

## I. My Guitar Shop Database

### A. Database Setup

- (1) Download CreateMyGuitarShop.sql from Project 1 directory on Blackboard and open it in SQL server management studio. Execute the entire script and show the message in the Message tab, indicating the script is executed successfully. A complete screenshot of execution result is required.

Sol:

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

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar reads "CreateMyGuitarShop (3).sql - DESKTOP-A19REFF\SQLEXPRESSS.MyGuitarShop (sa (63)) - Microsoft SQL Server Management Studio". The Object Explorer pane on the left shows a connection to "DESKTOP-A19REFF\SQLEXPRESSS (SQL Server)" with nodes for Databases, Security, Server Objects, Replication, PolyBase, Management, and XEvent Profiler. The central pane displays the SQL script "CreateMyGuitarShop (sa (63))". The script creates a database named "MyGuitarShop" if it does not already exist. The execution results in the Messages pane show multiple rows affected (4, 10, 485, 512, 41, 47) and a final message "Query executed successfully." at the bottom. The status bar at the bottom right shows the date and time: "4/1/2020 8:23 PM".

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

Sol: This is the column definition for Address Table:

Comments: Here is the Column definition for Address Table

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar indicates the connection is to DESKTOP-A19REFFV-p, database MyGuitarShop, and the current table is dbo.Addresses. The Object Explorer on the left shows the database structure, including tables like AP, book, Books, Examples, and MyGuitarShop. The main window displays the Table Designer for the Addresses table. The table structure is as follows:

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 pane shows the following settings for the AddressID column:

- (General)**: Name = AddressID, Allow Nulls = No, Data Type = int.
- Table Designer**: Collation = <database default>.

Comments : This is the column definition for Administrator table

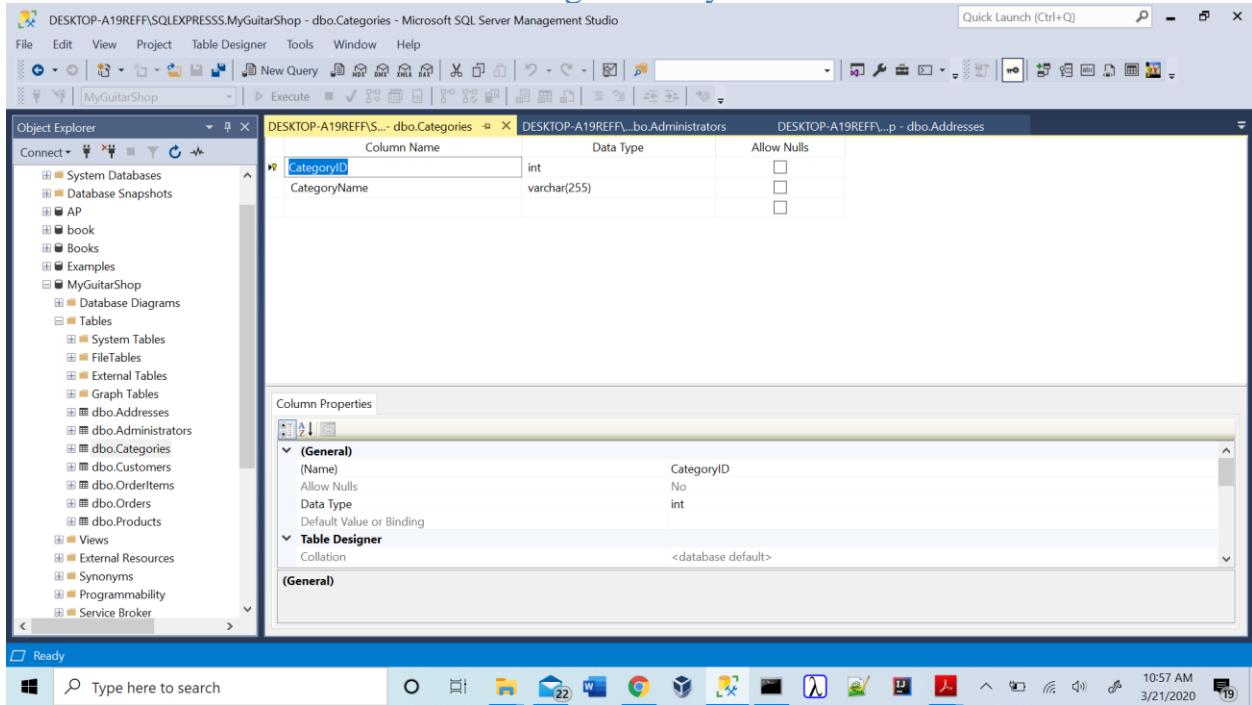
The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar indicates the connection is to DESKTOP-A19REFFV-p, database MyGuitarShop, and the current table is dbo.Administrators. The Object Explorer on the left shows the database structure, including tables like AP, book, Books, Examples, and MyGuitarShop. The main window displays the Table Designer for the Administrators table. The table structure is as follows:

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"/>

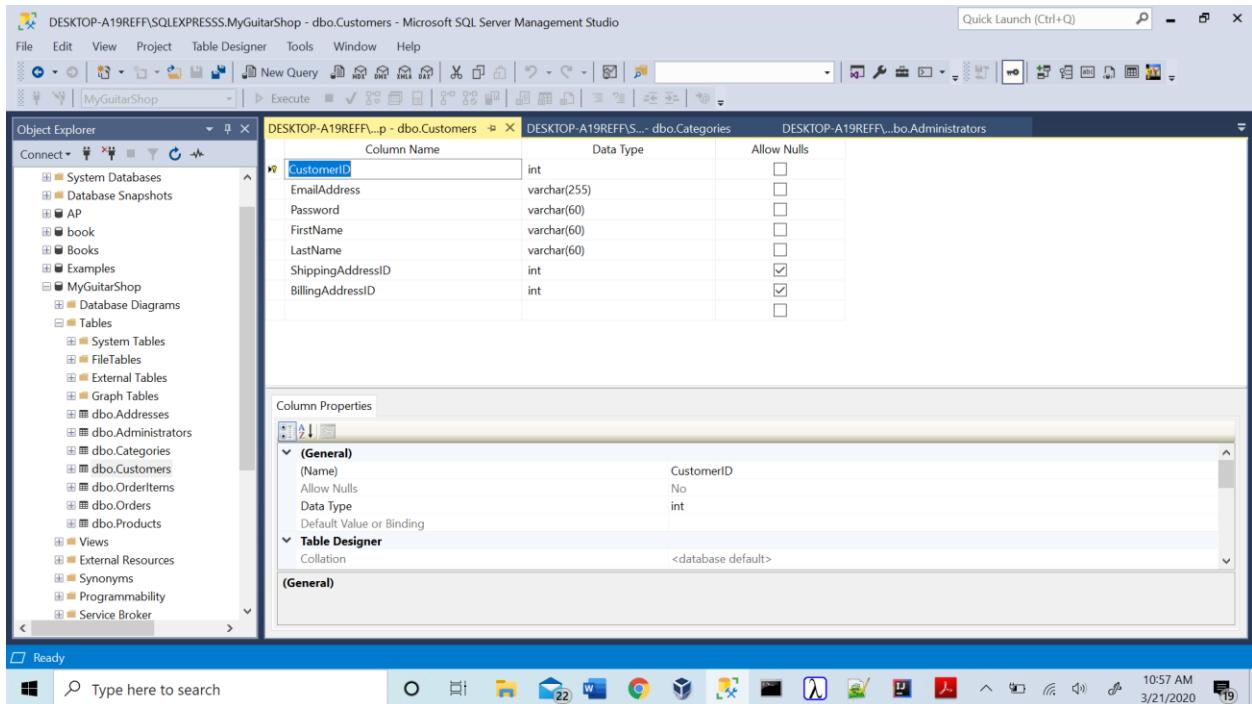
The Column Properties pane shows the following settings for the AdminID column:

- (General)**: Name = AdminID, Allow Nulls = No, Data Type = int.
- Table Designer**: Collation = <database default>.

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



Comments : Below is the Column Definition screenshot for Customers Table



Comments :Below is the Column Definition Screenshot for OrderItems Table

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar indicates the connection is to DESKTOP-A19REFF\SQLEXPRESSS.MyGuitarShop - dbo.OrderItems. The Object Explorer on the left shows the database structure, including System Databases, Database Snapshots, AP, book, Books, Examples, MyGuitarShop, Tables, System Tables, FileTables, External Tables, Graph Tables, dbo.Addresses, dbo.Administrators, dbo.Categories, dbo.Customers, dbo.OrderItems, dbo.Orders, dbo.Products, Views, External Resources, Synonyms, Programmability, and Service Broker.

The main window displays the column definition for the OrderItems table. The columns are:

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"/>

The Column Properties pane shows the following details for the ItemID column:

- (General)**: (Name) ItemID, Allow Nulls No, Data Type int, Default Value or Binding <database default>
- Table Designer**: Collation <database default>

Comments : Below is the Column Definition Screenshot for Order Table

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar indicates the connection is to DESKTOP-A19REFF\SQLEXPRESSS.MyGuitarShop - dbo.Orders. The Object Explorer on the left shows the database structure, including System Databases, Database Snapshots, AP, book, Books, Examples, MyGuitarShop, Tables, System Tables, FileTables, External Tables, Graph Tables, dbo.Addresses, dbo.Administrators, dbo.Categories, dbo.Customers, dbo.OrderItems, dbo.Orders, dbo.Products, Views, External Resources, Synonyms, Programmability, and Service Broker.

The main window displays the column definition for the Orders table. The columns are:

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"/>

The Column Properties pane shows the following details for the OrderID column:

- (General)**: (Name) OrderID, Allow Nulls No, Data Type int, Default Value or Binding <database default>
- Table Designer**: Collation <database default>

Comments: Below is the Column Definition screenshot for Products Table

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including tables like Products, Categories, Customers, and Orders. The main area displays the Table Designer for the 'Products' table. The 'Columns' tab shows columns such as ProductID (int), CategoryID (int), ProductCode (varchar(10)), ProductName (varchar(255)), Description (text), ListPrice (money), DiscountPercent (money), and DateAdded (datetime). The 'Allow Nulls' column indicates whether null values are allowed for each column. The 'Column Properties' pane on the right provides detailed settings for the ProductID column, such as its name, data type (int), and that it cannot be null. The status bar at the bottom shows the date and time as 3/21/2020 10:57 AM.

Query : select \* from MyGuitarShop.dbo.Products;

Comments :This query shows all details in Products table

The screenshot shows the Microsoft SQL Server Management Studio interface with a query window open. The query being run is 'select \* from MyGuitarShop.dbo.Products;'. The results grid displays 10 rows of data from the Products table, including columns like ProductID, CategoryID, ProductCode, ProductName, Description, ListPrice, DiscountPercent, and DateAdded. The status bar at the bottom indicates the query was executed successfully and returned 10 rows.

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	1	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	1	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	1	fg700s	Yamaha FG700S	The Yamaha FG700S solid top acoustic guitar has...	489.99	38.00	2016-06-01 11:12:59.000
5	1	washburn	Washburn D10S	The Washburn D10S acoustic guitar is superbly cra...	299.00	0.00	2016-07-30 13:58:35.000
6	1	rodriguez	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar...	415.00	39.00	2016-07-30 14:12:41.000
7	2	precision	Fender Precision	The Fender Precision bass guitar delivers the sound...	799.99	30.00	2016-06-01 11:29:35.000
8	2	hofner	Hofner Icon	With authentic details inspired by the original, the H...	499.99	25.00	2016-07-30 14:18:33.000
9	3	ludwig	Ludwig 5-piece Drum Set with Cymbals	This product includes a Ludwig 5-piece drum set an...	699.99	30.00	2016-07-30 12:46:40.000
10	3	tama	Tama 5-Piece Drum Set with Cymbals	The Tama 5-piece Drum Set is the most affordab...	799.99	15.00	2016-07-30 13:14:15.000

Query : select \* from MyGuitarShop.dbo.Categories;

Comments :This query shows all details in Categories table

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including databases, tables, and other objects. The central pane shows a query window with the following content:

```
SQLQuery4.sql - DESKTOP-A19REFF\SQLEXPRESSS.MyGuitarShop (sa (51))* - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
MyGuitarShop Execute ✓
Object Explorer
Connect ▾
DESKTOP-A19REFF\SQLEXPRESSS (SC)
  Databases
    System Databases
    Database Snapshots
    AP
    book
    Books
    Examples
    MyGuitarShop
      Database Diagrams
      Tables
        System Tables
        FileTables
        External Tables
        Graph Tables
        dbo.Addresses
        dbo.Administrators
        dbo.Categories
        dbo.Customers
        dbo.OrderItems
        dbo.Orders
        dbo.Products
      Views
      External Resources
      Synonyms
  Views
  External Resources
  Synonyms
SQLQuery4.sql - DE...uitarShop (sa (51))*
--Venkata Sai Pawan Komaravolu
SELECT * FROM MyGuitarShop.dbo.Categories;
--
```

The Results pane displays the following data:

CategoryID	CategoryName
1	Basses
2	Drums
3	Guitars
4	Keyboards

Message at the bottom: "Query executed successfully."

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

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

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

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

SQLQuery4.sql - DESKTOP-A19REFF\SQLEXPRESSS.MyGuitarShop (sa (51))\* - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query MDW DMW XML DAX Execute

Object Explorer

MyGuitarShop

SELECT (State+' '+ZipCode) AS Address  
FROM MyGuitarShop.dbo.Addresses  
WHERE State LIKE '[A-F]%'  
--Venkata Sai Pawan Komaravolu

Results Messages

Address
1 CA, 94110
2 CA, 94129
3 CA, 93711
4 CO, 80208
5 CA, 90023
6 CA, 90024
7 CA, 92801
8 AK, 99501
9 AK, 99515
10 AK, 99501
11 AK, 99501
12 CA, 97706

Query executed successfully.

DESKTOP-A19REFF\SQLEXPRESSS... sa (51) MyGuitarShop 00:00:00 136 rows

Ready Type here to search

Ln 5 Col 31 Ch 31 INS

8:26 PM 4/1/2020

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

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

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

```

SELECT OrderID, OrderDate, ShipDate
FROM MyGuitarShop.dbo.Orders
WHERE ShipDate IS NULL;
--Venkata Sai Pawan Komaravolu

```

OrderID	OrderDate	ShipDate
32	2016-05-01 01:23:23.000	NULL
38	2016-05-08 11:41:24.000	NULL
39	2016-05-08 22:22:26.000	NULL
40	2016-05-08 21:41:29.000	NULL
41	2016-05-09 07:52:55.000	NULL

Query executed successfully.

## C The essential SQL skills

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

Sol: Souce Code:

```

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

```

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

```

SELECT x.FirstName, y.OrderDate, w.ProductName,
z.ItemPrice, z.DiscountAmount, z.Quantity
FROM MyGuitarShop.dbo.Customers AS x JOIN MyGuitarShop.dbo.Orders AS y
ON x.CustomerID=y.CustomerID JOIN MyGuitarShop.dbo.OrderItems AS z
ON y.OrderID=z.OrderID JOIN MyGuitarShop.dbo.Products AS w
ON z.ProductID=w.ProductID
ORDER BY x.FirstName DESC, y.OrderDate, w.ProductName;
--Venkata Sai Pawan Komaravolu

```

	FirstName	OrderDate	ProductName	ItemPrice	DiscountAmount	Quantity
1	Yuki	2016-04-29 06:47:14.000	Fender Stratocaster	2517.00	1308.84	1
2	Yuki	2016-04-29 06:47:14.000	Rodriguez Caballero 11	699.00	209.70	1
3	Willard	2016-05-04 12:31:33.000	Hofner Icon	489.99	186.20	1
4	Veronika	2016-05-04 03:52:23.000	Gibson SG	799.99	240.00	1
5	Simona	2016-04-06 07:53:42.000	Ludwig 5-piece Drum Set with Cymbals	415.00	161.85	1
6	Simona	2016-04-06 07:53:42.000	Rodriguez Caballero 11	699.00	209.70	1
7	Sage	2016-04-08 12:21:31.000	Gibson Les Paul	1199.00	359.70	2
8	Mitsue	2016-04-06 17:24:28.000	Rodriguez Caballero 11	699.00	209.70	1
9	Minna	2016-04-11 08:21:32.000	Gibson SG	799.99	240.00	1
10	Meaghan	2016-04-21 17:52:24.000	Washburn D10S	699.99	210.00	1
11	Matte	2016-04-20 09:17:52.000	Fender Stratocaster	2517.00	1308.84	1
12	Manann	2016-05-06 14:15:21.000	Fender Stratocaster	2517.00	1308.84	1

Query executed successfully.

2. [4] Write a SELECT statement that selects CategoryName column from the Categories table. Return one row for each category that has been used. (Hint: Use an outer join and only return rows where the ProductID column does not contain a null value).

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

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

```

SELECT x.CategoryName
FROM MyGuitarShop.dbo.Categories as x LEFT JOIN MyGuitarShop.dbo.Products as y
ON x.CategoryID=y.CategoryID
where y.ProductID IS NOT NULL;
--Venkata Sai Pawan Komaravolu

```

CategoryName
Guitars
Basses
Basses
Drums
Drums

Query executed successfully.

3. [4] Write a SELECT statement that returns one row for each customer that has orders with these columns: a) The EmailAddress column from the Customers table b) The average of the item price in the OrderItems table multiplied by the quantity in the OrderItems table c) The average of the discount amount column in the OrderItems table multiplied by the quantity in the OrderItems table. Sort the result set in ascending sequence by the item price average for each customer.

Sol: Source Code:

```

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

```

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

```

SELECT x.EmailAddress, AVG(z.ItemPrice * z.Quantity) AS ItemPriceAVG,
AVG(z.DiscountAmount * z.Quantity) AS DiscountAmountAVG
FROM MyGuitarShop.dbo.Customers AS x JOIN MyGuitarShop.dbo.Orders AS y
ON x.CustomerID=y.CustomerID JOIN MyGuitarShop.dbo.OrderItems AS z
ON y.OrderID=z.OrderID
GROUP BY x.EmailAddress
ORDER BY ItemPriceAVG;
--Venkata Sai Pawan Komaravolu

```

EmailAddress	ItemPriceAVG	DiscountAmountAVG
eminv@gmail.com	299.00	0.00
barryz@gmail.com	489.99	186.20
willard@hotmail.com	489.99	186.20
gladys.rim@rim.org	499.99	125.00
simona@morasca.com	557.00	185.775
chanel.caudy@caudy.org	594.495	197.95
fletcher.flosi@yahoo.com	598.00	0.00
calibares@gmail.com	699.00	209.70
allene_iturbide@cox.net	699.00	209.70
amadead@gmail.com	699.00	209.70
mitsue_tollner@yahoo.com	699.00	209.70
ihut@gmail.com	699.00	209.70

Query executed successfully.

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

Sol: Source Code:

```

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

```

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

```

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

```

EmailAddress	NumberOfOrders	TotalAmount
donette_foller@cox.net	1	559.99
frankwilson@sbcglobal.net	1	559.99
minna_amigon@yahoo.com	1	559.99
vinouye@aol.com	1	559.99
gary_hernandez@yahoo.com	1	679.99
art@venere.org	1	679.99
bette_nicka@cox.net	1	839.30
mroyster@royster.com	1	1208.16
matlie@aol.com	1	1208.16
heatheresway@mac.com	1	1208.16
josephine_daraky@daraky.org	1	1208.16
wiki_who@raii.com	1	1208.16

Query executed successfully.

5. [4] (1) Write a SELECT statement that returns three columns: EmailAddress, OrderID, and the order average for each customer. To do this, you can group the result set by the EmailAddress and OrderID columns. In addition, you must calculate the order average from the columns in the OrderItemstable.  
(2) Write a second SELECT statement that uses the first SELECT statement in its FROM clause. The main query should return two columns: the customer's email address and the largest order average for that customer. To do this, you can group the result set by the EmailAddress column.

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

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

```

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

```

EmailAddress	OrderID	OrderAVG
allan.sherwood@yahoo.com	1	839.30
barryz@gmail.com	2	303.79
allan.sherwood@yahoo.com	3	730.655
christineb@solarone.com	4	1678.60
david.goldstein@hotmail.com	5	299.00
enrin@gmail.com	6	299.00
frankwilson@sbcglobal.net	7	513.3233
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
insenshine_darakin@darakin.org	12	1208.16

Query executed successfully.

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

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

```

SELECT EmailAddress,MAX(OrderAvg) as LargeOrder
FROM (SELECT x.EmailAddress,y.OrderID,
AVG((z.ItemPrice-z.DiscountAmount)*z.Quantity) AS OrderAvg
FROM MyGuitarShop.dbo.Customers as x JOIN MyGuitarShop.dbo.Orders as y
ON x.CustomerID=y.CustomerID JOIN MyGuitarShop.dbo.OrderItems as z
ON y.OrderID=z.OrderID
GROUP BY x.EmailAddress,y.OrderID) as a
GROUP BY a.EmailAddress;
--Venkata Sai Pawan Komaravolu

```

EmailAddress	LargeOrder
allisha@slusarski.com	1678.60
allan.shenwood@yahoo.com	839.30
allene_iturbide@cox.net	489.30
amaclead@gmail.com	489.30
art@venere.org	679.99
banyz@gmail.com	303.79
bette_nicka@cox.net	839.30
calibares@gmail.com	489.30
chanel.caudy@caudy.org	396.545
christineb@solarone.com	1678.60
david.goldstein@hotmail.com	2799.95
donna.toller@cox.net	550.99

Query executed successfully.

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

Sol:

```

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

```

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

```

SELECT x.EmailAddress,y.OrderID,y.OrderDate
FROM MyGuitarShop.dbo.Customers AS x JOIN MyGuitarShop.dbo.Orders AS y
ON x.CustomerID=y.CustomerID
WHERE y.OrderDate IN (SELECT MIN(OrderDate) FROM MyGuitarShop.dbo.Orders AS z
WHERE x.CustomerID = z.CustomerID);
--Venkata Sai Pawan Komaravolu

```

EmailAddress	OrderID	OrderDate
allan.sherwood@yahoo.com	1	2016-03-28 09:40:28.000
barryz@gmail.com	2	2016-03-28 11:23:20.000
christineb@solarone.com	4	2016-03-30 15:22:31.000
david.goldstein@hotmail.com	5	2016-03-31 05:43:11.000
erinn@gmail.com	6	2016-03-31 18:37:22.000
frankwilson@sbglobal.net	7	2016-04-01 23:11:12.000
gary_hernandez@yahoo.com	8	2016-04-02 11:26:38.000
heatheresway@mac.com	10	2016-04-03 14:59:20.000
jbutt@gmail.com	11	2016-04-04 06:24:44.000
josephine_darakyj@darakyj.org	12	2016-04-04 08:15:12.000
art@venere.org	13	2016-04-04 11:20:31.000
Inannockri@hotmail.com	14	2016-04-05 09:24:53.000

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

Sol: Source code:

```

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

```

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

```

SELECT ListPrice,
       CAST(ListPrice AS DECIMAL(8,3)),
       CONVERT(REAL,ListPrice),
       CAST(ListPrice AS INT)
  FROM MyGuitarShop.dbo.Products;
  
```

--Venkata Sai Pawan Komaravolu

	ListPrice	(No column name)	(No column name)	(No column name)
1	699.00	699.000	699	699
2	1199.00	1199.000	1199	1199
3	2517.00	2517.000	2517	2517
4	489.99	489.990	489.99	490
5	299.00	299.000	299	299
6	415.00	415.000	415	415
7	799.99	799.990	799.99	800
8	499.99	499.990	499.99	500
9	699.99	699.990	699.99	700
10	799.99	799.990	799.99	800

Query executed successfully.

8. [4] Write a SELECT statement that returns these columns from the Orders table: a) The CardNumber column b) The length of the CardNumber column c) The last three digits of the CardNumber column d) A column that displays the last four digits of the CardNumber column in this format: XXXXXX-XXXX-1234. In other words, use Xs for the first 12 digits of the card number and actual numbers for the last four digits of the number.

Sol:

```

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

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

```

SELECT CardNumber,
LEN(CardNumber) AS CardNumberLength,
RIGHT(CardNumber, 3) AS Last3Digits,
('XXXX-XXXX-XXXX-' + RIGHT(CardNumber, 4)) AS PrintedFormat
FROM MyGuitarShop.dbo.Orders;
--Venkata Sai Pawan Komaravolu

```

	CardNumber	CardNumberLength	Last3Digits	PrintedFormat
1	4111111111111111	16	111	XXXX-XXXX-XXXX-1111
2	4012888888881881	16	881	XXXX-XXXX-XXXX-1881
3	4111111111111111	16	111	XXXX-XXXX-XXXX-1111
4	378282463100005	16	005	XXXX-XXXX-XXXX-0005
5	4111111111111111	16	111	XXXX-XXXX-XXXX-1111
6	6011111111111117	16	117	XXXX-XXXX-XXXX-1117
7	5555555555554444	16	444	XXXX-XXXX-XXXX-4444
8	4012888888881881	16	881	XXXX-XXXX-XXXX-1881
9	4111111111111111	16	111	XXXX-XXXX-XXXX-1111
10	4111111111111111	16	111	XXXX-XXXX-XXXX-1111
11	4012888888881881	16	881	XXXX-XXXX-XXXX-1881
12	4111111111111111	16	111	XXXX-XXXX-XXXX-1111

Query executed successfully.

9. [4] Write a SELECT statement that returns these columns from the Orders table: a) The OrderID column  
b) The OrderDate column c) A column named ApproxShipDate that's calculated by adding 3 days to the OrderDate column d) The ShipDate column e) A column named MonthsToShip that shows the number of months between the order date and the ship date When you have this working, add a WHERE clause that retrieves just the orders for Jan to June 2016.

Sol:

```

SELECT OrderID, OrderDate,
dateadd(day, 3, OrderDate) as ApproxShipDate, ShipDate,
datediff(month, OrderDate, ShipDate) as MonthsToShip
FROM MyGuitarShop.dbo.Orders
where OrderDate BETWEEN '01-01-2016' AND '06-01-2016';

```

Comments: We select OrderID,orderDate and a new date which is the Approx shipping date that is approx. 3 days from the day of ordering, from the orders table where orderdate year is 2016 and month is between jan and june

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, there is a connection to 'DESKTOP-A19REFF\SQLEXPRESS'. In the center pane, a query window titled 'p3.8.sql' is open, displaying the following SQL code:

```

SELECT OrderID, OrderDate,
       dateadd(day, 3, OrderDate) as ApproxShipDate, ShipDate,
       datediff(month, OrderDate, ShipDate) as MonthsToShip
FROM MyGuitarShop.dbo.Orders
WHERE OrderDate BETWEEN '01-01-2016' AND '06-01-2016';
--Venkata Sai Pawan Komaravolu

```

The results pane shows a table with 12 rows of data:

OrderID	OrderDate	ApproxShipDate	ShipDate	MonthsToShip
1	2016-03-28 09:40:28.000	2016-03-31 09:40:28.000	2016-03-31 09:41:11.000	0
2	2016-03-28 11:23:20.000	2016-03-31 11:23:20.000	2016-03-31 11:24:03.000	0
3	2016-03-29 09:44:58.000	2016-04-01 09:44:58.000	2016-04-01 09:45:41.000	1
4	2016-03-30 15:22:31.000	2016-04-02 15:22:31.000	2016-04-02 15:23:14.000	1
5	2016-03-31 05:43:11.000	2016-04-03 05:43:11.000	2016-04-03 05:43:54.000	1
6	2016-03-31 18:37:22.000	2016-04-03 18:37:22.000	2016-04-03 18:38:05.000	1
7	2016-04-01 23:11:12.000	2016-04-04 23:11:12.000	2016-04-04 23:11:55.000	0
8	2016-04-02 11:26:38.000	2016-04-05 11:26:38.000	2016-04-05 11:27:21.000	0
9	2016-04-03 12:22:31.000	2016-04-06 12:22:31.000	2016-04-06 12:23:14.000	0
10	2016-04-03 14:59:20.000	2016-04-06 14:59:20.000	2016-04-06 15:00:03.000	0
11	2016-04-04 06:24:44.000	2016-04-07 06:24:44.000	2016-04-07 06:25:27.000	0
12	2016-04-04 08:15:12.000	2016-04-07 08:15:12.000	2016-04-07 08:15:55.000	0

At the bottom of the results pane, it says 'Query executed successfully.' The status bar at the bottom right shows 'DESKTOP-A19REFF\SQLEXPRESS... sa (64) MyGuitarShop 00:00:00 41 rows'.

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

Sol : `INSERT INTO MyGuitarShop.dbo.Customers  
(EmailAddress,Password,FirstName,LastName)`

Comments: We insert in to customers table and add EmailAddress , password, first name and Lastname columns

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

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

```
INSERT INTO MyGuitarShop.dbo.Customers
(EmailAddress, Password, FirstName, LastName)
VALUES ('noorie@Krieger.com', 'secret', 'Noorie', 'Krieger');

--Venkata Sai Pawan Komaravolu
```

The status bar at the bottom indicates "Query executed successfully." and "1 row affected". The completion time is shown as 2020-04-01T20:33:33.5692874-07:00.

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

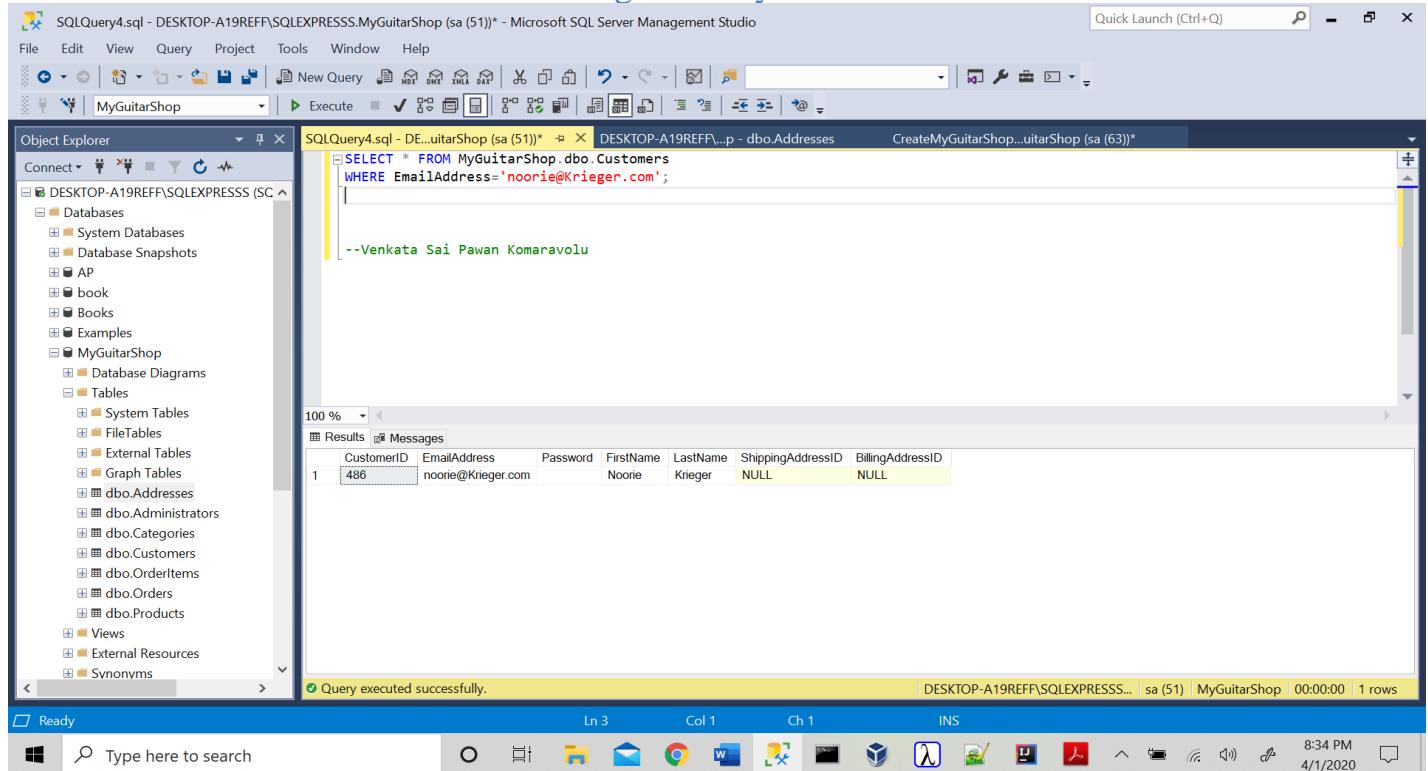
Comments: We select everything from Customer table where EmailAddress is 'noorie@Krieger.com'

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

Sol :

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

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



SQLQuery4.sql - DESKTOP-A19REFF\SQLEXPRESSS.MyGuitarShop (sa (51))\* - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query MDW DMW TBLA DAX Execute

Object Explorer

MyGuitarShop

SQLQuery4.sql - DE...uitarShop (sa (51)) \* DESKTOP-A19REFF...p - dbo.Addresses CreateMyGuitarShop...uitarShop (sa (63))\*

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

--Venkata Sai Pawan Komaravolu

Results Messages

CustomerID	EmailAddress	Password	FirstName	LastName	ShippingAddressID	BillingAddressID
1	486	noorie@Krieger.com	Noorie	Krieger	NULL	NULL

Query executed successfully.

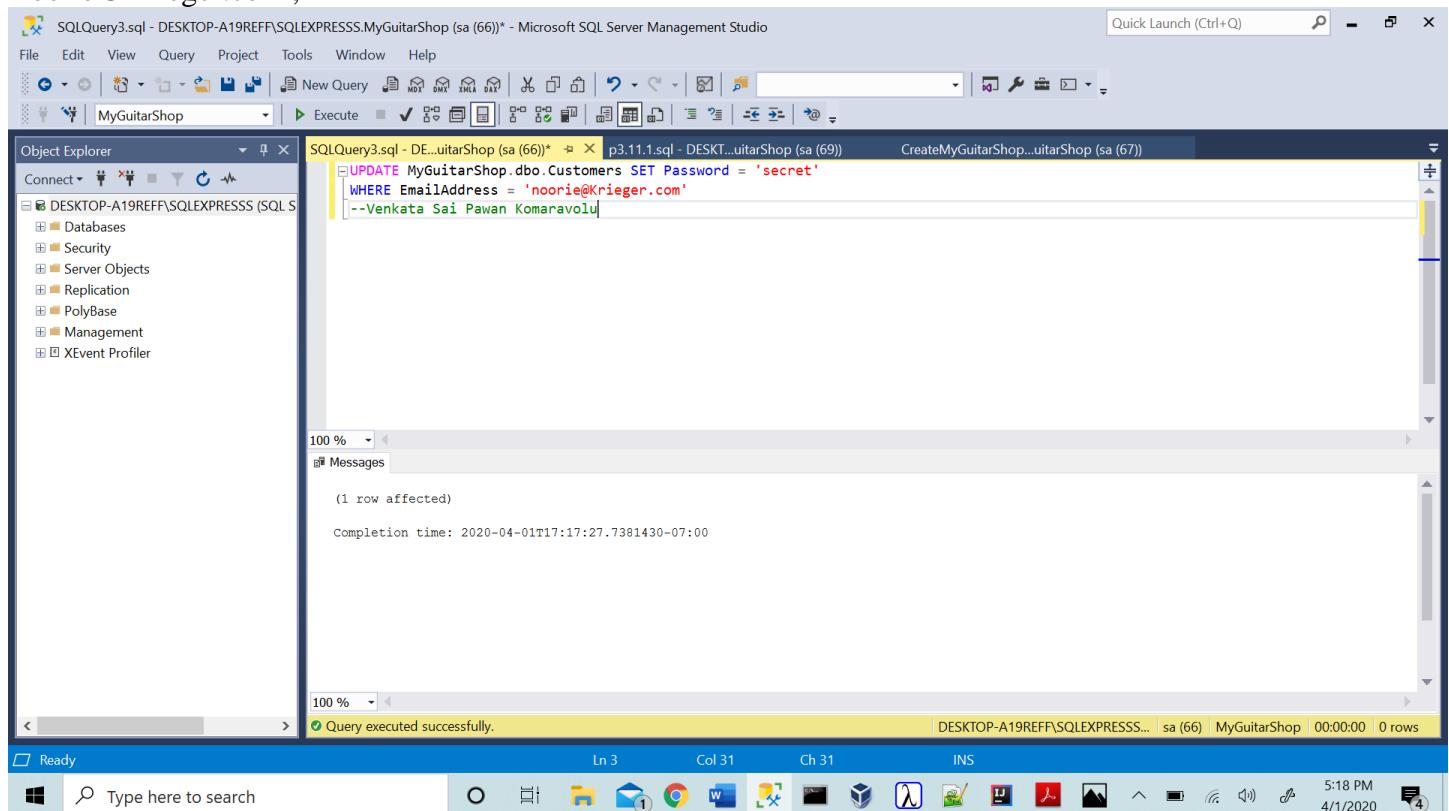
DESKTOP-A19REFF\SQLEXPRESSS... sa (51) MyGuitarShop 00:00:00 1 rows

Ready Type here to search Ln 3 Col 1 Ch 1 INS 8:34 PM 4/1/2020

UPDATE MyGuitarShop.dbo.Customers SET Password = 'secret'

WHERE EmailAddress = 'noorie@Krieger.com'

Comments: We update the Customers table and set password to 'secret' with emailaddress = 'noorie@Krieger.com';



SQLQuery3.sql - DESKTOP-A19REFF\SQLEXPRESSS.MyGuitarShop (sa (66)) \* - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query MDW DMW TBLA DAX Execute

Object Explorer

MyGuitarShop

SQLQuery3.sql - DE...uitarShop (sa (66)) \* p3.11.1.sql - DESKT...uitarShop (sa (69)) CreateMyGuitarShop...uitarShop (sa (67))

```
UPDATE MyGuitarShop.dbo.Customers SET Password = 'secret'
WHERE EmailAddress = 'noorie@Krieger.com'
```

--Venkata Sai Pawan Komaravolu

Messages

(1 row affected)

Completion time: 2020-04-01T17:17:27.7381430-07:00

Query executed successfully.

DESKTOP-A19REFF\SQLEXPRESSS... sa (66) MyGuitarShop 00:00:00 0 rows

Ready Type here to search Ln 3 Col 31 Ch 31 INS 5:18 PM 4/1/2020

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

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

```

p3.11.1.sql - DESKTOP-A19REFF\SQLEXPRESSS.MyGuitarShop (sa (69)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query MDX DML DML DAX Execute
MyGuitarShop Object Explorer
Connect DESKTOP-A19REFF\SQLEXPRESSS (SQL Server)
Databases Security Server Objects Replication PolyBase Management XEvent Profiler
p3.11.1.sql - DESKTOP-A19REFF\MyGuitarShop (sa (67)) CreateMyGuitarShop...uitarShop (sa (67)) SQLQuery1.sql - DE...SS.master (sa (57))*
--Venkata Sai Pawan Komaravolu
SELECT * FROM MyGuitarShop.dbo.Customers WHERE EmailAddress = 'noorie@Krieger.com';
100 %
Results Messages
CustomerID EmailAddress Password FirstName LastName ShippingAddressID BillingAddressID
1 486 noorie@Krieger.com secret Noorie Krieger NULL NULL
Query executed successfully.
DESKTOP-A19REFF\SQLEXPRESSS... sa (69) MyGuitarShop 00:00:00 1 rows
Ready Type here to search Ln 2 Col 31 Ch 31 INS
5:15 PM 4/1/2020

```

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

- [4] Create a view named OrderItemProducts that returns columns from the Orders, OrderItems, and Products tables. a) This view should return these columns from the Orders table: OrderID, OrderDate, TaxAmount, and ShipDate. b) This view should return these columns from the OrderItems table: DiscountAmount, FinalPrice (the discount amount subtracted from the item price), Quantity, and ItemTotal (the calculated total for the item). c) This view should return the ProductName column from the Products table.

Sol: CREATE VIEW OrderItemProducts AS

```

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

```

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

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer on the left, the database 'MyGuitarShop' is selected. In the center, a query window displays the creation of a view:

```

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

The status bar at the bottom indicates "Query executed successfully." and "Completion time: 2020-04-01T20:35:06.9746626-07:00".

**SELECT \* FROM OrderItemProducts;**

Comments: We select everything from orderItemProducts.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer on the left, the database 'MyGuitarShop' is selected. In the center, a query window displays the execution of the previous query:

```

SELECT * FROM OrderItemProducts;
  
```

The results pane shows a table with 47 rows of data. The columns are:

	OrderID	OrderDate	TaxAmount	ShipDate	ItemPrice	DiscountAmount	FinalPrice	Quantity	ItemTotal	ProductName
1	1	2016-03-28 09:40:28.000	58.75	2016-03-31 09:41:11.000	1199.00	359.70	839.30	1	839.30	Gibson Les Paul
2	2	2016-03-28 11:23:20.000	21.27	2016-03-31 11:24:03.000	489.99	186.20	303.79	1	303.79	Hofner Icon
3	3	2016-03-29 09:44:58.000	102.29	2016-04-01 09:45:41.000	2517.00	1308.84	1208.16	1	1208.16	Fender Stratocaster
4	3	2016-03-29 09:44:58.000	102.29	2016-04-01 09:45:41.000	415.00	161.85	253.15	1	253.15	Ludwig 5-piece Drum Set with Cymbals
5	4	2016-03-30 15:22:31.000	117.50	2016-04-02 15:23:14.000	1199.00	359.70	839.30	2	1678.60	Gibson Les Paul
6	5	2016-03-31 05:43:11.000	20.93	2016-04-03 05:43:54.000	299.00	0.00	299.00	1	299.00	Tama 5-Piece Drum Set with Cymbals
7	6	2016-03-31 18:37:22.000	20.93	2016-04-03 18:38:05.000	299.00	0.00	299.00	1	299.00	Tama 5-Piece Drum Set with Cymbals
8	7	2016-04-01 23:11:12.000	107.80	2016-04-04 23:11:55.000	699.99	210.00	489.99	1	489.99	Washburn D10S
9	7	2016-04-01 23:11:12.000	107.80	2016-04-04 23:11:55.000	799.99	240.00	559.99	1	559.99	Gibson SG
10	7	2016-04-01 23:11:12.000	107.80	2016-04-04 23:11:55.000	699.99	210.00	489.99	1	489.99	Washburn D10S
11	8	2016-04-02 11:26:38.000	47.60	2016-04-05 11:27:21.000	799.99	120.00	679.99	1	679.99	Yamaha FG700S
12	9	2016-04-03 12:22:31.000	102.75	2016-04-06 12:23:14.000	699.00	209.70	489.30	3	1467.90	Rodríguez Caballero 11

The status bar at the bottom indicates "Query executed successfully." and "Completion time: 2020-04-01T20:35:06.9746626-07:00".

2. [5] Create a view named Top5LeastSelling that uses the view you created in Section D Question 1. This view should return some summary information about five least selling products. Each row should include these columns: ProductName, OrderTotal (the total sales for the product) and OrderCount (the number of times the product has been ordered).

Sol: CREATE VIEW Top5LeastSelling AS

```
SELECT TOP 5 ProductName, COUNT(Quantity) OrderCount,
SUM(Quantity* ItemPrice) AS OrderTotal
FROM OrderitemProducts
GROUP BY ProductName
ORDER BY OrderCount ;
```

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

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

```
CREATE VIEW Top5LeastSelling AS
SELECT TOP 5 ProductName, COUNT(Quantity) OrderCount,
SUM(Quantity* ItemPrice) AS OrderTotal
FROM OrderitemProducts
GROUP BY ProductName
ORDER BY OrderCount ;
--Venkata Sai Pawan Komaravolu
```

The status bar at the bottom indicates "Query executed successfully." and "0 rows". The title bar shows the connection details: DESKTOP-A19REFF\SQLEXPRESSSS.MyGuitarShop (sa (51))\* - Microsoft SQL Server Management Studio.

SELECT \* FROM Top5LeastSelling;

Comments: We select everything from top5LeastSelling view.

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar reads "SQLQuery4.sql - DESKTOP-A19REFF\SQLEXPRESSS.MyGuitarShop (sa (51))\* - Microsoft SQL Server Management Studio". The Object Explorer sidebar shows the database structure for "MyGuitarShop", including tables like "AP", "book", "Books", "Examples", and "Tables". The main results pane displays the output of a query:

```
SELECT * FROM Top5LeastSelling;
--Venkata Sai Pawan Komaravolu
```

The results table shows the following data:

	ProductName	OrderCount	OrderTotal
1	Fender Precision	2	999.98
2	Ludwig 5-piece Drum Set with Cymbals	2	830.00
3	Hofner Icon	3	1469.97
4	Tama 5-Piece Drum Set with Cymbals	3	1196.00
5	Washburn D10S	3	2099.97

Below the results, a message states "Query executed successfully." The status bar at the bottom right shows "8:36 PM 4/1/2020".

3. [5] Write a script that creates and calls a stored procedure named spUpdateProductDiscount that updates the DiscountPercent column in the Products table. This procedure should have one parameter for the product ID and another for the discount percent. If the value for the DiscountPercent column is a negative number, the stored procedure should raise an error that indicates that the value for this column must be a positive number. Code at least two EXEC statements that test this procedure.

Sol: Source Code : CREATE PROCEDURE spUpdateProductDiscount

```
(  
    @ProductID int,  
    @DiscountPercent int  
)  
AS  
BEGIN  
BEGIN TRY  
BEGIN TRANSACTION;  
UPDATE Products set DiscountPercent = @DiscountPercent where ProductID = @ProductID;  
COMMIT TRANSACTION;  
END TRY  
BEGIN CATCH  
IF @DiscountPercent < 0  
ROLLBACK TRANSACTION;  
PRINT 'DiscountPercent must be positive';  
END CATCH  
END  
GO
```

## CSE 581 Introduction to Database Management Systems

SPRING 2020

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

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure for 'MyGuitarShop'. The central pane displays a T-SQL script named 'SQLQuery4.sql' containing a stored procedure definition:

```
BEGIN
BEGIN TRY
    BEGIN TRANSACTION;
        UPDATE Products set DiscountPercent = @DiscountPercent where ProductID = @ProductID;
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @DiscountPercent < 0
            ROLLBACK TRANSACTION;
        PRINT 'DiscountPercent must be positive';
    END CATCH
END
GO
```

Below the script, the message pane shows:

- Commands completed successfully.
- Completion time: 2020-04-01T20:37:10.9590441-07:00

The status bar at the bottom indicates: DESKTOP-A19REF\SQLEXPRESSSS... sa (51) MyGuitarShop 00:00:00 0 rows.

SELECT \* FROM MyGuitarShop.dbo.Products;

Comments: We select everything from Products table.

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left is the Object Explorer pane, which lists various databases, tables, and objects within the 'MyGuitarShop' database. The main central area displays the results of a SQL query:

```
SELECT * FROM MyGuitarShop.dbo.Products;
```

The results grid shows 10 rows of data from the 'Products' table, with columns including ProductID, CategoryID, ProductCode, ProductName, Description, ListPrice, DiscountPercent, and DateAdded. The data includes various guitar and drum models from brands like Fender, Gibson, Yamaha, Washburn, Rodriguez, Hofner, and Ludwig.

ProductID	CategoryID	ProductCode	ProductName	Description	ListPrice	DiscountPercent	DateAdded
1	1	strat	Fender Stratocaster	The Fender Stratocaster is the electric guitar design...	699.99	30.00	2015-10-30 09:32:40.000
2	1	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	1	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	1	fg700s	Yamaha FG700S	The Yamaha FG700S solid top acoustic guitar has ...	489.99	38.00	2016-06-01 11:12:59.000
5	1	washburn	Washburn D10S	The Washburn D10S acoustic guitar is superbly cra...	299.00	0.00	2016-07-30 13:58:35.000
6	1	rodriguez	Rodriguez Caballero 11	Featuring a carefully chosen, solid Canadian cedar...	415.00	39.00	2016-07-30 14:12:41.000
7	2	precision	Fender Precision	The Fender Precision bass guitar delivers the soun...	799.99	30.00	2016-06-01 11:29:35.000
8	2	hofner	Hofner Icon	With authentic details inspired by the original, the H...	499.99	25.00	2016-07-30 14:18:33.000
9	3	ludwig	Ludwig 5-piece Drum Set with Cymbals	This product includes a Ludwig 5-piece drum set an...	699.99	30.00	2016-07-30 12:46:40.000
10	3	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

At the bottom of the results grid, a message states "Query executed successfully." The status bar at the bottom right shows the date and time as 4/1/2020 8:38 PM.

4. [5] Write a script that calculates the common factors between 8 and 24. To find a common factor, you can use the modulo operator (%) to check whether a number can be evenly divided into both numbers. Then, this script should print lines that display the common factors like this: Common factors of 8 and 24  
1 2 4 8

Sol: Source Code: DECLARE @a INT;

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

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

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 'SQLQuery4.sql' contains the following T-SQL script:

```

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

```

The results pane shows the output of the query:

	(No column name)
1	Common Factors of 8 and 24 1 2 4 8

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

Sol: USE MyGuitarShop;

GO

CREATE FUNCTION fnDiscountPrice (@ItemID INT)

RETURNS INT

BEGIN

RETURN (SELECT (ItemPrice - DiscountAmount) AS Discount\_Price

FROM MyGuitarShop.dbo.OrderItems

WHERE ItemID = @ItemID);

END;

GO

PRINT 'Discount price: \$' + CONVERT(varchar, dbo.fnDiscountPrice(3),1);

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

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

```

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

--Venkata Sai Pawan Komaravolu

```

The 'Messages' pane below the query window shows the output of the execution:

```

Discount price: $1208
Completion time: 2020-04-01T20:39:26.9429940-07:00

```

The status bar at the bottom indicates "Query executed successfully." and shows the session details: DESKTOP-A19REFF\SQLEXPRESSSS... sa (51) MyGuitarShop | 00:00:00 | 0 rows.

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

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

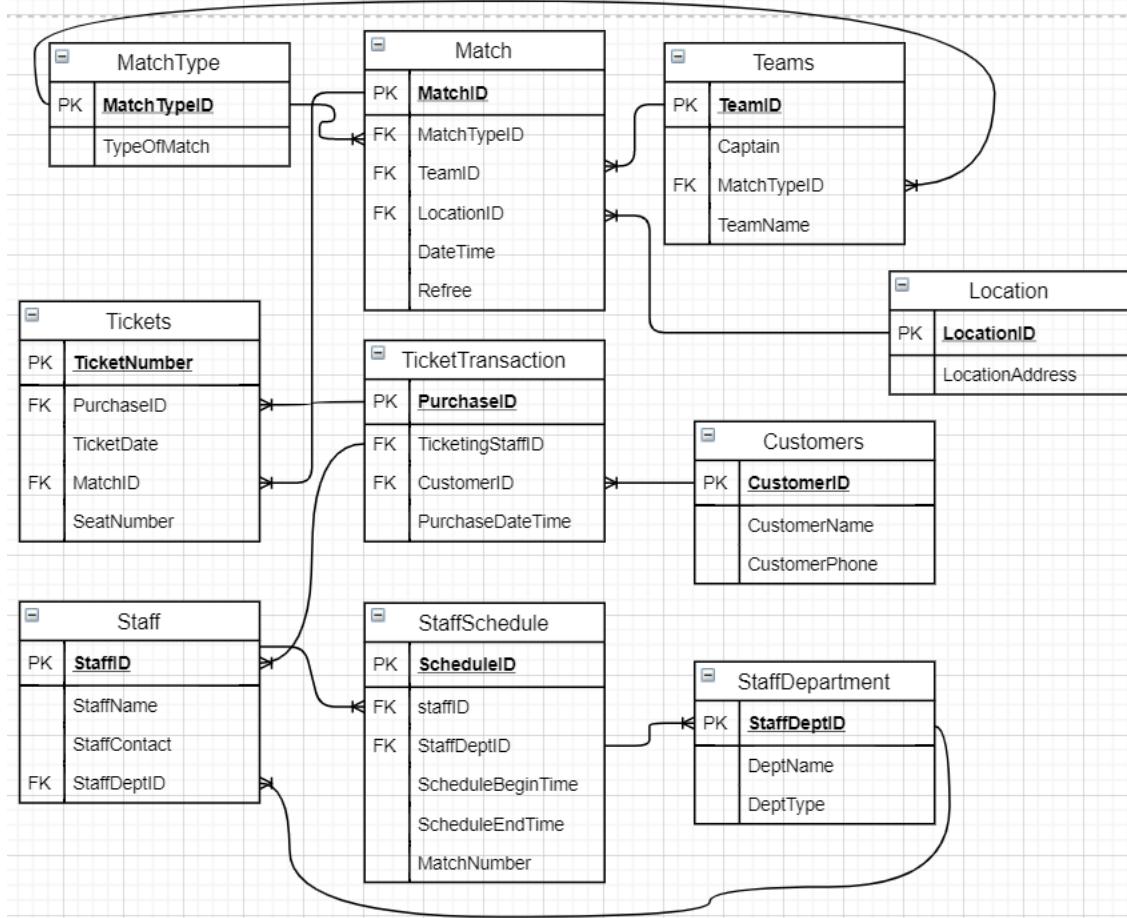
The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar reads "SQLQuery4.sql - DESKTOP-A19REFF\SQLEXPRESSS.MyGuitarShop (sa (51))\* - Microsoft SQL Server Management Studio". The main window has three tabs: "Object Explorer", "SQLQuery4.sql - DE...uitarShop (sa (51))", and "CreateMyGuitarShop...uitarShop (sa (63))". The "Object Explorer" pane shows the database structure of "MyGuitarShop" including databases, tables, and stored procedures. The central pane displays a T-SQL script for creating a function named "ItemTotal". The script uses a temporary table #t to store intermediate results and then selects the sum of discounted prices multiplied by quantity for a given item ID. The "Messages" pane at the bottom shows the output: "Total Amount is \$299.00" and "Completion time: 2020-04-01T20:39:54.3831335-07:00". A status bar at the bottom right indicates "Query executed successfully." and shows the session details: "DESKTOP-A19REFF\SQLEXPRESSS... sa (51) MyGuitarShop | 00:00:00 | 0 rows".

```

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

```

- II.** Database Design [20 pts.] Create a sample database design for Cuse's Box Office for all Sports matches (Basketball, Soccer, Lacrosse, etc) and draw one database model for it. The Box Office needs to keep a track of all types of sports matches taking place at the Dome as well as all other 'away' locations. By tracking the type of sport, its location, and the teams (and their captains) involved per match, the Box Office wants to sell tickets for all these matches. Each ticket transaction will have different ticketing-staff and customers involved with all critical information per customer, ticketing-staff, and transaction recorded. Similarly, each match will also have other staff allotted: security-staff and café-staff. All the three types of staff members (ticketing-staff, security-staff, café-staff) will have schedules allotted. Your design could add more things to the existing requirements, but all the given requirements should be first met.
1. [3] Design the database that makes sense for the problem and select fields that make the most sense. A complete screenshot of your final design model is required.



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

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

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

#### For MatchType table:-

In this table, I have used MatchTypeID and TypeofMatch as columns.

Primary key used in this table is MatchTypeID.

#### For Match table:-

In this table, I have used MatchID,MatchTypeID, TeamID, LocationID, DateTime and Referee as columns.

Primary keys used in this table are MatchID

Foreign keys used in this table are MatchTypeID,LocationID,TeamID.

#### For Teams table:-

In this table, I have used TeamID, MatchTypeID and Captain,TeamName as columns.

Primary key in this table is TeamID

**For Locations table:-**

In this table, I have used LocationID and LocationAddress as columns.

Foreign key in this table is LocationID.

**For Tickets table:-**

In this table, I have used TicketNumber, TicketDate, MatchID, SeatNumber and PurchaseID as columns.

Primary key in this table is TicketNumber

Foreign key in this table is MatchID.

**For TicketTransaction table:-**

In this table, I have used PurchaseID, CustomerID, PurchaseDateTime, TicketingStaffID as columns.

Primary key in this table is PurchaseID

Foreign keys in this table are TicketingStaffID,CustomerID.

**For Customers table:-**

In this table, I have used CustomerID, CustomerName, CustomerPhone as columns.

Primary key in this table is CustomerID.

**For Staff table:-**

In this table, I have used StaffID, StaffName, StaffContact and StaffDeptID as columns.

Primary Keys in this table is StaffID

Foreign keys in this table are StaffDeptID.

**For StaffSchedule table:-**

In this table, I have used ScheduleID, StaffID, StaffDeptID, ScheduleBeginTime and ScheduleEndTime as columns.

Primary key in this table is ScheduleID.

Foreign Key in this table is StaffID and StaffDeptID

**For StaffDepartment table:-**

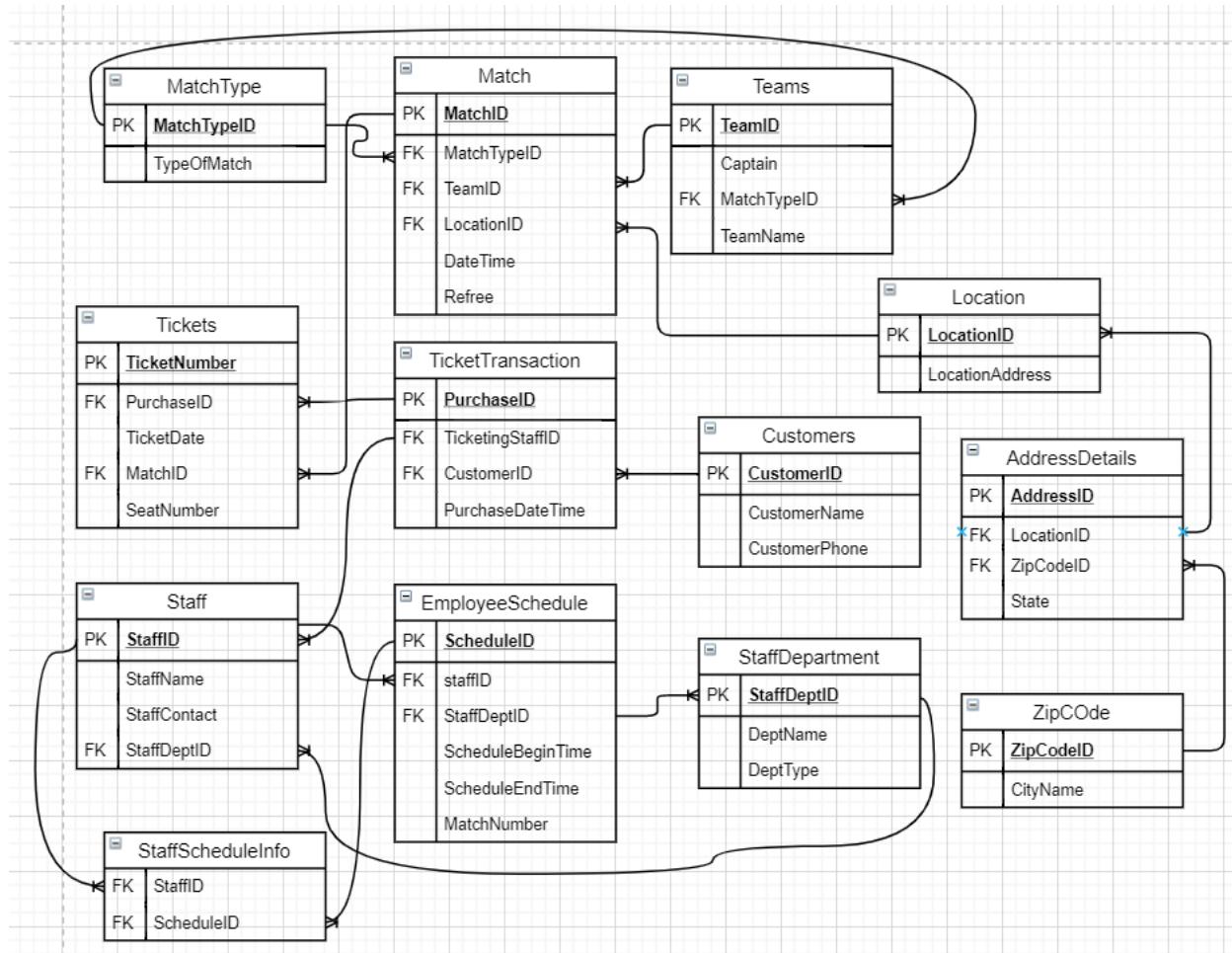
In this table, I have used StaffDeptID, DeptName and DeptType as columns.

Primary key in this table is StaffDeptID.

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

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

- There is a one-to-many relationship between Staff and StaffSchedule.
  - There is a many-to-many relationship between Staff and StaffDepartment.
3. [5] Normalize your design into 3rd Normal Form. Please use MS Visio or any similar database design tool.



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

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

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

#### For StaffScheduleInfo:-

In this table, I have used StaffID and ScheduleID as columns.

Foreign keys in this table are StaffID and ScheduleID.

#### For AddressDetails table:-

In this table, I have used AddressID, LocationID and ZipCodeID , state as columns.

Primary key in this table is AddressID.

Foreign key in this table is ZipCodeID,LocationID.

**For ZipCode table:-**

In this table, I have used ZipCodeID and CityName as columns.

Primary key in this table is ZipCodeID.

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

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

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

There are 4new relationships added between the tables:

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

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