

## Project 1

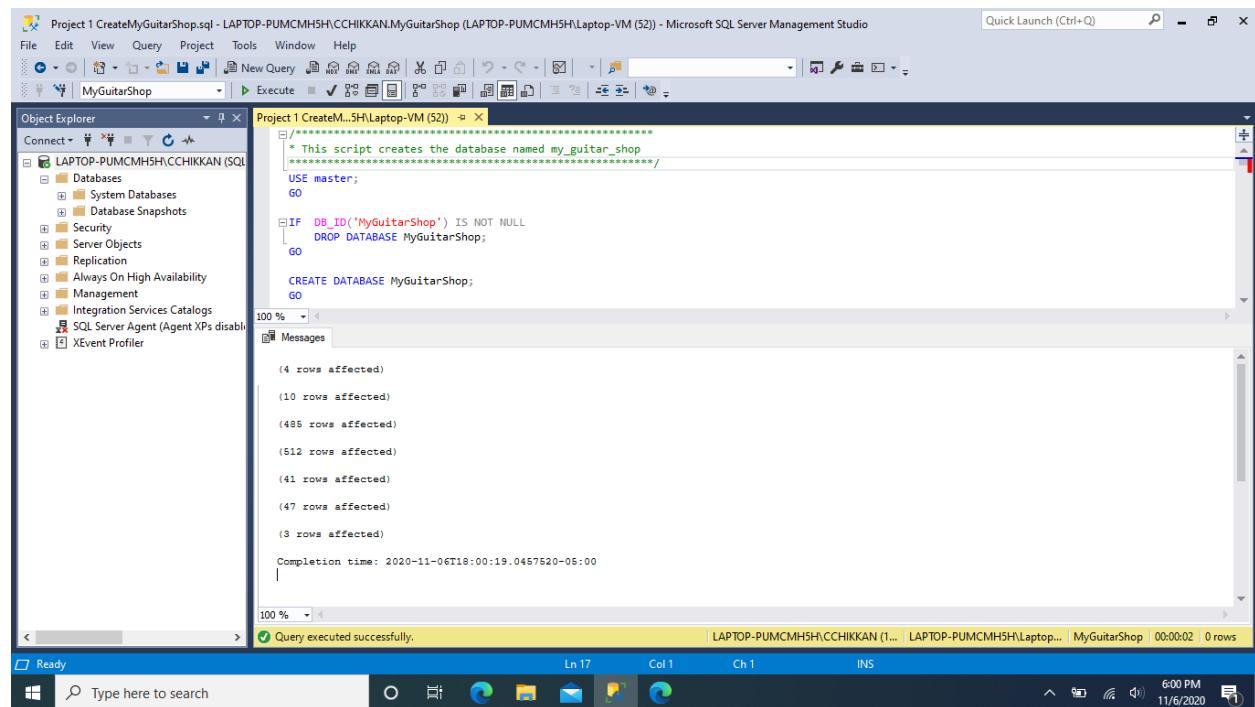
### I. My Guitar Shop Database

#### 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.

Solution:

The CreateMyGuitarShop sql file is downloaded from the Blackboard and opened in SQL server management studio. Once executed, a new database named **MyGuitarShop** is created in the management studio.



The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar reads "Project 1 CreateMyGuitarShop.sql - LAPTOP-PUMCMH5H\CHIKKAN.MyGuitarShop (LAPTOP-PUMCMH5H\Laptop-VM (52)) - Microsoft SQL Server Management Studio". The Object Explorer pane on the left shows the connection to "LAPTOP-PUMCMH5H\CHIKKAN (SQL Server)" with various database-related objects like System Databases, Security, and Replication. The main Results pane displays the execution of the SQL script "CreateMyGuitarShop.sql". The script creates a database named "MyGuitarShop" if it does not already exist. The execution results in the Messages tab show multiple rows affected (4, 10, 485, 512, 41, 47, 3) and a completion time of 2020-11-06T18:00:19.0457520-05:00. A status bar at the bottom indicates "Query executed successfully."

- (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.

Solution:

The column definitions for Addresses Table:

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: Addresses, Administrators, Categories, Customers, OrderItems, Orders, and Products. The main pane displays the column definitions for the Addresses table. The table has the following columns:

Column Name	Data Type	Allow Nulls
AddressID	int	<input type="checkbox"/>
CustomerID	int	<input checked="" type="checkbox"/>
Line1	varchar(60)	<input type="checkbox"/>
Line2	varchar(60)	<input checked="" type="checkbox"/>
City	varchar(40)	<input type="checkbox"/>
State	varchar(2)	<input type="checkbox"/>
ZipCode	varchar(10)	<input type="checkbox"/>
Phone	varchar(12)	<input type="checkbox"/>
Disabled	int	<input type="checkbox"/>

The Column Properties window on the right shows the following details for the AddressID column:

- (General)**: (Name) AddressID, Allow Nulls No, Data Type int
- Table Designer**: (General)

The column definitions for Administrators Table:

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: Addresses, Administrators, Categories, Customers, OrderItems, Orders, and Products. The main pane displays the column definitions for the Administrators table. The table has the following columns:

Column Name	Data Type	Allow Nulls
AdminID	int	<input type="checkbox"/>
EmailAddress	varchar(255)	<input type="checkbox"/>
Password	varchar(255)	<input type="checkbox"/>
FirstName	varchar(255)	<input type="checkbox"/>
LastName	varchar(255)	<input type="checkbox"/>

The Column Properties window on the right shows the following details for the AdminID column:

- (General)**: (Name) AdminID, Allow Nulls No, Data Type int
- Table Designer**: (General)

The column definitions for Categories Table:

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar indicates the connection is to 'LAPTOP-PUMCMH5H\CCHIKKAN.MyGuitarShop - dbo.Categories'. The Object Explorer on the left shows the database structure, including tables like 'Categories', 'Products', and 'Orders'. The main pane displays the 'Columns' tab for the 'Categories' table. It lists two columns: 'CategoryID' (int, Allow Nulls) and 'CategoryName' (varchar(255), Allow Nulls). A 'Column Properties' window is open for 'CategoryID', showing its name, data type (int), and other properties. The status bar at the bottom right shows the date and time as 11/6/2020 5:57 PM.

The column definitions for Customers Table:

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar indicates the connection is to 'LAPTOP-PUMCMH5H\CCHIKKAN.MyGuitarShop - dbo.Customers'. The Object Explorer on the left shows the database structure, including tables like 'Customers', 'Products', and 'Orders'. The main pane displays the 'Columns' tab for the 'Customers' table. It lists six columns: 'CustomerID' (int, Allow Nulls), 'EmailAddress' (varchar(255), Allow Nulls), 'Password' (varchar(60), Allow Nulls), 'FirstName' (varchar(60), Allow Nulls), 'LastName' (varchar(60), Allow Nulls), 'ShippingAddressID' (int, Allow Nulls), and 'BillingAddressID' (int, Allow Nulls). A 'Column Properties' window is open for 'CustomerID', showing its name, data type (int), and other properties. The status bar at the bottom right shows the date and time as 11/6/2020 5:57 PM.

The column definitions for OrderItems Table:

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar indicates the connection is to 'LAPTOP-PUMCMH5H\CCHIKKAN.MyGuitarShop - dbo.OrderItems'. The Object Explorer on the left shows the database structure, including tables like Administrators, Categories, Customers, and OrderItems. The main pane displays the column definitions for the OrderItems table:

Column Name	Data Type	Allow Nulls
ItemID	int	<input type="checkbox"/>
OrderID	int	<input checked="" type="checkbox"/>
ProductID	int	<input checked="" type="checkbox"/>
ItemPrice	money	<input type="checkbox"/>
DiscountAmount	money	<input type="checkbox"/>
Quantity	int	<input type="checkbox"/>

A 'Column Properties' window is open for the ItemID column, showing the following details:

- (General)**
  - Name:** ItemID
  - Allow Nulls:** No
  - Data Type:** int
  - Default Value or Binding:**
- Table Designer** (General)

The column definitions for Orders Table:

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar indicates the connection is to 'LAPTOP-PUMCMH5H\CCHIKKAN.MyGuitarShop - dbo.Orders'. The Object Explorer on the left shows the database structure, including tables like Categories, Customers, and Orders. The main pane displays the column definitions for the Orders table:

Column Name	Data Type	Allow Nulls
OrderID	int	<input type="checkbox"/>
CustomerID	int	<input checked="" type="checkbox"/>
OrderDate	datetime	<input type="checkbox"/>
ShipAmount	money	<input type="checkbox"/>
TaxAmount	money	<input type="checkbox"/>
ShipDate	datetime	<input checked="" type="checkbox"/>
ShipAddressID	int	<input type="checkbox"/>
CardType	varchar(50)	<input type="checkbox"/>
CardNumber	char(16)	<input type="checkbox"/>
CardExpires	char(7)	<input type="checkbox"/>
BillingAddressID	int	<input type="checkbox"/>

A 'Column Properties' window is open for the OrderID column, showing the following details:

- (General)**
  - Name:** OrderID
  - Allow Nulls:** No
  - Data Type:** int
  - Default Value or Binding:**
- Table Designer** (General)

The column definitions for Products Table:

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure for 'MyGuitarShop'. The 'Products' table is selected in the 'Tables' section. The 'LAPTOP-PUMCMH5H...p - dbo.Products' tab is active at the top. Below it, the 'Column Name' and 'Data Type' columns are displayed for the 'Products' table. The 'ProductID' column is defined as an int type with 'Allow Nulls' checked. Other columns include CategoryID, ProductCode, ProductName, Description, ListPrice, DiscountPercent, and DateAdded. A 'Column Properties' pane on the right provides detailed information for the 'ProductID' column, showing it is not nullable and has a data type of int.

Query to show details in Orders table and OrdersItems table using SELECT statement:

**SELECT \* from MyGuitarShop.dbo.Orders;**

The screenshot shows the Microsoft SQL Server Management Studio interface with multiple tabs open. The 'a2.sql' tab is active, displaying the query 'SELECT \* FROM Orders;'. The results pane below shows the data from the 'Orders' table. The table has columns: OrderID, CustomerID, OrderDate, ShipAmount, TaxAmount, ShipDate, ShipAddressID, CardType, CardNumber, CardExpires, and BillingAddressID. The results show 41 rows of data, with the first few rows being:

OrderID	CustomerID	OrderDate	ShipAmount	TaxAmount	ShipDate	ShipAddressID	CardType	CardNumber	CardExpires	BillingAddressID
1	1	2016-03-28 09:40:28.000	5.00	58.75	2016-03-31 09:41:11.000	1	Visa	4111111111111111	04/2018	2
2	2	2016-03-28 11:23:20.000	5.00	21.27	2016-03-31 11:24:03.000	3	Visa	401288888881881	08/2020	3
3	1	2016-03-29 09:44:58.000	10.00	102.29	2016-04-01 09:45:41.000	1	Visa	4111111111111111	06/2020	2
4	3	2016-03-30 15:22:31.000	10.00	117.50	2016-04-02 15:23:14.000	4	American Express	37828246310005	02/2017	4

**SELECT \* from MyGuitarShop.dbo.OrdersItems;**

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 displays the following SQL code:

```
SELECT * FROM Orders;
SELECT * FROM OrderItems;
```

The results pane shows a table with the following data:

	ItemID	OrderID	ProductID	UnitPrice	DiscountAmount	Quantity
1	1	1	2	1199.00	359.70	1
2	2	2	8	489.99	186.20	1
3	3	3	1	2517.00	1308.84	1
4	4	3	9	415.00	161.85	1
5	5	4	2	1199.00	359.70	2
6	6	5	10	299.00	0.00	1
7	7	6	10	299.00	0.00	1
8	8	7	5	699.99	210.00	1
9	9	7	3	799.99	240.00	1
10	10	7	5	699.99	210.00	1
11	11	8	4	799.99	120.00	1

Below the results, a message indicates: "Query executed successfully." The status bar at the bottom right shows the date and time: 11/6/2020 9:22 PM.

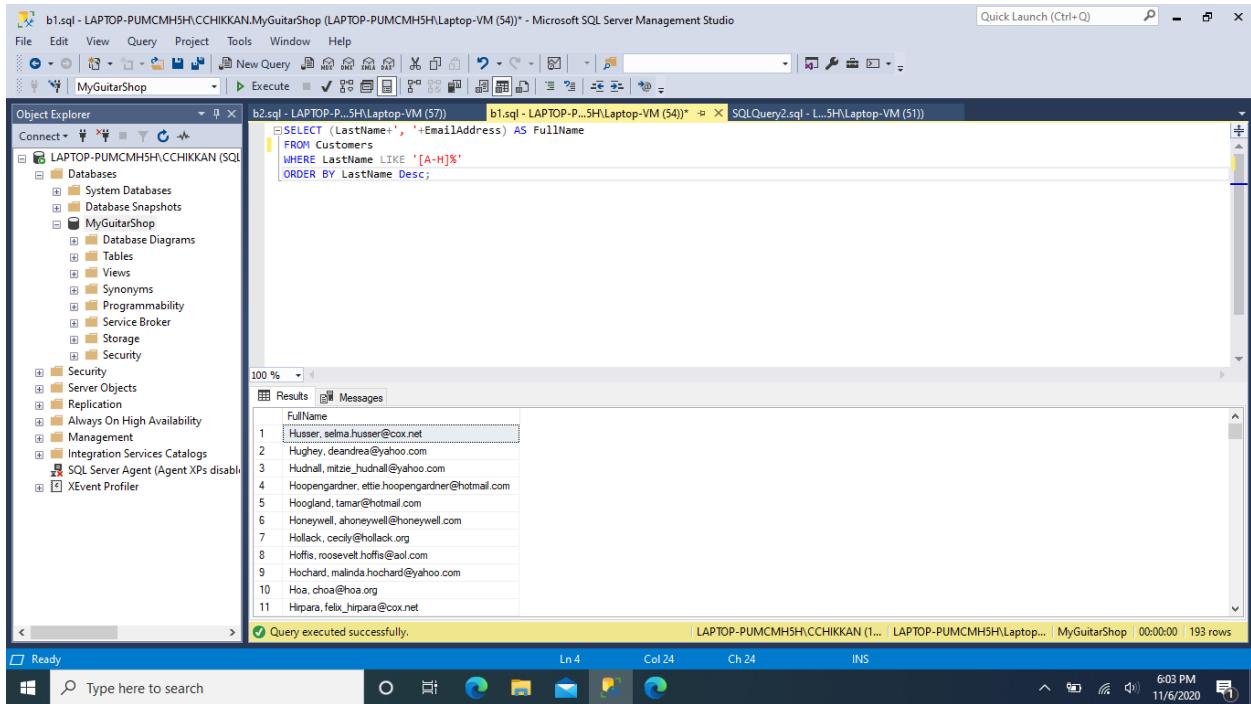
## B. An Introduction to SQL [6 pts.]

- [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.

Solution:

The below query returns the Full name of customers by combining the last name and email address columns and sorted by last name in descending order.

```
SELECT (LastName+', '+EmailAddress) AS FullName
FROM Customers
WHERE LastName LIKE '[A-H]%'
ORDER BY LastName Desc;
```



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.

**Solution:**

The below query that returns OrderID, OrderDate and ShipDate columns and only those rows where ShipDate is not null.

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

```

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

```

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

Query executed successfully.

### C. The essential SQL skills [44 pts.]

- [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

Solution:

The above query is the SELECT statement that returns LastName, ShipDate, ProductName, Description, ItemPrice, DiscountAmount, and Quantity columns from Customers, Orders, OrderItems, and Products tables and also sorting the result set by LastName, ShipDate, and ProductName.

```

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

```

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'MyGuitarShop' is selected. In the center pane, a query window displays a SELECT statement. The results grid shows 47 rows of data from the query:

Last Name	Ship Date	Product Name	Description	Item Price	Discount Amount	Quantity
Albarez	2016-04-23 08:15:28.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar...	699.00	209.70	1
Amigón	2016-04-14 08:22:15.000	Gibson SG	This Gibson SG electric guitar takes the best of th...	799.99	240.00	1
Brown	2016-04-02 15:23:14.000	Gibson Les Paul	This Les Paul guitar offers a carved top and humb...	1199.00	359.70	2
Butt	2016-04-07 06:25:27.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar...	699.00	209.70	1
Caldarera	2016-04-20 17:41:05.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar...	699.00	209.70	1
Caudy	NULL	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar...	699.00	209.70	1
Caudy	NULL	Hofner Icon	With authentic details inspired by the original, the ...	489.99	186.20	1
Dareky	2016-04-07 08:15:55.000	Fender Stratocaster	The Fender Stratocaster is the electric guitar desig...	2517.00	1308.84	1
Dillard	2016-04-09 18:42:36.000	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar...	699.00	209.70	1
Esway	2016-04-15 12:27:35.000	Fender Stratocaster	The Fender Stratocaster is the electric guitar desig...	2517.00	1308.84	1
Esway	2016-04-06 15:00:03.000	Fender Precision	The Fender Precision bass guitar delivers the soun...	499.99	125.00	1

At the bottom of the results grid, a message says "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.)

Solution:

The below query returns one row for each category that has never been used with CategoryName column.

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

```

c5.sql - LAPTOP-PUMCMH5H\CCHIKKAN.MyGuitarShop (LAPTOP-PUMCMH5H\ Laptop-VM (52)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
MyGuitarShop Execute New Query Find Replace All Open Object Explorer
Object Explorer
Connect + MyGuitarShop
LAPTOP-PUMCMH5H\CCHIKKAN (SQL Server)
Database System Databases Database Snapshots MyGuitarShop Database Diagrams Tables Views Synonyms Programmability Service Broker Storage Security
Security Server Objects Replication Always On High Availability Management Integration Services Catalogs SQL Server Agent (Agent XPs disabled)
XEvent Profiler
100 %
Results Messages
CategoryName
1 Keyboards
Query executed successfully.
LAPTOP-PUMCMH5H\CCHIKKAN (1... LAPTOP-PUMCMH5H\ Laptop... MyGuitarShop | 00:00:00 | 1 rows
Ready Type here to search Ln 1 Col 1 Ch 1 INS
6:05 PM 11/6/2020

```

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.

Solution:

The below query is a SELECT statement that returns one row for each customer that has orders with the columns mentioned in the above question.

```

SELECT a.EmailAddress, SUM(c.ItemPrice*c.Quantity) AS SumItem,
SUM(c.DiscountAmount*c.Quantity) AS SumDiscount
FROM Customers a JOIN Orders b ON a.CustomerID=b.CustomerID
JOIN OrderItems c ON b.OrderID=c.OrderID
GROUP BY a.EmailAddress
ORDER BY SumItem DESC;

```

```

c5.sql - LAPTOP-PUMCMH5H\CCHIKKAN.MyGuitarShop (LAPTOP-PUMCMH5H\ Laptop-VM (54)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
MyGuitarShop Execute
SELECT a.EmailAddress, SUM((c.ItemPrice-c.DiscountAmount) * c.Quantity) AS SumItem, SUM(c.DiscountAmount*c.Quantity) AS SumDiscount
FROM Customers a JOIN Orders b ON a.CustomerID=b.CustomerID
JOIN OrderItems c ON b.OrderID=c.OrderID
GROUP BY a.EmailAddress
ORDER BY a.EmailAddress DESC;

```

EmailAddress	SumItem	SumDiscount
david.goldstein@hotmail.com	6395.95	1829.10
allan.sherwood@yahoo.com	4131.00	1830.39
kris@gmail.com	3316.99	1428.84
yuki_whobrey@aol.com	3216.00	1518.54
heatheresway@mac.com	3016.99	1433.84
mroyter@royster.com	2517.00	1308.84
josephine_daraky@daraky.org	2517.00	1308.84
mattie@aol.com	2517.00	1308.84
gruta@cox.net	2398.00	719.40
lpaprocki@hotmail.com	2398.00	719.40
sage_wieser@cox.net	2398.00	719.40

Query executed successfully.

4. [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
  - A count of the number of orders
  - 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.

Solution:

The below query is a SELECT statement that returns one row for each customer that has orders with the columns mentioned above.

```

SELECT a.EmailAddress,COUNT(b.OrderID) AS NoOfOrders,
SUM((c.ItemPrice-c.DiscountAmount) * c.Quantity)
AS TotalAmount
FROM Customers a JOIN Orders b ON a.CustomerID=b.CustomerID
JOIN OrderItems c ON b.OrderID=c.OrderID
WHERE ItemPrice>500
GROUP BY a.EmailAddress
ORDER BY a.EmailAddress DESC;

```

```

SELECT a.EmailAddress, COUNT(b.OrderID) AS NoOfOrders, SUM((c.ItemPrice - c.DiscountAmount) * c.Quantity) AS TotalAmount
FROM Customers a JOIN Orders b ON a.CustomerID=b.CustomerID
JOIN OrderItems c ON b.OrderID=c.OrderID
WHERE ItemPrice>500
GROUP BY a.EmailAddress
ORDER BY a.EmailAddress DESC;

```

The screenshot shows the results of the executed query:

EmailAddress	NoOfOrders	TotalAmount
yuki_whobrey@aol.com	2	1697.46
vinouye@aol.com	1	559.99
simona@morsca.com	1	489.30
sage_wieser@cox.net	1	1678.60
mroyster@royster.com	1	1208.16
mitsue_tolner@yahoo.com	1	489.30
minnie_amigon@yahoo.com	1	559.99
meaghan@hotmail.com	1	489.99
mattie@aol.com	1	1208.16
lpaprocki@hotmail.com	1	1678.60
leota@hotmail.com	1	489.30

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.

Solution:

The below query is a SELECT statement that returns three columns: EmailAddress, OrderID, and the OrderTotal for each customer by grouping EmailAddress and OrderID.

```

SELECT a.EmailAddress,b.OrderID, SUM((c.ItemPrice - c.DiscountAmount) * c.Quantity) AS OrderTotal
FROM Customers a JOIN Orders b ON a.CustomerID=b.CustomerID
JOIN OrderItems c ON b.OrderID = c.OrderID
GROUP By a.EmailAddress, b.OrderID;

```

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 is being run:

```

SELECT a.EmailAddress, b.OrderID, SUM((c.ItemPrice - c.DiscountAmount) * c.Quantity) AS OrderTotal
FROM Customers a JOIN Orders b ON a.CustomerID=b.CustomerID
JOIN OrderItems c ON b.OrderID = c.OrderID
GROUP By a.EmailAddress, b.OrderID;

```

The results pane shows the following data:

EmailAddress	OrderID	OrderTotal
alan.sherwood@yahoo.com	1	839.30
bamy2@gmail.com	2	303.79
alan.sherwood@yahoo.com	3	1461.31
christineb@olarone.com	4	1678.60
david.goldstein@hotmail.com	5	299.00
erinv@gmail.com	6	299.00
frankwilson@abcglobal.net	7	1539.97
gary_hernandez@yahoo.com	8	679.99
david.goldstein@hotmail.com	9	1467.90
heatheresway@mac.com	10	374.99
jbutt@gmail.com	11	489.30

At the bottom of the results pane, it says "Query executed successfully." The status bar at the bottom right shows the date and time: "11/6/2020 6:07 PM".

(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.

Solution:

The below query is a SELECT statement that uses the SELECT statement which is written above in its FROM clause that returns two columns i.e. customer's email address and the largest order for that customer by grouping result set by the EmailAddress column.

```

SELECT EmailAddress, MAX(OrderTotal) as LargestOrder
FROM (SELECT a.EmailAddress, b.OrderID, SUM((c.ItemPrice - c.DiscountAmount) *
c.Quantity) AS OrderTotal
FROM Customers a JOIN Orders b ON a.CustomerID=b.CustomerID
JOIN OrderItems c ON b.OrderID = c.OrderID
GROUP By a.EmailAddress, b.OrderID) as sub
GROUP BY sub.EmailAddress

```

```

SELECT a.EmailAddress, b.OrderID, SUM((c.ItemPrice - c.DiscountAmount) * c.Quantity) AS OrderTotal
FROM Customers a JOIN Orders b ON a.CustomerID=b.CustomerID
JOIN OrderItems c ON b.OrderID = c.OrderID
GROUP By a.EmailAddress, b.OrderID;

SELECT EmailAddress, MAX(OrderTotal) as LargestOrder
FROM (SELECT a.EmailAddress, b.OrderID, SUM((c.ItemPrice - c.DiscountAmount) * c.Quantity) AS OrderTotal
      FROM Customers a JOIN Orders b ON a.CustomerID=b.CustomerID
      JOIN OrderItems c ON b.OrderID = c.OrderID
      GROUP By a.EmailAddress, b.OrderID) as sub
GROUP BY sub.EmailAddress
    
```

EmailAddress	LargestOrder
alisha@elusarski.com	1678.60
allan.sherwood@yahoo.com	1451.31
alfern.tubide@cox.net	489.30
amaclead@gmail.com	489.30
art@venere.org	679.99
bamyz@gmail.com	303.79
bette_nicks@cox.net	839.30
callares@gmail.com	489.30
chanel.caudy@caudy.org	793.09
christineb@solarone.com	1678.60
david.goldstein@hotmail.com	2799.95

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.

Solution:

The below query is a correlated subquery to return one row per customer, representing the customer's newest order

```

SELECT a.EmailAddress, b.OrderID, b.OrderDate
FROM Customers a JOIN Orders b ON a.CustomerID=b.CustomerID
WHERE b.OrderDate IN (SELECT MAX(OrderDate)
FROM Orders as c
WHERE a.CustomerID = c.CustomerID);
    
```

```

c6.sql - LAPTOP-PUMCMH5H\CCHIKKAN.MyGuitarShop (LAPTOP-PUMCMH5H\Laptop-VM (51)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query Execute
MyGuitarShop dbo.
Object Explorer
Connect + MyGuitarShop c6.sql - LAPTOP-PUMCMH5H\CCHIKKAN (Laptop-VM (51))
LAPTOP-PUMCMH5H\CCHIKKAN (SQL Server)
Database System Databases Database Snapshots MyGuitarShop Database Diagrams Tables Views Synonyms Programmability Service Broker Storage Security
Security Server Objects Replication Always On High Availability Management Integration Services Catalogs SQL Server Agent (Agent XPs disabled)
XEvent Profiler
100 % Results Messages
EmailAddress OrderID OrderDate
1 chanel.caudy@caudy.org 41 2016-05-09 07:52:55.000
2 allen.sturzide@cox.net 40 2016-05-08 21:41:29.000
3 alisha@lusaski.com 38 2016-05-08 11:41:24.000
4 mroyster@royster.com 37 2016-05-06 14:15:21.000
5 willard@hotmail.com 36 2016-05-04 12:31:33.000
6 vinouye@aol.com 35 2016-05-04 03:52:23.000
7 bette_nicks@cox.net 39 2016-05-08 22:22:26.000
8 fletcher.flos@yahoo.com 33 2016-05-01 09:11:51.000
9 yuki_whobrey@aol.com 31 2016-04-29 06:47:14.000
10 glady.srm@ym.org 30 2016-04-27 16:21:31.000
11 meaghan@hotmail.com 28 2016-04-21 17:52:24.000

```

Query executed successfully.

7. [4] Write a SELECT statement that returns these columns from the Products table:
- The ListPrice column
  - A column that uses the CAST function to return the ListPrice column with 2 digits to the right of the decimal point
  - A column that uses the CAST function to return the ListPrice column as a real number
  - A column that uses the CONVERT function to return the ListPrice column as an integer.

Solution:

Below is a SELECT statement that returns the columns from the Products table as mentioned in the above question by using CAST and CONVERT functions:

```

SELECT ListPrice, CAST(ListPrice AS DECIMAL(8,2)) AS Decimal, CAST(ListPrice AS real) AS Real, CONVERT(INT,ListPrice) AS Int
FROM Products;

```

```

c6.sql - LAPTOP-PUMCMH5H\CCHIKKAN.MyGuitarShop (LAPTOP-PUMCMH5H\ Laptop-VM (52)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
MyGuitarShop Execute New Query Object Explorer
Object Explorer
Connect + MyGuitarShop
LAPTOP-PUMCMH5H\CCHIKKAN (SQL
Database System Databases Database Snapshots MyGuitarShop Database Diagrams Tables Views Synonyms Programmability Service Broker Storage Security
Security Server Objects Replication Always On High Availability Management Integration Services Catalogs SQL Server Agent (Agent XPs disabled)
XEvent Profiler
Results Messages
ListPrice Decimal Real Int
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

```

Query executed successfully.

8. [4] Write a SELECT statement that returns these columns from the Orders table:
- The CardNumber column
  - The length of the CardNumber column
  - The last six digits of the CardNumber column
  - 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.

Solution:

The below is a select statement that returns the following four columns from the Orders table as mentioned in the above question.

```

SELECT CardNumber, LEN(CardNumber) AS CardNoLength, RIGHT(CardNumber,6)
AS Last6, ('XXXX-XXXX-XX'+SUBSTRING(CardNumber,11,2)+-
'+RIGHT(CardNumber,4)) AS CardNo
FROM Orders;

```

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a connection to 'LAPTOP-PUMCMH5H\CCHIKKAN' is selected, and the 'MyGuitarShop' database is expanded to show its tables, views, and other objects. In the center pane, there is a results grid for a query. The query is:

```
c6.sql - LAPTOP-P..5H\ Laptop-VM (51)* c11.sql - LAPTOP-P..5H\ Laptop-VM (57)* c7.sql - LAPTOP-P..5H\ Laptop-VM (52)* c8.sql - LAPTOP-P..5H\ Laptop-VM (54)*
SELECT CardNumber, LEN(CardNumber) AS CardNoLength, RIGHT(CardNumber, 6) AS Last6, ('XXXX-XXXX-XX'+SUBSTRING(CardNumber, 11, 2)+ '-' +RIGHT(CardNumber, 4)) AS CardNo
FROM Orders;
```

The results grid displays 11 rows of data from the Orders table, showing columns: CardNumber, CardNoLength, Last6, and CardNo. The data is as follows:

	CardNumber	CardNoLength	Last6	CardNo
1	4111111111111111	16	111111	XXXX-XXXX-XX11-1111
2	401288888881881	16	881881	XXXX-XXXX-XX68-1881
3	4111111111111111	16	111111	XXXX-XXXX-XX11-1111
4	378282463100005	16	100005	XXXX-XXXX-XX10-0005
5	4111111111111111	16	111111	XXXX-XXXX-XX11-1111
6	6011111111111117	16	111117	XXXX-XXXX-XX11-1117
7	5555555555554444	16	554444	XXXX-XXXX-XX55-4444
8	401288888881881	16	881881	XXXX-XXXX-XX68-1881
9	4111111111111111	16	111111	XXXX-XXXX-XX11-1111
10	4111111111111111	16	111111	XXXX-XXXX-XX11-1111
11	401288888881881	16	881881	XXXX-XXXX-XX68-1881

At the bottom of the results grid, it says 'Query executed successfully.'

9. [4] Write a SELECT statement that returns these columns from the Orders table:
- The OrderID column
  - The OrderDate column
  - A column named ApproxShipDate that's calculated by adding 1 month to the OrderDate column
  - The ShipDate column
  - 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.

Solution:

The below query is a SELECT statement that returns the columns from the Orders table as mentioned in the above question

```
SELECT OrderID, OrderDate, dateadd(month, 1, OrderDate) as ApproxShipDate,
ShipDate, datediff(day, OrderDate, ShipDate) as DaysToShip
FROM Orders
where year(OrderDate) = 2016 and month(OrderDate) = 4;
```

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'MyGuitarShop' is selected. In the center pane, a query window displays the following SQL code:

```

SELECT OrderID, OrderDate, datediff(month, 1, OrderDate) as ApproxShipDate, ShipDate, datediff(day, OrderDate, ShipDate) as DaysToShip
FROM Orders
WHERE year(OrderDate) = 2016 and month(OrderDate) = 4;

```

The Results tab shows the output of the query:

OrderID	OrderDate	ApproxShipDate	ShipDate	DaysToShip
1	2016-04-01 23:11:12.000	2016-05-01 23:11:12.000	2016-04-04 23:11:55.000	3
2	2016-04-02 11:26:38.000	2016-05-02 11:26:38.000	2016-04-05 11:27:21.000	3
3	2016-04-03 12:22:31.000	2016-05-03 12:22:31.000	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	2016-04-08 09:25:36.000	3
9	2016-04-05 14:52:17.000	2016-05-05 14:52:17.000	2016-04-08 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

The status bar at the bottom indicates "Query executed successfully." and shows the date and time as 11/6/2020 6:10 PM.

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.

Solution:

The below query is an INSERT INTO statement that adds a row into the table with the values mentioned above in the question.

```

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

```

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 displays the following SQL code:

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

Below the code, the 'Messages' tab shows the output:

```
(1 row affected)
Completion time: 2020-11-06T18:10:48.6036988-05:00
```

In the status bar at the bottom, it says 'Query executed successfully.' and shows the date and time as 11/6/2020.

The below SELECT statement verifies whether a new row was inserted into the table or not.

**SELECT \* FROM Customers  
WHERE EmailAddress='kriegerrobert@gmail.com';**

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 displays the following SQL code:

```
SELECT * FROM Customers
WHERE EmailAddress='kriegerrobert@gmail.com';
```

Below the code, the 'Results' tab shows the output:

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

In the status bar at the bottom, it says 'Query executed successfully.' and shows the date and time as 11/6/2020.

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.

Solution:

In this question, we have to perform update operation of password column. Initially we have some other value in the password column for “kriegerrobert@gmail.com”. We can see this in the result screenshot which was obtained below.

**SELECT \* FROM Customers WHERE EmailAddress = 'kriegerrobert@gmail.com';**

Update operation is performed by using the following query. It updates the password column to “secret@123” for the new customer.

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

The below SELECT statement verifies the result.

**SELECT \* FROM Customers WHERE EmailAddress = 'kriegerrobert@gmail.com';**

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the MyGuitarShop database. The main pane contains two queries. The top query is:

```
SELECT * FROM Customers WHERE EmailAddress = 'Kriegerrobert@gmail.com';
```

The bottom query is:

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

SELECT * FROM Customers WHERE EmailAddress = 'kriegerrobert@gmail.com';
```

The results pane shows two rows of data. The first row is from the initial SELECT query, and the second row is from the updated table after executing the UPDATE statement. Both rows have the same values: CustomerID 486, EmailAddress kriegerrobert@gmail.com, Password secret@1234, FirstName Krieger, LastName Robert, ShippingAddressID NULL, and BillingAddressID NULL.

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

At the bottom of the interface, a message says "Query executed successfully." and shows the status bar indicating "LAPTOP-PUMCMH5H\CCHIKKAN (1... : LAPTOP-PUMCMH5H\Laptop... : MyGuitarShop | 00:00:00 | 2 rows".

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

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.

Solution:

The below is a CREATE VIEW statement that creates a view named OrderItemProductsDetails that returns the columns mentioned in the above question from the Orders, OrderItems, and Products tables.

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

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

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

Below the code, the 'Messages' pane shows the output:

```
Commands completed successfully.
Completion time: 2020-11-06T18:22:58.9876789-05:00
```

At the bottom of the screen, the taskbar shows the Windows Start button, a search bar, and several pinned icons. The system tray indicates the date as 11/6/2020 and the time as 6:23 PM.

The below is a SELECT statement written to verify the weather the view contains all the information mentioned in the question. So, executing the select statement below returns all the information contained in the view.

**SELECT \* FROM OrderItemProductsDetails;**

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'MyGuitarShop'. The main pane displays a T-SQL script for creating a view named 'OrderItemProductsDetails' and then executing a SELECT statement against it. The results pane shows a table with 47 rows of data, including columns like OrderID, OrderDate, TaxAmount, ShipDate, ItemPrice, DiscountAmount, FinalPrice, Quantity, ItemTotal, ProductName, and Description. The status bar at the bottom indicates the query was executed successfully at 6:23 PM on 11/6/2020.

```

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

SELECT * FROM OrderItemProductsDetails;

```

OrderID	OrderDate	TaxAmount	ShipDate	ItemPrice	DiscountAmount	FinalPrice	Quantity	ItemTotal	ProductName	Description
1	2016-03-28 09:40:28.000	58.75	2016-03-31 09:41:11.000	1199.00	359.70	839.30	1	839.30	Gibson Les Paul	This Les Paul guitar o
2	2016-03-28 11:23:20.000	21.27	2016-03-31 11:24:03.000	489.99	186.20	303.79	1	303.79	Hofner Icon	With authentic details
3	2016-03-29 09:44:58.000	102.29	2016-04-01 09:45:41.000	2517.00	1308.84	1208.16	1	1208.16	Fender Stratocaster	The Fender Stratocat
4	2016-03-29 09:44:58.000	102.29	2016-04-01 09:45:41.000	415.00	161.85	253.15	1	253.15	Ludwig 5piece Drum Set with Cymbals	This product includes
5	2016-03-30 15:22:31.000	117.50	2016-04-02 15:23:14.000	1199.00	359.70	839.30	2	1678.60	Gibson Les Paul	This Les Paul guitar o
6	2016-03-31 05:43:11.000	20.93	2016-04-03 05:43:54.000	299.00	0.00	299.00	1	299.00	Tama 5-Piece Drum Set with Cymbals	The Tama 5-piece Dr
7	2016-03-31 18:37:22.000	20.93	2016-04-03 18:38:05.000	299.00	0.00	299.00	1	299.00	Tama 5-Piece Drum Set with Cymbals	The Tama 5-piece Dr
8	2016-04-01 23:11:12.000	107.80	2016-04-04 23:11:55.000	699.99	210.00	489.99	1	489.99	Washburn D10S	The Washburn D10S
9	2016-04-01 23:11:12.000	107.80	2016-04-04 23:11:55.000	799.99	240.00	559.99	1	559.99	Gibson SG	This Gibson SG elect
10	2016-04-01 23:11:12.000	107.80	2016-04-04 23:11:55.000	699.99	210.00	489.99	1	489.99	Washburn D10S	The Washburn D10S

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).

Solution:

The below is a CREATE VIEW statement that creates a view named Top5BestSelling that uses the view (OrderItemProductsDetails) which was created in the previous statement and returns the information about the 5 best-selling products.

```

CREATE VIEW Top5BestSelling AS
SELECT TOP 5 ProductName, SUM(Quantity*ItemPrice) AS OrderTotal,
COUNT(Quantity) AS OrderCount
FROM OrderItemProductsDetails
GROUP BY ProductName
ORDER BY OrderCount DESC;

```

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, under the 'MyGuitarShop' database, a new view named 'Top5BestSelling' is being created. The script pane contains the following T-SQL code:

```

CREATE VIEW Top5BestSelling AS
SELECT TOP 5 ProductName, SUM(Quantity*ItemPrice) AS OrderTotal, COUNT(Quantity) AS OrderCount
FROM OrderItemProductsDetails
GROUP BY ProductName
ORDER BY OrderCount DESC;
  
```

The 'Messages' pane at the bottom right shows the command completed successfully with a completion time of 2020-11-06T18:24:06.0311889-05:00.

Below is the SELECT statement written to verify the weather the view contains all the information mentioned in the question. Executing the select statement below returns all the information contained in the view.

**SELECT \* FROM Top5BestSelling;**

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, under the 'MyGuitarShop' database, the previously created view 'Top5BestSelling' is selected. The script pane contains the following T-SQL code:

```

SELECT * FROM Top5BestSelling
  
```

The results pane displays the data returned by the query:

ProductName	OrderTotal	OrderCount
Rodriguez Caballero 11	9786.00	12
Gibson Les Paul	14388.00	7
Fender Stratocaster	17619.00	7
Gibson SG	7199.91	5
Hofner Icon	1469.97	3

The 'Messages' pane at the bottom right shows the query executed successfully with a completion time of 2020-11-06T18:24:06.0311889-05:00.

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.

Solution:

The below script creates and calls a stored procedure named spUpdateProductDiscount that updates the DiscountPercent column in the Products table.

The procedure contains a single parameter for the product ID and another for the discount percent. If DiscountPercent column is a negative number, the procedure raises an error.

```
CREATE PROCEDURE spUpdateProductDiscount
(
    @ProductID int,
    @DiscountPercent int
)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        UPDATE Products set DiscountPercent = @DiscountPercent where
        ProductID = @ProductID;
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @DiscountPercent < 0
            ROLLBACK TRANSACTION;
        PRINT 'DiscountPercent column cant have negative values';
    END CATCH
END
GO
```

```

CREATE PROCEDURE spUpdateProductDiscount
(
    @ProductID int,
    @DiscountPercent int
)
AS
BEGIN TRY
    BEGIN TRANSACTION;
    UPDATE Products SET DiscountPercent = @DiscountPercent WHERE ProductID = @ProductID;
    COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    IF @DiscountPercent < 0
        ROLLBACK TRANSACTION;
    PRINT 'DiscountPercent column cant have negative values';
END CATCH

```

Completion time: 2020-11-06T18:25:18.6662283-05:00

Messages

Commands completed successfully.

The SELECT statement verifies whether the view contains all the information mentioned in the question. Executing the select statement below returns all the information contained in the view.

### SELECT \* FROM Products;

```

SELECT * FROM Products;

```

ProductID	CategoryID	ProductCode	ProductName	Description	ListPrice	DiscountPercent	DateAdded
1	1	strat	Fender Stratocaster	The Fender Stratocaster is the electric guitar design...	699.00	30.00	2015-10-30 09:32:40.000
2	2	les_paul	Gibson Les Paul	This Les Paul guitar offers a carved top and humbu...	1199.00	30.00	2015-12-05 16:33:13.000
3	3	sg	Gibson SG	This Gibson SG electric guitar takes the best of the...	2517.00	52.00	2016-02-04 11:04:31.000
4	4	fg700s	Yamaha FG700S	The Yamaha FG700S solid top acoustic guitar has ...	489.99	38.00	2016-06-01 11:12:59.000
5	5	washburn	Washburn D10S	The Washburn D10S acoustic guitar is superbly cr...	299.00	0.00	2016-07-30 13:58:35.000
6	6	rodriguez	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar ...	415.00	39.00	2016-07-30 14:12:41.000
7	7	precision	Fender Precision	The Fender Precision bass guitar delivers the soun...	799.99	30.00	2016-06-01 11:29:35.000
8	8	hofner	Hofner Icon	With authentic details inspired by the original, the H...	499.99	25.00	2016-07-30 14:18:33.000
9	9	ludwig	Ludwig 5-piece Drum Set with Cymbals	This product includes a Ludwig 5-piece drum set a...	699.99	30.00	2016-07-30 12:46:40.000
10	10	tama	Tama 5-piece Drum Set with Cymbals	The Tama 5-piece Drum Set is the most affordable...	799.99	15.00	2016-07-30 13:14:15.000

Results

Messages

Query executed successfully.

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
```

Solution:

The below script calculates the common factors between 15 and 30

```
DECLARE @a INT;  
DECLARE @b INT;  
DECLARE @x INT;  
DECLARE @final varchar(100);  
SET @a = 15;  
SET @b = 30;  
SET @x = 1;  
SET @final = 'Common Factors of 15 and 30' + CHAR(13);  
WHILE(@x < @a)  
BEGIN  
    IF(@a % @x = 0 AND @a % @x = 0)  
        SET @final = CONCAT (@final,CHAR(13),@x,CHAR(13));  
    SET @x+=1;  
END  
SELECT @final;
```

```

d4.sql - LAPTOP-PUMCMH5H\CCHIKKAN.MyGuitarShop (LAPTOP-PUMCMH5H\ Laptop-VM (55)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
MyGuitarShop Execute ✓
Object Explorer
d4.sql - LAPTOP-PUMCMH5H\CCHIKKAN (SQL Server)
  Database
    System Databases
    Database Snapshots
    MyGuitarShop
      Database Diagrams
      Tables
      Views
      System Views
        dbo.OrderItemProductsC
        dbo.TopBestSelling
      Synonyms
      Programmability
      Service Broker
      Storage
      Security
    Security
    Server Objects
    Replication
    Always On High Availability
    Management
    Integration Services Catalogs
    SQL Server Agent (Agent XPs disabled)
    XEvent Profiler
  Results Messages
  (No column name)
  1 | Common Factors of 15 and 30 1 3 5 |
  Query executed successfully.
  LAPTOP-PUMCMH5H\CCHIKKAN (1... LAPTOP-PUMCMH5H\ Laptop... MyGuitarShop 00:00:00 1 rows
  Ready Type here to search Ln 13 Col 9 Ch 3 INS
  6:38 PM 11/6/2020

```

5. [5] (1) Write a script that creates and calls a function named fnDiscountPrice that calculates the discount price of an item in the OrderItems table (discount amount subtracted from item price). To do that, this function should accept one parameter for the item ID, and it should return the value of the discount price for that item.
- (2) Write a script that creates and calls a function named fnItemTotal that calculates the total amount of an item in the OrderItems table (discount price multiplied by quantity). To do that, this function should accept one parameter for the item ID, it should use the DiscountPrice function that you created in (1), and it should return the value of the total for that item.

Solution:

The script creates and calls a function named fnDiscountPrice that calculates the discount price of an item in the OrderItems table. The function accepts a parameter for itemID and it returns the value of the discount price for that item.

```

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

```

```

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

```

Messages

Discount price = 1208  
Completion time: 2020-11-06T19:04:26.8673536-05:00

Query executed successfully.

The script creates and calls a function named fnItemTotal that calculates the total amount of an item in the OrderItems table. The function here accepts a parameter for itemID and it uses the DiscountPrice function that was created above and it returns the value of the discount price for that item.

```

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

```

```

END;
GO
PRINT 'Discount price = ' + CONVERT(varchar, dbo.fnDiscountPrice(3),1);

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

```

Messages

Total Amount = 299.00  
Completion time: 2020-11-06T19:47:53.7789407-05:00

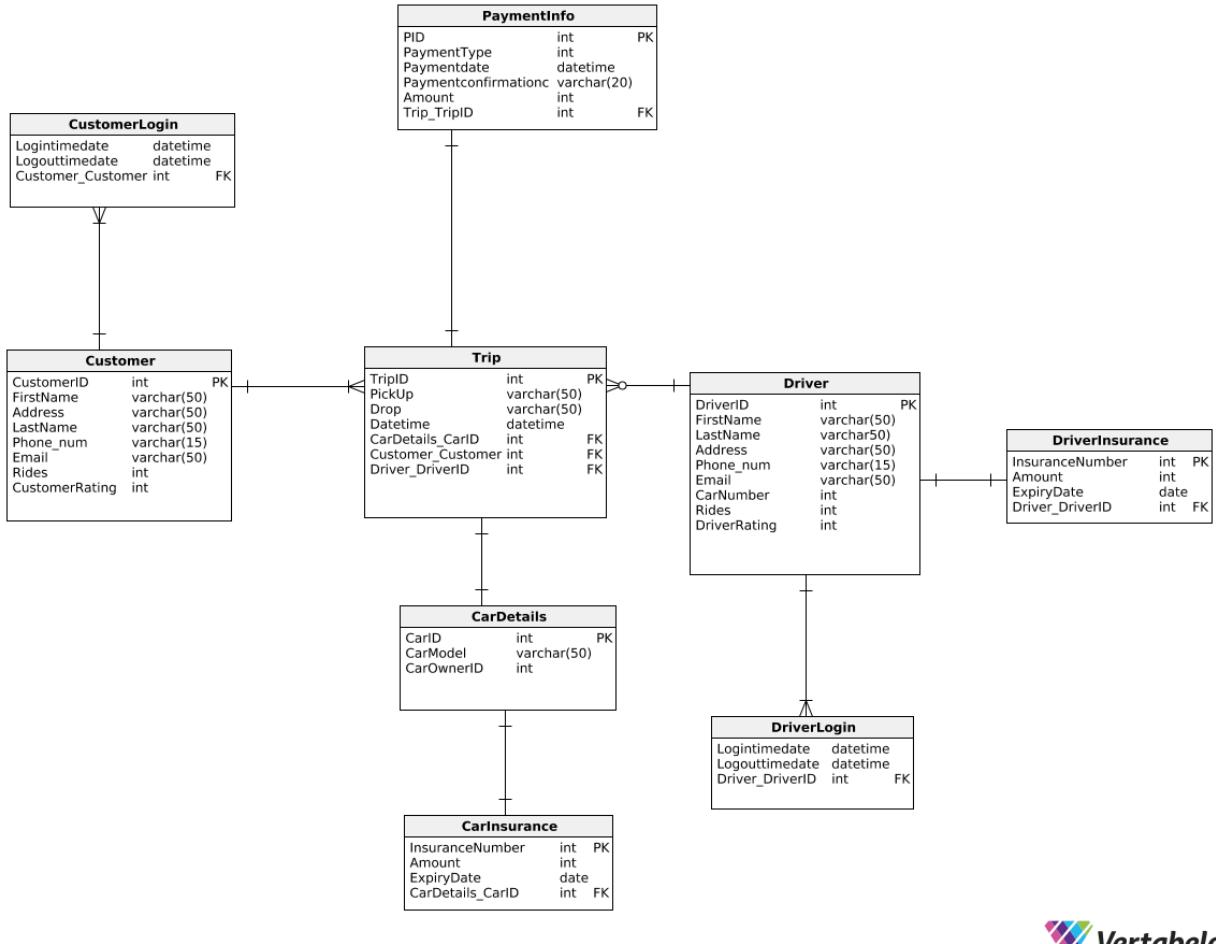
Query executed successfully.

## 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.

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



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

In this database design, I have included tables like Customer, Driver, Trip, CustomerLogin, DriverLogin, PaymentInfo, CarDetails, CarInsurance and DriverInsurance.

### Customer:

- This table contains the customer details. The columns in the table are:
- CustomerID, FirstName, LastName, Address, Phone\_num, Email, Rides and CustomerRating.
- CustomerID is the primary key in the table.
- CustomerRating column can be a nullable value when the driver chose not to give any ratings for a particular customer.

### Driver:

- This table contains the driver details. The columns are:
- DriverID, FirstName, LastName, Address, Phone\_num, Email, CarNumber, Rides and DriverRating

- DriverID is the primary key in the table.
- DriverRating column can be nullable when a customer chose not to give rating to the driver.

**Trip:**

- This table contains the information regarding the trip a driver and customer completed. The columns include:
- TripID, PickUp, Drop, Datetime, Customer\_CustomerID, Driver\_DriverID, CarDetails\_CarID.
- TripID is the primary key and Customer\_CustomerID, Driver\_DriverID, arDetails\_CarID are the foreign keys in the table.
- No nullable columns

**PaymentInfo:**

- This table contains the payment details of a completed trip. The columns are:
- PaymentID, PaymentType, PaymentDate, Paymentconfirmationcode, Amount, Trip\_TripID.
- PaymentID is the primary key and Trip\_TripID is the foreign key.
- No nullable column.

**CustomerLogin:**

- This table maintains the customer login details. The columns are:
- Logintimedate, logouttimedate, Customer\_CustomerID.
- Customer\_CustomerID is the foreign key.
- No nullable column

**DriverLogin:**

- This table maintains the customer login details. The columns are:
- Logintimedate, logouttimedate, Driver\_DriverID.
- Driver\_DriverID is the foreign key.
- No nullable column

**CarDetails:**

- This table contains the car details where the columns include:
- CarID, CarModel, CarOwnerID.
- CarID is the primary key

**CarInsurance:**

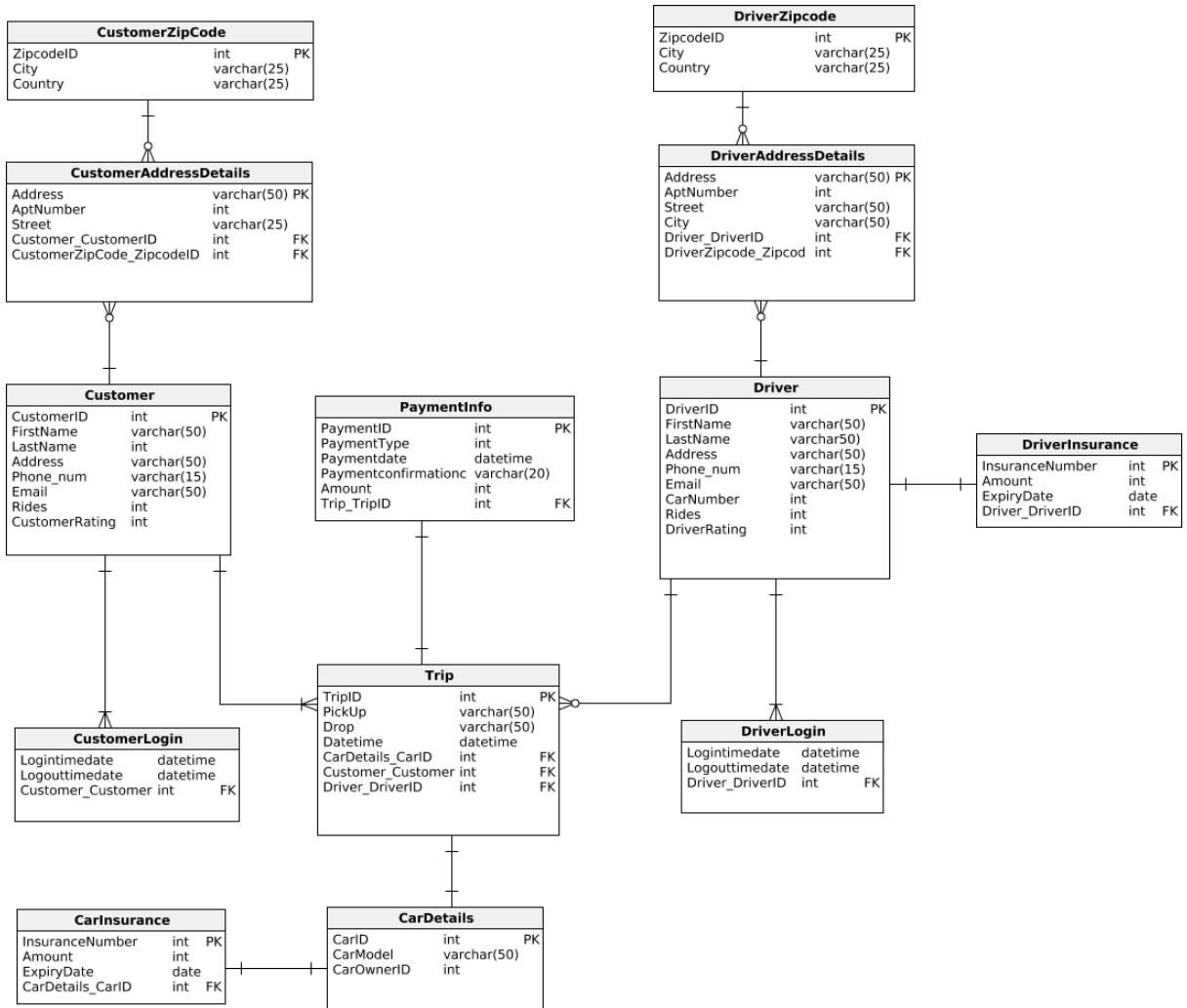
- This table contains the car insurance details with columns:
- InsuranceNumber, Amount, Expirydate, CarDetails\_CarID.
- InsuranceNumber is the primary key and CarDetails\_CarID is the foreign key.

**DriverInsurance:**

- This table contains the car insurance details with columns:
- InsuranceNumber, Amount, Expirydate, Driver\_DriverID.
- InsuranceNumber is the primary key and Driver\_DriverID is the foreign key.

**Relationships:**

- There is one-to-many relationship between Customer and Trip tables.
  - There is one-to-many relationship between Driver and Trip tables.
  - There is one-to-many relationship between Customer and CustomerLogin tables.
  - There is one-to-many relationship between Driver and DriverLogin tables.
  - There is one-to-one relationship between PaymentInfo and Trip tables.
  - There is one-to-one relationship between CarDetails and CarInsurance
  - There is one-to-one relationship between DriverDetails and DriverInsurance
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 the above design, I have added 4 more tables to normalize the database they are:

##### **CustomerAddressDetails:**

- This table contains the customer address details with 5 columns
- Address is the primary key and Customer\_CustomerID, CustomerZipcode\_ZipcodeID are foreign key.

##### **DriverAddressDetails:**

- This table contains the driver address details with 5 columns
- Address is the primary key and Driver\_DriverID, CustomerZipcode\_ZipcodeID are foreign key.

##### **CustomerZipCode:**

- ZipcodeID is the primary key.

**DriverZipCode:**

- ZipcodeID is the primary key.

**Relationships:**

- There is one-to-many relationship between Customer and CustomerAddressDetails
- There is one-to-many relationship between Customer and CustomerAddressDetails
- There is one-to-many relationship between CustomerAddressDetails and CustomerZipcode
- There is one-to-many relationship between DriverAddressDetails and DriverZipcode

For any table to be in SecondNormalForm, it should be in 1-NF and all non-key attributes must depend on the entire primary key but not on the partial partial primary key. In Customer and Driver table, Address column can be created as a separate table which contains further more information. The CustomerAddressDetails contain AddressID as primary key which is foreign key in Customer table and same in Driver and DriverAddressDetails table.

For any table to be in ThirdNormalForm, it should be in 1-NF, 2-NF and every column should depend only on the primary key and should not depend on the non-primary key. So here in my design ZipCode and CityName columns in a AddressDetails Table are dependent on each other. For example if ZipCode changes CityName value changes and also Some set of cities have same ZipCode. So we have to eliminate this. This can be eliminated by moving the ZipCode and CityName columns into a new table named ZipCode. These values in AddressDetails tables are replaced by primary code of ZipCode table i.e. by ZipCodeID which acts as foreign key to AddressDetails table.

**Remarks:**

By completing this project, I got user experience on various skills like Creating a new database, adding tables to it, assigning various datatypes each column by defining the size and updating and deleting the previously entered data.

Additional to creating and modifying the existing tables I even learnt how to output the user selected data from the database by using SELECT statements.

I also got to know the various operations performed on the data such as arithmetic, logical and analytical operations and I got a good hold on using JOINS in SQL to in two or more tables to return different columns from more than one table simultaneously.

Creating views are important in DBMS to manage the result set of large data and I learnt how to create a view which can be used as temporary tables and also writing scripts that perform various operations simultaneously by accepting a single input parameter and output the data after executing the script.

In the last part of the project, I learnt how to design a database and how to eliminate redundancy and various kind of dependencies like partial dependency and transitive dependency and how

to convert my database design into various normal forms. By creating a database, I even analyzed the relationships between the tables and the importance of primary and foreign key.

Finally, completing this project gave me knowledge on creating a database on my own and also performing various data manipulations on that.