Introduction to Database Management Systems

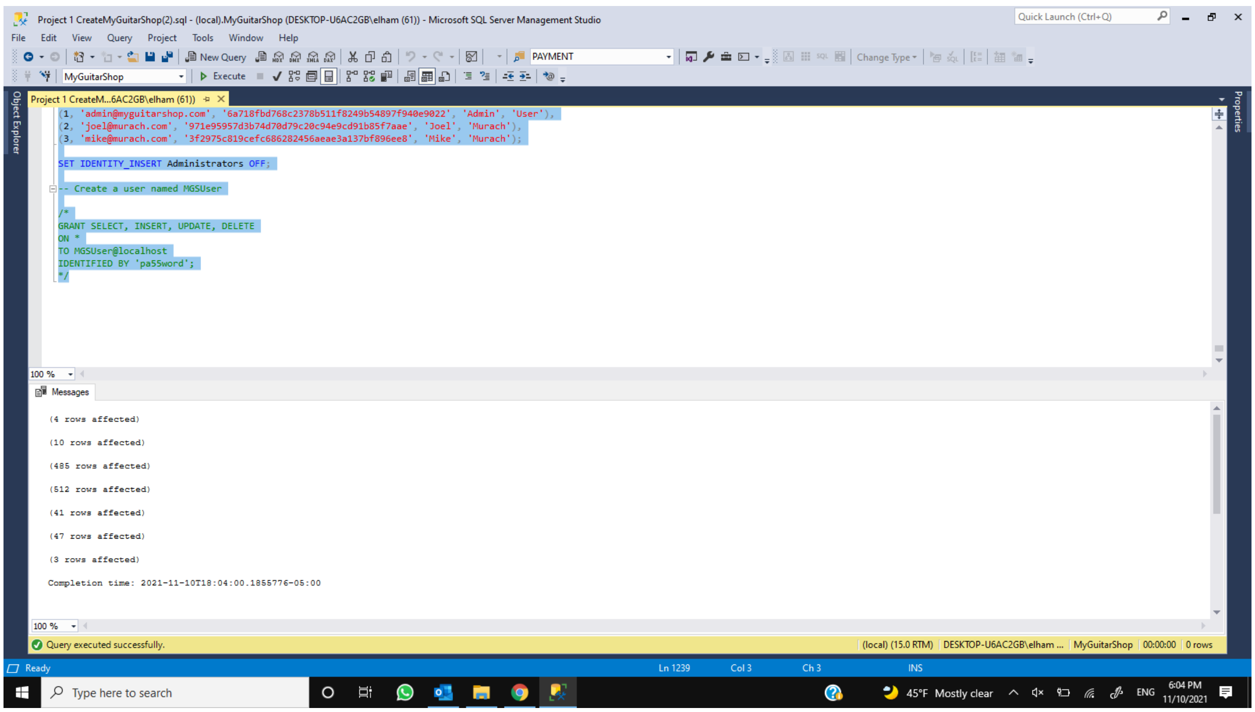
PROJECT 1

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.

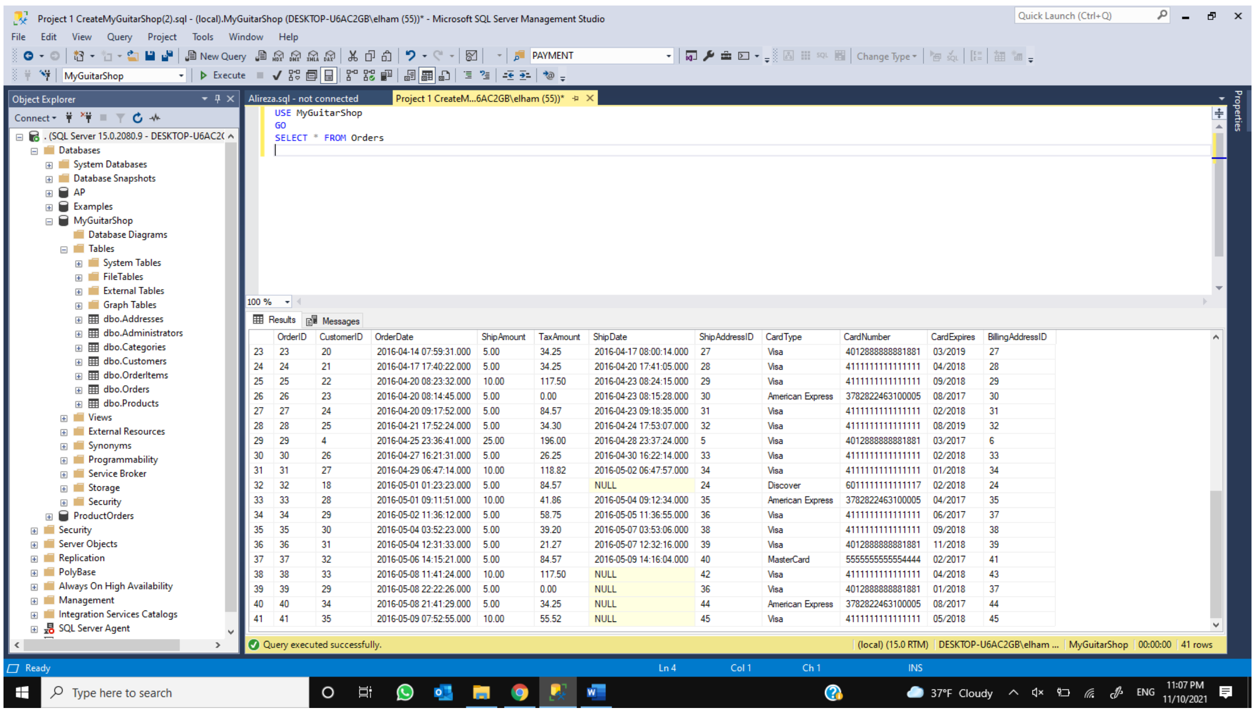
(1) Download CreateMyGuitarShop.sql from Project 1 directory on Blackboard and open it in SQL server management studio. Execute the entire script and show the message in the Message tab, indicating the script is executed successfully. A complete screenshot of execution result is required. Your screenshot 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.

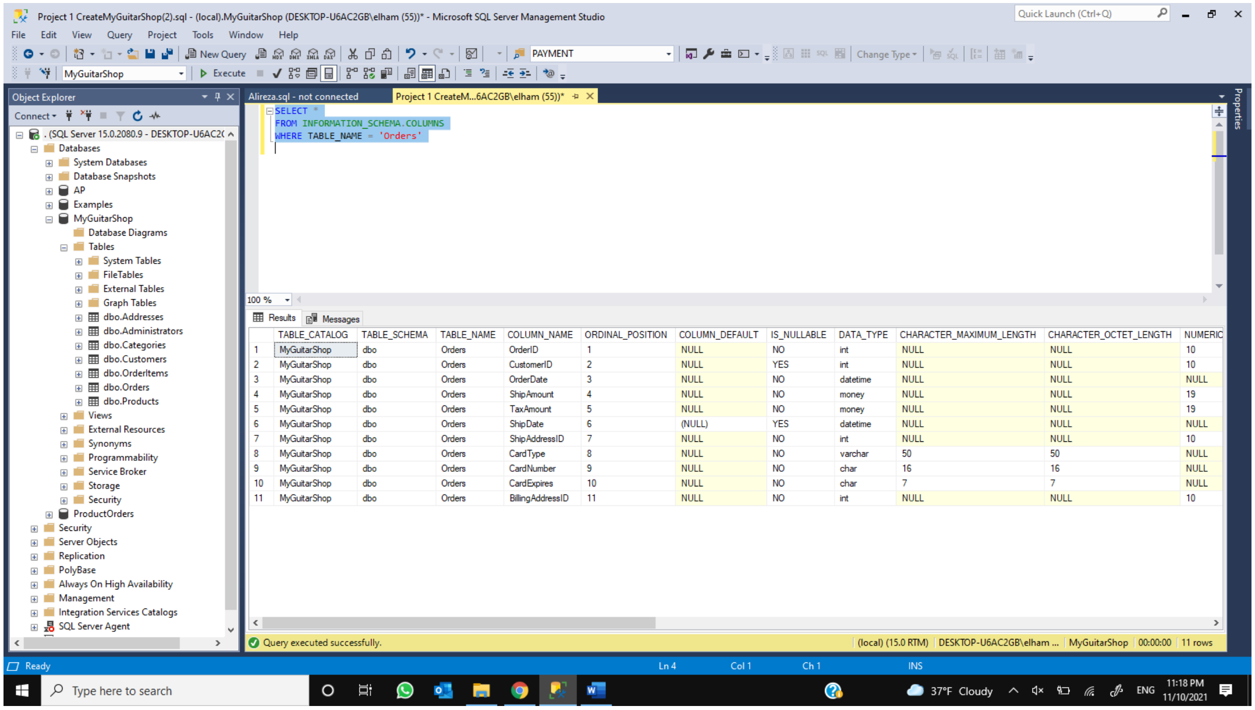
2. (2) Navigate through the database objects and view the column definitions for each table. Open a new Query Editor window. Show details in Customers table and Orders table using SELECT statement. Full screenshots of execution results are required as mentioned.

```sql
USE MyGuitarShop
GO
SELECT * FROM Orders
```



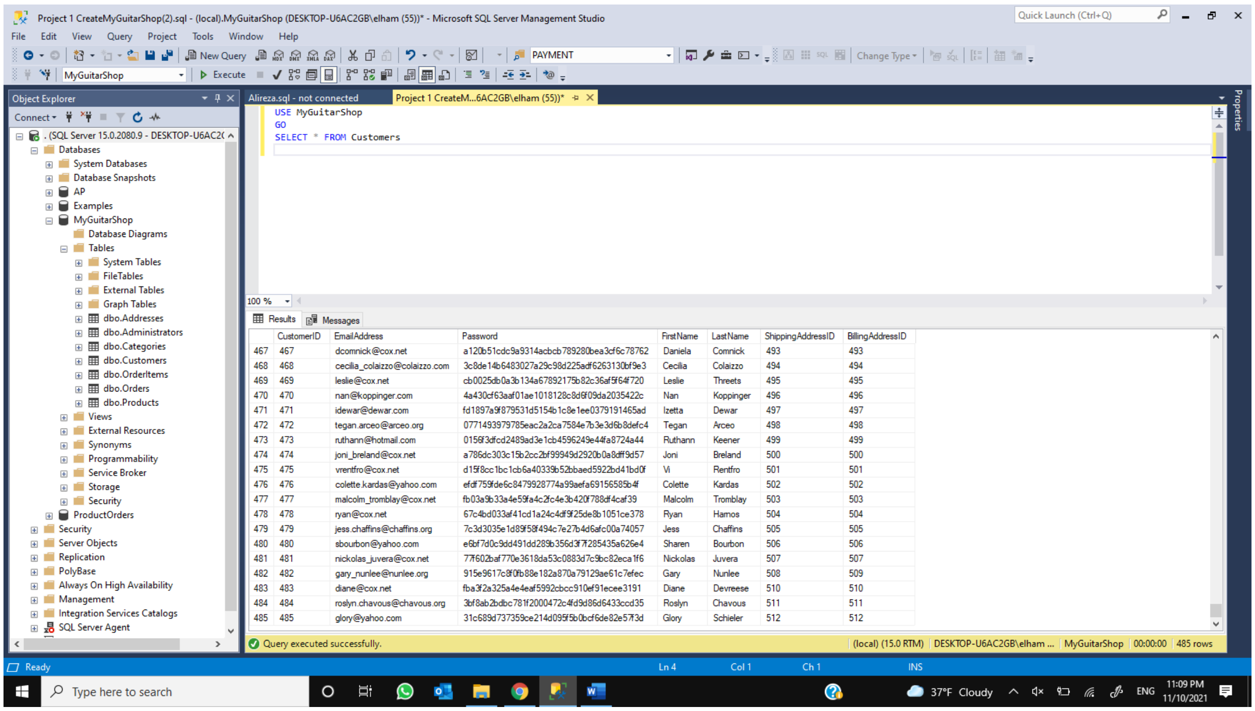
Remark: Here we can see the content of the Orders table.

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



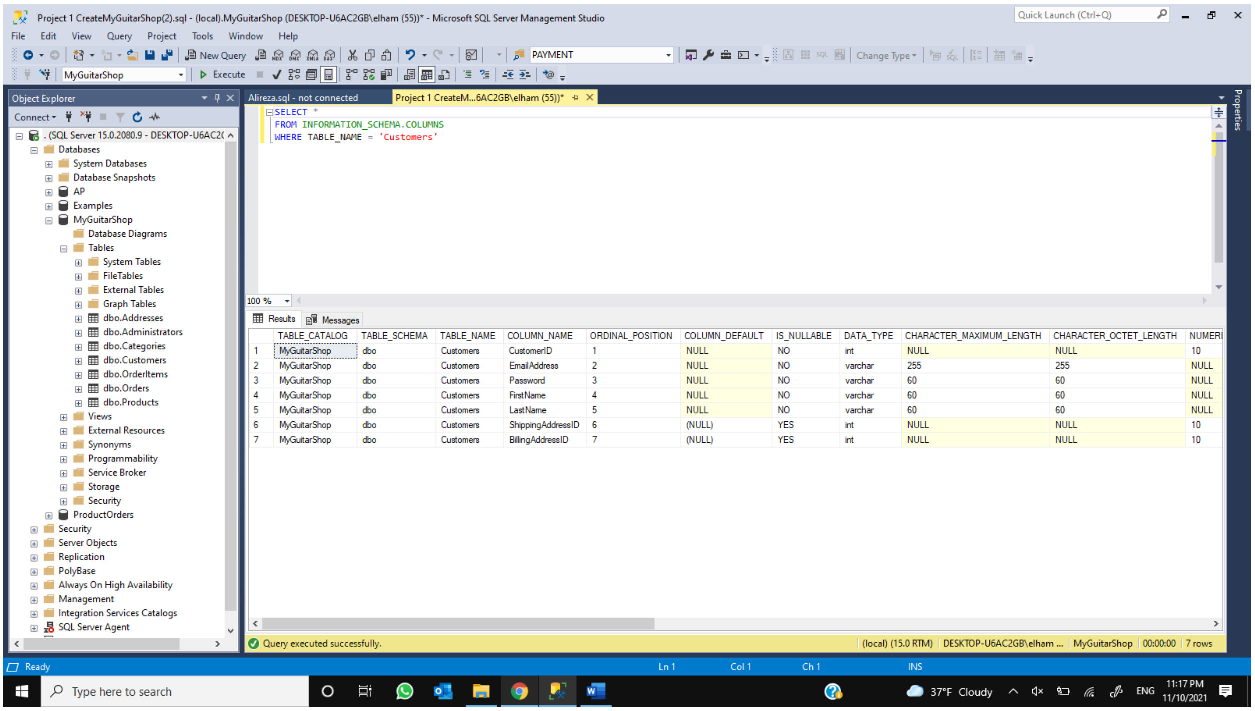
Remark: Here we can see the details for the Orders table.

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



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

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

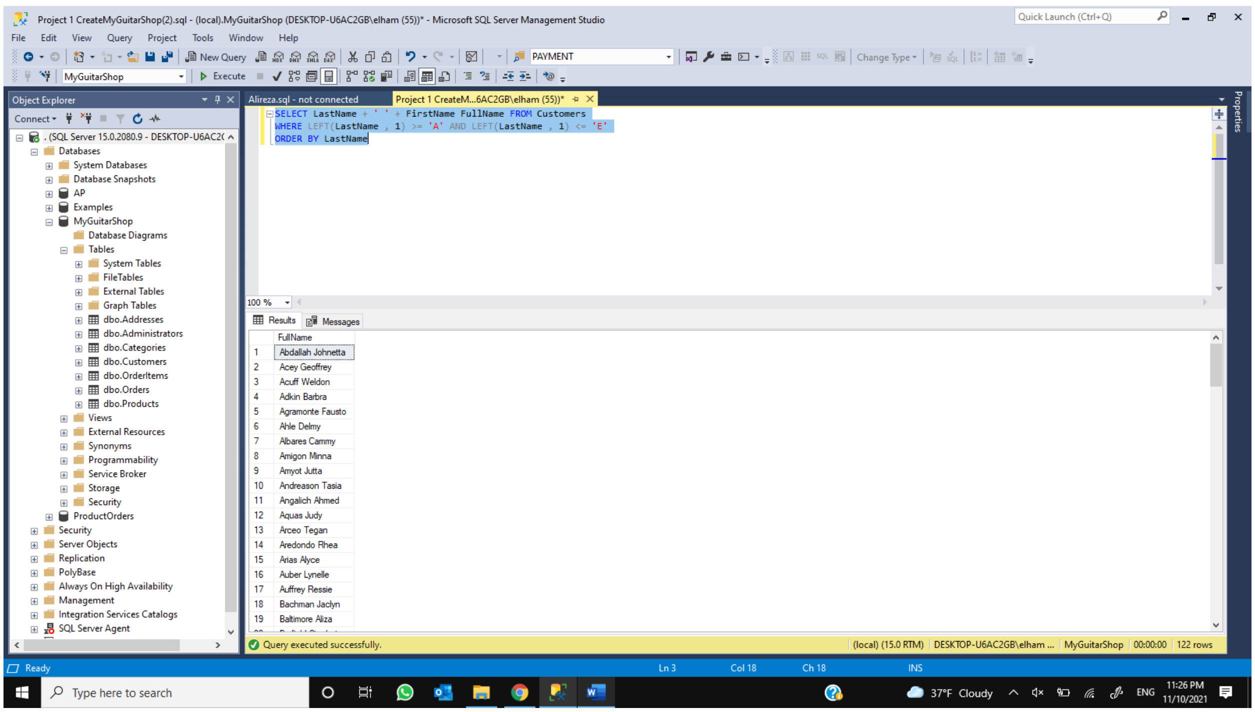


Remark: Here we can see the details for the Customers table.

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 aescending 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:

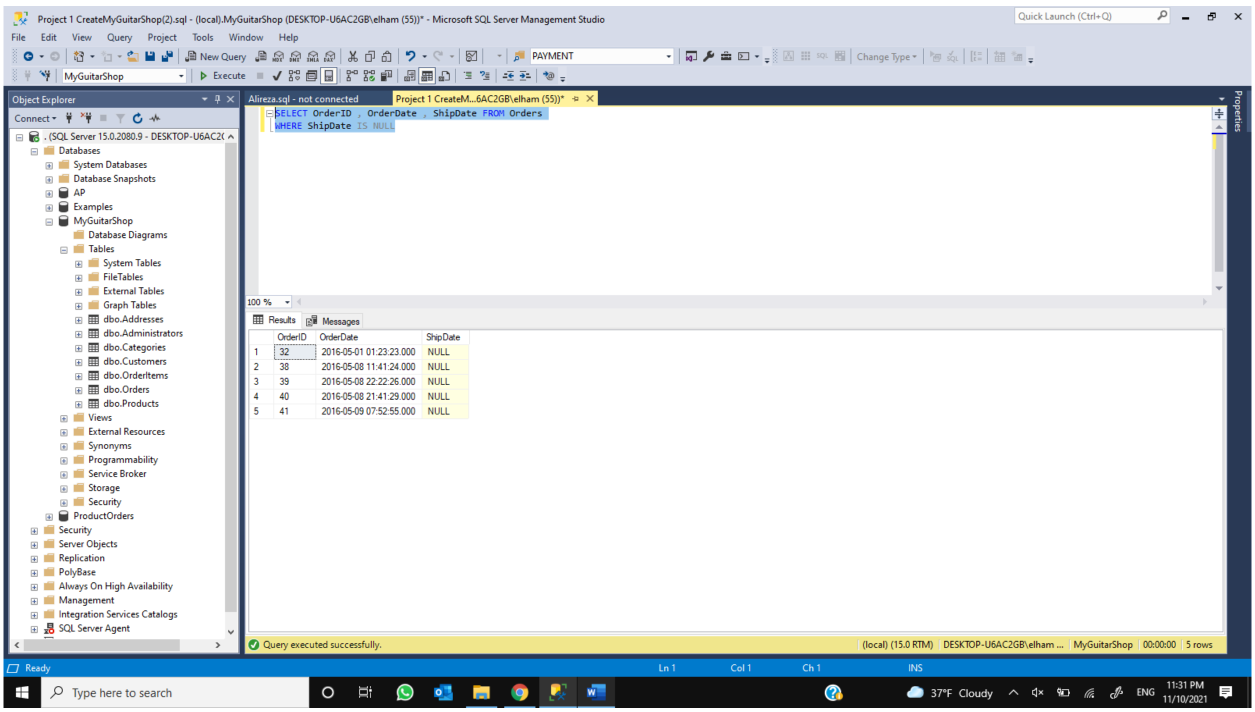
OrderID The OrderID column

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
```

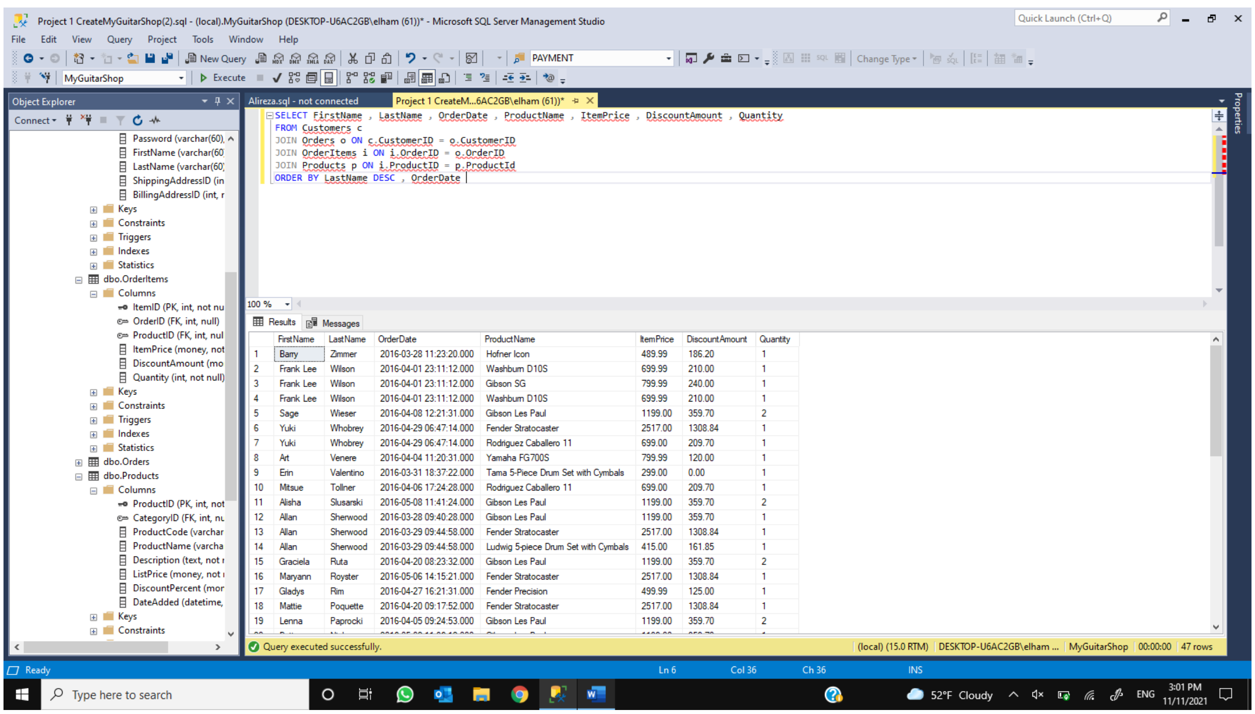


Remark: Here we can see the orders that has not been shipped or their shipped date has not been entered to the database (shipped date is null).

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.

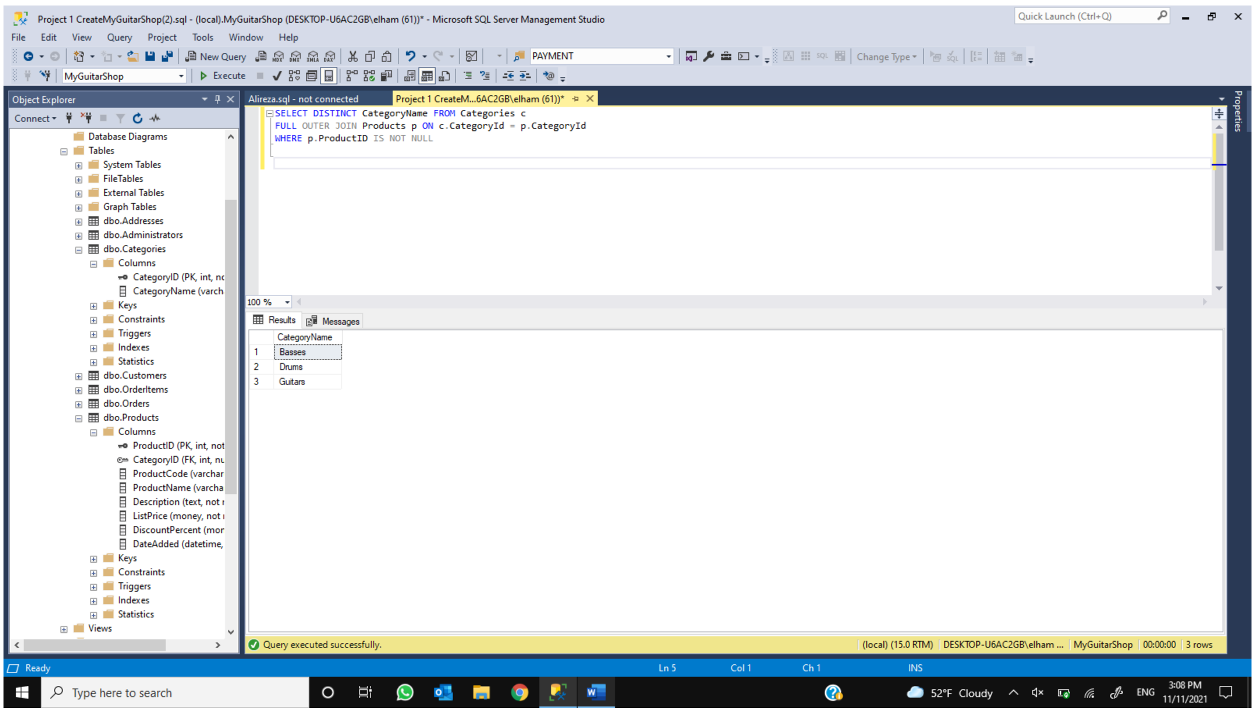
```
SELECT FirstName , LastName , OrderDate , ProductName , ItemPrice , DiscountAmount ,
Quantity
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN OrderItems i ON i.OrderID = o.OrderID
JOIN Products p ON i.ProductID = p.ProductId
ORDER BY LastName DESC , OrderDate
```



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.

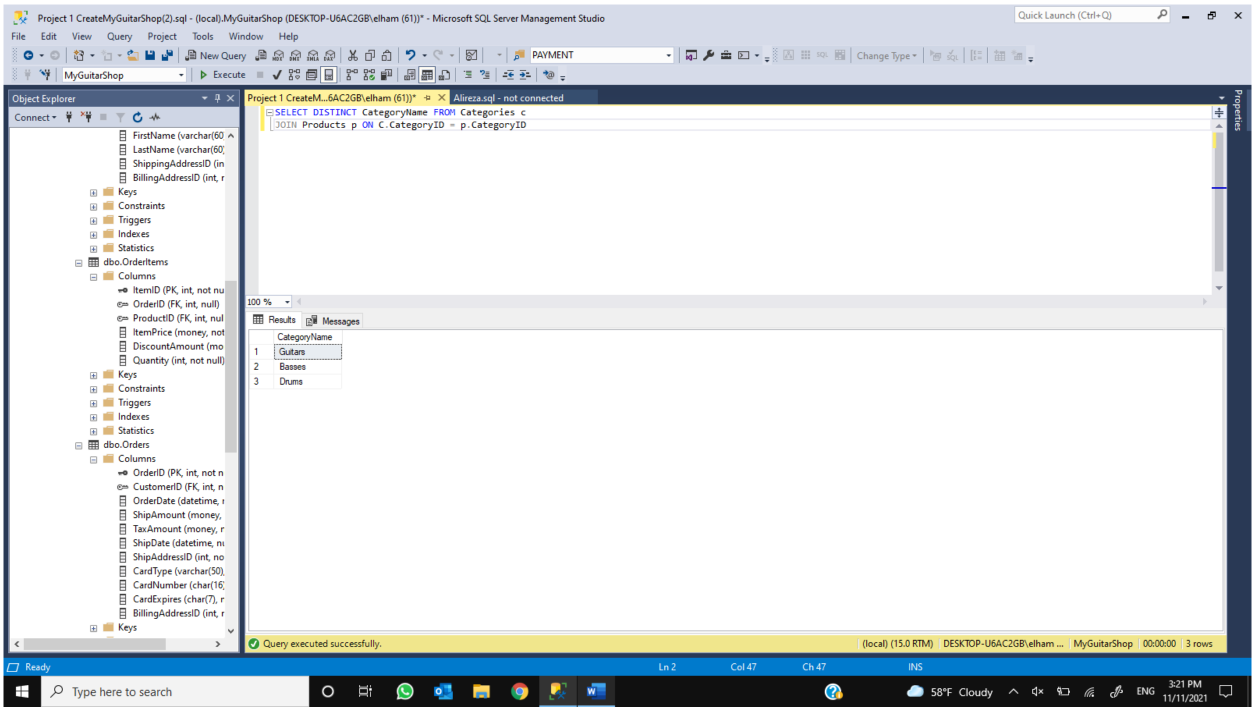
2. [4] Write a SELECT statement that returns CategoryName column from the Categories table. Return one row for each category that has been used. (Hint: Use an outer join and only return rows where the ProductID column does not contain a null value.)

```sql
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.

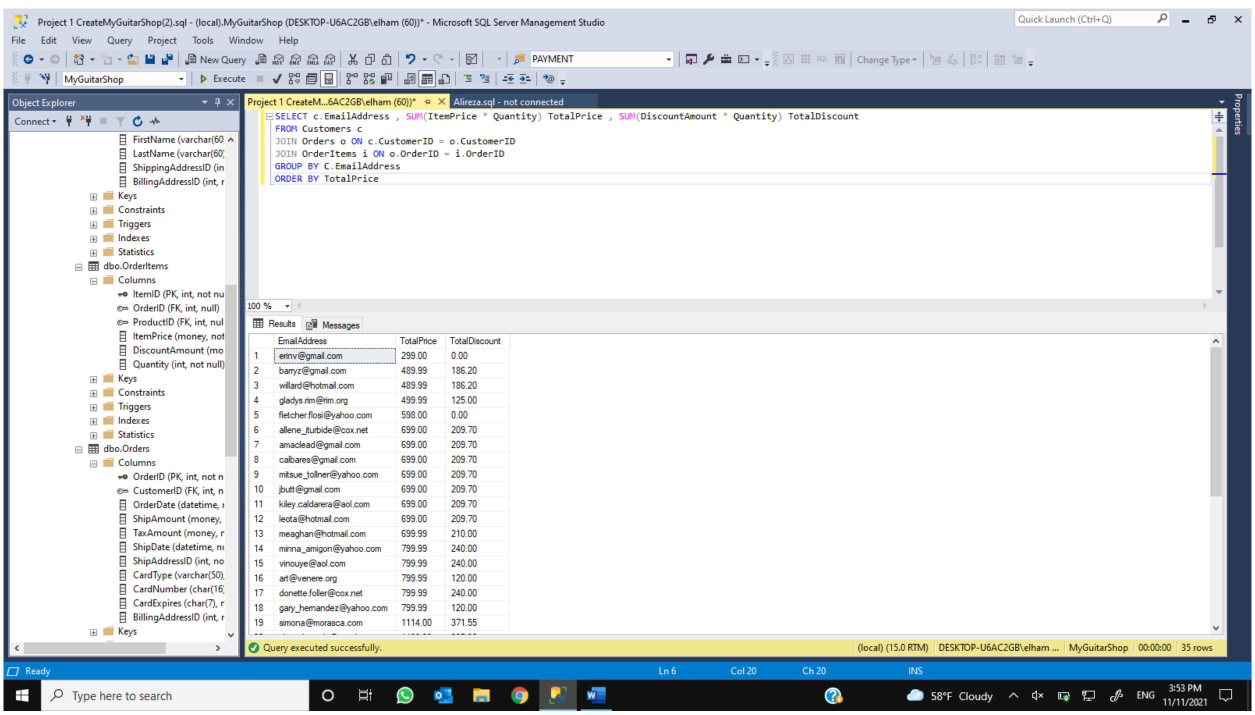
3. [4] Write a SELECT statement that returns one row for each customer that has orders with these columns:

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
```



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

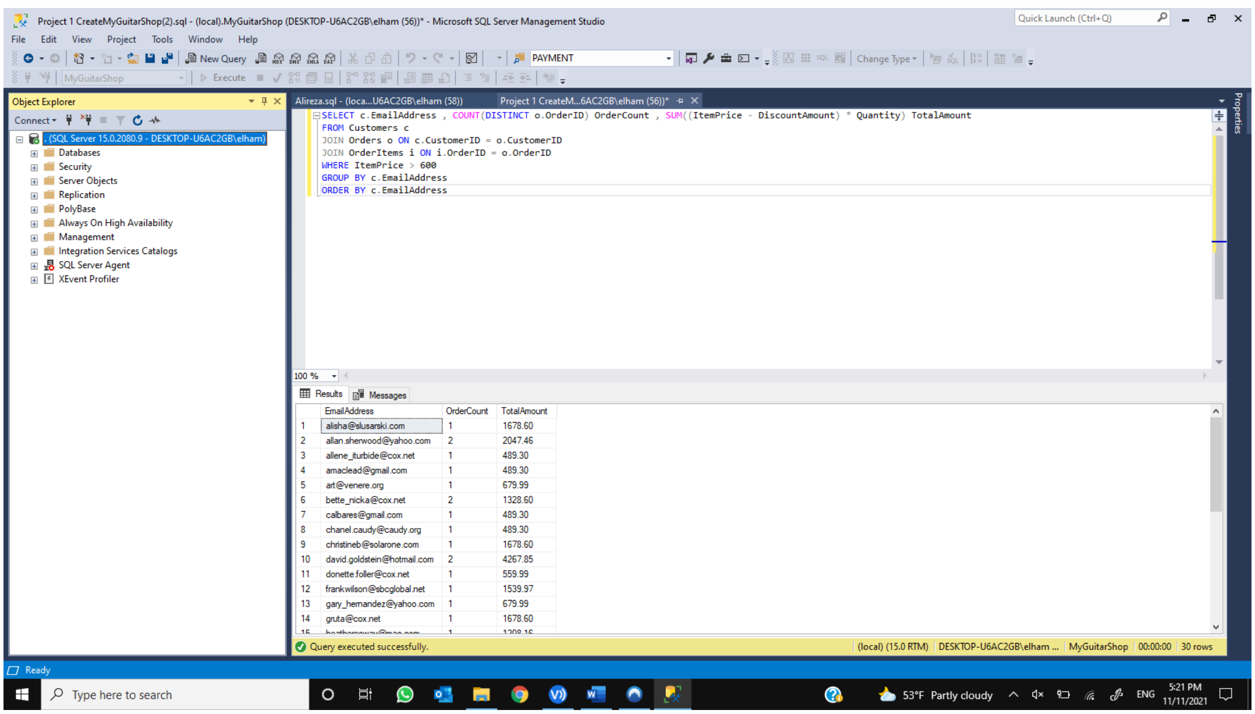
4. [4] Write a SELECT statement that returns one row for each customer that has orders with these columns:

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 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.

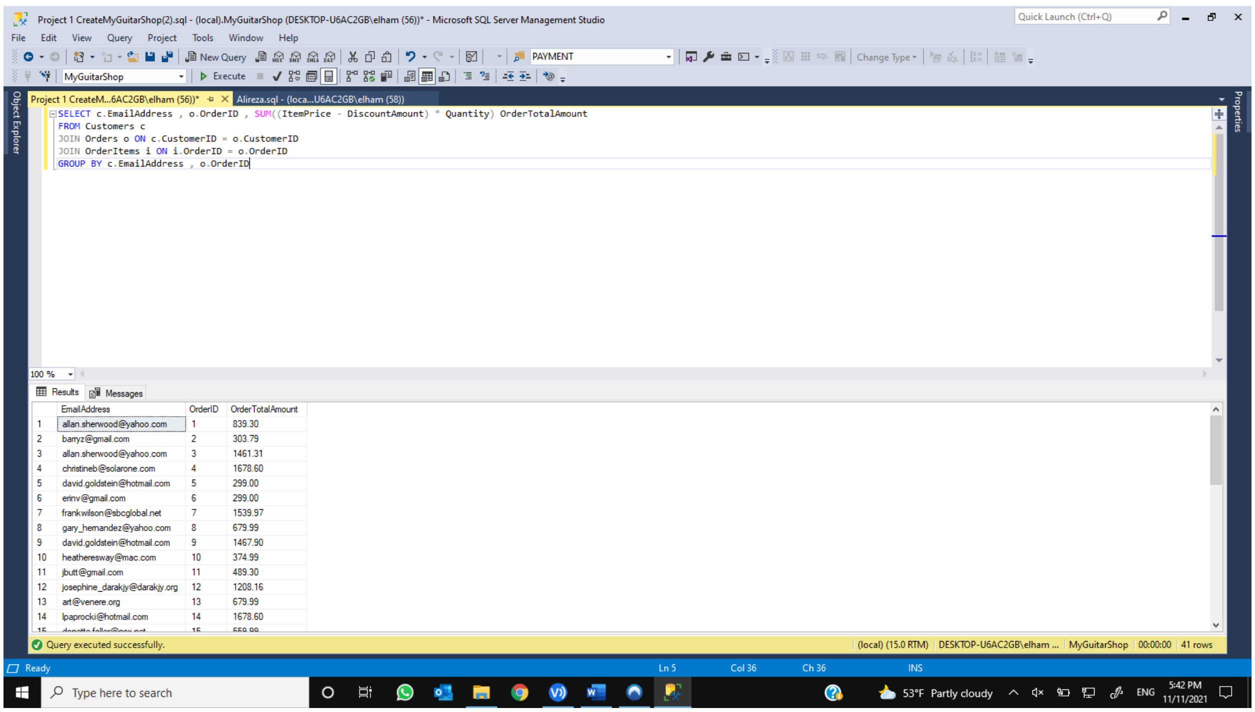
```
SELECT c.EmailAddress , COUNT(DISTINCT o.OrderID) OrderCount , SUM((ItemPrice -
DiscountAmount) * Quantity) TotalAmount
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN OrderItems i ON i.OrderID = o.OrderID
WHERE ItemPrice > 600
GROUP BY c.EmailAddress
ORDER BY c.EmailAddress
```



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)

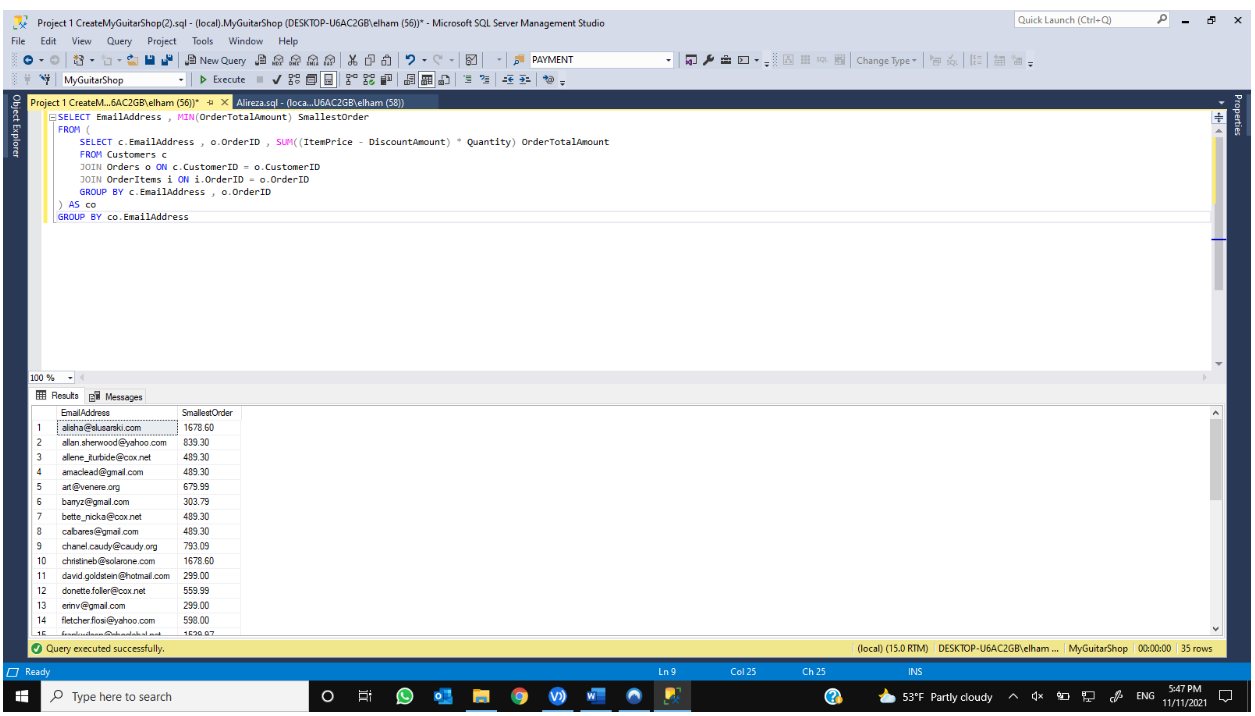
```
SELECT c.EmailAddress , o.OrderID , SUM((ItemPrice - DiscountAmount) * Quantity)
OrderTotalAmount
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN OrderItems i ON i.OrderID = o.OrderID
GROUP BY c.EmailAddress , o.OrderID
```



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.

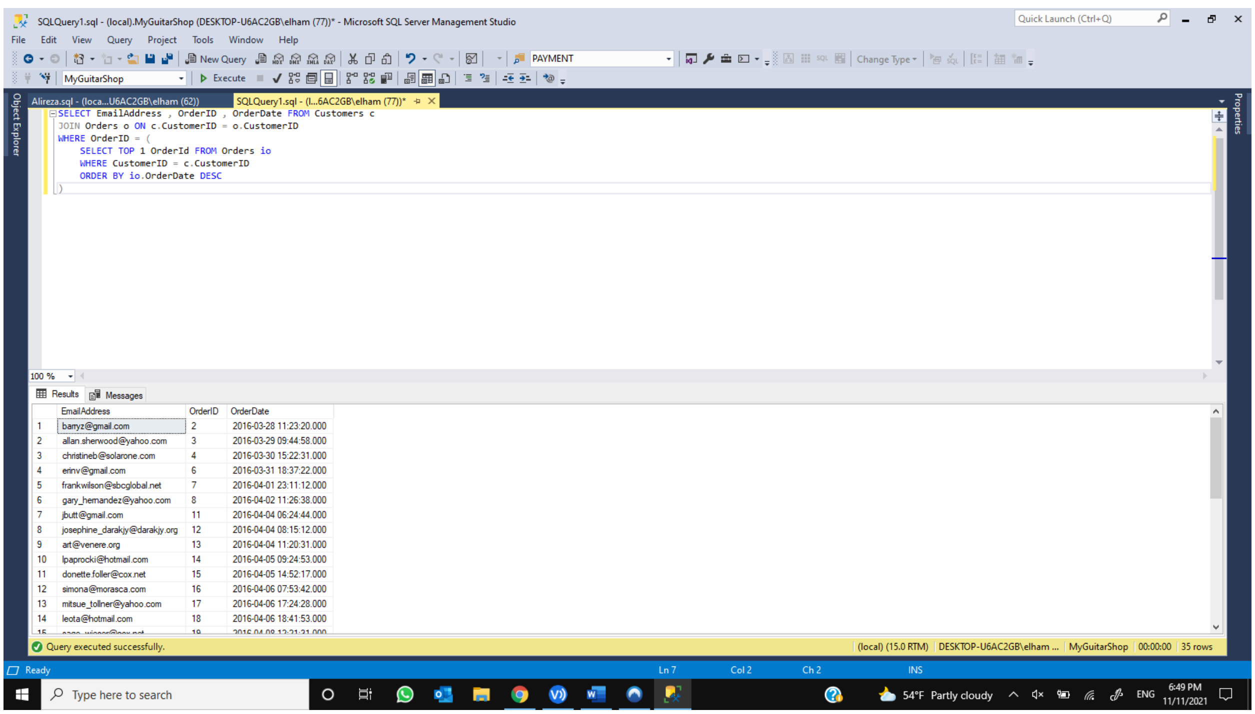
```
SELECT EmailAddress , MIN(OrderTotalAmount) SmallestOrder
FROM (
        SELECT c.EmailAddress , o.OrderID , SUM((ItemPrice - DiscountAmount) * Quantity)
OrderTotalAmount
        FROM Customers c
        JOIN Orders o ON c.CustomerID = o.CustomerID
        JOIN OrderItems i ON i.OrderID = o.OrderID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



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
)
```



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:

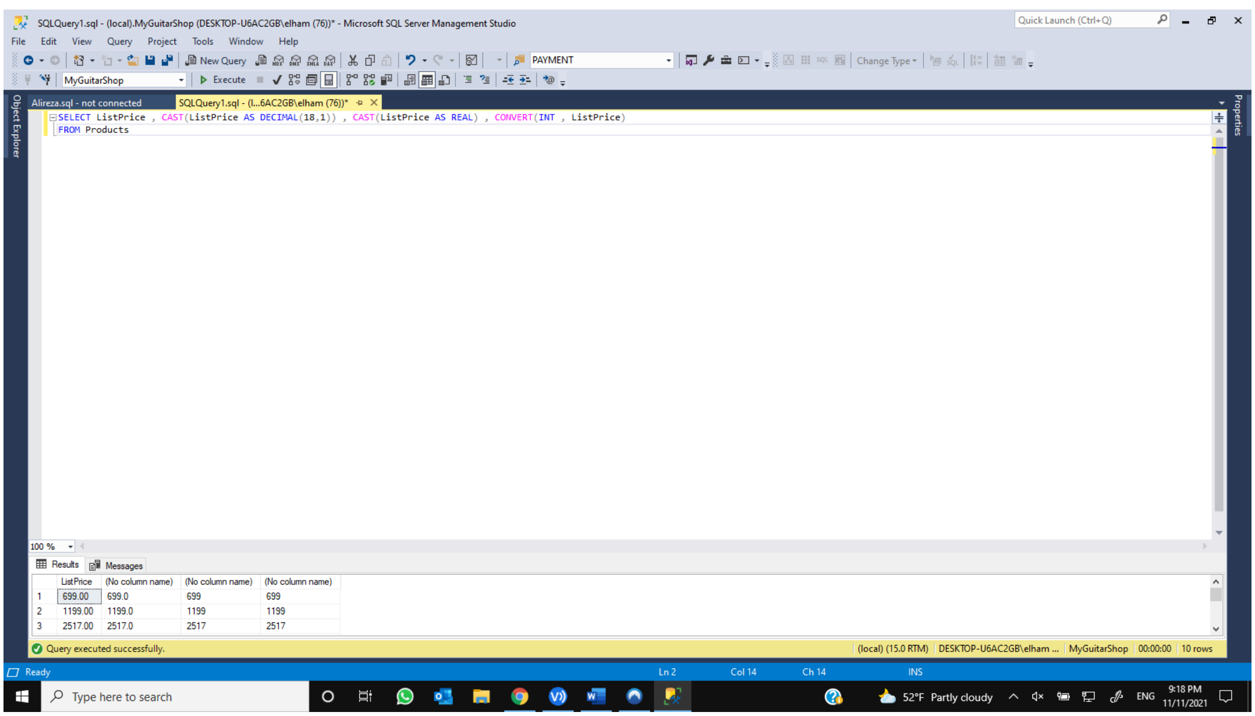
a) The ListPrice column

b) 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:

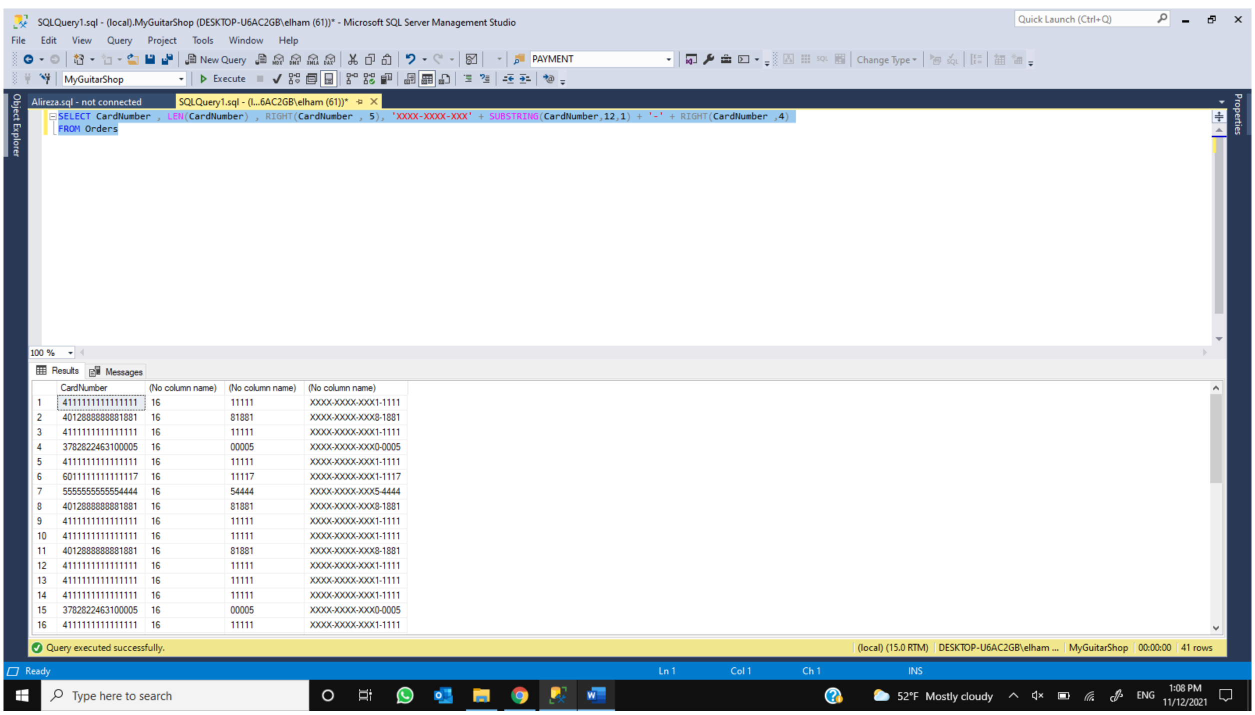
a) The CardNumber column

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-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

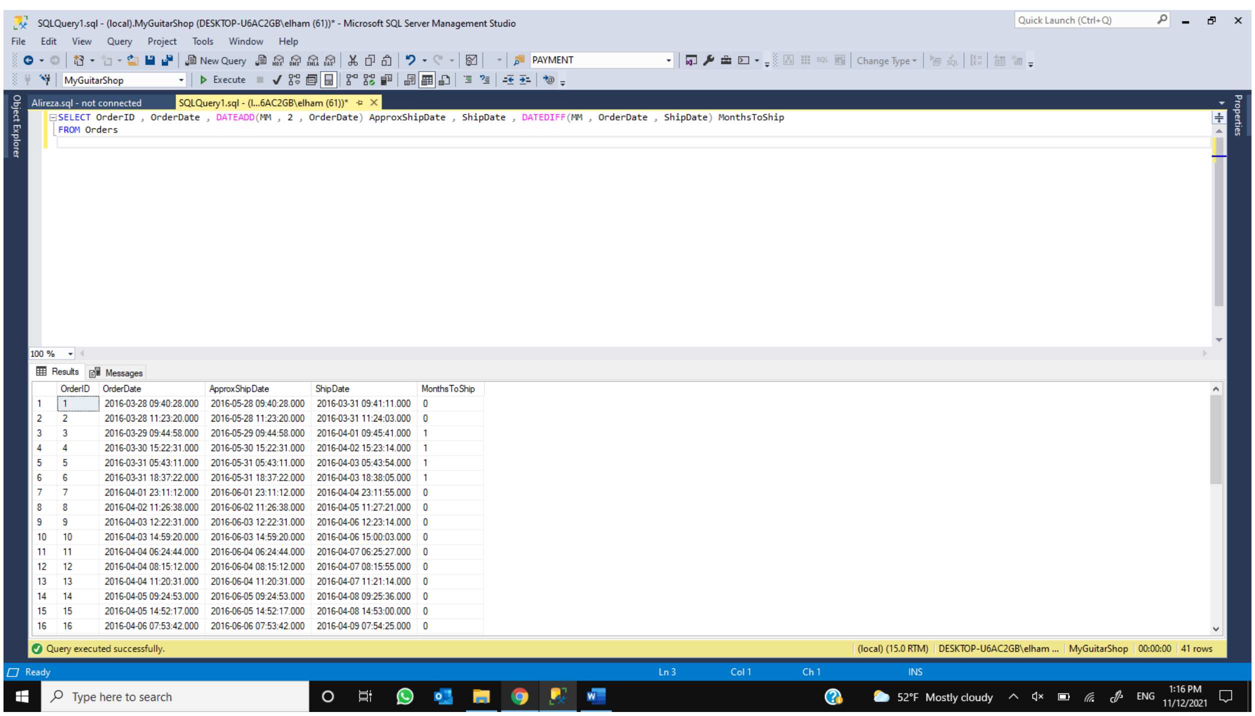
b) The OrderDate column

c) A column named ApproxShipDate that's calculated by adding 2 months to the OrderDate column

d) The ShipDate column

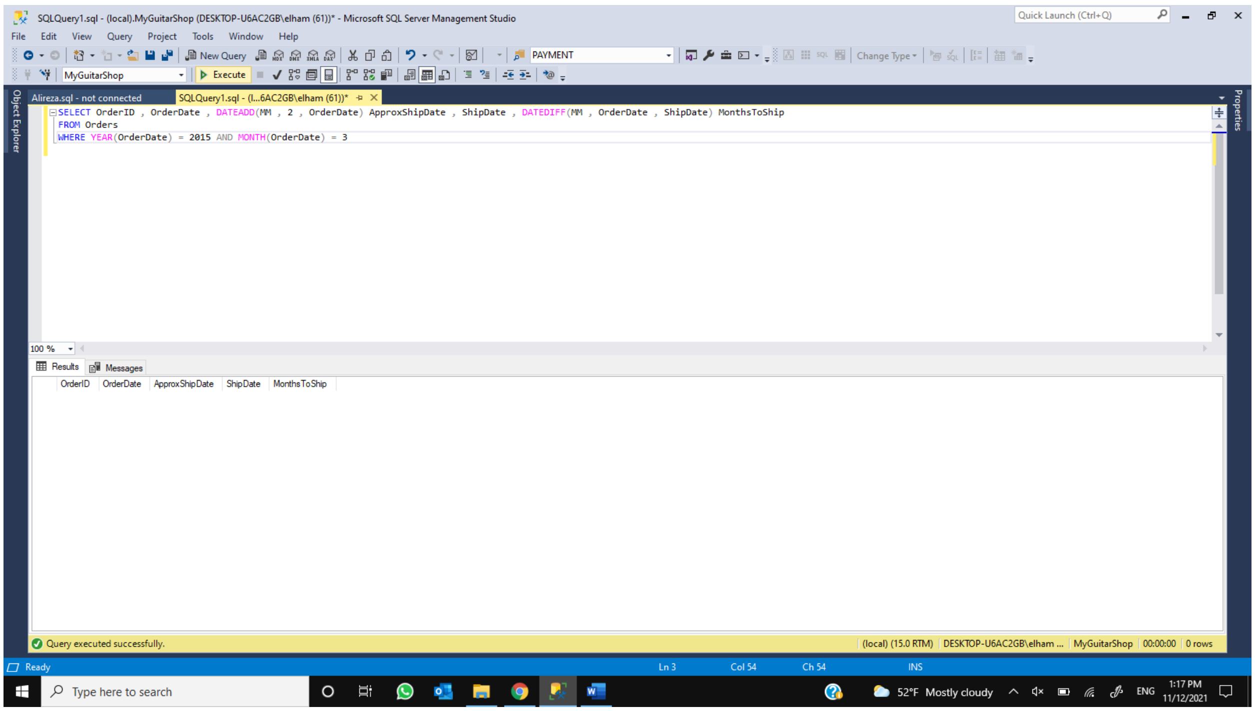
e) A column named MonthsToShip that shows the number of months between the order date and the ship date When you have this working, add a WHERE clause that retrieves just the orders for March 2015.

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



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:

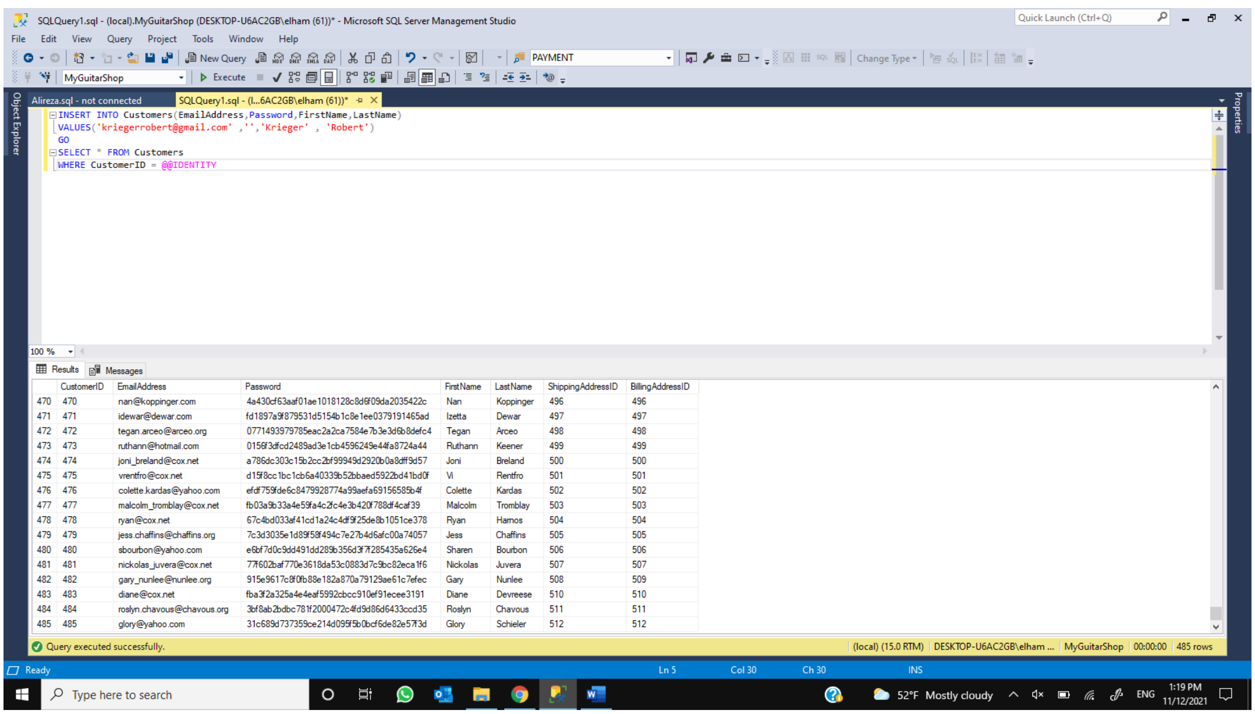
EmailAddress: kriegerrobert@gmail.com

Password: (empty string)

FirstName: Krieger

LastName: Robert Use a column list for this statement.

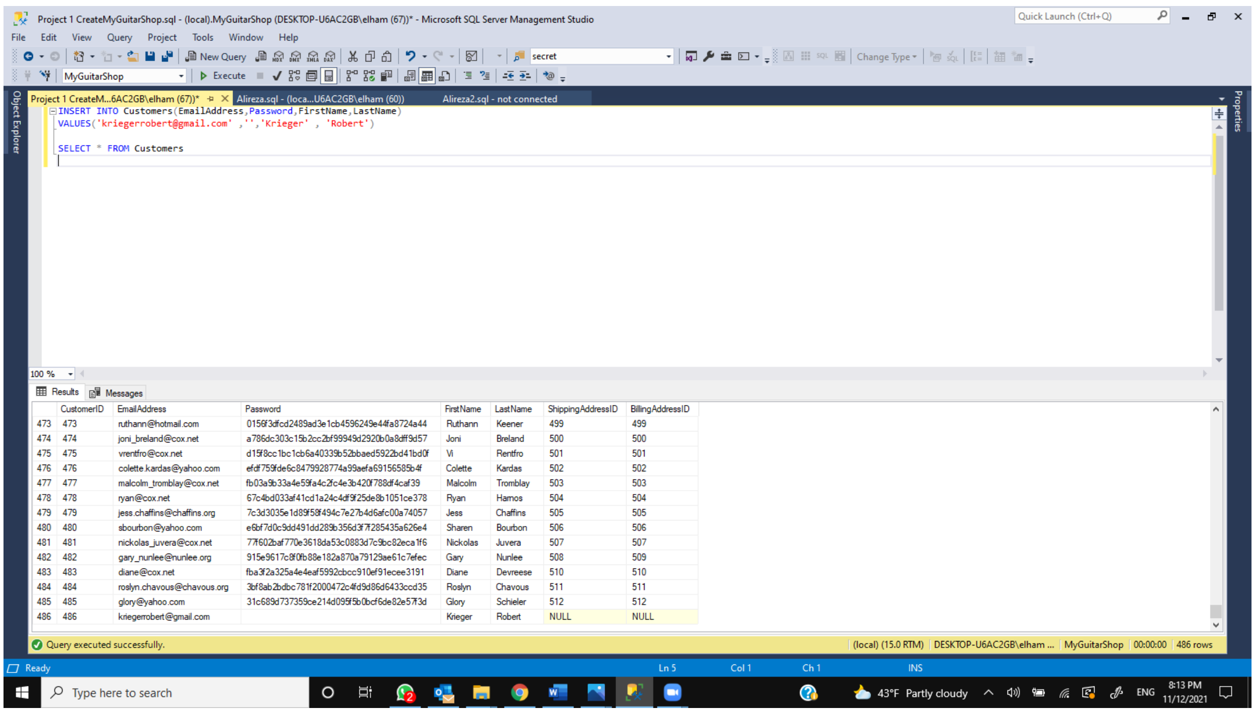
SELECT * FROM Customers



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.

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

SELECT * FROM Customers
```

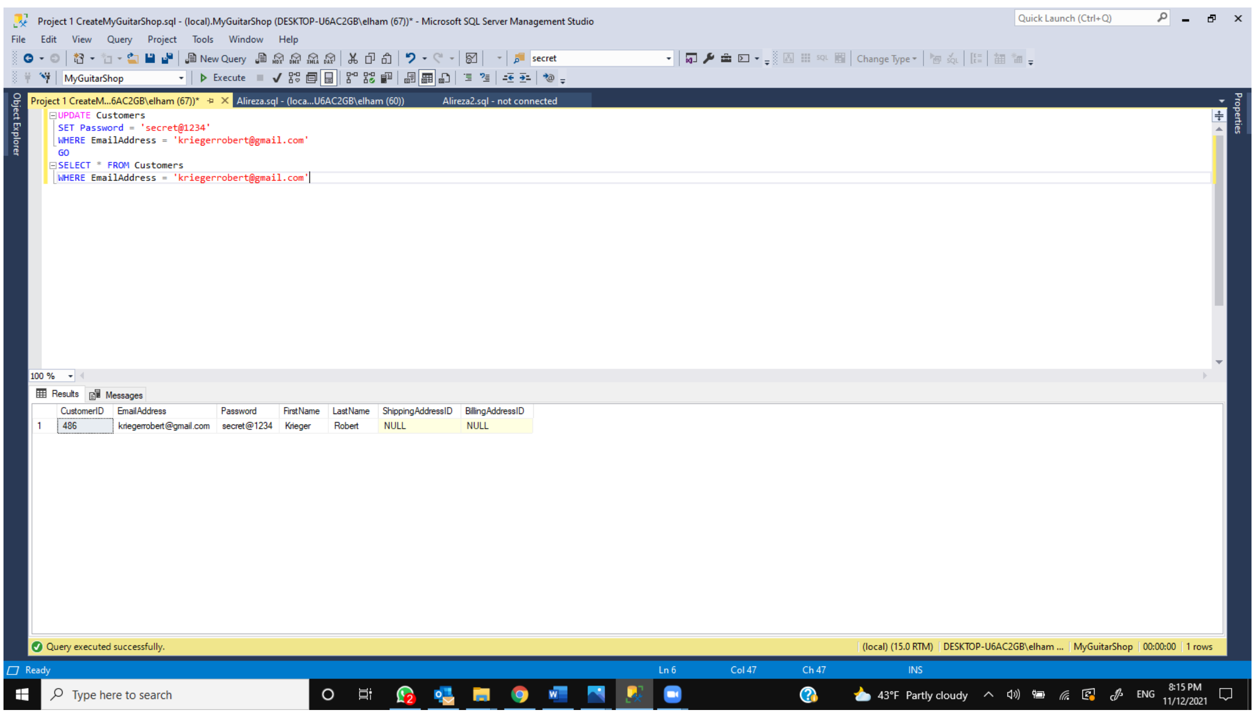


Remark: now we added the 486th row with customer information.

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.

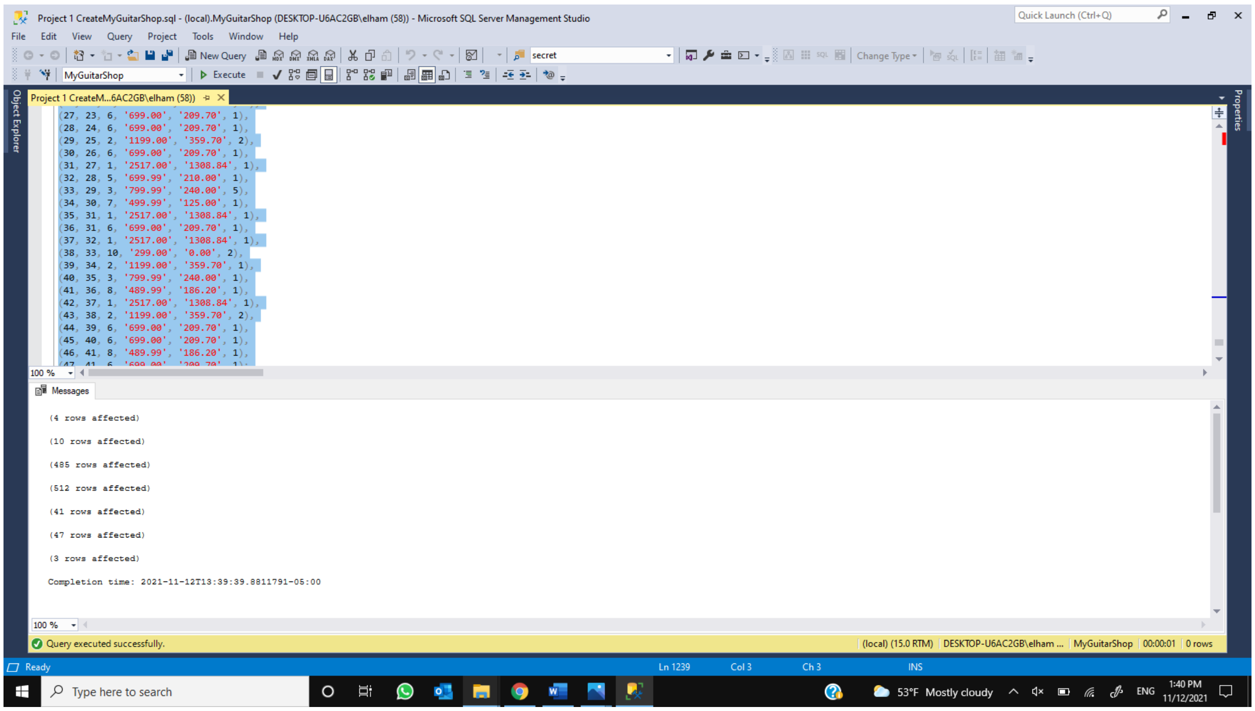
```
UPDATE Customers
SET Password = 'secret@1234'
WHERE EmailAddress = 'kriegerrobert@gmail.com'
GO
SELECT * FROM Customers
WHERE EmailAddress = 'kriegerrobert@gmail.com'
```



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

D. Advanced SQL skills (views/stores 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.

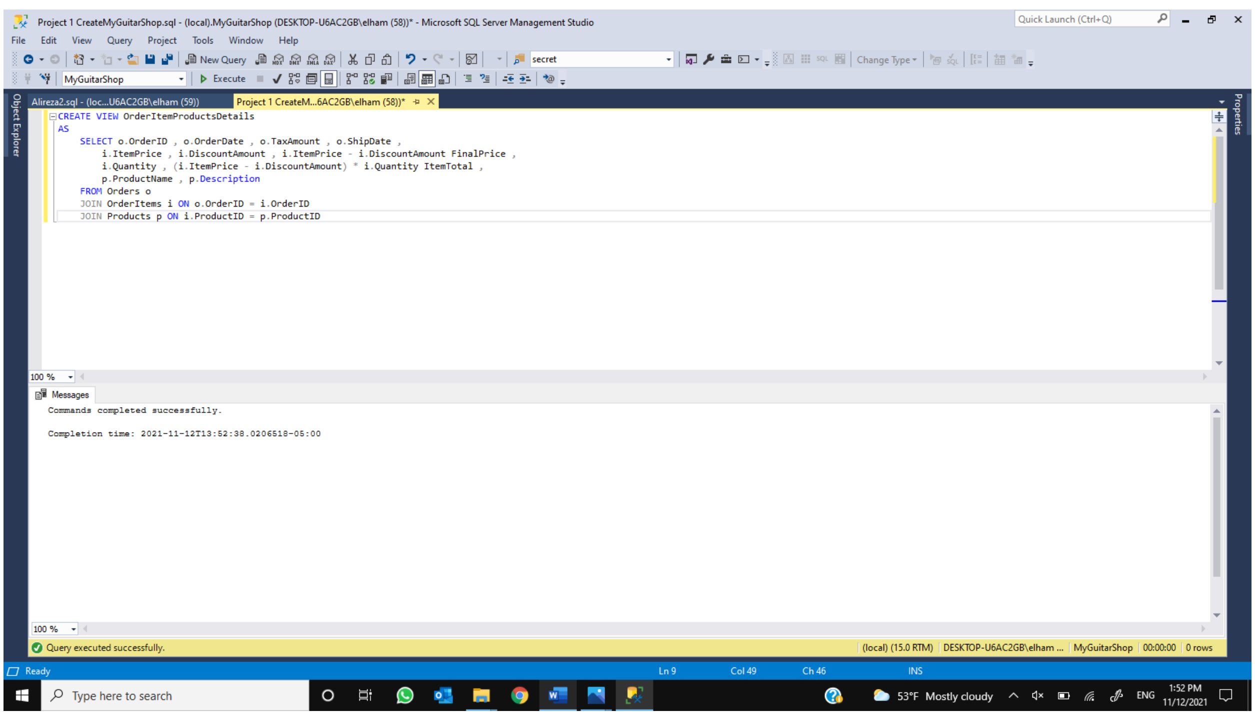
1. [4] Create a view named OrderItemProductsDetails that returns columns from the Orders, OrderItems, and Products tables.

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).

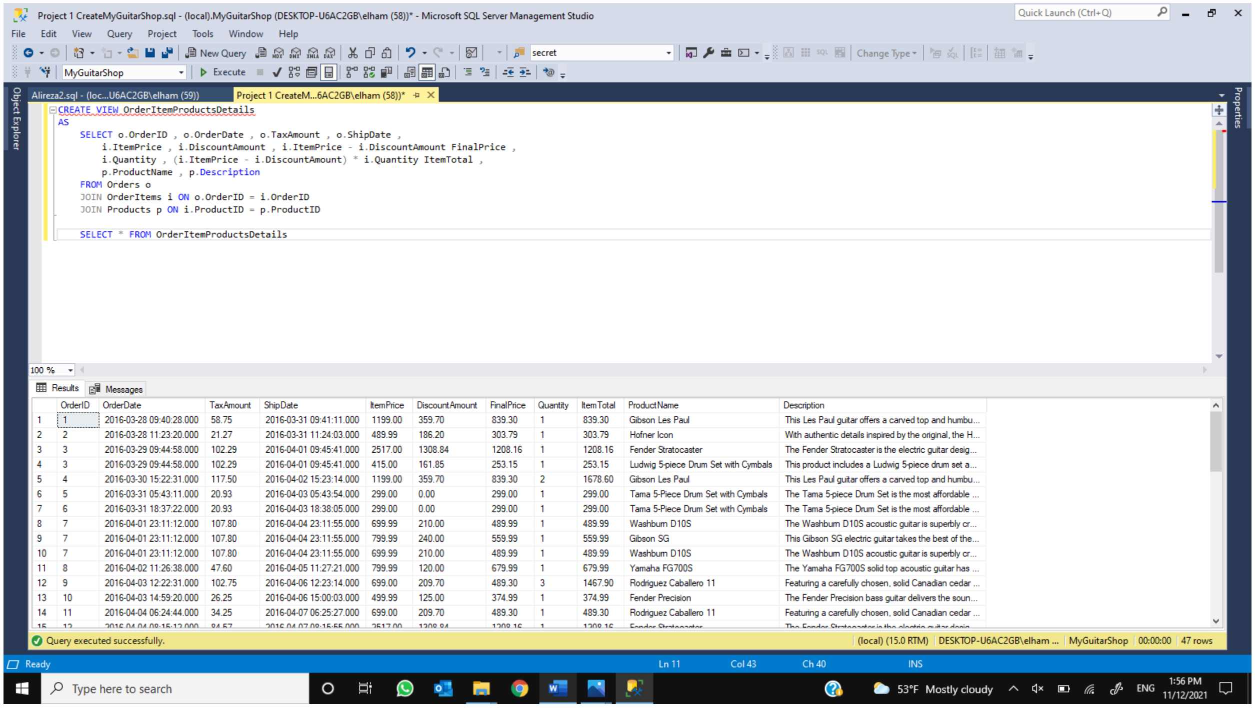
c) This view should return the ProductName and Description column from the Products table.

```
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 nest screenshot.

SELECT * FROM OrderItemProductsDetails

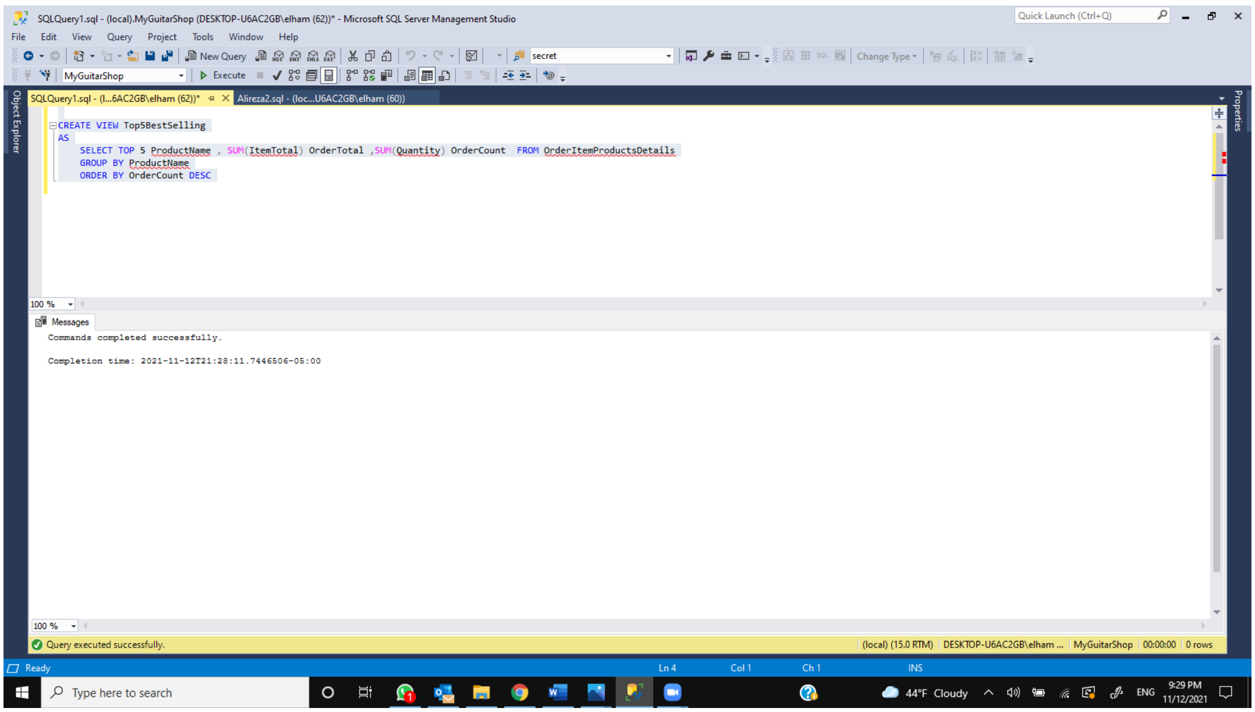


	OrderID	OrderDate	TaxAmount	ShipDate	ItemPrice	DiscountAmount	FinalPrice	Quantity	ItemTotal	ProductName	Description
1	1	2016-03-28 09:40:28.000	58.75	2016-03-31 09:41:11.000	1199.00	359.70	839.30	1	839.30	Gibson Les Paul	This Les Paul guitar offers a carved top and humbu...
2	2	2016-03-28 11:23:20.000	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	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	3	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	4	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	5	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	6	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	7	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	7	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	7	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	8	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	9	2016-04-03 12:22:31.000	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	10	2016-04-03 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	11	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.

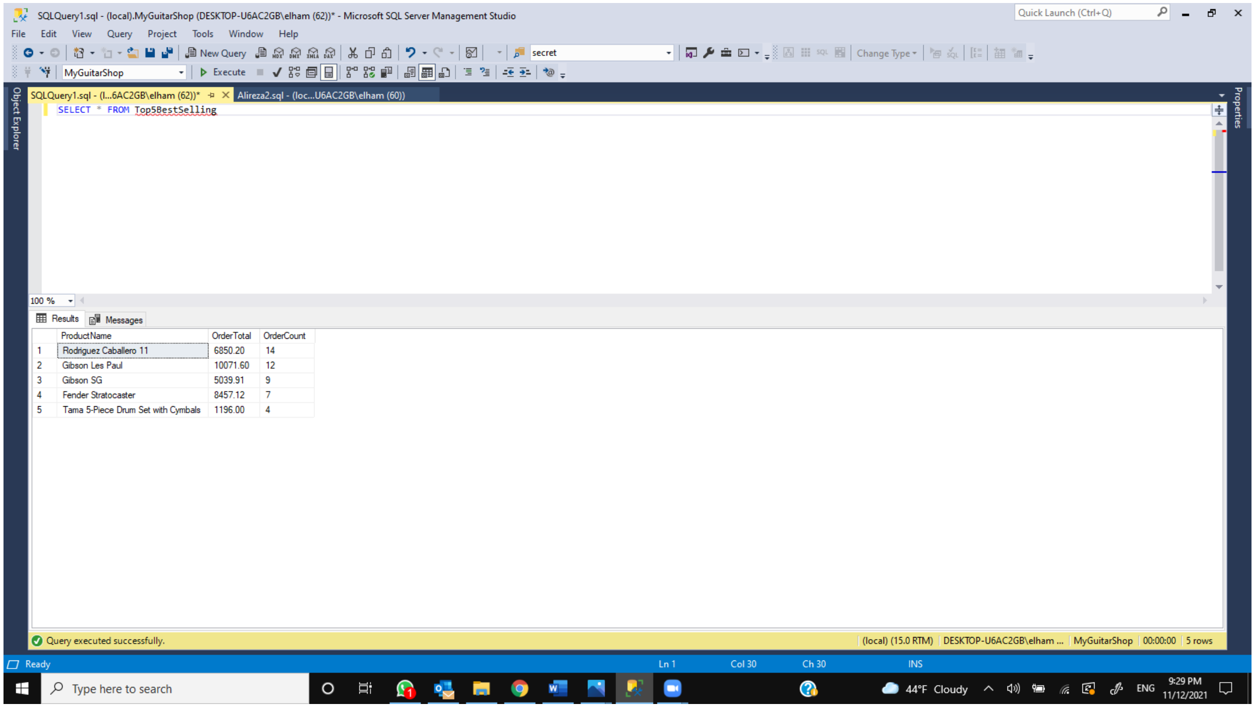
2. [5] Create a view named Top5BestSelling that uses the view you created in Section D Question 1. This view should return some summary information about five best selling products. Each row should include these columns: ProductName, OrderTotal (the total sales for the product) and OrderCount (the number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

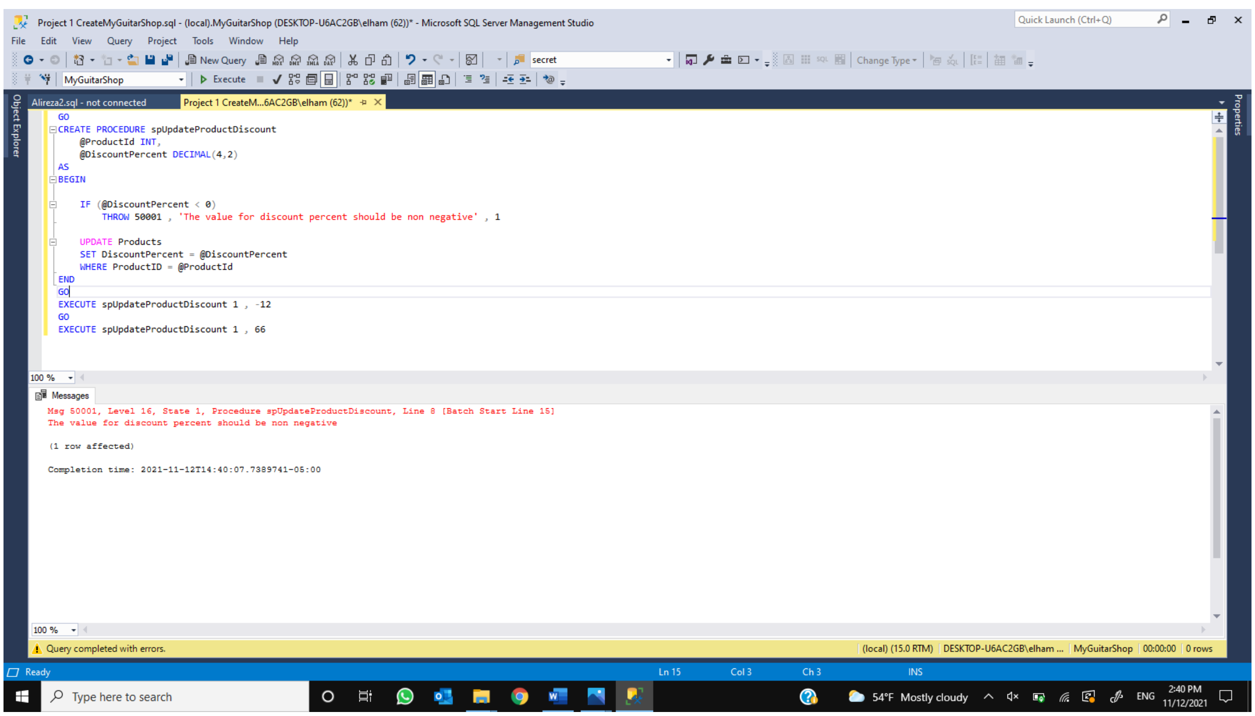
3. [5] Write a script that creates and calls a stored procedure named spUpdateProductDiscount that updates the DiscountPercent column in the Products table. This procedure should have one parameter for the product ID and another for the discount percent.

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

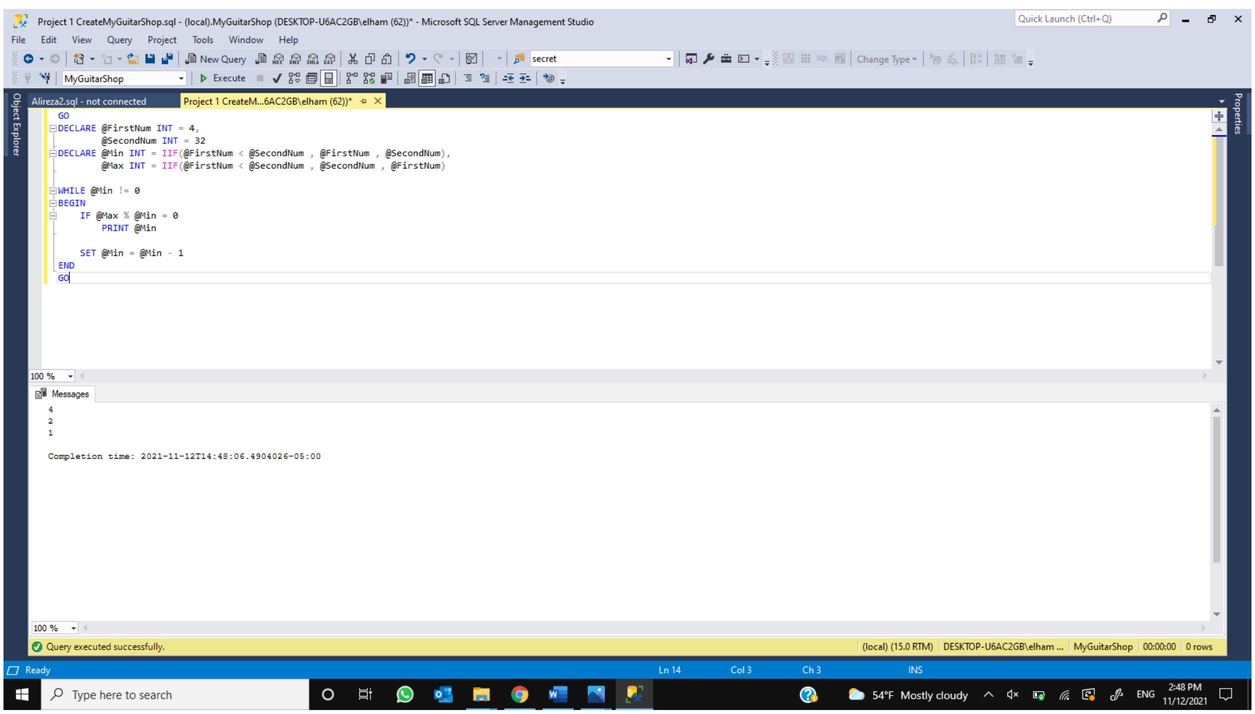
2

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
```



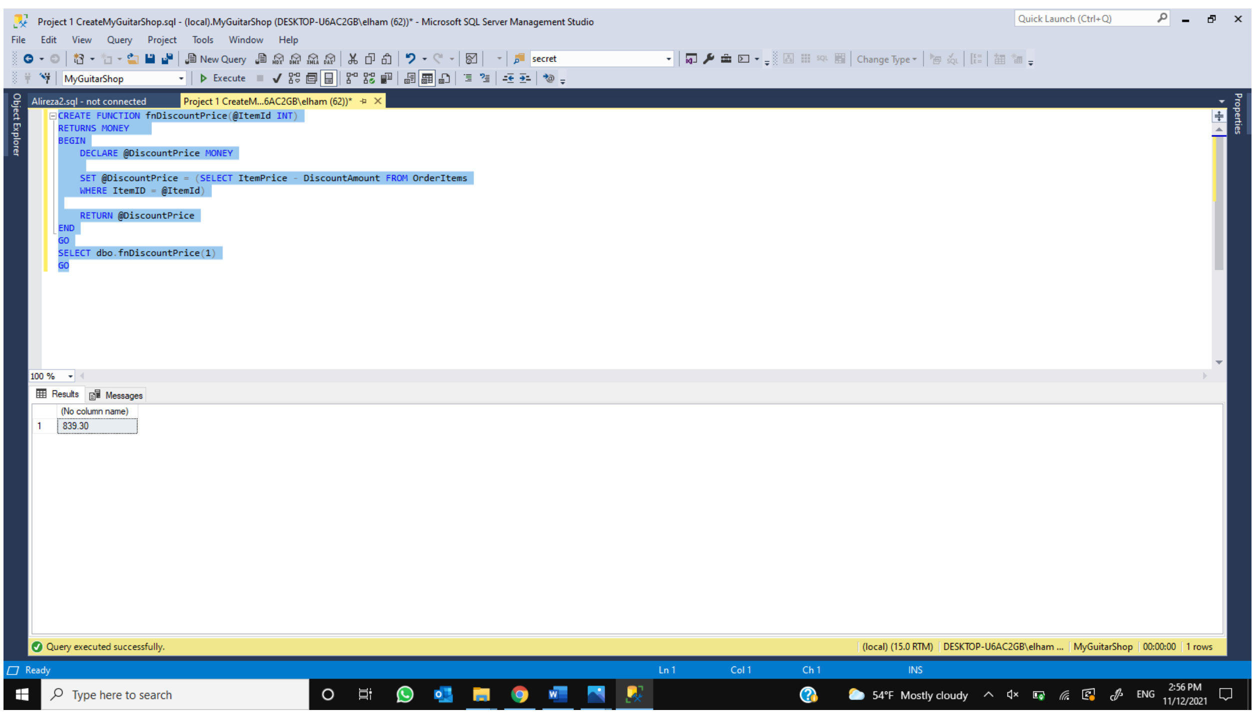
Remark: I created a script here which calculates the common factors between 4 and 32

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
```



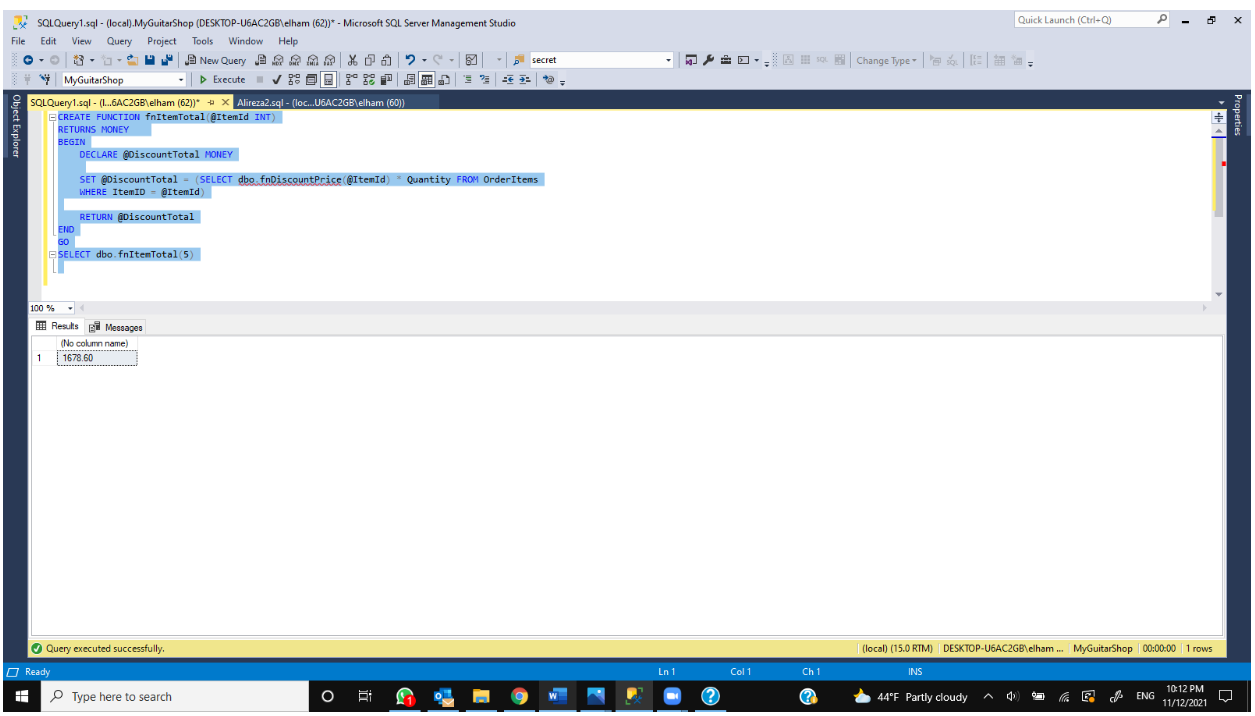
Remark: we created a function that calculates the discount price of an item. By entering 1 for item ID you will get 839.30.

(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.

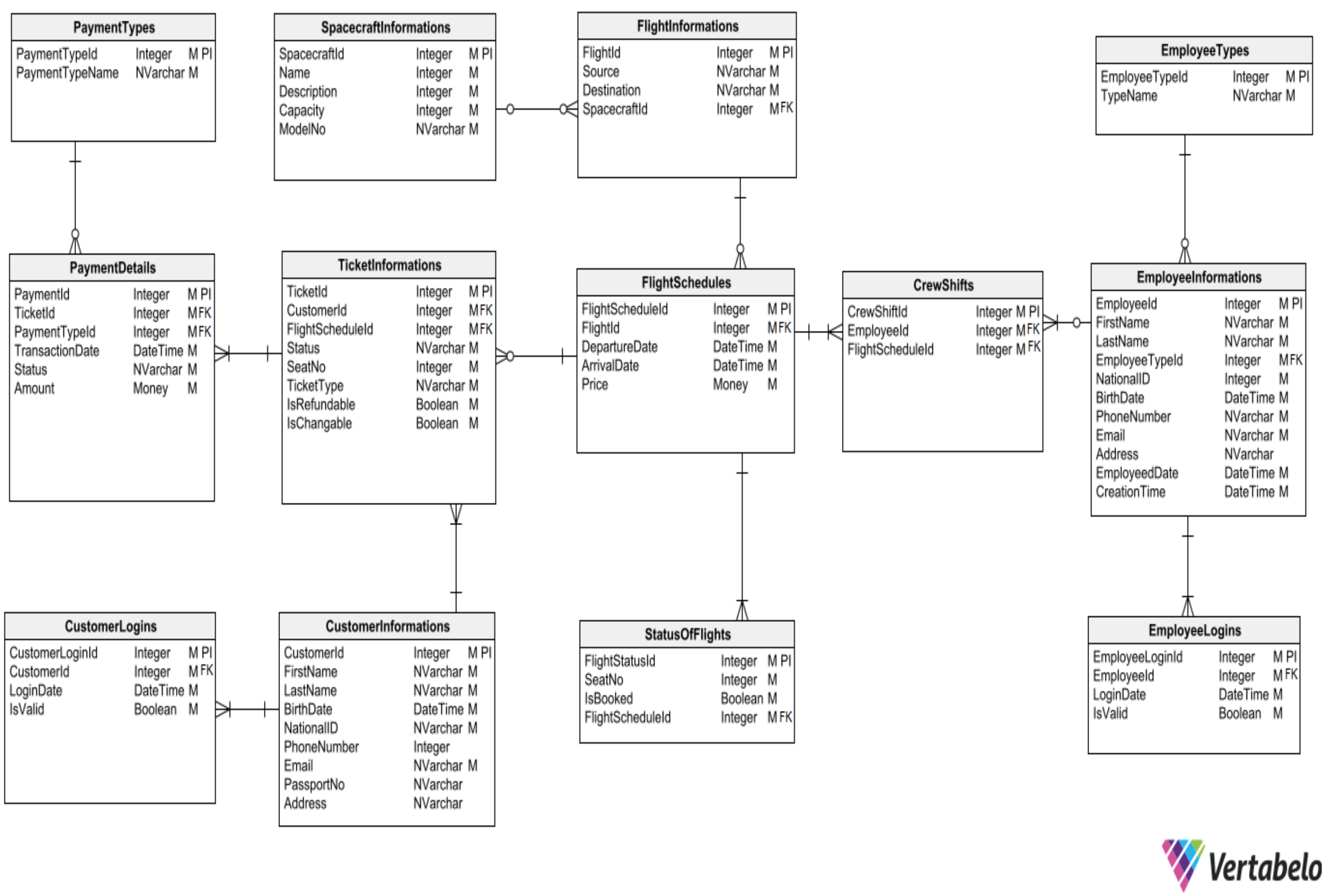
II. Database Design [20 pts.] Create a sample database design (with a name of your choice) for a space flight company like SpaceX and draw one database model for it. The system keeps track of flight information, customer information, employee information, customer and employee login information, rocket/spacecraft 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.

FlightInfortamtion 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.