

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. 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.

Project 1 CreateMyGuitarShop.sql - LAPTOP-2NPL54RH\SQLEXPRESS.MyGuitarShop (LAPTOP-2NPL54RH\Asus (53)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query New Results New Task List New Item Change Type

Execute #LastInvoice

Object Explorer

Connect To... MyGuitarShop Execute

Project Explorer

Connect To... MyGuitarShop Execute

Project1.Create... (NPL54RH\Asus (53))

```
*****  
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 PRIMARY KEY IDENTITY,  
CategoryName VARCHAR(100) NOT NULL  
100 rows  
# Messages  
  
(4 rows affected)  
(10 rows affected)  
(485 rows affected)  
(512 rows affected)  
(41 rows affected)  
(3 rows affected)  
  
Completion time: 2020-10-30T01:35:45.634285-04:00
```

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.

SQLQuery1.sql - LAPTOP-2NPL54RH\SQLEXPRESS.MyGuitarShop (LAPTOP-2NPL54RH\Asus (51)) - Microsoft SQL Server Management Studio

```

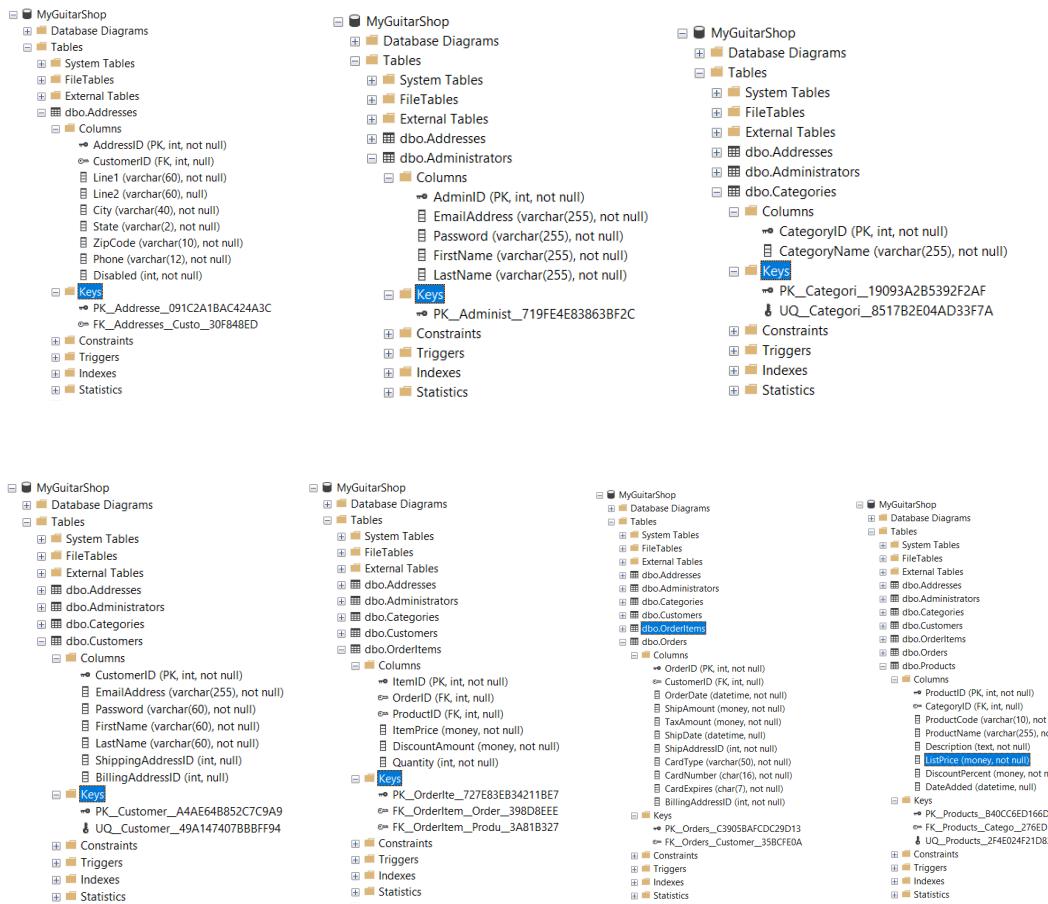
File Edit View Query Project Tools Window Help
Connect... New Query Object Explorer Properties Task List Quick Launch (Ctrl+Q)
#LastInvoice
Object Explorer
Project 1 CreateM... NPL54RH\Asus (51)
LAPTOP-2NPL54RH\SQLEXPRESS (SQL Server 11.0.5026 - LAPTOP-2NPL54RH\Asus (51))
Databases Security File and Filegroups Replication PolyBase Management XEvent Profiler
SQlQuery1.sql - L... NPL54RH\Asus (51)* Project 1 CreateM... NPL54RH\Asus (51)
Name: bo_L1
Date: 10.30.2020
USE [MyGuitarShop]
SELECT * FROM OrderItems

```

Results Messages

LineID	OrderID	ProductID	UnitPrice	DiscountAmount	Quantity
1	1	2	8	119.00	358.70
2	2	2	489.99	186.20	1
3	3	2	249.99	99.99	1
4	4	3	9	415.00	161.85
5	5	4	2	119.00	358.70
6	6	5	299.00	0.00	1
7	7	6	10	299.00	0.00
8	8	7	5	699.99	210.00
9	9	7	5	299.00	0.00
10	10	7	5	699.99	210.00
11	11	8	4	799.99	120.00
12	12	2	489.99	186.20	3
13	13	10	7	499.99	120.00
14	14	11	6	699.99	209.70
15	15	12	2	299.00	0.00
16	16	13	4	799.99	120.00
17	17	14	2	119.00	358.70

Query executed successfully.



B. An Introduction to SQL

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.

```
USE [MyGuitarShop]
```

```
-- Return LastName with EmailAddress
SELECT LastName + ', ' + EmailAddress AS FullName
FROM Customers
-- Last name begins with a letter from A-H
WHERE LEFT(LastName, 1) >= 'A' AND LEFT(LastName, 1) <= 'H'
ORDER BY LastName DESC;
```

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'MyGuitarShop'. The central pane displays the query: 'SELECT LastName + ', ' + EmailAddress AS FullName FROM Customers -- Last name begins with a letter from A-H WHERE LEFT(LastName, 1) >= 'A' AND LEFT(LastName, 1) <= 'H' ORDER BY LastName DESC;'. The results pane below shows a table with 18 rows, each containing a customer's first name, last name, and email address. The columns are labeled 'Name' and 'Email'.

Name	Email
John	John.Doe@email.com
Hannah	Hannah.Smith@email.com
Heather	Heather.Love@email.com
Hugh	Hugh.VanHalen@email.com
Holger	Holger.Schmid@email.com
Holley	Holley.Campbell@email.com
Howard	Howard.Mondello@yahoo.com
House	House.Wu.Breakaway@email.com
Hunter	Hunter.Snow@email.com
Ian	Ian.Huntington@email.com
Imani	Imani.Jones@email.com
Imani	Imani.Jones@yahoo.com

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

 OrderID The OrderID column

 OrderDate The OrderDate column

 ShipDate The ShipDate column

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

```
USE [MyGuitarShop]
```

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

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'MyGuitarShop'. The central pane displays the query: 'SELECT OrderID, OrderDate, ShipDate FROM Orders WHERE OrderDate IS NOT NULL;'. The results pane below shows a table with 17 rows, each containing an order ID, order date, and ship date. The columns are labeled 'OrderID', 'OrderDate', and 'ShipDate'.

OrderID	OrderDate	ShipDate
1	2016-03-29 09:00:00	2016-03-31 09:04:11.000
2	2016-03-29 11:23:00	2016-04-11 20:03:00
3	2016-03-29 04:43:00	2016-04-06 04:41:00
4	2016-03-29 04:43:00	2016-04-06 04:41:00
5	2016-03-29 04:43:00	2016-04-06 04:41:00
6	2016-03-29 04:43:00	2016-04-06 04:41:00
7	2016-04-01 23:11:00	2016-04-21 17:55:00
8	2016-04-01 23:11:00	2016-04-21 17:55:00
9	2016-04-01 23:11:00	2016-04-21 17:55:00
10	2016-04-01 23:11:00	2016-04-21 17:55:00
11	2016-04-09 22:44:00	2016-04-09 22:27:00
12	2016-04-11 22:30:00	2016-04-11 22:14:00
13	2016-04-11 22:30:00	2016-04-11 22:14:00
14	2016-04-11 22:30:00	2016-04-11 22:14:00
15	2016-04-16 14:17:00	2016-04-16 14:08:00
16	2016-04-16 14:17:00	2016-04-16 14:08:00
17	2016-04-16 17:24:00	2016-04-08 17:25:00

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: 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.

```
USE [MyGuitarShop]
```

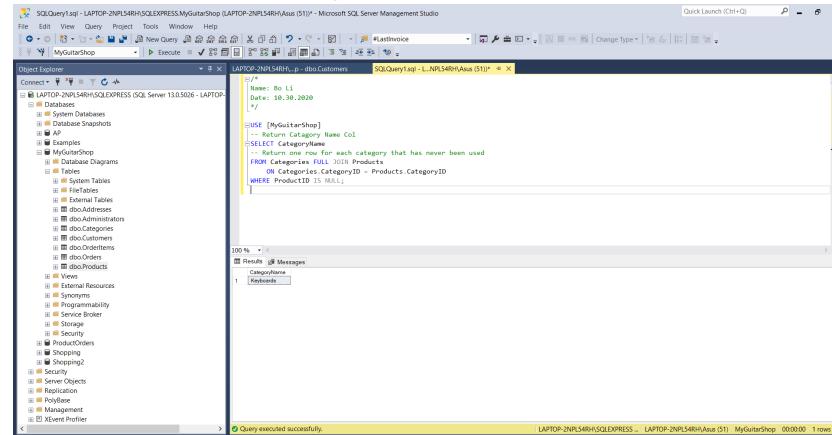
```
-- Return following columns in 4 tables
SELECT LastName, ShipDate, ProductName, Description, ItemPrice, DiscountAmount,
Quantity
-- Join tables Customers, Orders, OrderItems, Products
FROM Customers AS C JOIN Orders AS O ON C.CustomerID = O.CustomerID
JOIN OrderItems AS OI ON O.OrderID = OI.OrderID
JOIN Products AS P ON OI.ProductID = P.ProductID
-- Sort the result by LastName ascending, ShipDate decending, and ProductName
ORDER BY LastName ASC, ShipDate DESC, ProductName;
```

LastName	ShipDate	ProductName	Description	ItemPrice	DiscountAmount	Quantity
Albers	2016-04-08 08:18:28.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar...	699.00	209.70	1
Antoni	2016-04-14 02:22:15.000	Gibson SG	This Gibson SG electric guitar takes the best of the...	799.99	240.00	1
Brown	2016-04-02 15:22:14.000	Gibson Les Paul	This Les Paul guitar offers a carved top and humb...	1199.00	359.70	2
Bull	2016-04-07 06:25:27.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar...	699.00	209.70	1
Caroneira	2016-04-20 17:41:05.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar...	699.00	209.70	1
Cavali		N/NULL	With authentic wood finishes and a solid Canadian...	699.00	136.20	1
Darby	2016-04-07 01:15:00.000	Fender Stratocaster	Featuring a carefully chosen, solid Canadian cedar...	699.00	209.70	1
Dilard	2016-04-08 14:23:30.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar...	699.00	209.70	1
Han	2016-04-15 12:27:25.000	Fender Stratocaster	The Fender Stratocaster is the electric guitar desig...	2517.00	1308.84	1
Esway	2016-04-06 15:03:03.000	Fender Precision Bass	The Fender Precision bass guitar delivers the sound...	499.99	125.00	1
Flosi	2016-04-09 12:34:00.000	Tama 5-Piece Drum Set with Cymbals	The Tama 5-Piece Drum Set is the most affordable...	299.00	0.00	2
Foller	2016-04-09 12:34:00.000	Washburn D10B	This Gibson SG electric guitar takes the best of the...	799.99	240.00	1
Garcia	2016-04-24 17:53:07.000	Gibson SG	The Gibson SG electric guitar takes the best of the...	799.99	240.00	1
Goldstein	2016-04-28 23:37:24.000	Gibson SG	This Gibson SG electric guitar takes the best of the...	799.99	240.00	5
Goldstein	2016-04-06 12:23:14.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar...	699.00	209.70	3
Goldstein	2016-04-03 09:43:54.000	Tama 5-Piece Drum Set with Cymbals	The Tama 5-Piece Drum Set is the most affordable...	299.00	0.00	1

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.)

```
USE [MyGuitarShop]
-- Return Catagory Name Col
SELECT CategoryName
-- Return one row for each category that has never been used
FROM Categories FULL JOIN Products
    ON Categories.CategoryID = Products.CategoryID
WHERE ProductID IS NULL;
```



```
USE [MyGuitarShop]
-- Return Catagory Name Col
SELECT CategoryName
-- Return one row for each category that has never been used
FROM Categories FULL JOIN Products
    ON Categories.CategoryID = Products.CategoryID
WHERE ProductID IS NULL;
```

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

- The EmailAddress column from the Customers table
- The sum of the ItemPrice in the OrderItems table multiplied by the quantity in the OrderItems table
- 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.

```
USE [MyGuitarShop]
```

```
-- Return EmailAddress, Sum of ItemPrice * Quantity, Avg f DiscountAmount * Quantity
SELECT EmailAddress, SUM(ItemPrice * Quantity) AS Total,
       AVG(DiscountAmount * Quantity) AS Average
-- Join Customers, Orders and OrderItems
FROM Customers JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID
JOIN OrderItems
    ON Orders.OrderID = OrderItems.OrderID
-- Group by EmailAddress
GROUP BY EmailAddress
-- Sort by itemPrice total Descending
ORDER BY Total DESC;
```

EmailAddress	Total	Average
david.potter@hotmail.com	6395.96	609.70
allan.sherwood@yahoo.co.uk	4131.00	610.13
lucy.williams@btconnect.com	3216.00	759.27
yuki.whoo@red.com	3216.00	759.27
heathernew@mac.com	3016.90	716.92
mroyer@cyber.com	2917.00	730.84
josephine.morrell@btconnect.org	2917.00	730.84
mette@sol.com	2917.00	730.84
grut@cox.net	2386.00	719.40
louis@cox.net	2386.00	719.40
louise_wesley@cox.net	2386.00	719.40
louise_wesley@cox.net	2386.00	719.40
louis_wesley@cox.net	2386.00	719.40

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.

```
USE [MyGuitarShop]
-- NumberOfOrders: Count the number of Orders
-- TotalAmount: Total Amount for each order
SELECT EmailAddress, COUNT(Orders.OrderID) AS NumberOfOrders, SUM((ItemPrice - DiscountAmount) * Quantity) AS TotalAmount
FROM Customers JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID
JOIN OrderItems
    ON Orders.OrderID = OrderItems.OrderID
-- ItemPrice value have a more than 500
WHERE ItemPrice > 500
GROUP BY EmailAddress
-- Sort the result descending order of EmailAddress
ORDER BY EmailAddress DESC;
```

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left is the Object Explorer pane, which lists the database 'MyGuitarShop' and its objects. In the center is the main query editor window containing the SQL code provided above. Below the code is the Results pane, which displays the output of the query. The results are a table with three columns: EmailAddress, NumberOfOrders, and TotalAmount. The data is sorted by EmailAddress in descending order. The results are as follows:

EmailAddress	NumberOfOrders	TotalAmount
david.goldstein@hotmail.c...	2	4287.85
louis.schiffman@msn.com	1	1875.00
charles.cudby@usa.org	1	489.30
calbarres@gmail.com	1	489.30
louis.schiffman	2	1325.00
art@wave.org	1	879.99
amseale@gmail.com	1	489.30
alene_turbo@cox.net	1	489.30
shawn.schiffman@comcast.com	2	2474.45
ashin@vusenki.com	1	1670.60

At the bottom of the screen, a status bar indicates 'Query executed successfully'.

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.

```
USE [MyGuitarShop]
-- Return 3 columns including EmailAddress, OrderID and Total Amount
SELECT EmailAddress, Orders.OrderID, SUM((ItemPrice-DiscountAmount)*Quantity)
AS TotalAmount
FROM Customers JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID
JOIN OrderItems
    ON Orders.OrderID = OrderItems.OrderID
--Group the result by EmailAddress and OrderID
GROUP BY EmailAddress, Orders.OrderID;
```

The screenshot shows the SQL Server Management Studio interface with two tabs open: 'SQLQuery2.sql' and 'SQLQuery1.sql'. The 'SQLQuery2.sql' tab contains the provided T-SQL code. The 'Results' tab displays the output of the query, which is a table with three columns: EmailAddress, OrderID, and TotalAmount. The data consists of 17 rows, each representing a unique combination of EmailAddress and OrderID, with their corresponding TotalAmount values.

EmailAddress	OrderID	TotalAmount
alan.shewell@yahoo.com	1	839.30
alan.shewell@yahoo.com	2	233.79
alan.shewell@yahoo.com	3	1481.31
christina@varese.com	4	1676.60
christina@varese.com	5	298.90
erik@gmail.com	6	299.00
frank.wilson@global.net	7	1539.97
frank.wilson@global.net	8	232.29
david.potter@verizon.net	9	1487.90
heathernewman@me.com	10	374.99
heathernewman@me.com	11	232.29
josephine.jankay@telecity.org	12	1208.18
art@verne.org	13	679.99
art@verne.org	14	928.60
dorothy.lake@cox.net	15	599.99
ammon@msn.com	16	742.45
ammon@msn.com	17	499.30

- (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.

```
USE [MyGuitarShop]
-- Return emailAddress and its largest order
SELECT EmailAddress, MAX(Large.OrderCost) AS LargestOrder
FROM Customers JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID
JOIN OrderItems
    ON OrderItems.OrderID = Orders.OrderID
-- Derived Table from first Select
JOIN (SELECT Orders.OrderID, (ItemPrice-DiscountAmount)*Quantity AS OrderCost
      FROM Orders JOIN OrderItems
                  ON Orders.OrderID = OrderItems.OrderID) AS Large
    ON Large.OrderID = OrderItems.OrderID
-- Group the result by EmailAddress
GROUP BY EmailAddress;
```

```

-->
Name: Bo Li
Date: 10.30.2020
/*
--> USE [MyGuitarShop]
--> 
--> SELECT EmailAddress, MAX(OrderCost) AS LargestOrder
--> FROM Customers JOIN Orders
--> ON Customers.CustomerID = Orders.CustomerID
--> JOIN OrderItems
--> ON OrderItems.OrderID = Orders.OrderID
--> -- Derived Table from first SELECT
--> WITH LargeOrders AS (SELECT OrderID, SUM([UnitPrice] * Quantity) AS OrderCost
--> FROM Orders JOIN OrderItems
--> ON Orders.OrderID = OrderItems.OrderID) AS Large
--> -- Large Orders
--> Orders.OrderID = Large.OrderID
--> -- Group the result by EmailAddress
--> GROUP BY EmailAddress;
-->

```

EmailAddress	LargestOrder
alison@shaw.ca	1676.60
alan_shaw@shaw.ca	1268.16
alan_shaw@shaw.com	1268.16
amanda@tjw.net	489.30
amanda@tjw.net	679.30
amanda@tjw.net	729.30
bette_mckay@cox.net	838.30
cabane@gmail.com	482.30
carolyn@tjw.net	489.30
christine@shaw.ca	1676.60
david_golden@hotmail.com	2799.95
elaine@shaw.ca	666.95

Query executed successfully.

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.

```

USE [MyGuitarShop]

-- Return 3 cols named EmailAddress, OrderID, and OrderDate
SELECT EmailAddress, OrderID, OrderDate
FROM Customers JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
-- Represent the customer's newest order
WHERE OrderDate =
    (SELECT MAX(OrderDate)
    FROM Orders
    WHERE Customers.CustomerID = Orders.CustomerID);

```

```

-->
Name: Bo Li
Date: 10.30.2020
/*
--> USE [MyGuitarShop]
--> 
--> SELECT EmailAddress, OrderID, OrderDate
--> FROM Customers JOIN Orders
--> ON Customers.CustomerID = Orders.CustomerID
--> -- Represent the customer's newest order
--> WHERE OrderDate =
-->     (SELECT MAX(OrderDate)
-->     FROM Orders
-->     WHERE Customers.CustomerID = Orders.CustomerID);
-->

```

EmailAddress	OrderID	OrderDate
alison@shaw.ca	39	2016-05-09 07:02:30.000
alene_judith@cox.net	40	2016-05-21 14:21:29.000
alan@shaw.ca	38	2016-05-08 11:41:24.000
amanda@tjw.net	37	2016-05-06 14:10:21.000
amanda@tjw.net	36	2016-05-04 14:10:21.000
vincent@aol.com	38	2016-05-04 03:52:23.000
bette_mckay@cox.net	39	2016-05-06 22:22:26.000
carolyn@tjw.net	40	2016-05-04 14:10:21.000
yuki_whewhy@shuttle.com	31	2016-04-29 06:47:14.000
glady.melvin@org	30	2016-04-27 16:21:31.000
meaghan@ruralmail.com	29	2016-04-26 17:53:24.000
meaghan@ruralmail.com	27	2016-04-19 09:17:51.000

Query executed successfully.

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.

```
USE [MyGuitarShop]
```

```
SELECT ListPrice, CAST(ListPrice AS decimal(18, 2)) AS twoDigits,
CAST(ListPrice AS REAL) AS RealNum, CONVERT(int, ListPrice) AS ToInteger
FROM Products
```

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer pane displays the database structure of 'MyGuitarShop' with various tables like 'Categories', 'Customers', 'Orders', etc. In the center, the 'SQLQuery1.sql' window contains the provided T-SQL code. Below it, the 'Results' tab shows the output of the query, which is a table with four columns: 'ListPrice', 'twoDigits', 'RealNum', and 'ToInteger'. The data consists of 10 rows of product prices. At the bottom of the results pane, a status bar indicates 'Query executed successfully.' and provides connection information.

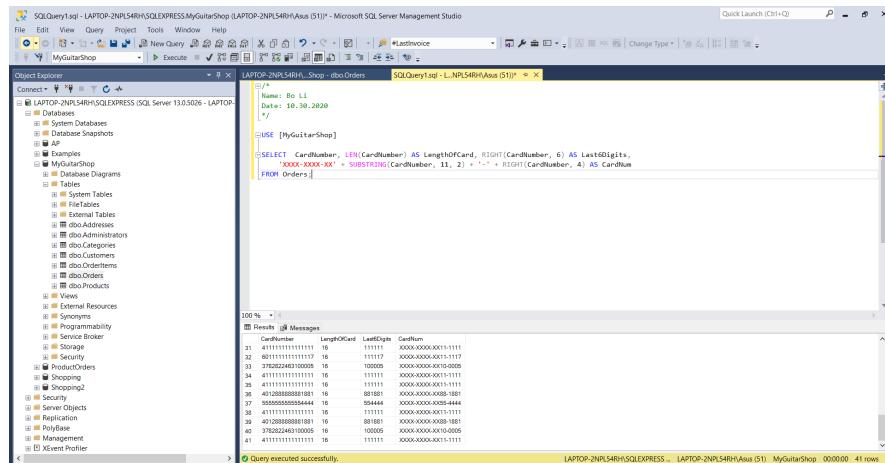
	ListPrice	twoDigits	RealNum	ToInteger
1	699.00	699.00	699	699
2	1199.00	1199.00	1199	1199
3	2517.00	2517.00	2517	2517
4	489.99	489.99	489.99	489
5	499.99	499.99	499	499
6	415.00	415.00	415	415
7	799.99	799.99	799.99	800
8	499.99	499.99	499.99	500
9	699.99	699.99	699.99	700
10	799.99	799.99	799.99	800

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.

USE [MyGuitarShop]

```
SELECT CardNumber, LEN(CardNumber) AS LengthOfCard, RIGHT(CardNumber, 6)
AS Last6Digits,
      'XXXX-XXXX-XX' + SUBSTRING(CardNumber, 11, 2) + ' - ' +
RIGHT(CardNumber, 4) AS CardNum
FROM Orders;
```



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.

```
USE [MyGuitarShop]
-- Return 4 cols in following
SELECT OrderID, OrderDate,
       DATEADD(MONTH, 1, OrderDate) AS ApproxShipDate,
       ShipDate,
       DATEDIFF(DAY, OrderDate, ShipDate) AS DaysToShip
FROM Orders
-- The OrderDate is in April 2016
WHERE YEAR(OrderDate) = 2016 AND MONTH(OrderDate) = 4;
```

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure of 'MyGuitarShop' including databases, tables, and views. In the center, a query window titled 'LAPTOP-2NPL54RH\Shop - dbo.Orders' contains the provided T-SQL code. Below the code, the 'Results' tab shows a grid of 25 rows of data from the 'Orders' table, with columns: OrderID, OrderDate, ApproxShipDate, ShipDate, and DaysToShip. At the bottom of the results grid, a message says 'Query executed successfully.'

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

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

Before:

The screenshot shows the Object Explorer on the left with 'MyGuitarShop' selected. The 'Tables' node under 'MyGuitarShop' is expanded, showing 'Customers'. The 'Customers' table is selected in the center pane. The data grid shows 14 rows of customer information, including columns: CustomerID, EmailAddress, Password, FirstName, LastName, and ShipVia. The 'EmailAddress' column contains various email addresses like 'luis.gallego@gmail.com', 'david.williams@yahoo.com', and 'kriegerrobert@gmail.com'. The 'FirstName' and 'LastName' columns show names like 'Luis', 'David', 'John', etc. The 'ShipVia' column shows values like 2, 3, 4, etc. The status bar at the bottom right indicates '485 rows'.

Insert:

```
USE [MyGuitarShop]
```

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

The screenshot shows the same environment as the previous one. The query window now displays the executed statement and the message 'Query executed successfully.' at the bottom. The status bar at the bottom right indicates '0 rows affected'.

After:

The screenshot shows the same environment. The 'Customers' table now has 15 rows, as indicated by the status bar '486 rows'. The new row is the last one in the list, with CustomerID 15, EmailAddress 'kriegerrobert@gmail.com', Password '' (empty string), FirstName 'Krieger', and LastName 'Robert'. The 'ShipVia' column shows the value 2.

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.

Before:

Update:

```
USE [MyGuitarShop]
```

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

After:

D. Advanced SQL skills

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.

```
USE [MyGuitarShop];
GO
-- Create view name OrderItemProductsDetails with folloing columms
CREATE VIEW OrderItemProductsDetails
AS
-- All Cols from tables Orders, OrderItems, and Products
SELECT Orders.OrderID, OrderDate, TaxAmount, ShipDate,
       ItemPrice, DiscountAmount, (ItemPrice-DiscountAmount) AS FinalPrice,
       Quantity, (Quantity*(ItemPrice-DiscountAmount+TaxAmount)) AS
ItemTotal, ProductName, Description
FROM Orders JOIN OrderItems
    ON Orders.OrderID = OrderItems.OrderID
JOIN Products
    ON Products.ProductID = OrderItems.ProductID;
```

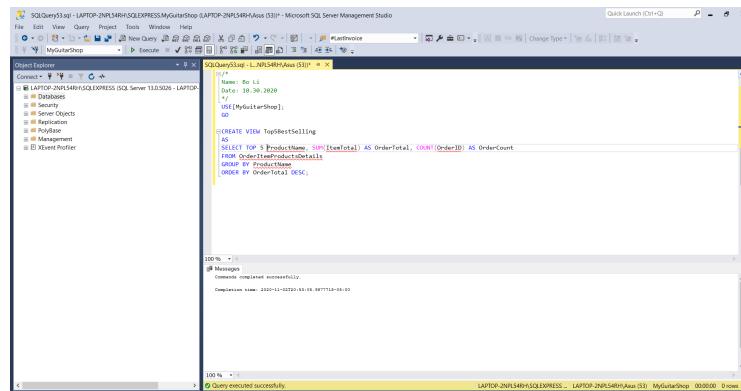
The screenshot shows the Microsoft SQL Server Management Studio interface. In the center, there's a query window titled "SQLQuery53.sql - LAPTOP-2NPL54RH\SQLEXPRESS MyGuitarShop (LAPTOP-2NPL54RH\Asus (S3))". It contains the T-SQL code for creating the view. Below the query window is a results pane showing the message "Commands completed successfully." and the completion time. At the bottom, it says "Query executed successfully."

Verification:

The screenshot shows the Microsoft SQL Server Management Studio interface again. This time, a query window titled "SQLQuery53.sql - LAPTOP-2NPL54RH\SQLEXPRESS MyGuitarShop (LAPTOP-2NPL54RH\Asus (S3))" contains a simple "SELECT * FROM OrderItemProductsDetails;" query. The results pane below it shows a table with 47 rows of data, each representing a different order item with its details like product name and description.

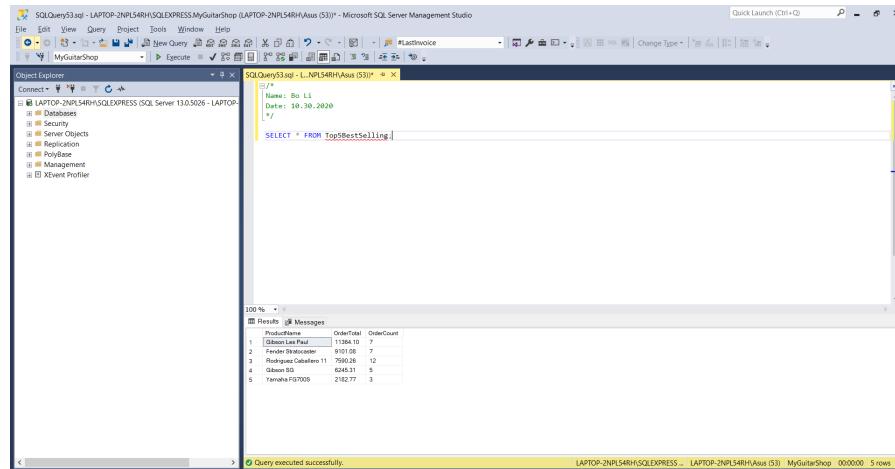
2. [5] Create a view named Top5BestSelling that uses the view you created in Section D
 Question 1. This view should return some summary information about five best selling products. Each row should include these columns: ProductName, OrderTotal (the total sales for the product) and OrderCount (the number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

```
USE[MyGuitarShop];
GO
CREATE VIEW Top5BestSelling
AS
SELECT TOP 5 ProductName, SUM(ItemTotal) AS OrderTotal, COUNT(OrderID) AS
OrderCount
FROM OrderItemProductsDetails
GROUP BY ProductName
ORDER BY OrderTotal DESC;
```



The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, a database named 'MyGuitarShop' is selected. In the center pane, a query window titled 'SQLQuery1.sql - L-NP154RH(Aus53)' contains the T-SQL code for creating the view. The code defines a view named 'Top5BestSelling' that selects the top 5 products by order total, including their names, total sales ('OrderTotal'), and the count of orders ('OrderCount'). The 'AS' keyword is used to alias these columns. The 'FROM' clause points to the 'OrderItemProductsDetails' table. The 'GROUP BY' clause groups the results by product name. The 'ORDER BY' clause sorts the results in descending order of 'OrderTotal'. The status bar at the bottom indicates 'Query executed successfully.'

Verification:



The screenshot shows the same SQL Server Management Studio interface. In the Object Explorer, the 'MyGuitarShop' database is selected. In the center pane, a query window titled 'SQLQuery2.sql - L-NP154RH(Aus53)' contains the T-SQL code 'SELECT * FROM Top5BestSelling;'. The results pane shows a table with three columns: 'ProductName', 'OrderTotal', and 'OrderCount'. The table lists five products: 'Bo Li' (OrderTotal: 11384.10, OrderCount: 1), 'Fender Stratocaster' (OrderTotal: 9107.08, OrderCount: 7), 'Rodriguez Caballero' (OrderTotal: 7950.26, OrderCount: 12), 'Gibson SG' (OrderTotal: 6245.31, OrderCount: 9), and 'Fender F5700B' (OrderTotal: 2182.77, OrderCount: 3). The status bar at the bottom indicates 'Query executed successfully.'

ProductName	OrderTotal	OrderCount
Bo Li	11384.10	1
Fender Stratocaster	9107.08	7
Rodriguez Caballero	7950.26	12
Gibson SG	6245.31	9
Fender F5700B	2182.77	3

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.

```
USE[MyGuitarShop];
GO

CREATE PROC spUpdateProductDiscount
    @Product_ID int,
    @Discount_Perc int
AS
IF @Discount_Perc < 0
    THROW 50001, 'Discount Precent must be positive!', 1;
ELSE
    UPDATE Products
    SET DiscountPercent = @Discount_Perc
    WHERE ProductID = @Product_ID;
```

Test 1 (With Error):

Test 2: (No Error)

Before:

SQLQuery10 - LAPTOP-2H54RH-Ause [3] - Microsoft SQL Server Management Studio

Object Explorer

Connect - MyGuitarShop -> L-NP54RH-Ause [3] -> [master] -> [MyGuitarShop]

File Edit View Query Project Tools Window Help

SQLQuery10 - L-NP54RH-Ause [3] -> SQLQuery10 - L-NP54RH-Ause [3] ->

Quick Launch (Ctrl+Q)

Results (0 rows)

Messages

```
/*
```

```
CREATE TABLE [dbo].[Products]
(
    [ProductID] INT NOT NULL,
    [CategoryID] INT NOT NULL,
    [SupplierID] INT NOT NULL,
    [ProductName] NVARCHAR(40) NOT NULL,
    [ProductCode] NVARCHAR(15) NOT NULL,
    [Description] NVARCHAR(100) NOT NULL,
    [UnitPrice] MONEY NOT NULL,
    [UnitsInStock] SMALLINT NOT NULL,
    [UnitsOnOrder] SMALLINT NOT NULL,
    [ReorderLevel] SMALLINT NOT NULL,
    [Discontinued] BIT NOT NULL
)
```

```
SELECT *
FROM Products
WHERE ProductID = 5
```

1 row(s) affected.

Query executed successfully.

LAPTOP-2H54RH-AUSE [3] -> LAPTOP-2H54RH-Ause [3] -> MyGuitarShop -> 0 rows

EXEC:

SQLQuery10 - LAPTOP-2H54RH-Ause [3] - Microsoft SQL Server Management Studio

Object Explorer

Connect - MyGuitarShop -> L-NP54RH-Ause [3] -> [master] -> [MyGuitarShop]

File Edit View Query Project Tools Window Help

SQLQuery10 - L-NP54RH-Ause [3] -> SQLQuery10 - L-NP54RH-Ause [3] ->

Quick Launch (Ctrl+Q)

Results (0 rows)

Messages

```
/*
```

```
CREATE TRY
BEGIN TRY
    UPDATE ProductSkuSet
    SET Discount_Percent = 10
    WHERE Product_ID = 3;
END TRY
BEGIN CATCH
    PRINT 'An Error Occurred!';
    PRINT 'Error Number: ' + CONVERT(varchar, ERROR_NUMBER());
    PRINT 'Error Message: ' + CONVERT(varchar, ERROR_MESSAGE());
END CATCH
```

100 %

Messages

1 row affected.

Completion time: 2019-01-10T09:31:00.3030000-05:00

Query executed successfully.

LAPTOP-2H54RH-AUSE [3] -> LAPTOP-2H54RH-Ause [3] -> MyGuitarShop -> 0 rows

AFTER:

SQLQuery10 - LAPTOP-2H54RH-Ause [3] - Microsoft SQL Server Management Studio

Object Explorer

Connect - MyGuitarShop -> L-NP54RH-Ause [3] -> [master] -> [MyGuitarShop]

File Edit View Query Project Tools Window Help

SQLQuery10 - L-NP54RH-Ause [3] -> SQLQuery10 - L-NP54RH-Ause [3] ->

Quick Launch (Ctrl+Q)

Results (0 rows)

Messages

```
/*
```

```
CREATE TABLE [dbo].[Products]
(
    [ProductID] INT NOT NULL,
    [CategoryID] INT NOT NULL,
    [SupplierID] INT NOT NULL,
    [ProductName] NVARCHAR(40) NOT NULL,
    [ProductCode] NVARCHAR(15) NOT NULL,
    [Description] NVARCHAR(100) NOT NULL,
    [UnitPrice] MONEY NOT NULL,
    [UnitsInStock] SMALLINT NOT NULL,
    [UnitsOnOrder] SMALLINT NOT NULL,
    [ReorderLevel] SMALLINT NOT NULL,
    [Discontinued] BIT NOT NULL
)
```

```
SELECT *
FROM Products
WHERE ProductID = 5
```

1 row(s) affected.

Results (1 row)

Details (0 messages)

ProductID	CategoryID	SupplierID	ProductName	ProductCode	Description	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
5	1	1	Gitarra Eléctrica	SG-100	The Gibson SG electric guitar takes the best...	297.00	50	0	100	0

Query executed successfully.

LAPTOP-2H54RH-AUSE [3] -> LAPTOP-2H54RH-Ause [3] -> MyGuitarShop -> 0 rows

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
```

```
CREATE PROC spCommonFactor
AS
BEGIN
    DECLARE @cnt int
    SET @cnt = 1;

    PRINT 'Common factors of 15 and 30';

    WHILE (@cnt <= 15)
        BEGIN
            IF (15 % @cnt = 0 AND 30 % @cnt = 0)
                PRINT @cnt;
            SET @cnt = @cnt + 1
        END;
    END;
GO

EXEC spCommonFactor;
```

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'MyGuitarShop' is selected. In the center pane, a query window titled 'SQLQuery27.sql - L_NPL54RH\Asus (54)*' contains the stored procedure code provided above. Below the code, the 'Messages' tab shows the output of the execution:

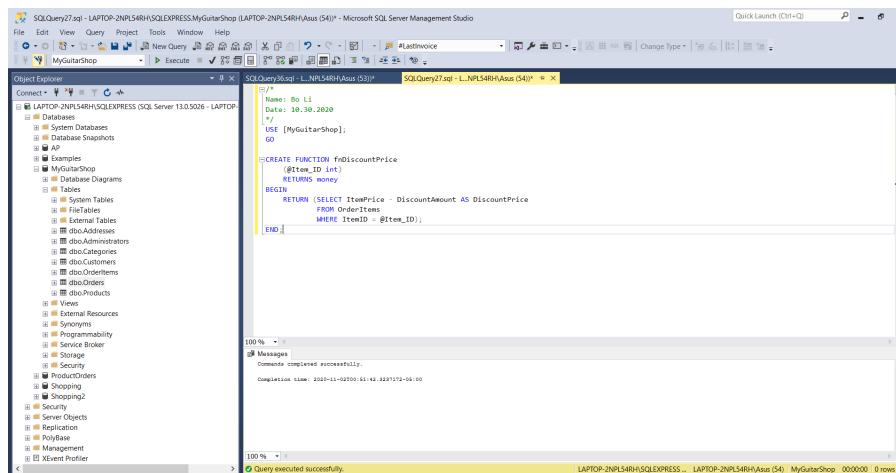
```
1
3
5
15
```

At the bottom of the message pane, it says 'Query executed successfully.'

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.

```
USE [MyGuitarShop];
GO

CREATE FUNCTION fnDiscountPrice
    (@Item_ID int)
    RETURNS money
BEGIN
    RETURN (SELECT ItemPrice - DiscountAmount AS DiscountPrice
            FROM OrderItems
            WHERE ItemID = @Item_ID);
END;
```



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

```
CREATE FUNCTION fnItemTotal
    (@Item_ID int)
    RETURNS money
BEGIN
    RETURN (
        SELECT dbo.fnDiscountPrice(@Item_ID) * Quantity AS TotalAmount
        FROM OrderItems
        WHERE ItemID = @Item_ID
    )
END;
```

SQLQuery3.sql - LAPTOP-2NPL54RH\SQLEXPRESS.MyGuitarShop (LAPTOP-2NPL54RH\Aus (5)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

Object Explorer

Connections MyGuitarShop

SQLQuery1.tot - L_NPL54RH\Aus (5) SQLQuery4.tot - L_NPL54RH\Aus (5) SQLQuery3.tot - L_NPL54RH\Aus (5) SQLQuery2.tot - L_NPL54RH\Aus (5)

```
CREATE FUNCTION [dbo].[TotalItemTotal]
(@Item_ID int)
RETURNS money
BEGIN
    RETURN(
        SELECT dbo.[DiscountPrice](@Item_ID) ^ @Quantity AS TotalAmount
        FROM [OrderItems]
        WHERE ItemID = @Item_ID
    );
END;
```

100% 0 Messages

Command completed successfully.

Completion time: 2022-11-07T01:17:08.5213238+08:00

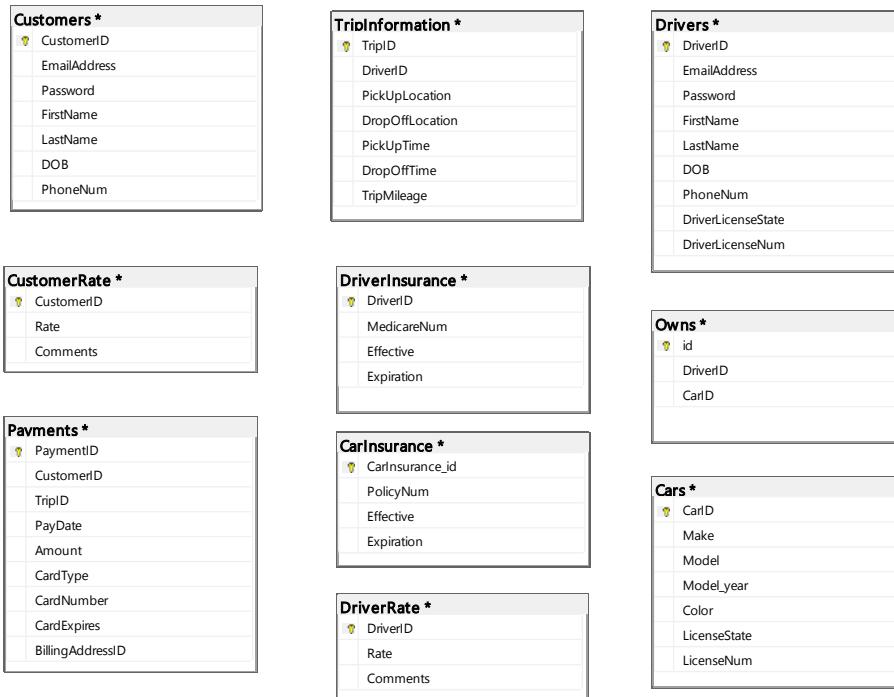
Query executed successfully.

LAPTOP-2NPL54RH\SQLEXPRESS... LAPTOP-2NPL54RH\Aus (5) MyGuitarShop 000000 0 rows.

II. Database Design

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.



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

Table	PK	FK
Customers	CustomerID	
Drivers	DriverID	
CustomerRate	CustomerID	CustomerID
DriverRate	DriverID	DriverID
TripInformation	TripID	DriverID
Payments	PaymentID	CustomerID, TripID
Cars	CarID	
Owns	id	DriverID, CarID
DriverInsurance	DriverID	DriverID
CarInsurance	CarInsurance_id	CarInsurance_id

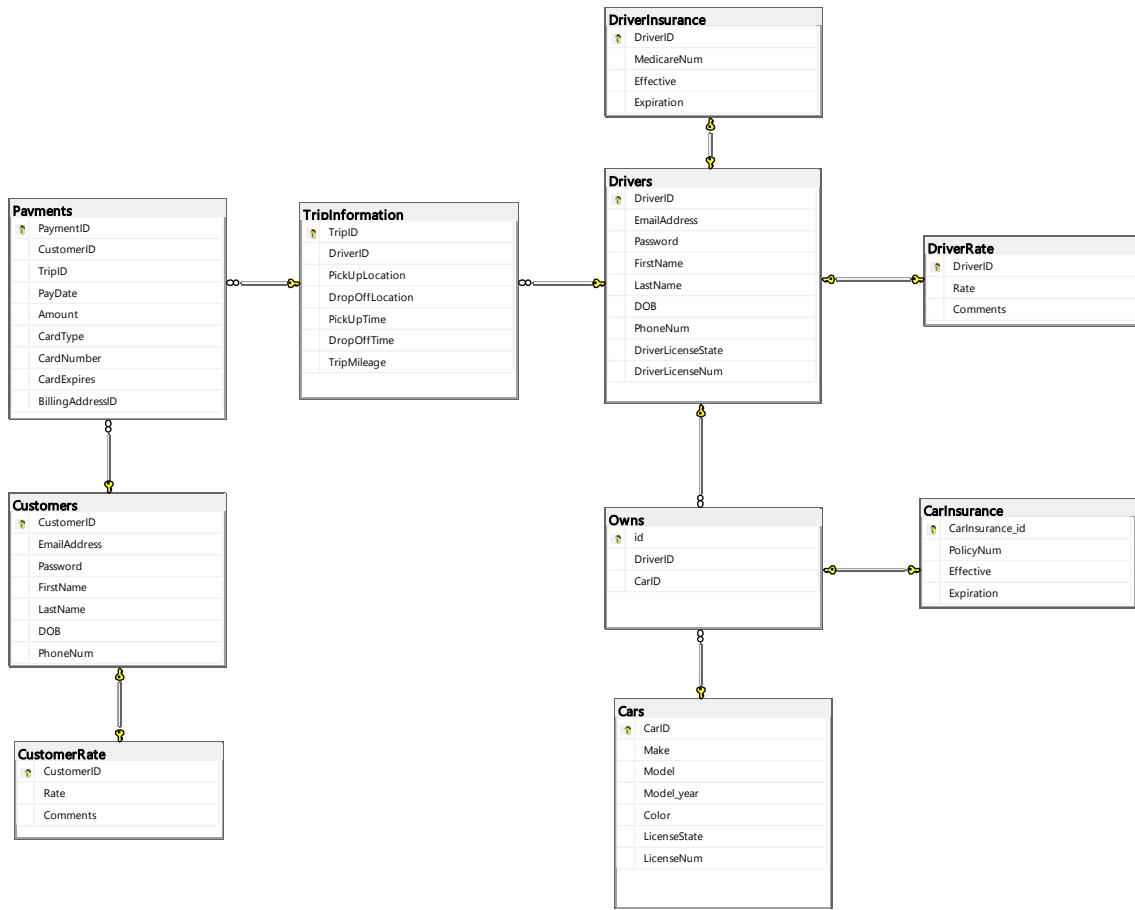
Nullabilities:

Table	Nullabilities Cols
CustomerRate	Comments
DriverRate	Comments

Relationships:

- 1) Customers - CustomerRate: One-One
- 2) Drivers – DriverRate: One-One
- 3) Drivers – DriverInsurance: One – One
- 4) Customers – Payments: One-Many
- 5) TripInformation – Payments: One – Many
- 6) Customers – TripInformation: Many – Many
- 7) Drivers – TripInformation: One – Many
- 8) Drivers - Cars: Many – Many
- 9) Owns – CarInsurance: One – One

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.

In this design, there are some main tables I need to consider creating first, such as Customers, Drivers, TripInformation, Cars, Payments. These tables are the keys of the database to be connected. Secondly Some minor tables, such as rate, insurance, need to be relevant to the major tables using FK -> PK. I also create a table named “Owns”, meaning that driver owns cars, which is relationship between drivers and cars. The following are the relationships among tables in the database.

- 1) **Customers - CustomerRate:** Each Customer has one overall rating statement, so that other users can view it clearly.
- 2) **Drivers – DriverRate:** Each Driver has one overall rating statement, so that other users can view it clearly.
- 3) **Drivers – DriverInsurance:** Each driver has only one insurance for their safety (Alike health insurance). Each insurance only can cover one driver.
- 4) **Customers – Payments:** Each Customer can make more payments for their orders.
- 5) **TripInformation – Payments:** Each Trip can be split by one or more payments. E.g. More customers share one trip payment.
- 6) **Customers – TripInformation:** A Customer can have more trips. More customers can share a trip together.
- 7) **Drivers – TripInformation:** A driver can have more trips by accepting orders.
- 8) **Drivers - Cars:** Each Driver can own one or more cars. Each car can be driven by one or more drivers. E.g. the car is sold to other drivers
- 9) **Owns – CarInsurance:** Each car insurance is only covered by unique car and driver. E.g. Other drivers use this car should put their names on the insurance.

Remark:

In this project, I gained a knowledge with much more deeper understanding of Database system and SQL server. In MyGuitarShop Database, I enhance my skills such as essential SQL skill, Advanced SQL Skill. Based on the exist database, I know how to retrieve data what I want by SELECT. How to create View, Procedures and functions in database. How to use conditions such as If statement in SQL as well. Also, I learned how to design and create a database with MS Visio.