

INFORMATION MANAGEMENT RETRIEVAL REPORT



Group Assignment:
Supermarket Information System



By - Group 77

INTRODUCTION

The scenario presented entails creating a system for a supermarket that allows the management parties to run and oversee the business operations of the supermarket.

This system is required to and is capable of restricting and controlling access via user authorization levels as well cover the stock management, billing and finance aspects of the business.

Furthermore, the solution is capable of advanced options such as providing financial reports and insights for management to make operational decisions regarding the business.

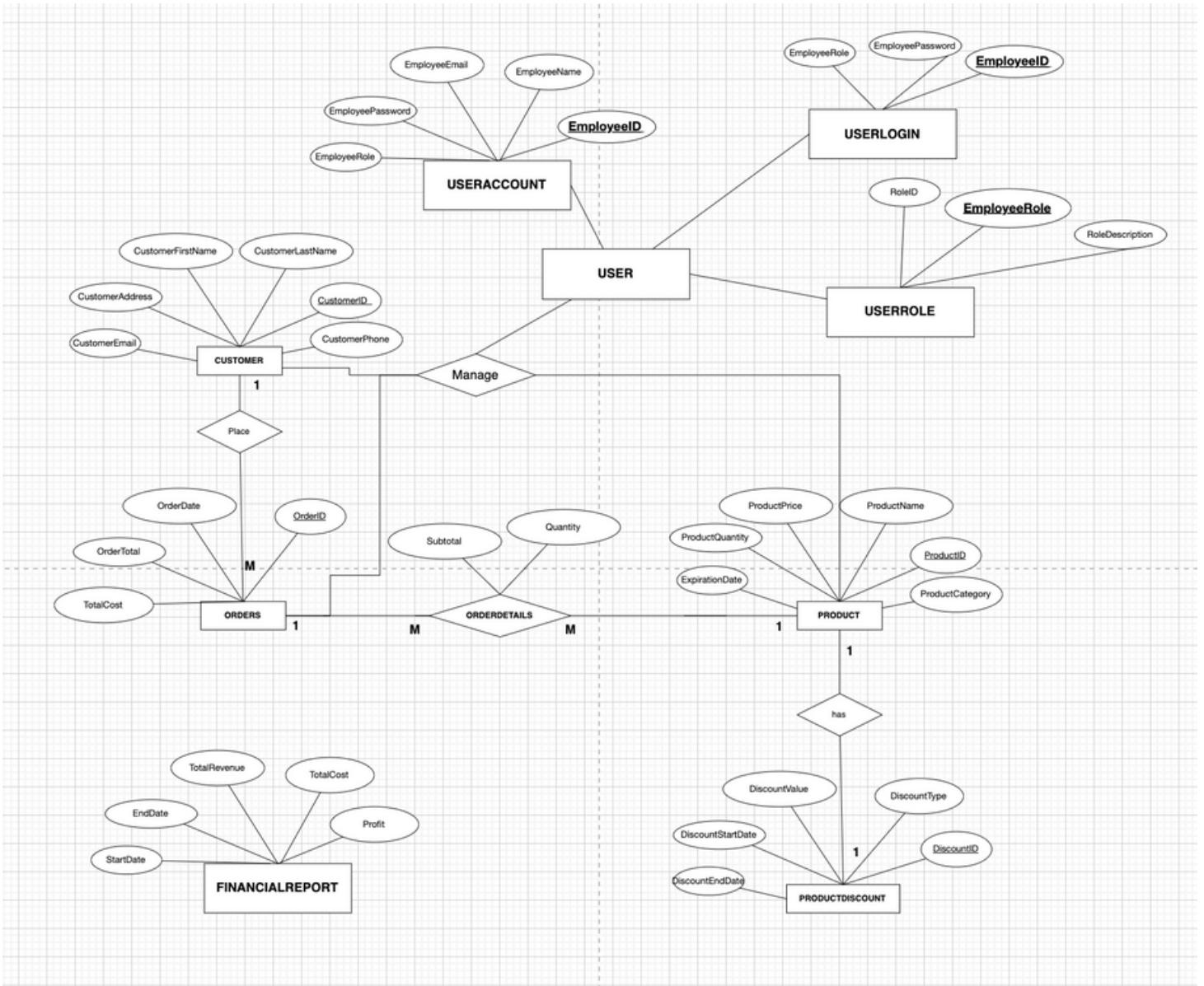
This database enables the staff to enter and maintain information about all of the store's activities. It also includes data validation mechanisms (constraints), triggers, and user-defined functions. Views, stored procedures, and user defined functions can be used by authorized parties to generate meaningful management reports and make strategic, long-term, and short-term managerial decisions for the supermarket.

The Supermarket database is intended to keep track of a retail store's products, customers, orders, and discounts. The Products table contains information about the store's products, such as the product ID, name, price, quantity, and category. Customers information, including contact information, can be found in the Customers table. The Orders table contains information about customer orders, such as the order ID, customer ID, order date, and total cost. The OrderDetails table contains information about the specific products ordered, as well as the quantity of each product in the order. These customer details are stored and can be used in the future for other purposes.

The ProductDiscounts table contains information about the discounts that are applied to products in the store, such as the discount ID, product ID, discount type, discount value, and the validity period. This database's schema also shows how these tables are related to one another via foreign keys. This also provides the ability for customers to receive special offers such as discounts.

Further details and information regarding this solution and its implementation is presented below in the rest of the report for perusal and review.

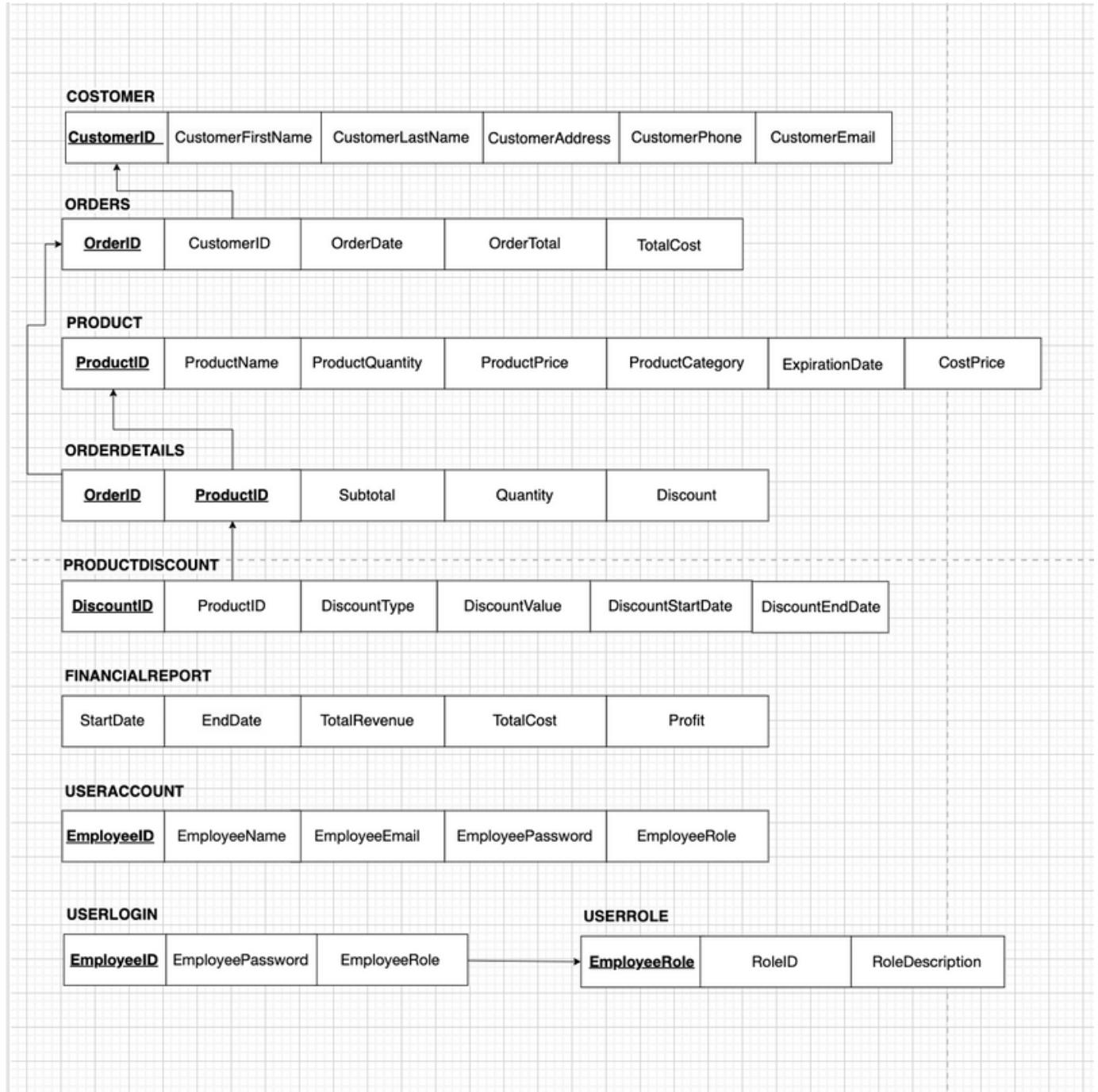
ER DIAGRAM



ER ASSUMPTIONS

- User is responsible for managing the customer, products and order and has user the details stored in UserAccounts, the authority level is stored in the UserRole (eg: admin, manager etc.) and the login details for each user is stored in UserLogin.
- The UserLogin table may be used to control access to the supermarket's systems, such as the point of sale system (not accounted for in this solution), inventory management system, and financial reporting system.
- The supermarket system to ensure that employees with different roles have the necessary access to perform their job functions, while maintaining the security of the system and data.
- Permissions are assumed, granular control over each individual permission is not accounted for. (Permission matrix is not included) for the admin login/access control component.
- Cost price is the buying/manufacturing cost of the products and product price is the selling price (with markup of profit margin). Therefore, product price should always be higher than cost price (to avoid loss).
- The FinancialReport table may be updated on a regular basis, such as daily, weekly, or monthly, to track the financial performance of the supermarket over time.
- The solution doesn't account for a customer loyalty card system.
- Assume that the supermarket is only in one location and there are no branches.
- Solution does not take employee payroll/salary into account for the financial performance aspect.

RELATIONAL MAP



DATA DICTIONARY

Table Name: Customers

Column Name	Data Type	Description
CustomerID	INT	Unique identifier for the customer
CustomerFirstName	NVARCHAR(255)	First name of the customer
CustomerLastName	NVARCHAR(255)	Last name of the customer
CustomerEmail	NVARCHAR(255)	Email address of the customer
CustomerAddress	NVARCHAR(255)	Address of the customer
CustomerPhone	NVARCHAR(255)	Phone number of the customer

Table Name: Products

Column Name	Data Type	Description
ProductID	INT	Unique identifier for the product
ProductName	NVARCHAR(255)	Name of the product
ProductPrice	DECIMAL(10, 2)	Selling price of the product
ProductQuantity	INT	Number of units of the product in stock
ProductCategory	NVARCHAR(255)	Category of the product
ExpirationDate	DATE	Expiry date of the product
CostPrice	DECIMAL(10, 2)	Cost price of the product

Table Name: Orders

Column Name	Data Type	Description
OrderID	INT	Unique identifier for the order
CustomerID	INT	Unique identifier for the customer who placed the order, foreign key referencing the 'Customers' table
OrderDate	DATE	Date on which the order was placed
OrderTotal	DECIMAL(10, 2)	Total of the order
TotalCost	INT	Total cost of the order for the supermarket

DATA DICTIONARY

Table Name: OrderDetails

Column Name	Data Type	Description
OrderID	INT	Unique identifier for the order, foreign key referencing the 'Orders' table
ProductID	INT	Unique identifier for the product, foreign key referencing the 'Products' table
Quantity	INT	Number of units of the product in the order
Subtotal	DECIMAL(10, 2)	Subtotal cost for the units of the product in the order
Discount	DECIMAL(10, 2)	Order Discounts

Table Name: ProductDiscounts

Column Name	Data Type	Description
DiscountID	INT	Unique identifier for the discount
ProductID	INT	Unique identifier for the product the discount applies to, foreign key referencing the 'Product' table
DiscountType	NVARCHAR(255)	Type of the discount, like "BOGO" for Buy one get one free or "Percentage" for percentage off
DiscountValue	DECIMAL(10,2)	Value of the discount, either a percentage or a fixed amount
DiscountStartDate	DATE	The date from which the discount starts
DiscountEndDate	DATE	The date on which the discount ends

Table Name: FinancialReport

Column Name	Data Type	Description
StartDate	date	The start date of the financial period being reported on.
EndDate	date	The end date of the financial period being reported on.
TotalRevenue	decimal(10,2)	The total amount of revenue generated during the financial period.
TotalCost	decimal(10,2)	The total amount of cost incurred during the financial period.
Profit	decimal(10,2)	The profit generated during the financial period (calculated as TotalRevenue - TotalCost).

DATA DICTIONARY

Table Name: UserAccount

Column Name	Data Type	Description
EmployeeID	INT	Primary key, unique identifier for each employee
EmployeeName	VARCHAR(50)	Employee's name
EmployeeEmail	VARCHAR(255)	Employee's email address
EmployeePassword	VARCHAR(255)	Employee's password
EmployeeRole	VARCHAR(20)	Employee's role (e.g. Manager, Employee)

Table Name: UserRole

Column Name	Data Type	Description
EmployeeRole	VARCHAR(20)	Employee's role, it is a primary key
RoleID	INT	Unique identifier for the role
RoleDescription	VARCHAR(255)	Description of the Role

Table Name: UserLogin

Column Name	Data Type	Description
EmployeeID	INT	Unique identifier for each employee
EmployeePassword	VARCHAR(255)	Employee's password, not null value
EmployeeRole	VARCHAR(20)	Employee's role, it references the UserRole table, not null value

CREATING TABLES AND CONSTRAINTS

```
-- Create the database
CREATE DATABASE Supermarket;
GO

USE Supermarket;
GO

-- Create a table for products
CREATE TABLE Products
(
    ProductID int PRIMARY KEY,
    ProductName nvarchar(255) NOT NULL,
    ProductPrice decimal(10, 2) NOT NULL,
    ProductQuantity int NOT NULL,
    ProductCategory nvarchar(255) NOT NULL
);

-- Create a table for customers
CREATE TABLE Customers
(
    CustomerID int PRIMARY KEY,
    CustomerFirstName nvarchar(255) NOT NULL,
    CustomerLastName nvarchar(255) NOT NULL,
    CustomerEmail nvarchar(255) NOT NULL,
    CustomerAddress nvarchar(255) NOT NULL,
    CustomerPhone nvarchar(255) NOT NULL
);

-- Create a table for orders
CREATE TABLE Orders
(
    OrderID int PRIMARY KEY,
    CustomerID int FOREIGN KEY REFERENCES Customers(CustomerID),
    OrderDate date NOT NULL,
    OrderTotal decimal(10, 2) NOT NULL
);

-- Create a table for order details
CREATE TABLE OrderDetails
(
    OrderID int FOREIGN KEY REFERENCES Orders(OrderID),
    ProductID int FOREIGN KEY REFERENCES Products(ProductID),
    Quantity int NOT NULL,
    Subtotal decimal(10, 2) NOT NULL
);
```

CREATING TABLES AND CONSTRAINTS

```
CREATE TABLE ProductDiscounts
(
    DiscountID int PRIMARY KEY,
    ProductID int FOREIGN KEY REFERENCES Products(ProductID),
    DiscountType nvarchar(255) NOT NULL,
    DiscountValue decimal(10,2) NOT NULL,
    DiscountStartDate date NOT NULL,
    DiscountEndDate date NOT NULL
);

CREATE TABLE FinancialReport
(
    StartDate DATE,
    EndDate DATE,
    TotalRevenue DECIMAL(10,2),
    TotalCost DECIMAL(10,2),
    Profit DECIMAL(10,2)
);

CREATE TABLE UserAccount
(
    EmployeeID INT PRIMARY KEY IDENTITY(1,1),
    EmployeeName VARCHAR(50) NOT NULL,
    EmployeeEmail VARCHAR(255) NOT NULL,
    EmployeePassword VARCHAR(255) NOT NULL,
    EmployeeRole VARCHAR(20) NOT NULL
);

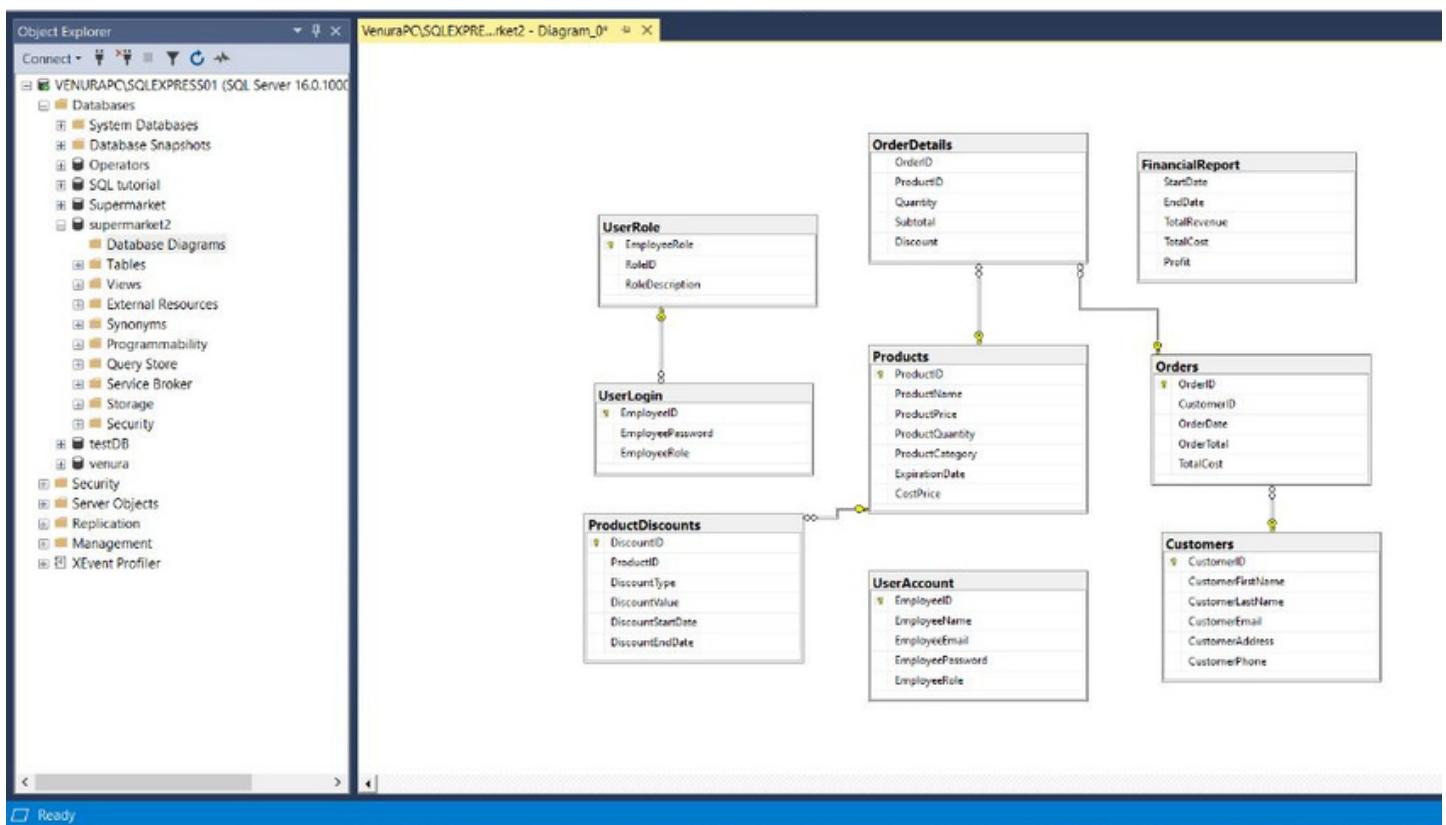
CREATE TABLE UserRole
(
    EmployeeRole VARCHAR(20) PRIMARY KEY,
    RoleID INT,
    RoleDescription VARCHAR(100)
);

CREATE TABLE UserLogin
(
    EmployeeID INT PRIMARY KEY IDENTITY (1,1),
    EmployeePassword VARCHAR(255) NOT NULL,
    EmployeeRole VARCHAR(20) FOREIGN KEY REFERENCES UserRole(EmployeeRole)
);
```

Please Note:

That these are the initial code snippets we used to build the database but on the process of building this database and some tables has been altered accordingly to move forward with the new triggers, user-defined functions, stored procedures and views .

DATABASE DIAGRAM



SAMPLE RECORDS

Data insertion for the table Products

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'supermarket2'. The main query window contains an `INSERT INTO` statement for the 'Products' table, listing 20 different products with their details. The status bar at the bottom indicates 'Query executed successfully.' and '0 rows' affected.

```
USE supermarket2
GO
INSERT INTO Products (ProductID, ProductName, ProductPrice, ProductQuantity, ProductCategory, ExpirationDate)
VALUES (1, 'Apple', 0.50, 100, 'Fruit', '2022-02-20'),
(2, 'Banana', 0.30, 200, 'Fruit', '2022-03-15'),
(3, 'Bread', 2.00, 50, 'Bakery', '2022-04-10'),
(4, 'Toilet Paper', 0.50, 50, 'Personal Care', '2022-05-05'),
(5, 'Shampoo', 6.00, 50, 'Personal Care', '2022-06-01'),
(6, 'Carrot', 0.40, 150, 'Vegetable', '2022-07-10'),
(7, 'Lemon', 0.60, 100, 'Fruit', '2022-08-15'),
(8, 'Ice Cream', 5.00, 25, 'Dairy', '2022-09-20'),
(9, 'Cheese', 8.00, 20, 'Dairy', '2022-10-15'),
(10, 'Eggs', 1.50, 50, 'Dairy', '2022-11-30'),
(11, 'Milk', 2.00, 100, 'Bakery', '2022-02-20'),
(12, 'Salt', 1.50, 150, 'Spice', '2022-03-15'),
(13, 'Water Bottle', 2.00, 50, 'Beverage', '2022-04-10'),
(14, 'Soda', 3.00, 25, 'Beverage', '2022-05-05'),
(15, 'Shampoo', 6.00, 50, 'Personal Care', '2022-06-01'),
(16, 'Lotion', 3.00, 25, 'Personal Care', '2022-07-10'),
(17, 'Toothpaste', 2.50, 75, 'Personal Care', '2022-08-15'),
(18, 'Chips', 3.50, 50, 'Snacks', '2022-09-20'),
(19, 'Chocolate', 5.00, 25, 'Snacks', '2022-10-25'),
(20, 'Cookies', 2.50, 50, 'Snacks', '2022-11-30');
```

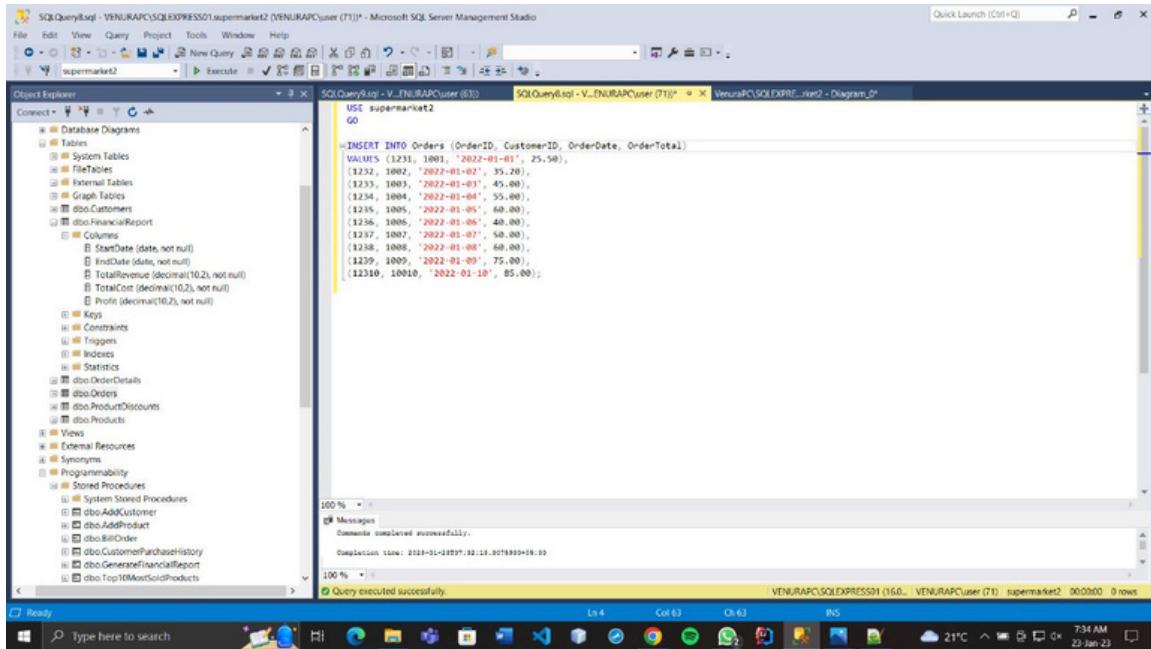
Data insertion for the table Products

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'supermarket2'. The main query window contains an `INSERT INTO` statement for the 'Customers' table, listing 20 different customers with their details. The status bar at the bottom indicates 'Query executed successfully.' and '0 rows' affected.

```
USE supermarket2
GO
INSERT INTO Customers (CustomerID, CustomerFirstName,
CustomerLastName, CustomerEmail, CustomerAddress, CustomerPhone)
VALUES (1001, 'John', 'Doe', 'johndoe@email.com', '123 Main St',
'555-555-5555'),
(1002, 'Jane', 'Smith', 'janesmith@email.com', '456 Park Ave',
'555-555-5556'),
(1003, 'Michael', 'Johnson', 'michaeljohnson@email.com', '789 Elm St',
'555-555-5557'),
(1004, 'Emily', 'Williams', 'emilyw@email.com', '321 Oak St',
'555-555-5558'),
(1005, 'Matthew', 'Brown', 'mattheaubrown@email.com', '654 Pine St',
'555-555-5559'),
(1006, 'Sarah', 'Jones', 'sarahj@email.com', '987 Cedar St',
'555-555-5560'),
(1007, 'Olivia', 'Miller', 'oliviemail@email.com', '125 Birch St', '555-555-5561'),
(1008, 'Sophia', 'Davis', 'sophia@email.com', '246 Maple St',
'555-555-5562'),
(1009, 'Isabella', 'Garcia', 'isabelag@email.com', '369 Cherry St',
'555-555-5563'),
(1010, 'Avery', 'Rodriguez', 'avery@email.com', '150 Oak St',
'555-555-5564'),
(1011, 'Evelyn', 'Martinez', 'evelynn@email.com', '753 Pine St',
'555-555-5565'),
(1012, 'Abigail', 'Hernandez', 'abigailh@email.com', '951 Cedar St',
'555-555-5566');
```

SAMPLE RECORDS

Data insertion for the table Orders

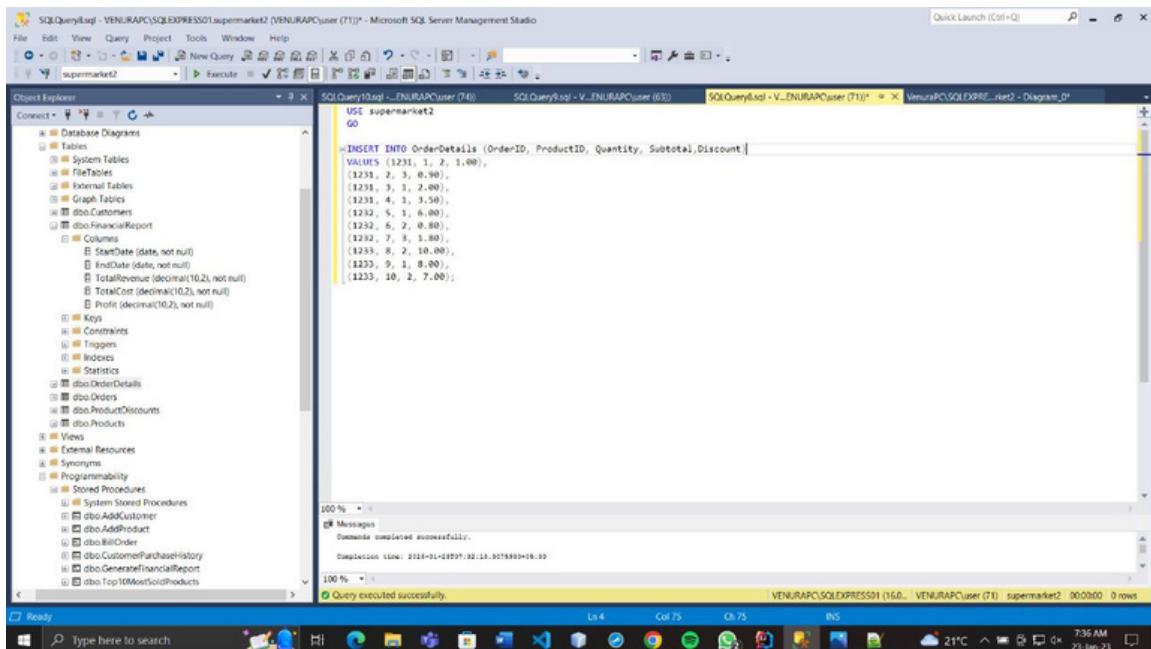


The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'supermarket2'. The central pane displays a T-SQL script for inserting data into the 'Orders' table:

```
USE supermarket2
GO
INSERT INTO Orders (OrderID, CustomerID, OrderDate, OrderTotal)
VALUES (1231, '1001', '2022-01-01', 25.50),
       (1232, '1002', '2022-01-02', 35.00),
       (1233, '1003', '2022-01-03', 45.00),
       (1234, '1004', '2022-01-04', 55.00),
       (1235, '1005', '2022-01-05', 60.00),
       (1236, '1006', '2022-01-06', 40.00),
       (1237, '1007', '2022-01-07', 30.00),
       (1238, '1008', '2022-01-08', 40.00),
       (1239, '1009', '2022-01-09', 75.00),
       (12310, '1010', '2022-01-10', 85.00);
```

The status bar at the bottom indicates the command completed successfully with a duration of 00:00:00.

Data insertion for the table OrderDetails



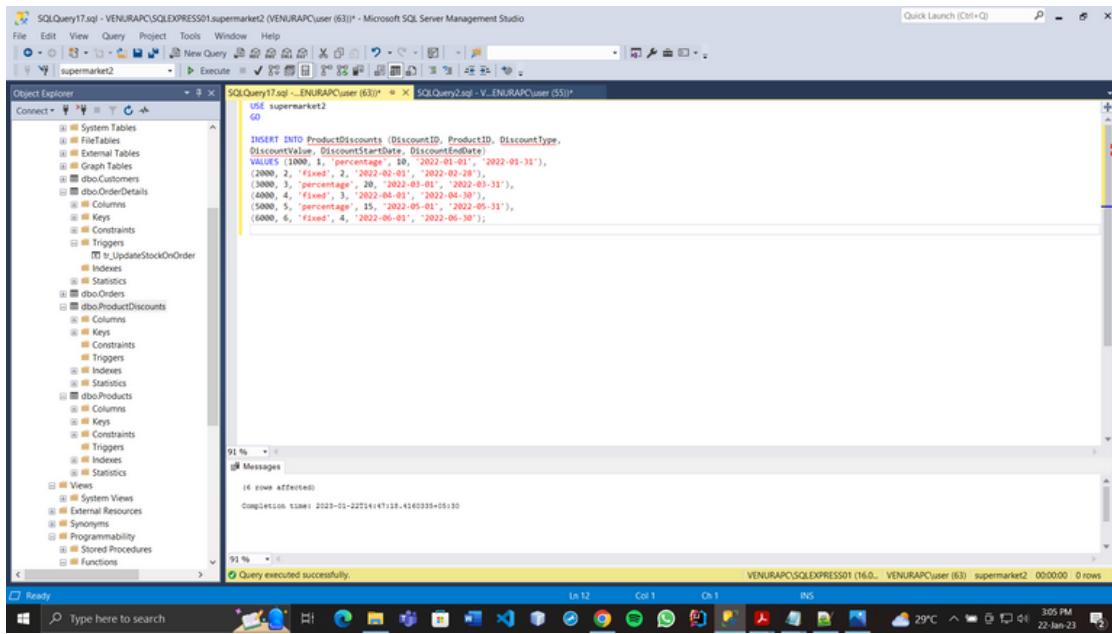
The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'supermarket2'. The central pane displays a T-SQL script for inserting data into the 'OrderDetails' table:

```
USE supermarket2
GO
INSERT INTO OrderDetails (OrderID, ProductID, Quantity, Subtotal, Discount)
VALUES (1231, 1, 2, 1.00),
       (1231, 2, 3, 0.90),
       (1231, 3, 1, 2.00),
       (1231, 4, 1, 3.50),
       (1232, 5, 1, 6.00),
       (1232, 6, 2, 0.80),
       (1232, 7, 3, 1.00),
       (1232, 8, 2, 10.00),
       (1232, 9, 1, 0.00),
       (1233, 10, 2, 7.00);
```

The status bar at the bottom indicates the command completed successfully with a duration of 00:00:00.

SAMPLE RECORDS

Data insertion for the table ProductDiscounts

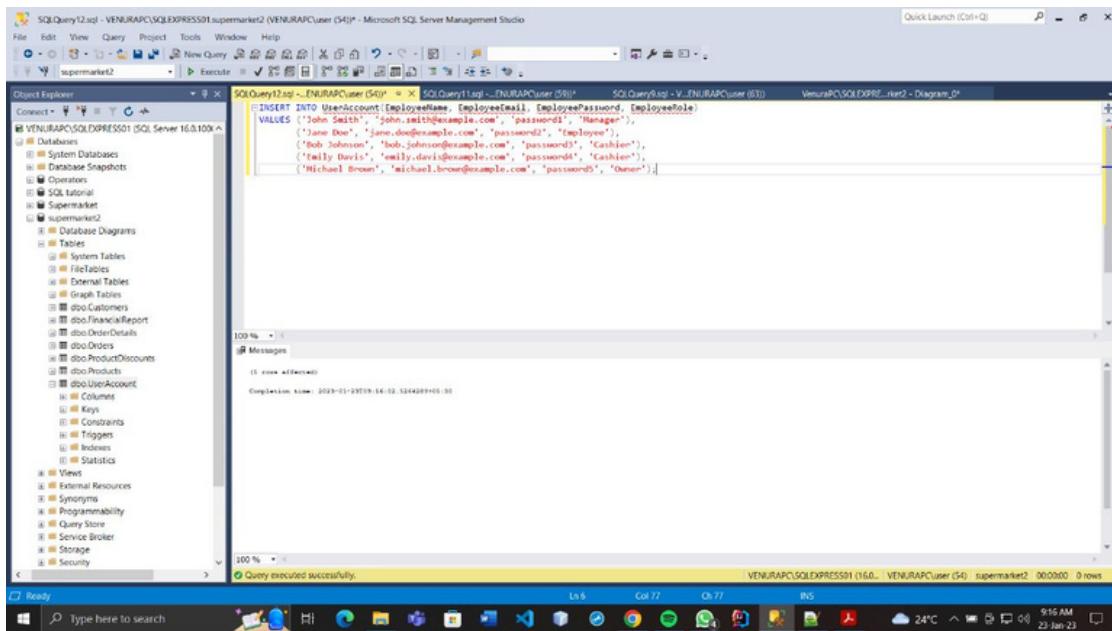


The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'supermarket2'. The central pane displays a query window with the following SQL code:

```
USE supermarket2
GO
INSERT INTO ProductDiscounts (DiscountID, ProductID, DiscountType,
                               DiscountValue, DiscountStartDate, DiscountEndDate)
VALUES (1000, 1, 'percentage', 10, '2022-01-01', '2022-01-31'),
       (2000, 2, 'Fixed', 2, '2022-02-01', '2022-02-28'),
       (3000, 3, 'percentage', 15, '2022-03-01', '2022-03-31'),
       (4000, 4, 'Fixed', 3, '2022-04-01', '2022-04-30'),
       (5000, 5, 'percentage', 15, '2022-05-01', '2022-05-31'),
       (6000, 6, 'Fixed', 4, '2022-06-01', '2022-06-30');
```

The status bar at the bottom indicates 'Query executed successfully.' and shows the completion time as 2023-01-22T14:47:12.4160335+00:00.

Data insertion for the table UserAccount



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'supermarket2'. The central pane displays a query window with the following SQL code:

```
USE supermarket2
GO
INSERT INTO UserAccount (EmployeeName, EmployeeEmail, EmployeePassword, EmployeeRole)
VALUES ('John Smith', 'john.smith@example.com', 'password1', 'Manager'),
       ('Jane Doe', 'jane.doe@example.com', 'password2', 'Employee'),
       ('Bob Johnson', 'bob.johnson@example.com', 'password3', 'Cashier'),
       ('Dally Davis', 'dally.davis@example.com', 'password4', 'Cashier'),
       ('Michael Brown', 'michael.brown@example.com', 'password5', 'Owner');
```

The status bar at the bottom indicates 'Query executed successfully.' and shows the completion time as 2023-01-23T09:14:12.5244289+00:00.

SAMPLE RECORDS

Data insertion for the table UserRole

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the supermarket2 database. The center pane contains a query window with the following SQL code:

```
INSERT INTO UserRole([EmployeeRole, RoleID, RoleDescription])
VALUES ('Manager', 1, 'Manages the employees and is responsible for the overall operation of the supermarket'),
       ('Employee', 2, 'Works in the supermarket and follows the instructions of the manager'),
       ('Cashier', 3, 'Handles the cash transactions and is responsible for the cash balance'),
       ('Owner', 4, 'Owner of the business');
```

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

Data insertion for the table UserLogin

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the supermarket2 database. The center pane contains a query window with the following SQL code:

```
USE supermarket2
GO

INSERT INTO UserLogin([EmployeeID, EmployeePassword, EmployeeRole])
VALUES (1, 'password1', 'Cashier'),
       (2, 'password2', 'Manager'),
       (3, 'password3', 'Employee'),
       (4, 'password4', 'Owner');

SET IDENTITY_INSERT UserLogin ON;
```

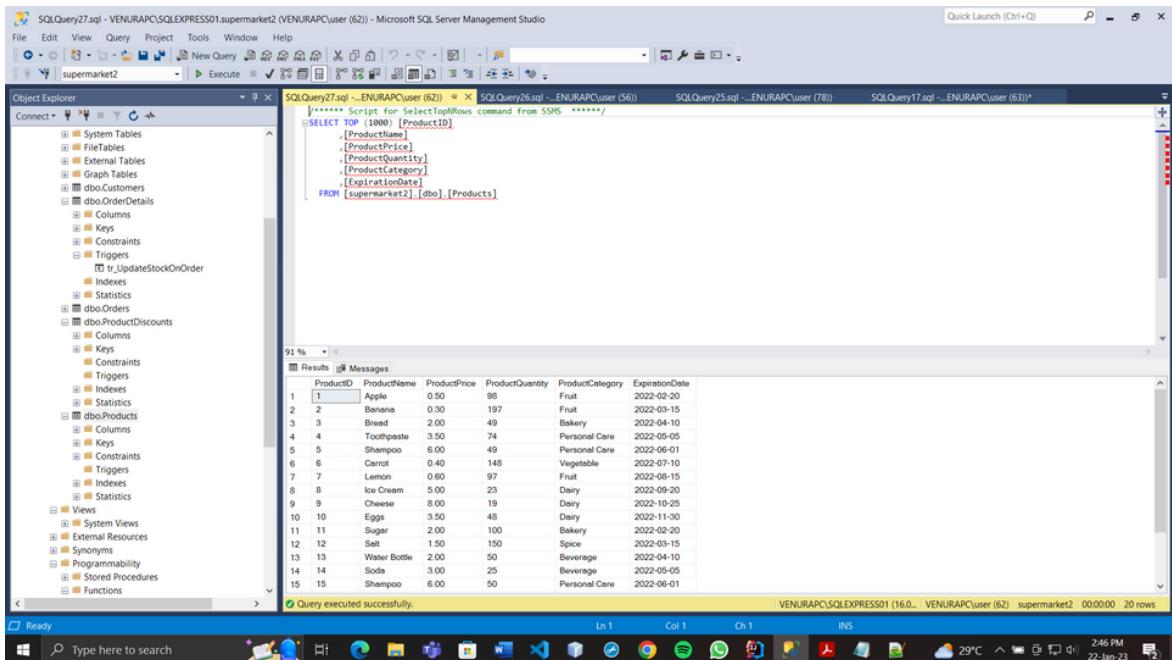
The status bar at the bottom indicates "Query executed successfully." and "0 rows".

Please Note:

That these are the initial code snippets we used to build the database but on the process of building this database and some tables has been altered accordingly by inserting new data to move forward with the new triggers, user-defined functions, stored procedures and views .

ALL TABLES

Products Table



SQL Server Management Studio - VENURAPC\SQLEXPRESS01.supermarket2 (VENURAPC\user (62))

Object Explorer

SQLQuery27.sql - ENURAPC\user (62) | SQLQuery28.sql - ENURAPC\user (56) | SQLQuery25.sql - ENURAPC\user (78) | SQLQuery17.sql - ENURAPC\user (63)*

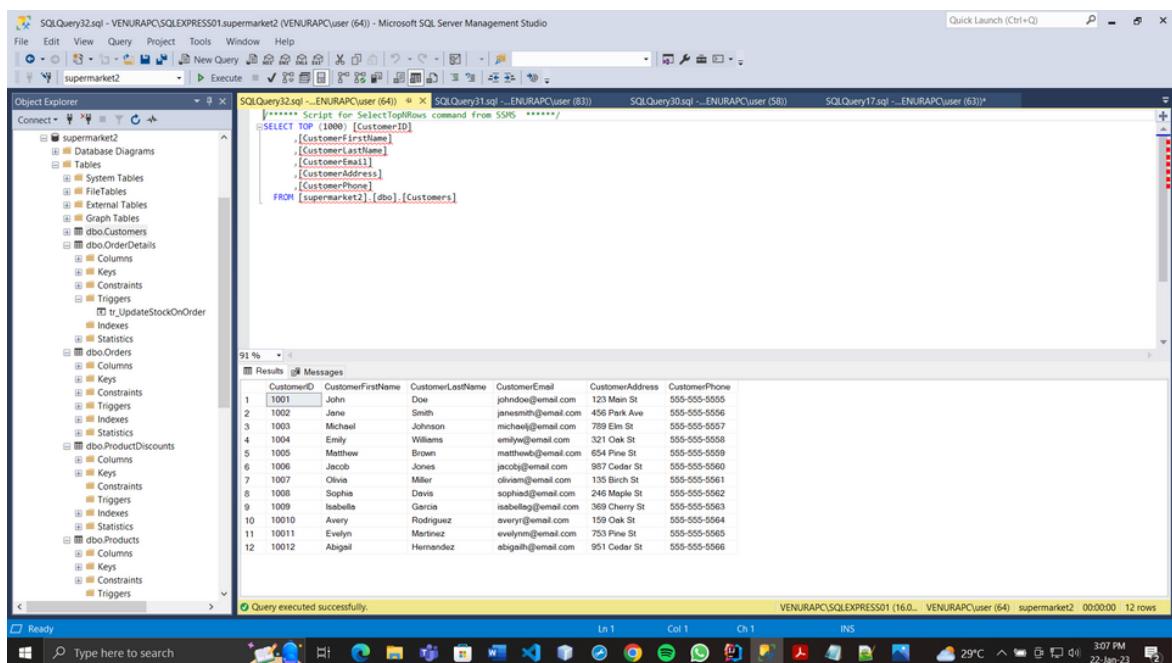
```
***** Script for SelectTopNRows command from SSMS *****
SELECT TOP (1000) [ProductID]
      ,[ProductName]
      ,[ProductPrice]
      ,[ProductQuantity]
      ,[ProductCategory]
      ,[ExpirationDate]
  FROM [supermarket2].[dbo].[Products]
```

Results Messages

ProductID	ProductName	ProductPrice	ProductQuantity	ProductCategory	ExpirationDate
1	Apple	0.50	98	Fruit	2022-02-20
2	Banana	0.30	197	Fruit	2022-03-15
3	Bread	2.00	49	Bakery	2022-04-10
4	Toothpaste	3.50	74	Personal Care	2022-05-05
5	Shampoo	6.00	49	Personal Care	2022-06-01
6	Carrot	0.50	148	Vegetable	2022-07-10
7	Lemon	0.60	97	Fruit	2022-08-15
8	Ice Cream	5.00	23	Dairy	2022-09-20
9	Cheese	6.00	19	Dairy	2022-10-25
10	Eggs	3.50	48	Dairy	2022-11-30
11	Sugar	2.00	100	Bakery	2022-03-20
12	Salt	1.50	150	Spice	2022-03-15
13	Water Bottle	2.00	50	Beverage	2022-04-10
14	Soda	3.00	25	Beverage	2022-05-05
15	Shampoo	6.00	50	Personal Care	2022-06-01

Query executed successfully.

Customers Table



SQL Server Management Studio - VENURAPC\SQLEXPRESS01.supermarket2 (VENURAPC\user (64))

Object Explorer

SQLQuery32.sql - ENURAPC\user (64) | SQLQuery31.sql - ENURAPC\user (83) | SQLQuery30.sql - ENURAPC\user (58) | SQLQuery17.sql - ENURAPC\user (63)*

```
***** Script for SelectTopNRows command from SSMS *****
SELECT TOP (1000) [CustomerID]
      ,[CustomerFirstName]
      ,[CustomerLastName]
      ,[CustomerEmail]
      ,[CustomerAddress]
      ,[CustomerPhone]
  FROM [supermarket2].[dbo].[Customers]
```

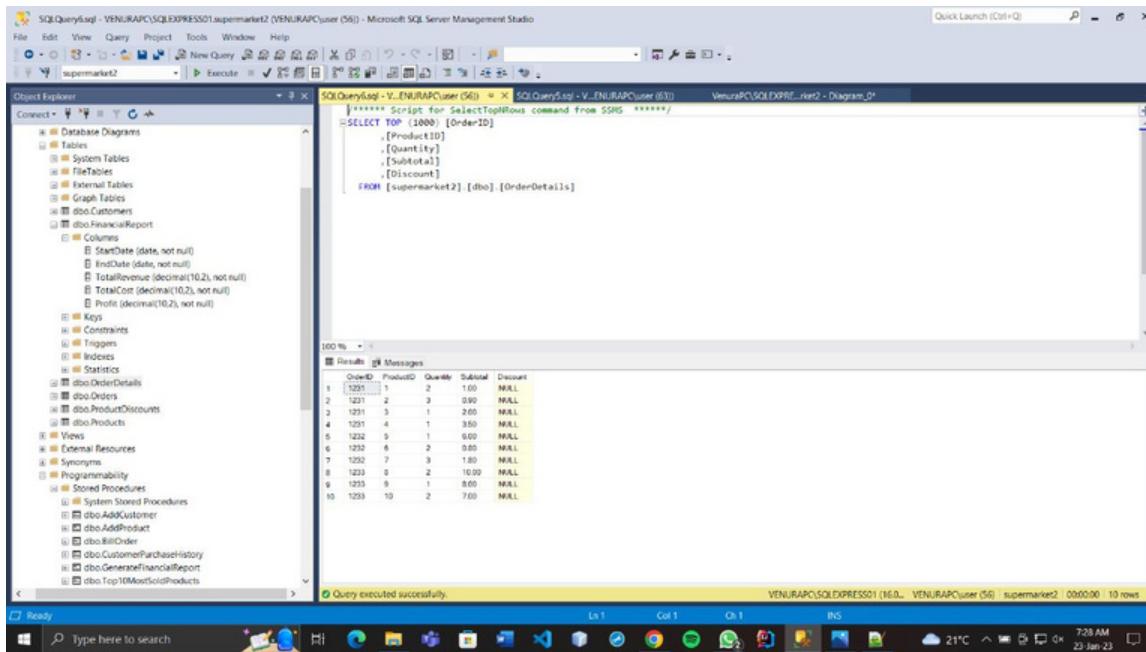
Results Messages

CustomerID	CustomerFirstName	CustomerLastName	CustomerEmail	CustomerAddress	CustomerPhone
1	John	Doe	john.doe@email.com	123 Main St	555-555-5555
2	Jane	Smith	janesmith@email.com	456 Park Ave	555-555-5556
3	Michael	Johnson	michael.j@email.com	789 Elm St	555-555-5557
4	Emily	Williams	emily.w@email.com	321 Oak St	555-555-5558
5	Matthew	Brown	matthew.b@email.com	654 Pine St	555-555-5559
6	Jacob	Jones	jacob.j@email.com	987 Cedar St	555-555-5560
7	Olivia	Miller	olivia.m@email.com	135 Birch St	555-555-5561
8	Sophia	Davis	sophia.d@email.com	248 Maple St	555-555-5562
9	Isabella	Garcia	isabella.g@email.com	369 Cherry St	555-555-5563
10	Avery	Rodriguez	avery.r@email.com	159 Oak St	555-555-5564
11	Evelyn	Martinez	evelynn.m@email.com	753 Pine St	555-555-5565
12	Abigail	Hernandez	abigail.h@email.com	951 Cedar St	555-555-5566

Query executed successfully.

ALL TABLES

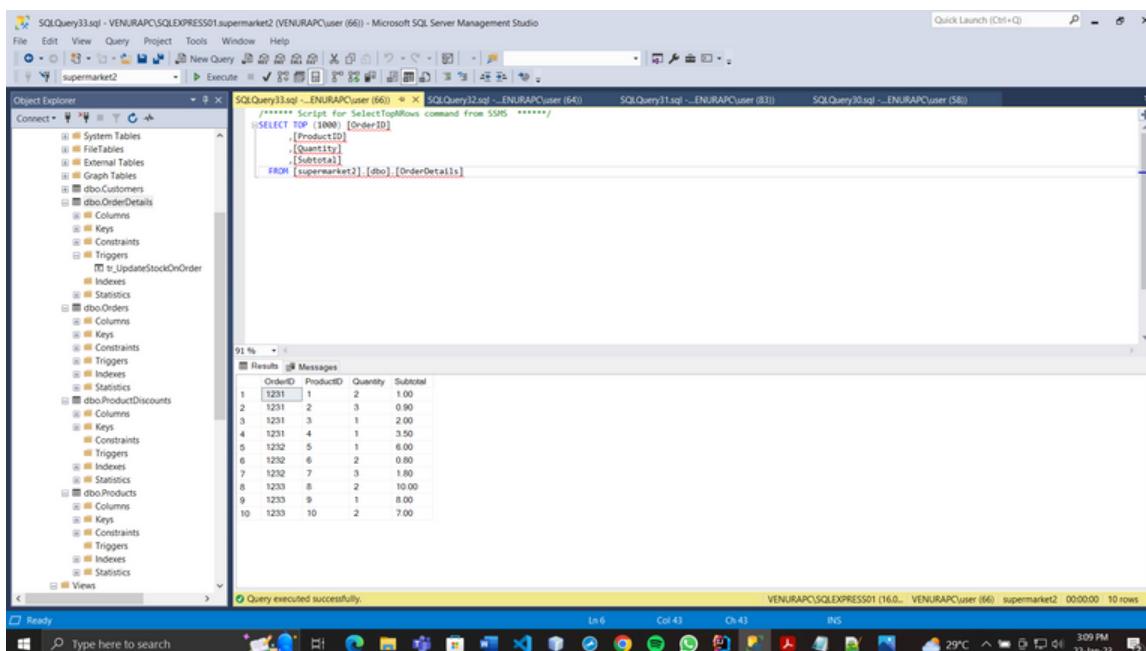
Orders Table



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including tables like 'Customers', 'Products', and 'OrderDetails'. The central pane displays a query results grid for the 'Orders' table, which contains 10 rows of data. The bottom status bar indicates the query was executed successfully at 00:00:00 on 23-Jan-23.

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight	ShipName	ShipAddress	ShipCity	ShipRegion	ShipPostalCode	ShipCountry
1	ALFKI	1	1997-02-13 00:00:00	1997-03-13 00:00:00	1997-02-22 00:00:00	1	10.00	MULLER GOURMET FOODS	27 Prinzenstr.	Berlin	West Berlin	1075	Germany
2	ALFKI	2	1997-02-13 00:00:00	1997-03-13 00:00:00	1997-02-22 00:00:00	1	9.90	MULLER GOURMET FOODS	27 Prinzenstr.	Berlin	West Berlin	1075	Germany
3	ALFKI	3	1997-02-13 00:00:00	1997-03-13 00:00:00	1997-02-22 00:00:00	1	2.00	MULLER GOURMET FOODS	27 Prinzenstr.	Berlin	West Berlin	1075	Germany
4	ALFKI	4	1997-02-13 00:00:00	1997-03-13 00:00:00	1997-02-22 00:00:00	1	3.50	MULLER GOURMET FOODS	27 Prinzenstr.	Berlin	West Berlin	1075	Germany
5	ALFKI	5	1997-02-13 00:00:00	1997-03-13 00:00:00	1997-02-22 00:00:00	1	6.00	MULLER GOURMET FOODS	27 Prinzenstr.	Berlin	West Berlin	1075	Germany
6	ALFKI	6	1997-02-13 00:00:00	1997-03-13 00:00:00	1997-02-22 00:00:00	1	0.80	MULLER GOURMET FOODS	27 Prinzenstr.	Berlin	West Berlin	1075	Germany
7	ALFKI	7	1997-02-13 00:00:00	1997-03-13 00:00:00	1997-02-22 00:00:00	1	1.80	MULLER GOURMET FOODS	27 Prinzenstr.	Berlin	West Berlin	1075	Germany
8	ALFKI	8	1997-02-13 00:00:00	1997-03-13 00:00:00	1997-02-22 00:00:00	1	10.00	MULLER GOURMET FOODS	27 Prinzenstr.	Berlin	West Berlin	1075	Germany
9	ALFKI	9	1997-02-13 00:00:00	1997-03-13 00:00:00	1997-02-22 00:00:00	1	8.00	MULLER GOURMET FOODS	27 Prinzenstr.	Berlin	West Berlin	1075	Germany
10	ALFKI	10	1997-02-13 00:00:00	1997-03-13 00:00:00	1997-02-22 00:00:00	2	7.00	MULLER GOURMET FOODS	27 Prinzenstr.	Berlin	West Berlin	1075	Germany

OrderDetails Table



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including tables like 'Customers', 'Products', and 'OrderDetails'. The central pane displays a query results grid for the 'OrderDetails' table, which contains 10 rows of data. The bottom status bar indicates the query was executed successfully at 00:00:00 on 22-Jan-23.

OrderID	ProductID	Quantity	Subtotal
1	1	2	1.00
2	1	3	0.90
3	1	1	2.00
4	1	1	3.50
5	1	1	6.00
6	1	2	0.80
7	1	3	1.80
8	1	2	10.00
9	1	1	8.00
10	1	2	7.00

ALL TABLES

Product Discounts Table

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure for 'supermarket2'. The central pane displays a query results grid titled 'Results' for the 'Product Discounts' table. The table has columns: DiscountID, ProductID, DiscountType, DiscountValue, DiscountStartDate, and DiscountEndDate. The data shows six rows of discount information.

DiscountID	ProductID	DiscountType	DiscountValue	DiscountStartDate	DiscountEndDate
1	1000	percentage	10.00	2022-01-01	2022-01-31
2	2000	fixed	2.00	2022-02-01	2022-02-28
3	3000	percentage	20.00	2022-03-01	2022-03-31
4	4000	fixed	3.00	2022-04-01	2022-04-30
5	5000	percentage	15.00	2022-05-01	2022-05-31
6	6000	fixed	4.00	2022-06-01	2022-06-30

User Account Table

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure for 'supermarket2'. The central pane displays a query results grid titled 'Results' for the 'User Account' table. The table has columns: EmployeeID, EmployeeName, EmployeeEmail, EmployeePassword, and EmployeeRole. The data shows five rows of employee account information.

EmployeeID	EmployeeName	EmployeeEmail	EmployeePassword	EmployeeRole
1	John Smith	john.smith@example.com	password1	Manager
2	Jane Doe	jane.doe@example.com	password2	Employee
3	Bob Johnson	bob.johnson@example.com	password3	Cashier
4	Emily Davis	emily.davis@example.com	password4	Cashier
5	Michael Brown	michael.brown@example.com	password5	Owner

ALL TABLES

User Role Table

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including the 'supermarket2' database which contains tables like 'dbo.Customers', 'dbo.OrderDetails', and 'dbo.Orders'. The central Results pane displays the output of a query:

```
***** Script for SelectTopNRows command from SSMS *****
--SELECT TOP (1000) [EmployeeRole]
--  ,[RoleID]
--  ,[RoleDescription]
-- FROM [supermarket2].[dbo].[UserRole]
```

EmployeeRole	RoleID	RoleDescription
Cashier	3	Handles the cash transactions and is responsible for...
Employee	2	Works in the supermarket and follows the instructions...
Manager	1	Manages the employees and is responsible for the...
Owner	4	Owner of the business

Below the results, a message indicates "Query executed successfully." The status bar at the bottom shows "VENURAPC\SOLEXPRESS01 (16.0.1000) VENURAPC\User (55) supermarket2 00:00:00 4 rows".

User Login Table

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including the 'supermarket2' database which contains tables like 'dbo.Customers', 'dbo.OrderDetails', and 'dbo.Orders'. The central Results pane displays the output of a query:

```
***** Script for SelectTopNRows command from SSMS *****
--SELECT TOP (1000) [EmployeeID]
--  ,[EmployeePassword]
--  ,[EmployeeRole]
-- FROM [supermarket2].[dbo].[UserLogin]
```

EmployeeID	EmployeePassword	EmployeeRole
1	password1	Cashier
2	password2	Manager
3	password3	Employee
4	password4	Manager

Below the results, a message indicates "Query executed successfully." The status bar at the bottom shows "VENURAPC\SOLEXPRESS01 (16.0.1000) VENURAPC\User (56) supermarket2 00:00:00 4 rows".

TRIGGERS

Trigger to Update Stock on Purchase

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'Supermarket' with various tables like 'Products', 'OrderDetails', and 'Orders'. The 'Triggers' folder under 'Products' contains a single trigger named 'tr_UpdateStockOnPurchase'. The central query editor window displays the T-SQL code for this trigger:

```
USE Supermarket;
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
----- Author: <Author>, Name: <Create Date>
----- Create date: <Create Date>
----- Description: <Description>
GO
CREATE TRIGGER tr_UpdateStockOnPurchase
ON Products
AFTER INSERT
AS
BEGIN
    UPDATE Products
    SET ProductQuantity = Products.ProductQuantity + OrderDetails.Quantity
    FROM Products
    JOIN OrderDetails
    ON Products.ProductID = OrderDetails.ProductID
END
```

The status bar at the bottom indicates the query was completed successfully.

Trigger to Update Stock on Expiration

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'Supermarket' with tables like 'Products', 'OrderDetails', and 'Orders'. The 'Triggers' folder under 'Products' contains a trigger named 'tr_UpdateStockOnExpiration'. The central query editor window displays the T-SQL code for this trigger:

```
USE Supermarket;
GO
CREATE TRIGGER tr_UpdateStockOnExpiration
ON Products
AFTER UPDATE
AS
BEGIN
    DECLARE @ExpiredQuantity INT;
    SET @ExpiredQuantity = 0;
    SELECT @ExpiredQuantity = Products.ProductQuantity - OrderDetails.Quantity
    FROM Products
    JOIN OrderDetails
    ON Products.ProductID = OrderDetails.ProductID
    WHERE Products.ExpirationDate < GETDATE();
    IF @ExpiredQuantity > 0
    BEGIN
        UPDATE Products
        SET ProductQuantity = @ExpiredQuantity
        FROM Products
        WHERE ProductID = inserted.ProductID
        AND inserted.ExpirationDate < GETDATE();
    END
    ELSE
    BEGIN
        UPDATE Products
        SET ProductQuantity = Products.ProductQuantity + OrderDetails.Quantity
        FROM Products
        JOIN OrderDetails
        ON Products.ProductID = OrderDetails.ProductID
        WHERE inserted.ExpirationDate < GETDATE();
    END
    END
GO
```

The status bar at the bottom indicates the query was completed successfully.

TRIGGERS

Trigger to Update Stock on Order

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'supermarket2' with various tables like 'Products', 'Orders', and 'OrderDetails'. The 'Triggers' folder under 'OrderDetails' contains a single trigger named 'tr_UpdateStockOnOrder'. The central pane displays the T-SQL script for this trigger:

```
USE [supermarket2]
GO
/****** Object: Trigger [dbo].[tr_UpdateStockOnOrder] Script Date: 22-Jan-23 3:41:14 PM *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER [dbo].[tr_UpdateStockOnOrder]
ON [dbo].[OrderDetails]
AFTER INSERT
AS
BEGIN
    UPDATE Products
    SET ProductQuantity = ProductQuantity - inserted.Quantity
    FROM Products
    JOIN inserted
    ON Products.ProductID = inserted.ProductID
END
```

Trigger to Calculate Total Cost

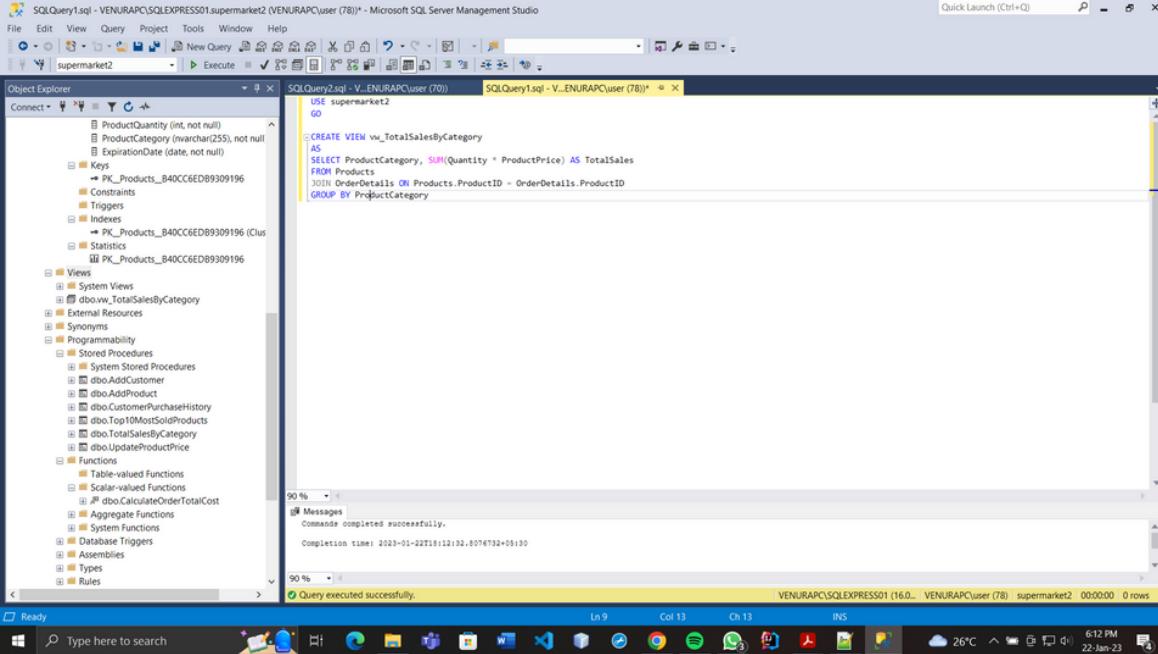
The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the same 'supermarket2' database with its tables. The 'Triggers' folder under 'OrderDetails' contains a trigger named 'tr_CalculateTotalCost'. The central pane displays the T-SQL script for this trigger:

```
-- the definition of the function.
USE supermarket2
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TRIGGER tr_CalculateTotalCost
ON OrderDetails
AFTER INSERT
AS
BEGIN
    DECLARE @OrderID INT;
    DECLARE @TotalCost DECIMAL(10, 2);
    SELECT @OrderID = OrderID, @TotalCost = SUM(Quantity * ProductPrice)
    FROM OrderDetails
    JOIN Products ON OrderDetails.ProductID = Products.ProductID
    GROUP BY OrderID
    UPDATE Orders
    SET TotalCost = @TotalCost
    WHERE OrderID = @OrderID
END
GO
```

At the bottom of the screen, a message indicates the command was completed successfully.

DATABASE VIEWS

View to get the Total Sales by Category

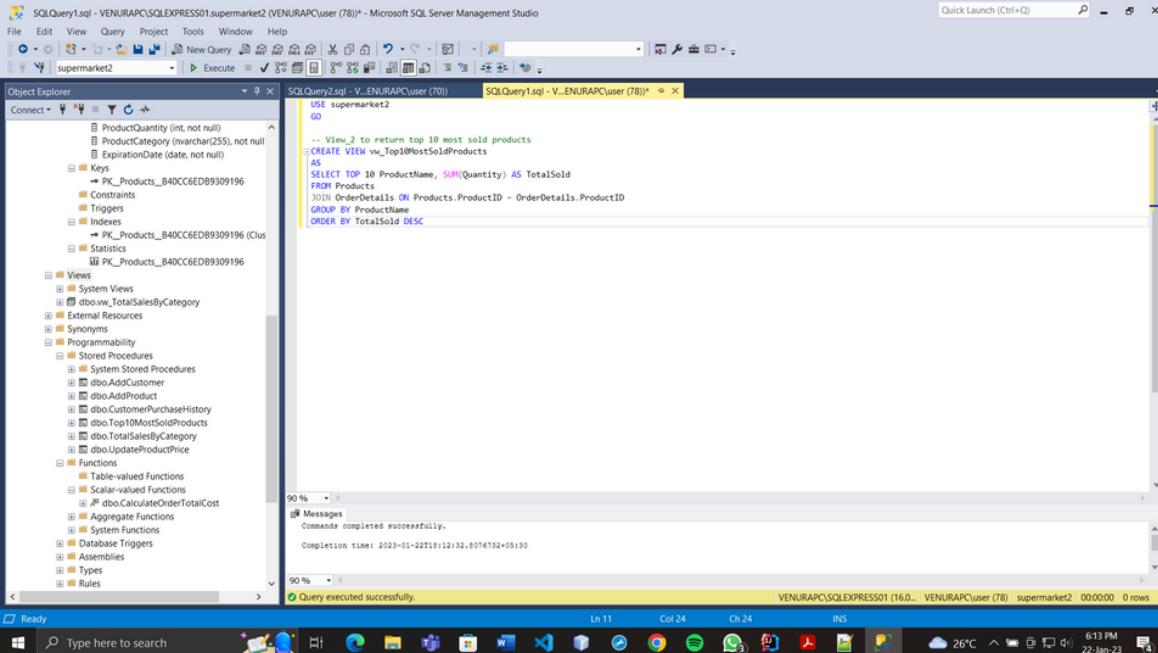


```
USE supermarket2
GO
CREATE VIEW vu_TotalSalesByCategory
AS
SELECT ProductCategory, SUM(Quantity * ProductPrice) AS TotalSales
FROM Products
JOIN OrderDetails ON Products.ProductID = OrderDetails.ProductID
GROUP BY ProductCategory
```

Messages
Command completed successfully.
Completion time 2023-01-22T18:12:32.8077732+08:00

Query executed successfully.

View to get the top 10 most sold Products



```
USE supermarket2
GO
-- View_2 to return top 10 most sold products
CREATE VIEW vu_Top10MostSoldProducts
AS
SELECT TOP 10 ProductName, SUM(Quantity) AS TotalSold
FROM Products
JOIN OrderDetails ON Products.ProductID = OrderDetails.ProductID
GROUP BY ProductName
ORDER BY TotalSold DESC
```

Messages
Command completed successfully.
Completion time 2023-01-22T18:12:32.8077732+08:00

Query executed successfully.

DATABASE VIEWS

View to get the Remaining Stock Level

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'supermarket2' is selected. In the center pane, a query window titled 'SQLQuery2.sql - VENURAPC\SQLEXPRESS01.supermarket2 (VENURAPC\user (78))' contains the following T-SQL code:

```
USE supermarket2
GO

-- View to get the remaining stock level
CREATE VIEW vu_RemainingStock
AS
SELECT Products.ProductID, ProductName, ProductQuantity - SUM(Quantity) as RemainingStock
FROM Products
LEFT JOIN OrderDetails ON Products.ProductID = OrderDetails.ProductID
GROUP BY Products.ProductID, ProductName, ProductQuantity
```

The status bar at the bottom right indicates the command was completed successfully at 6:15 PM on 22-Jan-23.

View to get the Customer Purchase History

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'supermarket2' is selected. In the center pane, a query window titled 'SQLQuery2.sql - VENURAPC\SQLEXPRESS01.supermarket2 (VENURAPC\user (78))' contains the following T-SQL code:

```
USE supermarket2
GO

-- View to return customer purchase history
CREATE VIEW vu_CustomerPurchaseHistory
AS
SELECT Customers.CustomerID, Customers.CustomerFirstnames,
Customers.CustomerLastname, ProductName, Quantity, ProductPrice,
OrderDate
FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID
JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
JOIN Products ON OrderDetails.ProductID = Products.ProductID
```

The status bar at the bottom right indicates the command was completed successfully at 6:14 PM on 22-Jan-23.

DATABASE FUNCTIONS

User Defined Function to Calculate Total Cost

```
USE supermarket2
GO
-- User-defined Function_1 to calculate the total cost of an order
CREATE FUNCTION CalculateOrderTotalCost(@OrderID INT)
RETURNS DECIMAL(10, 2)
AS
BEGIN
    DECLARE @TotalCost DECIMAL(10, 2);
    SELECT @TotalCost = SUM(Quantity * ProductPrice)
    FROM OrderDetails
    JOIN Products ON OrderDetails.ProductID = Products.ProductID
    WHERE OrderDetails.OrderID = @OrderID
    RETURN @TotalCost;
END
```

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists database objects like tables, views, and stored procedures. The main pane displays the creation script for a user-defined function named 'CalculateOrderTotalCost'. The status bar at the bottom indicates 'Query completed with errors.' and 'VENURAPC\SQLEXPRESS01 (16.0... VENURAPC\user (78) supermarket2 00:00:00 0 rows'.

Function inside Procedure to Add Customer

```
USE supermarket2
GO
CREATE PROCEDURE AddCustomer
@CustomerFirstName nvarchar(255),
@CustomerLastName nvarchar(255),
@CustomerEmail nvarchar(255),
@CustomerAddress nvarchar(255),
@CustomerPhone nvarchar(255),
@CustomerID INT OUTPUT
AS
BEGIN
    INSERT INTO Customers (CustomerFirstName, CustomerLastName,
    CustomerEmail, CustomerAddress, CustomerPhone)
    VALUES (@CustomerFirstName, @CustomerLastName, @CustomerEmail,
    @CustomerAddress, @CustomerPhone);
    SET @CustomerID = SCOPE_IDENTITY();
END
```

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists database objects. The main pane displays a 'CREATE PROCEDURE' statement for adding a customer. The status bar at the bottom indicates 'Commands completed successfully.' and 'VENURAPC\SQLEXPRESS01 (16.0... VENURAPC\user (78) supermarket2 00:00:00 0 rows'.

DATABASE FUNCTIONS

Function inside Procedure to Add Product

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer pane displays the database structure for 'supermarket2', including tables like 'Products' and various stored procedures. In the center, the 'SQLQuery2.sql' window contains the following T-SQL code:

```
USE supermarket2
GO
CREATE PROCEDURE AddProduct
    @ProductName nvarchar(255),
    @ProductPrice decimal(10, 2),
    @ProductQuantity int,
    @ProductCategory nvarchar(255),
    @ExpirationDate date,
    @ProductID INT OUTPUT
AS
BEGIN
    INSERT INTO Products (ProductName, ProductPrice, ProductQuantity, ProductCategory, ExpirationDate)
    VALUES (@ProductName, @ProductPrice, @ProductQuantity, @ProductCategory, @ExpirationDate)
    SET @ProductID = SCOPE_IDENTITY();
END
```

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

Function inside Procedure to Update Product Price

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer pane displays the database structure for 'supermarket2', including tables like 'Products' and various stored procedures. In the center, the 'SQLQuery2.sql' window contains the following T-SQL code:

```
USE [supermarket2]
GO
/*
***** (Object: StoredProcedure [dbo].[UpdateProductPrice] Script Date: 23-Jan-23 7:22:51 AM *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateProductPrice]
    @ProductId INT,
    @NewPrice decimal(10, 2),
    @RowsAffected INT OUTPUT
AS
BEGIN
    UPDATE Products
    SET ProductPrice = @NewPrice
    WHERE ProductId = @ProductId;
    SET @RowsAffected = @@ROWCOUNT;
END
```

The status bar at the bottom indicates "Connected. (1/1)" and "0 rows".

STORED PROCEDURES

Stored Procedure to get the Customer Purchase History

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'supermarket2'. In the center, the 'SQLQuery2.sql' window contains the following T-SQL code:

```
USE supermarket2
GO
CREATE PROCEDURE CustomerPurchaseHistory
@CustomerID INT
AS
BEGIN
SELECT ProductName, Quantity, ProductPrice, OrderDate
FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID
JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
JOIN Products ON OrderDetails.ProductID = Products.ProductID
WHERE Customers.CustomerID = @CustomerID
END
```

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

Stored Procedure to get the Bill Order

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'supermarket2'. In the center, the 'SQLQuery2.sql' window contains the following T-SQL code:

```
USE supermarket2
GO
-- Stored procedure _1 to get the bill order
CREATE PROCEDURE BillOrder(@OrderID INT)
AS
BEGIN
DECLARE @TotalCost DECIMAL(10, 2);
SET @TotalCost = 0;

-- Update stock levels
UPDATE Products
SET ProductQuantity = ProductQuantity - OrderDetails.Quantity
FROM Products
JOIN OrderDetails
ON Products.ProductID = OrderDetails.ProductID
WHERE OrderDetails.OrderID = @OrderID

-- Calculate total cost
SELECT @TotalCost = SUM(Quantity * ProductPrice)
FROM OrderDetails
JOIN Products ON OrderDetails.ProductID = Products.ProductID
WHERE OrderDetails.OrderID = @OrderID

-- Update total cost in the Orders table
UPDATE Orders
SET TotalCost = @TotalCost
WHERE OrderID = @OrderID
END
```

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

STORED PROCEDURES

Stored Procedure to get the Top 10 most sold Products

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including tables like 'Products' and 'OrderDetails'. The central pane displays the following T-SQL code:

```
USE supermarket2
GO
CREATE PROCEDURE Top10MostSoldProducts
AS
BEGIN
SELECT TOP 10 ProductName, SUM(Quantity) AS TotalSold
FROM Products
JOIN OrderDetails ON Products.ProductID = OrderDetails.ProductID
GROUP BY ProductName
ORDER BY TotalSold DESC
END
```

The status bar at the bottom indicates "Query executed successfully." and "Completion time: 2023-01-22T18:10:54.5742083+05:30".

Stored Procedure to Generate Financial Report

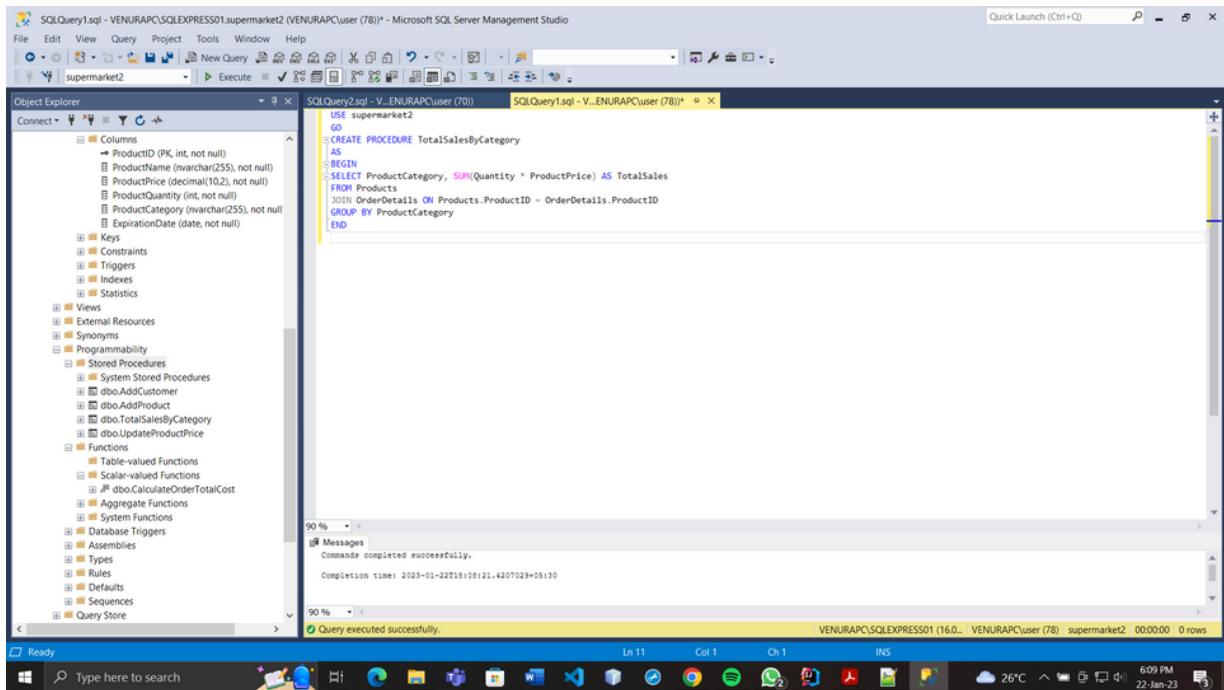
The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including tables like 'Products' and 'OrderDetails'. The central pane displays the following T-SQL code:

```
USE supermarket
GO
-- Stored procedure to get financial performance of the supermarket within that date range
CREATE PROCEDURE GenerateFinancialReport(@StartDate DATE, @EndDate DATE)
AS
BEGIN
DECLARE @TotalRevenue DECIMAL(10, 2);
DECLARE @TotalCost DECIMAL(10, 2);
DECLARE @Profit DECIMAL(10, 2);
-- Calculate total revenue, total cost, and profit
SELECT @TotalRevenue = SUM(Quantity * ProductPrice * (100 - OrderDetails.Discount)) / 100,
@TotalCost = SUM(Quantity * Products.CostPrice)
FROM OrderDetails
JOIN Products ON OrderDetails.ProductID = Products.ProductID
JOIN Orders ON OrderDetails.OrderID = Orders.OrderID
WHERE OrderDate BETWEEN @StartDate AND @EndDate
SET @Profit = @TotalRevenue - @TotalCost
-- Insert results into a table
INSERT INTO FinancialReport (StartDate, EndDate, TotalRevenue, TotalCost, Profit)
VALUES (@StartDate, @EndDate, @TotalRevenue, @TotalCost, @Profit)
END
```

The status bar at the bottom indicates "Query executed successfully." and "Completion time: 2023-01-22T19:01:50.2650308+05:30".

STORED PROCEDURES

Stored Procedure to get the Total Sales by Category



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'supermarket2' with various objects like Columns, Keys, Constraints, Triggers, Indexes, Statistics, Views, External Resources, Synonyms, Programmability (Stored Procedures, Functions), and Database Triggers. The central pane displays a T-SQL script for creating a stored procedure:

```
USE supermarket2
GO
CREATE PROCEDURE TotalSalesByCategory
AS
BEGIN
    SELECT ProductCategory, SUM(Quantity * ProductPrice) AS TotalSales
    FROM Products
    JOIN OrderDetails ON Products.ProductID = OrderDetails.ProductID
    GROUP BY ProductCategory
END
```

The status bar at the bottom indicates the command was completed successfully with a completion time of 2023-01-22T18:08:21.4207029+05:30.

CRITICAL EVALUATION

- In our solution we have implemented the ability to restrict and control access levels to the system. Only the users (such as admin/owner) with the necessary permissions can perform operations or modifications to the system and the database.
- Also we provide a detailed financial report providing an analysis of the financial performance of the business and other insights
- One aspect that we can improve upon and isn't covered in depth, in our system is the payment tracking feature. For example: Providing the ability to track the payment mode of the customers, and other insight into customer purchase payment trends.
- Additionally, the ability to evaluate supplier performance and history as well as other data to further provide insight into the performance of the business is not currently supported in our system.
- These shortcomings can be improved by tracking, collecting and storing the necessary data related to these metrics as well as coming up with effective methods to analyse this data in order to get useful reports that can lead to better decision making from the parties in charge of management and operation of the supermarket.
- In conclusion we believe our system manages to cover all the requirements and various verticals and aspects of the business and provides an effective solution to the management in order for them to run and oversee the supermarket business operations effectively and efficiently. Overall, the system is a complete solution to the problem scenario presented.

FUTURE IMPLEMENTATIONS

- One feature that can be looked into further and implemented in the future is to expand on the permissions aspect of the system. Providing granular control of permissions such as read, write, update, delete, etc. through the system by assigning permissions to the users of the system.
- Also, the advanced reporting aspect can be improved. Providing deeper insight using various data analysis methods to analyze trends, history and provide accurate predictions that are tailor-made according to the data in the system. These can be generated in the form of additional reports.
- Following up on the advanced data analysis even the accounting feature can be further improved by implementing advanced accounting analytics and techniques to further get an accurate picture/ representation for the owners, admins or any other management party to accurately make informed decisions that critical in day-to-day operations of the supermarket
- The system can also be further expanded upon, depending on multiple locations or branches that are opened by the business in the future and all of them can be connected to this system to manage all operations smoothly.

TEAM AND WORKBREAKDOWN



PU ID: 10820799
Name: HIKKADUWAGE SILVA

Planning and design and evaluation ER, Report, Create views, Create triggers normalization, SQL coding to create DB



PU ID: 10818175
Name: VENURA DEEGALLA

Access Level creation, ER, Report, Create triggers, Create functions, SQL coding to create DB tables



PU ID: 10818172
Name: SHAUN HENNEDIGE

Planning and design, Database Diagram,, Data dictionary, Report, Create stored procedures.



PU ID: 10750069
NAME: MADHAVA DOMBAGAHA

Presentation, ER, SQL coding to insert data into tables, Report.



PU ID: 10818163
Name: RATHNAYAKA RATHNAYAKA

Presentation, SQL coding to create DB tables, Data dictionary, Relational Schema



PU ID: 10819571
Name: DILEEP ABEYSINGHE

Presentation, Relational Schema, SQL coding to create DB tables, insert data into tables,