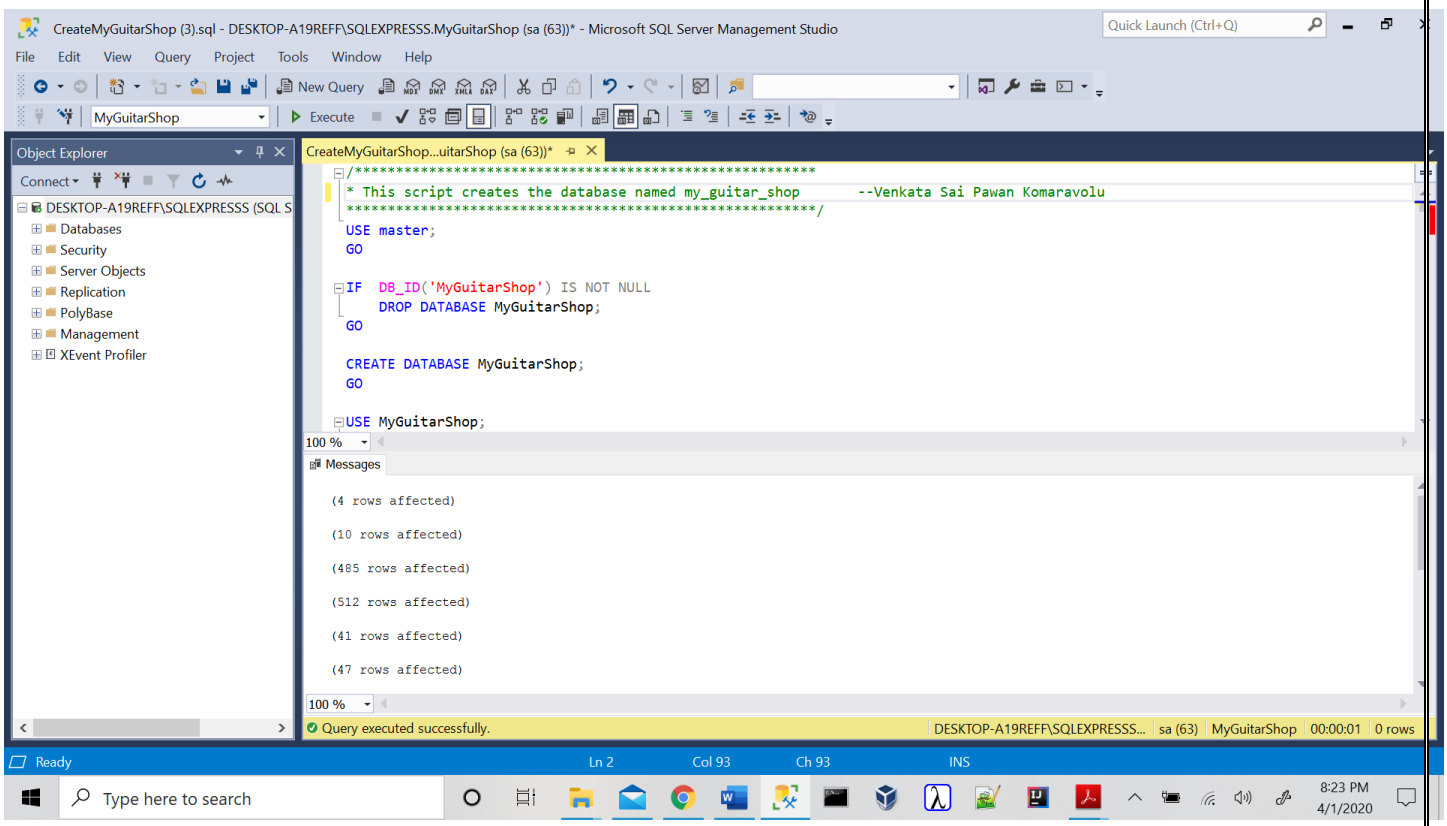
I. My Guitar Shop Database

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.

Sol:

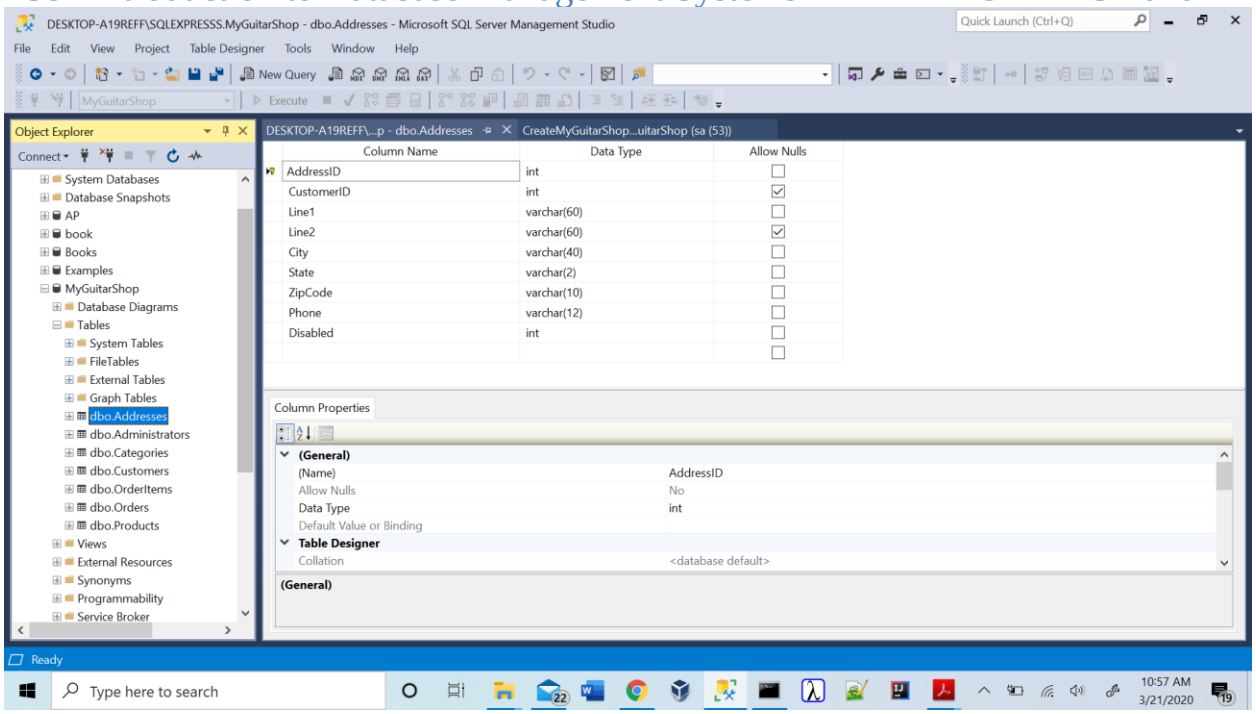
Comments: We have downloaded CreateMyGuitarShop.sql file executed the entire script. The Screenshot below shows the result that the script is executed.



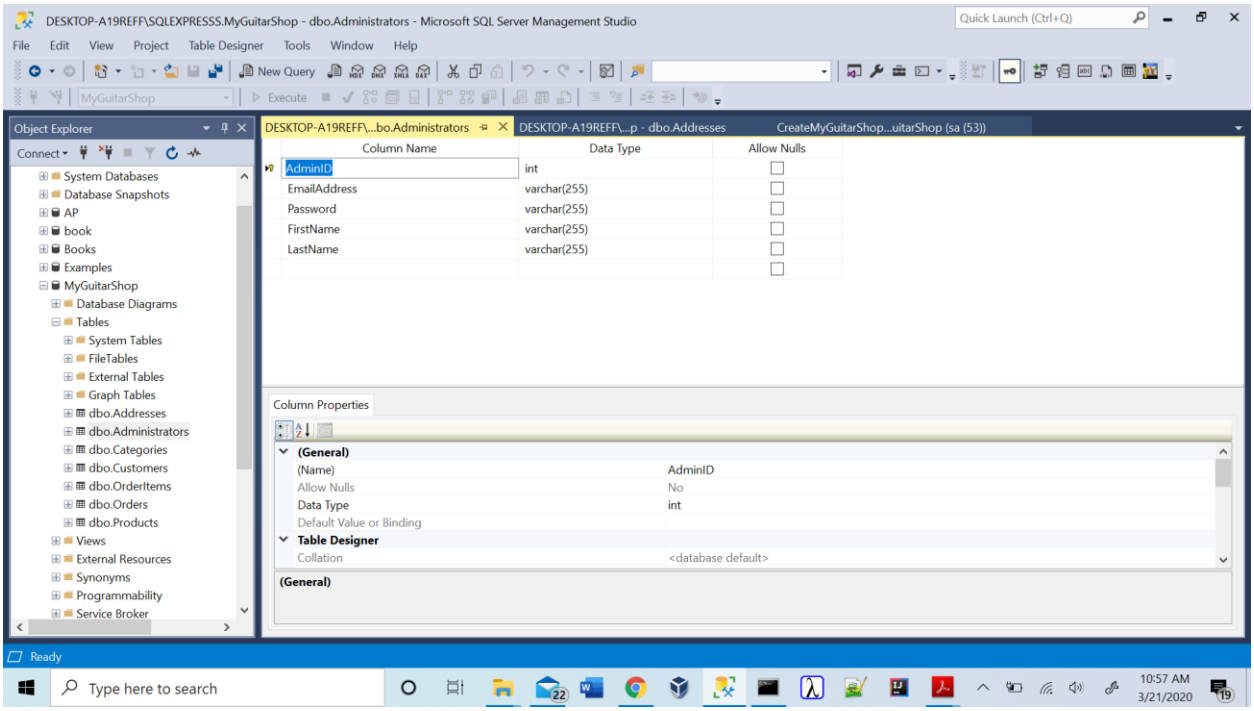
(2) Navigate through the database objects and view the column definitions for each table. Open a new Query Editor window. Show details in Categories table and Products table using SELECT statement. Full screenshots of execution results are required.

Sol: This is the column definition for Address Table:

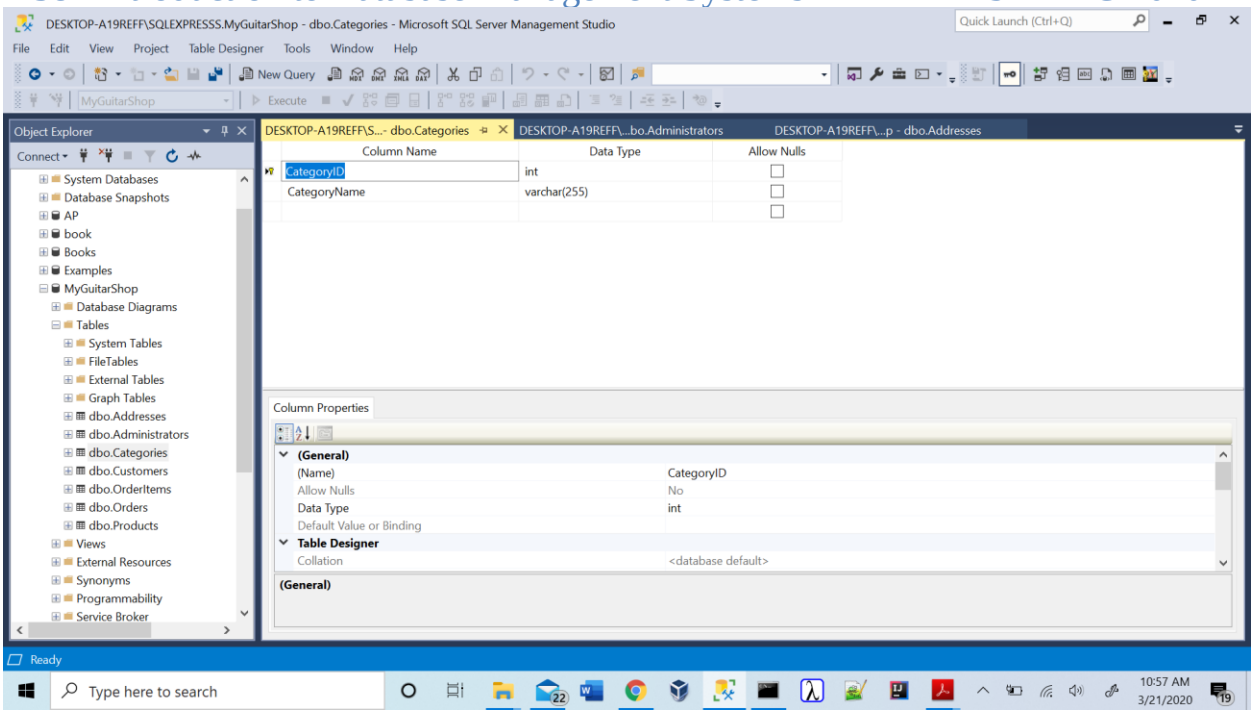
Comments: Here is the Column definition for Address Table



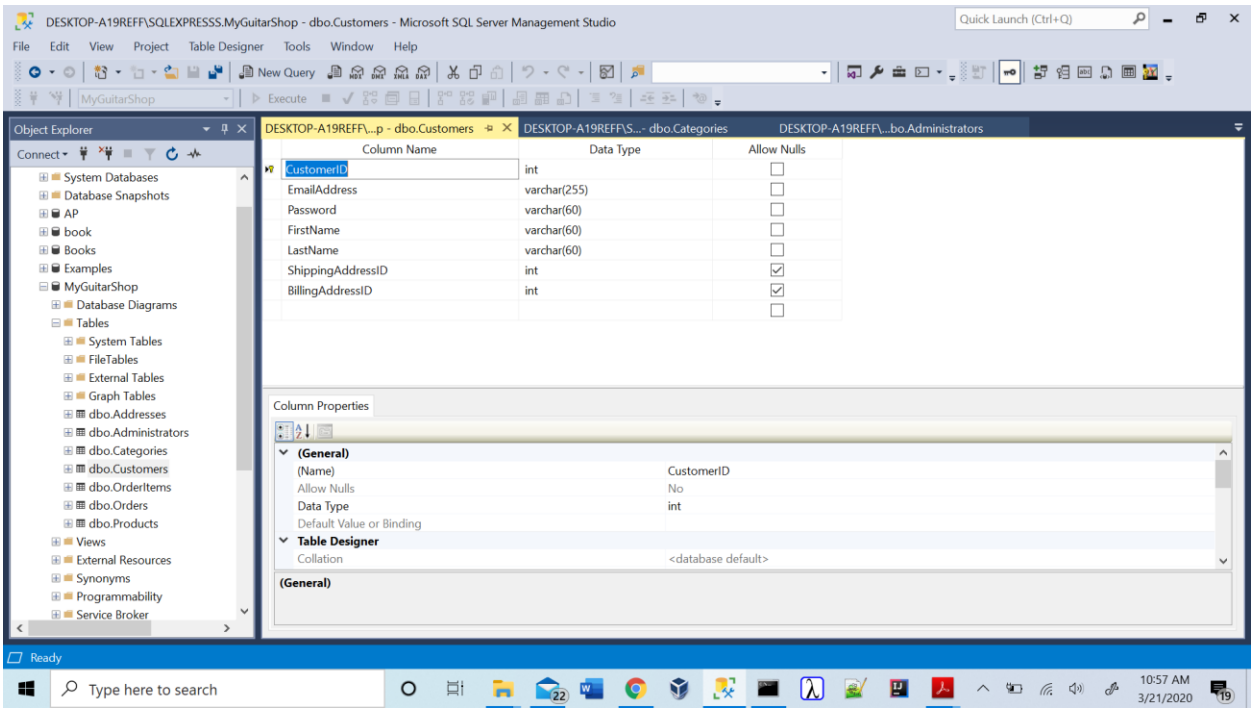
Comments : This is the column definition for Administrator table



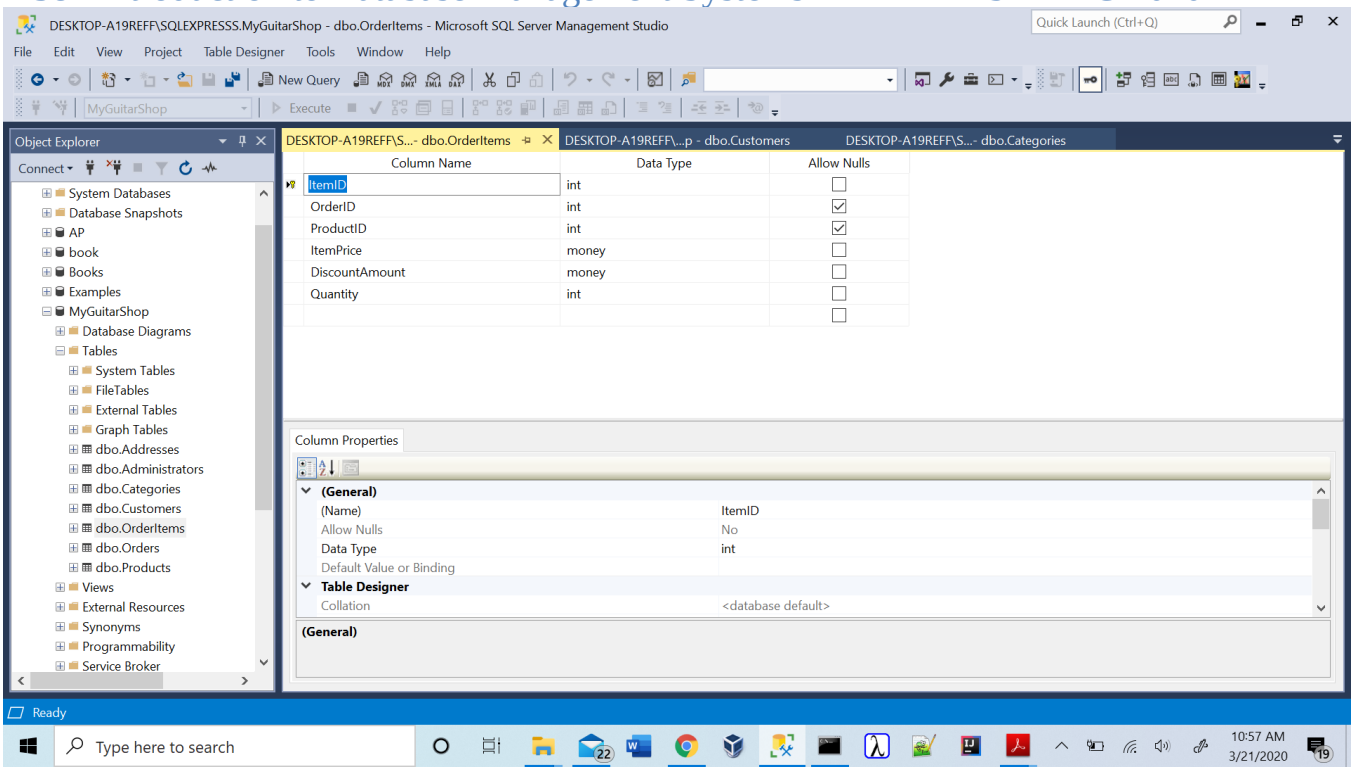
Comments : Below is the Column Definition Screenshot for Categories table.



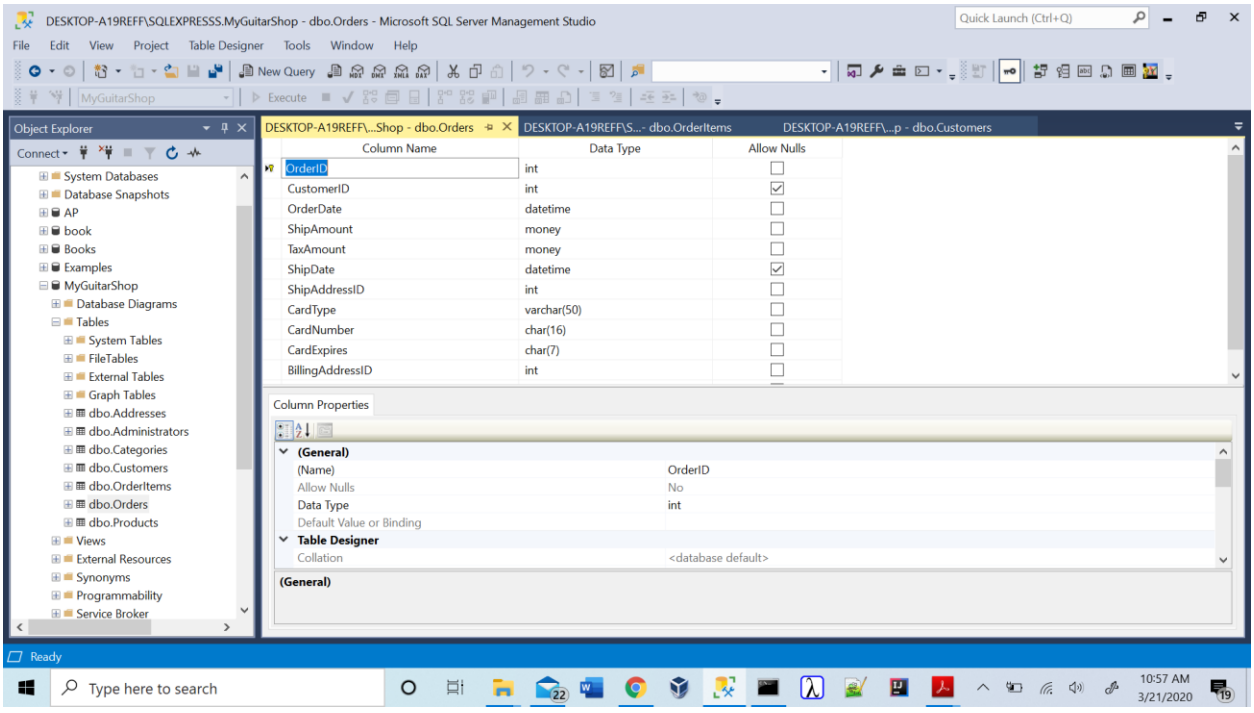
Comments : Below is the Column Definition screenshot for Customers Table



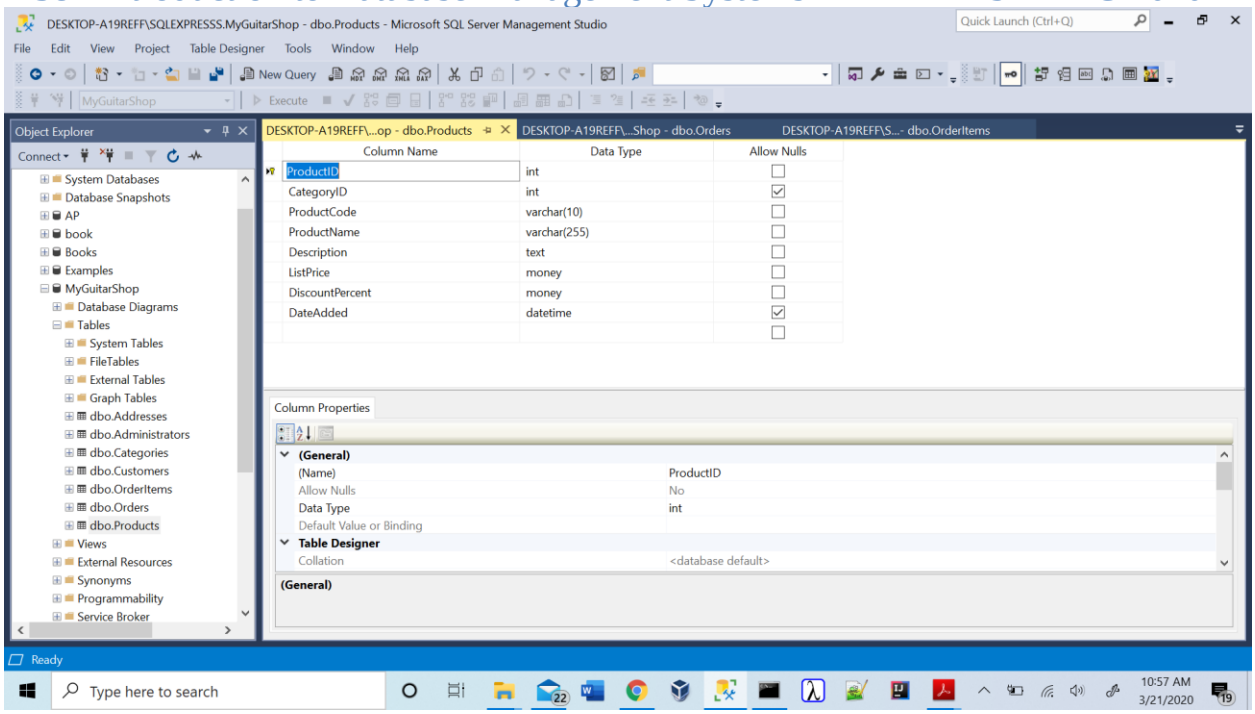
Comments :Below is the Column Definition Screenshot for OrderItems Table



Comments : Below is the Column Definition Screenshot for Order Table

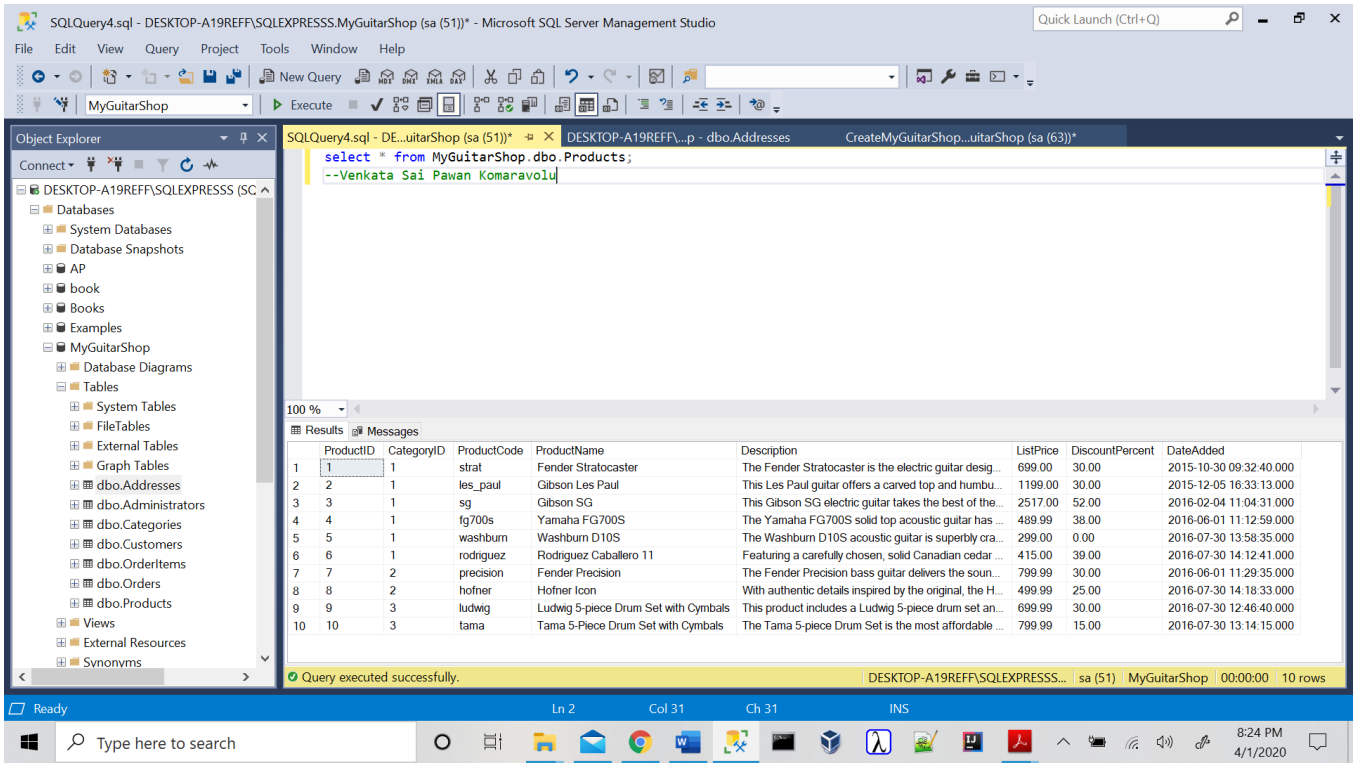


Comments: Below is the Column Definition screenshot for Products Table



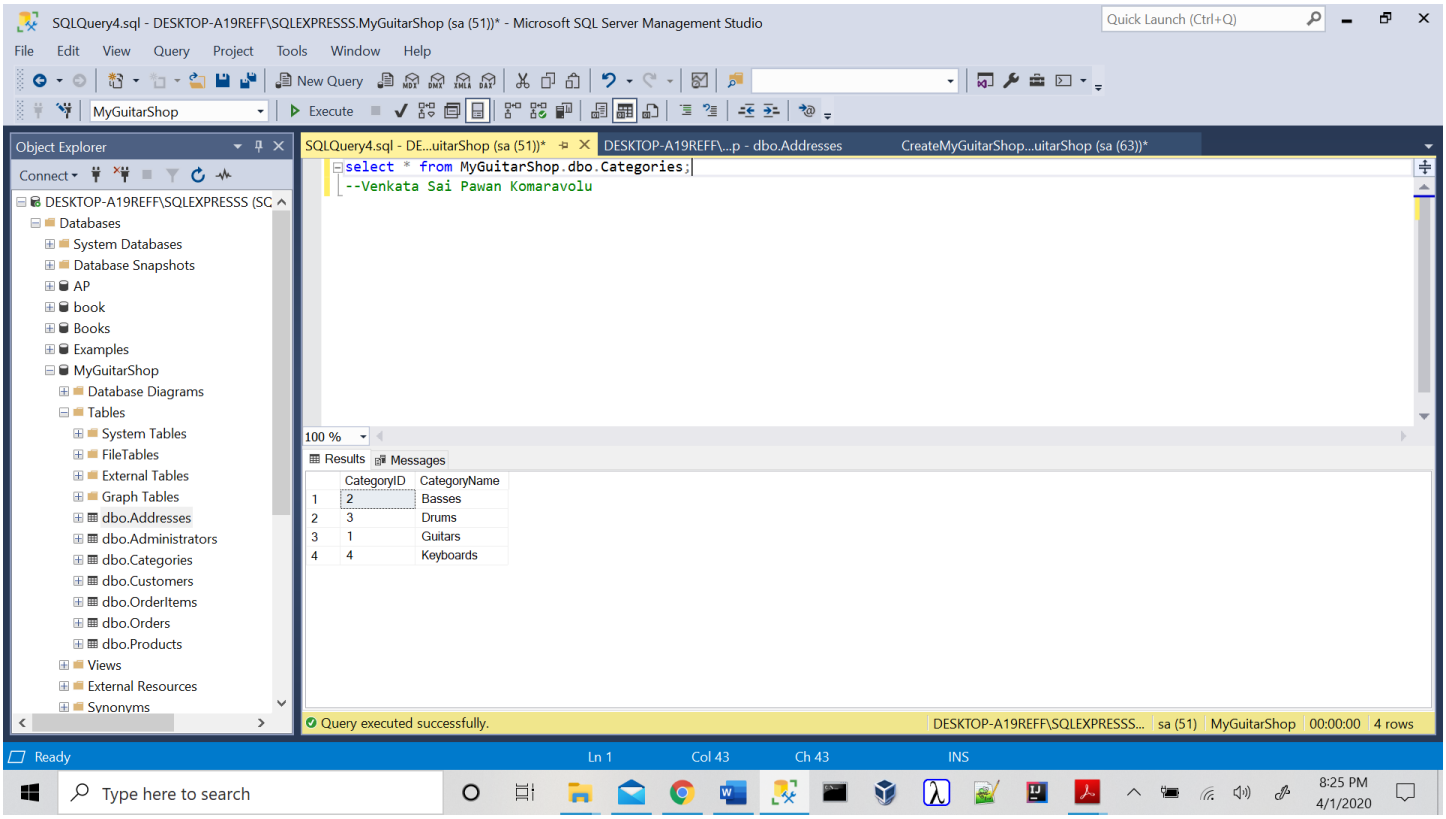
Query : select * from MyGuitarShop.dbo.Products;

Comments :This query shows all details in Products table



Query : select * from MyGuitarShop.dbo.Categories;

Comments :This query shows all details in Categories table

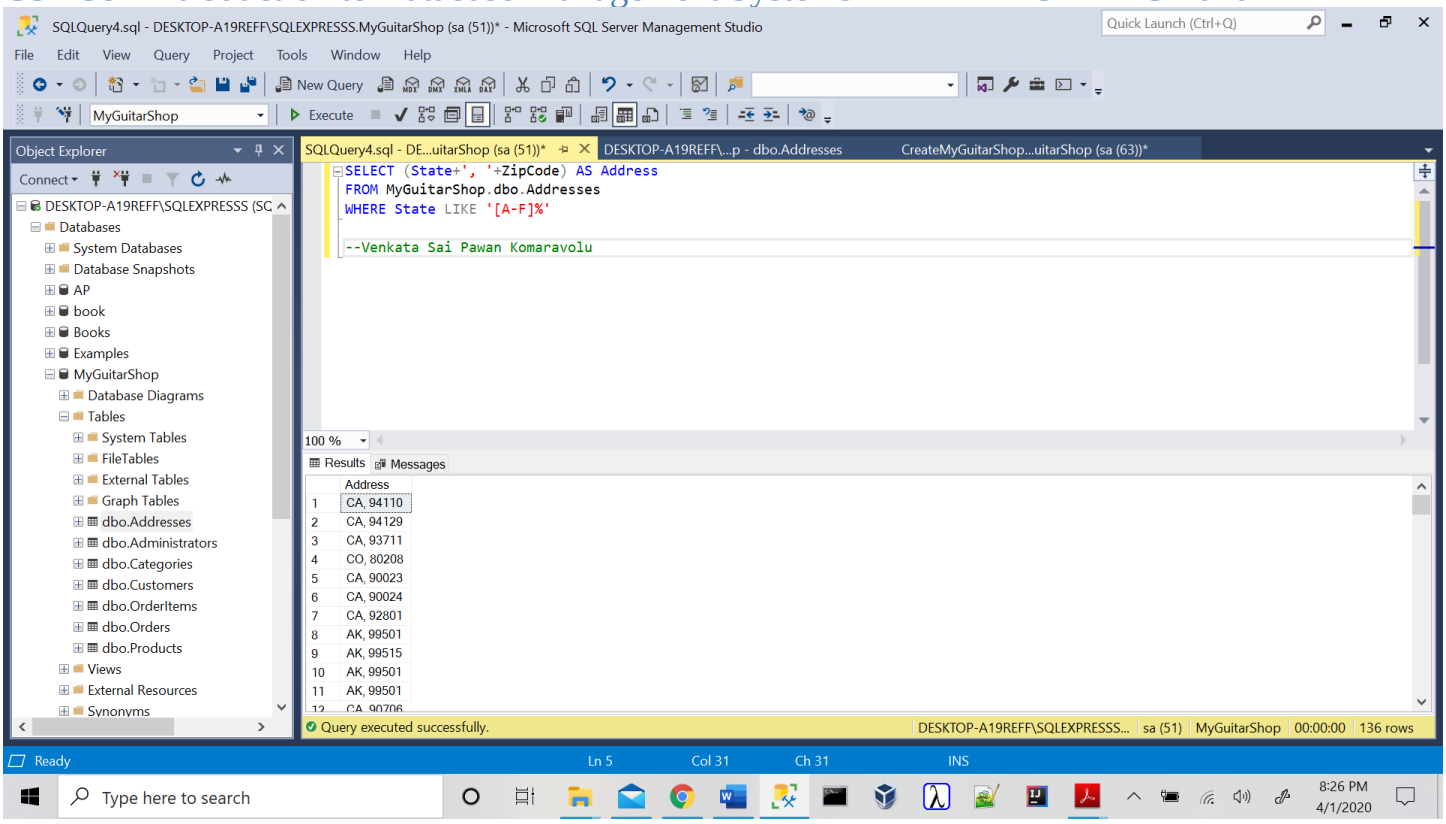


B An introduction to SQL [6 pts.]

1. [3] Write a SELECT statement that returns one column from the Addresses table named Address that joins the State and ZipCode columns. Format this column with the State, a comma, a space, and the ZipCode like this: NY, 13210 Add an ORDER BY clause to this statement that sorts the result set by State in ascending sequence. Return only the addresses whose state names begin with a letter from A to F.

Sol: SELECT (State+', '+ZipCode) AS Address
FROM MyGuitarShop.dbo.Addresses
WHERE State LIKE '[A-F]%'
ORDER BY State;

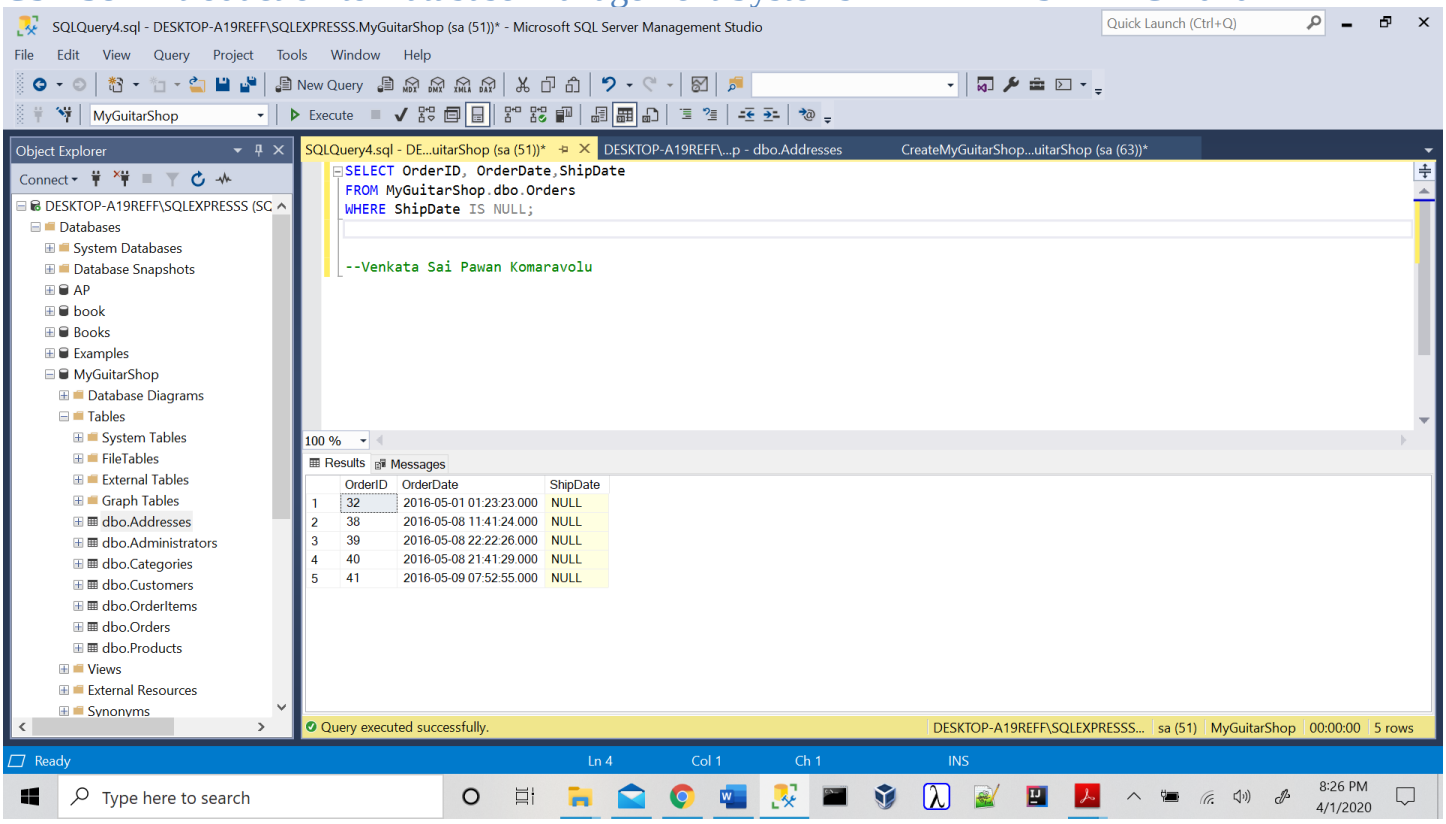
Commennts :We select State , ZipCode from Address table with state beginning with A,B,C,D,E,F and these are ordered by state.



2. [3] Write a SELECT statement that returns three columns: OrderID, OrderDate, ShipDate from the Orderstable. Return only the rows where the ShipDate column contains a null value.

Sol: SELECT OrderID, OrderDate,ShipDate
FROM MyGuitarShop.dbo.Orders
WHERE ShipDate IS NULL;

Comments: We select OrderID, OrderDate,ShipDate from orders table on condition that the shipdate is null



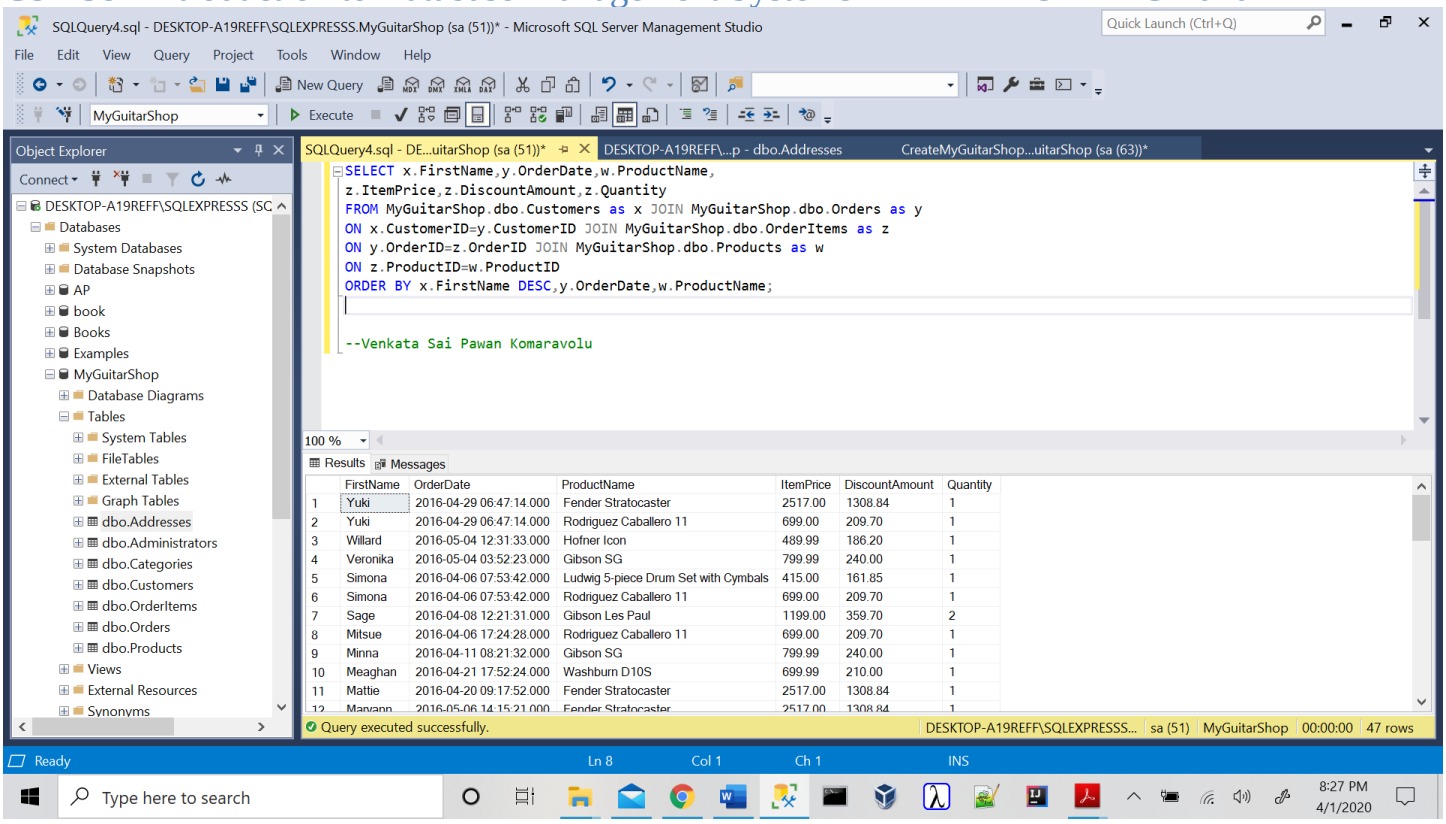
C The essential SQL skills

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

Sol: Souce Code:

```
SELECT x.FirstName,y.OrderDate,w.ProductName,
z.ItemPrice,z.DiscountAmount,z.Quantity
FROM MyGuitarShop.dbo.Customers as x JOIN MyGuitarShop.dbo.Orders as y
ON x.CustomerID=y.CustomerID JOIN MyGuitarShop.dbo.OrderItems as z
ON y.OrderID=z.OrderID JOIN MyGuitarShop.dbo.Products as w
ON z.ProductID=w.ProductID
    ORDER BY x.FirstName DESC,y.OrderDate,w.ProductName;
```

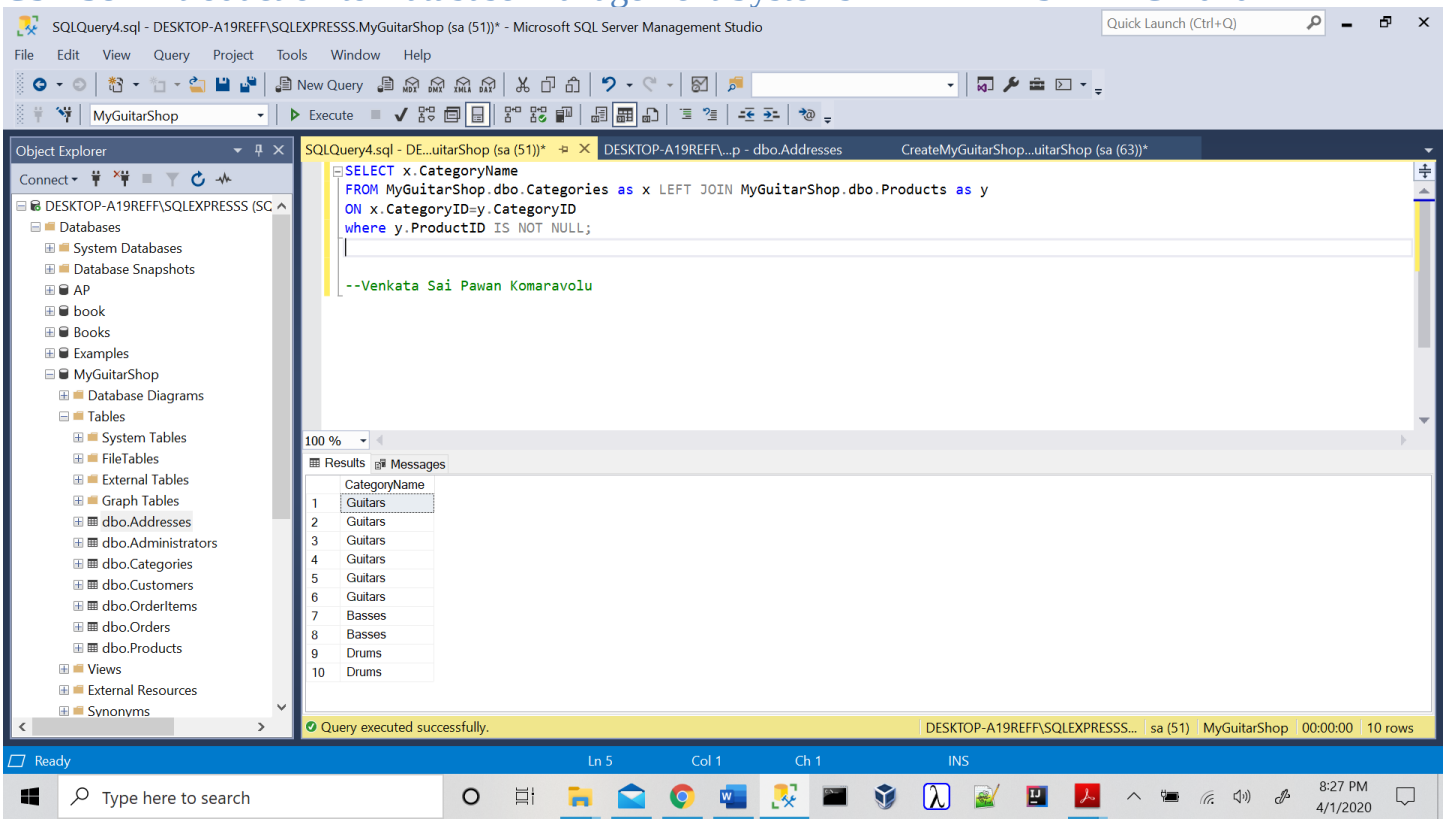
Comments: We first name the alias x for customers table , y for orders table , w for products table and z for Orderitems and then select firstname , orderdate, productname, Itemprice,DiscountAmount, quantity from their respective tables and use joinand then orderby firstname in Descending order and orderdate and product name is ascending order



2. [4] Write a SELECT statement that selects 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).

Sol: SELECT x.CategoryName
FROM MyGuitarShop.dbo.Categories as x LEFT JOIN MyGuitarShop.dbo.Products as y
ON x.CategoryID=y.CategoryID
where y.ProductID IS NOT NULL;

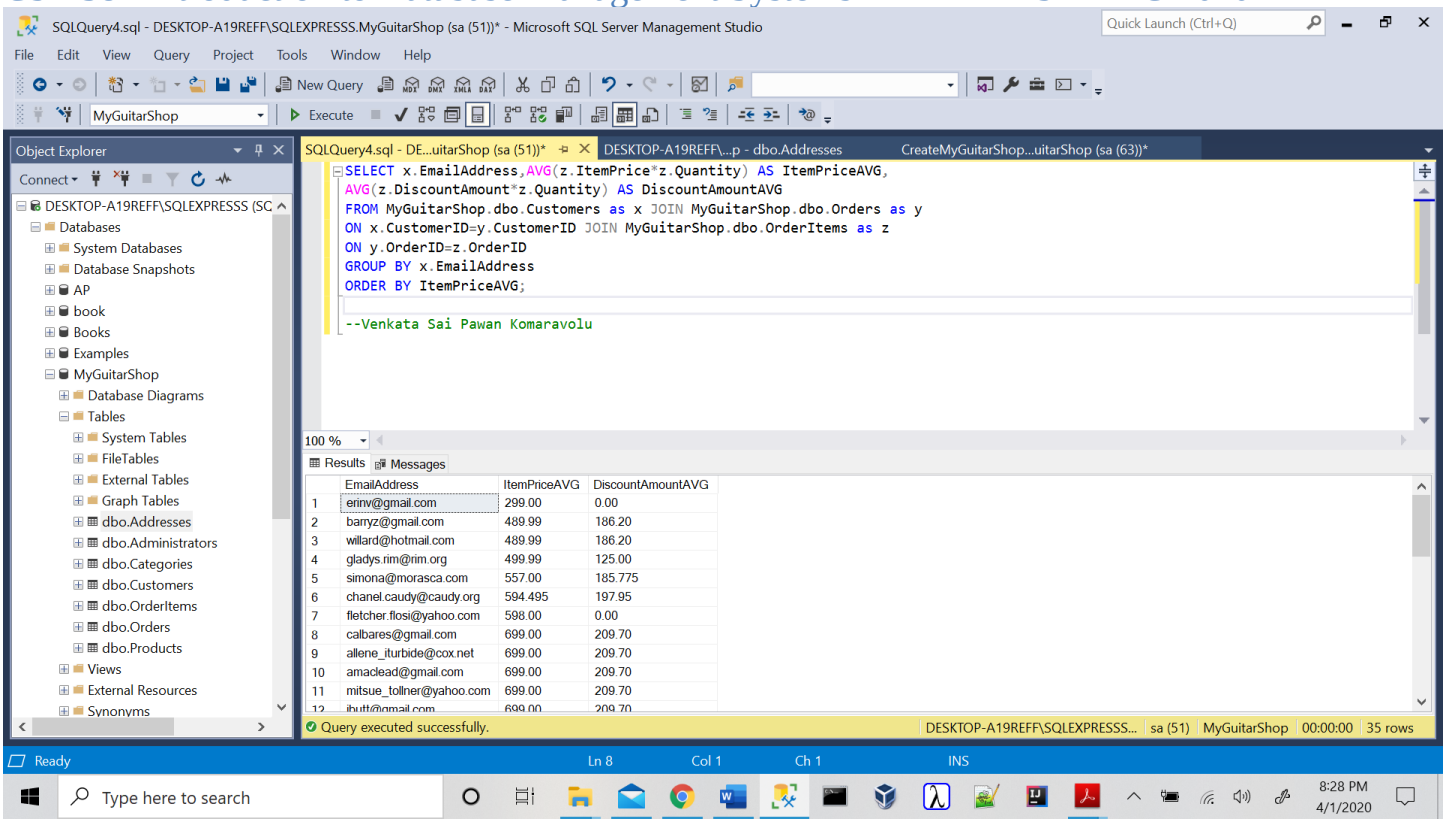
Comments: We select CategoryName from Categories table and we are doing left join with products table havinf categories alias as x and products alias as y on same CategoryID and ProductID of products in not null.



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 average of the item price in the OrderItems table multiplied by the quantity in the OrderItems table c) The average of the discount amount column in the OrderItems table multiplied by the quantity in the OrderItems table. Sort the result set in ascending sequence by the item price average for each customer.

Sol: Source Code: SELECT x.EmailAddress,AVG(z.ItemPrice*z.Quantity) AS ItemPriceAVG,
AVG(z.DiscountAmount*z.Quantity) AS DiscountAmountAVG
FROM MyGuitarShop.dbo.Customers as x JOIN MyGuitarShop.dbo.Orders as y
ON x.CustomerID=y.CustomerID JOIN MyGuitarShop.dbo.OrderItems as z
ON y.OrderID=z.OrderID
GROUP BY x.EmailAddress
ORDER BY ItemPriceAVG;

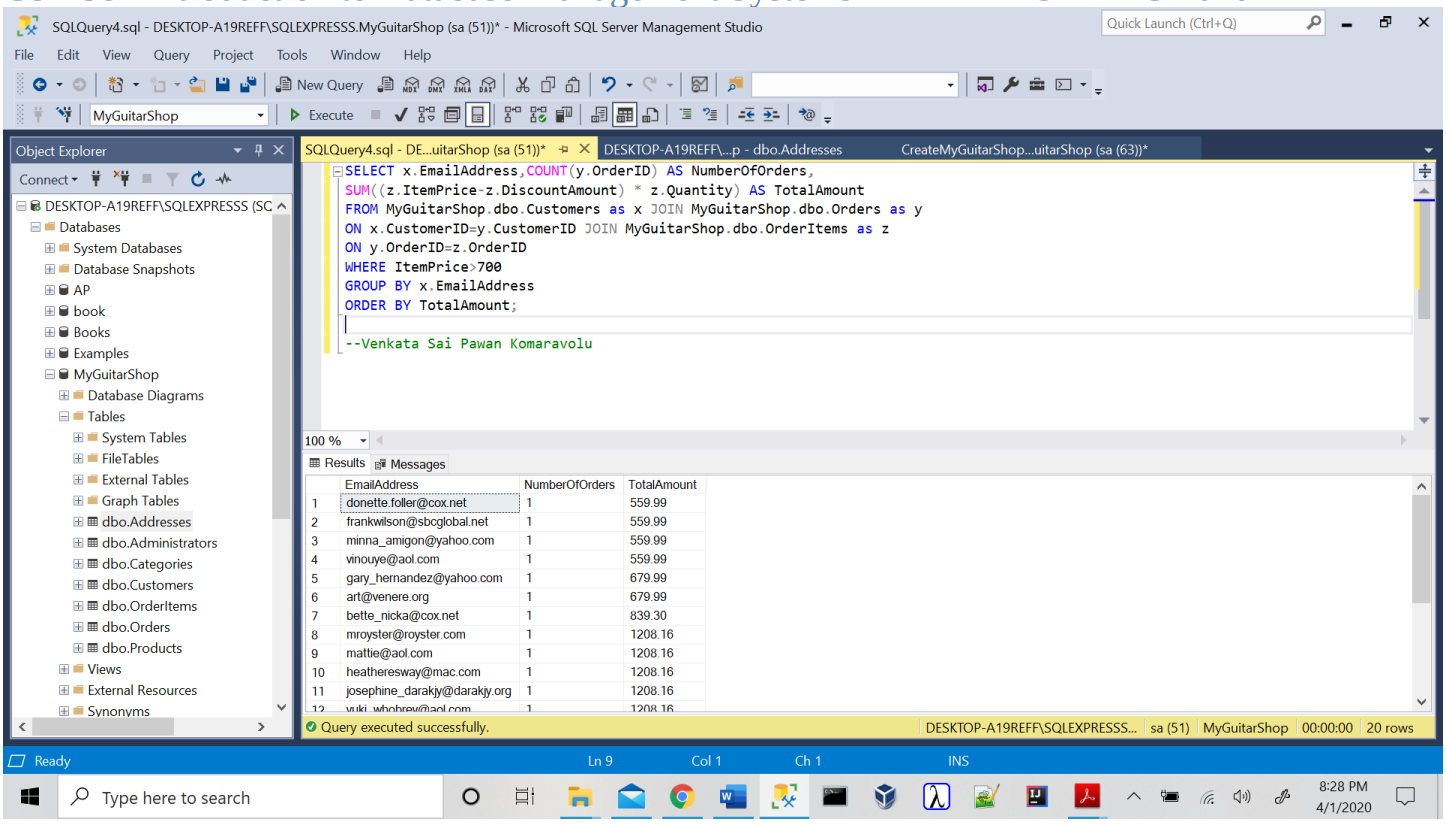
Comments: We select emailaddress from Customers table , Average of the product of Itemprice and quantiy and average of product of Discount Amount and quantity from their respective table grouped by emailed and Ordered byItemPriceAVG.



4. [4] Write a SELECT statement that returns one row for each customer that has orders with these columns: a) The EmailAdress 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 700 ItemPrice value. Sort the result set in ascending sequence by the sum of the line item amounts.

Sol: Source Code: SELECT x.EmailAddress,COUNT(y.OrderID) AS NumberOfOrders,
SUM((z.ItemPrice-z.DiscountAmount) * z.Quantity) AS TotalAmount
FROM MyGuitarShop.dbo.Customers as x JOIN MyGuitarShop.dbo.Orders as y
ON x.CustomerID=y.CustomerID JOIN MyGuitarShop.dbo.OrderItems as z
ON y.OrderID=z.OrderID
WHERE ItemPrice>700
GROUP BY x.EmailAddress
ORDER BY TotalAmount;

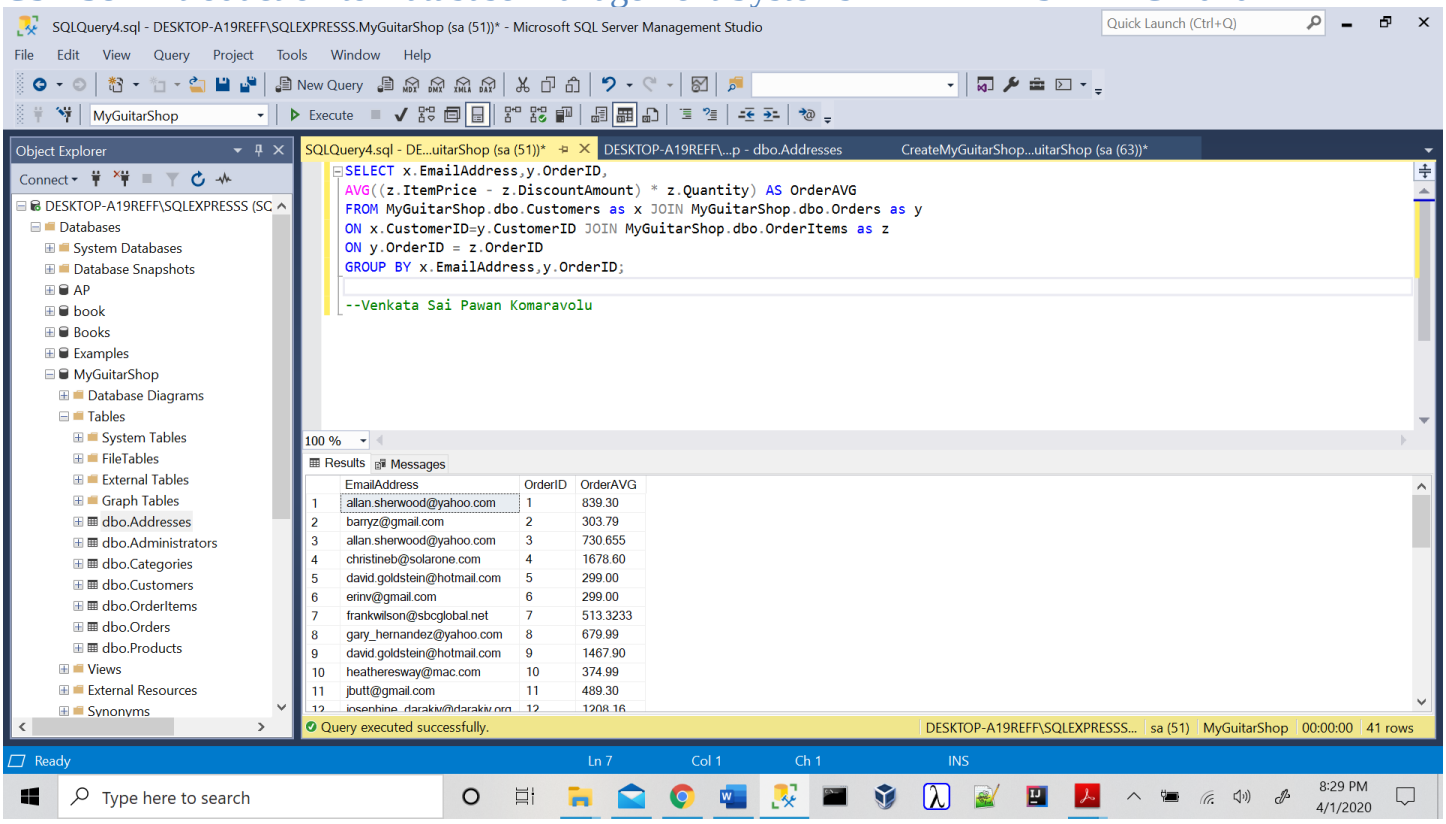
Comments: We select email Address from Customers table using alias x , then we count orderID from Orders table using alias y , we then make Sum of difference between ItemPrice and Discount Amount and multiply with the Quantity as we name it as TotalAmount on same CustomerID condition where Itemprice is > 700 and we group by email address and order by the Total Amount.



5. [4] (1) Write a SELECT statement that returns three columns: EmailAddress, OrderID, and the order average 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 average from the columns in the OrderItemstable. (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 average for that customer. To do this, you can group the result set by the EmailAddress column.

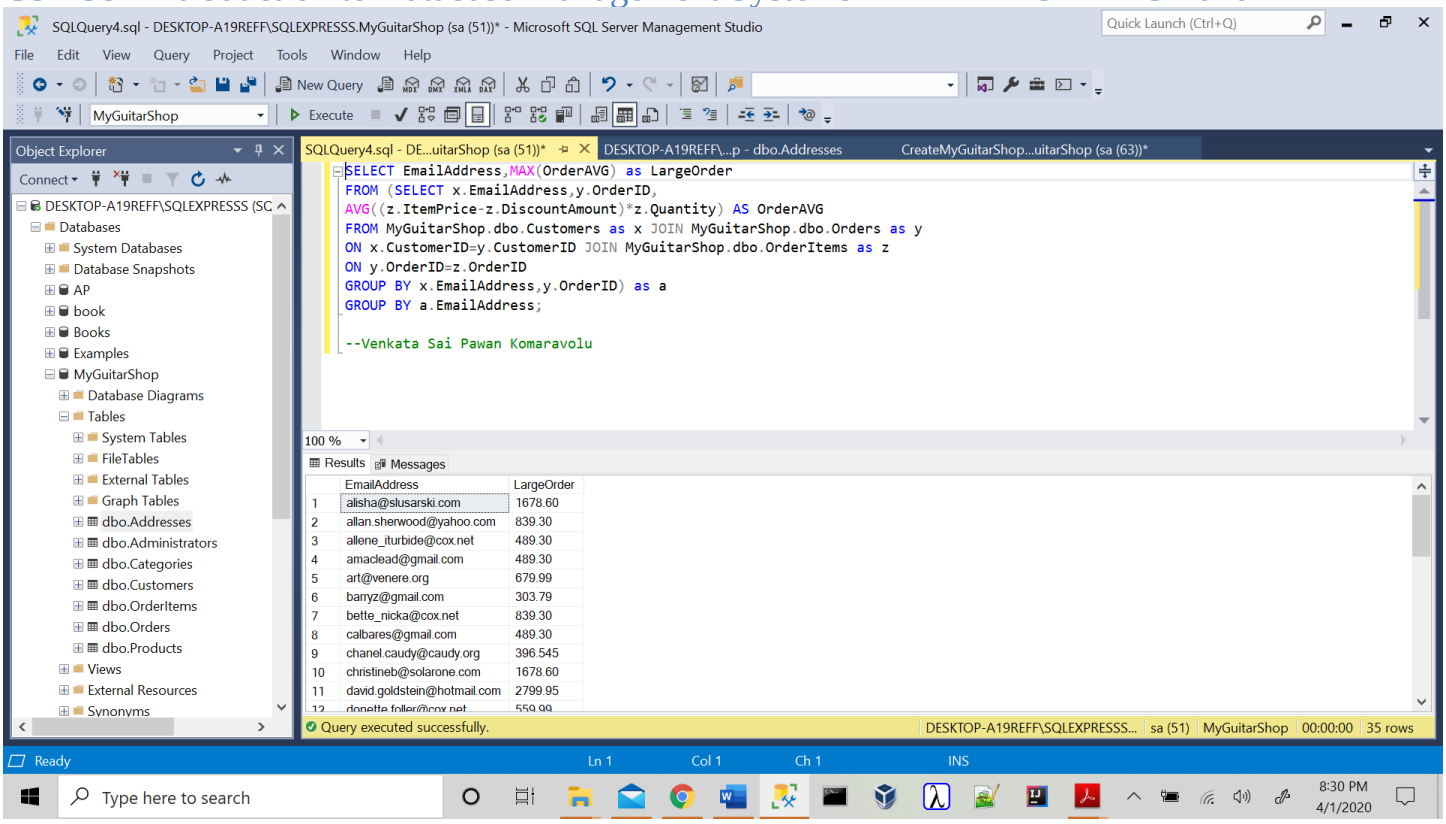
Sol: Source Code: SELECT x.EmailAddress,y.OrderID,
AVG((z.ItemPrice - z.DiscountAmount) * z.Quantity) AS OrderAVG
FROM MyGuitarShop.dbo.Customers as x JOIN MyGuitarShop.dbo.Orders as y
ON x.CustomerID=y.CustomerID JOIN MyGuitarShop.dbo.OrderItems as z
ON y.OrderID = z.OrderID
GROUP BY x.EmailAddress,y.OrderID;

Comments: We are using alias for Customers table as x , Orders table as y, OrdersItems as z and hence we select emailaddress , ordered,average of product of difference of Itemprice and Discountamount and quantityon common customerID and orderID condition.



2) Source code: SELECT EmailAddress,MAX(OrderAVG) as LargeOrder
FROM (SELECT x.EmailAddress,y.OrderID,
AVG((z.ItemPrice-z.DiscountAmount)*z.Quantity) AS OrderAVG
FROM MyGuitarShop.dbo.Customers as x JOIN MyGuitarShop.dbo.Orders as y
ON x.CustomerID=y.CustomerID JOIN MyGuitarShop.dbo.OrderItems as z
ON y.OrderID=z.OrderID
GROUP BY x.EmailAddress,y.OrderID) as a
GROUP BY a.EmailAddress;

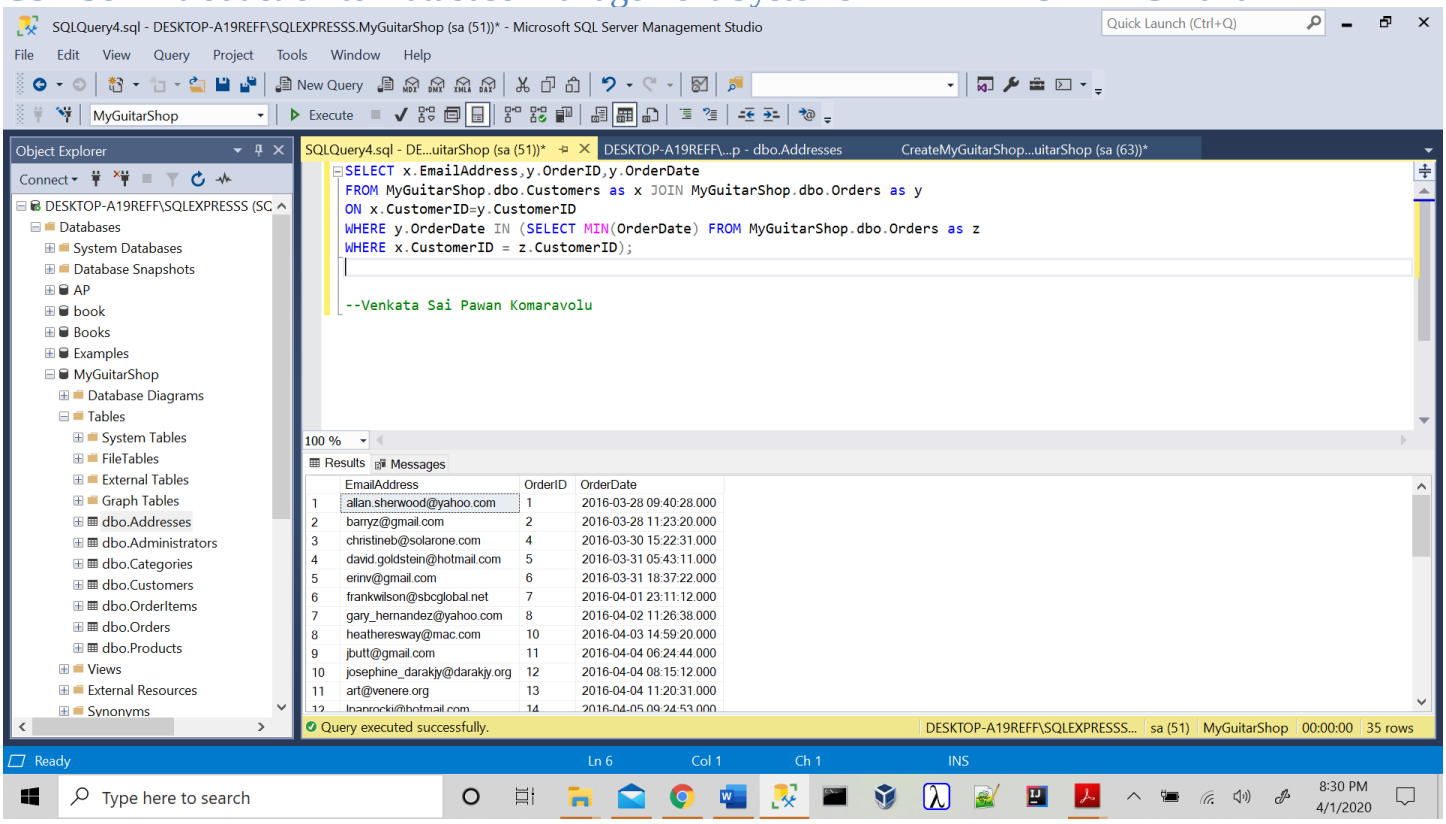
Comments: The subquery returns emailaddress , ordered, orderavg from Customers , Orders and OrderItemstables by joining CustomerId and OrderID columns and the result set is grouped by emailed and orderID.



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

Sol: SELECT x.EmailAddress,y.OrderID,y.OrderDate
FROM MyGuitarShop.dbo.Customers as x JOIN MyGuitarShop.dbo.Orders as y
ON x.CustomerID=y.CustomerID
WHERE y.OrderDate IN (SELECT MIN(OrderDate) FROM MyGuitarShop.dbo.Orders as z
WHERE x.CustomerID = z.CustomerID);

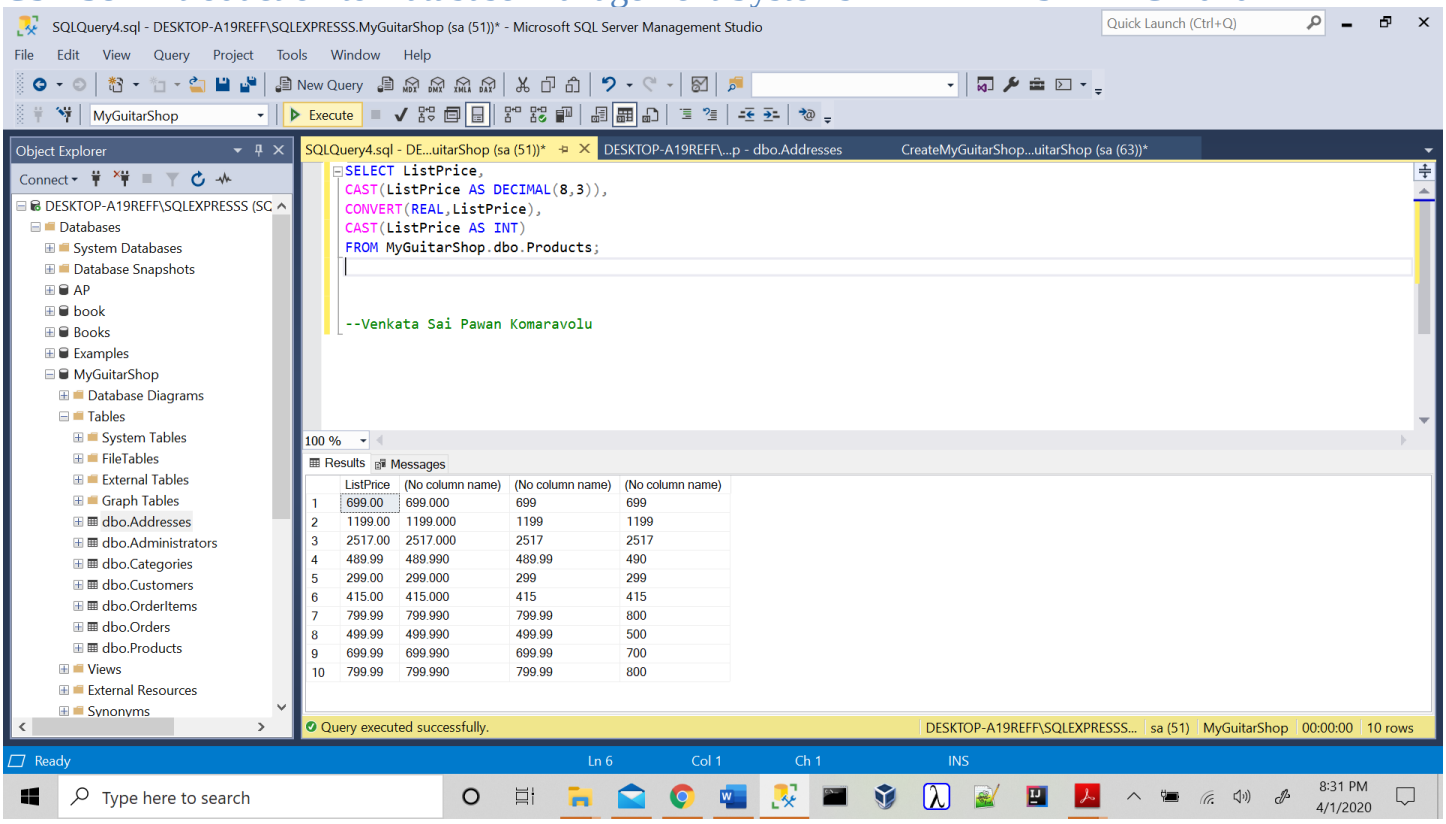
Comments: We use correlated subquery for Customers table as x, Orders table as y and select email address, orderID ,orderDate from their respective tables on common CustomerID where orderDate is thee oldestdate in the orders on common CustomerID.



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 3 digits to the right
   of the decimal point c) A column that uses the CONVERT function to return the ListPrice column as a real number d) A column that uses the CAST function to return the ListPrice column as an integer

Sol: Source code: SELECT ListPrice,
CAST(ListPrice AS DECIMAL(8,3)),
CONVERT(REAL,ListPrice),
CAST(ListPrice AS INT)
FROM MyGuitarShop.dbo.Products;

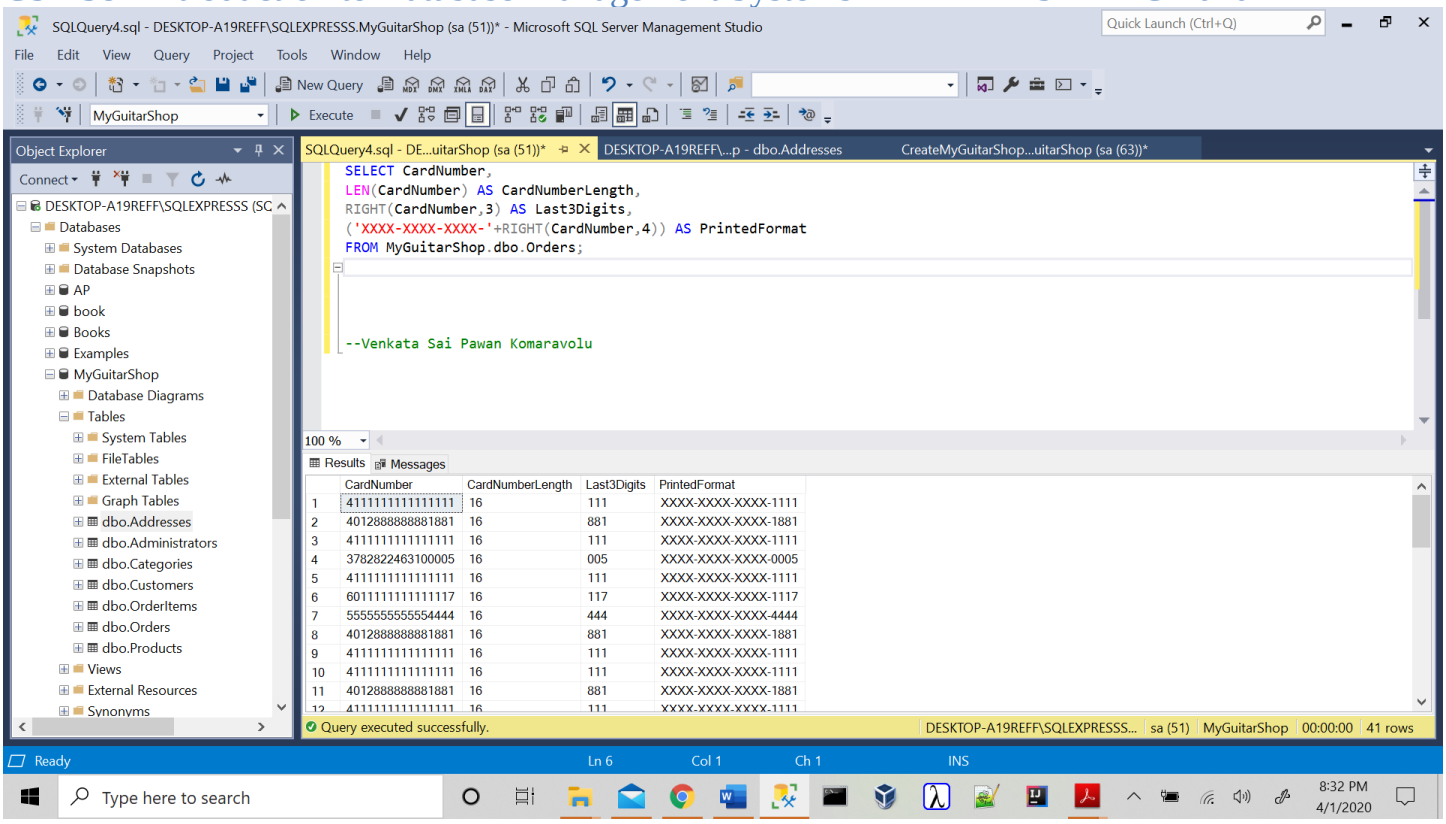
Comments: We select ListPrice, and cast the listprice to a decimal with 3 digits to right and covert the listprice to real and again cast the listprice as int from the Products table.



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 three digits of the CardNumber column d) A column that displays the last four digits of the CardNumber column in this format: XXXXXXXX-XXXX-1234. In other words, use Xs for the first 12 digits of the card number and actual numbers for the last four digits of the number.

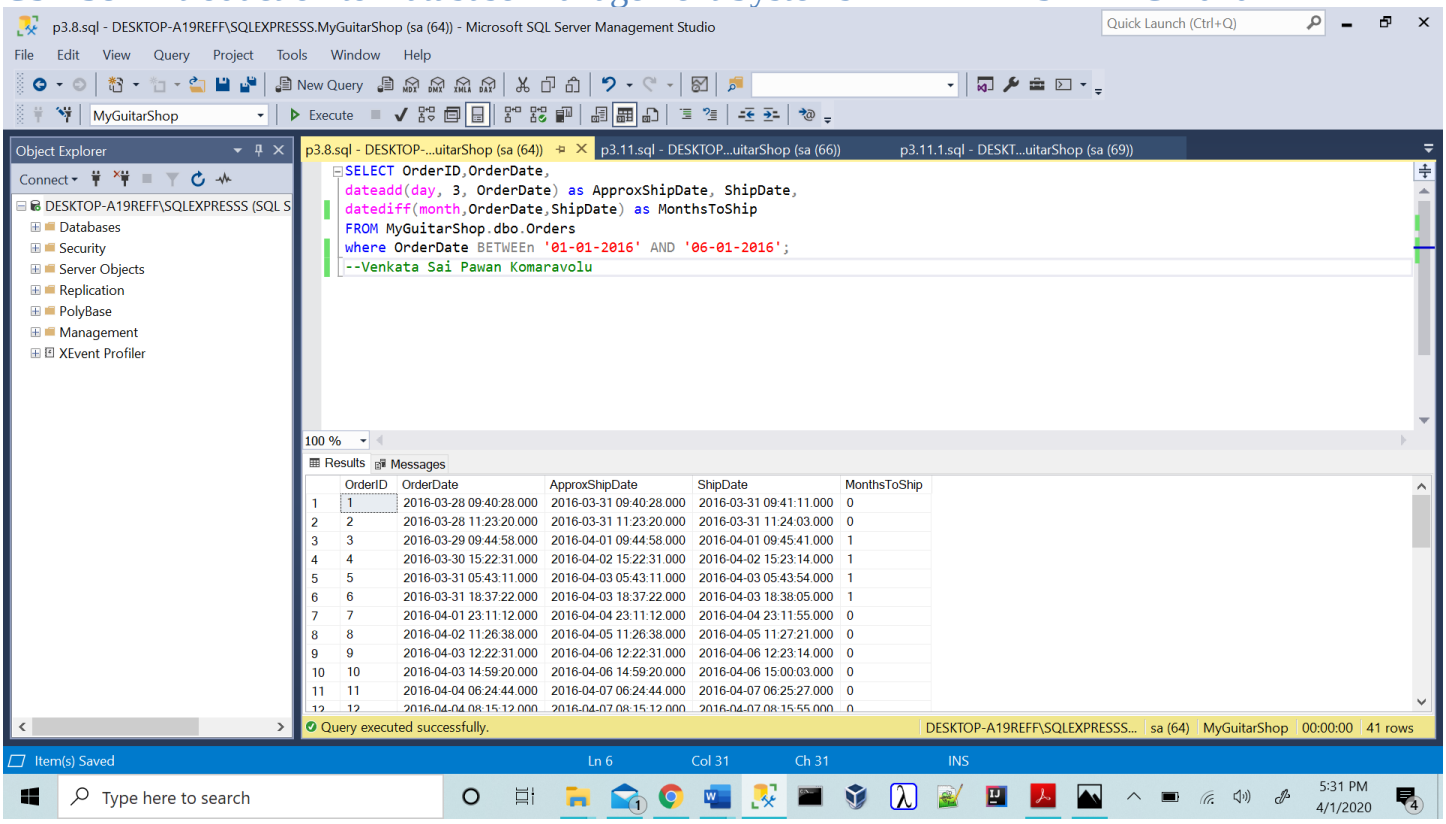
Sol: SELECT CardNumber,
LEN(CardNumber) AS CardNumberLength,
RIGHT(CardNumber,3) AS Last3Digits,
('XXXX-XXXX-XXXX-'+RIGHT(CardNumber,4)) AS PrintedFormat
FROM MyGuitarShop.dbo.Orders;

Comments: We select CardNumber , length of the cardnumber and the last 3 digits of the card and we display xxx..with last 4 digits of the card from the orders table.



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 3 days 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 Jan to June 2016.

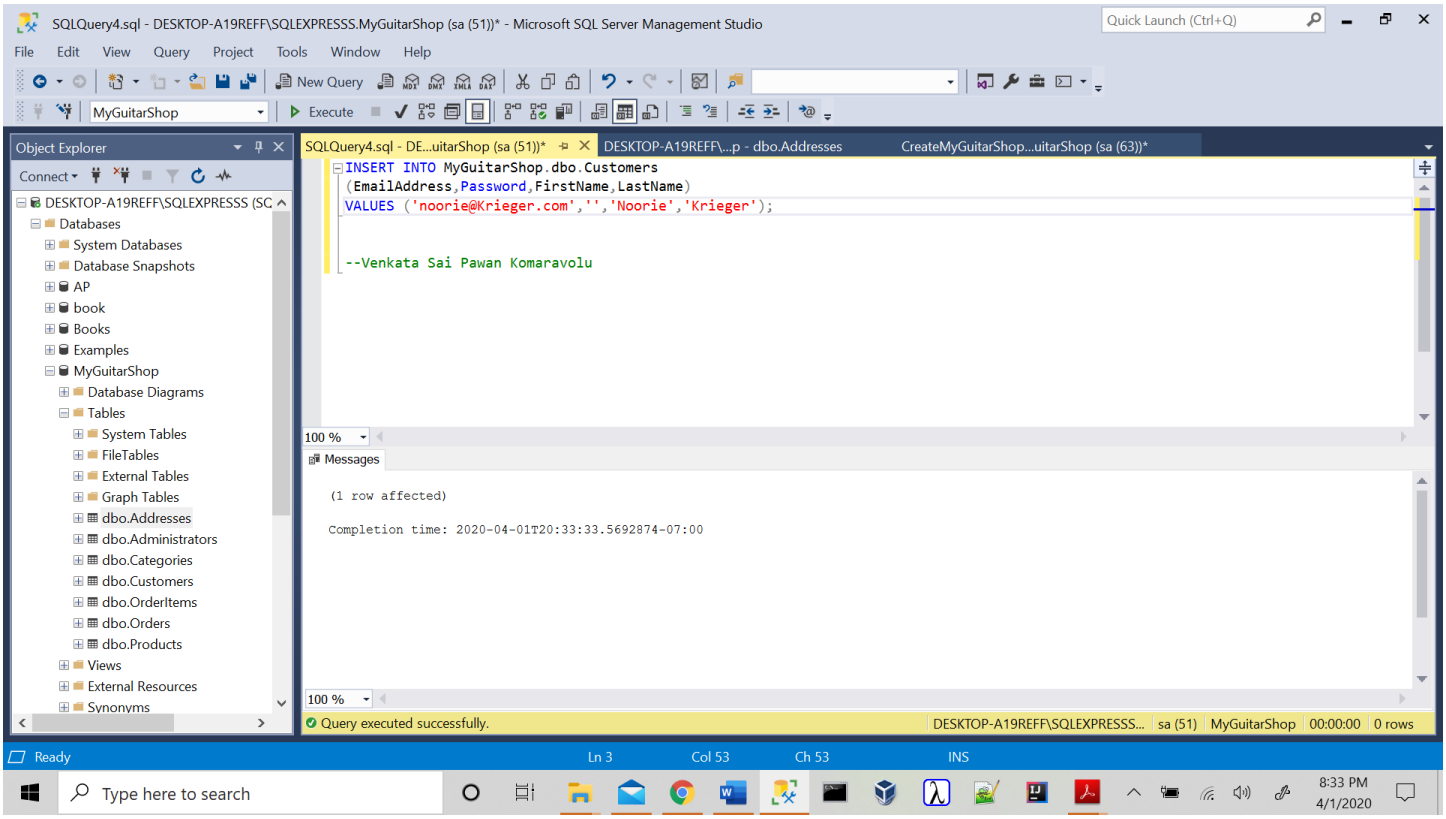
Sol: SELECT OrderID,OrderDate,
dateadd(day, 3, OrderDate) as ApproxShipDate, ShipDate,
datediff(month,OrderDate,ShipDate) as MonthsToShip
FROM MyGuitarShop.dbo.Orders
where OrderDate BETWEEn '01-01-2016' AND '06-01-2016';
Comments: We select OrderID,orderDate and a new date which is the Approx shipping date that is approx. 3 days from the day of ordering, from the orders tablewhere orderdate year is 2016 and month is between jan and june



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

Sol : INSERT INTO MyGuitarShop.dbo.Customers
(EmailAddress,Password,FirstName,LastName)
Comments: We insert in to customers table and add EmailAddress , password, first name and Lastname columns

VALUES ('noorie@Krieger.com','','Noorie','Krieger');



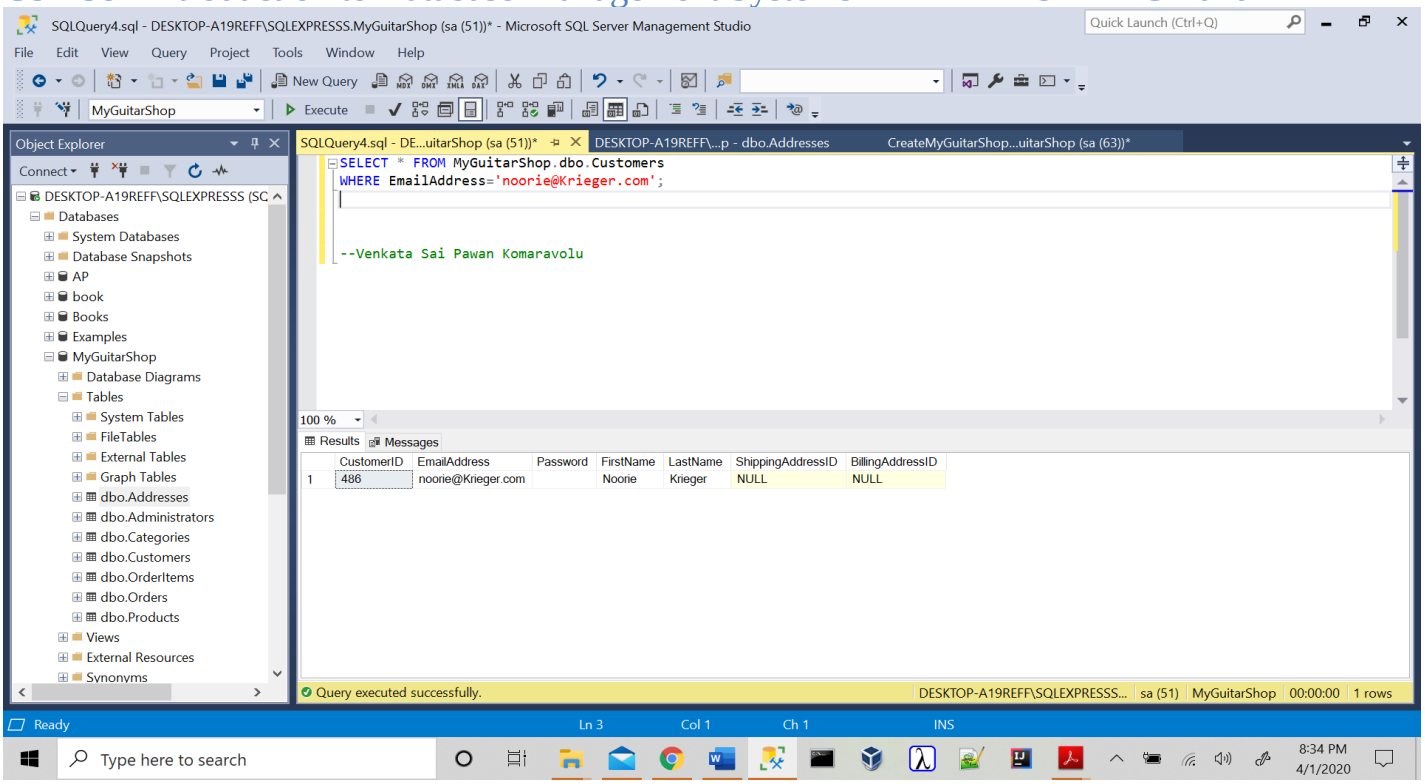
Source: SELECT * FROM MyGuitarShop.dbo.Customers
WHERE EmailAddress='noorie@Krieger.com';

Comments: We select everything from Customer table where EmailAddress is 'noorie@Krieger.com'

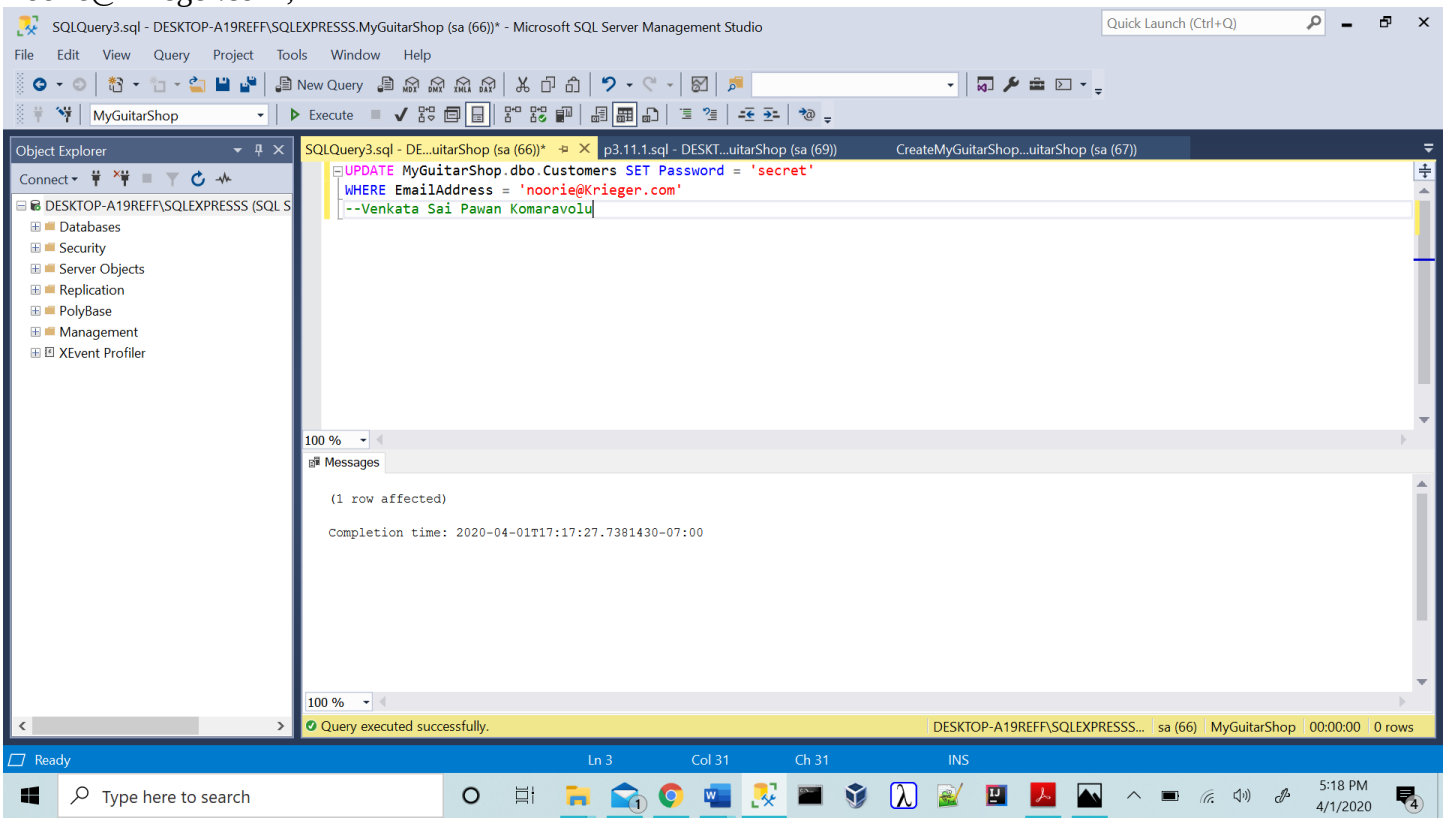
11. [4] Write an UPDATE statement that modifies the Customers table. Change the password column to "secret" for the customer with an email address of noorie@krieger.com.

Sol :
SELECT * FROM MyGuitarShop.dbo.Customers WHERE EmailAddress = 'noorie@Krieger.com';

Comments: We select everything form Customers table where emil is is= 'noorie@Krieger.com';

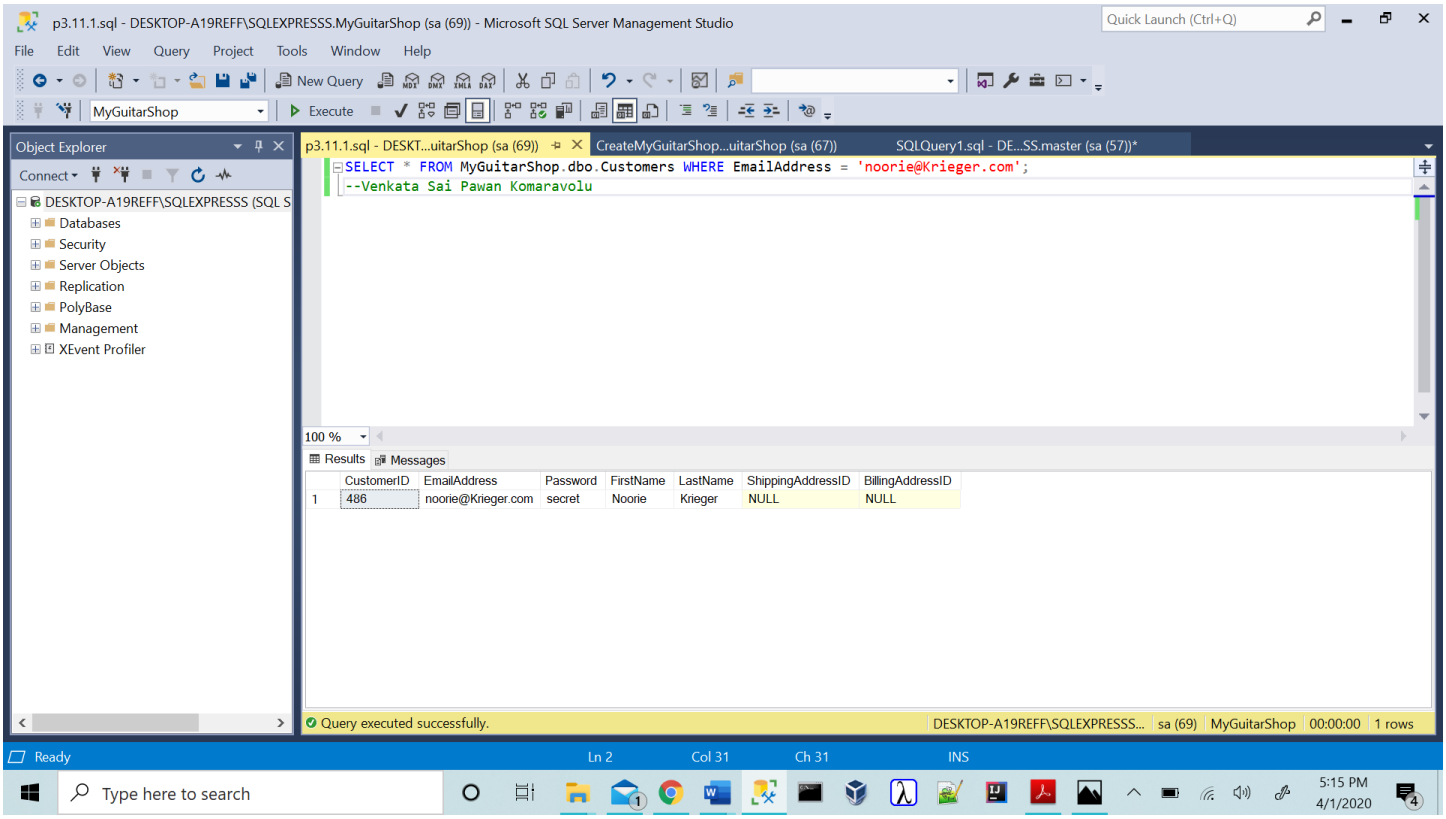


UPDATE MyGuitarShop.dbo.Customers SET Password = 'secret'
WHERE EmailAddress = 'noorie@Krieger.com'
Comments: We update the Customers table and set password to ‘secret’ with emailaddress = 'noorie@Krieger.com';



SELECT * FROM MyGuitarShop.dbo.Customers WHERE EmailAddress = 'noorie@Krieger.com';

Comments: we select after update statement and see the password the emailaddress = 'noorie@Krieger.com';

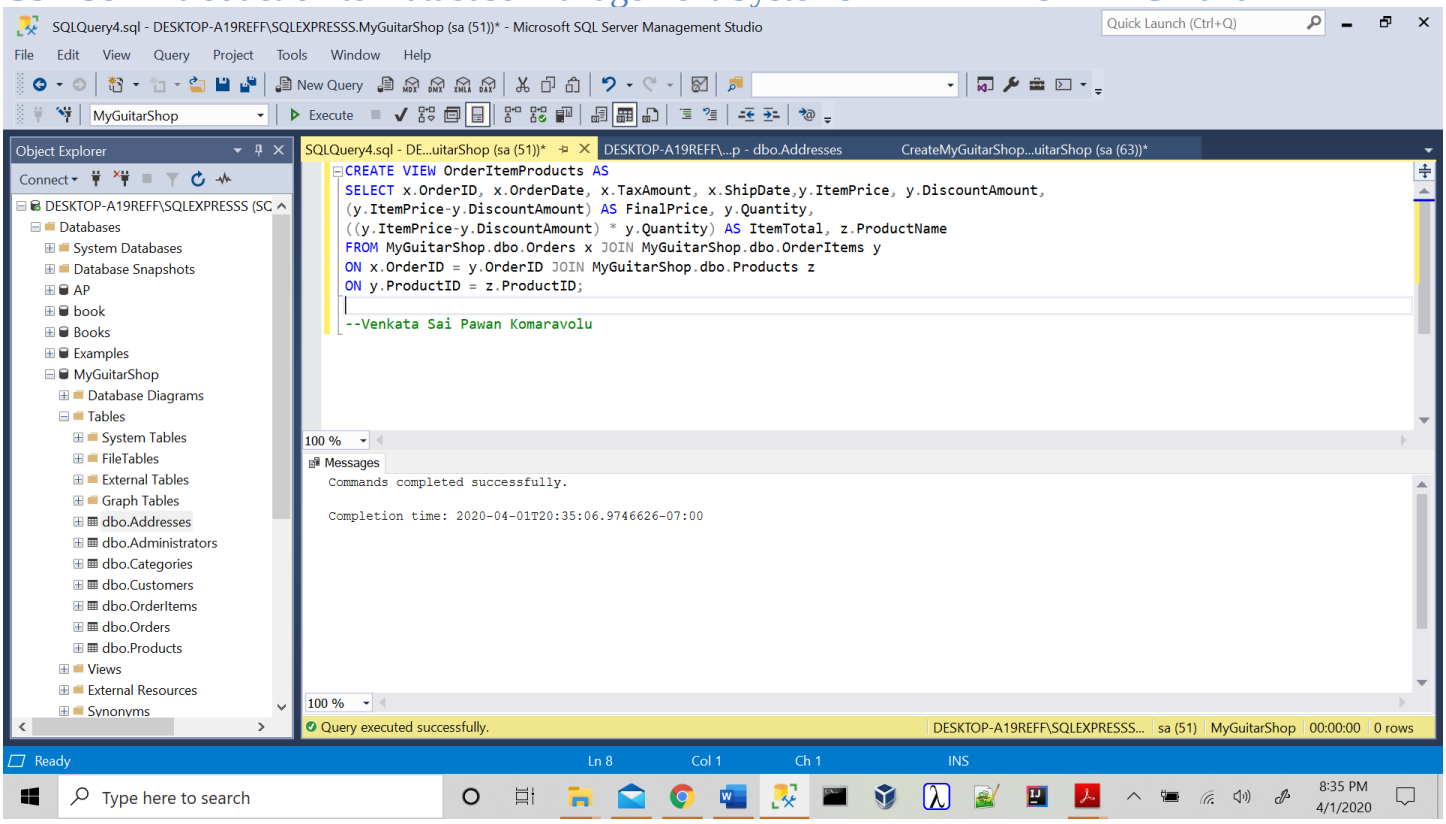


D Advanced SQL skills (views/stored procedures/functions/scripts) [24 pts.]

1. [4] Create a view named OrderItemProducts 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: 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 column from the Products table.

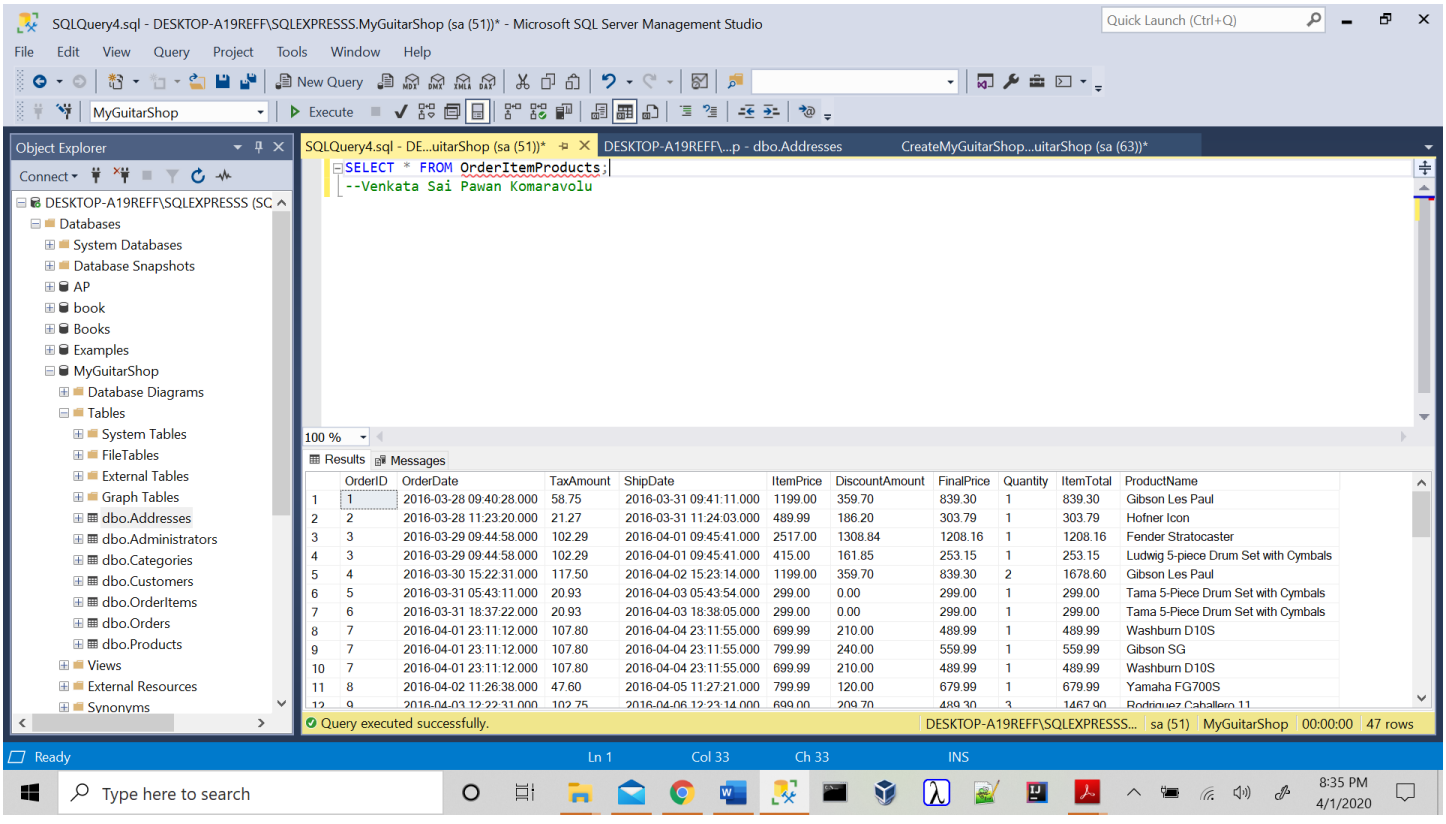
Sol: CREATE VIEW OrderItemProducts AS
SELECT x.OrderID, x.OrderDate, x.TaxAmount, x.ShipDate,y.ItemPrice, y.DiscountAmount,
(y.ItemPrice-y.DiscountAmount) AS FinalPrice, y.Quantity,
((y.ItemPrice-y.DiscountAmount) * y.Quantity) AS ItemTotal, z.ProductName
FROM MyGuitarShop.dbo.Orders x JOIN MyGuitarShop.dbo.OrderItems y
ON x.OrderID = y.OrderID JOIN MyGuitarShop.dbo.Products z
ON y.ProductID = z.ProductID;

Comments: We create a view with view name as OrderItemProducts and we select orderiD, OrderDate,TaxAmount , ShipDate ,ItemPrice , DiscoutAmount and difference between ItemPrice and DiscountAmount as FinalPrice , quantiy, and product of difference of ItemPrice and DiscountAmount as ItemTotal, ProductNamr from the orderstable,orderitemstable on common orderID and Common product ID.



SELECT * FROM OrderItemProducts;

Comments: We select everything form orderItemProducts.

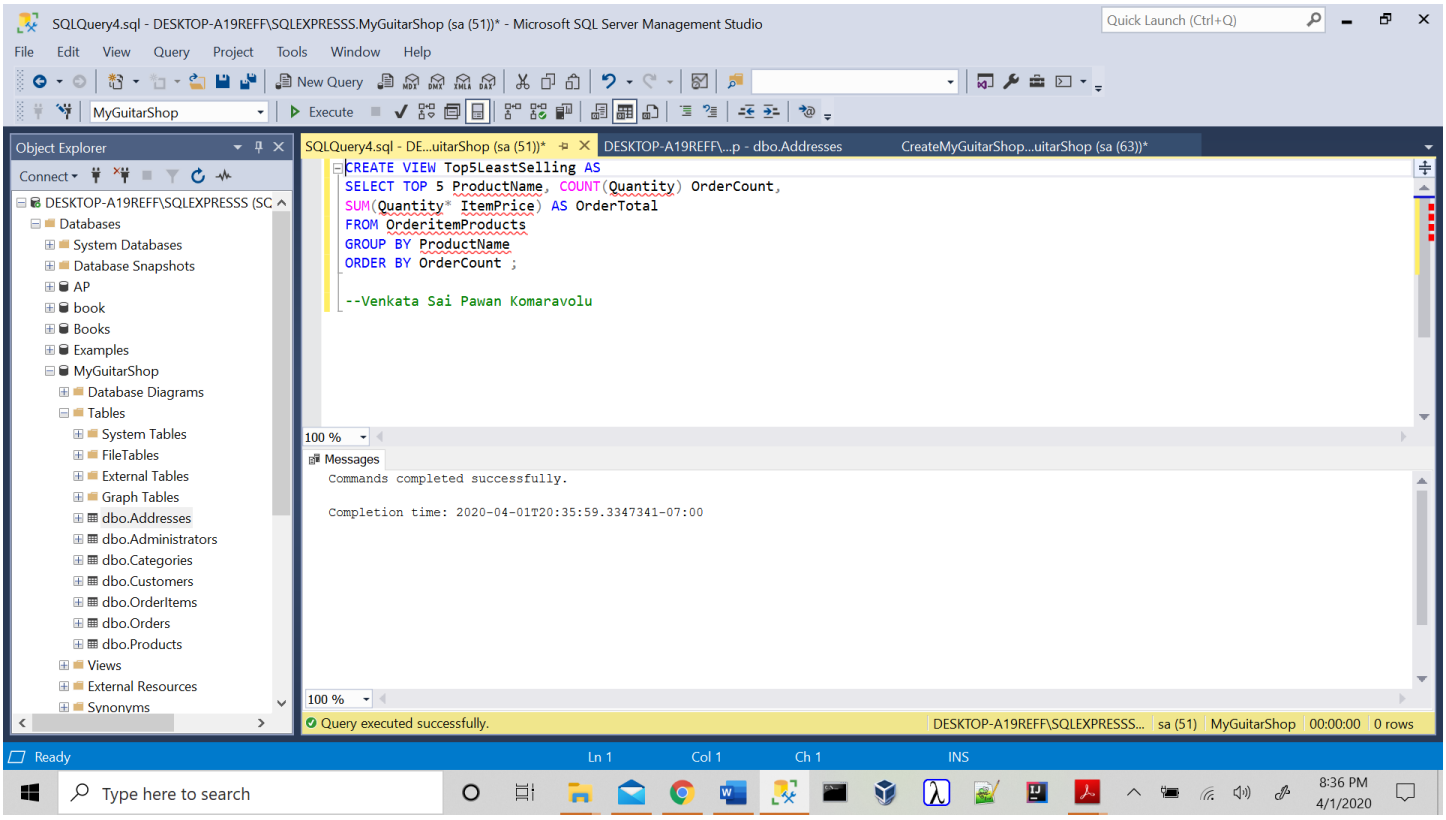


	OrderID	OrderDate	TaxAmount	ShipDate	ItemPrice	DiscountAmount	FinalPrice	Quantity	ItemTotal	ProductName
1	1	2016-03-28 09:40:28.000	58.75	2016-03-31 09:41:11.000	1199.00	359.70	839.30	1	839.30	Gibson Les Paul
2	2	2016-03-28 11:23:20.000	21.27	2016-03-31 11:24:03.000	489.99	186.20	303.79	1	303.79	Hofner Icon
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
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
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
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
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
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
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
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
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
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

2. [5] Create a view named Top5LeastSelling that uses the view you created in Section D Question 1. This view should return some summary information about five least 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).

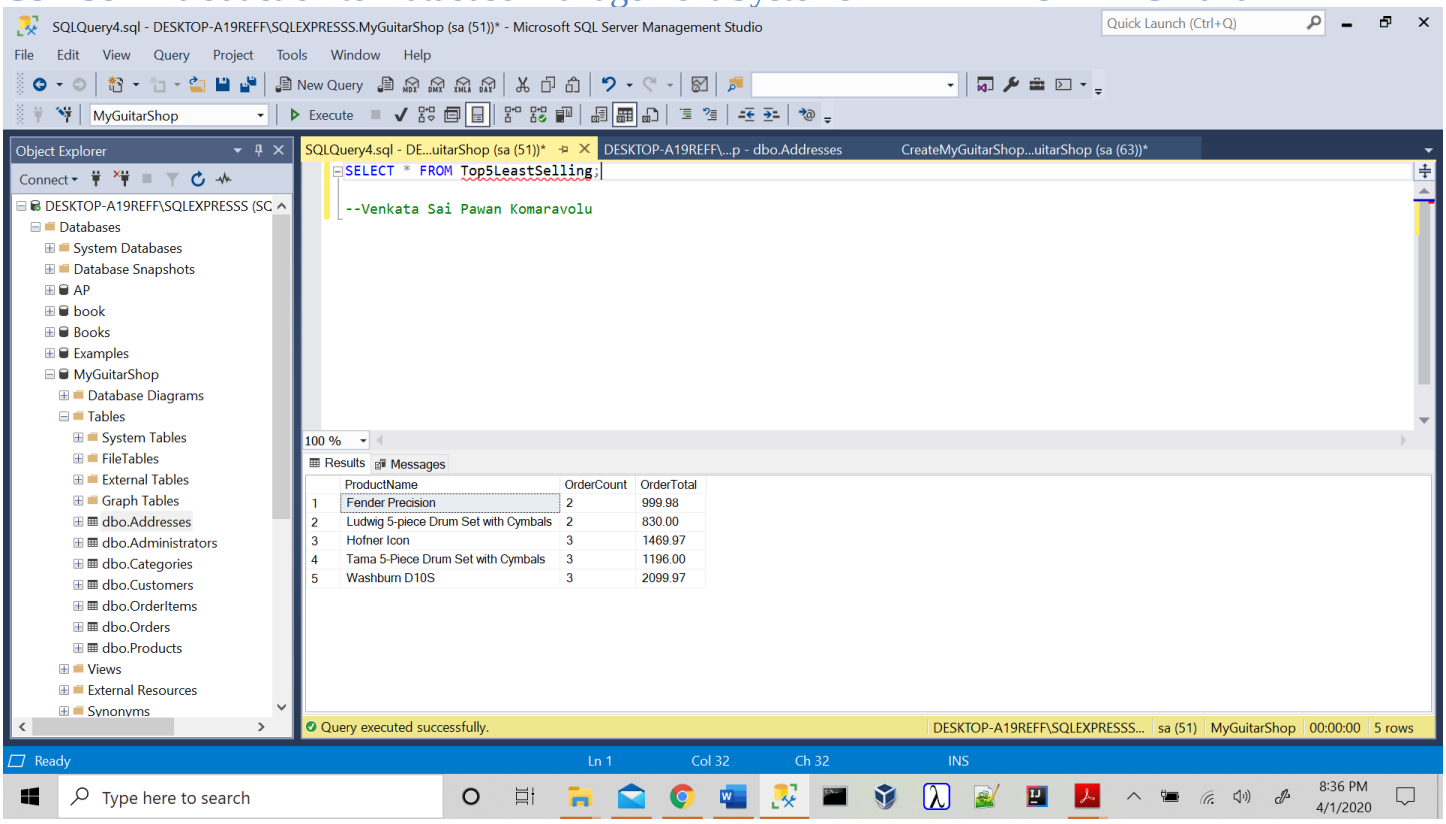
Sol: CREATE VIEW Top5LeastSelling AS
SELECT TOP 5 ProductName, COUNT(Quantity) OrderCount,
SUM(Quantity* ItemPrice) AS OrderTotal
FROM OrderitemProducts
GROUP BY ProductName
ORDER BY OrderCount ;

Comments: We create a view as Top5LeastSelling and select top 5 productname , count of quantity,ordercount,sumof product of quantity and Itemprice as ordertotal from orderitemproducts which are grouped by Productname and ordered by OrderCount.



SELECT * FROM Top5LeastSelling;

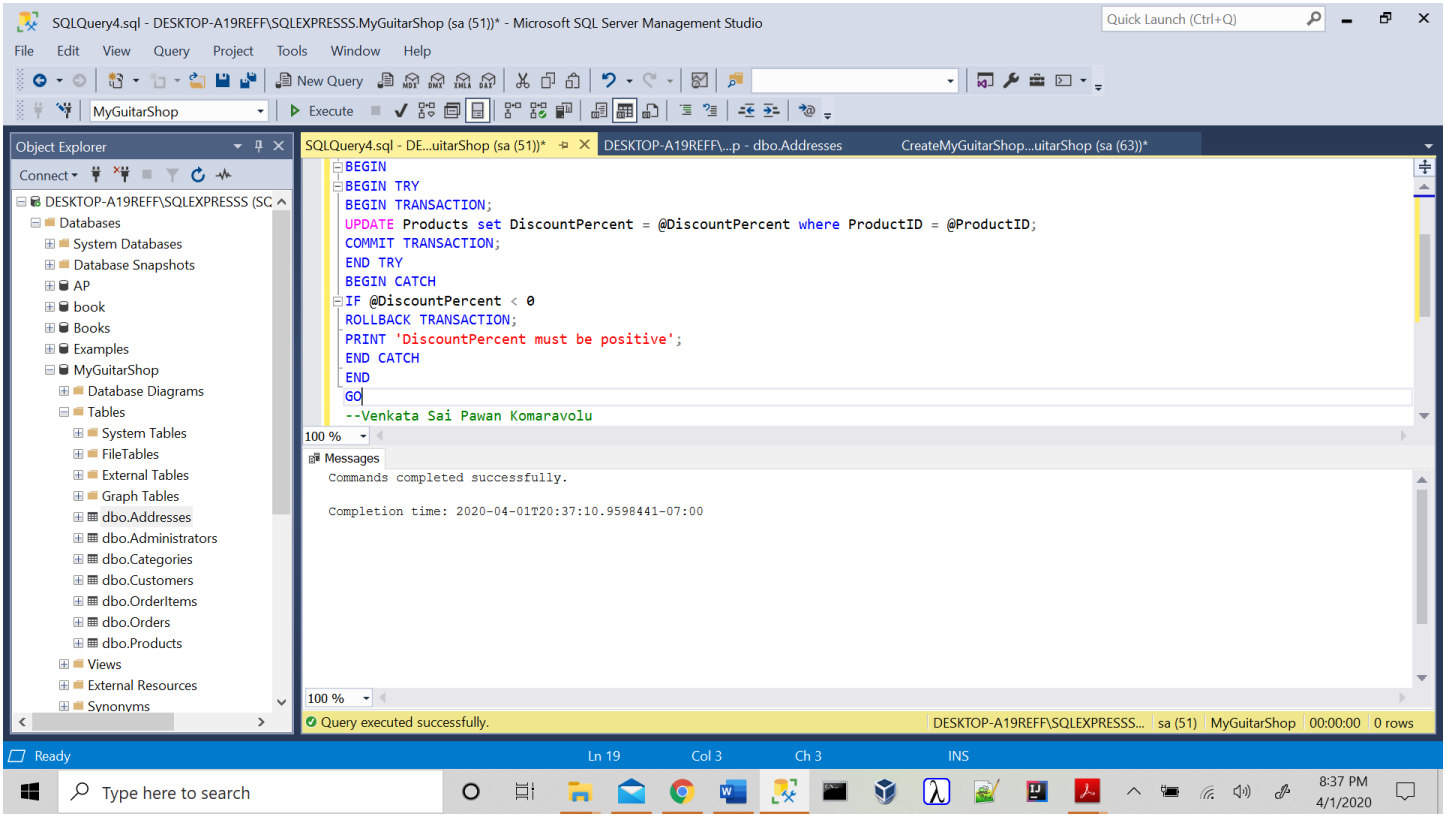
Comments: We select everything from top5LeastSelling view.



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.

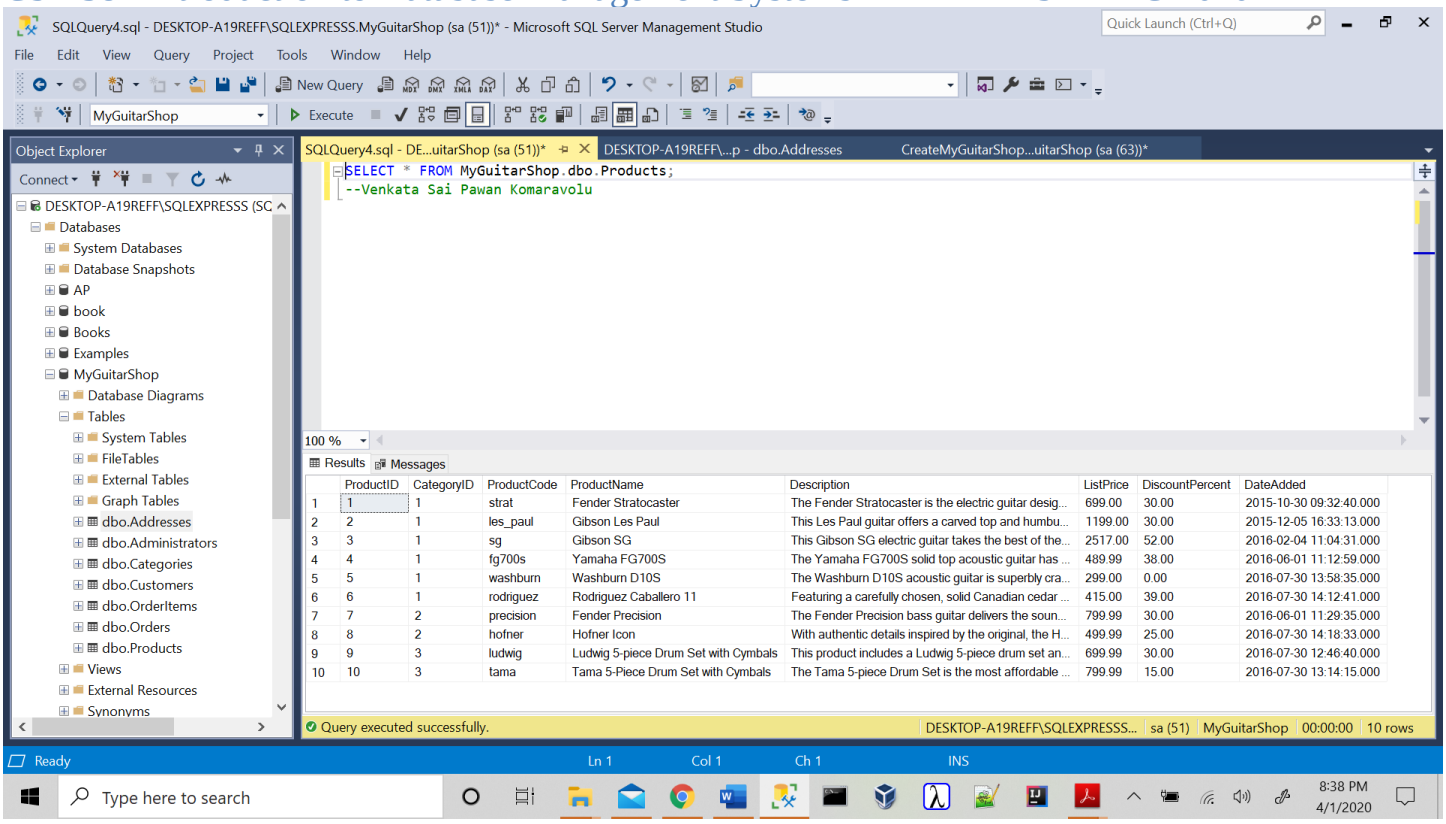
Sol: Source Code : CREATE PROCEDURE spUpdateProductDiscount
(
@ProductID int,
@DiscountPercent int
)
AS
BEGIN
BEGIN TRY
BEGIN TRANSACTION;
UPDATE Products set DiscountPercent = @DiscountPercent where ProductID = @ProductID;
COMMIT TRANSACTION;
END TRY
BEGIN CATCH
IF @DiscountPercent < 0
ROLLBACK TRANSACTION;
PRINT 'DiscountPercent must be positive';
END CATCH
END
GO

Comments: We create a procedure spUpdateProductDiscount and declare the parameters ProductID and DiscountPercent and begin the transaction and update the prodcutset Discountpercent which is equal to discountpercent where productid = parameter productid and then we commit the transaction. And end the try and we begin the cath if the the DiscountPercent is < 0 and we rollback the transaction and end.



SELECT * FROM MyGuitarShop.dbo.Products;

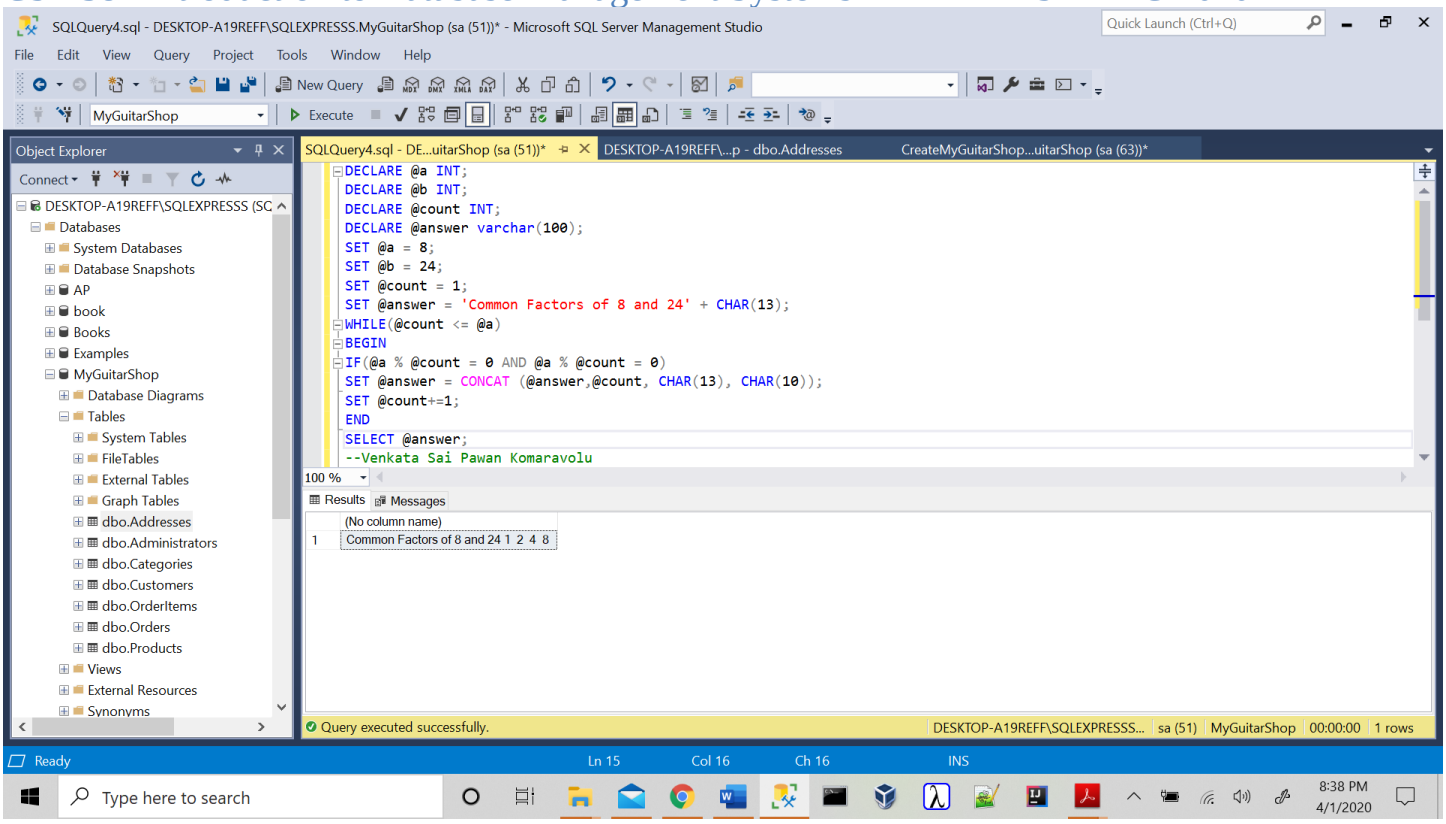
Comments: We select everything from Products table.



4. [5] Write a script that calculates the common factors between 8 and 24. 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 8 and 24 1 2 4 8

Sol: Source Code: DECLARE @a INT;
DECLARE @b INT;
DECLARE @count INT;
DECLARE @answer varchar(100);
SET @a = 8;
SET @b = 24;
SET @count = 1;
SET @answer = 'Common Factors of 8 and 24' + CHAR(13);
WHILE(@count <= @a)
BEGIN
IF(@a % @count = 0 AND @a % @count = 0)
SET @answer = CONCAT (@answer,@count, CHAR(13), CHAR(10));
SET @count+=1;
END
SELECT @answer;

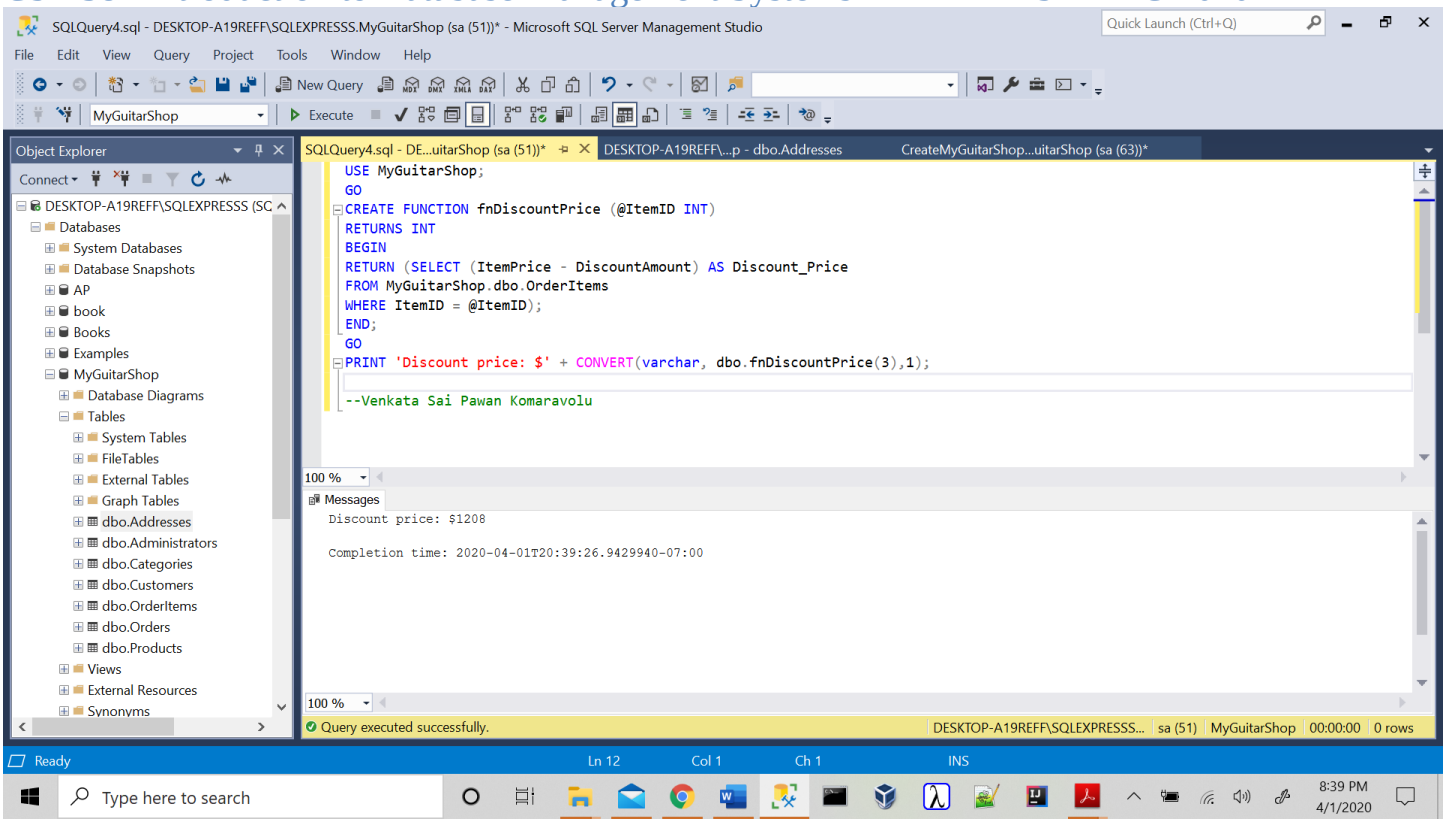
Comments: We declare a , b , count, answer and set a =8 , b = 24 , count =1 and answer = 'Common Factors of 8 and 24, whilecount <= a and we begin if a % count = 0 and set answer as concatenation of answer and count and increase the count and select the answer.



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. (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.

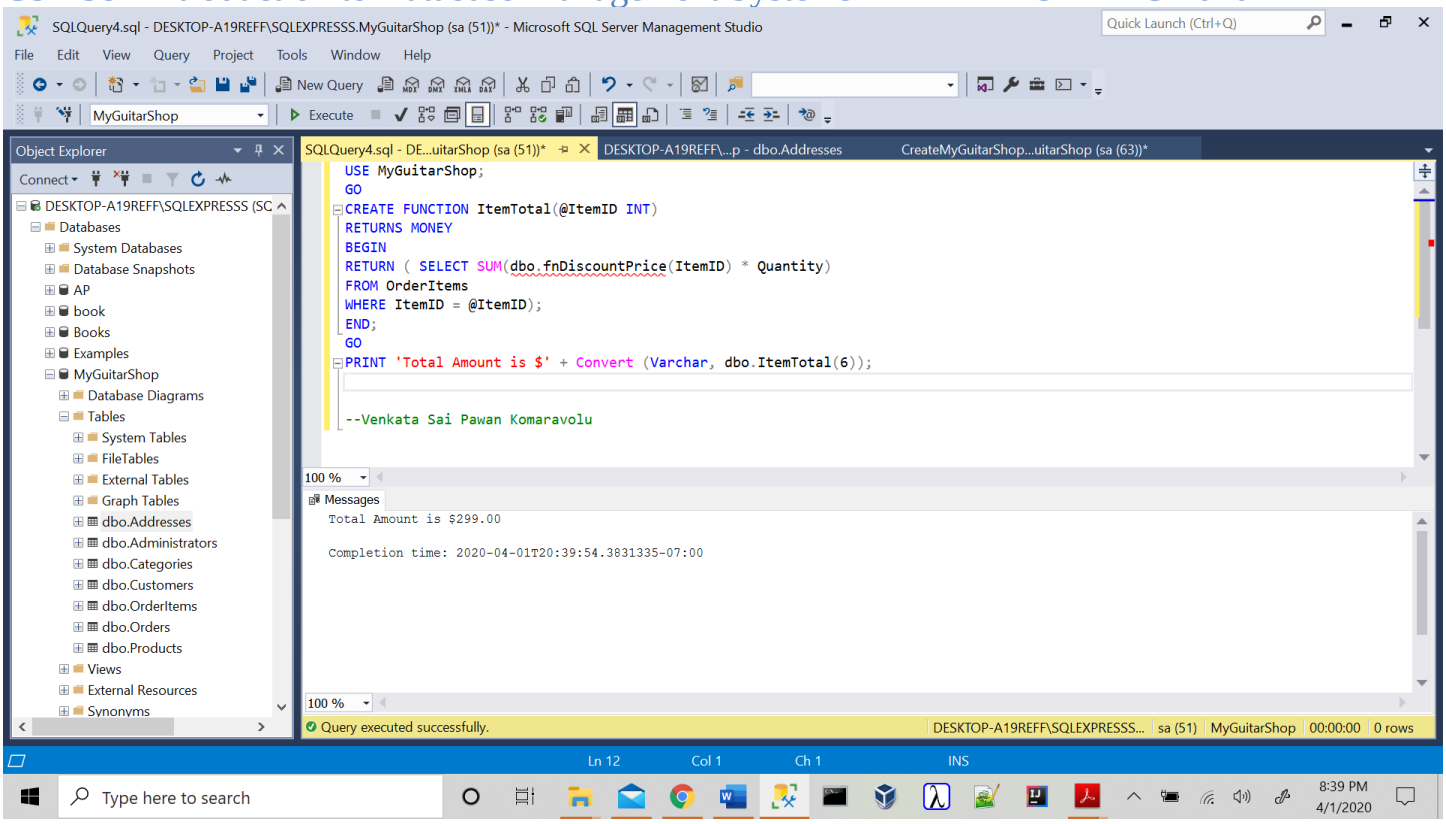
Sol: USE MyGuitarShop;
GO
CREATE FUNCTION fnDiscountPrice (@ItemID INT)
RETURNS INT
BEGIN
RETURN (SELECT (ItemPrice - DiscountAmount) AS Discount_Price
FROM MyGuitarShop.dbo.OrderItems
WHERE ItemID = @ItemID);
END;
GO
PRINT 'Discount price: $' + CONVERT(varchar, dbo.fnDiscountPrice(3),1);

Comments: We use the database MyGuitarShop and Create a function fnDiscountPrice with itemID as Paramenter that returns an INT and we begin and return the sdifference between ItemPrice and DIscountAmount as Discount Price form the OrderItems table on Common Item ID and print 'Discount price:



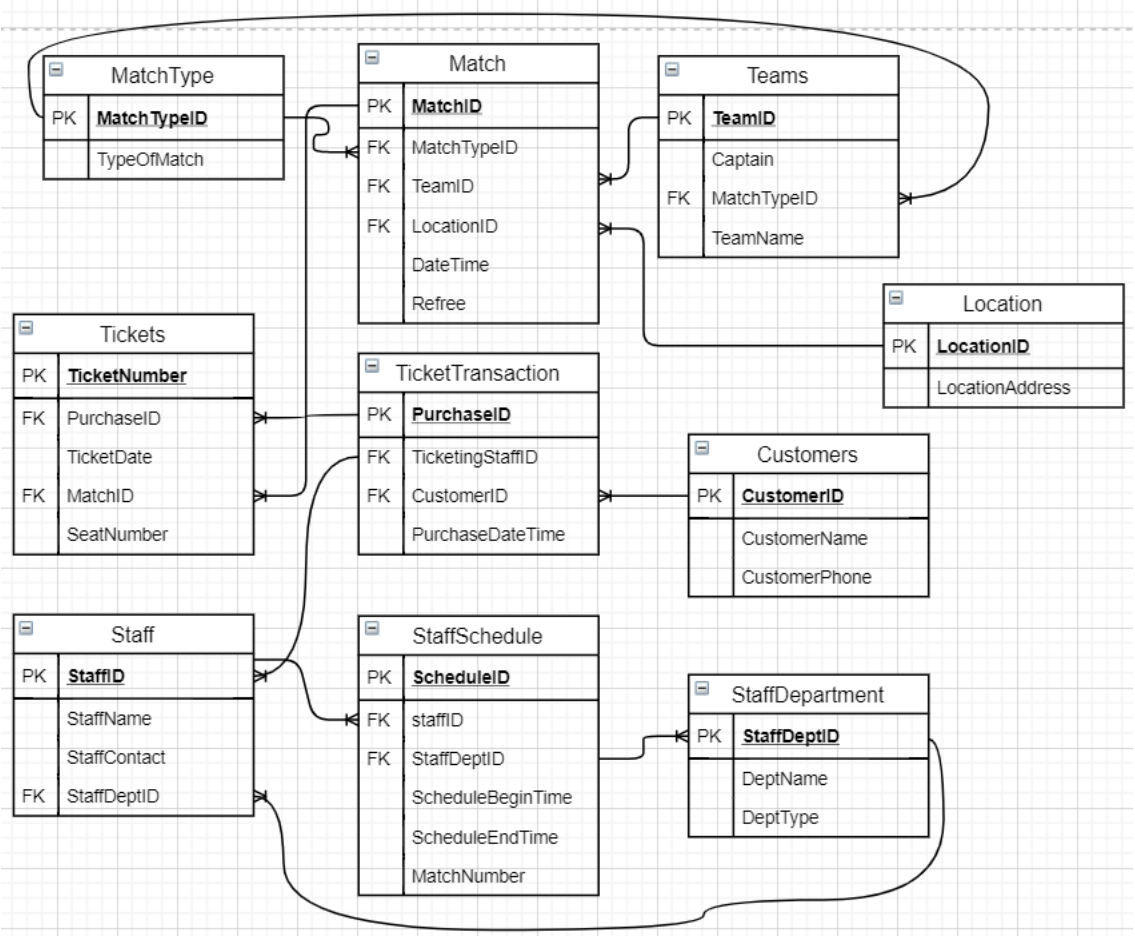
Source: USE MyGuitarShop;
GO
CREATE FUNCTION ItemTotal(@ItemID INT)
RETURNS MONEY
BEGIN
RETURN (SELECT SUM(dbo.fnDiscountPrice(ItemID) * Quantity)
FROM OrderItems
WHERE ItemID = @ItemID);
END;
GO
PRINT 'Total Amount is $' + Convert (Varchar, dbo.ItemTotal(6));

Comments: We use MyGuitarShop Database and Create a function ItemTotal with ItemID as Parameter that returns money and begin the return sum of all DiscountPriceItemID * quantity from OrderItems on Common ItemID and Print 'Total Amount.



II. Database Design [20 pts.] Create a sample database design for Cuse's Box Office for all Sports matches (Basketball, Soccer, Lacrosse, etc) and draw one database model for it. The Box Office needs to keep a track of all types of sports matches taking place at the Dome as well as all other 'away' locations. By tracking the type of sport, its location, and the teams (and their captains) involved per match, the Box Office wants to sell tickets for all these matches. Each ticket transaction will have different ticketing-staff and customers involved with all critical information per customer, ticketing-staff, and transaction recorded. Similarly, each match will also have other staff allotted: security-staff and café-staff. All the three types of staff members (ticketing-staff, security-staff, café-staff) will have schedules allotted. Your design could add more things to the existing requirements, but all the given requirements should be first 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.



Here is the complete screenshot of my final design model with all the tables and its attributes and the relationships among the tables.

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

In my database design, I have used the following tables: MatchType, Match, Teams, Location, Tickets, TicketTransaction, Customers, Staff , StaffSchedule,StaffDepartment.

For MatchType table:-
In this table, I have used MatchTypeID and TypeofMatch as columns.
Primary key used in this table is MatchTypeID.

For Match table:-
In this table, I have used MatchID,MatchTypeID, TeamID, LocationID, DateTime and Referee as columns.
Primary keys used in this table are MatchID
Foreign keys used in this table are MatchTypeID,LocationID,TeamID.

For Teams table:-
In this table, I have used TeamID, MatchTypeID and Captain,TeamName as columns.
Primary key in this table is TeamID

For Locations table:-
In this table, I have used LocationID and LocationAddress as columns.
Foreign key in this table is LocationID.

For Tickets table:-
In this table, I have used TicketNumber, TicketDate, MatchID, SeatNumber and PurchaseID as columns.
Primary key in this table is TicketNumber
Foreign key in this table is MatchID.

For TicketTransaction table:-
In this table, I have used PurchaseID, CustomerID, PurchaseDateTime, TicketingStaffID as columns.
Primary key in this table is PurchaseID

Foreign keys in this table are TicketingStaffID,CustomerID.

For Customers table:-
In this table, I have used CustomerID, CustomerName, CustomerPhone as columns.
Primary key in this table is CustomerID.

For Staff table:-
In this table, I have used StaffID, StaffName, StaffContact and StaffDeptID as columns.

Primary Keys in this table is StaffID
Foreign keys in this table are StaffDeptID.

For StaffSchedule table:-
In this table, I have used ScheduleID, StaffID, StaffDeptID, ScheduleBeginTime and ScheduleEndTime as columns.
Primary key in this table is ScheduleID.

Foreign Key in this table is StaffID and StaffDeptID

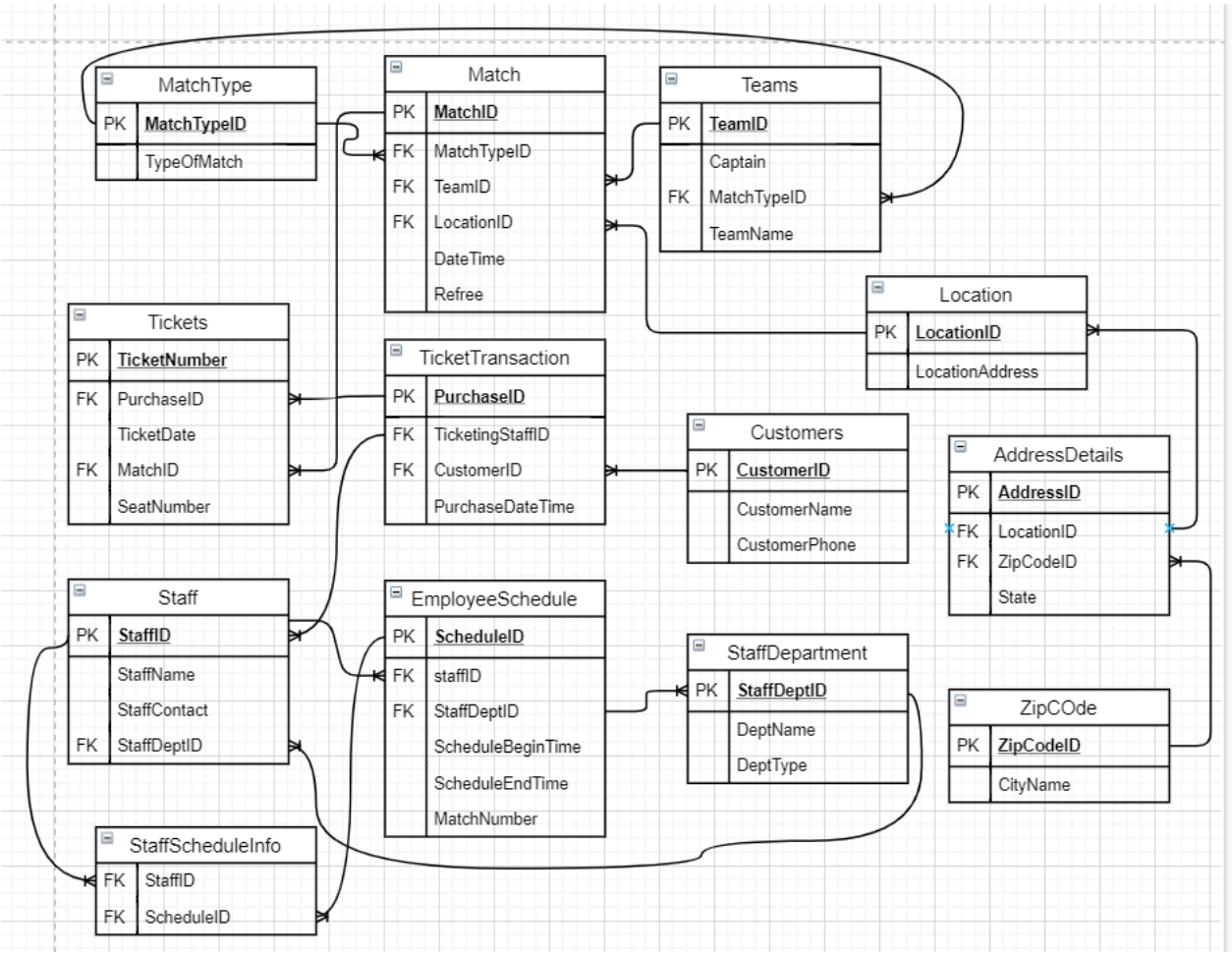
For StaffDepartment table:-
In this table, I have used StaffDeptID, DeptName and DeptType as columns.
Primary key in this table is StaffDeptID.

In this database, I have used 10 relations. They are:

- There is a one-to-many relationship between MatchType and Match.
- There is a one-to-many relationship between MatchType and Teams.
- There is a one-to-many relationship between Match and Teams.
- There is a one-to-many relationship between Locations and Matches.
- There is a one-to-many relationship between Match and Tickets.
- There is a one-to-many relationship between Tickets and TicketTransaction.
- There is a one-to-many relationship between Customer and TicketTransaction.
- There is a many-to-one relationship between Staff and TicketTransaction.

- There is a one-to-many relationship between Staff and StaffSchedule.
- There is a many-to-many relationship between Staff and StaffDepartment.

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



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

The database which I've designed in the first are not in 1NF,2NF or 3NF due to many redundancies .

In order to normalize the above design into third normal form, I have introduced 3 new tables StaffScheduleInfo , AddressDetails and ZipCode.

For StaffScheduleInfo:-
In this table, I have used StaffID and ScheduleID as columns.
Foreign keys in this table are StaffID and ScheduleID.

For AddressDetails table:-
In this table, I have used AddressID, LocationID and ZipCodeID , state as columns.
Primary key in this table is AddressID.
Foreign key in this table is ZipCodeID,LocationID.

For ZipCode table:-

In this table, I have used ZipCodeID and CityName as columns.
Primary key in this table is ZipCodeID.

For the database to be in First Normal form, all the table elements should contain only scalar values. All the tables contains scalar valuesHence the Database is already in 1NF.

For the database to be in SecondNormalForm, it should be in 1-NF and all non-key attributes must depend on the entire primary key but not on the partial partial primary key. In the previous design, LocationAddress column in the Location table will not follow this and it can be further divided into State, ZipCode and CityName. So by creating a new table AddressDetails, for storing all the above details, we will convert our database into second normal form.

For any database to be in ThirdNormalForm, it should be in 1-NF, 2-NF and every column should depend only on the primary key and should not depend on the non-primary key. So here in my design ZipCode and CityName columns in a AddressDetails table are dependent on each other. For example if ZipCode changes CityName value changes and also some set of cities have same ZipCode. So to eliminate this, we move the ZipCode and CityName columns into a new table named ZipCode.

There are 4new relationships added between the tables:

- There is a one-to-many relationship between Staff and StaffScheduleInfo.
- There is a one-to-many relationship between StaffSchedule and StaffScheduleInfo.
- There is a one-to-many relationship between AddressDetails and Customers.
- There is a one-to-many relationship between ZipCode and AddressDetails.

Remarks: I've learnt to setup the database , get the column definition of various address table , Administor table , Categories table , Customers table , Orderitems table , orders table and products table and also used SQL query to display details of categories and products table. I've learnt to retrieve data from the database and learnt the use of LIKE operator usage of null ,not null values , usage of correlated queries , usage of AVG ,SUM , MAX, MIN aggregate functions , subqueries, Cast , Convert functions , Left, Right functions , Between functionality and also usage of dates, Insert and Update DML statements , Usage of views and selecting query to display view contents, creating a procedure and using procedure to display the contents using the procedure, creating function and designing a database and writing the relationship between the tables and converting it into 1NF, 2NF,3 NF and finally describing each table and rerlationship between the tables.