Task: CSE 581 Spring 2018 Project 1
Deadline: Monday, April 9 at 12.00 PM Eastern Time

==========

Student Name : Sonal Patil
SUID : 997435672

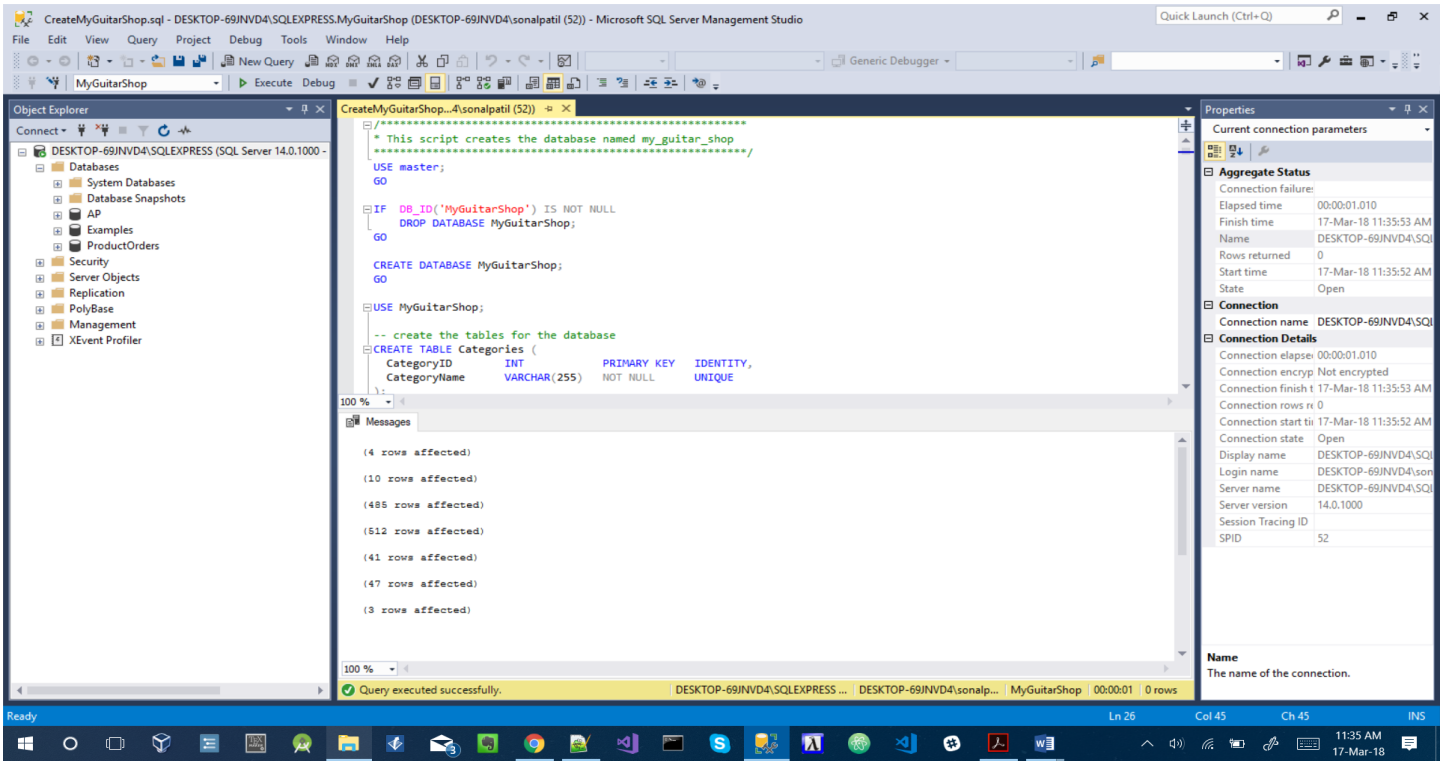
==========

I. My Guitar Shop Database

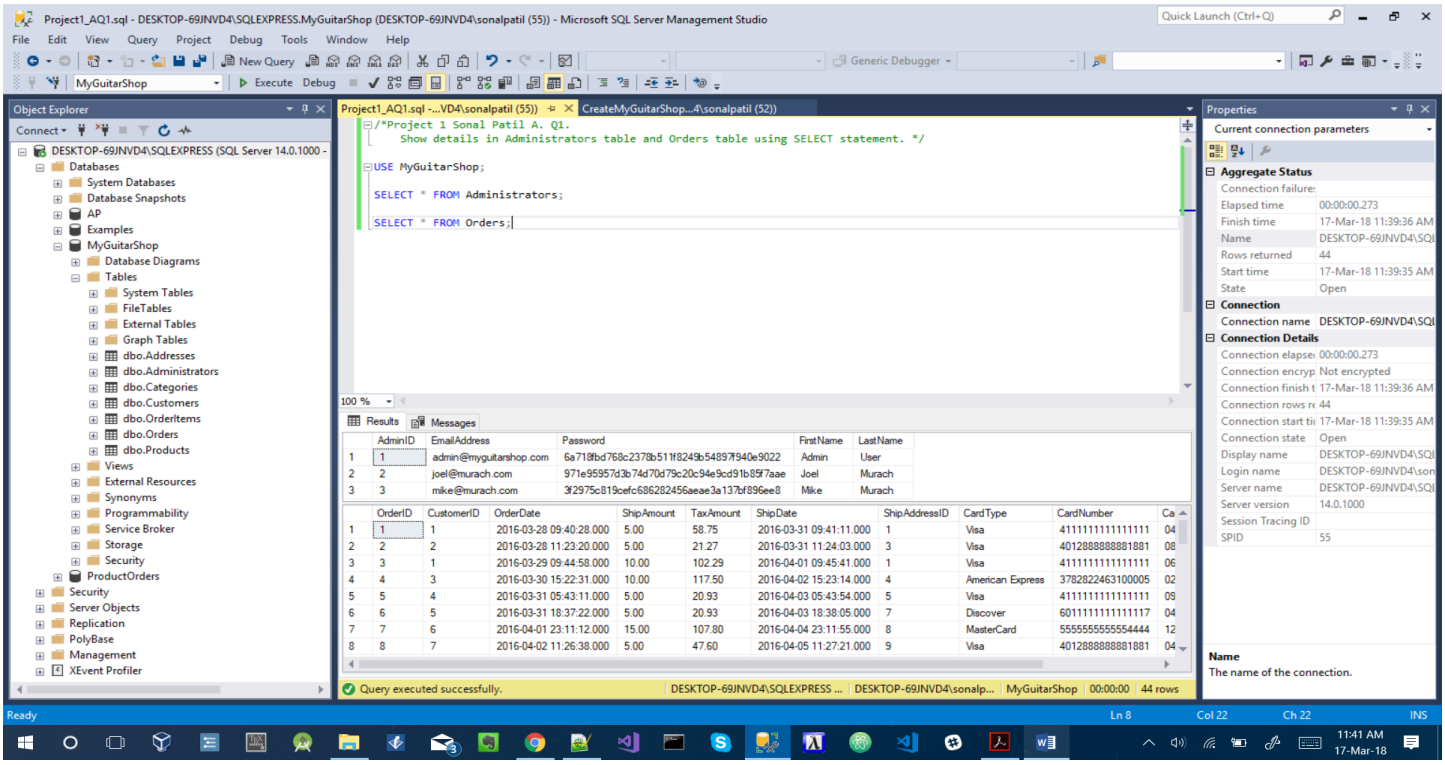
In part I, you will use SQL Server Management Studio to create the MyGuitarShop database, to review the tables in the MyGuitarShop database, and to enter SQL statements and run them against this database.

A. Database Setup [2 pts.]

1. Download CreateMyGuitarShop.sql from Blackboard Project 1 directory and open it in SQL server management studio. Execute the entire script by clicking the Execute button in the SQL Editor toolbar or by pressing F5. Show the message in the Message tab, indicating the script is executed successfully. A full screenshot of execution result is required.



Navigate through the database objects and view the column definitions for each table. Open a new Query Editor window. Show details in Administrators table and Orders table using SELECT statement. A full screenshot of execution result is required.

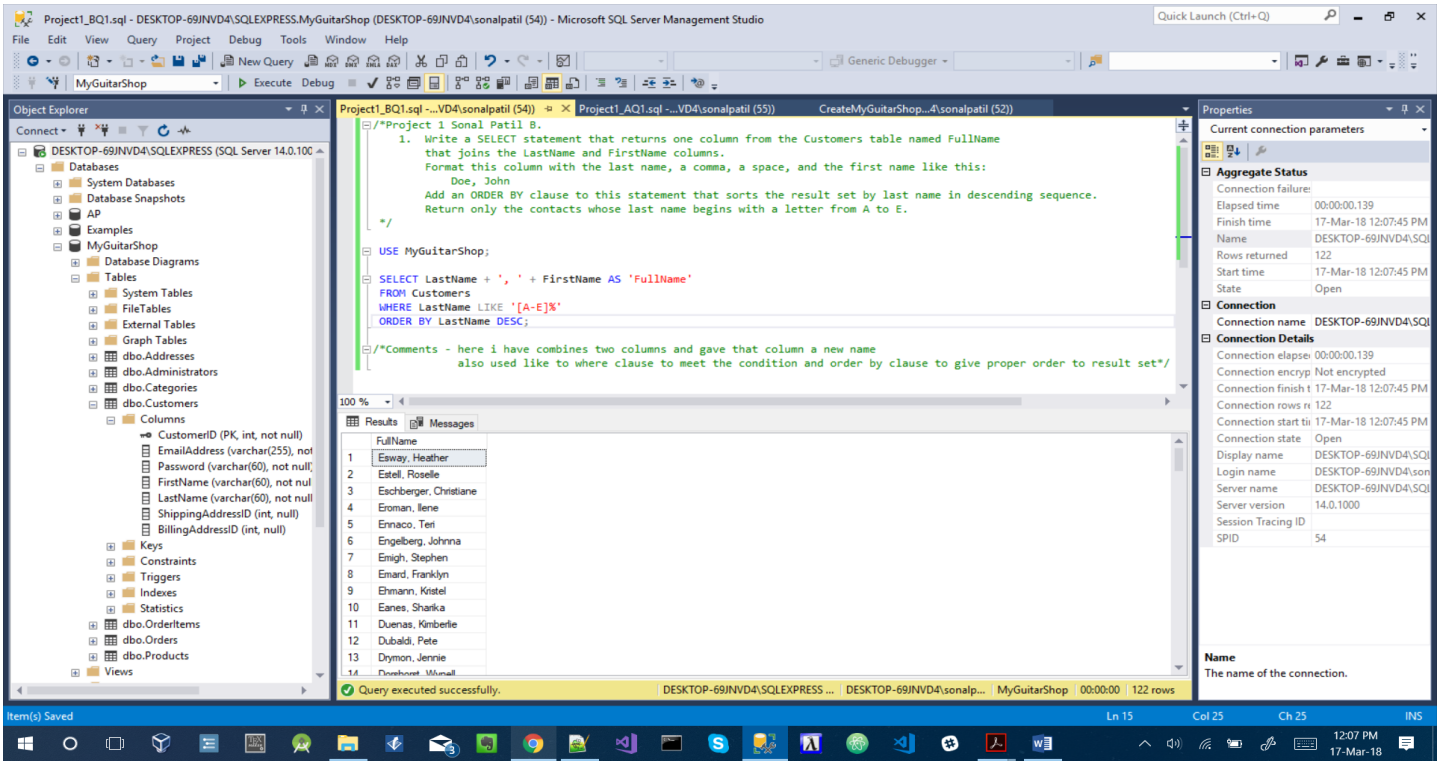


B. An introduction to SQL [6 pts.]

1. 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 comma, a space, and the first name like this:

Doe, John

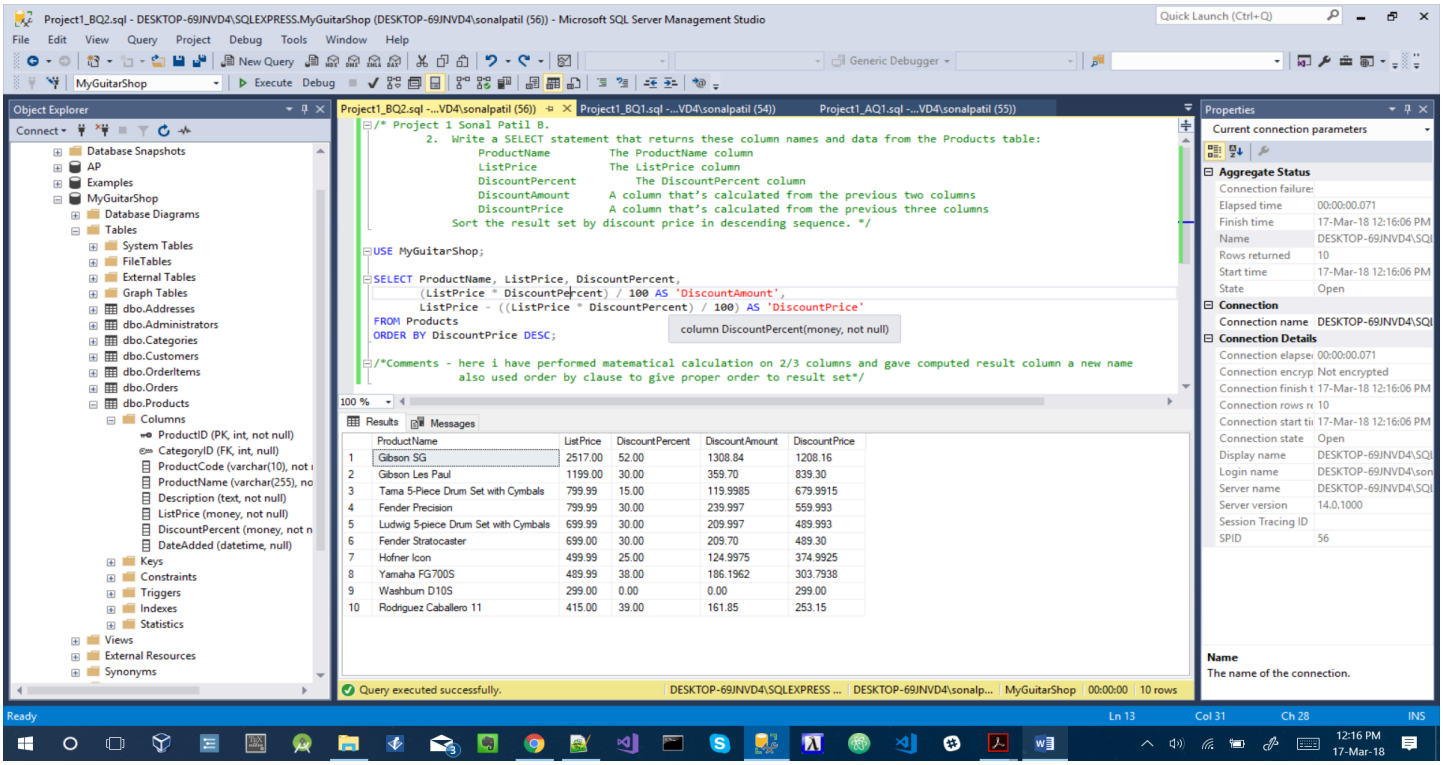
Add an ORDER BY clause to this statement that sorts the result set by last name in descending sequence. Return only the contacts whose last name begins with a letter from A to E.



2. Write a SELECT statement that returns these column names and data from the Products table:

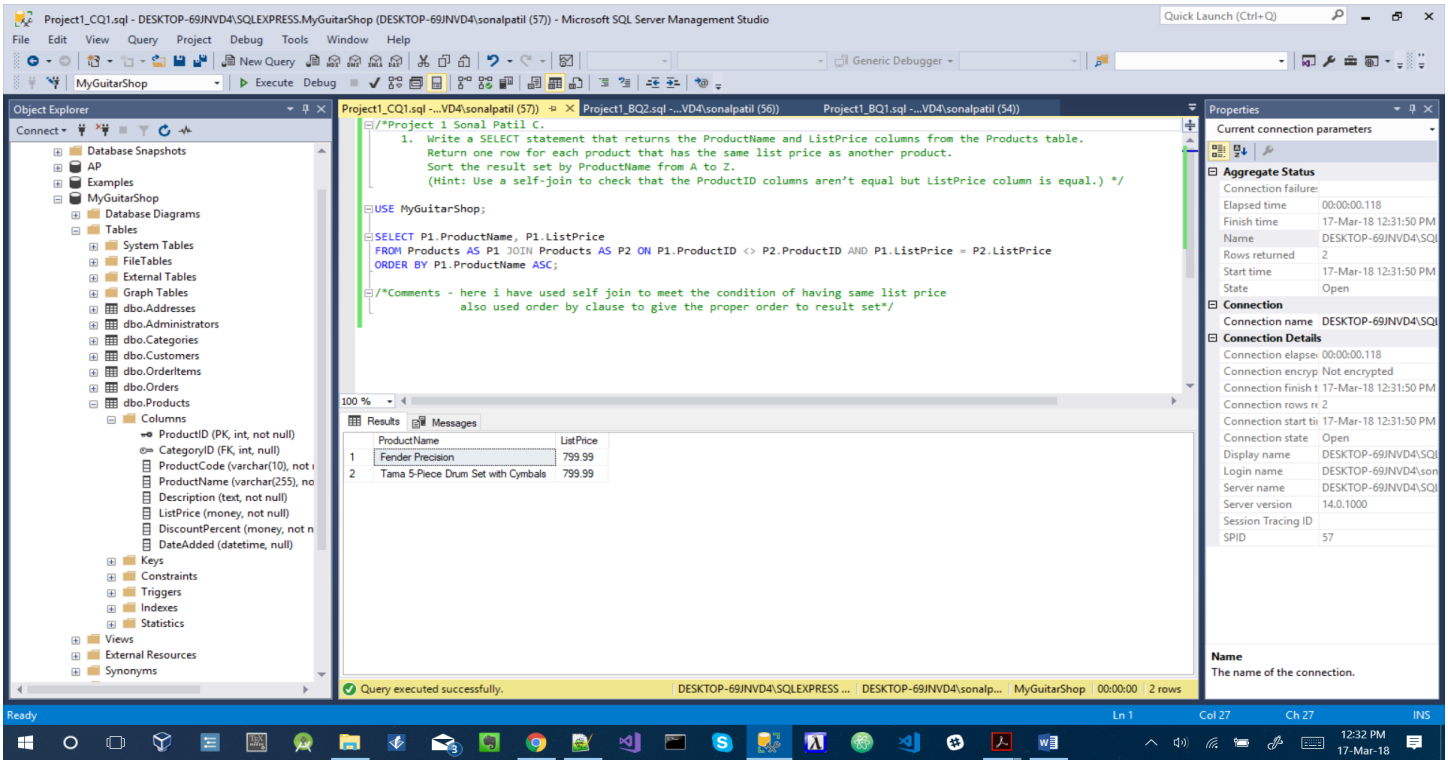
ProductName	The ProductName column
ListPrice	The ListPrice column
DiscountPercent	The DiscountPercent column
DiscountAmount	A column that's calculated from the previous two columns
DiscountPrice	A column that's calculated from the previous three columns

Sort the result set by discount price in descending sequence.

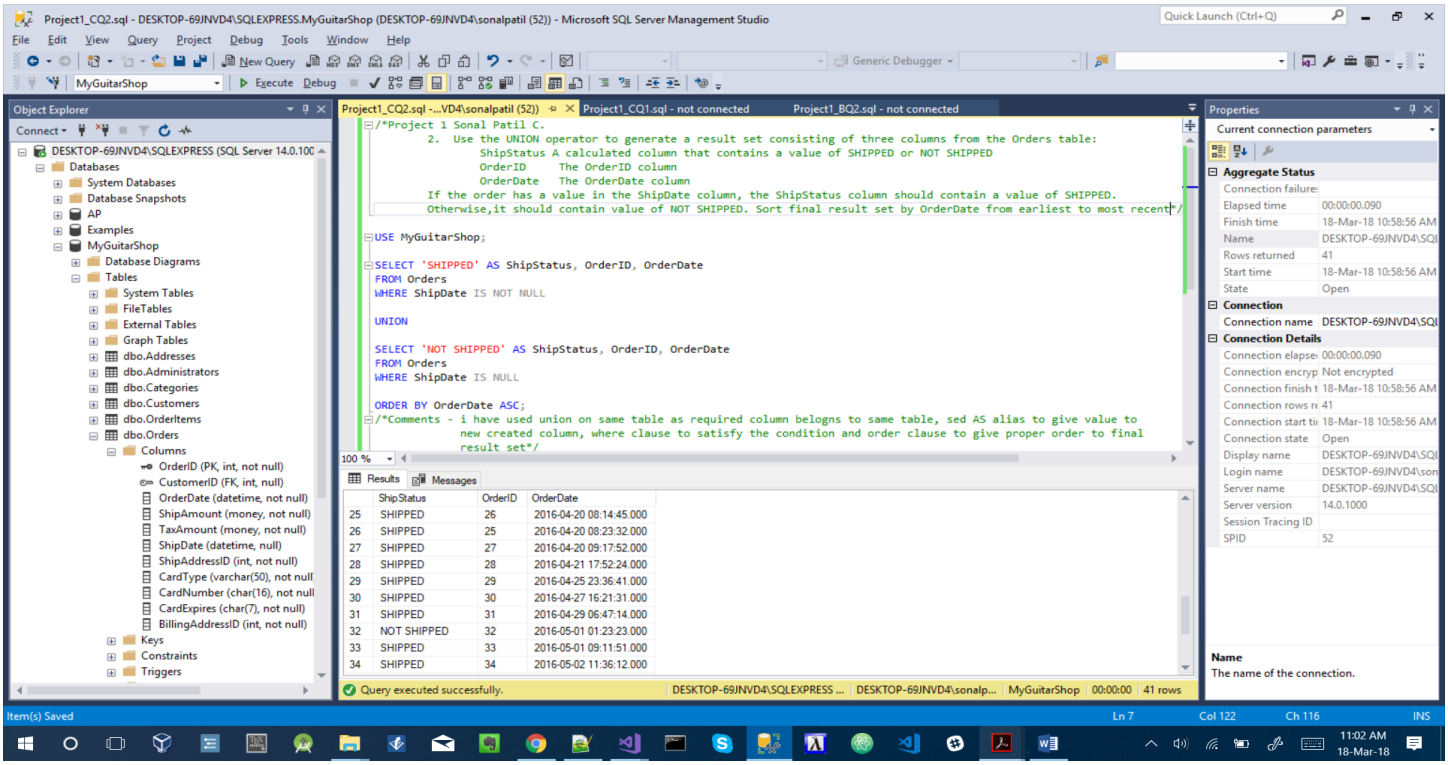


C. The essential SQL skills [44 pts.]

1. Write a SELECT statement that returns the ProductName and ListPrice columns from the Products table. Return one row for each product that has the same list price as another product. Sort the result set by ProductName from A to Z.
(Hint: Use a self-join to check that the ProductID columns aren't equal but ListPrice column is equal.)

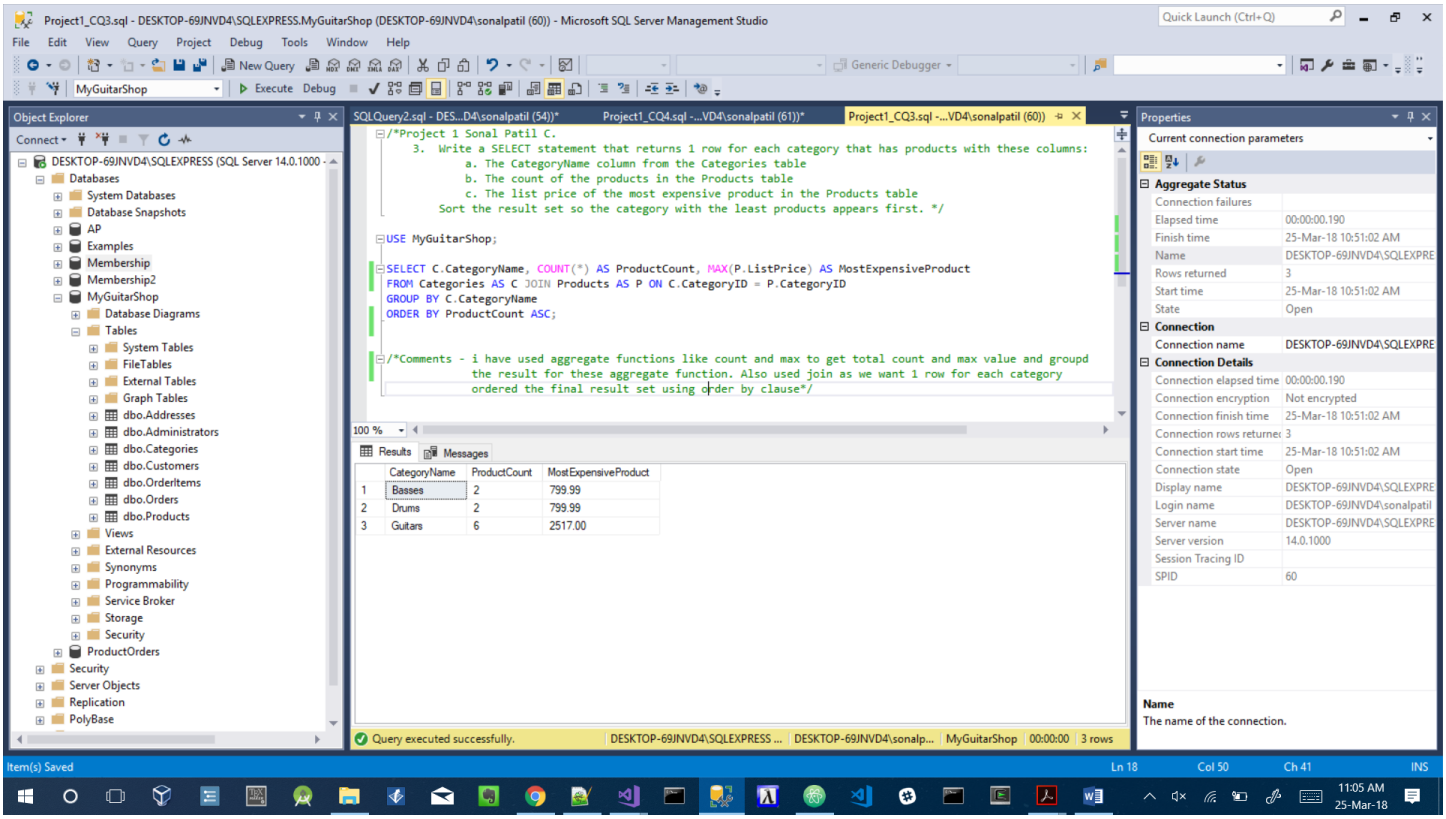


2. Use the UNION operator to generate a result set consisting of three columns from the Orders table:
ShipStatus A calculated column that contains a value of SHIPPED or NOT SHIPPED
OrderID The OrderID column
OrderDate The OrderDate column
If the order has a value in the ShipDate column, the ShipStatus column should contain a value of SHIPPED. Otherwise, it should contain a value of NOT SHIPPED. Sort the final result set by OrderDate from earliest to most recent.



3. Write a SELECT statement that returns 1 row for each category that has products with these columns:
    a. The CategoryName column from the Categories table
    b. The count of the products in the Products table
    c. The list price of the most expensive product in the Products table

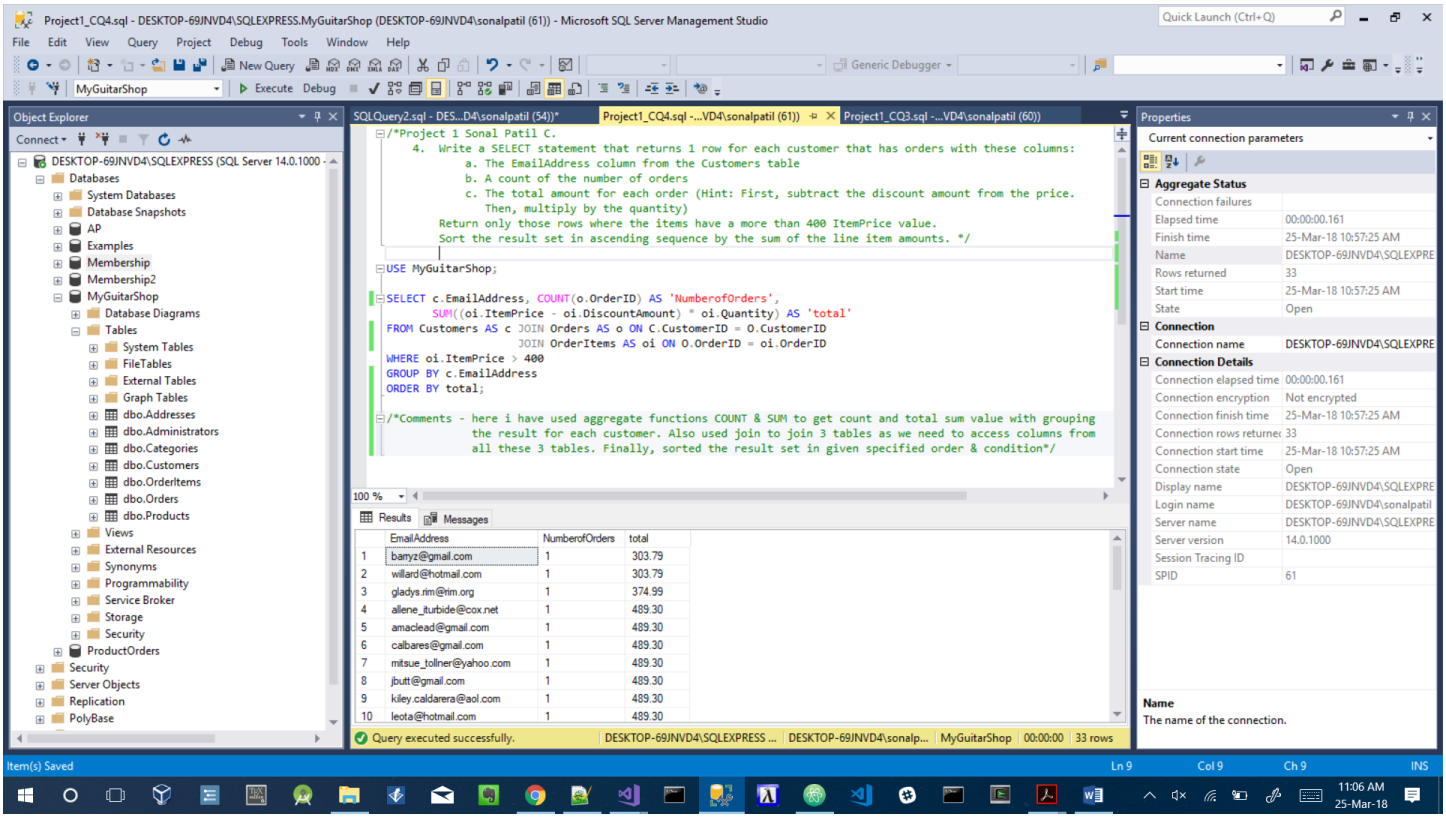
   Sort the result set so the category with the least products appears first.



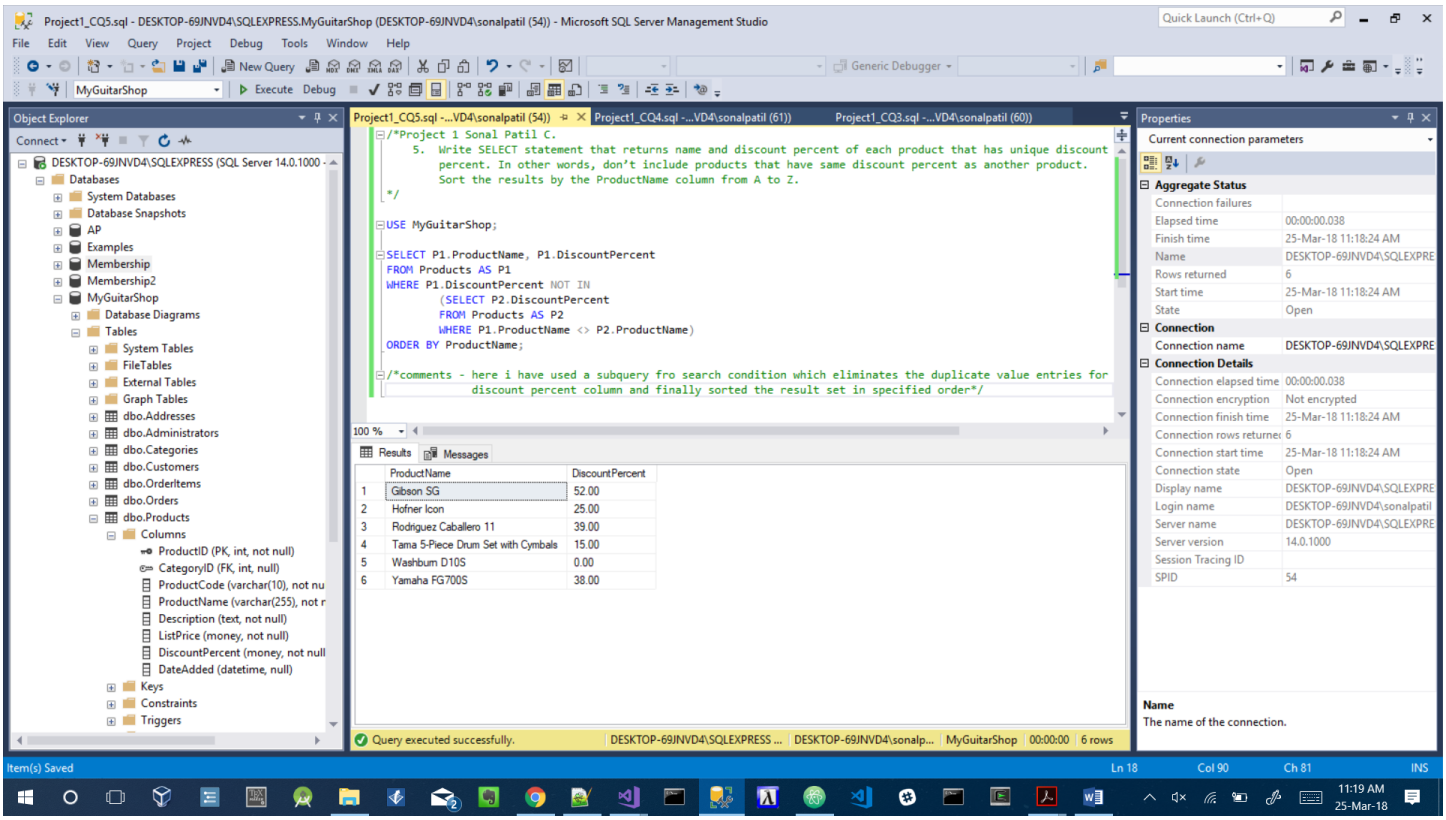
4. Write a SELECT statement that returns 1 row for each customer that has orders with these columns:
    a. The EmailAddress column from the Customers table
    b. A count of the number of orders
    c. The total amount for each order (Hint: First, subtract the discount amount from the price. Then, multiply by the quantity)

   Return only those rows where the items have a more than 400 ItemPrice value.

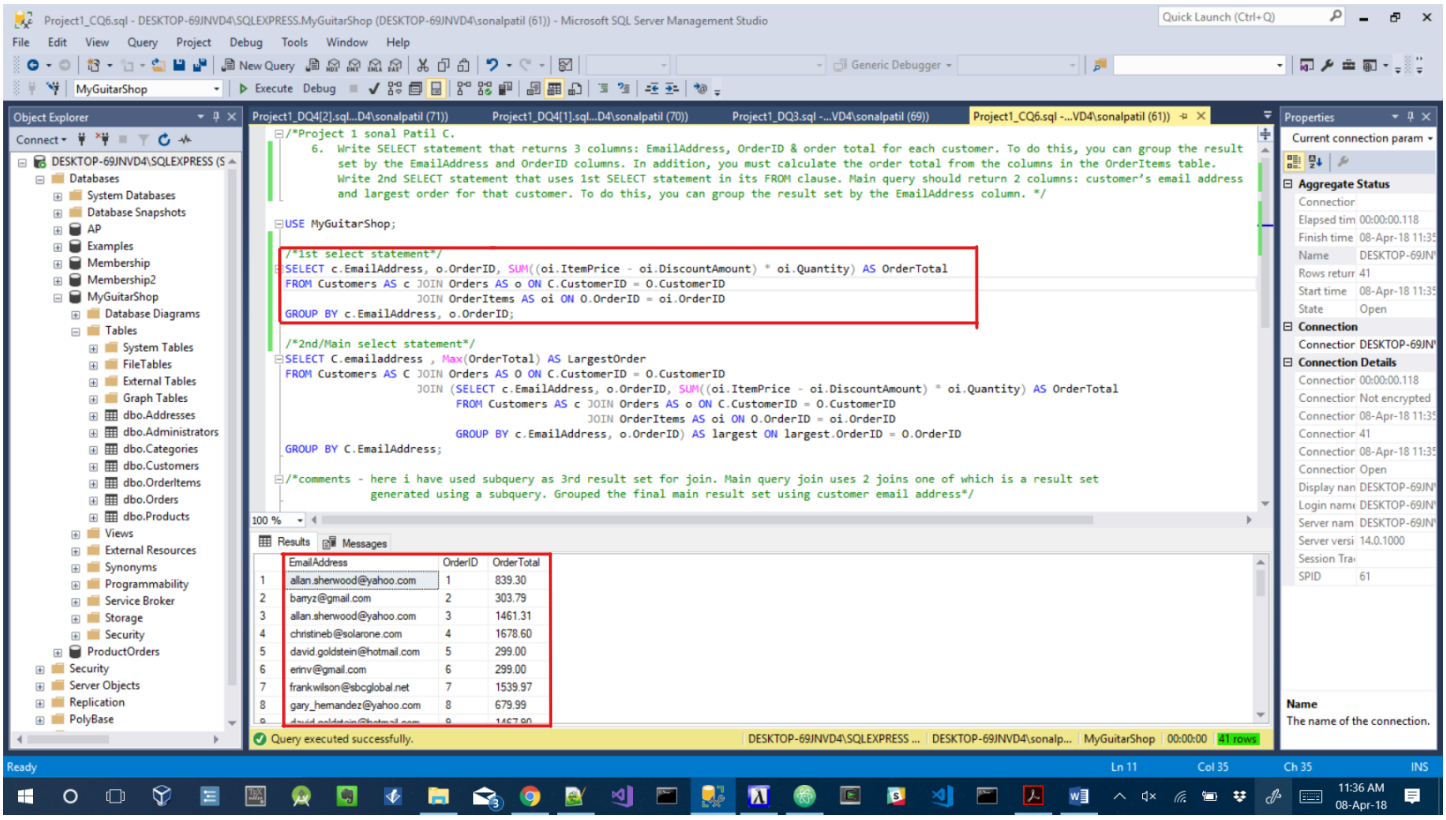
   Sort the result set in ascending sequence by the sum of the line item amounts.

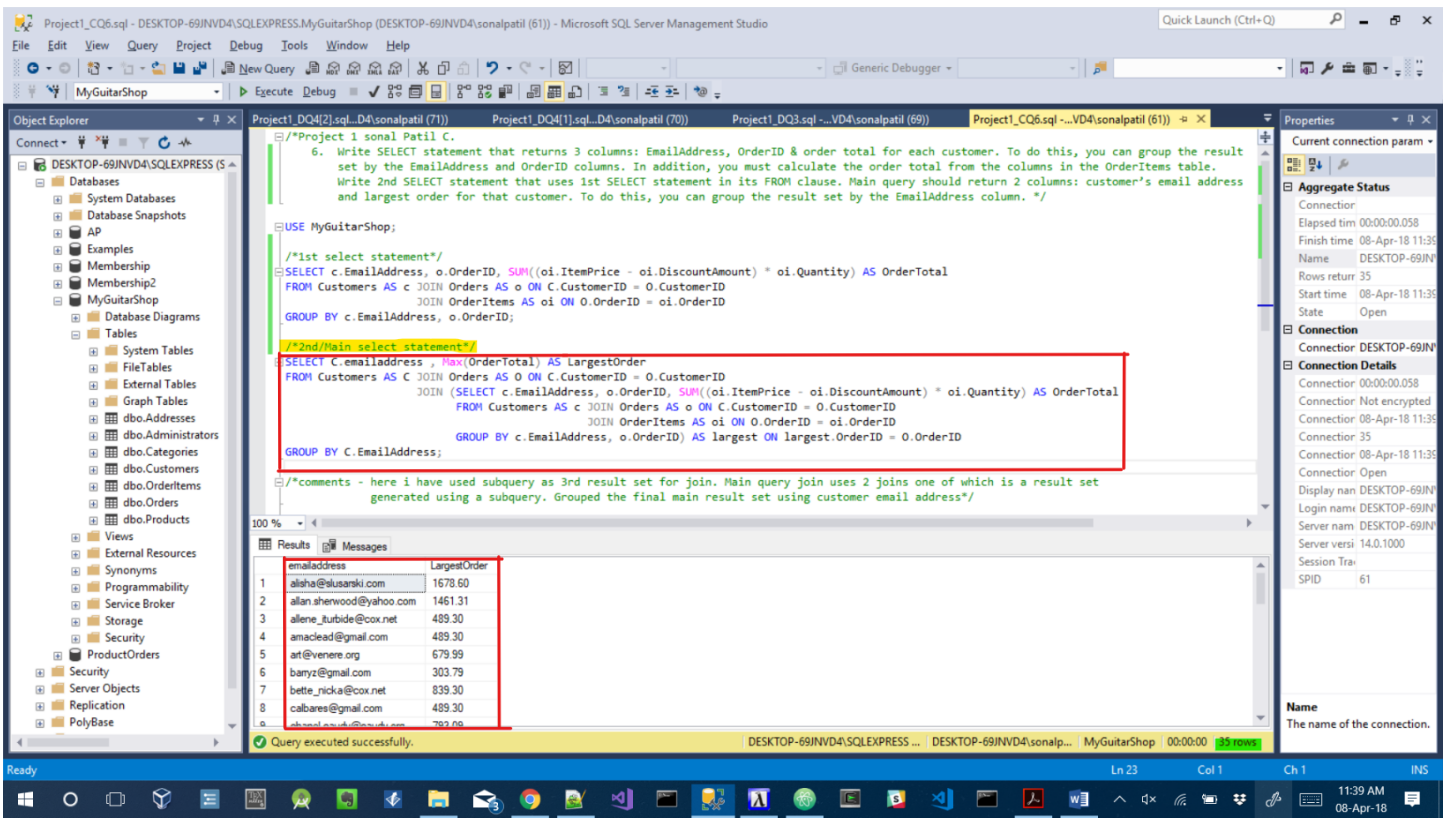


5. Write a SELECT statement that returns the name and discount percent of each product that has a unique discount percent. In other words, don't include products that have the same discount percent as another product.
   Sort the results by the ProductName column from A to Z.



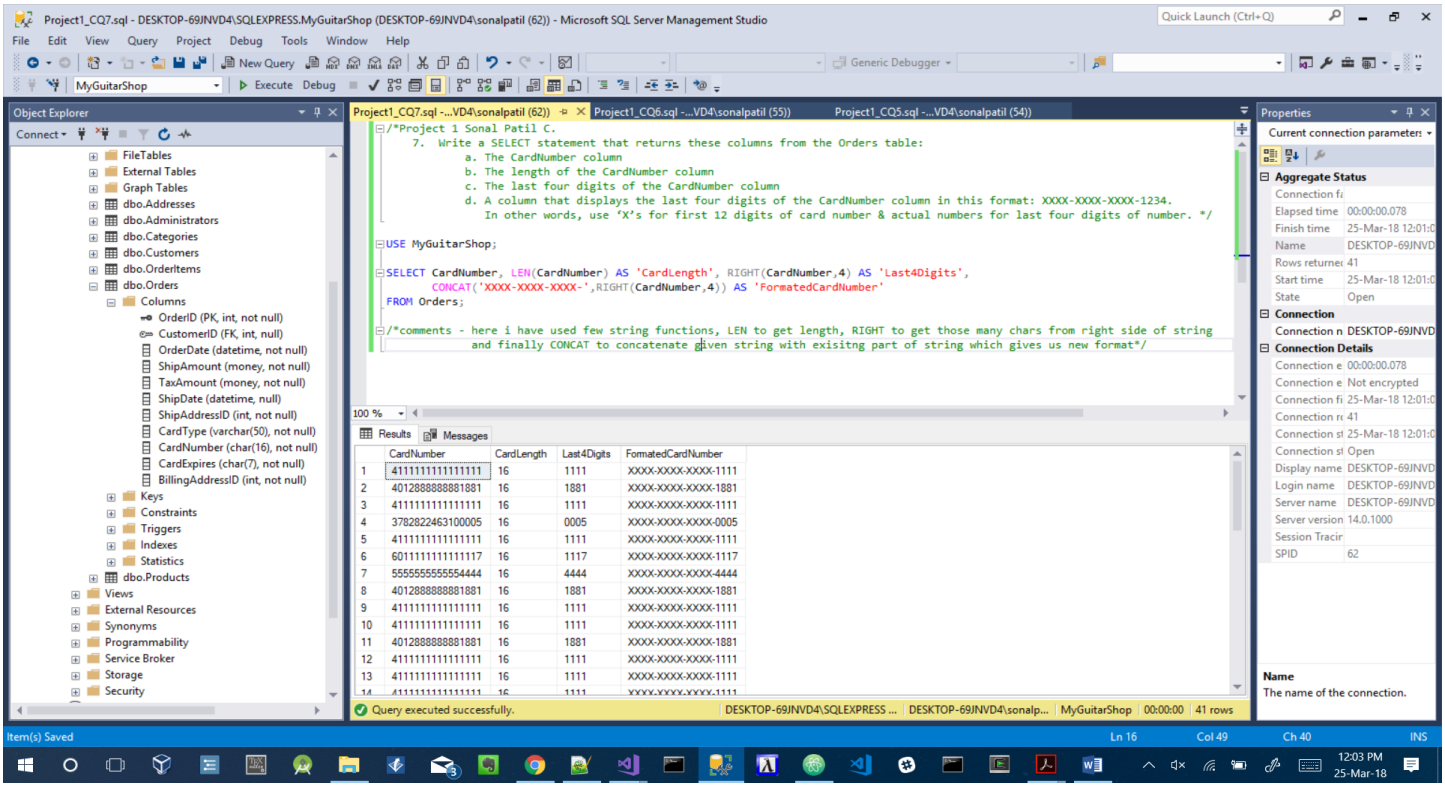
6. Write a SELECT statement that returns three columns: EmailAddress, OrderID, and the order total for each customer. To do this, you can group the result set by the EmailAddress and OrderID columns. In addition, you must calculate the order total from the columns in the OrderItems table.
   Write a second SELECT statement that uses the first SELECT statement in its FROM clause. The main query should return two columns: the customer's email address and the largest order for that customer. To do this, you can group the result set by the EmailAddress column.





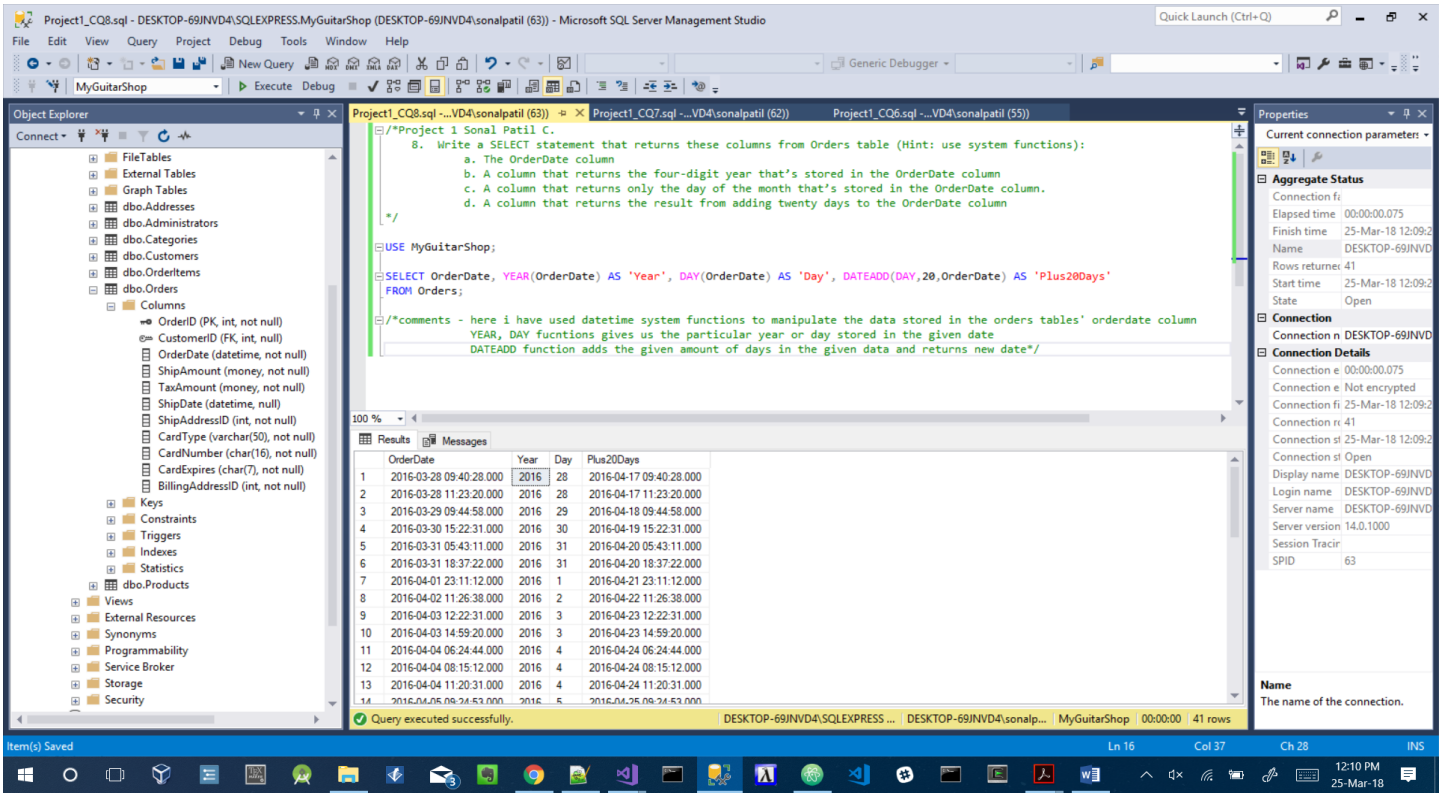
7. Write a SELECT statement that returns these columns from the Orders table:

   a. CardNumber column b. length of CardNumber column c. last 4 digits of CardNumber column
   d. Column that displays last 4 digits of CardNumber column in format: XXXX-XXXX-XXXX-1234. In other words, use 'X's for 1st 12 digits of CardNumber & actual numbers for last 4 digits of number.



8. Write a SELECT statement that returns these columns from Orders table (Hint: use system functions):

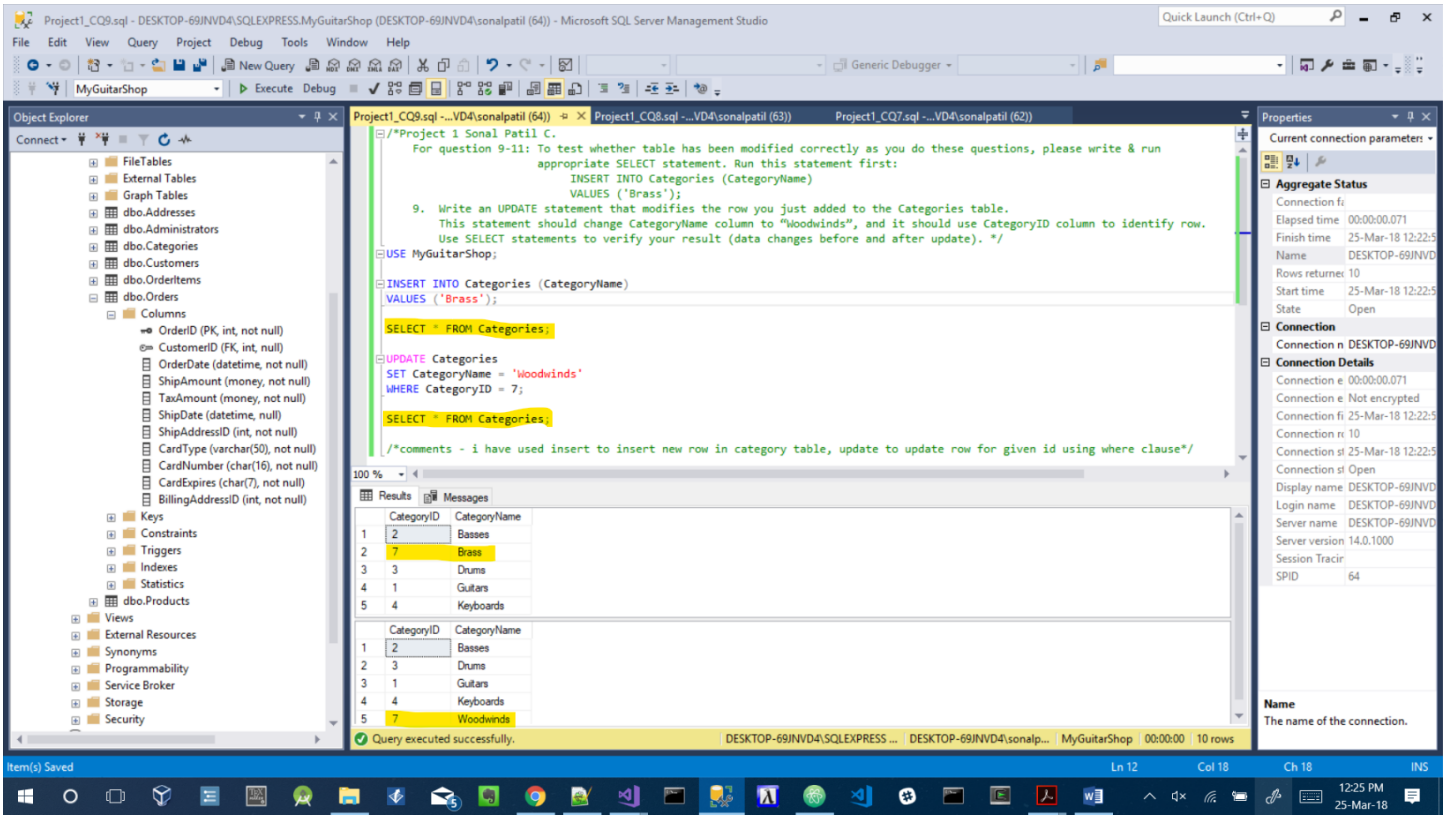
   a. OrderDate column b. A column that returns four-digit year that's stored in OrderDate column
   c. A column that returns only the day of the month that's stored in the OrderDate column.
   d. A column that returns the result from adding twenty days to the OrderDate column



For question 9-11: To test whether a table has been modified correctly as you do these questions, please write and run an appropriate SELECT statement.
Run this statement first:

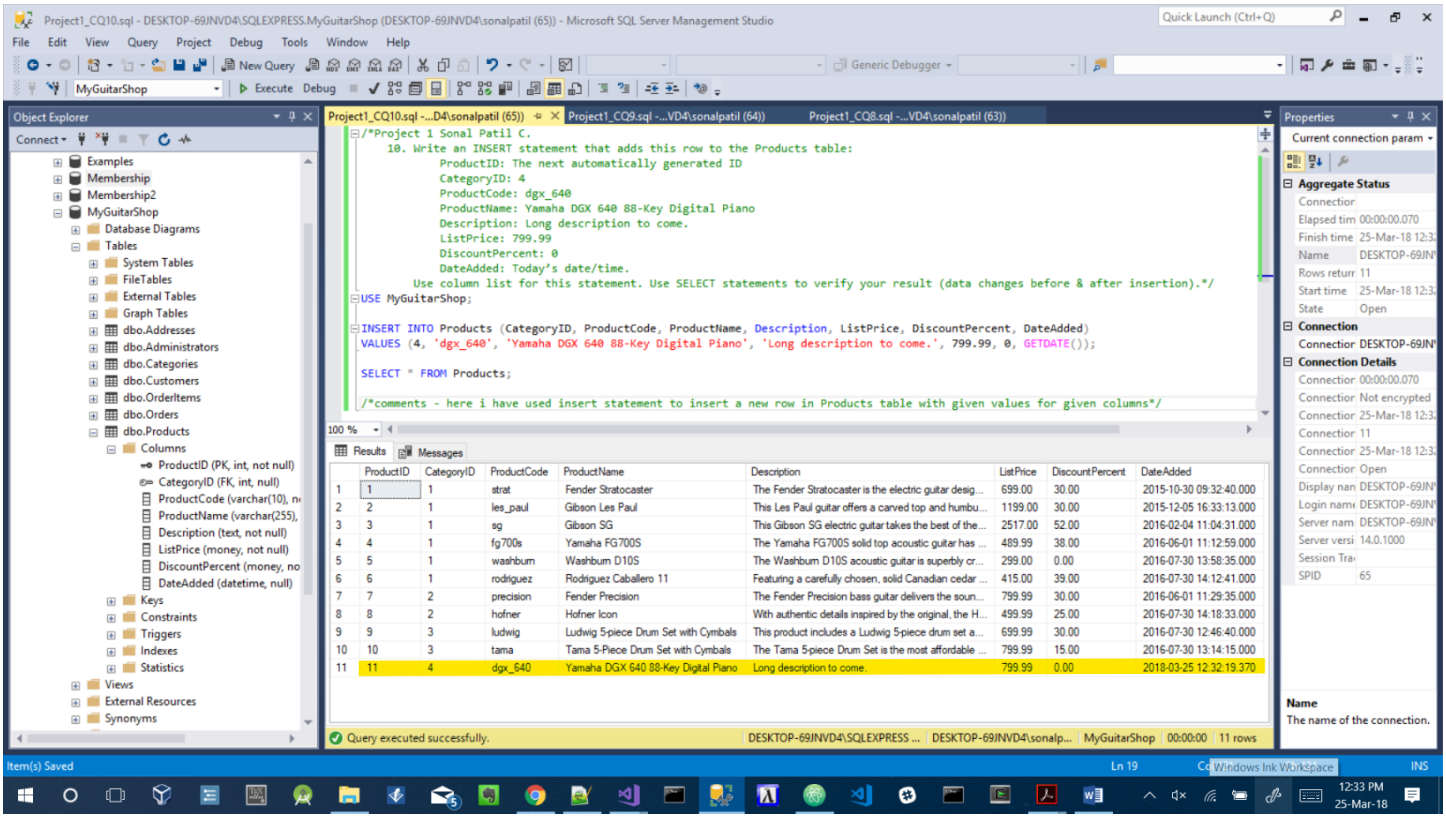
```
INSERT INTO Categories (CategoryName)
VALUES ('Brass');
```

9. Write an UPDATE statement that modifies the row you just added to the Categories table. This statement should change the CategoryName column to “Woodwinds”, and it should use the CategoryID column to identify the row. Use SELECT statements to verify your result (data changes before and after update).

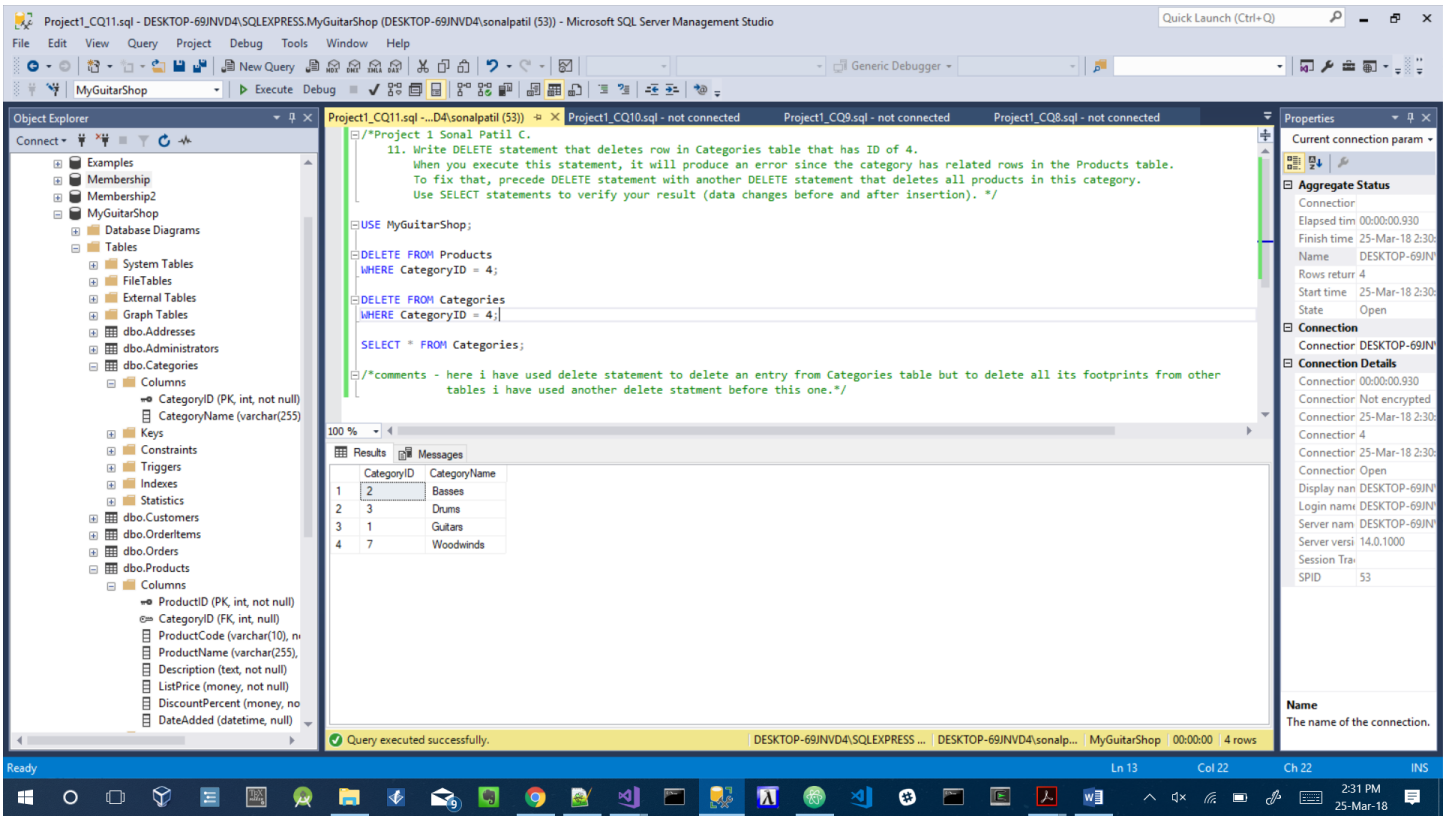


10. Write an INSERT statement that adds this row to the Products table:
    ProductID: The next automatically generated ID
    CategoryID: 4
    ProductCode: dgx_640
    ProductName: Yamaha DGX 640 88-Key Digital Piano
    Description: Long description to come.
    ListPrice: 799.99
    DiscountPercent: 0
    DateAdded: Today’s date/time.

    Use a column list for this statement. Use SELECT statements to verify your result (data changes before and after insertion).



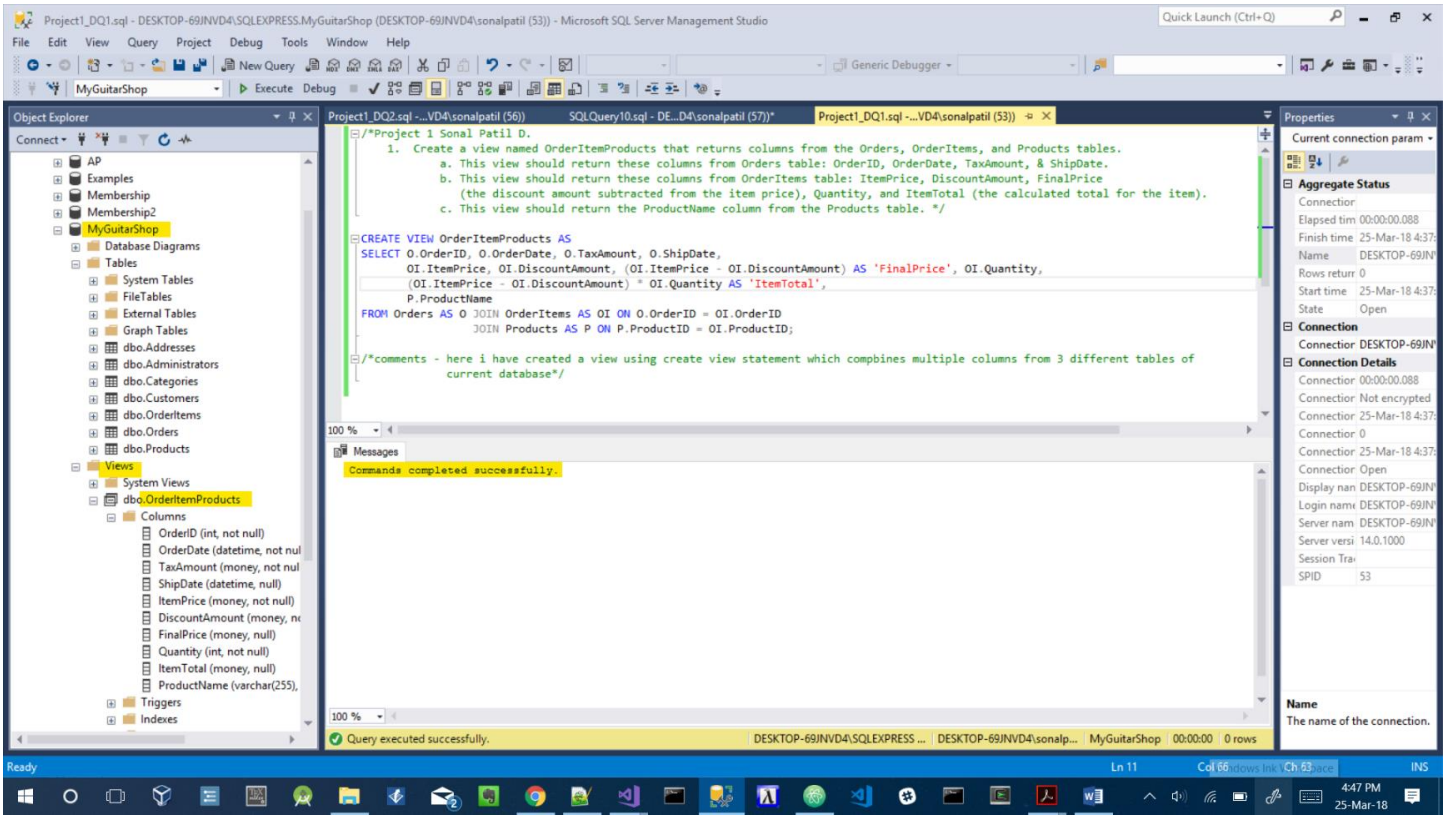
11. Write a DELETE statement that deletes the row in the Categories table that has an ID of 4. When you execute this statement, it will produce an error since the category has related rows in Products table. To fix that, precede DELETE statement with another DELETE statement that deletes all products in this category. Use SELECT statements to verify your result (data changes before & after insertion).

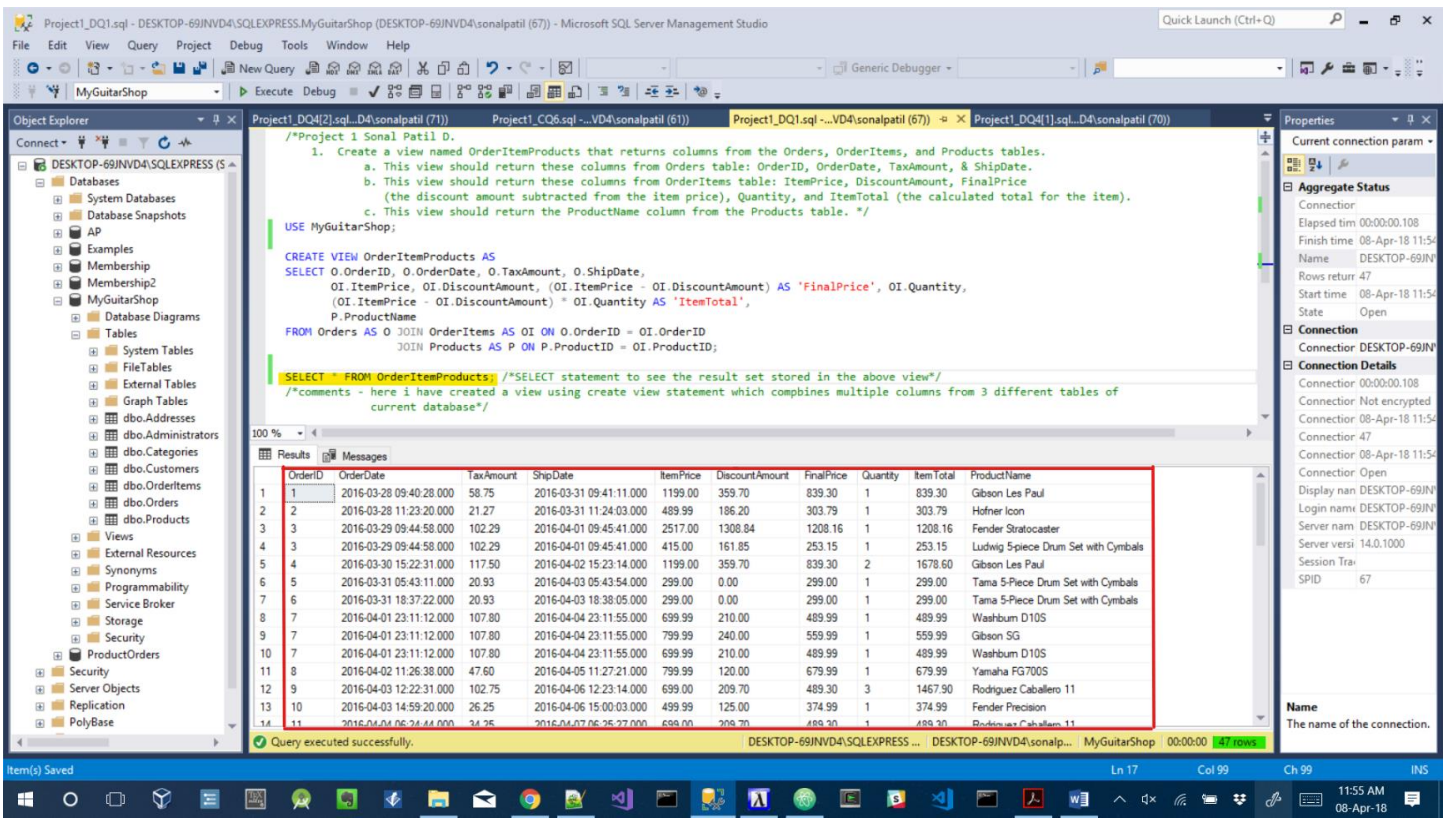


D. Advanced SQL skills (views/stored procedures/functions/scripts) [24 pts.]

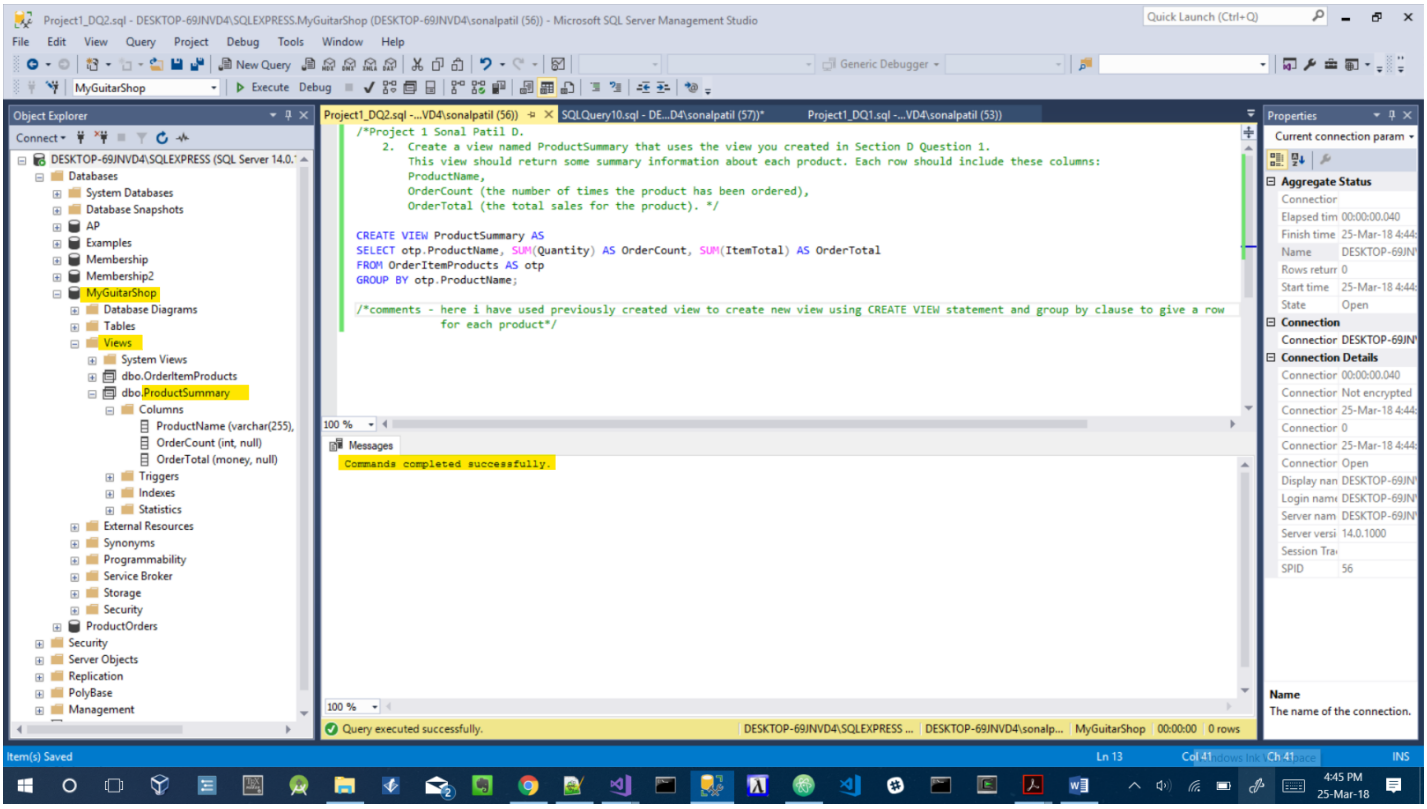
Open script CreateMyGuitarShop.sql & run it. That should restore data. Complete questions in SectionD.

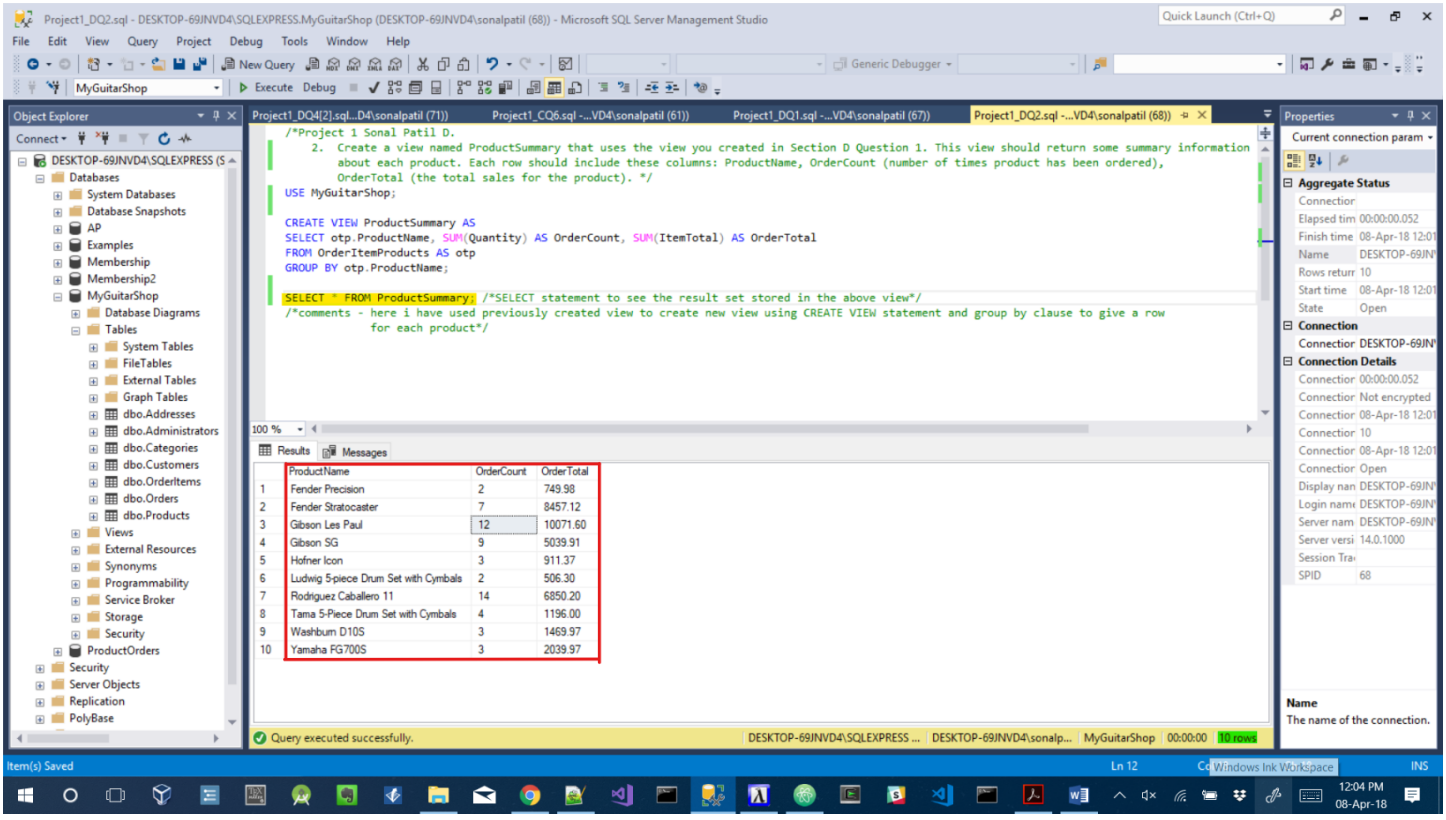
1. Create view OrderItemProducts that returns columns from Orders, OrderItems, & Products tables. a. columns from Orders table: OrderID, OrderDate, TaxAmount, & ShipDate. b. columns from OrderItems table: ItemPrice, DiscountAmount, FinalPrice (discount amount subtracted from item price), Quantity & ItemTotal (calculated total for item). c. column from Products table: ProductName.





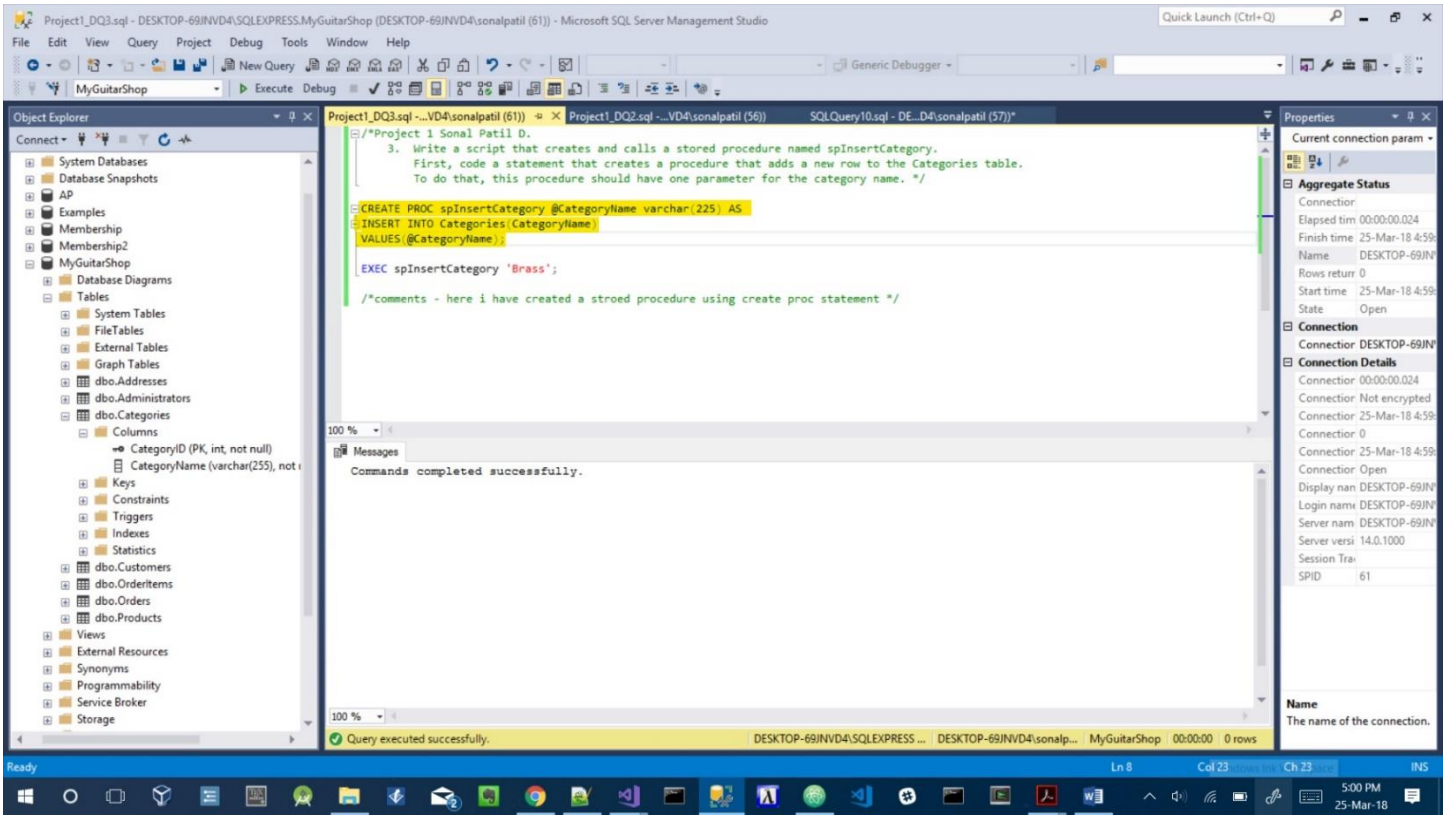
2. Create a view named ProductSummary that uses the view you created in Section D Question 1. This view should return some summary information about each product. Each row should include these columns: ProductName, OrderCount (the number of times the product has been ordered), and OrderTotal (the total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



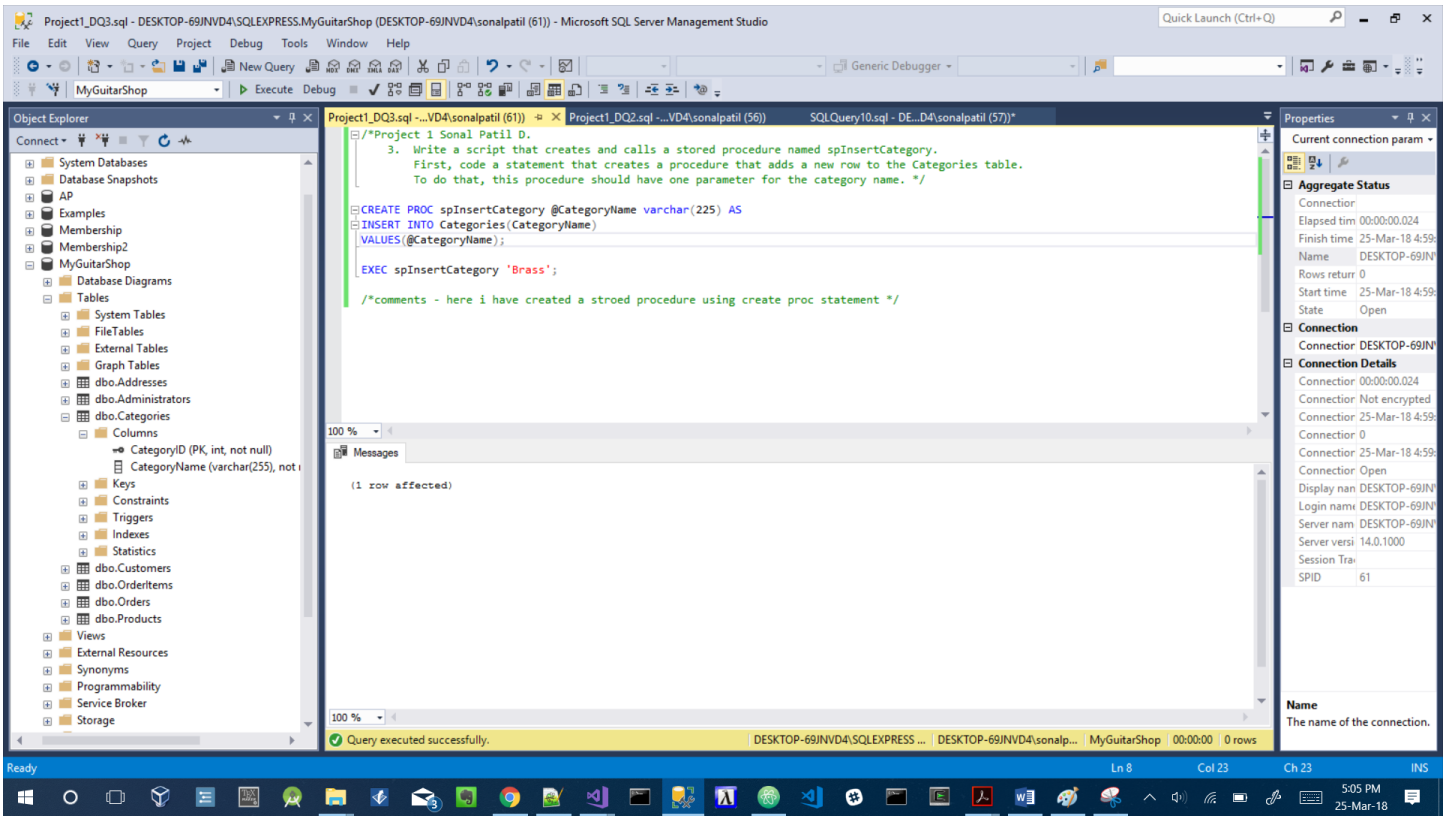


3. Write a script that creates and calls a stored procedure named spInsertCategory. First, code a statement that creates a procedure that adds a new row to the Categories table. To do that, this procedure should have one parameter for the category name. Code at least two EXEC statements that test this procedure. (Note that this table doesn't allow duplicate category names.)

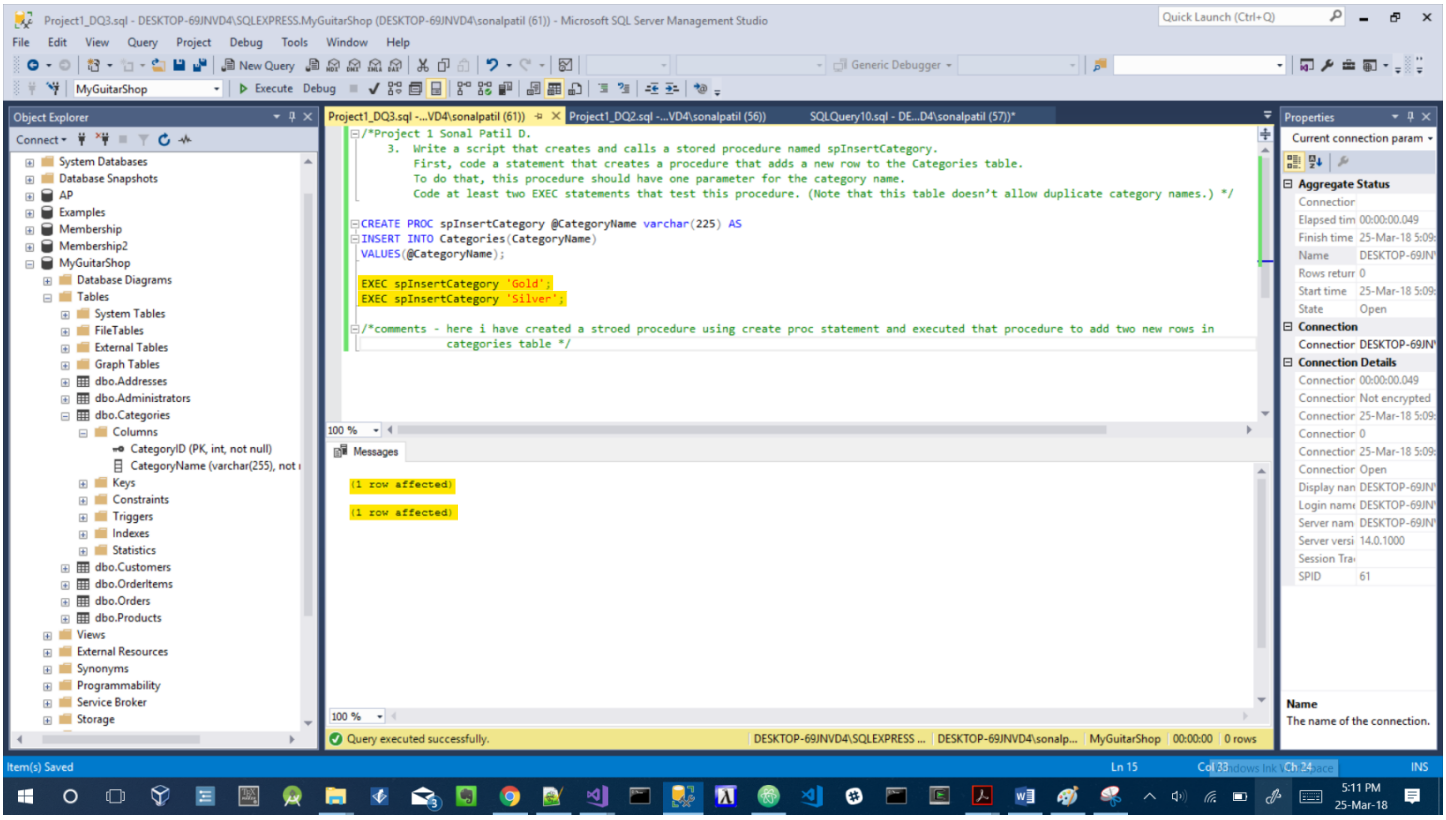
Created stored procedure –



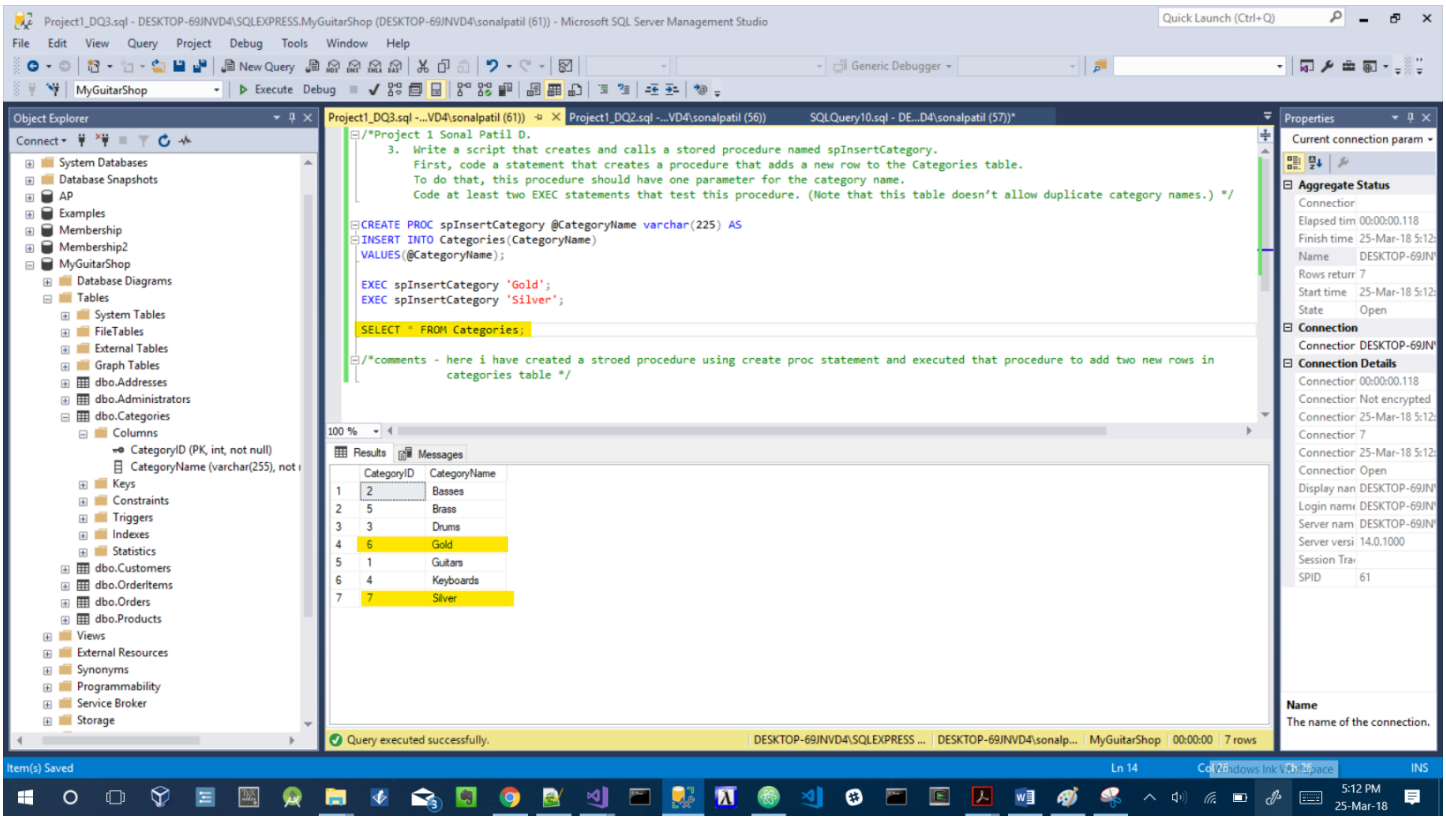
Executed created store procedure statement -



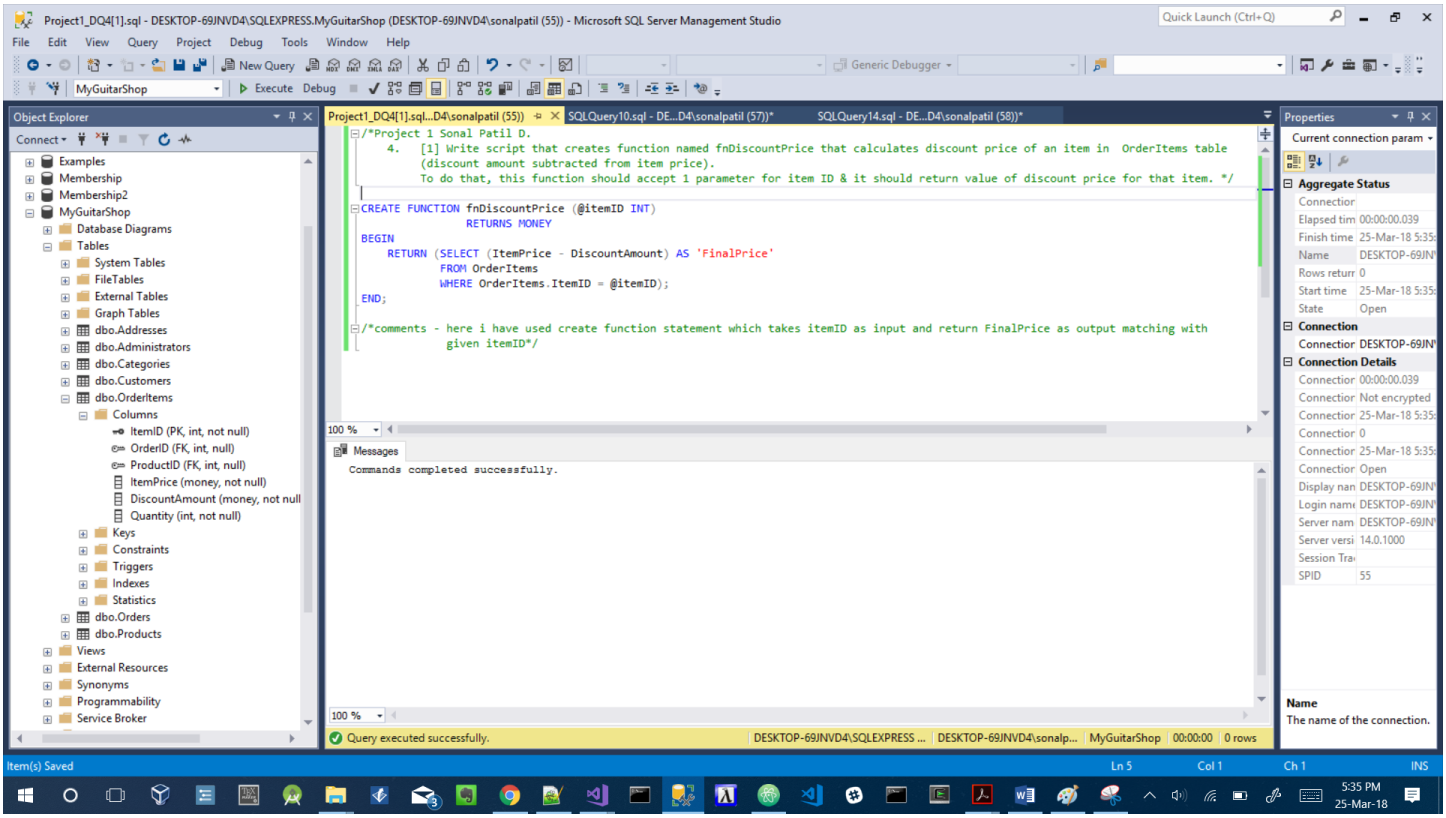
Called Stored Procedure –



Showed result using select statement –

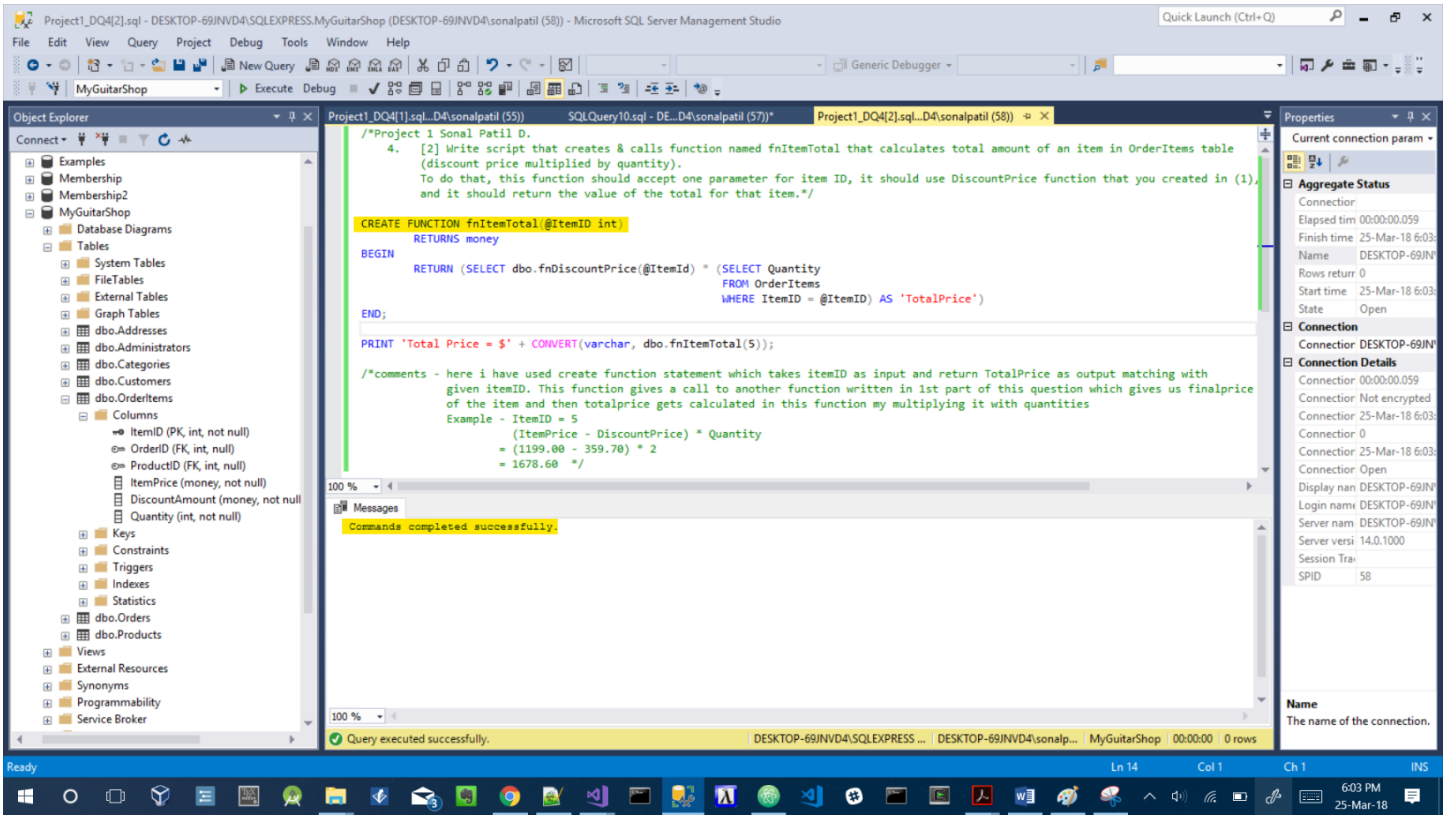


4. [1] Write a script that creates a function named fnDiscountPrice that calculates the discount price of an item in the OrderItems table (discount amount subtracted from item price). To do that, this function should accept one parameter for the item ID, and it should return the value of the discount price for that item.

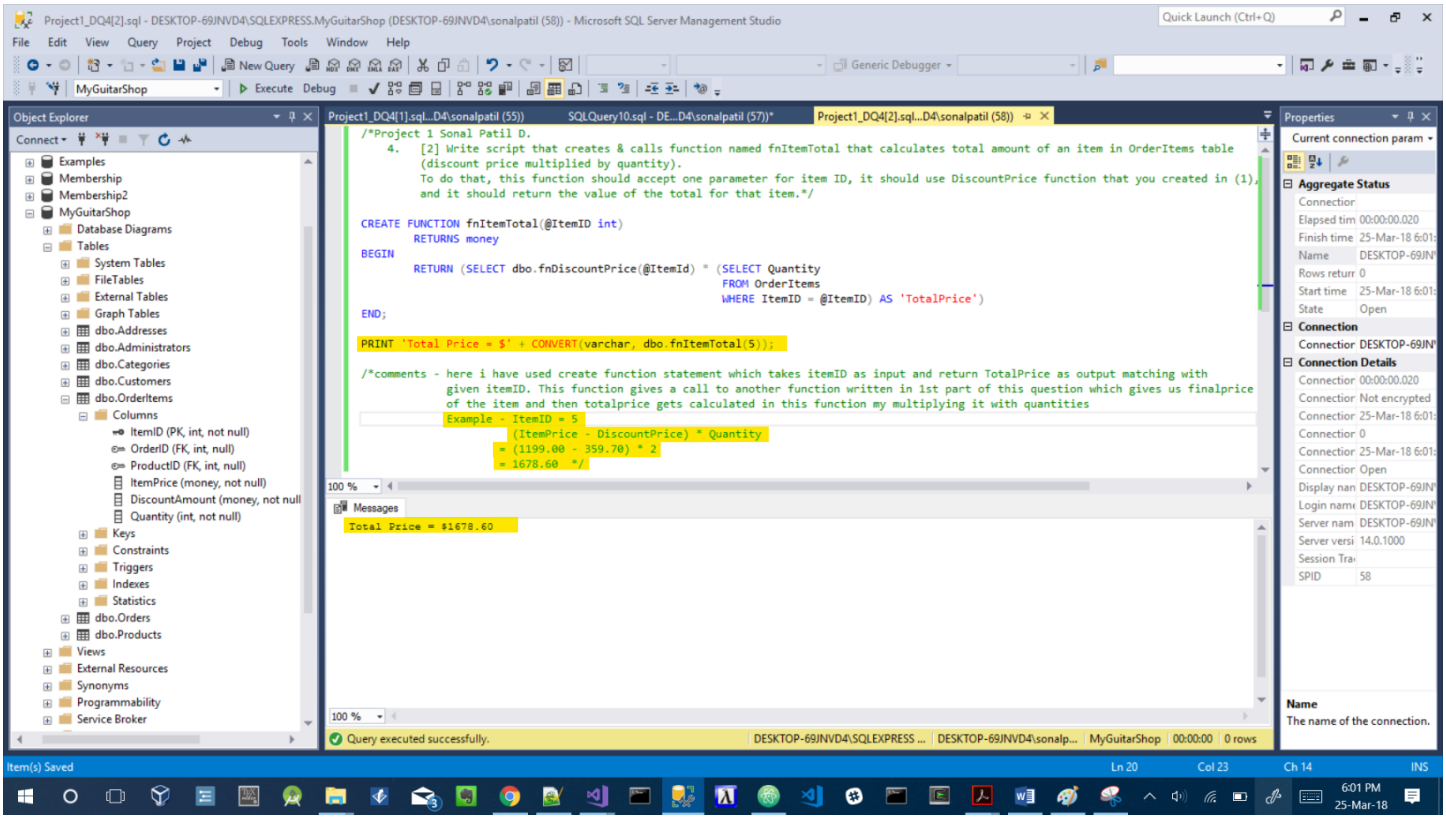


[2] Write a script that creates and calls a function named fnItemTotal that calculates the total amount of an item in the OrderItems table (discount price multiplied by quantity). To do that, this function should accept one parameter for the item ID, it should use the DiscountPrice function that you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

Created the function –



Called the function –



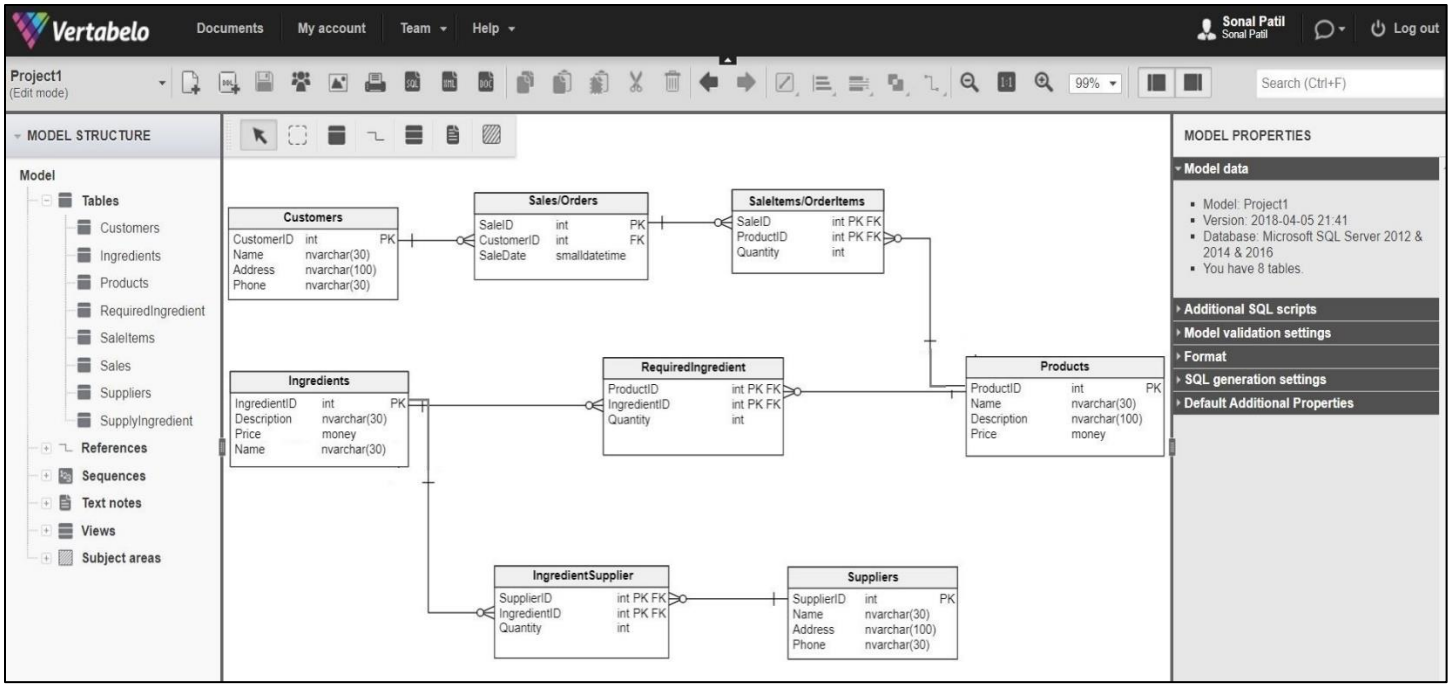
II. Database Design [20 pts.]

Create sample database design for a small bakery and draw one database model for it.
The bakery is interested in keeping track of information about its customers, suppliers, products, ingredients and sales. Bakery sells their products to customers & use different ingredients to create their products. Suppliers provide ingredients to bakery. Customers buy different products.

1. Design database that makes sense for business problem & select fields that make the most sense. A full screenshot of your final design is required.
2. Determine the tables, columns, primary keys, datatypes, nullabilities, and show relationships between tables(one-one/one-many/many-many).
3. Explain your design, including relationship between tables.

Here, I have designed total 8 tables out of which 5 are demanded by the requirement of creating a Bakery database and 3 are used to have the relationship between them.
Database design explanation & Full design screenshot –

Customers:
Customers will place one or multiple orders which will contain products that Bakery produces. Uniquely identifying key i.e. Primary key is CustomerID. Each Customer will have only one unique ID. Non-Nullable Values are CustomerID which is a primary key and it cannot be null also Name column must have value. Nullable values are Address and Phone [Optional data].



Sales[/Orders]:
This table will get the Order/SaleID and maps it with the customerID.
Uniquely identifying key i.e. Primary key is SaleID. Each Sale/Order will have only one unique ID.
Foreign key is CustomerID. This maps with the unique record from Customers to Sales. This key indicates one:many relationship between Customers and Sales/Orders table. One Customer can have one or more order entries in Orders table.
Non-Nullable Values are SaleID, CustomerID. Right now, in the Sales table, only SaleDate is non-key column, so I chose not to keep it available for Null values.

SaleItems[/OrderItems]:
This table gets the Order from Order table and maps it with the ProductIDs that means when customer orders multiple products, the Order/SaleID will have multiple ProductIDs rows.
Composite key i.e. Multiple Primary keys are SaleID & ProductID.
Foreign keys are SaleID & ProductID. This key indicates one:many relationship between Sales/Orders & SaleItems table and between Product & SaleItems.
Non-Nullable Values are SaleID, ProductID. Right now, in the SaleItems table, only Quantity is non-key column, so I chose not to keep it available for Null values.

Products:
This table keeps track of Product made by Bakery and Ingredients used for making that Product.
Uniquely identifying key i.e. Primary key is ProductID. Each Product will have only one unique ID.
Non-Nullable Values are ProdcutID which is a primary key and it cannot be null also Name & Price column must have values. Nullable value can be placed for Description column [Optional data].

RequiredIngredients:
This table maps the Ingredients used for creating single product. That means it contains the ProductID and equivalent IngredientIDs used to make that Product.
Composite key i.e. Multiple Primary keys are ProductID & IngredientID.

Foreign keys are ProductID & IngredientID. This key indicates one:many relationship between Products & RequiredIngredients and between RequiredIngredients & Ingredients table.
Non-Nullable Values are ProductID, IngredientID. Right now, in the RequiredIngredients table, only Quantity is non-key column, so I chose not to keep it available for Null values.

Ingredients:
This table keeps data stored related to each Ingredient.
Uniquely identifying key i.e. Primary key is IngredientID. Each Ingredient will have only 1 unique ID.
Non-Nullable Values are IngredientID which is a primary key and it cannot be null also Name & Price column must have values. Nullable value can be placed for Description column [Optional data].

IngredientSupplier:
This table maps the Ingredient with it's supplier.
Composite key i.e. Multiple Primary keys are IngredientID & SupplierID.
Foreign keys are IngredientID & SupplierID. This key indicates one:many relationship between IngredientSupplier & Suppliers and between IngredientSupplier & Ingredient table.
Non-Nullable Values are SupplierID, IngredientID. Right now, in the IngredientSupplier table, only Quantity is non-key column, so I chose not to keep it available for Null values.

Suppliers:
This table keeps data stored related to each supplier.
Uniquely identifying key i.e. Primary key is SupplierID. Each Supplier will have only one unique ID.
Non-Nullable Values are SupplierID which is a primary key and it cannot be null also Name column must have values. Nullable value can be placed for Address & Phone column [Optional data].

4. Normalize your design into its 3rd Normal Form. Please use MS Visio or any similar tool.
   The designed database is already in 3rd Normal form since every non-key column of all tables depends only on its own primary key.

Remarks –

This project helped me revise all the concepts I have learned so far in this course.

For first Section, I created a new database called MyGuiterShop using given CreateMyGuiterShop.sql script and then performed some basic & advance level functionality on it.

In the 1st section of this project, I learned & played with the features of SQL server management studio and executed some basic SQL commands to get to know the database structure more. Modified the database tables using INSERT, UPDATE, DELETE statements. Created new views, stored procedures.

In the 2nd section, I learned how to design a database from scratch when you have given the requirements or jobs one real world entity does. Here, how bakery works was given as a part of requirement and then I designed the database matching those requirements.

It also helped me to understand how to transform real world entities data into database tables, how to relate them with each other using relationships available to us. Also gave me knowledge of database design engine 'Vertabelo'.