881	03/2017	6
30	30	26	2016-04-27 16:21:31.000	5.00	26.25	2016-04-30 16:22:14.000	33	Visa	4111111111111111	02/2018	33
31	31	27	2016-04-29 06:47:14.000	10.00	118.82	2016-05-02 06:47:57.000	34	Visa	4111111111111111	01/2018	34
32	32	18	2016-05-01 01:23:23.000	5.00	84.57	NULL	24	Discover	6011111111111111	02/2018	24
33	33	28	2016-05-01 09:11:51.000	10.00	41.86	2016-05-04 09:12:34.000	35	American Express	3782822463100005	04/2017	35
34	34	29	2016-05-02 11:36:12.000	5.00	58.75	2016-05-05 11:36:55.000	36	Visa	4111111111111111	06/2017	37
35	35	30	2016-05-04 03:52:23.000	5.00	39.20	2016-05-07 03:53:06.000	38	Visa	4111111111111111	09/2018	38
36	36	31	2016-05-04 12:31:33.000	5.00	21.27	2016-05-07 12:32:16.000	39	Visa	4012888888881881	11/2018	39
37	37	32	2016-05-06 14:15:21.000	5.00	84.57	2016-05-09 14:16:04.000	40	MasterCard	5555555555554444	02/2017	41
38	38	33	2016-05-08 11:41:24.000	10.00	117.50	NULL	42	Visa	4111111111111111	04/2018	43
39	39	29	2016-05-08 22:22:26.000	5.00	0.00	NULL	36	Visa	4012888888881881	01/2018	37
40	40	34	2016-05-08 21:41:28.000	5.00	34.25	NULL	44	American Express	3782822463100005	08/2017	44
41	41	35	2016-05-09 07:52:55.000	10.00	55.52	NULL	45	Visa	4111111111111111	05/2018	45

Query executed successfully.

(local) (15.0 RTM) DESKTOP-U6AC2GB\elham ... MyGuitarShop | 00:00:00 | 41 rows

ReadyLn4Col1Ch1INS

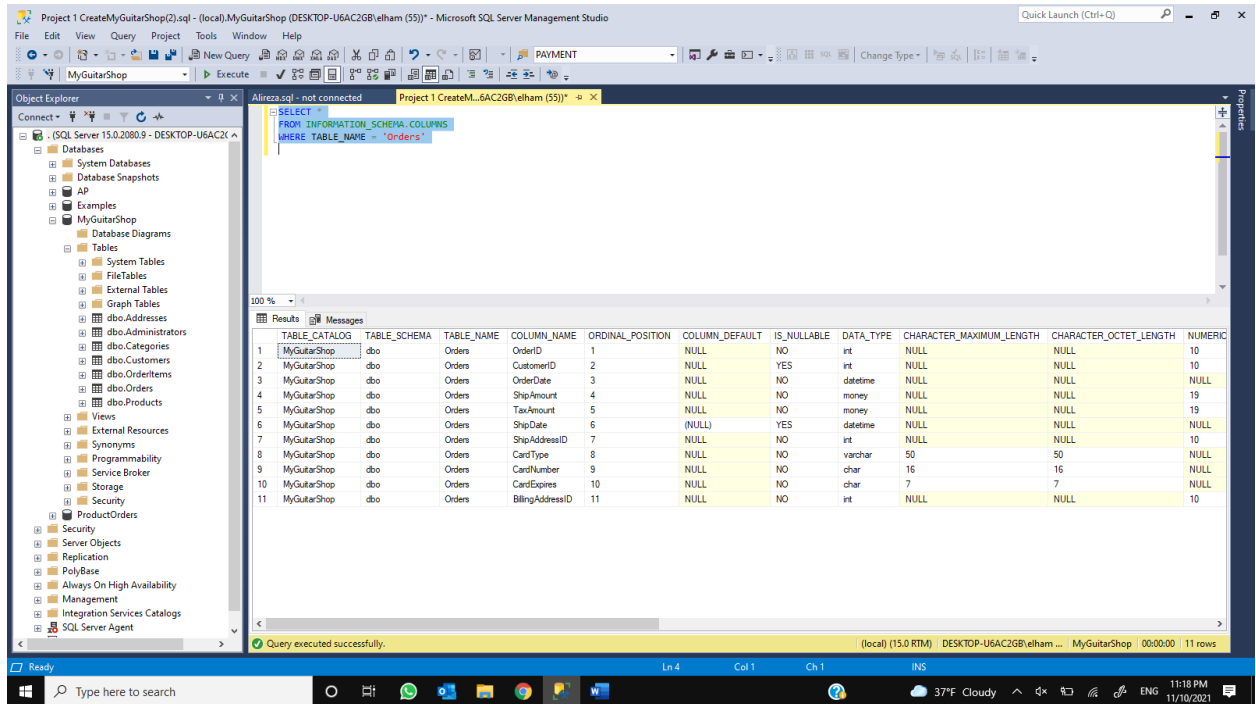
Type here to search

37°F Cloudy

11:07 PM11/10/2021

Remark: Here we can see the content of the Orders table.

```
SELECT *  
FROM INFORMATION_SCHEMA.COLUMNS  
WHERE TABLE_NAME = 'Orders'
```



	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUMN_DEFAULT	IS_NULLABLE	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH	CHARACTER_OCTET_LENGTH	NUMERIC
1	MyGuitarShop	dbo	Orders	OrderID	1	NULL	NO	int	NULL	NULL	10
2	MyGuitarShop	dbo	Orders	CustomerID	2	NULL	YES	int	NULL	NULL	10
3	MyGuitarShop	dbo	Orders	OrderDate	3	NULL	NO	datetime	NULL	NULL	NULL
4	MyGuitarShop	dbo	Orders	ShipAmount	4	NULL	NO	money	NULL	NULL	19
5	MyGuitarShop	dbo	Orders	TaxAmount	5	NULL	NO	money	NULL	NULL	19
6	MyGuitarShop	dbo	Orders	ShipDate	6	(NULL)	YES	datetime	NULL	NULL	NULL
7	MyGuitarShop	dbo	Orders	ShipAddressID	7	NULL	NO	int	NULL	NULL	10
8	MyGuitarShop	dbo	Orders	CardType	8	NULL	NO	varchar	50	50	NULL
9	MyGuitarShop	dbo	Orders	CardNumber	9	NULL	NO	char	16	16	NULL
10	MyGuitarShop	dbo	Orders	CardExpires	10	NULL	NO	char	7	7	NULL
11	MyGuitarShop	dbo	Orders	BillingAddressID	11	NULL	NO	int	NULL	NULL	10

Remark: Here we can see the details for the Orders table.

USE MyGuitarShop
GO
SELECT * FROM Customers

The screenshot shows the Microsoft SQL Server Enterprise Manager interface. The Query window contains the following SQL script:

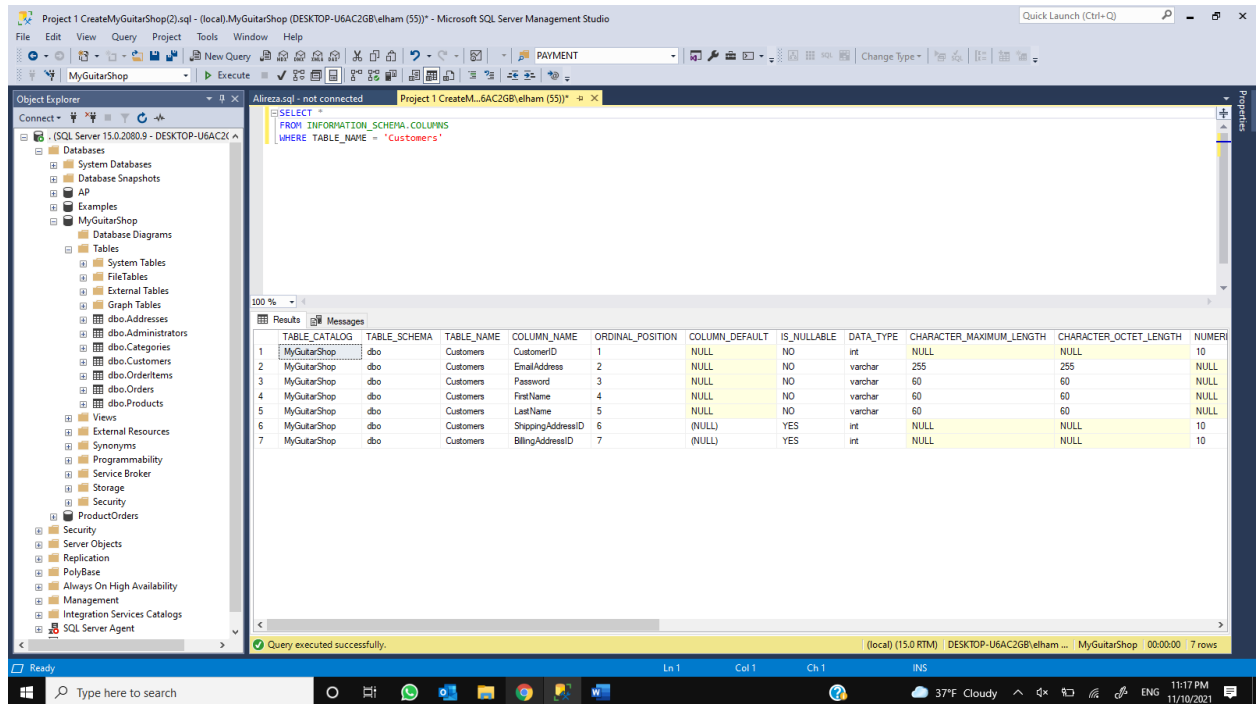
```
USE MyGuitarShop
GO
SELECT * FROM Customers
```

The Results pane displays the following data (partial view):

CustomerID	EmailAddress	Password	FirstName	LastName	ShippingAddressID	BillingAddressID
467	doomnick@cox.net	a120b51cdc9a9314acbc0789280bea3cf5c78762	Daniela	Connick	493	493
468	cecilia_colazzo@colazzo.com	3c08e14b6483027a29c98d225ad6263130bf9e3	Cecilia	Colazzo	494	494
469	leslie@cox.net	cb0025db0a3b134a67892179b62c36af9f64720	Leslie	Threets	495	495
470	nan@koppinger.com	4a430cfd30aef01ae1018125c3d9f09da2035422c	Nan	Koppinger	496	496
471	idevar@devar.com	f81897a9f79531d5154b1c9e1ee0379191465ad	Iveta	Devar	497	497
472	tegan.arco@arceo.org	0771493978785ac2a2ca7594e7b3e3d9b3d6fc4	Tegan	Arceo	498	498
473	ruthann@hotmail.com	01593dcd2489ad3e1cb4596249a4fa6724e44	Ruthann	Keener	499	499
474	jori_breland@cox.net	a786dc303c19b2cc2bf9994942920a0a093d57	Jori	Breland	500	500
475	vrentro@cox.net	d19f8cc1bc1cb6a40339b52bbaed5922b441bd0f	Vi	Rrentro	501	501
476	colette.kardas@yahoo.com	efdf759fde6c0479928774a99aef59156589e4f	Colette	Kardas	502	502
477	malcolm_tromblay@cox.net	fb03a9b33a4e59a4c2f4e3a420788d4caf39	Malcolm	Tromblay	503	503
478	ryan@cox.net	67c4b033af41cd1a24c4df925de9b1051ce378	Ryan	Hamos	504	504
479	jess.chaffins@chaffins.org	7c3d3035e1d89f58f494c7a27b4d6af00a74057	Jess	Chaffins	505	505
480	sbourbon@yahoo.com	e6bf7d0c9d491dd289b396d3787285435e625e4	Sharen	Bourbon	506	506
481	nickolas_juvera@cox.net	778602af770e3618da53c0883d7c9bc02eca1f6	Nickolas	Juvera	507	507
482	gary_nunlee@nunlee.org	915e9617c0f8b88e182a070a79129ae61c7efec	Gary	Nunlee	508	508
483	diane@cox.net	fb432a325a4e4eaf5992bccc910ef91ecce3191	Diane	Devreese	510	510
484	roslyn.chavous@chavous.org	3bf8ab2bdc781f2000472c4d36956d433c0d35	Roslyn	Chavous	511	511
485	glory@yahoo.com	31c689d737359e214d099f9b0bcf6de62e5773d	Glory	Scheler	512	512

Remark: Here we can see the content of the Customers table.

```
SELECT *  
FROM INFORMATION_SCHEMA.COLUMNS  
WHERE TABLE_NAME = 'Customers'
```



The screenshot displays the Microsoft SQL Server Enterprise Manager interface. The left pane shows the Object Explorer with the 'MyGuitarShop' database selected. The right pane shows the 'Query Results' window with the following data:

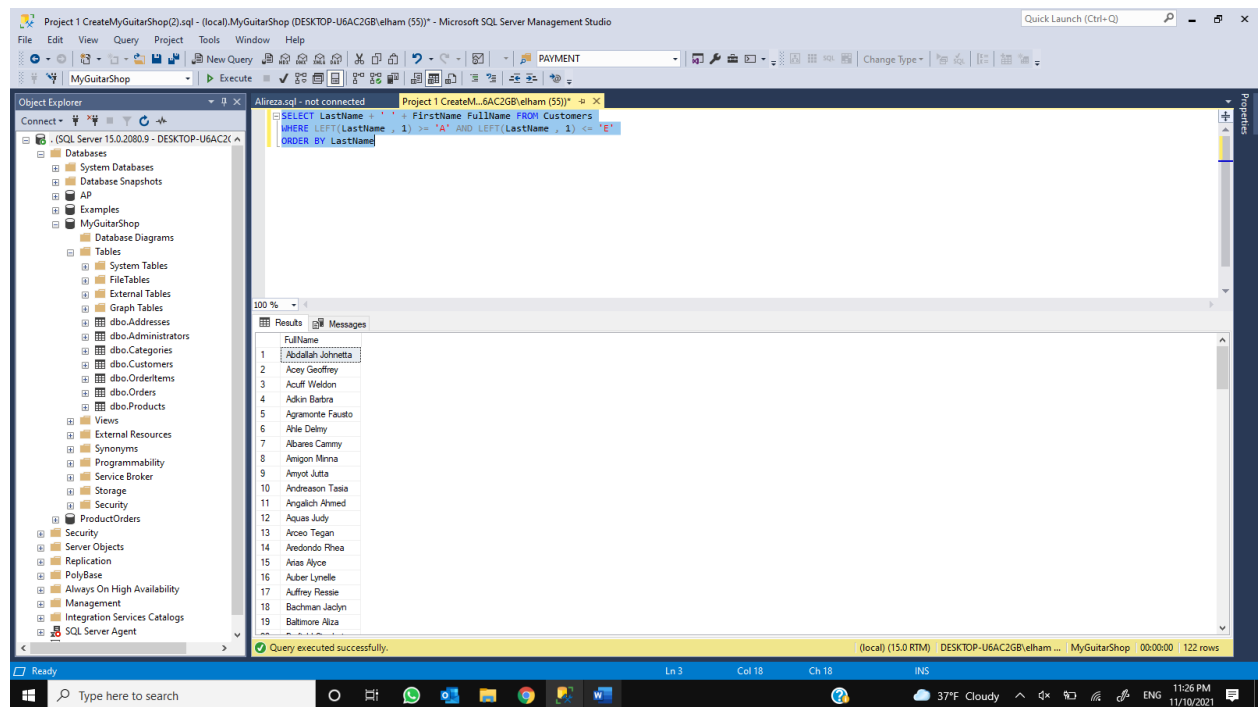
TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUMN_DEFAULT	IS_NULLABLE	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH	CHARACTER_OCTET_LENGTH	NUMERIC_PRECISION
MyGuitarShop	dbo	Customers	CustomerID	1	NULL	NO	int	NULL	NULL	10
MyGuitarShop	dbo	Customers	EmailAddress	2	NULL	NO	varchar	255	255	NULL
MyGuitarShop	dbo	Customers	Password	3	NULL	NO	varchar	60	60	NULL
MyGuitarShop	dbo	Customers	FirstName	4	NULL	NO	varchar	60	60	NULL
MyGuitarShop	dbo	Customers	LastName	5	NULL	NO	varchar	60	60	NULL
MyGuitarShop	dbo	Customers	ShippingAddressID	6	(NULL)	YES	int	NULL	NULL	10
MyGuitarShop	dbo	Customers	BillingAddressID	7	(NULL)	YES	int	NULL	NULL	10

Remark: Here we can see the details for the Customers table.

B. An Introduction to SQL [6 pts.]

1. [3] Write a SELECT statement that returns one column from the Customers table named FullName that joins the LastName and FirstName columns. Format this column with the last name, a space, and then first name like this: Gail Kitty Add an ORDER BY clause to this statement that sorts the result set by last name in ascending sequence. Return only the contacts whose last name begins with a letter from A to E.

```
SELECT LastName + ' ' + FirstName FullName FROM Customers
WHERE LEFT(LastName, 1) >= 'A' AND LEFT(LastName, 1) <= 'E'
ORDER BY LastName
```



Remark: Here we have the full name of the customers started with last name and followed by first name, who's their last name starts with letters A to E. results has been ordered by the last name.

2. [3] Write a SELECT statement that returns these columns from the Orders table:

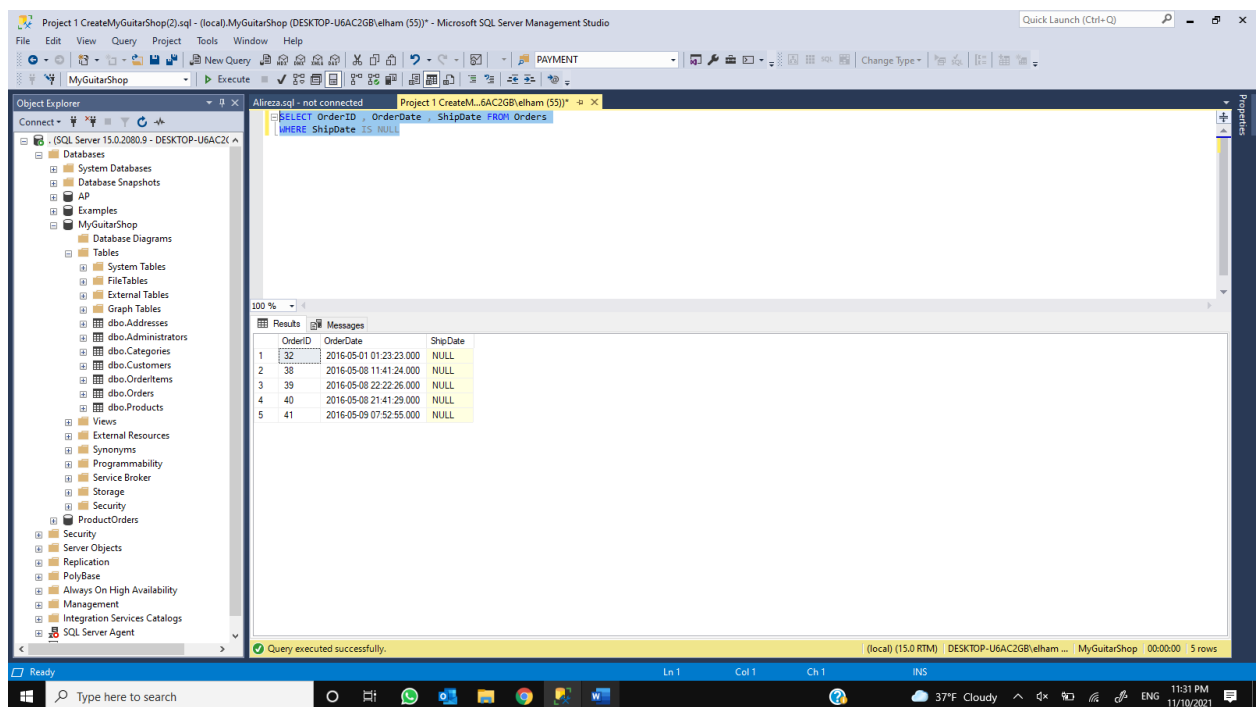
OrderID The OrderID column

OrderDate The OrderDate column

ShipDate The ShipDate column

Return only the rows where the ShipDate column does contains a null value.

```
SELECT OrderID , OrderDate , ShipDate FROM Orders  
WHERE ShipDate IS NULL
```



Remark: Here we can see the orders that has not been shipped or their shipped date has not been entered to the database (shipped date is null).

C. The essential SQL skills [44 pts.]

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

```
SELECT FirstName , LastName , OrderDate , ProductName , ItemPrice , DiscountAmount ,  
Quantity  
FROM Customers c  
JOIN Orders o ON c.CustomerID = o.CustomerID  
JOIN OrderItems i ON i.OrderID = o.OrderID  
JOIN Products p ON i.ProductID = p.ProductID  
ORDER BY LastName DESC , OrderDate
```

The screenshot shows the Microsoft SQL Server Enterprise Manager interface. The query window displays the following SQL statement:

```
SELECT FirstName , LastName , OrderDate , ProductName , ItemPrice , DiscountAmount , Quantity  
FROM Customers c  
JOIN Orders o ON c.CustomerID = o.CustomerID  
JOIN OrderItems i ON i.OrderID = o.OrderID  
JOIN Products p ON i.ProductID = p.ProductID  
ORDER BY LastName DESC , OrderDate
```

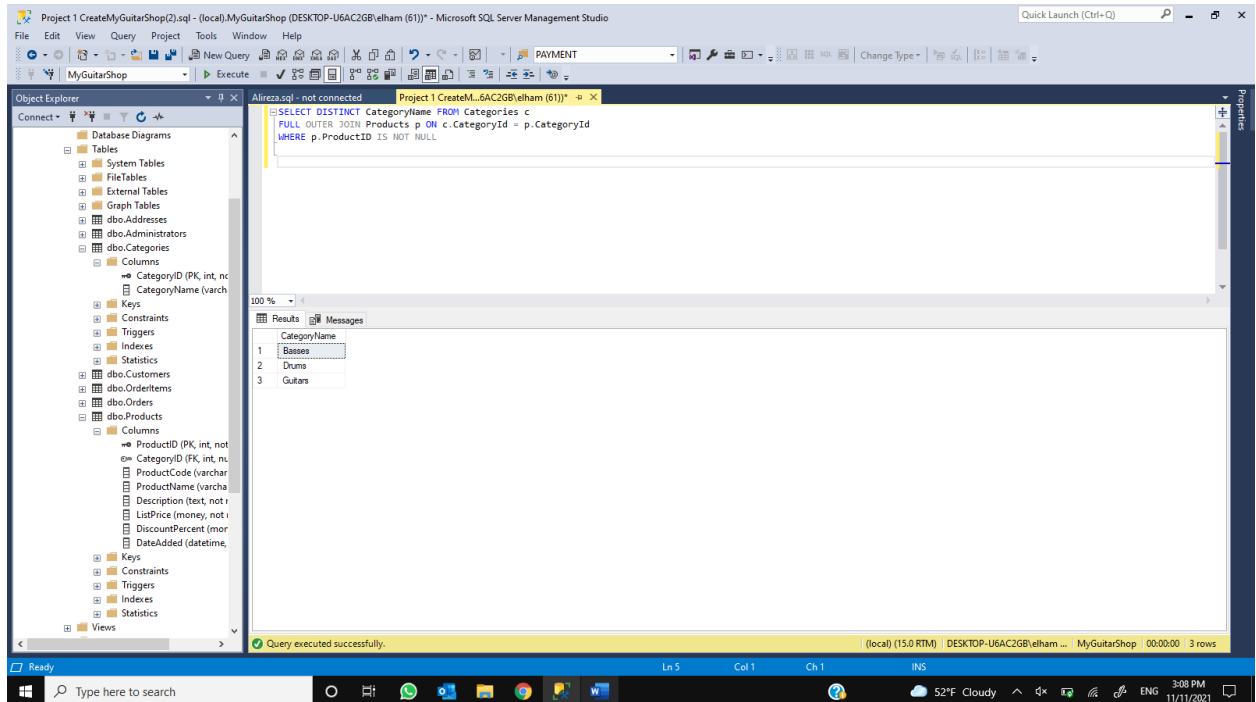
The Results pane shows the following data:

	FirstName	LastName	OrderDate	ProductName	ItemPrice	DiscountAmount	Quantity
1	Bary	Zimmer	2016-03-28 11:23:20.000	Hofner Icon	489.99	186.20	1
2	Frank Lee	Wilson	2016-04-01 23:11:12.000	Washburn D10S	699.99	210.00	1
3	Frank Lee	Wilson	2016-04-01 23:11:12.000	Gibson SG	799.99	240.00	1
4	Frank Lee	Wilson	2016-04-01 23:11:12.000	Washburn D10S	699.99	210.00	1
5	Sage	Wieser	2016-04-08 12:21:31.000	Gibson Les Paul	1199.00	359.70	2
6	Yuki	Whobrey	2016-04-29 06:47:14.000	Fender Stratocaster	2517.00	1308.84	1
7	Yuki	Whobrey	2016-04-29 06:47:14.000	Rodriguez Caballero 11	699.00	209.70	1
8	Art	Venere	2016-04-04 11:20:31.000	Yamaha FG700S	799.99	120.00	1
9	Eri	Valentino	2016-03-31 18:37:22.000	Tama 5-Piece Drum Set with Cymbals	299.00	0.00	1
10	Mitsue	Tolner	2016-04-06 17:24:28.000	Rodriguez Caballero 11	699.00	209.70	1
11	Aisha	Slusanski	2016-05-08 11:41:24.000	Gibson Les Paul	1199.00	359.70	2
12	Allan	Sherwood	2016-03-28 09:40:28.000	Gibson Les Paul	1199.00	359.70	1
13	Allan	Sherwood	2016-03-29 09:44:58.000	Fender Stratocaster	2517.00	1308.84	1
14	Allan	Sherwood	2016-03-29 09:44:58.000	Ludwig 5-Piece Drum Set with Cymbals	415.00	161.85	1
15	Graciela	Ruta	2016-04-20 08:23:32.000	Gibson Les Paul	1199.00	359.70	2
16	Maryann	Roytzer	2016-05-08 14:15:21.000	Fender Stratocaster	2517.00	1308.84	1
17	Gladys	Rim	2016-04-27 16:21:31.000	Fender Precision	499.99	125.00	1
18	Mattie	Poquette	2016-04-20 09:17:52.000	Fender Stratocaster	2517.00	1308.84	1
19	Lenna	Paprocki	2016-04-05 09:24:53.000	Gibson Les Paul	1199.00	359.70	2

Remark: Here we joined tables to be able to get the details of the customers like their first name and last name, the details of the orders like order date, discount, Item price and quantity, and the detail of the products like product name.

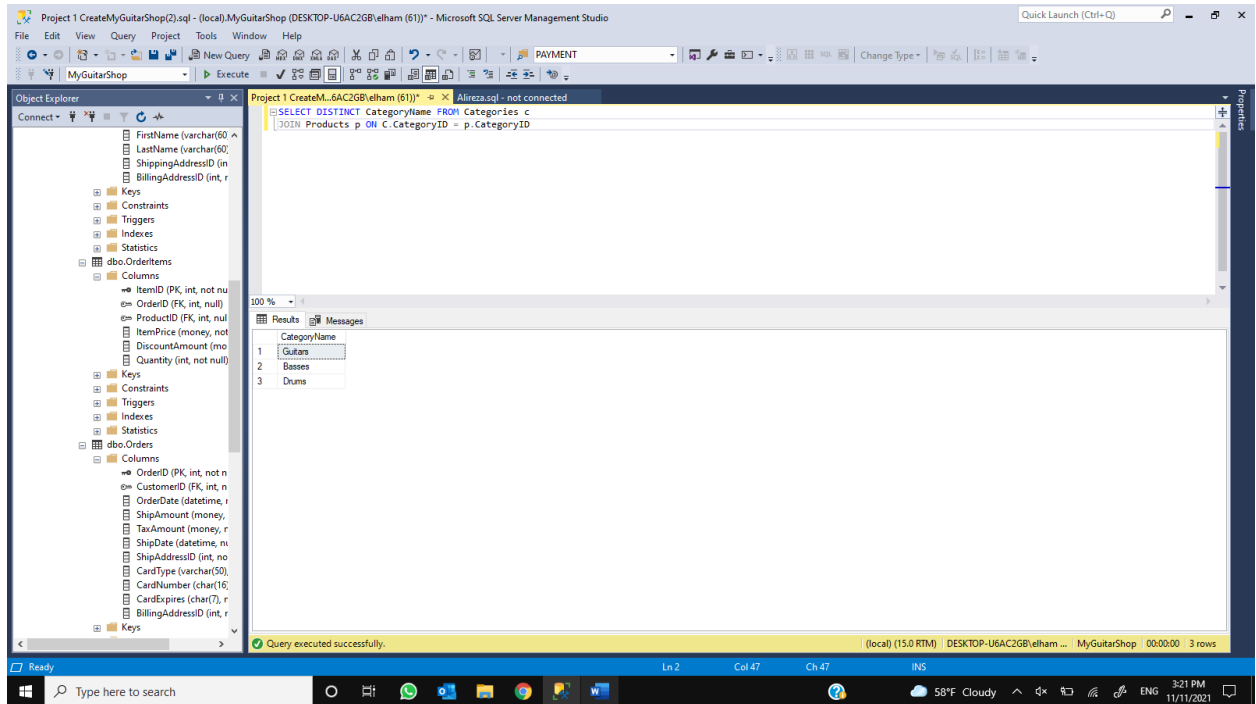
2. [4] Write a SELECT statement that returns CategoryName column from the Categories table. Return one row for each category that has been used. (Hint: Use an outer join and only return rows where the ProductID column does not contain a null value.)

```
SELECT DISTINCT CategoryName FROM Categories c
FULL OUTER JOIN Products p ON c.CategoryId = p.CategoryId
WHERE p.ProductID IS NOT NULL
```



or we can simply inner join two tables for non-null values of CategoryID in products

```
SELECT DISTINCT CategoryName FROM Categories c  
JOIN Products p ON C.CategoryID = p.CategoryID
```



Remark: Here we can see all the categories that has been used.

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 sum of the DiscountAmount column in the OrderItems table multiplied by the quantity in the OrderItems table. Sort the result set in ascending sequence by the item price sum (i.e. Second column [b]) for each customer.

```
SELECT c.EmailAddress , SUM(ItemPrice * Quantity) TotalPrice , SUM(DiscountAmount *
Quantity) TotalDiscount
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN OrderItems i ON o.OrderID = i.OrderID
GROUP BY C.EmailAddress
ORDER BY TotalPrice
```

The screenshot shows the Microsoft SQL Server Management Studio interface. The query editor contains the following SQL statement:

```
SELECT c.EmailAddress , SUM(ItemPrice * Quantity) TotalPrice , SUM(DiscountAmount * Quantity) TotalDiscount
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN OrderItems i ON o.OrderID = i.OrderID
GROUP BY C.EmailAddress
ORDER BY TotalPrice
```

The Results pane displays the output of the query, showing a list of customers with their email addresses, total prices, and total discounts. The data is sorted by TotalPrice in ascending order.

EmailAddress	TotalPrice	TotalDiscount
etrv@gmail.com	299.00	0.00
baryz@gmail.com	489.99	186.20
willard@hotmail.com	489.99	186.20
gladys.rm@rim.org	499.99	125.00
fletcher.flooi@yahoo.com	598.00	0.00
allene_jurbide@cox.net	699.00	209.70
amacleod@gmail.com	699.00	209.70
calabrese@gmail.com	699.00	209.70
mitsue_jolner@yahoo.com	699.00	209.70
jbut@gmail.com	699.00	209.70
kiley_caldarera@aol.com	699.00	209.70
leota@hotmail.com	699.00	209.70
meaghan@hotmail.com	699.99	210.00
minna_amigon@yahoo.com	799.99	240.00
vinouye@aol.com	799.99	240.00
at@venere.org	799.99	120.00
donette_foller@cox.net	799.99	240.00
gary_hernandez@yahoo.com	799.99	120.00
simona@morasca.com	1114.00	371.55

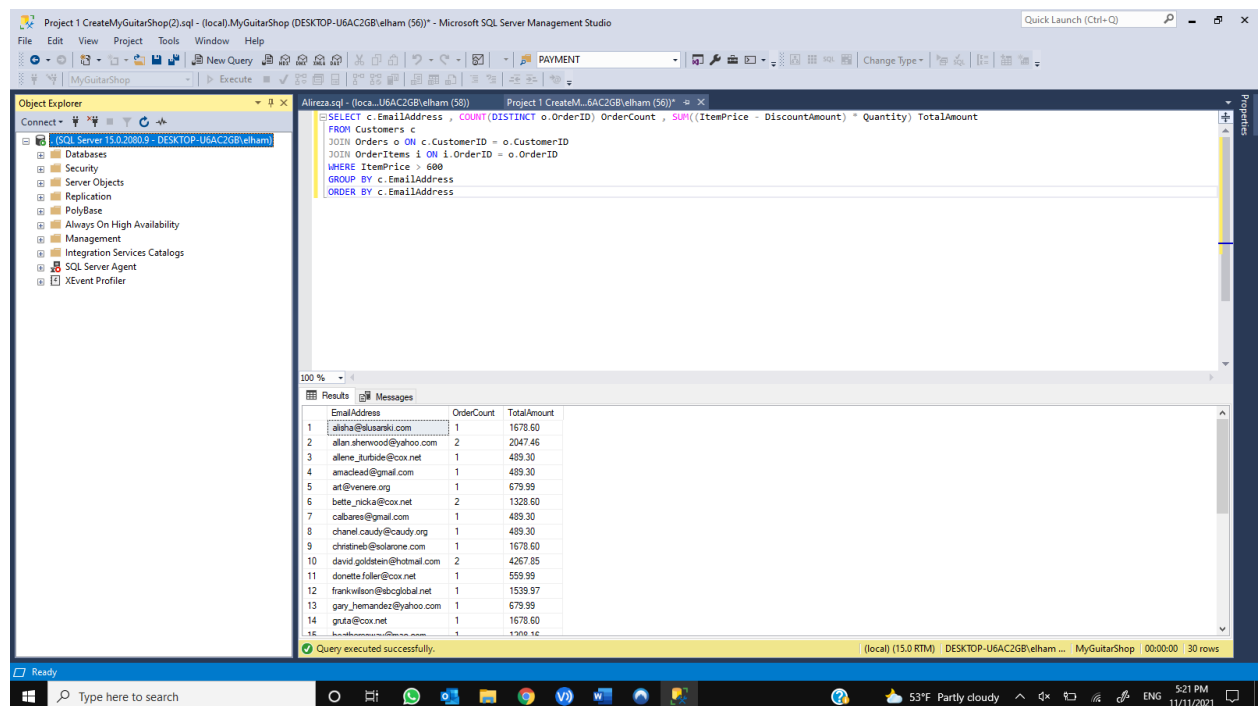
The status bar at the bottom indicates that the query was executed successfully, returning 35 rows.

Remark: Here we can see customers emails and the total price and total discount related to each customer.

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

- The EmailAddress column from the Customers table
- A count of the number of orders
- The total amount for each order (Hint: First, subtract the discount amount from the item price. Then, multiply by the quantity) Return only those rows where items have a more than 600 ItemPrice value. Sort the result set in ascending order of EmailAddress column.

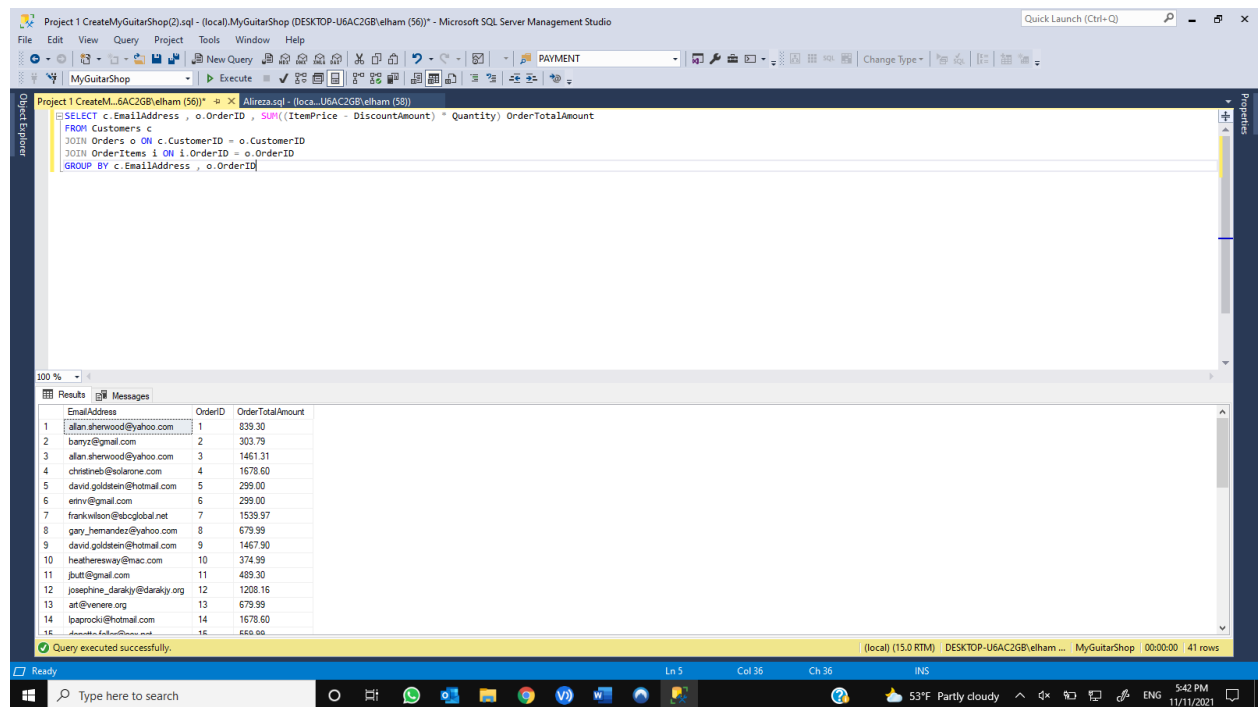
```
SELECT c.EmailAddress , COUNT(DISTINCT o.OrderID) OrderCount , SUM((ItemPrice -
DiscountAmount) * Quantity) TotalAmount
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN OrderItems i ON i.OrderID = o.OrderID
WHERE ItemPrice > 600
GROUP BY c.EmailAddress
ORDER BY c.EmailAddress
```



Remark: here we can see the customers email, total number of orders and total amount that has been paid included discount.

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. (order total amount: First, subtract the discount amount from the item price. Then, multiply by the quantity)

```
SELECT c.EmailAddress , o.OrderID , SUM((ItemPrice - DiscountAmount) * Quantity)
OrderTotalAmount
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN OrderItems i ON i.OrderID = o.OrderID
GROUP BY c.EmailAddress , o.OrderID
```



The screenshot shows the Microsoft SQL Server Management Studio interface. The query editor at the top contains the following SQL query:

```
SELECT c.EmailAddress , o.OrderID , SUM((ItemPrice - DiscountAmount) * Quantity) OrderTotalAmount
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN OrderItems i ON i.OrderID = o.OrderID
GROUP BY c.EmailAddress , o.OrderID
```

The Results pane at the bottom displays the output of the query as a table with 41 rows. The columns are EmailAddress, OrderID, and OrderTotalAmount.

	EmailAddress	OrderID	OrderTotalAmount
1	allan.sherwood@yahoo.com	1	839.30
2	banyz@gmail.com	2	303.79
3	allan.sherwood@yahoo.com	3	1461.31
4	christineb@solarone.com	4	1678.60
5	david.goldstein@hotmail.com	5	299.00
6	ediv@gmail.com	6	299.00
7	frankwilson@bcglobal.net	7	1539.97
8	pavy_hernandez@yahoo.com	8	678.99
9	david.goldstein@hotmail.com	9	1467.90
10	heatheresway@mac.com	10	374.99
11	jbutt@gmail.com	11	489.30
12	josephine_darakjy@darakiy.org	12	1208.16
13	art@venere.org	13	679.99
14	lpaprocki@hotmail.com	14	1678.60
15

Remark: here we have the email address for each customer and order ID and total amount for the order.

(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 smallest order for that customer. To do this, you can group the result set by the EmailAddress column.

```
SELECT EmailAddress , MIN(OrderTotalAmount) SmallestOrder
FROM (
    SELECT c.EmailAddress , o.OrderID , SUM((ItemPrice - DiscountAmount) * Quantity)
    OrderTotalAmount
    FROM Customers c
    JOIN Orders o ON c.CustomerID = o.CustomerID
    JOIN OrderItems i ON i.OrderID = o.OrderID
    GROUP BY c.EmailAddress , o.OrderID
) AS co
GROUP BY co.EmailAddress
```

The screenshot shows the Microsoft SQL Server Management Studio interface. The query editor displays the following SQL query:

```
SELECT EmailAddress , MIN(OrderTotalAmount) SmallestOrder
FROM (
    SELECT c.EmailAddress , o.OrderID , SUM((ItemPrice - DiscountAmount) * Quantity) OrderTotalAmount
    FROM Customers c
    JOIN Orders o ON c.CustomerID = o.CustomerID
    JOIN OrderItems i ON i.OrderID = o.OrderID
    GROUP BY c.EmailAddress , o.OrderID
) AS co
GROUP BY co.EmailAddress
```

The Results pane shows the output of the query, which is a table with two columns: EmailAddress and SmallestOrder. The results are as follows:

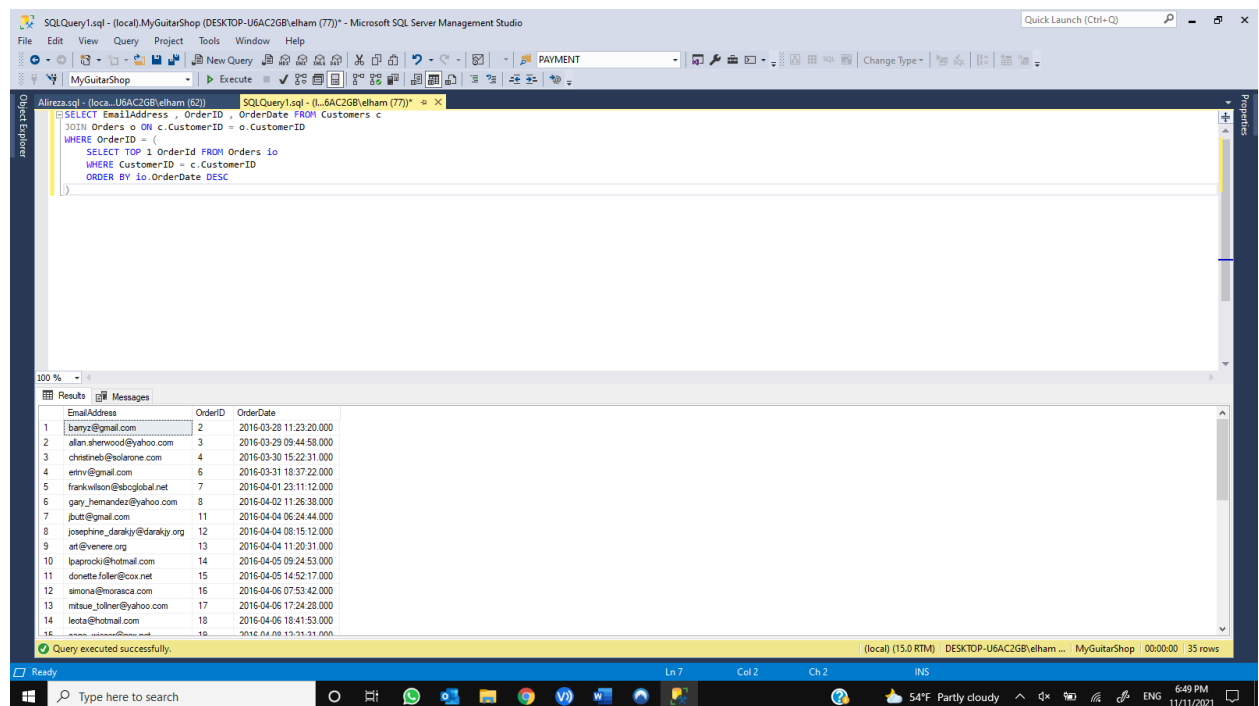
EmailAddress	SmallestOrder
alisha@alianski.com	1578.50
allan.sherwood@yahoo.com	839.30
allene_turbide@cox.net	489.30
amadlead@gmail.com	489.30
art@venere.org	679.99
banyx@gmail.com	303.79
bette_ricka@cox.net	489.30
cabares@gmail.com	489.30
chanel.caudy@caudy.org	793.09
christineb@jolanone.com	1678.60
david.goldstein@hotmail.com	299.00
dorotte.foller@cox.net	559.99
erin@gmail.com	299.00
fletcher.fiosi@yahoo.com	590.00
frankie@alianski.com	1578.50

The status bar at the bottom indicates that the query was executed successfully and returned 35 rows.

Remark: here we have smallest order for each customer and their email address.

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.

```
SELECT EmailAddress , OrderID , OrderDate FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE OrderID = (
    SELECT TOP 1 OrderId FROM Orders io
    WHERE CustomerID = c.CustomerID
    ORDER BY io.OrderDate DESC
)
```



The screenshot shows the Microsoft SQL Server Enterprise interface. The query editor at the top contains the same SQL query as shown in the previous block. The Results pane at the bottom displays the output of the query, which is a table with three columns: EmailAddress, OrderID, and OrderDate. The table contains 15 rows of data, sorted by OrderDate in descending order. The status bar at the bottom indicates that the query was executed successfully and returned 35 rows.

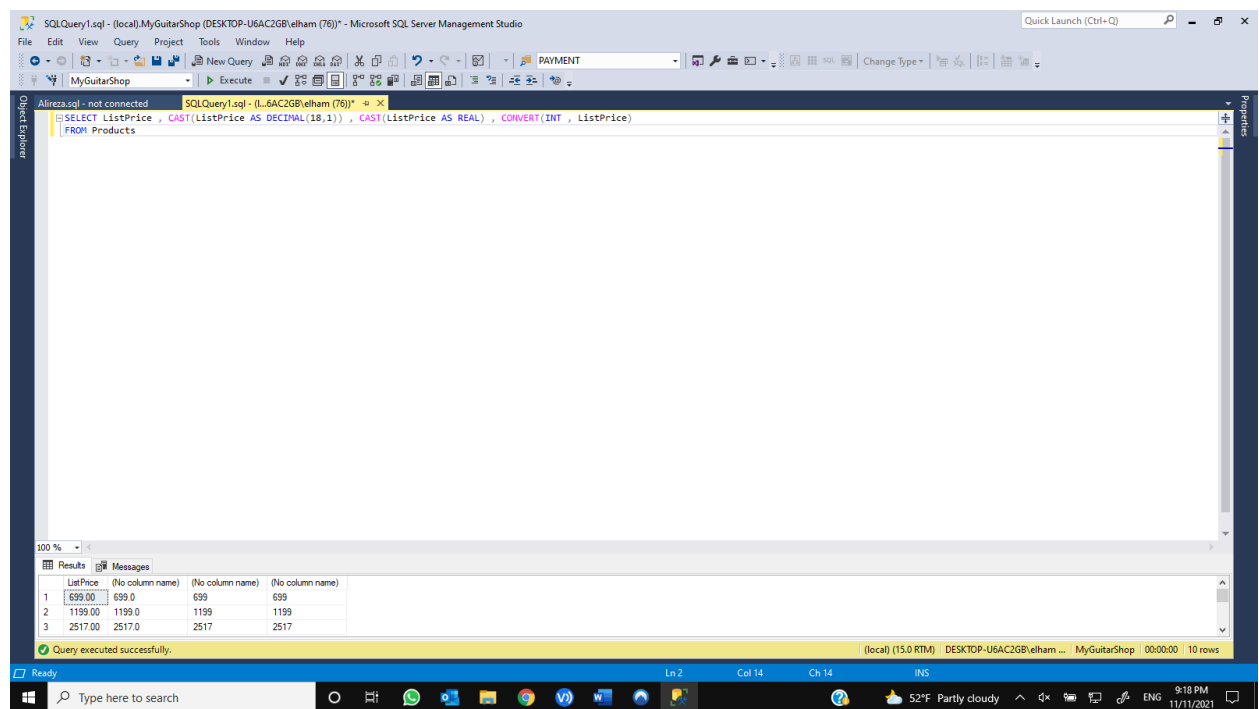
EmailAddress	OrderID	OrderDate
baryz@gmail.com	2	2016-03-28 11:23:20.000
alan.sherwood@yahoo.com	3	2016-03-29 09:44:58.000
christineb@solarone.com	4	2016-03-30 15:22:31.000
etiv@gmail.com	6	2016-03-31 18:37:22.000
frankwilson@vbcglobal.net	7	2016-04-01 23:11:12.000
gary_hernandez@yahoo.com	8	2016-04-02 11:26:35.000
jbutt@gmail.com	11	2016-04-04 06:24:44.000
josephine_daskiy@daskiy.org	12	2016-04-04 08:15:12.000
ait@vsnor.org	13	2016-04-04 11:20:31.000
lspirocki@hotmail.com	14	2016-04-05 09:24:53.000
dorthe.foller@cox.net	15	2016-04-05 14:52:17.000
simona@morasca.com	16	2016-04-06 07:53:42.000
mitsue_tolner@yahoo.com	17	2016-04-06 17:24:28.000
leota@hotmail.com	18	2016-04-06 18:41:53.000
www.vbcglobal.net	19	2016-04-08 13:51:51.000

Remark: I used a correlated subquery to get the latest order date and the order ID and the email of customer who created that order.

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 1 digit 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.

```
SELECT ListPrice , CAST(ListPrice AS DECIMAL(18,1)) , CAST(ListPrice AS REAL) ,  
CONVERT(INT , ListPrice)  
FROM Products
```

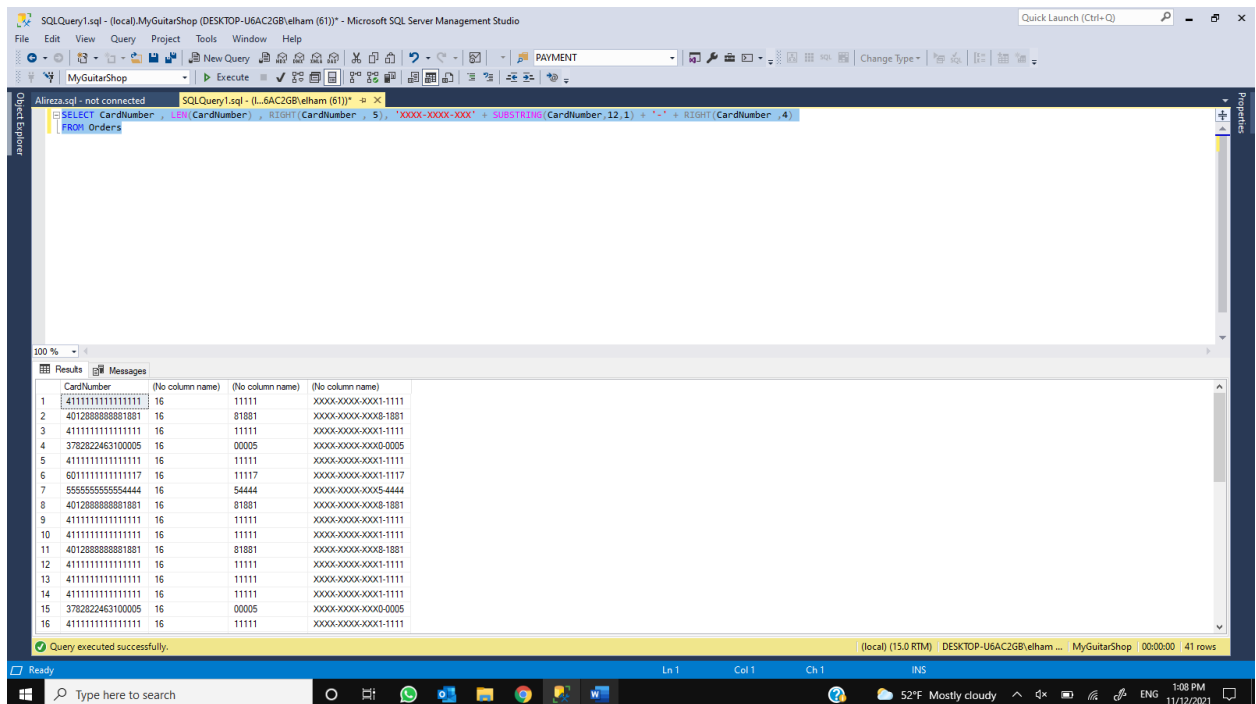


Remark: here I returned the list price, List price with one number after decimal point, List price as a real number using the Cast function. I also converted list price to integer.

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 five digits of the CardNumber column
- d) A column that displays the last five digits of the CardNumber column in this format: XXXX-XXXX-XXX-XXXX-XXX1-2345. In other words, use X's for the first 11 digits of the card number and actual numbers for the last five digits of the number and include dash symbols as specified in the format.

```
SELECT CardNumber , LEN(CardNumber) , RIGHT(CardNumber , 5) , 'XXXX-XXXX-XXX' +
SUBSTRING(CardNumber,12,1) + '-' + RIGHT(CardNumber ,4)
FROM Orders
```

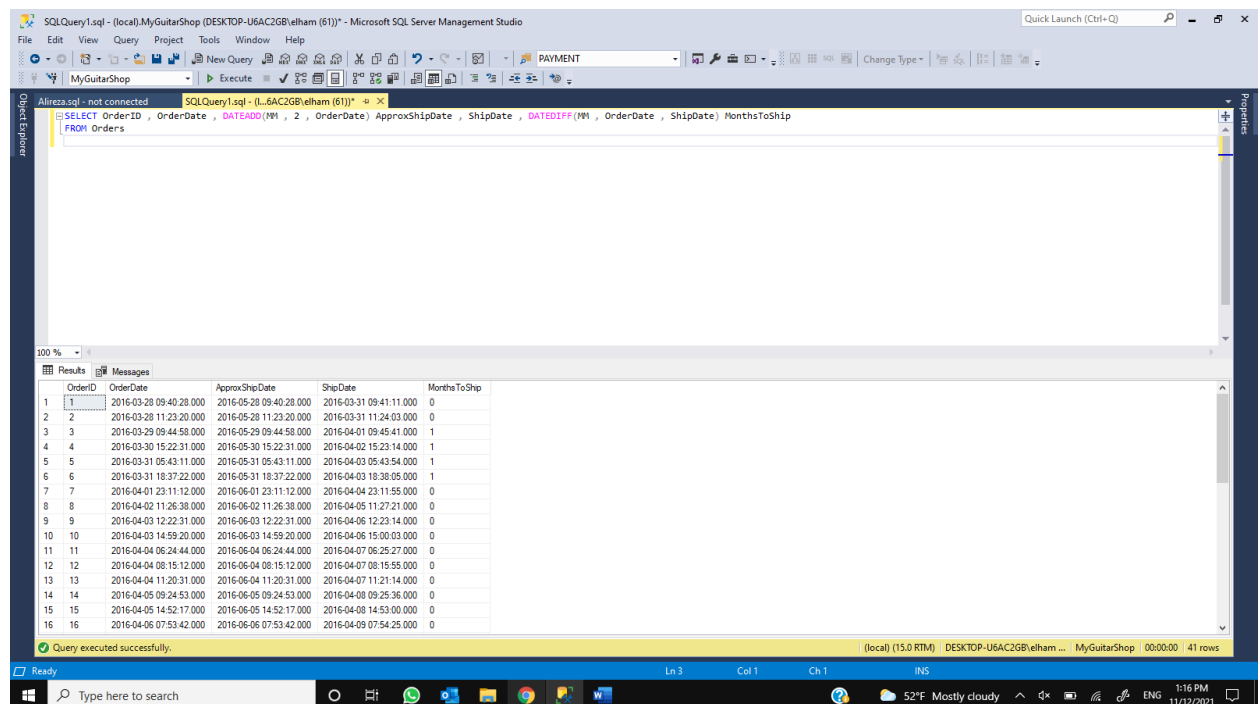


Remark: here we get the card number, its length, its last five digit and the partially hidden card number in required format.

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 2 months to the OrderDate column
- d) The ShipDate column
- e) A column named MonthsToShip that shows the number of months between the order date and the ship date When you have this working, add a WHERE clause that retrieves just the orders for March 2015.

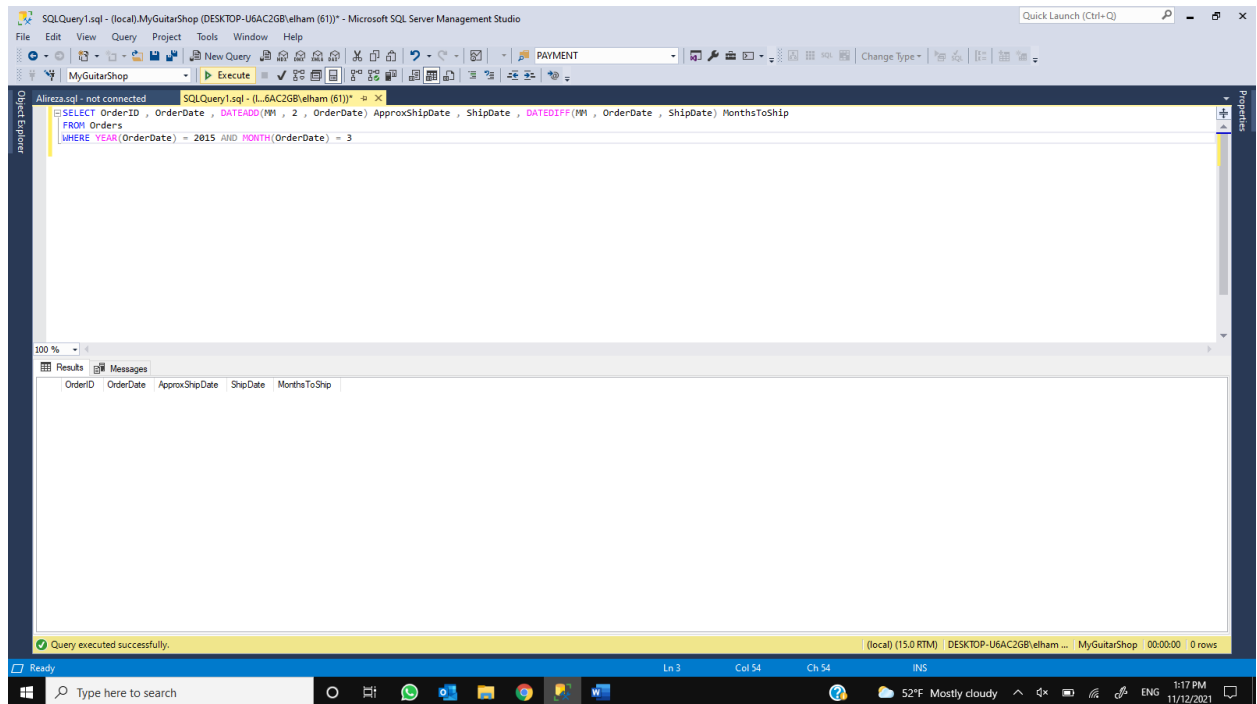
```
SELECT OrderID , OrderDate , DATEADD(MM , 2 , OrderDate) ApproxShipDate , ShipDate ,  
DATEDIFF(MM , OrderDate , ShipDate) MonthsToShip  
FROM Orders
```



OrderID	OrderDate	ApproxShipDate	ShipDate	MonthsToShip
1	2016-03-28 09:40:28.000	2016-05-28 09:40:28.000	2016-03-31 09:41:11.000	0
2	2016-03-28 11:23:20.000	2016-05-28 11:23:20.000	2016-03-31 11:24:03.000	0
3	2016-03-29 09:44:58.000	2016-05-29 09:44:58.000	2016-04-01 09:45:41.000	1
4	2016-03-30 15:22:31.000	2016-05-30 15:22:31.000	2016-04-02 15:23:14.000	1
5	2016-03-31 05:43:11.000	2016-05-31 05:43:11.000	2016-04-03 05:43:54.000	1
6	2016-03-31 18:37:22.000	2016-05-31 18:37:22.000	2016-04-03 18:38:05.000	1
7	2016-04-01 23:11:12.000	2016-06-01 23:11:12.000	2016-04-04 23:11:55.000	0
8	2016-04-02 11:26:38.000	2016-06-02 11:26:38.000	2016-04-05 11:27:21.000	0
9	2016-04-03 12:22:31.000	2016-06-03 12:22:31.000	2016-04-06 12:23:14.000	0
10	2016-04-03 14:59:20.000	2016-06-03 14:59:20.000	2016-04-06 15:00:03.000	0
11	2016-04-04 06:24:44.000	2016-06-04 06:24:44.000	2016-04-07 06:25:27.000	0
12	2016-04-04 08:15:12.000	2016-06-04 08:15:12.000	2016-04-07 08:15:55.000	0
13	2016-04-04 11:20:31.000	2016-06-04 11:20:31.000	2016-04-07 11:21:14.000	0
14	2016-04-05 09:24:53.000	2016-06-05 09:24:53.000	2016-04-08 09:25:36.000	0
15	2016-04-05 14:52:17.000	2016-06-05 14:52:17.000	2016-04-08 14:53:00.000	0
16	2016-04-06 07:53:42.000	2016-06-06 07:53:42.000	2016-04-09 07:54:25.000	0

Remark: this is just a test to get all the order ID, order date, approximate ship date, ship date and the number of months it took to ship the product. But we haven't filtered result.

```
SELECT OrderID , OrderDate , DATEADD(MM , 2 , OrderDate) ApproxShipDate , ShipDate ,  
DATEDIFF(MM , OrderDate , ShipDate) MonthsToShip  
FROM Orders  
WHERE YEAR(OrderDate) = 2015 AND MONTH(OrderDate) = 3
```



Remark: here we filtered the result to get the orders for March 2015. As can be seen, there is none.

For question 10-11: To test whether a table has been modified correctly as you do these questions, please write, and run an appropriate SELECT statement and take full screenshots of the verification without cropping any part of you SQL server window.

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.

`SELECT * FROM Customers`

The screenshot shows the Microsoft SQL Server Enterprise Manager interface. The query window displays the following SQL code:

```

INSERT INTO Customers (EmailAddress, Password, FirstName, LastName)
VALUES ('kriegerrobert@gmail.com', '', 'Krieger', 'Robert')
GO
SELECT * FROM Customers
WHERE CustomerID = @@IDENTITY

```

The Results pane shows a table with 485 rows of customer data. The table has the following columns: CustomerID, EmailAddress, Password, FirstName, LastName, ShippingAddressID, and BillingAddressID. The data is displayed in a grid format.

CustomerID	EmailAddress	Password	FirstName	LastName	ShippingAddressID	BillingAddressID
470	nan@koppinger.com	4a430d63aaf01ae1018128c8d6f09da2035422c	Nan	Koppinger	496	496
471	idewar@dewar.com	fd1897a9f79531d5154b1c8e1ee0379191465ad	Izetta	Dewar	497	497
472	tegan.arceo@arceo.org	077143979785eac2a2ca7584e7b3e3d8b8dfc4	Tegan	Arceo	498	498
473	ruthann@hotmail.com	0159f3d0c2485ad3e1cb4556249e44a8724a44	Ruthann	Keener	499	499
474	joni_breland@cox.net	a785dc303c19b2cc2bf99949d2920b0a8d9f9657	Joni	Breland	500	500
475	ventrino@cox.net	c15f8cc70c1d65a4033952b0aef5522b0416cd3f	Vi	Pierro	501	501
476	colette.kardas@yahoo.com	eaf759f4fc8479928774a99eaf69156559b4f	Colette	Kardas	502	502
477	malcolm_tromblay@cox.net	6b03a9c33a4e59a4c26c4e3e42078b8f4caf39	Malcolm	Tromblay	503	503
478	ryan@cox.net	67c4bd033af41cd1a24c4df925a8b1051ce3378	Ryan	Hamos	504	504
479	jess.chaffins@chaffins.org	7c3d3035e1d895f8494c7a27b4d6afcd0a74057	Jess	Chaffins	505	505
480	abouton@yahoo.com	e8af7d0c9dd491dd289b35563f7285435a625e4	Sharen	Bouton	506	506
481	nickolas_juvera@cox.net	77602baf770e3618da53c0883d7c9bc82eca1f6	Nickolas	Juvera	507	507
482	gary_runlee@runlee.org	915e9617c0f0b08e182a870a79123ae61c7efec	Gary	Runlee	508	508
483	diane@cox.net	fba3f2a325a4e4eaf5992cbcc910ef91ecce3191	Diane	Devreese	510	510
484	roslyn.chavous@chavous.org	3bf8ab2dbdc781f2000472c4f9d866433cccd35	Roslyn	Chavous	511	511
485	glory@yahoo.com	31c689d737359ce214d099f9b0dcf6de82a573d	Glory	Schieler	512	512

The status bar at the bottom indicates that the query was executed successfully and returned 485 rows.

Remark: First by running the above query (I just run the `SELECT * FROM Customers`

Part) we get the original table. We can see that it has 485 rows.

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

```
SELECT * FROM Customers
```

The screenshot shows the Microsoft SQL Server Management Studio interface. The query editor at the top contains the following SQL code:

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

SELECT * FROM Customers
```

The query was executed successfully, and the results are displayed in the lower pane. The table has 7 columns: CustomerID, EmailAddress, Password, FirstName, LastName, ShippingAddressID, and BillingAddressID. The new customer, Robert Krieger, is listed as the 486th row.

CustomerID	EmailAddress	Password	FirstName	LastName	ShippingAddressID	BillingAddressID
473	richardn@rmail.com	01593dfcd2489ed3e1cb4596249e44a87224a44	Ruthann	Koener	499	499
474	joni_breland@cox.net	a786dc303c19e2cc3af9994942920b0a9f9a57	Joni	Breland	500	500
475	vrenfro@cox.net	d19f8cc1bc1cb6a40339b52baed5922b41bd0f	Vi	Pentfro	501	501
476	colette_kardas@yahoo.com	edf7598de6c8479928774a95aeaf69156589b4f	Colette	Kardas	502	502
477	malcolm_tromblay@cox.net	fb03a9e33a4e59fa4c26c4e3e420788df4ca139	Malcolm	Tromblay	503	503
478	ryan@cox.net	67c4b0d033af41cd1a24c4df925de8b1051ce378	Ryan	Hamos	504	504
479	jess.chaffins@chaffins.org	7c3d3d3035e1d895f8494c7e27b4d6af00a74057	Jess	Chaffins	505	505
480	sbourbon@yahoo.com	e8af7d0c9dd491dd289c356c3f7285435a625e4	Sharen	Bourbon	506	506
481	nickolas_juvera@cox.net	77602baf770e3618da53c0883d7c9bc82acca1f6	Nickolas	Juvera	507	507
482	gary_nunlee@nunlee.org	915e9617c0f0b08e182a870a79129ae61c7efec	Gary	Nunlee	508	508
483	diane@cox.net	fb492a325a4e4eaf5992b0cc910ef91ecce3191	Diane	Devreese	510	510
484	roslyn.chavous@chavous.org	3bf8ab2dbdc781f2000472c4f9d986d6433cc035	Roslyn	Chavous	511	511
485	glory@yahoo.com	31c689d737359ce214d099f9b0bcfd6e8a573d	Glory	Schwieler	512	512
486	kriegerrobert@gmail.com		Krieger	Robert	NULL	NULL

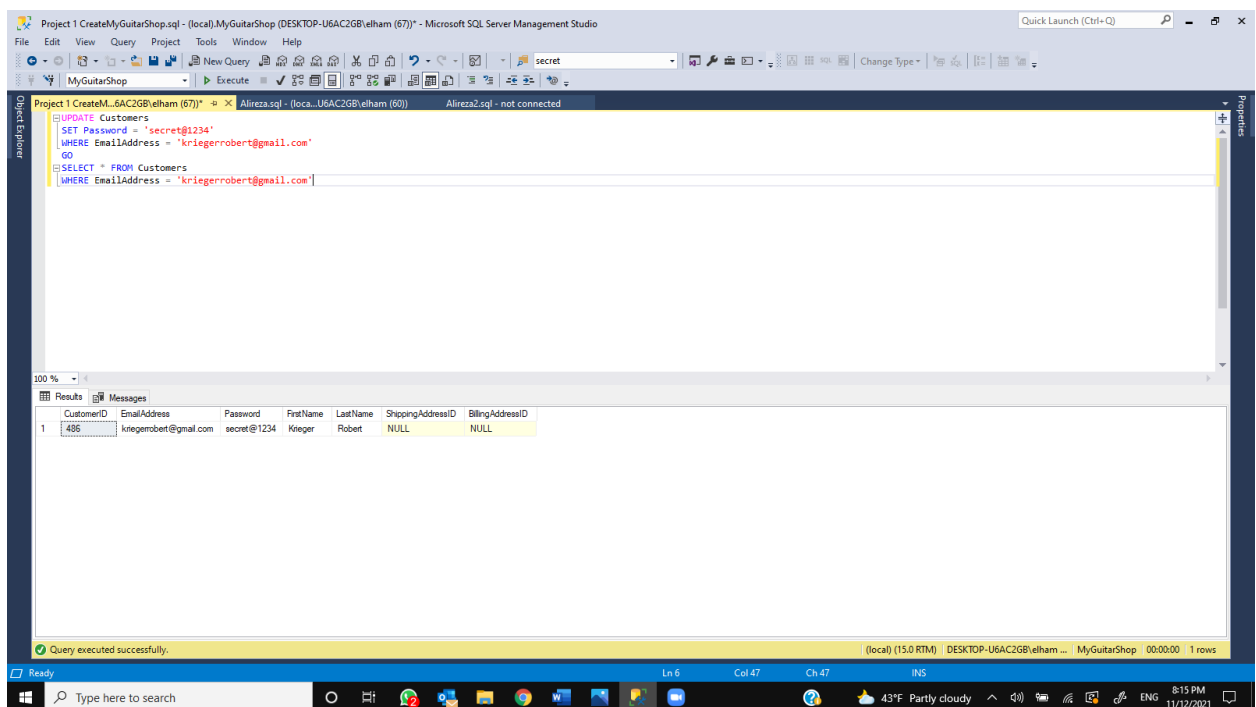
The status bar at the bottom indicates that the query was executed successfully and that there are 486 rows in the result set.

Remark: now we added the 486th row with customer information.

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.

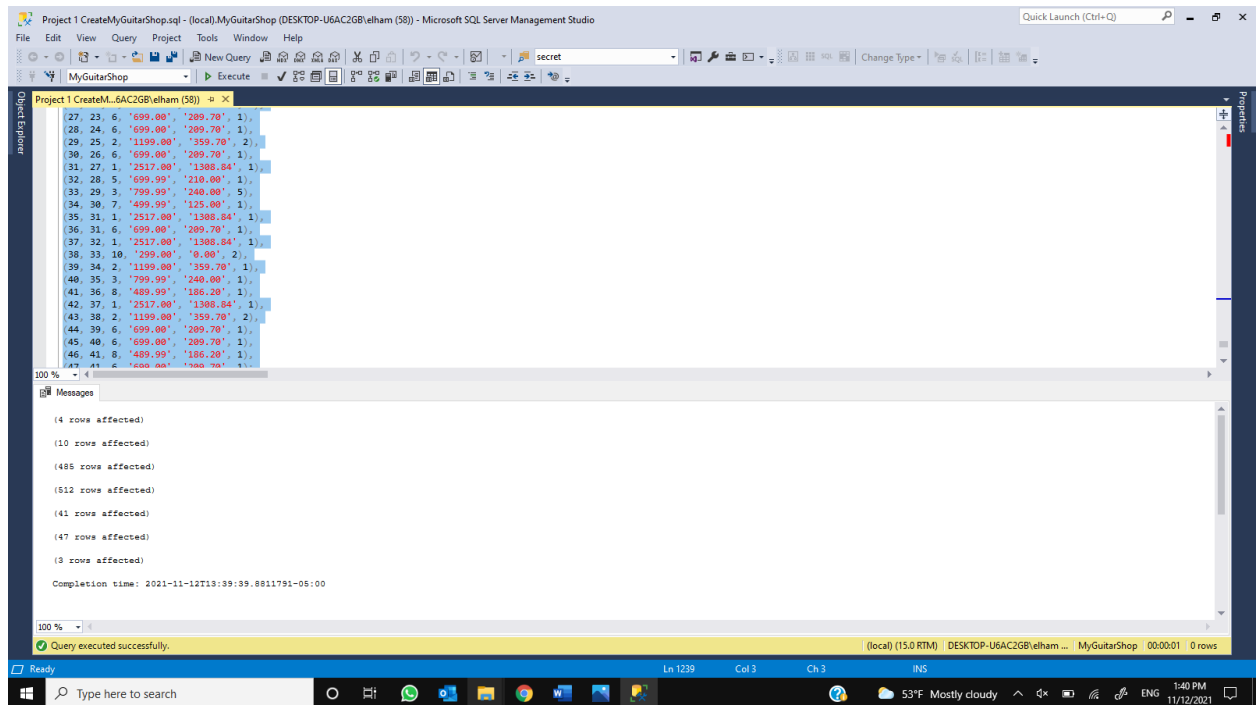
```
UPDATE Customers
SET Password = 'secret@1234'
WHERE EmailAddress = 'kriegerrobert@gmail.com'
GO
SELECT * FROM Customers
WHERE EmailAddress = 'kriegerrobert@gmail.com'
```



Remark: we updated the customer password and set that to secret@1234

D. Advanced SQL skills (views/stored procedures/ functions / scripts) [24 pts.]

Open the script named CreateMyGuitarShop.sql and run this script again. That should restore the data that's in the database. Then complete questions in Section D. Screenshot of execution is required for each question. Please also use SELECT statement to verify results of your codes (Full Screenshots of verification are required).



Remark: Here we set up the database again.

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.

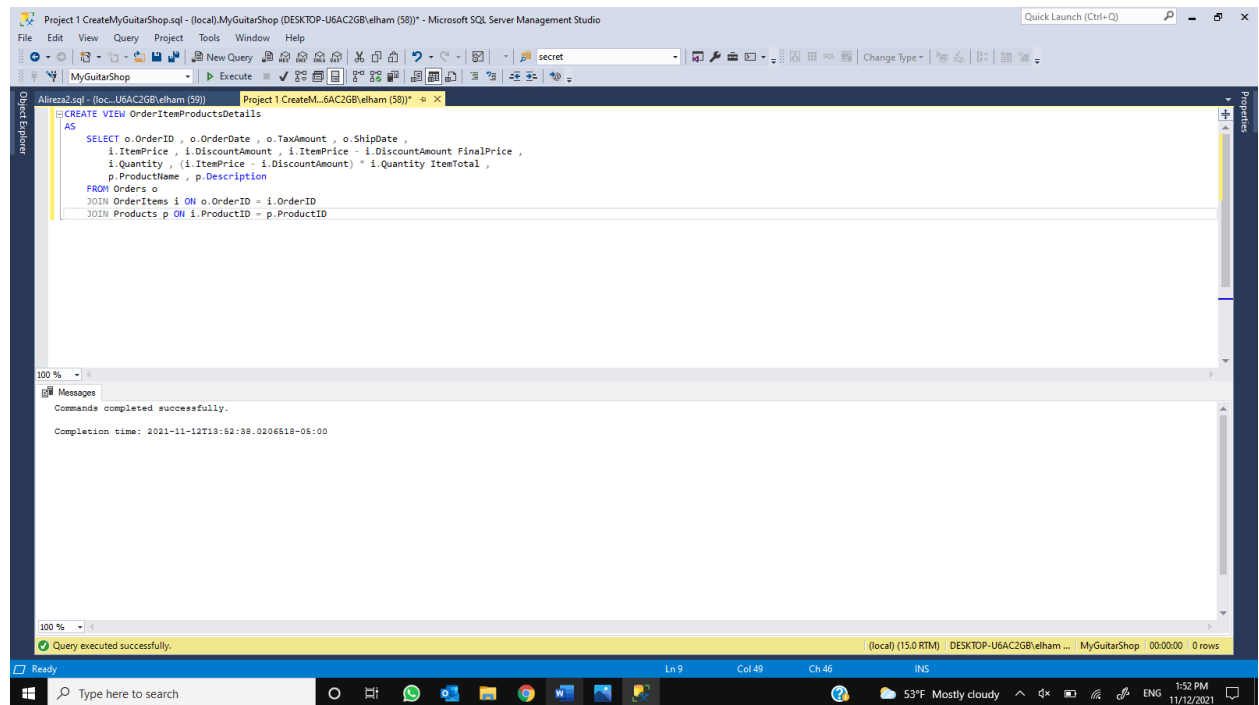
```
CREATE VIEW OrderItemProductsDetails
AS
```

```
SELECT o.OrderID , o.OrderDate , o.TaxAmount , o.ShipDate ,
       i.ItemPrice , i.DiscountAmount , i.ItemPrice - i.DiscountAmount FinalPrice ,
       i.Quantity , (i.ItemPrice - i.DiscountAmount) * i.Quantity ItemTotal ,
       p.ProductName , p.Description
```

```
FROM Orders o
```

```
JOIN OrderItems i ON o.OrderID = i.OrderID
```

```
JOIN Products p ON i.ProductID = p.ProductID
```



Remark: here we created the view and we can see the result in the next screenshot.

`SELECT * FROM OrderItemProductsDetails`

The screenshot shows the Microsoft SQL Server Enterprise Manager interface. The query editor displays the following SQL code:

```

CREATE VIEW OrderItemProductsDetails
AS
SELECT o.OrderID, o.OrderDate, o.TaxAmount, o.ShipDate,
       i.ItemPrice, i.DiscountAmount, i.ItemPrice - i.DiscountAmount FinalPrice,
       i.Quantity, (i.ItemPrice - i.DiscountAmount) * i.Quantity ItemTotal,
       p.ProductName, p.Description
FROM Orders o
JOIN OrderItems i ON o.OrderID = i.OrderID
JOIN Products p ON i.ProductID = p.ProductID

SELECT * FROM OrderItemProductsDetails

```

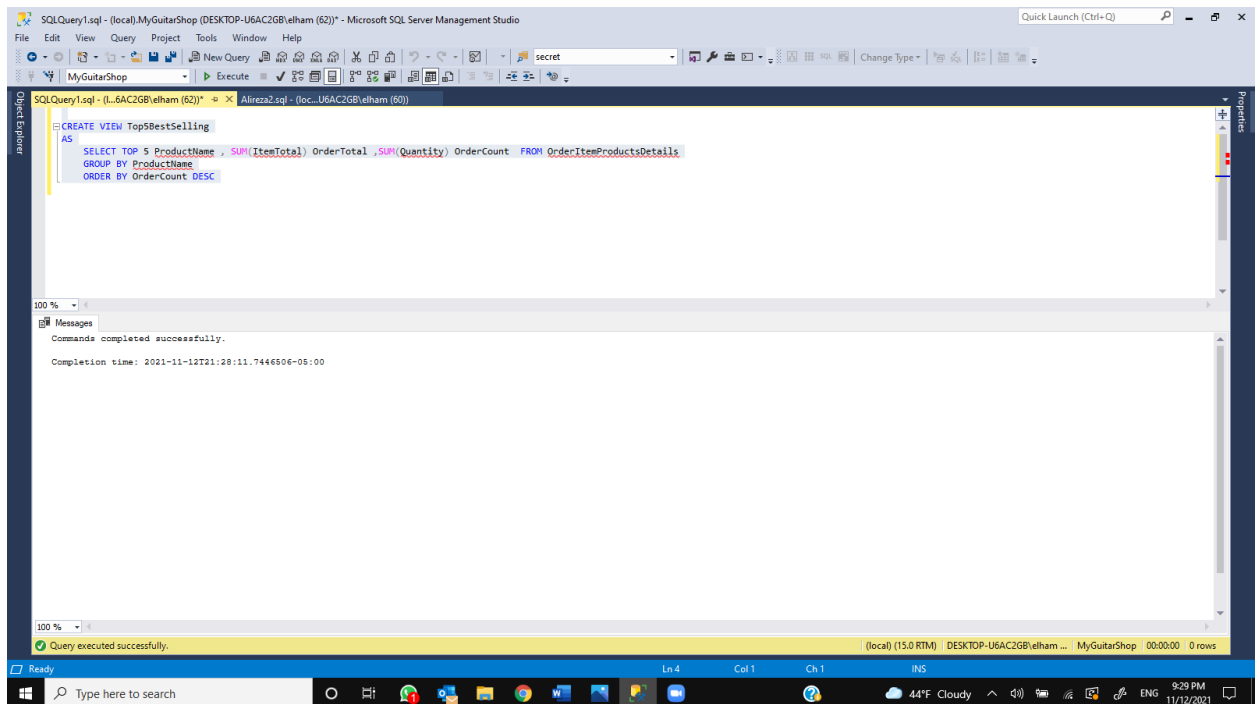
The Results pane shows the output of the query, which is a table with 12 columns: OrderID, OrderDate, TaxAmount, ShipDate, ItemPrice, DiscountAmount, FinalPrice, Quantity, ItemTotal, ProductName, and Description. The table contains 14 rows of data.

OrderID	OrderDate	TaxAmount	ShipDate	ItemPrice	DiscountAmount	FinalPrice	Quantity	ItemTotal	ProductName	Description
1	2016-03-28 09:40:28.000	58.75	2016-03-31 09:41:11.000	1199.00	359.70	839.30	1	839.30	Gibson Les Paul	This Les Paul guitar offers a carved top and humbu...
2	2016-03-28 11:23:20.000	21.27	2016-03-31 11:24:03.000	489.99	186.20	303.79	1	303.79	Hofner Icon	With authentic details inspired by the original, the H...
3	2016-03-29 09:44:58.000	102.29	2016-04-01 09:45:41.000	2517.00	1308.84	1208.16	1	1208.16	Fender Stratocaster	The Fender Stratocaster is the electric guitar desig...
4	2016-03-29 09:44:58.000	102.29	2016-04-01 09:45:41.000	415.00	161.85	253.15	1	253.15	Ludwig 5-piece Drum Set with Cymbals	This product includes a Ludwig 5-piece drum set a...
5	2016-03-30 15:22:31.000	117.50	2016-04-02 15:23:14.000	1199.00	359.70	839.30	2	1678.60	Gibson Les Paul	This Les Paul guitar offers a carved top and humbu...
6	2016-03-31 05:43:11.000	20.93	2016-04-03 05:43:54.000	299.00	0.00	299.00	1	299.00	Tama 5-Piece Drum Set with Cymbals	The Tama 5-piece Drum Set is the most affordable ...
7	2016-03-31 18:37:22.000	20.93	2016-04-03 18:38:05.000	299.00	0.00	299.00	1	299.00	Tama 5-Piece Drum Set with Cymbals	The Tama 5-piece Drum Set is the most affordable ...
8	2016-04-01 23:11:12.000	107.80	2016-04-04 23:11:55.000	699.99	210.00	489.99	1	489.99	Washburn D10S	The Washburn D10S acoustic guitar is superbly cr...
9	2016-04-01 23:11:12.000	107.80	2016-04-04 23:11:55.000	799.99	240.00	559.99	1	559.99	Gibson SG	This Gibson SG electric guitar takes the best of the...
10	2016-04-01 23:11:12.000	107.80	2016-04-04 23:11:55.000	699.99	210.00	489.99	1	489.99	Washburn D10S	The Washburn D10S acoustic guitar is superbly cr...
11	2016-04-02 11:26:38.000	47.60	2016-04-05 11:27:21.000	799.99	120.00	679.99	1	679.99	Yamaha FG700S	The Yamaha FG700S solid top acoustic guitar has ...
12	2016-04-03 12:23:31.000	102.75	2016-04-06 12:23:14.000	699.00	209.70	489.30	3	1467.90	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar ...
13	2016-04-04 14:59:20.000	26.25	2016-04-06 15:00:03.000	499.99	125.00	374.99	1	374.99	Fender Precision	The Fender Precision bass guitar delivers the soun...
14	2016-04-04 06:24:44.000	34.25	2016-04-07 06:25:27.000	699.00	209.70	489.30	1	489.30	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar ...

Remark: using select statement we can see the view. It retrieves order ID, date, total amount and discount, ship date, the price for the item and final price, quantity ordered, product name and description.

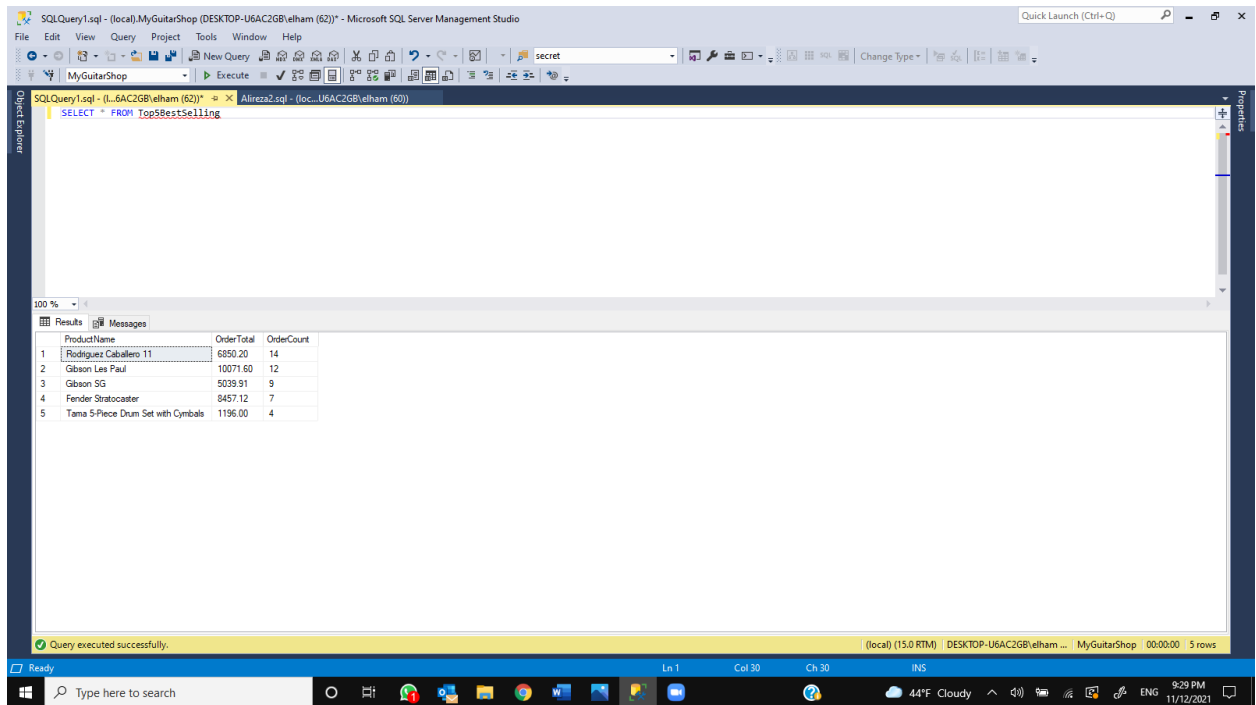
2. [5] Create a view named Top5BestSelling that uses the view you created in Section D Question 1. This view should return some summary information about five best selling products. Each row should include these columns: ProductName, OrderTotal (the total sales for the product) and OrderCount (the number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

```
CREATE VIEW Top5BestSelling
AS
    SELECT TOP 5 ProductName , SUM(ItemTotal) OrderTotal ,SUM(Quantity)
    OrderCount FROM OrderItemProductsDetails
    GROUP BY ProductName
    ORDER BY OrderCount DESC
```



Remark: First we created the view that gives us the best 5 products regarding sale.

```
SELECT * FROM Top5BestSelling
```



Remark: here we used a select statement to retrieve the view table. It has the first 5 bestselling product, their names, total amount and number sold.

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.

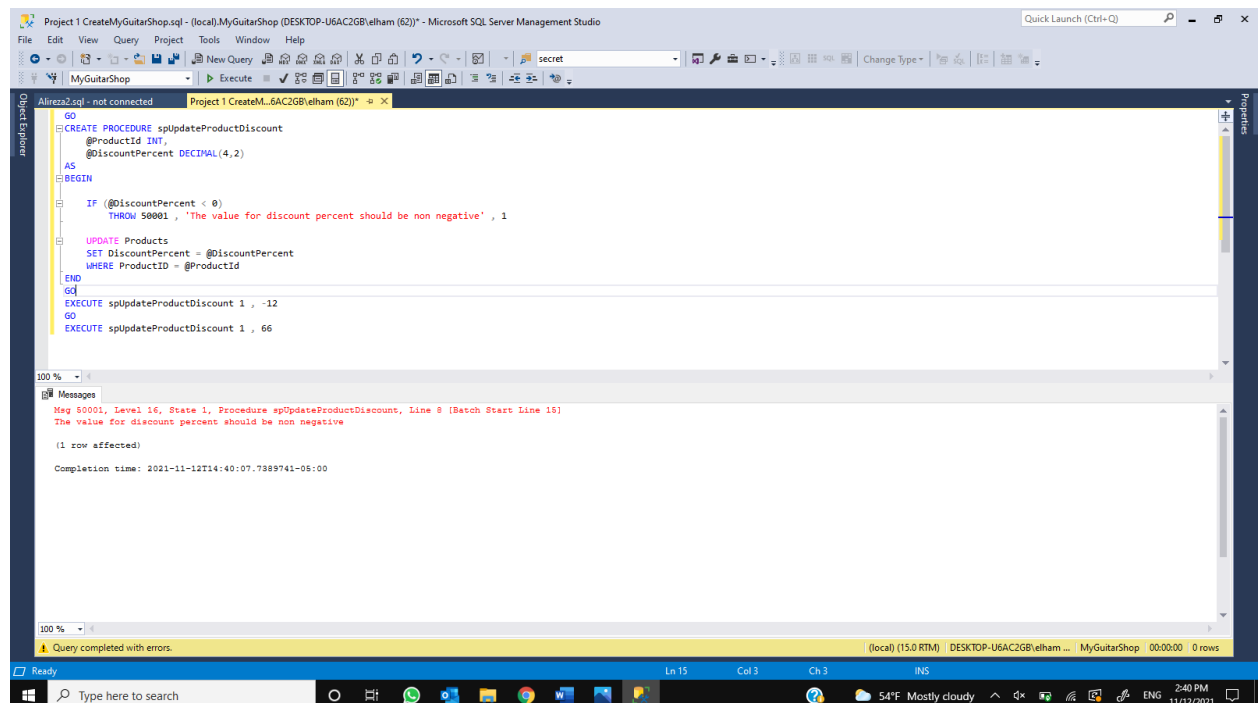
If the value for the `DiscountPercent` column is a negative number, the stored procedure should raise an error that indicates that the value for this column must be a positive number. Code at least two `EXEC` statements that test this procedure.

```
GO
CREATE PROCEDURE spUpdateProductDiscount
    @ProductId INT,
    @DiscountPercent DECIMAL(4,2)
AS
BEGIN

    IF (@DiscountPercent < 0)
        THROW 50001, 'The value for discount percent should be non negative', 1

    UPDATE Products
    SET DiscountPercent = @DiscountPercent
    WHERE ProductID = @ProductId

END
GO
EXECUTE spUpdateProductDiscount 1, -12
GO
EXECUTE spUpdateProductDiscount 1, 66
```



Remark: here we created a procedure that raise an error if the `DiscountPercent` is negative. If not, updates the `DiscountPercent` in the `Products` table.

4. [5] Write a script that calculates the common factors between 4 and 32. 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:

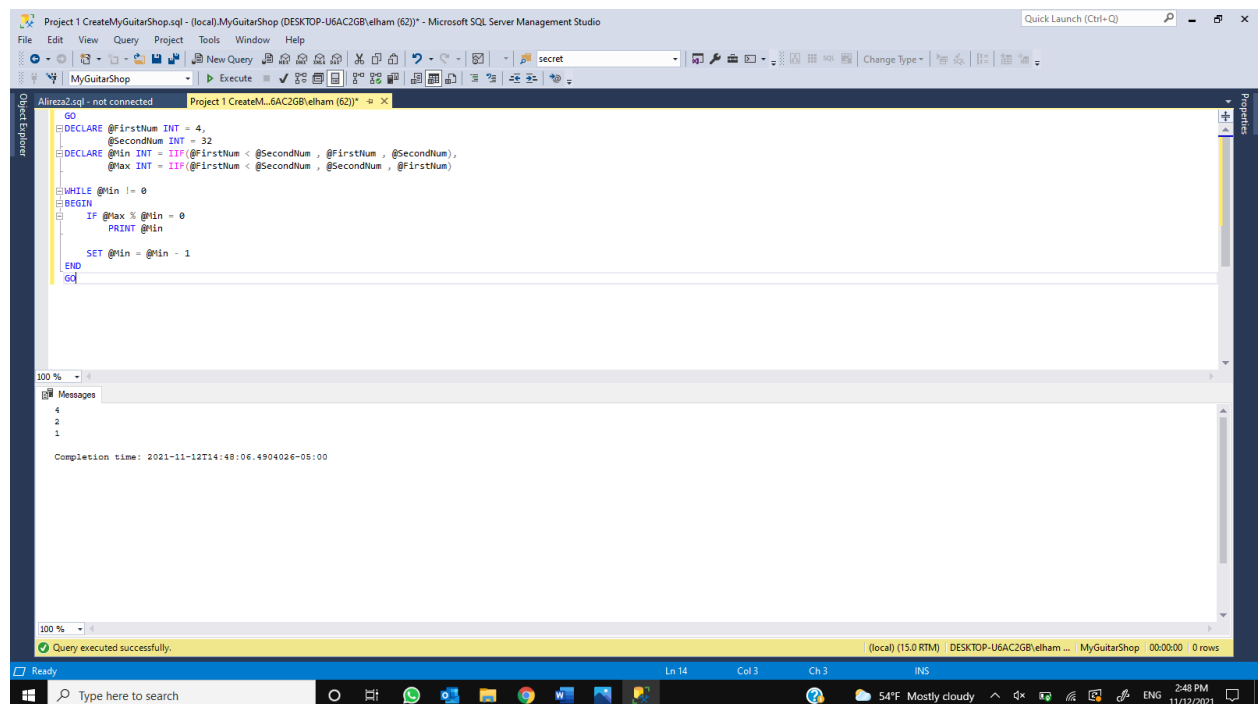
Common factors of 4 and 32

1
2
4

```
GO
DECLARE @FirstNum INT = 4,
        @SecondNum INT = 32
DECLARE @Min INT = IIF(@FirstNum < @SecondNum , @FirstNum , @SecondNum),
        @Max INT = IIF(@FirstNum < @SecondNum , @SecondNum , @FirstNum)

WHILE @Min != 0
BEGIN
    IF @Max % @Min = 0
        PRINT @Min

    SET @Min = @Min - 1
END
GO
```



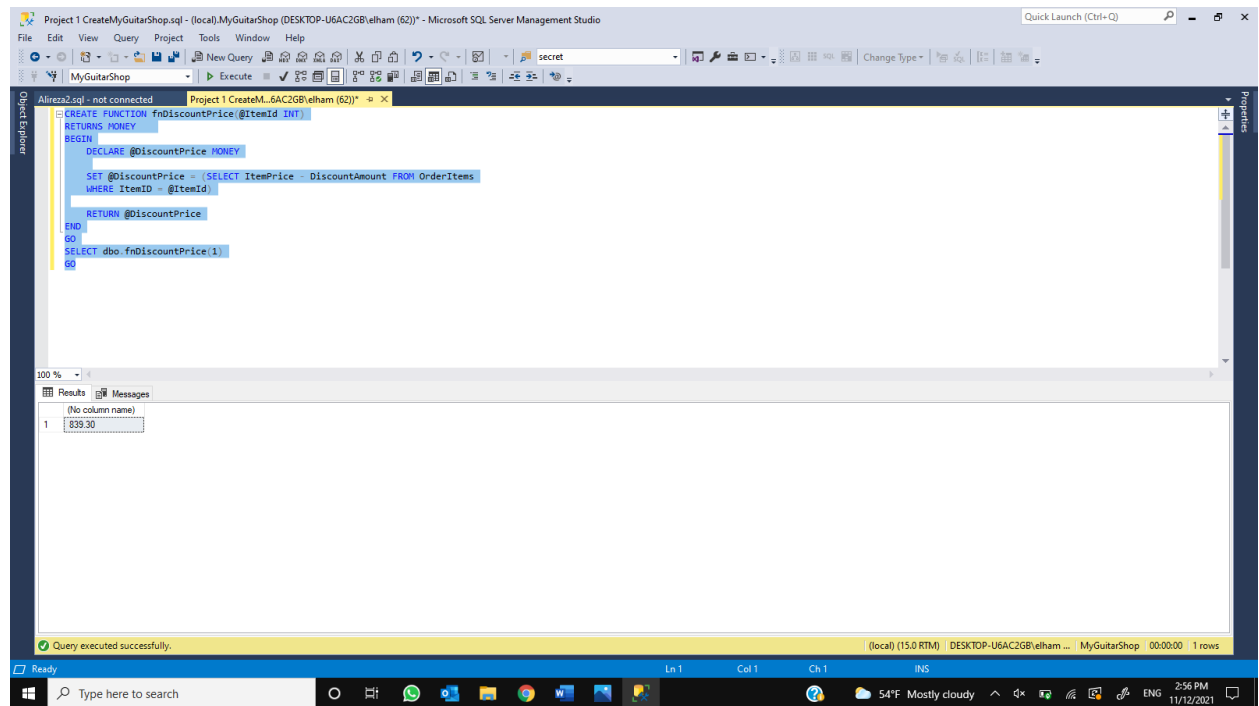
Remark: I created a script here which calculates the common factors between 4 and 32

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.

```
CREATE FUNCTION fnDiscountPrice (@ItemId INT)
RETURNS MONEY
BEGIN
    DECLARE @DiscountPrice MONEY

    SET @DiscountPrice = (SELECT ItemPrice - DiscountAmount FROM OrderItems
    WHERE ItemID = @ItemId)

    RETURN @DiscountPrice
END
GO
SELECT dbo.fnDiscountPrice(1)
GO
```



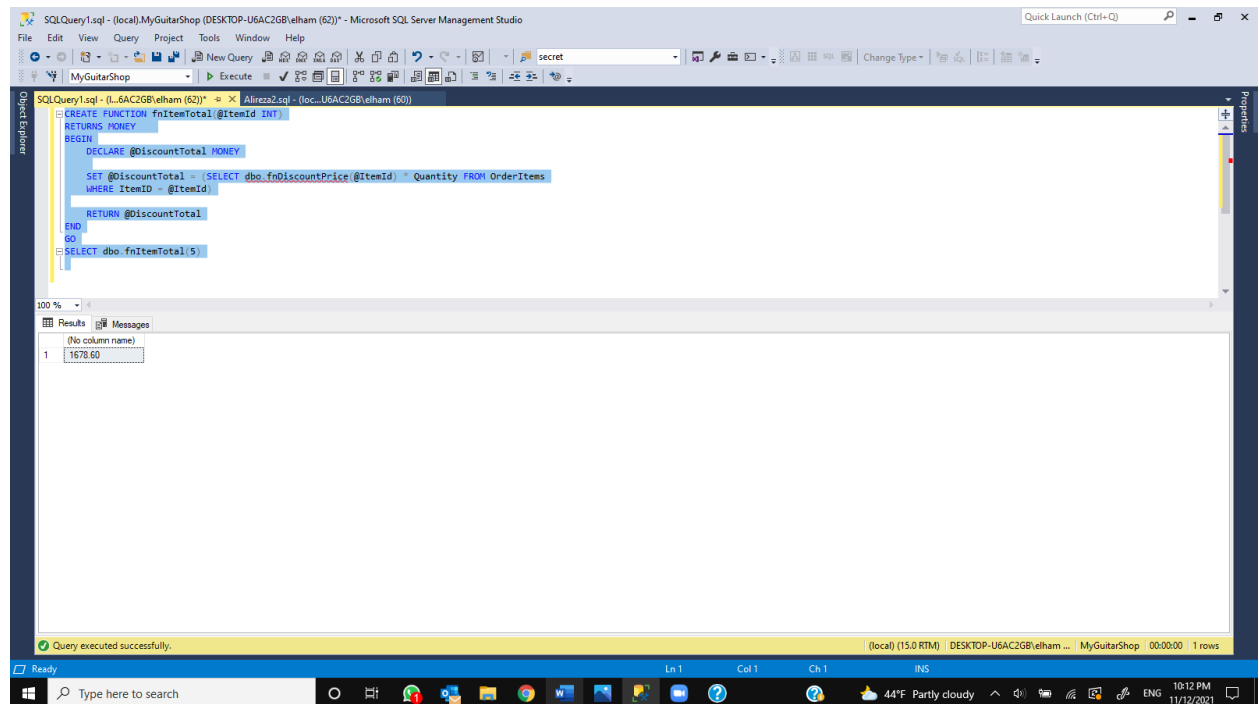
Remark: we created a function that calculates the discount price of an item. By entering 1 for item ID you will get 839.30.

(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(@ItemId INT)
RETURNS MONEY
BEGIN
    DECLARE @DiscountTotal MONEY

    SET @DiscountTotal = (SELECT dbo.fnDiscountPrice(@ItemId) * Quantity FROM
OrderItems
WHERE ItemID = @ItemId)

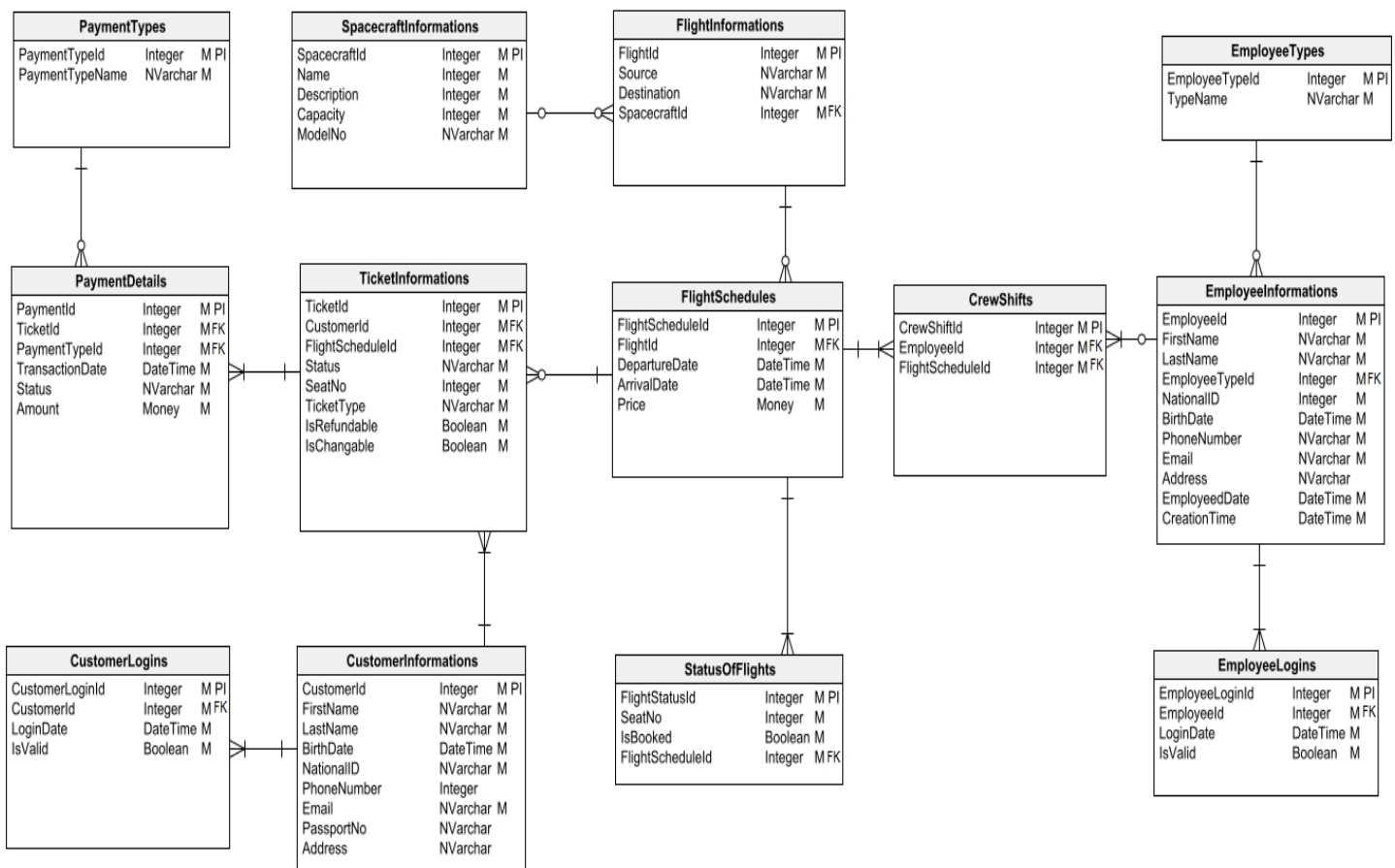
    RETURN @DiscountTotal
END
GO
SELECT dbo.fnItemTotal(5)
```



Remark: the created function calculates the total amount of an item in the `OrderItems` table. By entering 5 you will get 1678.60.

II. Database Design [20 pts.] Create a sample database design (with a name of your choice) for a space flight company like SpaceX and draw one database model for it. The system keeps track of flight information, customer information, employee information, customer and employee login information, rocket/spacecraft information (like Falcon 9), ticket information (like when a customer books a ticket, whether it can be changed or refunded), status of flights (which seat has booked and which has not), flight schedule and payment details etc. 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).
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.



Descriptions:

First, we as we are considering doing to space which is not a short trip, we assume that employees won't be customers

CustomerLoginID is the same as username that customer uses for his/her account.

As we are talking about space traveling, which is a demanding task, one employee cannot have two positions. They cannot be captain and crew at the same time.

CustomerLogins and EmployeeLogins are the tables that record the history of login activity for customers and employees. EmployeeLogins table have a foreign key of EmployeeId from EmployeeInformations table. CustomersLogins table have a foreign key of CustomerId from CustomerInformations table.

Payment Type Refers to the type of payment like cash or credit.

Status in payment details can be values (paid, refunded).

TicketInformations table contains ticket information's that purchased by each customer, and it has relation with payment details table as shown in the diagram. Column status in TicketInformation table has values (booked, canceled) which indicate the situation of the ticket.

FlightInformation Table:

Source: the original planet or orbit that space craft departs from.

Destination: the planet or orbit that space craft going to.

The reason that we have price in FlightSchedules and amount in paymentDetail is that we can have multiple payment for a ticket.

Remark on the project: This project helped me a lot in reviewing the SQL skills. I wish there was more description for the database design parts.

Sincerely yours,

Seyed

In your typed report (pdf or docx only), please include the following items:

1. Complete SQL source codes in typed format (not screenshots allowed for this).
2. Comments for SQL codes (or for Screenshots, if codes aren't to be used for a particular question)
3. Full screenshots of the requested actions or each question including the total number of rows in result set and your name. 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.
4. Your cumulative remarks on the project [4 pts.]. The remarks should be specific thoughts and references you learned that are related with knowledge points.

Submit your report on Blackboard.