

Name: Heli Shah

SU ID: 733914949

Course: CSE 581 Introduction to Database Management System

Semester: Fall 2019

PROJECT 2

AMAZON COMPETITOR DATABASE

An Amazon-like database for it's competitor

1. ABSTRACT:

For the project 2,

The Amazon Competitor Database is created keeping in mind the functionality of Amazon and its day to day operations. The database contains various tables and there is a very complex relationship among them.

The purpose of this database is to bring a model for an Amazon competitor by bringing into design all the important aspects that a company like Amazon would need. The database contains critical information of a particular company's data and the data can be used in a number of ways as per the need of the company. This database is implemented in Azure Data Studio. This relational database that has been implemented provides more than enough relationship and tables to store relevant information for the company's growth and survival. The database takes care of the Order Management, Shipping, Warehousing and Stocking information along with the Customer and Product information.

However, the database is designed in a way that it is just not limited to the tables provided in here and can be upgraded to include more features. This is due to the ease of adding the referential integrity of SQL. The tables are created and the sample entries are already added to the tables for the better understanding of the structure. In addition to that, store procedures, functions, triggers, column constraints while making the tables and making different views are created and implemented on this database. Certain **security features** are also added to the database. This database is all set to serve a company which is inline the operations of Amazon.

2. DESIGN:

2.1 INTRODUCTION:

In this project a database for an Amazon competitor is designed, tested and verified. The purpose of such a database is to obtain the necessary information a company in the online-store field would require. Firstly, the database is designed with a number of essential features and then built upon by adding relevant tables, tables that add critical information to the database. It is ensured that a database with a number of entities should be normalized. Here all these entities provide the vital information to construct a logistics database. This competitor will be able to test the database with a number of T-SQL commands and various views and queries. The main purpose to design a proper database is to ensure that it is normalized and the entities are not directly dependent on each other, which indeed is a very crucial factor. The database for the competitor has a number of relationships including one-to-many, many-to-many, many-to-one and one-to-one. All these relationships provide necessary information and tell how the data is to be stored. Use of such a database increases the efficiency of the business operation and reduces overall costs. The database uses the perfect schema to allow updating and viewing data at ease. Also, one of the reasons why this design supersedes other databases is its security information. Moreover, the database design ensures that redundant information is not stored in it. When less redundant information is stored on the database the database efficiency increases by speed and storage. More the data stored in the memory more difficult it would become to access data. A unique feature of this database is that it gives vital information for each and every entity and its attributes. The design is made in such a way that any important attribute can be added to the particular entity easily using SQL server. Since this database has been designed in SQL server data retrieval and data access have never become easier since. Queries used here are powerful tools that provide information instantly. In this database most of the entities one can look for in the substantial growth of the company has been identified and entered and also newer entities can be added when necessary due the primary key and foreign key constraints this database has provided. The Business problem in this industry is uniquely identified in each domain and then the tables have been constructed to solve these problems. A reason would be to make the database more understandable for the user. Overall the database has been carefully designed, implemented and tested to tackle most of the business problems an Amazon competitor would endure.

2.2 DESIGN EXPLAINATION:

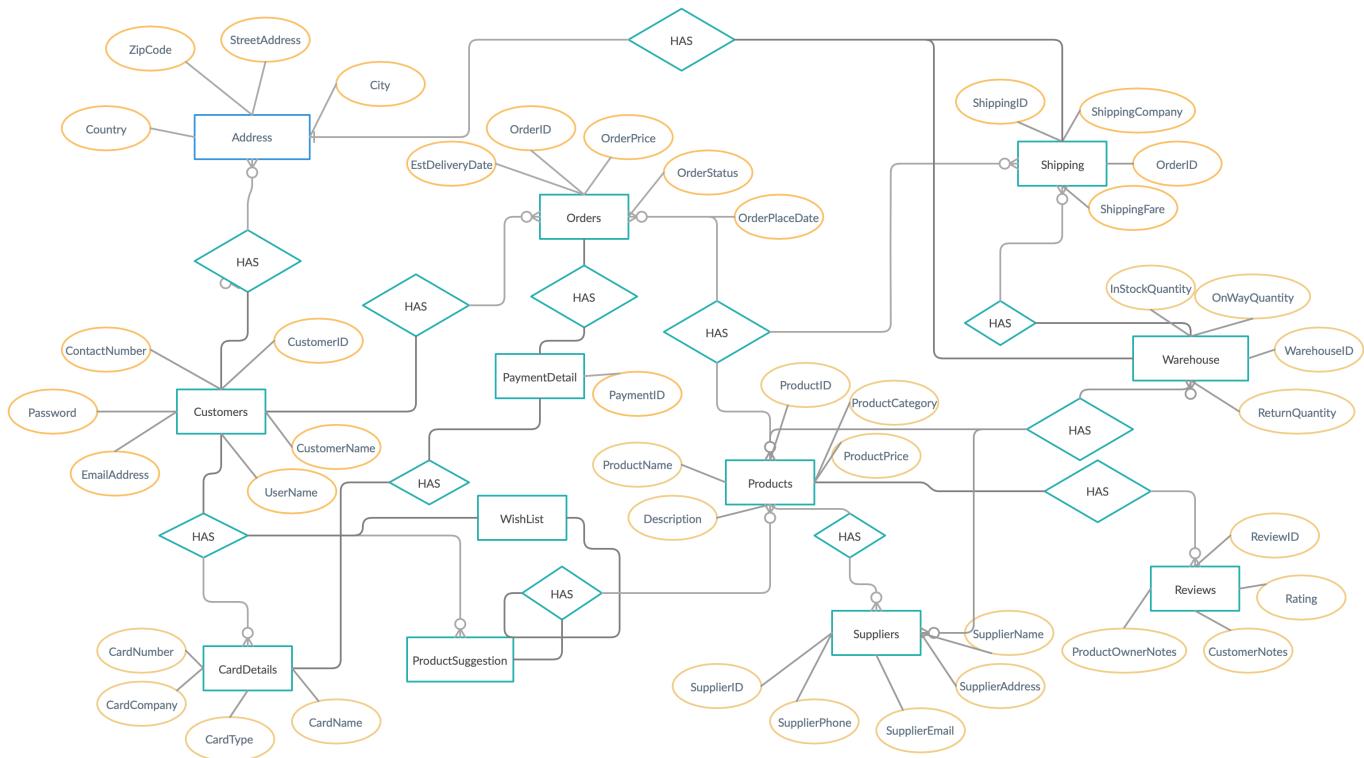
The database has been designed by taking into account a number of considerations to solve the given business problem. There are a total of 17 tables taken into consideration while addressing a company like Amazon. There are many relationships among the tables: one-to-one, one-to-many and many-to-many. These relationships can be clearly seen in the Entity-Relationship diagram below. the testing is done using a number of SQL queries and T-SQL commands such as **stored procedures**, **functions**, **triggers** and also with views. There are many constraints that are taken care of while designing the tables like columns having NOT NULL so that there is an entry mandatorily, providing default values to some columns in case values are not added. Each table record is additionally identified uniquely by its Primary Key. A number of Foreign Key constraints have been added to provide a proper data flow between the related tables. There is no redundancy in the tables which is ensured as the database is reduced to its 3rd normal form. Thus, it is also ensured that all tables have only one primary key and the foreign key is related only to the primary key of another table.

This database contains the personal information of customers (or the account holders) like their name, address, contact information like email, phone number and address and also their payment method details. The address of the customers is whole new entity that has the Street address, City, State and Zip Code. The card details contain the Card number, Card Company, Type and name on the card. The customers personal information also contains the login information of the customer like username and password. The other important aspect of the database is the Orders. This entity contains OrderID, OrderStatus, OrderPrice, OrderPlaceDate, EstimatedDeliveryDate. This entity is related to the customers as customers have many orders. Order is also related to the PaymentDetails entity which is in turn related to the CardDetails entity. Customers have a WishList and ProductSuggestion entity. Both of them contains the product information in the respective details. Another important entity introduced here is the Products. This products entity contains the ProductID, ProductName, Description, ProductCategory and Product Price. It is related to Orders entity in a many-to-many manner. Two important entities that are added here are Suppliers and Warehouse. The Suppliers entity has attributes like ID, Contact, Address and is related to the product in a many-to-many way. The reason for this is one product may be supplied by more than one supplier and one supplier can deliver more than one type of product. Coming to Warehouse, it is related to the products with the exact same relation as Suppliers. It just contains the entities: ID, InStock Quantity of products, OnTheWay to deliver the products and the return products. One more relation is

between Warehouse and entity Shipping. It is linked in a way that one warehouse unit is connected to many shipping units. The Shipping unit is in turn related to the Orders unit where one order can have more than one shipping label (if the items are being shipped in many different parts). There is one more entity of reviews that is related to the Products entity in a way that each product can have many reviews.

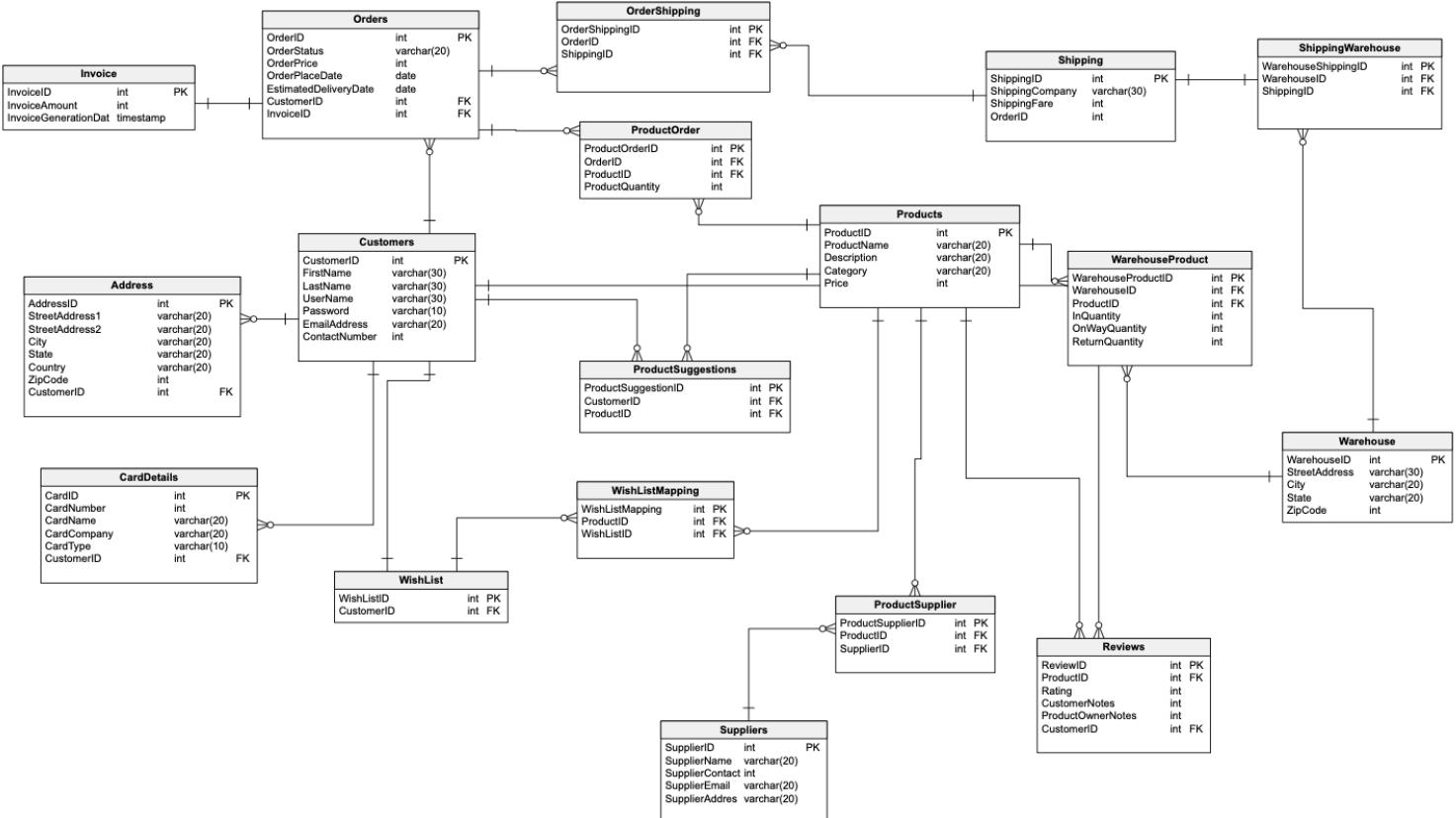
2.3 ENTITY RELATIONSHIP DESIGN:

This whole relationship is described through ER diagram which is posted below:



This Entity Relationship diagram depicts all the relationships among the entities and also the attributes of each of the entities.

2.4 DATABASE DESIGN:



This Database Design Diagram is reduced to its 3rd Normal form and so every reference is made to only the primary key of another table. Many-to-many relationships are taken care of and so are the redundancies.

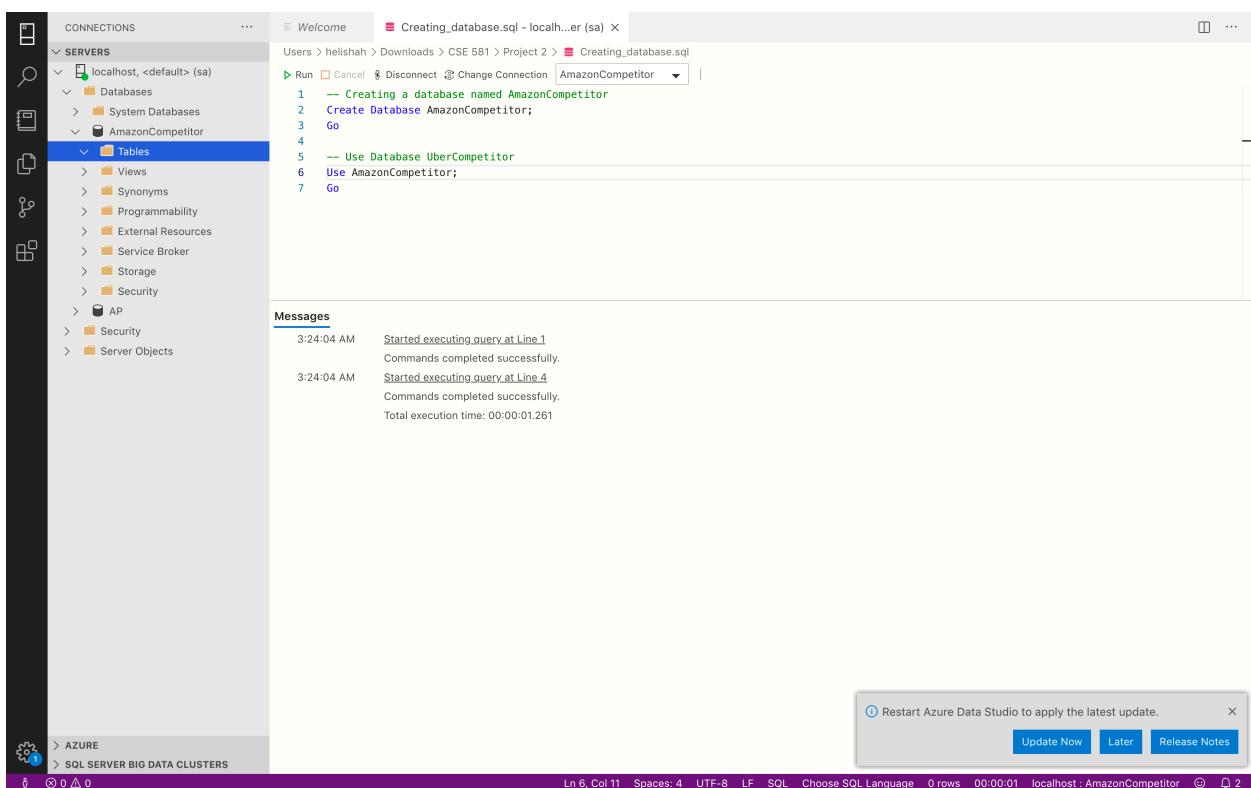
3. IMPLEMENTATION:

In the actual implementation of the above mentioned design for the Amazon Competitor Database is carried out in Azure Data Studio. A new database is created and all the tables are defined with it. After that, all the tables are fed with 10 records each so as to make this database feel as an actual one and perform testing on it.

For each implementation, respective code for explanation is given with the query and the output is also shown at needed places. All this is included in the screenshots below:

3.1 Creation of Database:

The database named AmazonCompetitor is created and then used the same.



A screenshot of the Azure Data Studio interface. On the left, the Object Explorer sidebar shows a tree structure with 'SERVERS' expanded, showing 'localhost, <default> (sa)', 'Databases', 'System Databases', and 'AmazonCompetitor'. 'AmazonCompetitor' is selected and expanded, showing 'Tables', 'Views', 'Synonyms', 'Programmability', 'External Resources', 'Service Broker', 'Storage', and 'Security'. Below this is an 'AP' node and 'Server Objects'. In the center, the main pane displays a SQL editor titled 'Creating_database.sql - localhost (sa)'. The code in the editor is:

```
1 -- Creating a database named AmazonCompetitor
2 Create Database AmazonCompetitor;
3 Go
4
5 -- Use Database UberCompetitor
6 Use AmazonCompetitor;
7 Go
```

Below the editor, the 'Messages' section shows log entries:

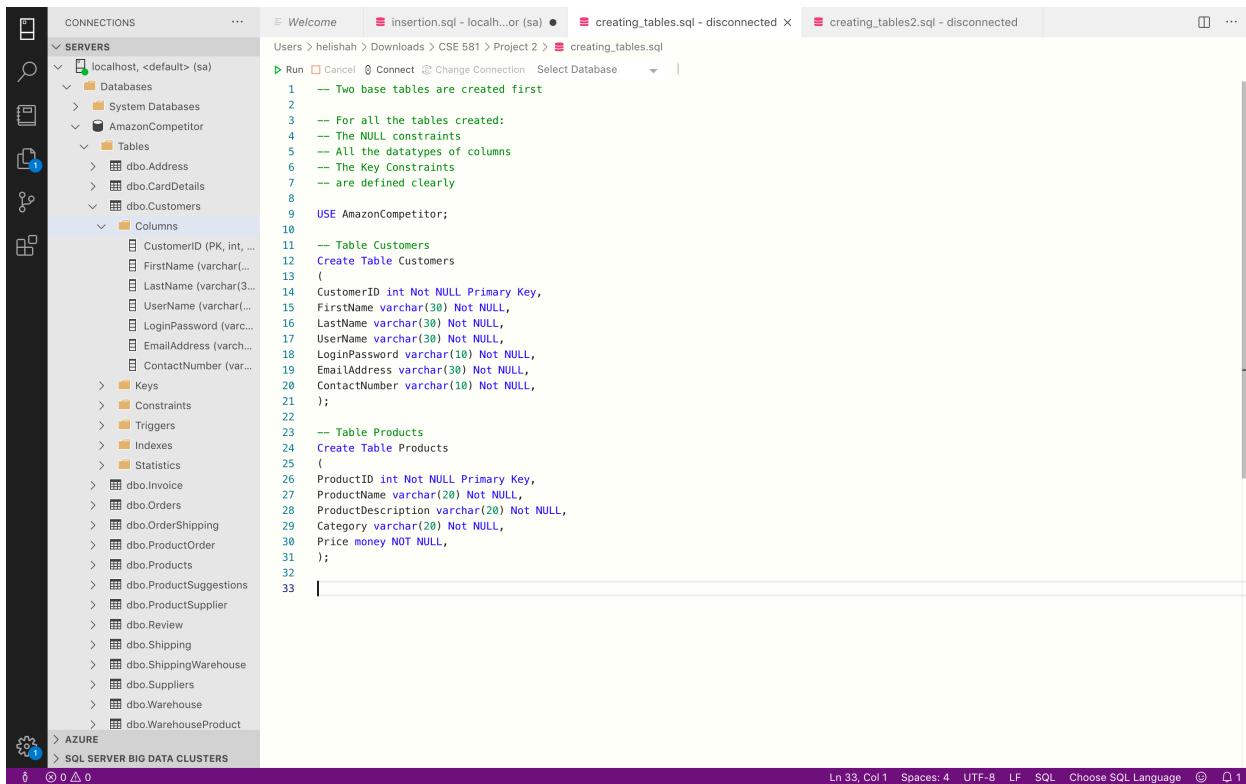
- 3:24:04 AM Started executing query at Line 1
Commands completed successfully.
- 3:24:04 AM Started executing query at Line 4
Commands completed successfully.
Total execution time: 00:00:01.261

At the bottom right, there is a notification bar: 'Restart Azure Data Studio to apply the latest update.' with buttons 'Update Now', 'Later', and 'Release Notes'.

3.2 Creating the Tables:

The necessary tables are created by using the ‘Create Table’ command here the necessary constraints, primary key and foreign key are provided. Also, all the nullability constraint for each table is included.

- A. First, the base tables around which the whole database revolves, are made: Customers and Products



The screenshot shows the SSMS interface. On the left is the Object Explorer tree, which is expanded to show the 'SERVERS' node under 'localhost, <default> (sa)'. It lists several databases, including 'System Databases', 'AmazonCompetitor', and 'dbo' containing 'Address', 'CardDetails', and 'Customers' tables, along with their respective columns, keys, constraints, triggers, indexes, and statistics. Below the main tree, there's a 'AZURE' node and a 'SQL SERVER BIG DATA CLUSTERS' node. On the right is a large query editor window. At the top of the editor, there are tabs for 'Welcome', 'insertion.sql - localh...or (sa)', 'creating_tables.sql - disconnected', and 'creating_tables2.sql - disconnected'. Below the tabs are buttons for 'Run', 'Cancel', 'Connect', 'Change Connection', and 'Select Database'. The main area of the editor contains the following SQL code:

```
1  -- Two base tables are created first
2
3  --- For all the tables created:
4  --- The NULL constraints
5  --- All the datatypes of columns
6  --- The Key Constraints
7  --- are defined clearly
8
9  USE AmazonCompetitor;
10
11 --- Table Customers
12 Create Table Customers
13 (
14 CustomerID int Not NULL Primary Key,
15 FirstName varchar(30) Not NULL,
16 LastName varchar(30) Not NULL,
17 UserName varchar(30) Not NULL,
18 LoginPassword varchar(10) Not NULL,
19 EmailAddress varchar(30) Not NULL,
20 ContactNumber varchar(10) Not NULL,
21 );
22
23 --- Table Products
24 Create Table Products
25 (
26 ProductID int Not NULL Primary Key,
27 ProductName varchar(20) Not NULL,
28 ProductDescription varchar(20) Not NULL,
29 Category varchar(20) Not NULL,
30 Price money NOT NULL,
31 );
32
33 |
```

At the bottom of the editor, there are status indicators: 'Ln 33, Col 1', 'Spaces: 4', 'UTF-8', 'LF', 'SQL', 'Choose SQL Language', and a '1' icon.

B. Creating all the remaining 16 tables keeping in mind their constraints and Foreign References:

The tables created follow a set of **constraints** and some foreign references. They are all depicted in this SQL code:

--

All other tables are created now and given thier respective references according to the design

```
-- For all the tables created:  
-- The NULL constraints  
-- All the datatypes of columns  
-- The Key Constraints  
-- are defined clearly
```

```
USE AmazonCompetitor;
```

```
-- Table Address for storing Customer's Address  
Create Table Address  
(  
AddressID int Not NULL Primary Key,  
CustomerID int Not NULL Foreign Key References Customers(Custome  
rID),  
StreetAddress1 varchar(30) Not NULL,  
StreetAddress2 varchar(30) Not NULL,  
City varchar(30) Not NULL,  
AddressState varchar(10) Not NULL,  
Country varchar(30) Not NULL,  
ZipCode int Not NULL,  
);
```

```
-- Table CardDetails for storing Customer's Payment Information  
Create Table CardDetails  
(  
AddressID int Not NULL Primary Key,
```

```

CustomerID int Not NULL Foreign Key References Customers(CustomerID),
StreetAddress1 varchar(30) Not NULL,
StreetAddress2 varchar(30) Not NULL,
City varchar(30) Not NULL,
AddressState varchar(10) Not NULL,
Country varchar(30) Not NULL,
ZipCode varchar(5) Not NULL,
);

-- Table Invoice for storing Invoice Information for an Order
Create Table Invoice
(
InvoiceID int Not NULL Primary Key,
InvoiceAmount money Not NULL,
InvoiceGenerationDate date Not NULL,
);

-- Table Orders for storing Customer's Order Information
Create Table Orders
(
OrderID int Not NULL Primary Key,
CustomerID int Not NULL Foreign Key References Customers(CustomerID),
OrderStatus varchar(20) Not NULL,
OrderPrice money Not NULL,
OrderPlaceDate date Not NULL,
EstDeliveryDate date Not NULL,
InvoiceID int Not NULL Foreign Key References Invoice(InvoiceID)
,
);
-- Table WishList for storing Customer's WishList Products Information
Create Table WishList

```

```

(
WishListID int Not NULL Primary Key,
CustomerID int Not NULL Foreign Key References Customers(CustomerID),
);

--  

Table WishListMapping for mapping Customer's WishList Information to Product Information  

Create Table WishListMapping
(
WishListMappingID int Not NULL Primary Key,
WishListID int Not NULL Foreign Key References WishList(WishListID),
ProductID int Not NULL Foreign Key References Products(ProductID),
);
  

-- Table ProductSuggestions for storing the mapping of Product-Customer  

-- According to the Products suggested to the Customer  

Create Table ProductSuggestions
(
ProductSuggestionID int Not NULL Primary Key,
CustomerID int Not NULL Foreign Key References Customers(CustomerID),
ProductID int Not NULL Foreign Key References Products(ProductID),
);
  

--  

Table Shipping for storing the shipping Information for an Order  

Create Table Shipping
(
ShippingID int Not NULL Primary Key,

```

```

ShippingCompany varchar(30) Not NULL,
ShippingFare money Not NULL,
OrderID int Not NULL Foreign Key References Orders(OrderID),
);

-- 
Table ProductOrder for mapping the Product Information for an Order
-- There can be
Create Table ProductOrder
(
ProductOrderID int Not NULL Primary Key,
OrderID int Not NULL Foreign Key References Orders(OrderID),
ProductID int Not NULL Foreign Key References Products(ProductID),
),
ProductQuantity int Not NULL,
);

-- 
Table OrderShipping for mapping the Shipping Information for an Order
-- There can be more than one Shipping for just one Order
Create Table OrderShipping
(
ProductOrderID int Not NULL Primary Key,
OrderID int Not NULL Foreign Key References Orders(OrderID),
ProductID int Not NULL Foreign Key References Products(ProductID),
),
);

-- 
Table Reviews for storing the reviews for a particular Product
-- There can be more than one reviews for just one Product
-- 
The comments can be by both concerned parties: the customer and
the product owner

```

```

Create Table Review
(
ReviewID int Not NULL Primary Key,
CustomerID int Not NULL Foreign Key References Orders(OrderID),
ProductID int Not NULL Foreign Key References Products(ProductID
),
Rating int Not NULL,
CustomerNotes varchar(30),
ProductOwnerNotes varchar(30),
);

--  

Table Suppliers for storing the Information about the Suppliers  

for a particular Product
Create Table Suppliers
(
SupplierID int Not NULL Primary Key,
SupplierName varchar(30) Not NULL,
SupplierContact int Not NULL,
SupplierEmail varchar(30) Not NULL,
SupplierAddress varchar(50) Not NULL,
);

--  

Table ProductSupplier for mapping the Suppliers to the Products  

-- Many-to-Many relationship
Create Table ProductSupplier
(
ProductSupplierID int Not NULL Primary Key,
SupplierName varchar(30) Not NULL,
ProductID int Not NULL Foreign Key References Products(ProductID
),
SupplierID int Not NULL Foreign Key References Suppliers(SupplierID),
);

```

```

--  

Table Warehouse for storing the Information about the warehouse  

for a particular product  

-- This is also useful for shipping  

Create Table Warehouse  

(  

WarehouseID int Not NULL Primary Key,  

StreetAddress varchar(30) Not NULL,  

City varchar(30) Not NULL,  

WarehouseState varchar(30) Not NULL,  

ZipCode int Not NULL,  

);  

--  

Table WarehouseProduct for mapping the products to a particular  

warehouse  

-- Same products maybe in multiple warehouses  

Create Table WarehouseProduct  

(  

WarehouseProductID int Not NULL Primary Key,  

WarehouseID int Not NULL Foreign Key References Warehouse(WarehouseID),  

ProductID int Not NULL Foreign Key References Products(ProductID),  

ProductInQuantity int Not NULL,  

ProductOnWayQuantity int Not NULL,  

ReturnQuantity int Not NULL,  

);  

--  

Table ShippingWarehouse for mapping the shipping order to a par  

ticular warehouse  

-- Same orders may have different shipping  

-- But, same shipping must have same warehouses  

Create Table ShippingWarehouse  

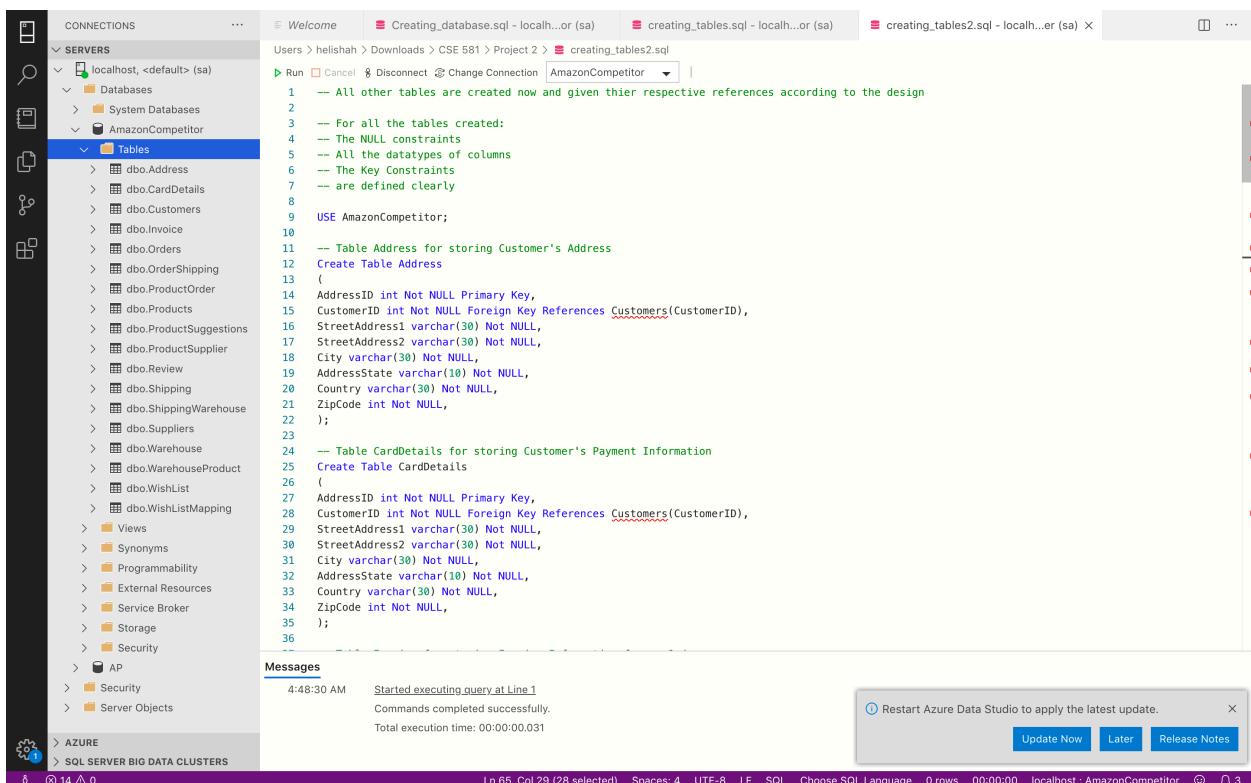
(

```

```

ShippingWarehouseID int Not NULL Primary Key,
WarehouseID int Not NULL Foreign Key References Warehouse(WarehouseID),
ShippingID int Not NULL Foreign Key References Shipping(ShippingID),
);

```



The screenshot shows the Azure Data Studio interface with the following details:

- Left Sidebar (Object Explorer):** Shows the database structure under "AmazonCompetitor". The "Tables" node is selected.
- Center Area (SQL Editor):** Displays the SQL script for creating tables. The script includes comments explaining the design choices and constraints for tables like Address, CardDetails, Customers, and CardDetails.
- Bottom Status Bar:** Provides information about the current session, including the connection name (localhost), the schema (AmazonCompetitor), and the execution status.
- Bottom Right Corner:** A notification box from Microsoft encourages updating Azure Data Studio.

```

-- All other tables are created now and given their respective references according to the design
-- For all the tables created:
-- The NULL constraints
-- All the datatypes of columns
-- The Key Constraints
-- are defined clearly
USE AmazonCompetitor;
-- Table Address for storing Customer's Address
Create Table Address
(
    AddressID int Not NULL Primary Key,
    CustomerID int Not NULL Foreign Key References Customers(CustomerID),
    StreetAddress1 varchar(30) Not NULL,
    StreetAddress2 varchar(30) Not NULL,
    City varchar(30) Not NULL,
    AddressState varchar(10) Not NULL,
    Country varchar(30) Not NULL,
    ZipCode int Not NULL,
);
-- Table CardDetails for storing Customer's Payment Information
Create Table CardDetails
(
    AddressID int Not NULL Primary Key,
    CustomerID int Not NULL Foreign Key References Customers(CustomerID),
    StreetAddress1 varchar(30) Not NULL,
    StreetAddress2 varchar(30) Not NULL,
    City varchar(30) Not NULL,
    AddressState varchar(10) Not NULL,
    Country varchar(30) Not NULL,
    ZipCode int Not NULL,
);

```

The screenshot shows the Azure Data Studio interface with the following details:

- Left Sidebar:** Shows the connection to "localhost, <default> (sa)" under the "Servers" section. The "Tables" node is selected.
- Central Area:** A code editor window displays the SQL script for creating tables. The script includes:
 - Table `Invoice` with columns: `InvoiceID` (int, primary key), `InvoiceAmount` (money), and `InvoiceGenerationDate` (date).
 - Table `Orders` with columns: `OrderID` (int, primary key), `CustomerID` (int, foreign key to `Customers`), `OrderStatus` (varchar(20)), `OrderPrice` (money), `OrderPlaceDate` (date), and `EstDeliveryDate` (date).
 - Table `WishList` with columns: `WishListID` (int, primary key), `CustomerID` (int, foreign key to `Customers`).
 - Table `WishListMapping` with columns: `WishListMappingID` (int, primary key), `WishListID` (int, foreign key to `WishList`), and `ProductID` (int, foreign key to `Products`).
- Messages Panel:** Shows the execution log:
 - Started executing query at Line 1
 - Commands completed successfully.
 - Total execution time: 00:00:00.031
- Bottom Bar:** Includes tabs for "Ln 65, Col 29 (28 selected)", "Spaces: 4", "UTF-8", "LF", "SQL", "Choose SQL Language", "0 rows", "00:00:00", "localhost : AmazonCompetitor", and icons for "Update Now", "Later", and "Release Notes".

The screenshot shows the Azure Data Studio interface with the following details:

- Left Sidebar:** Shows the connection to "localhost, <default> (sa)" under the "Servers" section. The "Tables" node is selected.
- Central Area:** A code editor window displays the SQL script for creating tables. The script includes:
 - Table `ProductSuggestions` with columns: `ProductSuggestionID` (int, primary key), `CustomerID` (int, foreign key to `Customers`), and `ProductID` (int, foreign key to `Products`).
 - Table `Shipping` with columns: `ShippingID` (int, primary key), `ShippingCompany` (varchar(30)), `ShippingFare` (money), and `OrderID` (int, foreign key to `Orders`).
 - Table `ProductOrder` with columns: `ProductOrderID` (int, primary key), `OrderID` (int, foreign key to `Orders`), and `ProductID` (int, foreign key to `Products`).
 - Table `OrderShipping` with columns: `ProductOrderID` (int, primary key), `OrderID` (int, foreign key to `Orders`), and `ProductID` (int, foreign key to `Products`).
- Messages Panel:** Shows the execution log:
 - Started executing query at Line 1
 - Commands completed successfully.
 - Total execution time: 00:00:00.031
- Bottom Bar:** Includes tabs for "Ln 65, Col 29 (28 selected)", "Spaces: 4", "UTF-8", "LF", "SQL", "Choose SQL Language", "0 rows", "00:00:00", "localhost : AmazonCompetitor", and icons for "Update Now", "Later", and "Release Notes".

```

    Welcome   Creating_database.sql - localhost...or (sa)   creating_tables.sql - localhost...or (sa)   creating_tables2.sql - localhost...er (sa)   ...
Users > helishah > Downloads > CSE 581 > Project 2 > creating_tables2.sql
Run Cancel Disconnect Change Connection AmazonCompetitor

107 );
108 -- Table Reviews for storing the reviews for a particular Product
109 -- There can be more than one reviews for just one Product
110 -- The comments can be by both concerned parties: the customer and the product owner
111 Create Table Review
112 (
113     ReviewID int Not NULL Primary Key,
114     CustomerID int Not NULL Foreign Key References Orders(OrderID),
115     ProductID int Not NULL Foreign Key References Products(ProductID),
116     Rating int Not NULL,
117     CustomerNotes varchar(30),
118     ProductOwnerNotes varchar(30),
119 );
120
121
122 -- Table Suppliers for storing the Information about the Suppliers for a particular Product
123 Create Table Suppliers
124 (
125     SupplierID int Not NULL Primary Key,
126     SupplierName varchar(30) Not NULL,
127     SupplierContact int Not NULL,
128     SupplierEmail varchar(30) Not NULL,
129     SupplierAddress varchar(50) Not NULL,
130 );
131
132 -- Table ProductSupplier for mapping the Suppliers to the Products
133 -- Many-to-Many relationship
134 Create Table ProductSupplier
135 (
136     ProductSupplierID int Not NULL Primary Key,
137     SupplierName varchar(30) Not NULL,
138     ProductID int Not NULL Foreign Key References Products(ProductID),
139     SupplierID int Not NULL Foreign Key References Suppliers(SupplierID),
140 );
141
142 -- Table Warehouse for storing the Information about the warehouse for a particular product

```

Messages

4:48:30 AM Started executing query at Line 1
Commands completed successfully.
Total execution time: 00:00:00.031

Restart Azure Data Studio to apply the latest update.

Update Now Later Release Notes

```

    Welcome   Creating_database.sql - localhost...or (sa)   creating_tables.sql - localhost...or (sa)   creating_tables2.sql - localhost...er (sa)   ...
Users > helishah > Downloads > CSE 581 > Project 2 > creating_tables2.sql
Run Cancel Disconnect Change Connection AmazonCompetitor

141
142 -- Table Warehouse for storing the Information about the warehouse for a particular product
143 -- This is also useful for shipping
144 Create Table Warehouse
145 (
146     WarehouseID int Not NULL Primary Key,
147     StreetAddress varchar(30) Not NULL,
148     City varchar(30) Not NULL,
149     WarehouseState varchar(30) Not NULL,
150     ZipCode int Not NULL,
151 );
152
153 -- Table WarehouseProduct for mapping the products to a particular warehouse
154 -- Same products maybe in multiple warehouses
155 Create Table WarehouseProduct
156 (
157     WarehouseProductID int Not NULL Primary Key,
158     WarehouseID int Not NULL Foreign Key References Warehouse(WarehouseID),
159     ProductID int Not NULL Foreign Key References Products(ProductID),
160     ProductInQuantity int Not NULL,
161     ProductOnWayQuantity int Not NULL,
162     ReturnQuantity int Not NULL,
163 );
164
165 -- Table ShippingWarehouse for mapping the shipping order to a particular warehouse
166 -- Same orders may have different shipping
167 -- But, same shipping must have same warehouses
168 Create Table ShippingWarehouse
169 (
170     ShippingWarehouseID int Not NULL Primary Key,
171     WarehouseID int Not NULL Foreign Key References Warehouse(WarehouseID),
172     ShippingID int Not NULL Foreign Key References Shipping(ShippingID),
173 );
174

```

Messages

4:48:30 AM Started executing query at Line 1
Commands completed successfully.
Total execution time: 00:00:00.031

Restart Azure Data Studio to apply the latest update.

Update Now Later Release Notes

With this, the implementation of our database is completed and now we shall go to the phase of testing the database.

4. TESTING THE DATABASE:

4.1 Inserting Values:

Inserting into Customers and Products:

```
USE AmazonCompetitor;
Insert Into Customers values
('1','Carlos','Leal','cgleal','crlgleal','csg@gmail.com','5683959285'),
,
('2','Heli','Shah','hpshah','crlgleal','hps@gmail.com','4532425436'),
('3','Hetal','Bhatia','htshah','bfdgbdfsd','hbha@gmail.com','568986457
85'),
('4','Ishi','Shah','ishah','cbfle','ishh@gmail.com','8753959285'),
('5','Robert','Harris','rbhar','dsfbbf','robbb@gmail.com','9883959285
),
('6','Michael','Singhal','mikeg','vfbfdb','mikeg@gmail.com','345395928
5'),
('7','Aabhas','Perez','aasing','bfbfd','aabhp@gmail.com','0796959285
),
,
('8','John','Shah','jshah','bdfb','jshah@gmail.com','86453959285'),
('9','Paresh','Patel','ppatel','bfbfbd','ppat@gmail.com','5463959285
),
,
('10','Rajan','Shah','rshah','nhgnh','shahr@gmail.com','56895094367');
```

```
Insert Into Products values
```

```
('1','Shampoo','HairWashShampoo','house','25'),
('2','Tomato','HairWashShampoo','grocery','2'),
('3','Tshirt','HairWashShampoo','apparel','56'),
('4','Shoes','HairWashShampoo','apparel','98'),
('5','Belt','HairWashShampoo','accessory','30'),
('6','Conditioner','HairWashShampoo','house','20'),
('7','Tissue','HairWashShampoo','house','5'),
('8','Paper','HairWashShampoo','stationery','1'),
('9','Pen','HairWashShampoo','stationry','4'),
('10','Paracetamol','HairWashShampoo','mdicine','9');
```

Insertion in Products and Customers results:

The screenshot shows the SSMS interface with the following details:

- Connections:** localhost, <default> (sa)
- Servers:** localhost, <default> (sa) - expanded to show Databases, System Databases, AmazonCompetitor, Tables, dbo.Address, dbo.CardDetails, dbo.Customers, Columns, Keys, Constraints, Triggers, Indexes, Statistics, dbo.Invoice, dbo.Orders, dbo.OrderShipping, dbo.ProductOrder, dbo.Products, dbo.ProductSuggestions, dbo.ProductSupplier, dbo.Review, dbo.Shipping, dbo.ShippingWarehouse, dbo.Suppliers, dbo.Warehouse, dbo.WarehouseProduct.
- Script:**

```

1 USE AmazonCompetitor;
2
3 Insert Into Customers values
4 ('1','Carlos','Leal','cgleal','csg@gmail.com','5683959285'),
5 ('2','Heli','Shah','hpshan','crlgleal','hps@gmail.com','4532425436'),
6 ('3','Hetal','Bhatia','htshah','bfdfdbfsd','hbha@gmail.com','5689864575'),
7 ('4','Ishi','Shah','ishah','cbfle','ishh@gmail.com','8753959285'),
8 ('5','Robert','Harris','rbhar','dsfbfb','robbb@gmail.com','9883959285'),
9 ('6','Michael','Singhal','mikeg','vfbfdb','mikeg@gmail.com','3453959285'),
10 ('7','Abbas','Perez','aasing','bfbfd','aabhp@gmail.com','0796959285'),
11 ('8','John','Shah','jshah','bdfb','jshah@gmail.com','8645395925'),
12 ('9','Paresh','Patel','ppatel','bfbfdb','ppat@gmail.com','5463959285'),
13 ('10','Rajan','Shah','rshah','nhgnh','shahr@gmail.com','5689509437');
14
15 SELECT * FROM Customers;
16
17 Insert Into Products values
18 ('1','Shampoo','HairWash','house','25'),
19 ('2','Tomato','Vegetables','grocery','2'),
20 ('3','Tshirt','Clothing upper','apparel','56'),
21 ('4','Shoes','Feet wear','apparel','98'),
22 ('5','Belt','accessories','accessory','30'),
23 ('6','Conditioner','HairWash','house','20'),
24 ('7','Tissue','Cleaning','house','5'),
25 ('8','Paper','Writing','stationery','1'),
26 ('9','Pen','Writing','stationery','4'),
27 ('10','Paracetamol','Medical','medicine','9');
28
29 SELECT * FROM Products;

```
- Results:** Started executing query at Line 1
4:22:51 PM
(10 rows affected)
(10 rows affected)
(10 rows affected)
(10 rows affected)
Total execution time: 00:00:00.007
- Messages:**
- Status Bar:** Ln 25, Col 42 | Spaces: 4 | UTF-8 | LF | SQL | Choose SQL Language | 20 rows | 00:00:00 | localhost : AmazonCompetitor | 0 1

The screenshot shows the SSMS interface with the following details:

- Connections:** localhost, <default> (sa)
- Servers:** localhost, <default> (sa) - expanded to show Databases, System Databases, AmazonCompetitor, Tables, dbo.Address, dbo.CardDetails, dbo.Customers, Columns, Keys, Constraints, Triggers, Indexes, Statistics, dbo.Invoice, dbo.Orders, dbo.OrderShipping, dbo.ProductOrder, dbo.Products, dbo.ProductSuggestions, dbo.ProductSupplier, dbo.Review, dbo.Shipping, dbo.ShippingWarehouse, dbo.Suppliers, dbo.Warehouse, dbo.WarehouseProduct.
- Results:**

CustomerID	FirstName	LastName	UserName	LoginPassword	EmailAddress	ContactNumber
1	Carlos	Leal	cgleal	crgleal	csg@gmail.com	5683959285
2	Heli	Shah	hpshan	crlgleal	hps@gmail.com	4532425436
3	Hetal	Bhatia	htshah	bfdfdbfsd	hbha@gmail.com	5689864575
4	Ishi	Shah	ishah	cbfle	ishh@gmail.com	8753959285
5	Robert	Harris	rbhar	dsfbfb	robbb@gmail.com	9883959285
6	Michael	Singhal	mikeg	vfbfdb	mikeg@gmail.com	3453959285
7	Abbas	Perez	aasing	bfbfd	aabhp@gmail.com	0796959285
8	John	Shah	jshah	bdfb	jshah@gmail.com	8645395925
9	Paresh	Patel	ppatel	bfbfdb	ppat@gmail.com	5463959285
10	Rajan	Shah	rshah	nhgnh	shahr@gmail.com	5689509437

ProductID	ProductName	ProductDescription	Category	Price
1	Shampoo	HairWash	house	25.0000
2	Tomato	Vegetables	grocery	2.0000
3	Tshirt	Clothing upper	apparel	56.0000
4	Shoes	Feet wear	apparel	98.0000
5	Belt	accessories	accessory	30.0000
6	Conditioner	HairWash	house	20.0000
7	Tissue	Cleaning	house	5.0000
8	Paper	Writing	stationery	1.0000
9	Pen	Writing	stationery	4.0000
10	Paracetamol	Medical	medicine	9.0000
- Messages:**
- Status Bar:** Ln 25, Col 42 | Spaces: 4 | UTF-8 | LF | SQL | Choose SQL Language | 20 rows | 00:00:00 | localhost : AmazonCompetitor | 0 1

The insert statement format for a table:

The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. On the left, the Object Explorer displays a tree view of the database structure under 'localhost, <default> (sa)'. The 'Tables' node is expanded, showing 'Address' as the only table. The 'Security' and 'Server Objects' nodes are also visible. The central pane shows a query window titled 'insertion2.sql - localhost...or (sa)'. The query itself is a script for inserting data into the 'Address' table, starting with a comment to insert into all other tables and then listing 16 rows of address data. Below the query, the 'Messages' section shows the execution log: it started at 4:55:16 PM, executed at Line 1, affected 10 rows, and took a total execution time of 00:00:00.014. At the bottom, there are status indicators for Azure and SQL Server Big Data Clusters, and a toolbar with various icons.

CONNECTIONS ...

SERVERS

localhost, <default> (sa)

> Databases

> Security

> Server Objects

Welcome insertion.sql - localhost...or (sa) insertion2.sql - localhost...or (sa) creating_tables2.sql - disconnected

Run Cancel Disconnect Change Connection AmazonCompetitor

```
1 -- Inserting into all other tables now
2 USE AmazonCompetitor;
3
4 Insert Into Address values
5 ('1','3','315 Greenwood Place','#2','Syracuse','New York','USA', '13210'),
6 ('2','2','643 Applewood Place','#4','New York','New York','USA', '42242'),
7 ('3','1','643 OrangePark Place','#6','Edison','New Jersey','USA', '53231'),
8 ('4','3','234 Concord Place','#8','Miami','Florida','USA', '65423'),
9 ('5','6','543 Lalala Place','#1','Miami','Florida','USA', '87324'),
10 ('6','7','32 Oakwood Place','#9','Tampa','Florida','USA', '56738'),
11 ('7','8','434 Applewood Place','#20','Los Angeles','California','USA', '87632'),
12 ('8','9','5 Asgard Place','#52','Seattle','Washington','USA', '65738'),
13 ('9','10','433 Capital Place','#76','Seattle','Washington','USA', '34567'),
14 ('10','4','453 Toral Place','#92','Chicago','Illinois','USA', '87125');
15
16
17
```

Messages

4:55:16 PM Started executing query at Line 1
(10 rows affected)
Total execution time: 00:00:00.014

AZURE

SQL SERVER BIG DATA CLUSTERS

Ln 16, Col 1 Spaces: 4 UTF-8 LF SQL Choose SQL Language 0 rows 00:00:00 localhost : AmazonCompetitor

Inserting values into all other remaining tables:

The screenshot shows the SSMS interface with the following details:

- Left Sidebar:** Shows the "CONNECTIONS" node under "SERVERS". The "AmazonCompetitor" server is expanded, showing "Tables", "Columns", "Keys", "Constraints", "Triggers", "Indexes", "Statistics", "Views", and "Synonyms".
- Top Bar:** Displays four tabs: "Welcome", "insertion.sql - localhost\sa (sa)", "insertion2.sql - localhost\sa (sa)", and "creating_tables2.sql - disconnected".
- Toolbar:** Includes icons for New Query, Run, Cancel, Disconnect, Change Connection, and Save.
- Query Editor:** Contains the following T-SQL script:

```
1 --- Inserting into all other tables now
2 USE AmazonCompetitor;
3
4 SELECT * FROM Address;
5 SELECT * FROM CardDetails;
6 SELECT * FROM Invoice;
7 SELECT * FROM Orders;
8 SELECT * FROM WishList;
9 SELECT * FROM WishListMapping;
10 SELECT * FROM ProductSuggestions;
11 SELECT * FROM Shipping;
12 SELECT * FROM ProductOrder;
13 SELECT * FROM Review;
14 SELECT * FROM Suppliers;
15 SELECT * FROM ProductSupplier;
16 SELECT * FROM Warehouse;
17 SELECT * FROM WarehouseProduct;
18 SELECT * FROM ShippingWarehouse;
19
```
- Results Tab:** Shows the execution log:

Time	Message
6:27:46 PM	Started executing query at Line 1
	(10 rows affected)
	Total execution time: 00:00:00.069
- Status Bar:** Shows the number of rows (150), execution time (00:00:00.069), and the connection string (localhost\AmazonCompetitor).

While implementing the database tables, the Foreign Key constraints have been implemented correctly. Here is a proof that where the constraints are enforced, SQL won't allow any incorrect value:

Here, for the table WarehouseProduct, the third Column is referenced to ProductID of Products table and '58' is not an entry of any ProductID in Products table (can refer the above screenshot of Products table entries). So, it gave the error for Foreign Key constraint.

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer tree shows the database structure under 'AmazonCompetitor'. In the center, the 'Insertion2.sql' query window contains the following SQL code:

```
18 Insert Into WarehouseProduct values
19 ('1','5','1','4','6','1'),
20 ('2','2','4','2','6','9'),
21 ('3','3','2','4','6','2'),
22 ('4','1','4','6','8','1'),
23 ('5','8','5','7','3','1'),
24 ('6','5','8','8','7','4'),
25 ('7','2','5','1','9','1'),
26 ('8','3','3','3','2','1'),
27 ('9','6','58','5','5','5'),
28 ('10','6','6','4','8','1');
```

Line 28, which attempts to insert a row with ProductID '58', is highlighted in red. Below the code, the 'Messages' pane shows the execution log:

```
Started executing query at Line 1
Msg 547, Level 16, State 0, Line 18
The INSERT statement conflicted with the FOREIGN KEY constraint "FK_Warehouse_Produc_114A936A". The conflict occurred in database "AmazonCompetitor", table "dbo.WarehouseProduct".
The statement has been terminated.
Total execution time: 0:00:00.001
```

4.2 Stored Procedures:

4.2.1 This procedure gives us the products rated between the max and min rating provided. While executing, the max or min or both ratings are provided.

The screenshot shows the SSMS interface with the following details:

- Connections:** Localhost, <default> (sa)
- Servers:** AmazonCompetitor
- Object Explorer:** Shows the database structure including Tables, Views, Synonyms, and Programmability (Stored Procedures).
- Code Editor:** Procedure1.sql (highlighted in red). The code defines a stored procedure that returns products with ratings between specified values or displays the maximum rating if no minimum is provided.
- Messages:** Displays log entries for the execution of the stored procedure.
- Status Bar:** Ln 13, Col 47, Spaces: 4, UTF-8, LF, SQL, Choose SQL Language, 0 rows, 00:00:00, localhost:AmazonCompetitor.

The screenshot shows the SSMS interface with the following details:

- Connections:** Localhost, <default> (sa)
- Servers:** AmazonCompetitor
- Object Explorer:** Shows the database structure including Tables, Views, Synonyms, and Programmability (Stored Procedures).
- Code Editor:** ProcedureOutput.sql (highlighted in red). The code executes the HighRatedProduct stored procedure with different parameters and shows the results.
- Results Grid:** Displays the output of the stored procedure, showing product reviews and their details.
- Status Bar:** Ln 3, Col 41, Spaces: 4, UTF-8, LF, SQL, Choose SQL Language, 16 rows, 00:00:00, localhost:AmazonCompetitor.

4.2.2 This procedure gives details of all the suppliers of a particular product

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, <default> (sa)
- Object Explorer:** Shows the database structure for 'AmazonCompetitor' including Tables, Views, Synonyms, and Stored Procedures.
- Script Editor:** Contains the T-SQL code for creating the 'SupplierOfProduct' stored procedure.
- Messages:** Displays the execution log with the following entries:
 - 7:45:11 PM Started executing query at Line 1
 - Commands completed successfully.
 - 7:45:11 PM Started executing query at Line 3
 - Commands completed successfully.
 - Total execution time: 00:00:00.006
- Status Bar:** Shows the connection status as 'localhost - AmazonCompetitor'.

```

USE AmazonCompetitor;
GO
CREATE Procedure SupplierOfProduct @ProductName VARCHAR(50) = NULL
AS
SELECT P.ProductName, S.SupplierID, SupplierName, SupplierEmail
FROM ProductSupplier AS PS JOIN Products AS P ON P.ProductID = PS.ProductID
JOIN Suppliers AS S ON PS.SupplierID = S.SupplierID
WHERE P.ProductName = @ProductName;
GO
-- EXEC SupplierOfProduct @ProductName = 'Columbia Boots';

```

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, <default> (sa)
- Object Explorer:** Shows the database structure for 'AmazonCompetitor'.
- Script Editor:** Contains the T-SQL code for executing the 'SupplierOfProduct' stored procedure with the parameter '@ProductName = 'Tshirt''. It also includes comments for correctness and multiple SELECT statements to show the data flow.
- Results Grid:** Displays the output of the stored procedure execution, showing three rows of supplier information for 'Tshirt'.
- Results Grid:** Displays the 'Products' table with 10 rows of product data.
- Results Grid:** Displays the 'ProductSupplier' table with 10 rows of supplier details.
- Status Bar:** Shows the connection status as 'localhost - AmazonCompetitor'.

```

EXEC SupplierOfProduct @ProductName = 'Tshirt';
-- for referring the correctness of result
SELECT * FROM Products;
SELECT * FROM ProductSupplier;
SELECT * FROM Suppliers;

```

	ProductName	SupplierID	SupplierName	SupplierEmail
1	Tshirt	6	Dhruv Patel	dfds@gmail.com
2	Tshirt	8	John Cena	gfs@gmail.com
3	Tshirt	2	Heli Patel	fr3@gmail.com

	ProductID	ProductName	ProductDescription	Category	Price
1	1	Shampoo	HairWash	house	25.0000
2	2	Tomato	Vegetables	grocery	2.0000
3	3	Tshirt	Clothing upper	apparel	56.0000
4	4	Shoes	Feet wear	apparel	98.0000
5	5	Belt	accessories	accessory	30.0000
6	6	Conditioner	HairWash	house	20.0000
7	7	Tissue	Cleaning	house	5.0000
8	8	Paper	Writing	stationery	1.0000
9	9	Pen	Writing	stationery	4.0000
10	10	Paracetamol	Medical	medicine	9.0000

	ProductSupplierID	ProductID	SupplierID
1	1	5	1
2	2	1	3
3	3	3	6
4	4	3	8
5	5	2	3
6	6	6	5
7	7	3	2
8	8	6	9

4.2.3 This procedure lists the customers living in the state New York

```

USE AmazonCompetitor;
GO
CREATE Procedure CustomerAddress @State VARCHAR (30) = NULL
AS
SELECT C.CustomerID, StreetAddress1, City, ZipCode
FROM Address AS A JOIN Customers AS C ON A.CustomerID = C.CustomerID
WHERE A.AddressState = @State;

```

Messages

- 7:53:56 PM Started executing query at Line 1
- Commands completed successfully.
- 7:53:56 PM Started executing query at Line 3
- Commands completed successfully.
- Total execution time: 00:00:00.011

```

EXEC CustomerAddress 'New York';
-- for checking the correctness of output:
SELECT * FROM Address;

```

	CustomerID	StreetAddress1	City	ZipCode
1	3	315 Greenwood Place	Syracuse	13210
2	2	643 Applewood Place	New York	42242

	AddressID	CustomerID	StreetAddress1	StreetAddress2	City	AddressState	Country	ZipCode
1	1	3	315 Greenwood Place	#2	Syracuse	New York	USA	13210
2	2	2	643 Applewood Place	#4	New York	New York	USA	42242
3	3	1	643 OrangePark Place	#6	Edison	New Jersey	USA	53231
4	4	3	234 Concord Place	#8	Miami	Florida	USA	65423
5	5	6	543 Lalala Place	#1	Miami	Florida	USA	87324
6	6	7	32 Oakwood Place	#89	Tampa	Florida	USA	56738
7	7	8	434 Applewood Place	#20	Los Angeles	California	USA	87632
8	8	9	5 Asgard Place	#52	Seattle	Washington	USA	65738
9	9	10	433 Capital Place	#76	Seattle	Washington	USA	34567
10	10	4	453 Toral Place	#92	Chicago	Illinois	USA	87125

4.2.4 This procedure gives all the customers those have more than 2 products in their WishList

```

USE AmazonCompetitor;
GO
-- StoredProcedure to find the customers that have atleast twi items in thier WishList
CREATE PROCEDURE BigWishList
AS
BEGIN
Select C.CustomerID, FirstName+' '+LastName AS CustomerName
From Wishlist AS W Join Customers AS C ON W.CustomerId = C.CustomerId
Join
WishlistMapping AS WM ON W.WishListID = WM.WishListID
WHERE WM.WishListID>1
END

```

Messages

- 8:17:58 PM Started executing query at Line 1
Commands completed successfully.
- 8:17:58 PM Started executing query at Line 3
Commands completed successfully.
Total execution time: 00:00:00.010

LN 9, Col 31 Spaces: 4 UFT-8 LF SQL Choose SQL Language 0 rows 00:00:00 localhost:AmazonCompetitor

CustomerID	CustomerName
2	Heli Shah
3	Hetal Bhatia
4	Ishi Shah
4	Ishi Shah
5	Robert Harris
6	Michael Singhal
7	Rajan Shah

LN 4, Col 18 Spaces: 4 UFT-8 LF SQL Choose SQL Language 7 rows 00:00:00 localhost:AmazonCompetitor

4.3 TRANSACTIONS:

4.3.1

This transaction updates the OrderPrice for a particular OrderID. It just basically makes sure that it changes it and then allows any other statement to access that record.

The screenshot shows the SSMS interface with a query window containing a transaction script. The script drops existing triggers if they exist, creates a new trigger named 'tr_Orders_Update' to update the OrderPrice, and then updates the 'Orders' table for OrderID 8. The 'Messages' pane shows the execution log with timestamps and command completion status. The status bar at the bottom indicates the connection is to 'localhost : AmazonCompetitor'.

```
1 USE AmazonCompetitor;;
2 GO
3
4 IF EXISTS (SELECT * FROM sys.triggers WHERE object_id = OBJECT_ID('tr_Orders_Update'))
5 DROP TRIGGER [dbo].[tr_Orders_Update];
6
7 IF EXISTS (SELECT * FROM sys.triggers WHERE object_id = OBJECT_ID('trInsertDisplay'))
8 DROP TRIGGER [dbo].[trInsertDisplay];
9 GO
10 BEGIN TRAN;
11
12 UPDATE Orders
13 SET OrderPrice = 1000
14 WHERE OrderID = 8
15
16 COMMIT TRAN;
17
```

Messages

```
10:32:57 PM Started executing query at Line 1
Commands completed successfully.
10:32:57 PM Started executing query at Line 3
Commands completed successfully.
10:32:57 PM Started executing query at Line 10
(1 row affected)
Total execution time: 00:00:00.021
```

AZURE > SQL SERVER BIG DATA CLUSTERS

LN 13, COL 7 SPACES: 4 UFT-8 LF SQL Choose SQL Language 0 rows 00:00:00 localhost : AmazonCompetitor

The following row is the changed record:

The screenshot shows a query window with a single SELECT statement that retrieves all columns for the row where OrderID equals 8. The results pane displays the data, with the 'OrderPrice' column value highlighted in blue, indicating it has been modified.

```
1 USE AmazonCompetitor;;
2 GO
3
4
5 SELECT * FROM Orders WHERE OrderID=8;
```

Results

	OrderID	CustomerID	OrderStatus	OrderPrice	OrderPlaceDate	EstDeliveryDate	Invoi
1	8	10	Complete	1000.0000	2019-05-23	2019-06-01	10

4.4 VIEWS AND REPORTS USEFUL FOR THE BUSINESS:

All these views can be also taken in regards as a report that is important for the business analysis for a company like Amazon. Many important patterns about customers-product relation can be deduced from these reports/views.

4.4.1

This view Customer Orders gives us the information about all those customers who have placed an order along with their order details.

The screenshot shows the SSMS interface with the 'Views' node selected in the Object Explorer. A new view named 'CustomerOrders' is being created in the 'AmazonCompetitor' database. The code pane contains the following SQL script:

```
USE AmazonCompetitor;
GO
-- This view displays all the customers who placed an order and their respective order details
CREATE VIEW CustomerOrders
AS
SELECT FirstName + ' ' + LastName AS CustomerName, OrderID, OrderPrice, OrderStatus
FROM Customers AS C JOIN Orders AS A ON C.CustomerID = A.CustomerID
GO
SELECT * FROM CustomerOrders;
```

The results pane displays the data returned by the query:

	CustomerName	OrderID	OrderPrice	OrderStatus
1	Carlos Leal	1	126.0000	InComplete
2	Carlos Leal	2	738.0000	Complete
3	Aabhas Perez	3	4872.0000	Complete
4	Heli Shah	4	837.0000	InComplete
5	Aabhas Perez	5	210.0000	Complete
6	Parekh Patel	6	456.0000	InComplete
7	Hetal Bhatia	7	859.0000	Complete
8	Rajan Shah	8	1240.0000	Complete
9	Carlos Leal	9	657.0000	InComplete
10	Heli Shah	10	768.0000	Complete

4.4.2

This view Top3RatedProducts extracts the products from Reviews that have the top rating and outputs the Product Number.

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, <default> (sa)
- Connections:** sql - localh...or (sa), Function1.sql - localh...or (sa), Function2.sql - disconnected, View1.sql - localh...or (sa), View2.sql - localh...er (sa)
- Object Explorer:** Shows the database structure under "AmazonCompetitor". The "Views" node is expanded, and "Top3RatedProducts" is selected.
- Scripting:** A script window titled "View2.sql" contains the following T-SQL code:

```
1 USE AmazonCompetitor;
2 GO
3
4 -- Describes the count of Products that are maximum rated from User's ratings
5 CREATE VIEW Top3RatedProducts
6 AS
7 SELECT TOP 3 COUNT(R.ProductID) AS ProductID
8 FROM Review AS R JOIN Products AS P ON R.ProductID = P.ProductID
9 GROUP BY P.ProductID
10 ORDER BY COUNT(R.Rating) DESC;
11 GO
12
13 SELECT * FROM Top3RatedProducts;
```

- Results:** A table titled "Results" shows the output of the query:

ProductID
1
2
3

- Status Bar:** Shows "Ln 4, Col 78" and "localhost : AmazonCompetitor".

4.4.3

This View describes the Details of a Customer and his/her ratings for a particular product and the notes for the product.

The screenshot shows the SSMS interface with the following details:

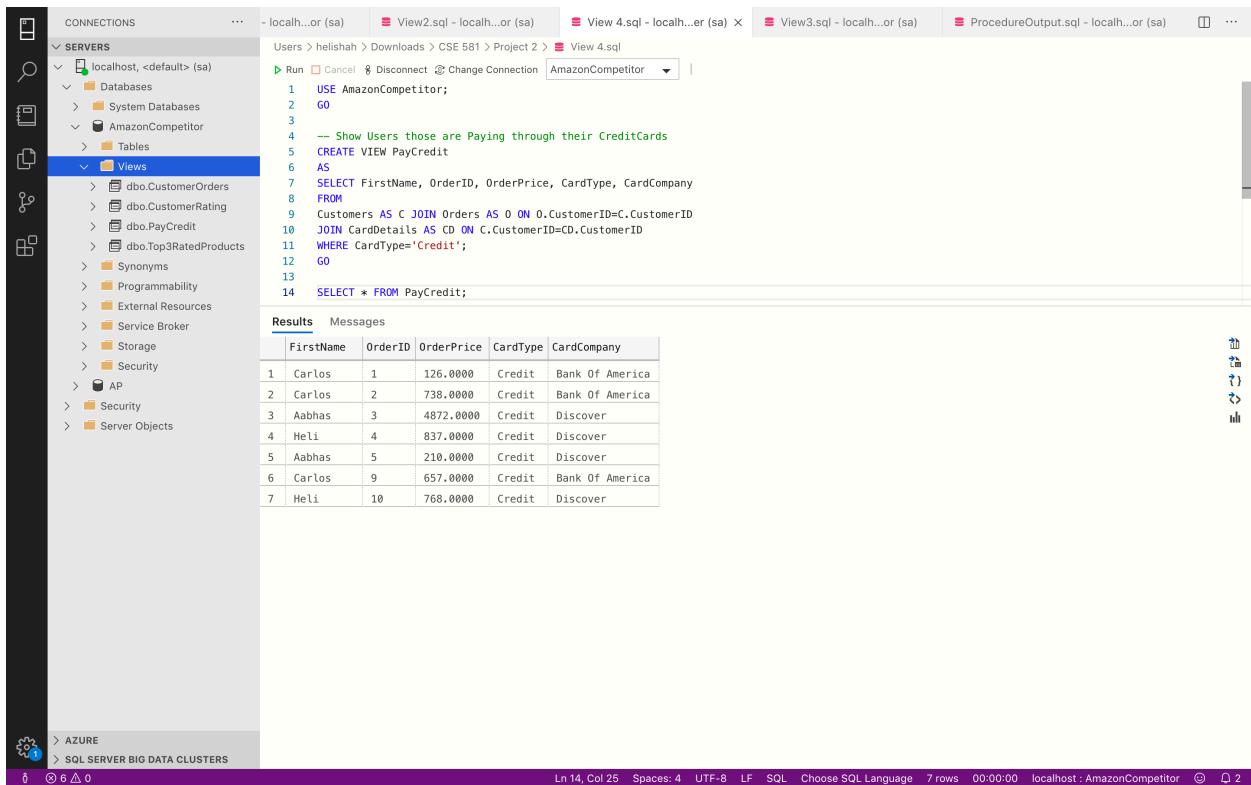
- Servers:** localhost, <default> (sa)
- Views:** The 'Views' node is selected in the Object Explorer.
- SQL Editor:** The code pane contains the following T-SQL script:

```
USE AmazonCompetitor;
GO
-- Customer Rating Details View
CREATE VIEW CustomerRating
AS
SELECT C.CustomerID, FirstName + ' ' + LastName AS CustomerName, Rating,
CustomerNotes, ProductName
FROM Review AS R JOIN Customers AS C ON C.CustomerID = R.CustomerID JOIN
Products AS P ON R.ProductID = P.ProductID;
GO
SELECT * FROM CustomerRating;
```
- Results Pane:** The results of the query are displayed in a table:

	CustomerID	CustomerName	Rating	CustomerNotes	ProductName
1	4	Ishi Shah	8	Excellent product	Tissue
2	8	John Shah	2		Pen
3	1	Carlos Leal	9	Excellent product	Shampoo
4	2	Heli Shah	1	Poor Quality	Tomato
5	3	Hetal Bhatia	10	Excellent product	Tshirt
6	1	Carlos Leal	9		Tomato
7	5	Robert Harris	4		Tshirt
8	4	Ishi Shah	2	Bad	Conditioner
9	2	Heli Shah	6	Okish	Tomato
10	7	Aabhas Perez	7	Fairly good	Tissue
- Status Bar:** Shows the following information: Line 4, Col 27, Spaces: 4, UTF-8, LF, SQL, Choose SQL Language, 10 rows, 00:00:00, localhost : AmazonCompetitor, and a session ID of 2.

4.4.4

This view gives in the output the customers and their orders who have placed an order and that have their payment method set as Credit Card for that Order.



The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, <default> (sa)
- Connections:** localh...or (sa), View2.sql - localh...or (sa), View 4.sql - localh...er (sa) X, View3.sql - localh...or (sa), ProcedureOutput.sql - localh...or (sa)
- Object Explorer:** SERVERS node expanded, showing Databases, System Databases, AmazonCompetitor, Tables, Views (selected), CustomerOrders, CustomerRating, PayCredit, Top3RatedProducts, Synonyms, Programmability, External Resources, Service Broker, Storage, Security, AP, Security, and Server Objects.
- Query Editor:** USE AmazonCompetitor; GO; -- Show Users those are Paying through their CreditCards; CREATE VIEW PayCredit AS SELECT FirstName, OrderID, OrderPrice, CardType, CardCompany FROM Customers AS C JOIN Orders AS O ON O.CustomerID=C.CustomerID JOIN CardDetails AS CD ON C.CustomerID=CD.CustomerID WHERE CardType='Credit'; GO; SELECT * FROM PayCredit;
- Results Grid:** A table showing the results of the query. The columns are FirstName, OrderID, OrderPrice, CardType, and CardCompany. The data is as follows:

	FirstName	OrderID	OrderPrice	CardType	CardCompany
1	Carlos	1	126.0000	Credit	Bank Of America
2	Carlos	2	738.0000	Credit	Bank Of America
3	Aabhas	3	4872.0000	Credit	Discover
4	Heli	4	837.0000	Credit	Discover
5	Aabhas	5	210.0000	Credit	Discover
6	Carlos	9	657.0000	Credit	Bank Of America
7	Heli	10	768.0000	Credit	Discover

4.5 FUNCTIONS:

4.5.1 Using in-built Aggregate Functions to generate important statistical results. Here, these functions generate the cheapest order and the costliest order placed.

The screenshot shows the Object Explorer on the left with the path: SERVERS > localhost, <default> (sa) > AmazonCompetitor > Functions > Scalar-valued Functions. The right pane displays the following T-SQL code:

```
1 USE AmazonCompetitor;
2 GO
3
4 CREATE FUNCTION fnRating()
5 RETURNS INT
6 BEGIN
7 RETURN
8 (SELECT MIN(Rating) FROM Review)
9 END
```

The Messages pane at the bottom shows the execution results:

```
8:29:56 PM Started executing query at Line 1
Commands completed successfully.
8:29:56 PM Started executing query at Line 3
Commands completed successfully.
Total execution time: 00:00:00.014
```

The screenshot shows the Object Explorer on the left with the path: SERVERS > localhost, <default> (sa) > AmazonCompetitor > Tables > dbo.Orders. The right pane displays the following T-SQL code:

```
1 USE AmazonCompetitor;
2 GO
3
4 -- Using 2 predefined aggregate funtions
5 --- This query makes use of built-in aggregate functions like MIN and MAX
6 --- Simple select statement is executed to get costliest order and least cost order placed by a customer.
7
8 SELECT MAX(OrderPrice) AS CostliestOrder,
9 MIN(OrderPrice) AS CheapestOrder
10 FROM Orders;
11
12
```

The Results pane shows the output of the query:

	CostliestOrder	CheapestOrder
1	4872.0000	126.0000

4.5.2

The user defined function fnPoorRating defines a function that returns the value of minimum rating and then this rating can be used for some task. (Maybe to improve the quality of the product, or to satisfy the customer).

The screenshot shows the SSMS interface with the following details:

- Servers:** Localhost, <default> (sa)
- Databases:** AmazonCompetitor
- Functions:** Scalar-valued Functions
- Function Definition:**

```
1 USE AmazonCompetitor;
2 GO
3
4 CREATE FUNCTION fnPoorRating()
5 RETURNS INT
6 BEGIN
7     RETURN
8     (SELECT MIN(Rating) FROM Review)
9 END
```
- Messages:**
 - 10:06:48 PM Started executing query at Line 1
 - Commands completed successfully.
 - 10:06:48 PM Started executing query at Line 3
 - Commands completed successfully.
 - Total execution time: 00:00:00.009
- Status Bar:** Line 5, Col 9, Spaces: 4, UTF-8, LF, SQL, Choose SQL Language, 0 rows, 00:00:00, localhost - AmazonCompetitor, 2

Then, in this select statement, the above defined function is called and used to generate the customer information and his/her relevant product review information.

The screenshot shows the SSMS interface. The Object Explorer on the left lists the database structure under 'localhost, <default> (sa)'. The 'Functions' node is expanded, showing 'Scalar-valued Functions' and 'dbo.fnPoorRating'. The 'dbo.fnPoorRating' function is selected. The central pane contains a query window with the following code:

```
1 USE AmazonCompetitor;
2 GO
3 -- Using the User defined function fnPoorRating that we just created.
4 -- In this SELECT Query to use the return value from that function
5
6 SELECT C.CustomerID, FirstName, ReviewID, CustomerNotes, Rating
7 FROM Customers AS C JOIN Review AS R ON C.CustomerID=R.CustomerID
8 WHERE Rating=dbo.fnPoorRating();
```

The 'Results' tab is selected, displaying the output of the query:

	CustomerID	FirstName	ReviewID	CustomerNotes	Rating
1	2	Heli	4	Poor Quality	1

The status bar at the bottom shows: Line 6, Col 7, Spaces: 4, UTF-8, LF, SQL, Choose SQL Language, 1 rows, 00:00:00, localhost - AmazonCompetitor.

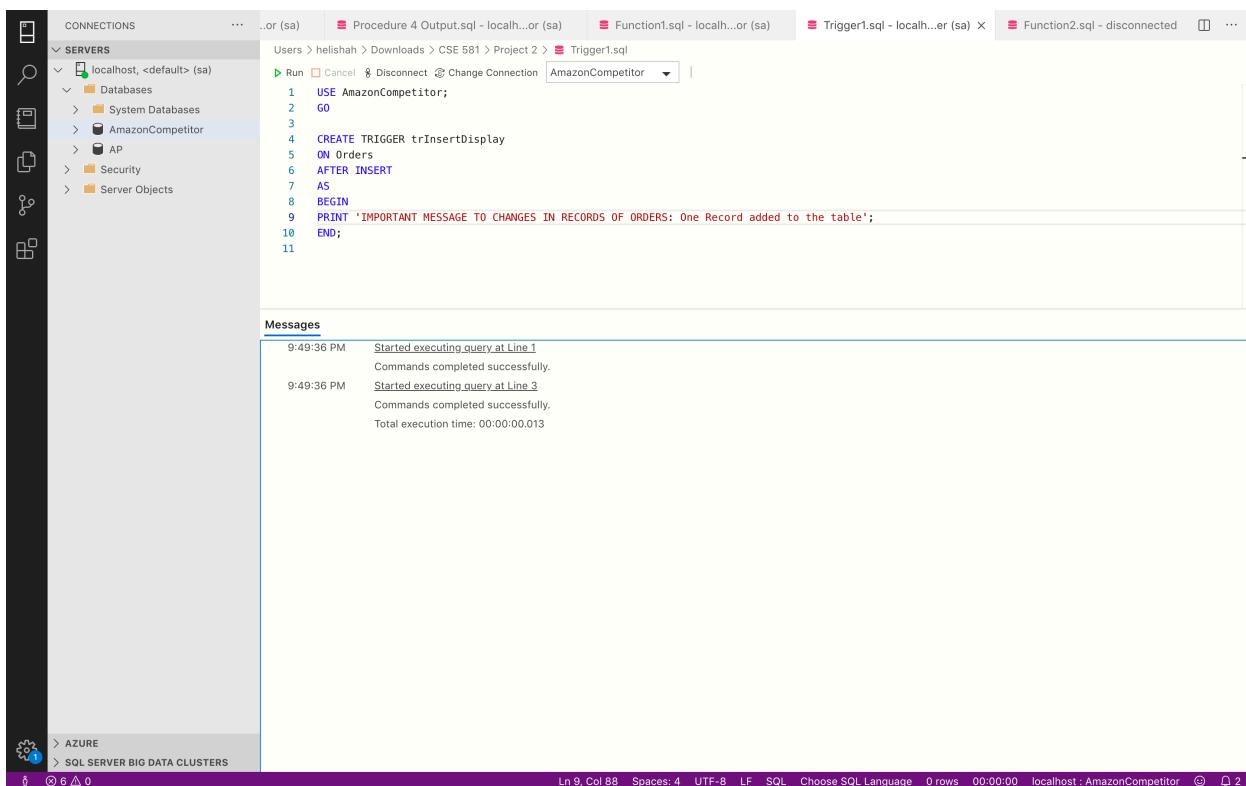
4.6 CREATING TRIGGERS:

A **trigger** is a special type of stored procedure that automatically runs when an event occurs in the database server. DML triggers run when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view. These triggers fire when any valid event fires, whether table rows are affected or not.

We here will be create 2 triggers one is basic and other is complex trigger, both on same tables to have better understanding of the operation trigger.

Trigger 1

This trigger is created on Orders table which will get automatically executed when an insert operation gets performed on Orders table. When this trigger fires it will display a statement defined in print clause.



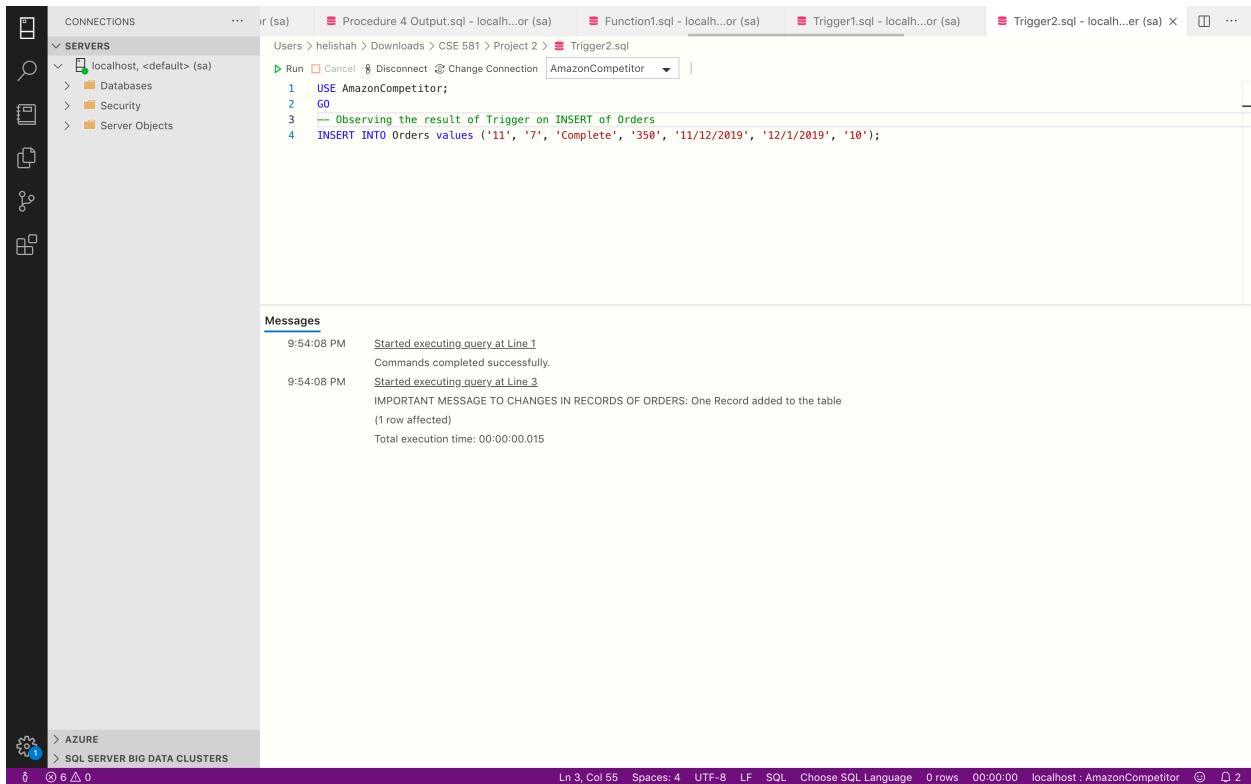
The screenshot shows the SSMS interface with the following details:

- Connections:** localhost, <default> (sa)
- Servers:** localhost, <default> (sa) - expanded, showing:
 - Databases
 - System Databases
 - AmazonCompetitor
 - AP
 - Security
 - Server Objects
- Object Explorer:** Procedure 4 Output.sql - localh...or (sa), Function1.sql - localh...or (sa), Trigger1.sql - localh...er (sa), Function2.sql - disconnected
- Scripting:** Run, Cancel, Disconnect, Change Connection, AmazonCompetitor
- Code Editor:** Contains the following T-SQL script:

```
1 USE AmazonCompetitor;
2 GO
3
4 CREATE TRIGGER trInsertDisplay
5 ON Orders
6 AFTER INSERT
7 AS
8 BEGIN
9 PRINT 'IMPORTANT MESSAGE TO CHANGES IN RECORDS OF ORDERS: One Record added to the table';
10 END;
11
```
- Messages:** Shows execution logs:
 - 9:49:36 PM Started executing query at Line 1
Commands completed successfully.
 - 9:49:36 PM Started executing query at Line 3
Commands completed successfully.
Total execution time: 00:00:00.013
- Status Bar:** Ln 9, Col 88, Spaces: 4, UTF-8, LF, SQL, Choose SQL Language, 0 rows, 00:00:00, localhost : AmazonCompetitor, 2

Here our Trigger is completed successfully, so when we fire Insert on Orders table, this will be executed.

Now, see the output while inserting a record to the Orders table.
The trigger will be fired first and then the statement of Insert will execute.



```
USE AmazonCompetitor;
GO
-- Observing the result of Trigger on INSERT of Orders
INSERT INTO Orders values ('11', '7', 'Complete', '350', '11/12/2019', '12/1/2019', '10');
```

Messages

Time	Message
9:54:08 PM	Started executing query at Line 1
	Commands completed successfully.
9:54:08 PM	Started executing query at Line 3
	IMPORTANT MESSAGE TO CHANGES IN RECORDS OF ORDERS: One Record added to the table (1 row affected)
	Total execution time: 00:00:00.015

AZURE > SQL SERVER BIG DATA CLUSTERS

Ln 3, Col 55 Spaces: 4 UTF-8 LF SQL Choose SQL Language 0 rows 00:00:00 localhost:AmazonCompetitor ② 2

4.7 ROLES, LOGIN AND PASSWORD ENCRYPTION: (SECURITY LEVELS FOR THE DATABASE)

This statements create a role named EntryOf Product which has been granted Update permission on Products.

It has Insert, Update permission on OrderItems tables.

This role also got permission to read data from all tables in current database by adding as a member db-datareader.

The screenshot shows the SSMS interface. The Object Explorer on the left displays the server structure under 'localhost, <default> (sa)'. The 'DBMS' node is selected. The 'Tables' node under DBMS contains 'Products' and 'OrderItems'. The 'Script' button next to 'Products' is highlighted. The 'Properties' button next to 'OrderItems' is also highlighted. The 'Messages' pane at the bottom shows two successful command executions. The status bar at the bottom right indicates the connection is to 'localhost : master'.

```
1 USE AmazonCompetitor;
2 GO
3
4 CREATE ROLE EntryOfProduct;
5 GRANT UPDATE
6 ON Products
7 TO EntryOfProduct;
8 GRANT INSERT, UPDATE
9 ON Orders
10 TO EntryOfProduct;
11
12 ALTER ROLE db_datareader ADD MEMBER EntryOfProduct;
13
14 USE AmazonCompetitor;
15 CREATE LOGIN DBMS WITH PASSWORD = 'Heli@123#',
16 DEFAULT_DATABASE = AmazonCompetitor;
17
18 CREATE USER Heli FOR LOGIN DBMS;
19 ALTER ROLE EntryOfProduct ADD MEMBER Heli;
20
```

Next, we have assigned a password to LOGIN named DBMS using create login script and assigning default database as our current database, i.e. AmazonCompetitor. We also created member named Heli for DBMS login and added him as member to EntryOfProduct by which all grant permissions of EntryOfProduct got transferred to member Heli.

5. CONCLUSION:

5.1 Project Analysis:

The project Amazon Competitor Database has been created and various tables have been created. Sample data was input to the database and it was tested using stored procedures, function, views, triggers and various constraints have been used. Overall the working of the project gives the required outcome. The database was normalized and there is no redundancy in it this makes the database simple and efficient to use. The project has been designed in such a way that it could meet future requirements.

5.2 Remarks:

The Amazon competitor database introduces all the concepts of creating, implementing and testing a database. It also made me learn about the actual industrial requirements in the companies these days. For implementing, the labs and the concepts introduced in class helped in developing the database. This project was a very good practice for all the topics covered in the lecture. The in-class presentations were also a lot helpful to solve this problems in the project.

I used <https://my.vertabelo.com/drive> for designing the database.

I used <https://app.creately.com/manage/project/home> to create my ER diagram.