

Project I

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.

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar reads "Project 1 CreateMyGuitarShop.sql - DESKTOP-PTHKBA0\MyGuitarShop (DESKTOP-PTHKBA0\lyq (52)) - Microsoft SQL Server Management Studio". The left pane is the Object Explorer, showing a tree structure of databases, security, server objects, replication, and other system components. The right pane is the "Messages" window, which displays the execution of a SQL script named "Project 1 Create...". The script creates the "MyGuitarShop" database and its schema. The output shows the following messages:

```
=====
* This script creates the database named my_guitar_shop
=====

USE master;
GO

IF DB_ID('MyGuitarShop') IS NOT NULL
    DROP DATABASE MyGuitarShop;
GO

CREATE DATABASE MyGuitarShop;
GO

USE MyGuitarShop;

-- create the tables for the database
CREATE TABLE Categories (
    CategoryID int,
    CategoryName nvarchar(120),
    Description nvarchar(250),
    DisplayOrder int
);

-- create the categories
INSERT INTO Categories (CategoryID, CategoryName, Description, DisplayOrder)
VALUES (1, 'Guitars', 'All guitars and basses for sale.', 1);
INSERT INTO Categories (CategoryID, CategoryName, Description, DisplayOrder)
VALUES (2, 'Accessories', 'Guitar accessories like cases and cables.', 2);
INSERT INTO Categories (CategoryID, CategoryName, Description, DisplayOrder)
VALUES (3, 'Books', 'Guitar books and instructional materials.', 3);
INSERT INTO Categories (CategoryID, CategoryName, Description, DisplayOrder)
VALUES (4, 'Software', 'Guitar software for learning and practice.', 4);
INSERT INTO Categories (CategoryID, CategoryName, Description, DisplayOrder)
VALUES (5, 'Parts', 'Guitar parts and supplies for repair and customization.', 5);

Completion time: 2020-11-06T19:22:06.4606296+08:00
```

The status bar at the bottom of the SSMS window indicates "Query executed successfully." and shows the session details: DESKTOP-PTHKBA0 (15.0 RTM) | DESKTOP-PTHKBA0\lyq (52) | MyGuitarShop | 00:00:00 | 0 rows.

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

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure for 'DESKTOP-PTHKBA0' (SQL Server 15.0.2000.5), including databases, tables, and other objects. The central pane contains a query window with the following SQL code:

```
SELECT *
FROM OrderItems;
```

The results pane below shows the output of the query, which is a table with the following data:

ItemID	OrderID	ProductID	ItemPrice	DiscountAmount	Quantity	
1	1	2	1199.00	359.70	1	
2	2	8	489.99	186.20	1	
3	3	1	2517.00	1308.84	1	
4	4	3	415.00	161.85	1	
5	5	4	1199.00	359.70	2	
6	6	5	299.00	0.00	1	
7	7	6	299.00	0.00	1	
8	8	7	699.99	210.00	1	
9	9	7	799.99	240.00	1	
10	10	7	699.99	210.00	1	
11	11	8	799.99	120.00	1	
12	12	9	699.00	209.70	3	
13	13	10	7	499.99	125.00	1
14	14	11	699.00	209.70	1	

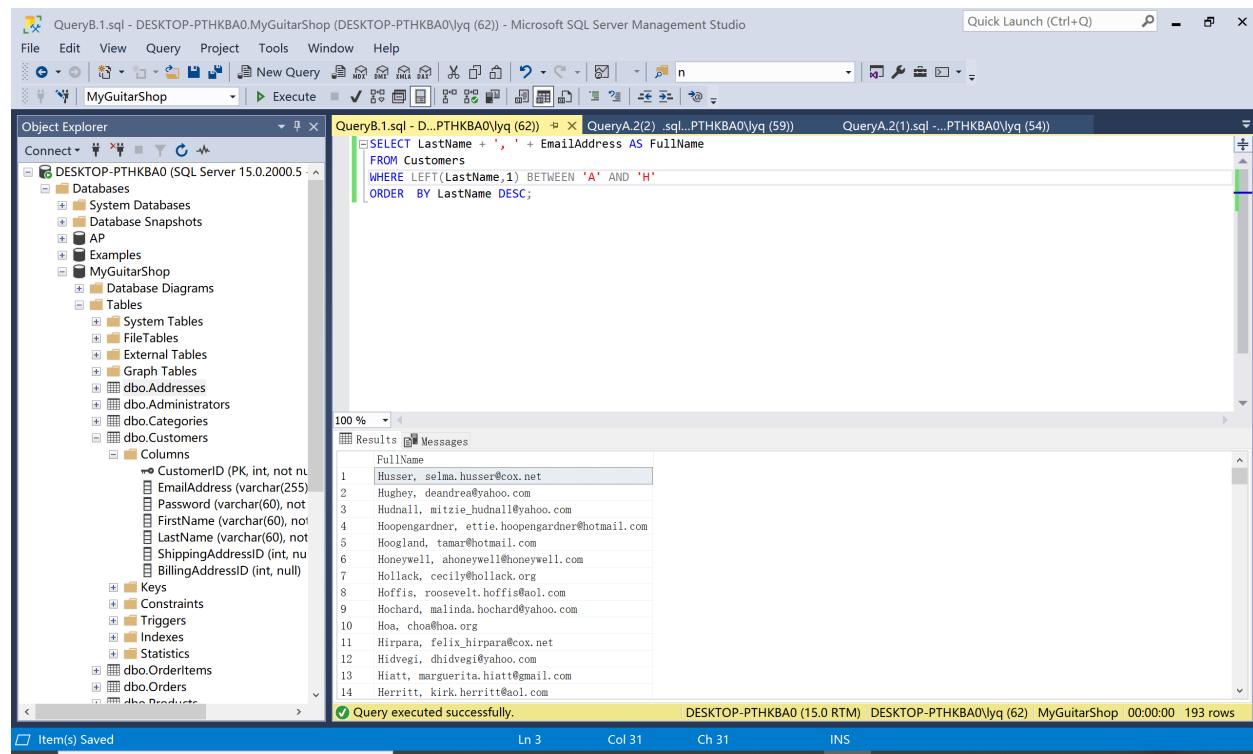
At the bottom of the results pane, a message indicates: "Query executed successfully." Below the results pane, the status bar shows: DESKTOP-PTHKBA0 (15.0 RTM) DESKTOP-PTHKBA0\lyq (54) MyGuitarShop 00:00:00 47 rows.

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 EmailAddress columns. Format this column with the last name, a comma, a space, and then Email Address like this:

Abdallah, johnetta_abdallah@aol.com

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



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'DESKTOP-PTHKBA0'. The central pane displays a query results grid titled 'Results' with 193 rows. The grid has a single column labeled 'FullName' containing names formatted as 'Last Name, Email Address'. The names listed are: Husser, selma.husser@cox.net; Hughey, deandree@yahoo.com; Hudnall, mitzie_hudnall@yahoo.com; Hoopengardner, ettie.hoopengardner@hotmail.com; Hoogland, tamara@hotmail.com; Honeywell, ahoneywell@honeywell.com; Hollack, cecily@hollack.org; Hoffmire, roosevelt.hoffmire@aol.com; Hochard, malinda.hochard@yahoo.com; Hoa, chao@hoa.org; Hirpara, felix_hirpara@cox.net; Hideggi, dhideggi@yahoo.com; Hiatt, marguerita.hiatt@gmail.com; Herritt, kirk.herritt@aol.com.

FullName
Husser, selma.husser@cox.net
Hughey, deandree@yahoo.com
Hudnall, mitzie_hudnall@yahoo.com
Hoopengardner, ettie.hoopengardner@hotmail.com
Hoogland, tamara@hotmail.com
Honeywell, ahoneywell@honeywell.com
Hollack, cecily@hollack.org
Hoffmire, roosevelt.hoffmire@aol.com
Hochard, malinda.hochard@yahoo.com
Hoa, chao@hoa.org
Hirpara, felix_hirpara@cox.net
Hideggi, dhideggi@yahoo.com
Hiatt, marguerita.hiatt@gmail.com
Herritt, kirk.herritt@aol.com

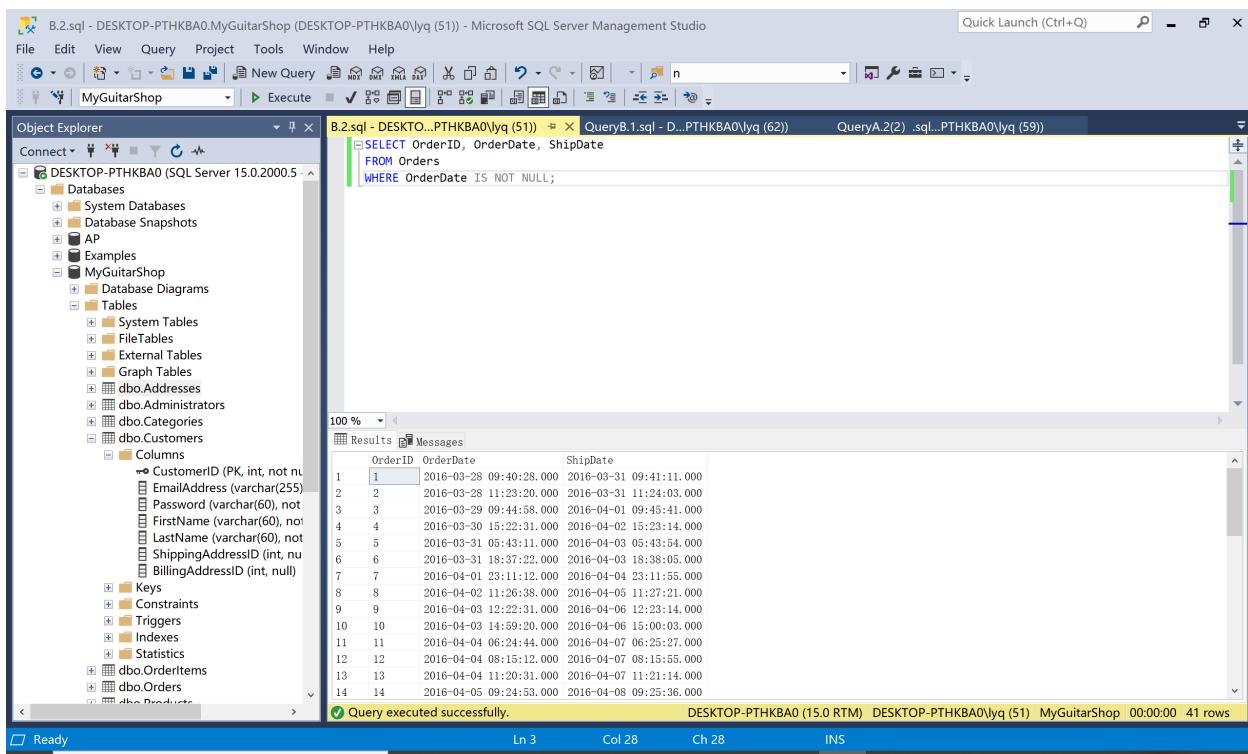
2. [3] Write a SELECT statement that returns these columns from the Orders table:

My Guitar Shop Database

OrderID OrderDate ShipDate

The OrderID column The OrderDate column The ShipDate column

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



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the MyGuitarShop database and its tables. The central pane displays a query window with the following SQL code:

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

The Results pane below shows the execution of the query, displaying 41 rows of data from the Orders table where OrderDate is not null. The columns are OrderID, OrderDate, and ShipDate. The data includes various dates ranging from March 28, 2016, to April 5, 2016.

OrderID	OrderDate	ShipDate
1	2016-03-28 09:40:28.000	2016-03-31 09:41:11.000
2	2016-03-28 11:23:20.000	2016-03-31 11:24:03.000
3	2016-03-29 09:44:58.000	2016-04-01 09:45:41.000
4	2016-03-30 15:22:31.000	2016-04-02 15:23:14.000
5	2016-03-31 05:43:11.000	2016-04-03 05:43:54.000
6	2016-03-31 18:37:22.000	2016-04-03 18:38:05.000
7	2016-04-01 23:11:12.000	2016-04-04 23:11:55.000
8	2016-04-02 11:26:38.000	2016-04-05 11:27:21.000
9	2016-04-03 12:22:31.000	2016-04-06 12:23:14.000
10	2016-04-03 14:59:20.000	2016-04-06 15:00:03.000
11	2016-04-04 06:24:44.000	2016-04-07 06:25:27.000
12	2016-04-04 08:15:12.000	2016-04-07 08:19:55.000
13	2016-04-04 11:20:31.000	2016-04-07 11:21:14.000
14	2016-04-05 09:24:53.000	2016-04-08 09:25:36.000

At the bottom of the results grid, a message indicates "Query executed successfully." The status bar at the bottom of the screen shows "DESKTOP-PTHKBA0 (15.0 RTM) DESKTOP-PTHKBA0\lyq (51) MyGuitarShop 00:00:00 41 rows".

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: LastName, ShipDate, ProductName, Description, ItemPrice, DiscountAmount, and Quantity. Use aliases for the tables. Sort the final result set by LastName in ascending order, and in descending order for ShipDate, and ProductName.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'MyGuitarShop'. A query window titled 'C.1.sql - DESKTOP-PTHKBA0\lyq (59)' contains the following SQL code:

```
SELECT LastName, ShipDate, ProductName, Description, ItemPrice, DiscountAmount, Quantity
FROM Customers JOIN Orders ON Customers.CustomerID = Orders.CustomerID
JOIN OrderItems ON Orders.OrderID = OrderItems.OrderID
JOIN Products ON Products.ProductID = OrderItems.ProductID
ORDER BY LastName ASC, ShipDate DESC, ProductName DESC;
```

The results pane displays a table with 13 rows of data from the query. The columns are: ShipDate, ProductName, Description, ItemPrice, DiscountAmount, and Quantity. The data includes various guitar models like Gibson SG, Les Paul, and Fender Stratocaster, along with their descriptions, prices, discounts, and quantities.

ShipDate	ProductName	Description	ItemPrice	DiscountAmount	Quantity
2016-04-23 08:15:28.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian ce...	699.00	209.70	1
2016-04-14 08:22:15.000	Gibson SG	This Gibson SG electric guitar takes the best o...	799.99	240.00	1
2016-04-02 15:23:14.000	Gibson Les Paul	This Les Paul guitar offers a carved top and hu...	1199.00	359.70	2
2016-04-07 06:25:27.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian ce...	699.00	209.70	1
2016-04-20 17:41:05.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian ce...	699.00	209.70	1
NULL	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian ce...	699.00	209.70	1
NULL	Hofner Icon	With authentic details inspired by the original...	489.99	186.20	1
2016-04-07 08:15:55.000	Fender Stratocaster	The Fender Stratocaster is the electric guitar ...	2517.00	1308.84	1
2016-04-09 18:42:36.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian ce...	699.00	209.70	1
2016-04-15 12:27:35.000	Fender Stratocaster	The Fender Stratocaster is the electric guitar ...	2517.00	1308.84	1
2016-04-06 15:00:03.000	Fender Precision	The Fender Precision bass guitar delivers the s...	499.99	125.00	1
2016-05-04 09:12:34.000	Tama 5-Piece Drum Set with Cymbals	The Tama 5-piece Drum Set is the most affordabl...	299.00	0.00	2
2016-04-08 14:53:00.000	Gibson SG	This Gibson SG electric guitar takes the best o...	799.99	240.00	1

At the bottom, a message indicates 'Query executed successfully.'

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

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a connection to DESKTOP-PTHKBA0 (SQL Server 15.0.2000.5) with the MyGuitarShop database selected. The Query Editor window on the right contains the following SQL code:

```
SELECT CategoryName, ProductID
FROM Categories LEFT JOIN Products ON Categories.CategoryID = Products.CategoryID
WHERE ProductID IS NULL;
```

The Results pane below the query shows a single row of data:

	CategoryName	ProductID
1	Keyboards	NULL

At the bottom of the Results pane, a message indicates the query was executed successfully.

Query Editor details at the bottom:

- Ln 3
- Col 25
- Ch 25
- INS

System status at the bottom:

- DESKTOP-PTHKBA0 (15.0 RTM)
- DESKTOP-PTHKBA0\lyq (53)
- MyGuitarShop
- 00:00:00
- 1 rows

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

- a) The EmailAddress column from the Customers table**
- b) The sum of the ItemPrice in the OrderItems table multiplied by the quantity in the OrderItems table**
- c) The average of the DiscountAmount column in the OrderItems table multiplied by the quantity in the OrderItems table.**

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

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database schema for 'MyGuitarShop', including tables like Addresses, Administrators, Categories, Customers, OrderItems, Orders, and Products. The SQL Query window in the center contains a query that joins the Customers and Orders tables, then joins the OrderItems table, groups by EmailAddress, and orders by TotalPrice in descending order. The Results pane below shows the output of the query, which lists 14 rows of customer information with their total price and discount applied. The status bar at the bottom indicates the query was executed successfully and completed in 0:00:00 with 35 rows processed.

```
SELECT EmailAddress, SUM(ItemPrice*Quantity) AS TotalPrice, AVG(DiscountAmount*Quantity) AS Discount
FROM Customers JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
JOIN OrderItems ON Orders.OrderID = OrderItems.OrderID
GROUP BY EmailAddress
ORDER BY TotalPrice DESC;
```

	EmailAddress	TotalPrice	Discount
1	david.goldstein@hotmail.com	6395.95	609.70
2	allan.sherwood@yahoo.com	4131.00	610.13
3	kris@gmail.com	3316.99	714.42
4	yuki_whobrey@aol.com	3216.00	759.27
5	heatheresway@mac.com	3016.99	716.92
6	mroyster@royster.com	2517.00	1308.84
7	josephine_darakjy@darakjy.org	2517.00	1308.84
8	mattie@aol.com	2517.00	1308.84
9	gruta@cox.net	2398.00	719.40
10	lpaprock1@hotmail.com	2398.00	719.40
11	sage_wieser@cox.net	2398.00	719.40
12	christineb@solarone.com	2398.00	719.40
13	alisha@lusarski.com	2398.00	719.40
14	frankwilson@sbcglobal.net	2199.97	220.00

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

- a) The EmailAddress column from the Customers table**
- b) A count of the number of orders**
- c) The total amount for each order (Hint: First, subtract the discount amount from the price. Then, multiply by the quantity)**

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

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists database objects for the 'MyGuitarShop' database, including tables like 'Addresses', 'Administrators', 'Categories', 'Customers', 'OrderItems', 'Orders', and 'Products'. The 'Customers' table is expanded to show columns such as CustomerID, EmailAddress, Password, FirstName, LastName, and ShippingAddressID. The 'OrderItems' table is also expanded to show columns for ItemID, OrderID, ProductID, ItemPrice, DiscountAmount, and Quantity. The central pane displays a T-SQL query that joins the 'Customers' and 'Orders' tables, groups by EmailAddress, filters for orders with a total amount greater than 500, and sorts the results by EmailAddress in descending order. The results grid shows 14 rows of data, each containing an EmailAddress, the number of orders, and the total amount.

	EmailAddress	NumberOfOrders	TotalAmount
1	yuki_whoibrey@aol.com	2	1697.46
2	vinouye@aol.com	1	559.99
3	simona@morasca.com	2	742.45
4	sage_wiser@cox.net	1	1678.60
5	mroyster@royster.com	1	1208.16
6	minna_amigon@yahoo.com	1	559.99
7	mattie@aol.com	1	1208.16
8	lpaprocki@hotmail.com	1	1678.60
9	kris@gmail.com	2	1888.15
10	josephine_darakjy@darakjy.org	1	1208.16
11	heatheresway@mac.com	2	1583.15
12	gruta@cox.net	1	1678.60
13	gary_hernandez@yahoo.com	1	679.99
14	frankwilson@sbcglobal.net	3	1539.97

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.

(1)

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists database objects: dbo.Addresses, dbo.Administrators, dbo.Categories, dbo.Customers, dbo.Columns, dbo.OrderItems, dbo.Orders, and dbo.Tables. The dbo.OrderItems node is expanded, showing its columns: ItemID, OrderID, ProductID, ItemPrice, DiscountAmount, and Quantity. The Results tab in the center displays a query result grid. The query is:

```

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

```

The results grid shows 14 rows of data:

EmailAddress	OrderID	OrderTotalAmount
allan.sherwood@yahoo.com	1	839.30
barryz@gmail.com	2	303.79
allan.sherwood@yahoo.com	3	1461.31
christineb@solarone.com	4	1678.60
david.goldstein@hotmail.com	5	299.00
erinv@gmail.com	6	299.00
frankwilson@sbcglobal.net	7	1539.97
gary_hernandez@yahoo.com	8	679.99
david.goldstein@hotmail.com	9	1467.90
heatheresway@mac.com	10	374.99
jbutte@gmail.com	11	489.30
josephine_darakjy@darakjy.org	12	1208.16
art@enere.org	13	679.99
lpatrick@hotmail.com	14	1678.60

At the bottom of the results grid, a status bar indicates: Query executed successfully. DESKTOP-PTHKBA0 (15.0 RTM) DESKTOP-PTHKBA0\lyq (51) MyGuitarShop 00:00:00 41 rows.

5(2) Write a second SELECT statement that uses the first SELECT statement in its FROM clause. The main query should return two columns: the customer's email address and the largest order for that customer. To do this, you can group the result set by the EmailAddress column.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database schema for 'MyGuitarShop'. The central pane contains a query window with the following SQL code:

```

SELECT EmailAddress, MAX(OrderTotalAmount) AS LargestOrder
FROM (SELECT EmailAddress, Orders.OrderID, SUM(ItemPrice*Quantity-DiscountAmount*Quantity) AS OrderTotalAmount
      FROM Customers JOIN Orders ON Customers.CustomerID = Orders.CustomerID
      JOIN OrderItems ON OrderItems.OrderID = Orders.OrderID
      GROUP BY EmailAddress, Orders.OrderID) AS Customers
GROUP BY Customers.EmailAddress;
  
```

The results pane below shows a table with 14 rows, each mapping an email address to its largest order amount. The table has two columns: 'EmailAddress' and 'LargestOrder'.

EmailAddress	LargestOrder
alisha@lusarski.com	1678.60
allan.sherwood@yahoo.com	1461.31
allene_iturbide@cox.net	489.30
amaclead@gmail.com	489.30
art@venere.org	679.99
barryz@gmail.com	303.79
bette_nicka@cox.net	839.30
calibres@gmail.com	489.30
chanel.caudy@caudy.org	793.09
christineb@solarone.com	1678.60
david.goldstein@hotmail.com	2799.95
donette.foller@cox.net	559.99
erinv@gmail.com	299.00
fletcher.fosi@yahoo.com	598.00

At the bottom of the results pane, a message indicates: "Query executed successfully." The status bar at the bottom of the screen shows: DESKTOP-PTHKBA0 (15.0 RTM) DESKTOP-PTHKBA0\lyq (66) MyGuitarShop 00:00:00 35 rows.

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.

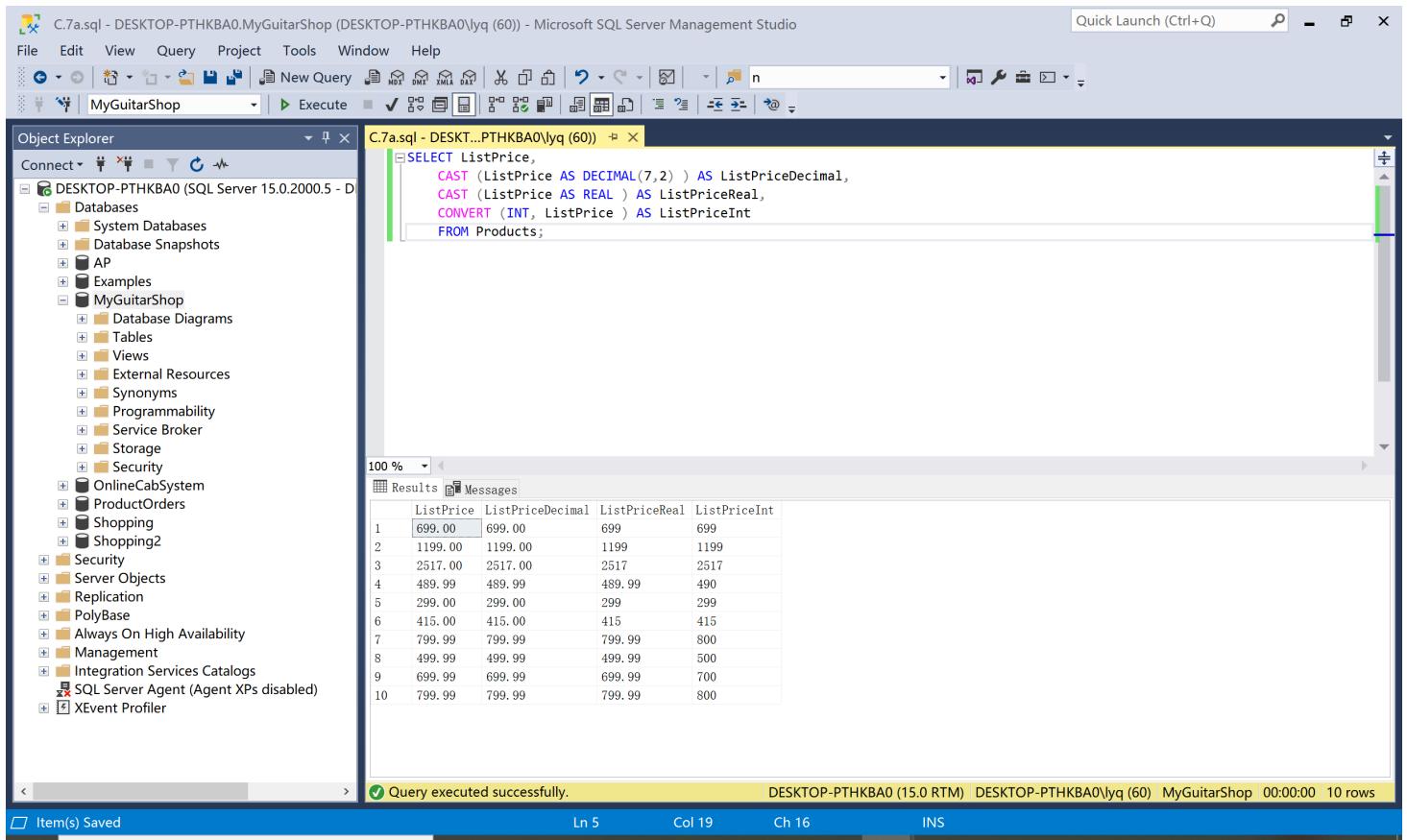
The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'MyGuitarShop'. The 'Tables' node under 'dbo' contains 'Orders'. The 'Columns' node under 'Orders' lists various columns including OrderID, CustomerID, OrderDate, ShipAmount, TaxAmount, ShipDate, ShipAddressID, CardType, CardNumber, CardExpires, and BillingAddressID. The 'Messages' pane at the bottom displays an error message: 'Msg 208, Level 16, State 1, Line 1 Invalid object name 'Orders''. The status bar at the bottom right indicates 'DESKTOP-PTHKBA0 (15.0 RTM) DESKTOP-PTHKBA0\lyq (51) master 00:00:00 0 rows'.

```
SELECT Customer.EmailAddress, OrderID, MAX(OrderDate)
FROM Customer , Orders
WHERE Customer.CustomerID = Orders.CustomerID;
```

Sorry, I think I got the wrong syntax and I have tried to figure out but it didn't work.

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

- a) The ListPrice column**
- b) A column that uses the CAST function to return the ListPrice column with 2 digits to the right of the decimal point**
- c) A column that uses the CAST function to return the ListPrice column as a real number**
- d) A column that uses the CONVERT function to return the ListPrice column as an integer.**



The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar reads "C.7a.sql - DESKTOP-PTHKBA0.MyGuitarShop (DESKTOP-PTHKBA0\lyq (60)) - Microsoft SQL Server Management Studio". The Object Explorer on the left shows the database structure, including the "MyGuitarShop" database. The central pane displays a query window titled "C.7a.sql - DESKTOP-PTHKBA0\lyq (60)". The query is:

```
SELECT ListPrice,
       CAST (ListPrice AS DECIMAL(7,2) ) AS ListPriceDecimal,
       CAST (ListPrice AS REAL ) AS ListPriceReal,
       CONVERT (INT, ListPrice ) AS ListPriceInt
  FROM Products;
```

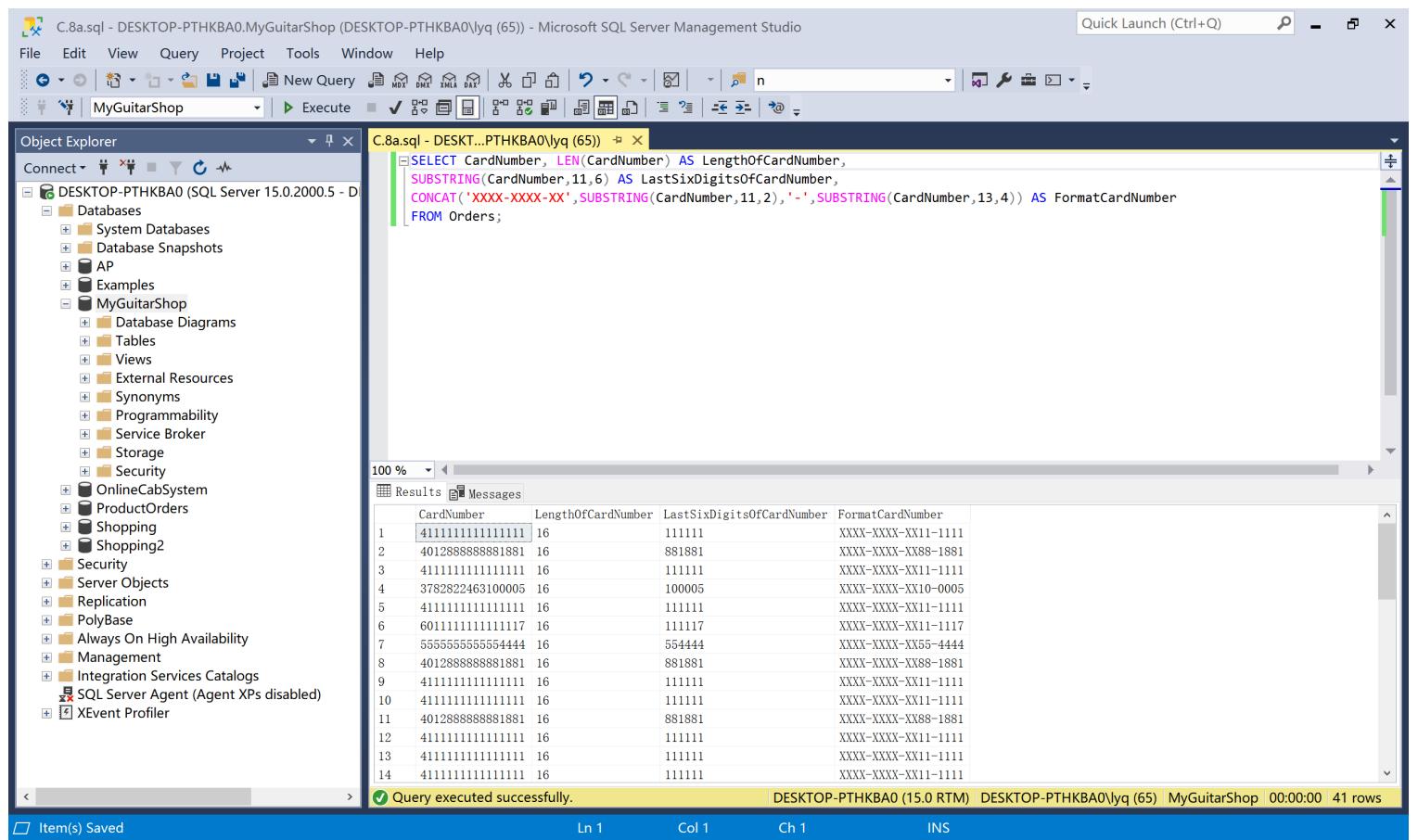
The results pane shows a table with four columns: ListPrice, ListPriceDecimal, ListPriceReal, and ListPriceInt. The data consists of 10 rows of product prices. The "ListPrice" column contains values like 699.00, 1199.00, etc. The "ListPriceDecimal" column shows the same values with two decimal places. The "ListPriceReal" column shows the same values rounded to one decimal place. The "ListPriceInt" column shows the values as integers (699, 1199, etc.).

	ListPrice	ListPriceDecimal	ListPriceReal	ListPriceInt
1	699.00	699.00	699	699
2	1199.00	1199.00	1199	1199
3	2517.00	2517.00	2517	2517
4	489.99	489.99	489.99	490
5	299.00	299.00	299	299
6	415.00	415.00	415	415
7	799.99	799.99	799.99	800
8	499.99	499.99	499.99	500
9	699.99	699.99	699.99	700
10	799.99	799.99	799.99	800

At the bottom of the results pane, a message says "Query executed successfully." The status bar at the bottom of the screen shows "DESKTOP-PTHKBA0 (15.0 RTM) DESKTOP-PTHKBA0\lyq (60) MyGuitarShop 00:00:00 10 rows".

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

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



The screenshot shows the Microsoft SQL Server Management Studio interface. The left pane is the Object Explorer, displaying the database structure for 'MyGuitarShop'. The right pane is the 'Results' tab, showing the output of a query against the 'Orders' table. The query retrieves four columns: CardNumber, LengthOfCardNumber, LastSixDigitsOfCardNumber, and FormatCardNumber. The results grid contains 14 rows of data, each showing a unique card number, its length (16), its last six digits, and its formatted version according to the specified pattern.

	CardNumber	LengthOfCardNumber	LastSixDigitsOfCardNumber	FormatCardNumber
1	4111111111111111	16	111111	XXXX-XXXX-XX11-1111
2	4012888888881881	16	881881	XXXX-XXXX-XX88-1881
3	4111111111111111	16	111111	XXXX-XXXX-XX11-1111
4	3782822463100005	16	100005	XXXX-XXXX-XX10-0005
5	4111111111111111	16	111111	XXXX-XXXX-XX11-1111
6	6011111111111117	16	111117	XXXX-XXXX-XX11-1117
7	55555555555444	16	554444	XXXX-XXXX-XX55-4444
8	4012888888881881	16	881881	XXXX-XXXX-XX88-1881
9	4111111111111111	16	111111	XXXX-XXXX-XX11-1111
10	4111111111111111	16	111111	XXXX-XXXX-XX11-1111
11	4012888888881881	16	881881	XXXX-XXXX-XX88-1881
12	4111111111111111	16	111111	XXXX-XXXX-XX11-1111
13	4111111111111111	16	111111	XXXX-XXXX-XX11-1111
14	4111111111111111	16	111111	XXXX-XXXX-XX11-1111

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

- a) The OrderID column**
- b) The OrderDate column**
- c) A column named ApproxShipDate that's calculated by adding 1 month to the OrderDate column**
- d) The ShipDate column**
- e) A column named DaysToShip that shows the number of days between the order date and the ship date**

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

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists databases, including 'MyGuitarShop'. The central pane contains a query editor with the following T-SQL code:

```
USE MyGuitarShop
SELECT OrderID, OrderDate, DATEADD(MONTH,1,OrderDate) AS ApproxShipDate,
       ShipDate,DATEDIFF(DAY,OrderDate,ShipDate) AS DayToShip
  FROM Orders
 WHERE DATEPART(MONTH,OrderDate)='4' AND DATEPART(YEAR,OrderDate)='2016';
```

The results grid below displays 25 rows of data with columns: OrderID, OrderDate, ApproxShipDate, ShipDate, and DayToShip. The data shows various order entries for April 2016, with the ApproxShipDate being one month after the OrderDate and the DayToShip value being 3 for all entries.

	OrderID	OrderDate	ApproxShipDate	ShipDate	DayToShip
1	7	2016-04-01 23:11:12.000	2016-05-01 23:11:12.000	2016-04-04 23:11:55.000	3
2	8	2016-04-02 11:26:38.000	2016-05-02 11:26:38.000	2016-04-05 11:27:21.000	3
3	9	2016-04-03 12:22:31.000	2016-05-03 12:22:31.000	2016-04-06 12:23:14.000	3
4	10	2016-04-03 14:59:20.000	2016-05-03 14:59:20.000	2016-04-06 15:00:03.000	3
5	11	2016-04-04 06:24:44.000	2016-05-04 06:24:44.000	2016-04-07 06:25:27.000	3
6	12	2016-04-04 08:15:12.000	2016-05-04 08:15:12.000	2016-04-07 08:15:55.000	3
7	13	2016-04-04 11:20:31.000	2016-05-04 11:20:31.000	2016-04-07 11:21:14.000	3
8	14	2016-04-05 09:24:53.000	2016-05-05 09:24:53.000	2016-04-08 09:25:36.000	3
9	15	2016-04-05 14:52:17.000	2016-05-05 14:52:17.000	2016-04-08 14:53:00.000	3
10	16	2016-04-06 07:53:42.000	2016-05-06 07:53:42.000	2016-04-09 07:54:25.000	3
11	17	2016-04-06 17:24:28.000	2016-05-06 17:24:28.000	2016-04-09 17:25:11.000	3
12	18	2016-04-06 18:41:53.000	2016-05-06 18:41:53.000	2016-04-09 18:42:36.000	3
13	19	2016-04-08 12:21:31.000	2016-05-08 12:21:31.000	2016-04-11 12:22:14.000	3
14	20	2016-04-10 09:33:23.000	2016-05-10 09:33:23.000	2016-04-13 09:34:06.000	3

At the bottom, a message indicates "Query executed successfully." and the status bar shows "Ready".

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 your SQL server window.

10. [4] Write an INSERT statement that adds this row to the Customers table:

EmailAddress: kriegerrobert@gmail.com

Password: FirstName: LastName:

(empty string) Krieger Robert

Use a column list for this statement..

Here are my test and we could found Krieger was inserted.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the MyGuitarShop database and its tables like Customers. The central pane displays a query window with the following SQL code:

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

SELECT *
FROM Customers
WHERE Password = '';
```

The results grid below shows one row inserted into the Customers table:

	CustomerID	EmailAddress	Password	FirstName	LastName	ShippingAddressID	BillingAddressID
1	486	kriegerrobert@gmail.com		Krieger	Robert	NULL	NULL

At the bottom, a message bar indicates "Query executed successfully."

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.

After the test, we could see as below that the password has been updated to “secret21234”.

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure of 'MyGuitarShop', including tables like 'Customers', 'Addresses', and 'Orders'. In the center, three tabs are open in the Query Editor:

- C.11.sql - DESKTOP-PTHKBA0\lyq (54) (highlighted)
- C.10.sql - DESKTOP-PTHKBA0\lyq (70)
- C.9e.sql - DESKTOP-PTHKBA0\lyq (53)

The C.11.sql tab contains the following SQL code:

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

Below the code, the Results pane shows two rows of data from the 'Customers' table:

CustomerID	EmailAddress	Password	FirstName	LastName	ShippingAddressID	BillingAddressID
1	486	kriegerrobert@gmail.com	Krieger	Robert	NULL	NULL

Underneath this, another Results pane shows the same row after the update:

CustomerID	EmailAddress	Password	FirstName	LastName	ShippingAddressID	BillingAddressID
1	486	kriegerrobert@gmail.com	secret@1234	Krieger	Robert	NULL

At the bottom of the screen, a message bar indicates: "Query executed successfully." and "2 rows".

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).

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.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the 'MyGuitarShop' database with various objects like Tables, Views, and Procedures. The central pane displays a query window with the following SQL code:

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

The status bar at the bottom indicates "Query executed successfully." and "0 rows".

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).

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'DESKTOP-PTHKBA0'. The central pane displays a query window titled 'D.2.sql - DESKTOP-PTHKBA0\lyq (53)' containing the following T-SQL code:

```
USE MyGuitarShop;
GO
CREATE VIEW Top5BestSelling
AS
SELECT Top 5 ProductName, SUM(ItemPrice * Quantity) AS OrderTotal,
COUNT(OrderItems.ProductID) AS OrderCount
FROM Products, OrderItems
WHERE Products.ProductID = OrderItems.ProductID
GROUP BY ProductName;
GO
SELECT *
FROM Top5BestSelling
```

The 'Results' tab below the query window shows the output of the executed query:

	ProductName	OrderTotal	OrderCount
1	Fender Precision	999.98	2
2	Fender Stratocaster	17619.00	7
3	Gibson Les Paul	14388.00	7
4	Gibson SG	7199.91	5
5	Hofner Icon	1469.97	3

The status bar at the bottom indicates 'Query executed successfully.' and '5 rows'.

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.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'MyGuitarShop'. The central pane displays a T-SQL script for creating a stored procedure named 'spUpdateProductDiscount'. The script includes logic to check if the procedure already exists, drop it if so, and then create it with parameters '@ProductId INT' and '@DiscountPercent FLOAT'. It includes a BEGIN-IF block to update the 'Products' table if the product exists and the discount percent is non-negative. If the discount percent is negative, it throws an error with code 50001 and message 'The value DiscountPercent must be a positive number.' The 'Messages' pane at the bottom shows the command completed successfully and the completion time.

```
USE MyGuitarShop
IF OBJECT_ID('spUpdateProductDiscount') IS NOT NULL
    DROP PROC spUpdateProductDiscount
GO

CREATE PROC spUpdateProductDiscount
    @ProductId INT , @DiscountPercent FLOAT
AS
BEGIN
    IF EXISTS(SELECT*FROM Products WHERE ProductID=@ProductId) AND @DiscountPercent >= 0
        Update Products
        SET DiscountPercent = @DiscountPercent;
    ELSE
        THROW 50001, 'The value DiscountPercent must be a positive number.',1;
END
GO
```

Commands completed successfully.
Completion time: 2020-11-07T05:01:32.2047638+08:00

Query executed successfully.

The EXEC statements are as below:

D.3.sql - DESKTOP-PTHKBA0.MyGuitarShop (DESKTOP-PTHKBA0\lyq (69)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

MyGuitarShop Execute

Object Explorer

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

D.3.sql - DESKTO...PTHKBA0\lyq (69)* SQLQuery28.sql ...THKBA0\lyq (58)* D.1.sql - DESKTO...PTHKBA0\lyq (57)*

```
ELSE
    THROW 50001, 'The value DiscountPercent must be a positive number.',1;
END
GO

USE MyGuitarShop
EXEC spUpdateProductDiscount @ProductID=1, @DiscountPercent = '15';
GO

USE MyGuitarShop
EXEC spUpdateProductDiscount @ProductID=5, @DiscountPercent = '-20';
GO
```

125 % Messages

Mag 50001, Level 16, State 1, Procedure spUpdateProductDiscount, Line 10 [Batch Start Line 22]
The value DiscountPercent must be a positive number.

Completion time: 2020-11-07T04:16:43.4781477+08:00

125 %

Query completed with errors.

DESKTOP-PTHKBA0 (15.0 RTM) DESKTOP-PTHKBA0\lyq (69) MyGuitarShop 00:00:00 0 rows

Item(s) Saved

Ln 23 Col 1 Ch 1 INS

D.3.sql - DESKTOP-PTHKBA0.MyGuitarShop (DESKTOP-PTHKBA0\lyq (69)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

MyGuitarShop Execute

Object Explorer

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

D.3.sql - DESKTO...PTHKBA0\lyq (69)* SQLQuery28.sql ...THKBA0\lyq (58)* D.1.sql - DESKTO...PTHKBA0\lyq (57)*

```
ELSE
    THROW 50001, 'The value DiscountPercent must be a positive number.',1;
END
GO

USE MyGuitarShop
EXEC spUpdateProductDiscount @ProductID=1, @DiscountPercent = '15';
GO

USE MyGuitarShop
EXEC spUpdateProductDiscount @ProductID=5, @DiscountPercent = '-20';
GO
```

125 % Messages

(10 rows affected)

Completion time: 2020-11-07T04:16:14.6268896+08:00

125 %

Query executed successfully.

DESKTOP-PTHKBA0 (15.0 RTM) DESKTOP-PTHKBA0\lyq (69) MyGuitarShop 00:00:00 0 rows

Item(s) Saved

Ln 19 Col 1 Ch 1 INS

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

Common factors of 15 and 30

**1
3
5
15**

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'MyGuitarShop' with various objects like Tables, Views, and Procedures. The central pane displays a T-SQL script:

```
DECLARE @Number1 INT = 15
DECLARE @Number2 INT = 30
DECLARE @Number3 INT = 1
PRINT 'Common factors of 15 and 30'
WHILE @Number3 <= 15
BEGIN
IF (@Number1%@Number3= 0)AND( @Number2%@Number3=0)
PRINT @Number3
SET @Number3=@Number3+1
END
GO
```

The 'Messages' pane at the bottom shows the output of the script:

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

Below the messages, the completion time is shown as 2020-11-07T04:56:50.2636215+08:00. The status bar at the bottom indicates "Query executed successfully." and "0 rows".

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.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure of 'MyGuitarShop' with various tables like 'Addresses', 'Administrators', 'Categories', 'Customers', 'OrderItems', 'Orders', and 'Products'. The central pane contains a query window titled 'D.5.sql - DESKTOP-PTHKBA0\lyq (66)' which contains the following SQL script:

```
USE MyGuitarShop
GO

IF OBJECT_ID('fnDiscountPrice') IS NOT NULL
    DROP FUNCTION fnDiscountPrice
GO

CREATE FUNCTION fnDiscountPrice (@ItemID INT)
RETURNS money
BEGIN
    RETURN (
        SELECT SUM(ItemPrice-DiscountAmount) AS DiscountPrice
        FROM OrderItems
        WHERE @ItemID = ItemID
        GROUP BY ItemID
    )
END
GO

PRINT 'Discount Price:' + CONVERT(varchar,dbo.fnDiscountPrice(2),1);
```

The 'Messages' pane at the bottom shows the output of the query:

```
Discount Price:$303.79
Completion time: 2020-11-07T06:03:16.3860817+08:00
```

The status bar at the bottom indicates 'Query executed successfully.' and provides system information: DESKTOP-PTHKBA0 (15.0 RTM) | DESKTOP-PTHKBA0\lyq (66) | MyGuitarShop | 00:00:00 | 0 rows.

5.(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.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists various database objects like Tables, Views, and Functions. In the center, a query window displays the creation of a function named fnItemTotal. The code is as follows:

```
USE MyGuitarShop
GO

IF OBJECT_ID('fnItemTotal') IS NOT NULL
    DROP FUNCTION fnItemTotal
GO

CREATE FUNCTION fnItemTotal (@ItemID INT)
RETURNS money
BEGIN
    RETURN ( SELECT SUM(ItemPrice*Quantity-DiscountAmount*Quantity) AS TotalAmountofItem
        FROM OrderItems
        WHERE @ItemID = ItemID
        GROUP BY ItemID,Quantity)
END
GO
```

Below the code, the Messages pane shows the output:

```
Total value:$1,208.16
Completion time: 2020-11-07T06:28:22.2868606+08:00
```

At the bottom, a status bar indicates "Query executed successfully." and provides session details: DESKTOP-PTHKBA0 (15.0 RTM) DESKTOP-PTHKBA0\lyq (68) MyGuitarShop 00:00:00 0 rows.

Database Design [20 pts.]

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

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

Here is my database design procedure and screenshot, As you could see I put 6 table in the new database for online cab system.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists databases, including 'DESKTOP-PTHKBA0' and 'OnlineCabSystem'. The 'Tables' node under 'OnlineCabSystem' is expanded, showing six tables: Customer1, Driver, PaymentDetails, Trip, LoginCustomers, and LoginDrivers. The 'Databases' node also lists 'System Tables', 'FileTables', 'External Tables', 'Graph Tables', 'dbo.Car', 'dbo.Customer1', 'dbo.Driver', 'dbo.LoginCustomers', 'dbo.LoginDrivers', 'dbo.PaymentDetails', and 'dbo.Trip'. The 'Messages' pane at the bottom right shows the message 'Commands completed successfully.'

```
USE OnlineCabSystem
CREATE TABLE Customer1
(
    CustomerID INT NOT NULL,
    CustomerName VARCHAR(50) NOT NULL,
    CustomerAddress VARCHAR(50) NOT NULL,
    CustomerPhone VARCHAR(50) NOT NULL,
    NumberOfOrders INT NOT NULL,
    CONSTRAINT Customer1_pk PRIMARY KEY (CustomerID)
);

CREATE TABLE Driver
(
    DriverID INT NOT NULL,
    DriverName VARCHAR(50) NOT NULL,
    DriverPhone VARCHAR(50) NOT NULL,
    AvgGrade INT NOT NULL,
    NumberOfTrips INT NOT NULL,
    CarNumber VARCHAR NOT NULL,
    CONSTRAINT Driver_pk PRIMARY KEY (DriverID)
);

CREATE TABLE PaymentDetails
(
    PaymentID INT NOT NULL,
    DriverID INT NOT NULL REFERENCES Driver(DriverID),
    CustomerID INT NOT NULL REFERENCES Customer1(CustomerID),
    PaymentMethod VARCHAR(50) NULL,
    Price MONEY NOT NULL DEFAULT 0 CHECK (Price >=0),
    CONSTRAINT PaymentDetails_pk PRIMARY KEY (PaymentID)
);
```

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

The relationship between Customer table and Payment is one-many.

The relationship between Drivers table and Payment is one-many.

The relationship between Customer table and Trip is one-many.

The relationship between Car table and Driver is one-one.

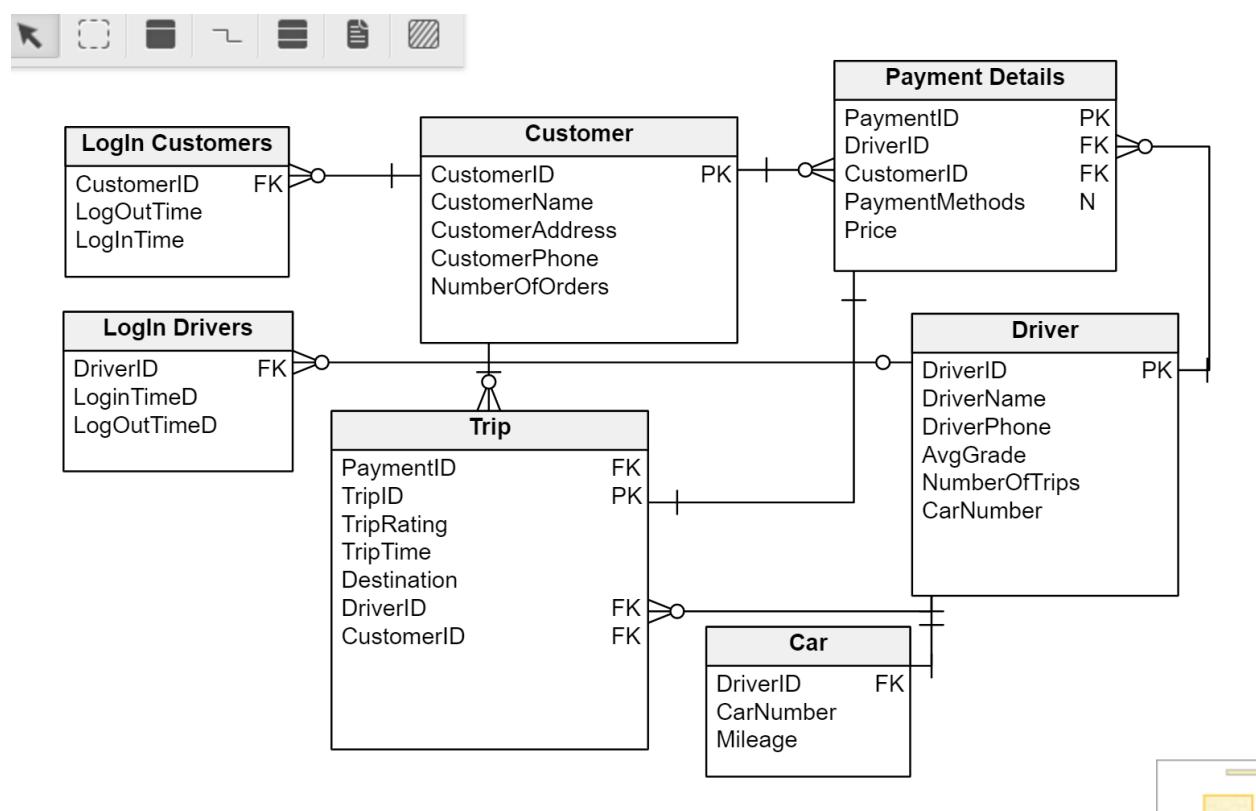
The relationship between Customer table and LoginC is one-many.

The relationship between Driver table and LoginD is one-many.

The Primary keys and Foreign keys are as below.

And the nullabilities is Mileage.

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.

I just suspect one driver only have one car so the relationship between is one-one and I also think people with one account will log lots time during time.

My remarks : I think this course is a little bit hard to understand for me but I strongly think this course is very useful for us to deal with statistics by SQL, it help us gather the statistics and analyze them. I think it's interesting.

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 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.

5