

Data Management Project

Part A Normalized Model

1a First Normal Form

Table Design

Sales_1NF			
	Column Name	Condensed Type	Nullable
🔑	Id	int	No
	OrderId	int	No
	DonutId	int	No
	Name	nvarchar(50)	No
	Description	nvarchar(250)	Yes
	UnitPrice	money	Yes
	Quantity	int	No
	SaleDate	date	No
	SpecialHandlingNotes	nvarchar(500)	Yes
	CustomerId	int	Yes
	CustomerFirstName	nvarchar(50)	Yes
	CustomerLastName	nvarchar(50)	Yes
	CustomerStreetAddress1	nvarchar(50)	Yes
	CustomerStreetAddress2	nvarchar(50)	Yes
	CustomerCity	nvarchar(50)	Yes
	CustomerState	nchar(2)	Yes
	CustomerZip	nchar(6)	Yes
	CustomerHomePhone	nchar(10)	Yes
	CustomerMobilePhone	nchar(10)	Yes
	CustomerOtherPhone	nchar(10)	Yes

```
CREATE TABLE [dbo].[Sales_1NF]
(
    [Id] INT NOT NULL IDENTITY(1,1),
    [OrderId] INT NOT NULL,
    [DonutId] INT NOT NULL,
    [Name] NVARCHAR(50) NOT NULL,
    [Description] NVARCHAR(250) NULL,
    [UnitPrice] MONEY NULL,
```

Duncan Nisbett
Student ID: #000856539
C170: Data Management - Applications


```
[Quantity] INT NOT NULL,  
[SaleDate] DATE NOT NULL,  
[SpecialHandlingNotes] NVARCHAR(500) NULL,  
[CustomerId] INT NULL,  
[CustomerFirstName] NVARCHAR(50) NULL,  
[CustomerLastName] NVARCHAR(50) NULL,  
[CustomerStreetAddress1] NVARCHAR(50) NULL,  
[CustomerStreetAddress2] NVARCHAR(50) NULL,  
[CustomerCity] NVARCHAR(50) NULL,  
[CustomerState] NCHAR(2) NULL,  
[CustomerZip] NCHAR(6) NULL,  
[CustomerHomePhone] NCHAR(10) NULL,  
[CustomerMobilePhone] NCHAR(10) NULL,  
[CustomerOtherPhone] NCHAR(10) NULL,  
CONSTRAINT [PK_Sales_1NF] PRIMARY KEY (Id)  
)
```

Reasoning


I took the Sales form sheet and reviewed the data to break out each individual artifact. The table has been broken up based on the requirements and the unique data points found within the form. From there I used a standard naming convention to give each data point a self describing name like, CustomerFirstName, to make a clear designation on the type of value one could find in the column. Each data point was also examined to determine what type of data it best represented. A whole number such as id or count column was assigned as an integer, any short text string stored as nchar, longer text strings stored as nvarchar, and then money for the unit price.

1b Second Normal Form


Table Design

Customer_2NF			
	Column Name	Condensed Type	Nullable
	CustomerId	int	No
	LastName	nvarchar(50)	No
	FirstName	nvarchar(50)	No
	MiddleName	nvarchar(50)	Yes
	Address1	nvarchar(250)	No
	Address2	nvarchar(250)	Yes
	City	nvarchar(50)	No
	State	nchar(2)	No
	Zip	nchar(6)	No
	CustomerHomePhone	nchar(10)	Yes
	CustomerMobilePhone	nchar(10)	Yes
	CustomerOtherPhone	nchar(10)	Yes

```
CREATE TABLE [dbo].[Customer_2NF]
(
    [CustomerId] INT NOT NULL IDENTITY(1,1),
    [LastName] NVARCHAR(50) NOT NULL,
    [FirstName] NVARCHAR(50) NOT NULL,
    [MiddleName] NVARCHAR(50) NULL,
    [Address1] NVARCHAR(250) NOT NULL,
    [Address2] NVARCHAR(250) NULL,
    [City] NVARCHAR(50) NOT NULL,
    [State] NCHAR(2) NOT NULL,
    [Zip] NCHAR(6) NOT NULL,
    [CustomerHomePhone] NCHAR(10) NULL,
    [CustomerMobilePhone] NCHAR(10) NULL,
    [CustomerOtherPhone] NCHAR(10) NULL,
    CONSTRAINT [PK_Customer_2NF] PRIMARY KEY (CustomerId)
)
```

Product_2NF			
	Column Name	Condensed Type	Nullable
	ProductId	int	No
	Name	nvarchar(50)	No
	Description	nvarchar(250)	No
	UnitPrice	money	No

```
CREATE TABLE [dbo].[Product_2NF]
(
    [ProductId] INT NOT NULL IDENTITY(1,1),
    [Name] NVARCHAR(50) NOT NULL,
    [Description] NVARCHAR(250) NOT NULL,
    [UnitPrice] MONEY NOT NULL,
    CONSTRAINT [PK_Product_2NF] PRIMARY KEY (ProductId)
)
```

Sales_2NF			
	Column Name	Condensed Type	Nullable
	Id	int	No
	OrderId	int	No
	SaleDate	date	No
	CustomerId	int	No
	ProductId	int	No
	Quantity	int	No
	SpecialHandlingNotes	nvarchar(500)	Yes

```
CREATE TABLE [dbo].[Sales_2NF]
(
    [Id] INT NOT NULL IDENTITY(1,1),
    [OrderId] INT NOT NULL,
    [SaleDate] DATE NOT NULL,
    [CustomerId] INT NOT NULL,
    [ProductId] INT NOT NULL,
    [Quantity] INT NOT NULL,
    [SpecialHandlingNotes] NVARCHAR(500) NULL,
    CONSTRAINT [PK_Sales_2NF] PRIMARY KEY (Id),
    CONSTRAINT [FK_Sales_Customer] FOREIGN KEY ([CustomerId]) REFERENCES
[Customer_2NF]([CustomerId]),
    CONSTRAINT [FK_Sales_Product] FOREIGN KEY ([ProductId]) REFERENCES
[Product_2NF]([ProductId])
)
```


)


Reasoning


The sales data was broken out to 3 sections. Product data to store the information into each individual item that can be sold. This allows the sales table to track each instance and the quantity that a product has been sold. The Customer table stores each separate customer to be able to re-use the same data for each occurrence of a sales instance. The Sales table tracks information specific to each transaction along with the individual components of the sales data. The current sales data still allows for duplication of information because each unique product sold on that sale requires a row. This will duplicate the OrderId, CustomerId, SaleDate, and the Special Handling Notes. Foreign Key constraints were added to require valid products and customers to be linked to a sales record.


1c Third Normal Form

Table Design

Customer_3NF			
	Column Name	Condensed Type	Nullable
	CustomerId	int	No
	LastName	nvarchar(50)	No
	FirstName	nvarchar(50)	No
	MiddleName	nvarchar(50)	Yes
	Address1	nvarchar(250)	No
	Address2	nvarchar(250)	Yes
	City	nvarchar(50)	No
	State	nchar(2)	No
	Zip	nchar(6)	No
	CustomerHomePhone	nchar(10)	Yes
	CustomerMobilePhone	nchar(10)	Yes
	CustomerOtherPhone	nchar(10)	Yes

Product_3NF			
	Column Name	Condensed Type	Nullable
	ProductId	int	No
	Name	nvarchar(50)	No
	Description	nvarchar(250)	No
	UnitPrice	money	No

Sales_3NF			
	Column Name	Condensed Type	Nullable
	Id	int	No
	OrderId	int	No
	SaleDate	date	No
	CustomerId	int	No
	SpecialHandlingNotes	nvarchar(500)	Yes

Line_Item_3NF			
	Column Name	Condensed Type	Nullable
	Id	int	No
	OrderId	int	No
	ProductId	int	No
	Quantity	int	No

Reasoning

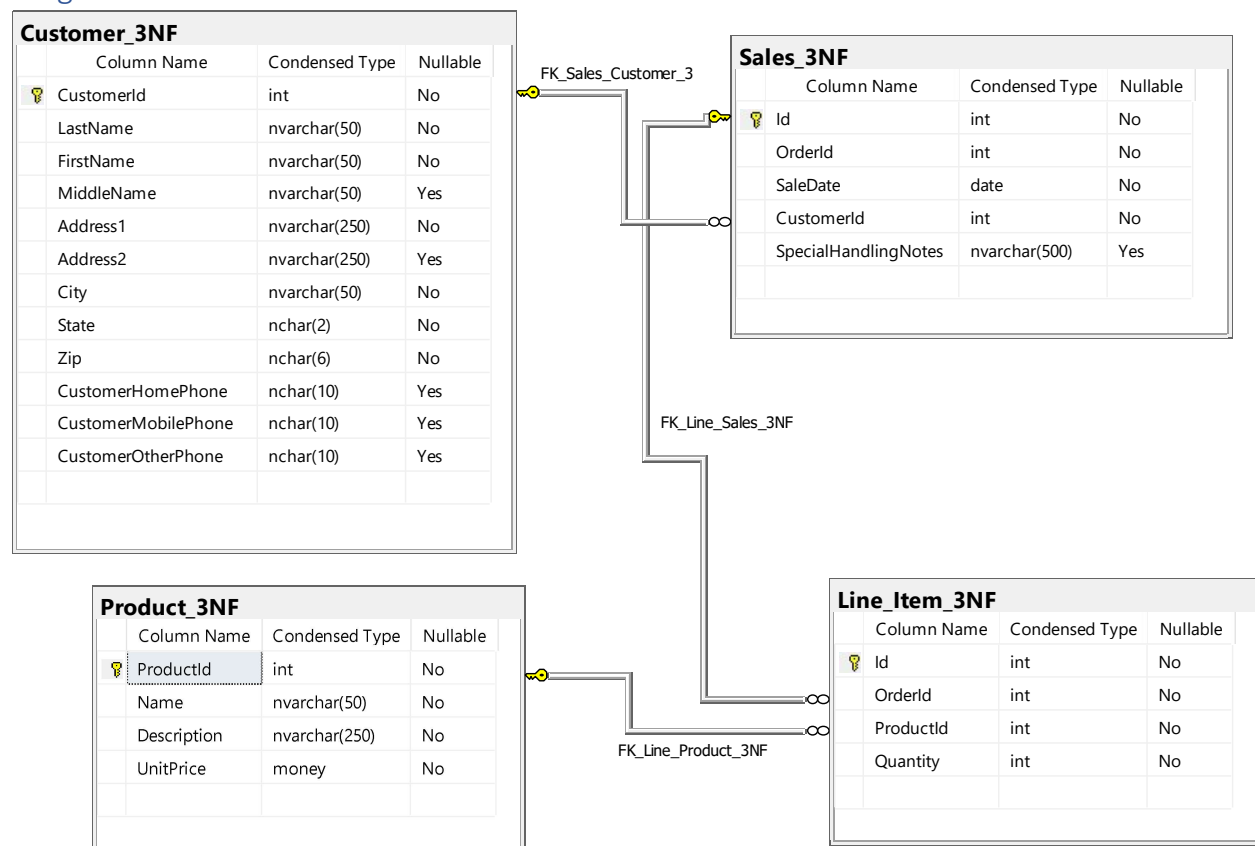
Most of the tables in this form are nearly the same as the second normal form. However, in the third normal form we have added an additional table called Line Item. This table breaks out the line item information that pertains to a sale to its own table. This allows the main Sales table to contain one row about the sale itself. No longer will there be multiple entries for the order number, sale date for that order, customer id for the order, or special handling notes. The Line Item table contains a foreign key constraint against the order table to only allow actual order records to be linked to a line item. It also contains a foreign key constraint against the product table to ensure we have only valid products linked

to an order. All tables use named primary keys for easier identification than the auto generated key names that SQL uses.

NOTE: The Line item table has a primary key as a singular column, but an improvement would be to either add a unique key using the OrderId and ProductId columns or to remove Id as a primary key and instead set the OrderId and ProductId as the primary key for the table. This would prevent duplicate line items on a single sales order record.

Part B Entity Relationship Diagram

Diagram



Explanation

The entities in my diagram are as requested in the requirements and were generated from within Microsoft SQL Server Management Studio. It includes 4 tables; Customer, Product, Sales, and Line Item table. The customer table is linked to the sales table as per the order form there is only one customer per order. This establishes a one-to-many relationship between customers and sales. A sale must have only one customer, but a customer can have many sales. A sale will contain 1 or more line items sold. The sale items are stored in the line item table. As a sale can have multiple line items, but a line item can only be linked to one sale this establishes a many-to-one relationship between sales and line items. The line item table is a ternary table as it intersects the data between a sale and a product. Finally, we have

the Product table which is tied to the line items. There can only be one instance of a donut in the product table, but that donut can be ordered many times in the line item table which indicates a one-to-many relationship between products and line items. Cardinality is enforced through restrictive foreign key constraints. FK_Sales_Customer_3 requires a valid customer to be required in the Sales table. FK_Line_Sales_3NF requires that any line items are linked to a valid sale record. FK_Line_Product_3NF requires that any product listed in the Line Item table exists within the Product table.

Part C Proof of Compiling

SQL Code

```
CREATE TABLE [dbo].[Customer_3NF]
(
    [CustomerId] INT NOT NULL IDENTITY(1,1),
    [LastName] NVARCHAR(50) NOT NULL,
    [FirstName] NVARCHAR(50) NOT NULL,
    [MiddleName] NVARCHAR(50) NULL,
    [Address1] NVARCHAR(250) NOT NULL,
    [Address2] NVARCHAR(250) NULL,
    [City] NVARCHAR(50) NOT NULL,
    [State] NCHAR(2) NOT NULL,
    [Zip] NCHAR(6) NOT NULL,
    [CustomerHomePhone] NCHAR(10) NULL,
    [CustomerMobilePhone] NCHAR(10) NULL,
    [CustomerOtherPhone] NCHAR(10) NULL,
    CONSTRAINT [PK_Customer_3NF] PRIMARY KEY (CustomerId)
)

CREATE TABLE [dbo].[Product_3NF]
(
    [ProductId] INT NOT NULL IDENTITY(1,1),
    [Name] NVARCHAR(50) NOT NULL,
    [Description] NVARCHAR(250) NOT NULL,
    [UnitPrice] MONEY NOT NULL,
    CONSTRAINT [PK_Product_3NF] PRIMARY KEY (ProductId)
)

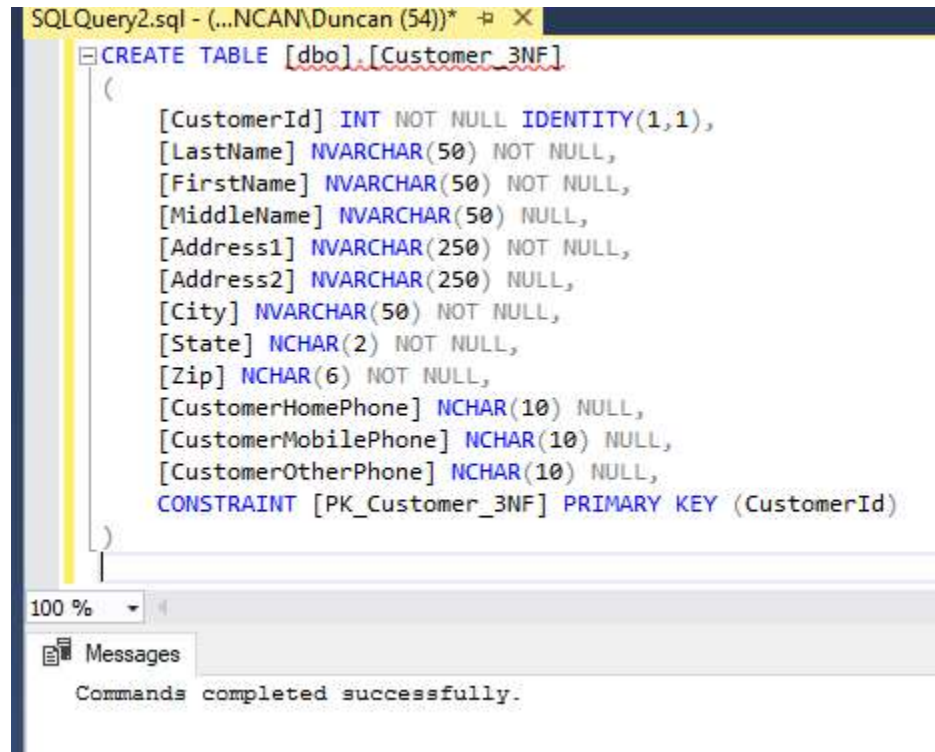
CREATE TABLE [dbo].[Sales_3NF]
(
    [Id] INT NOT NULL IDENTITY(1,1),
    [OrderId] INT NOT NULL,
    [SaleDate] DATE NOT NULL,
    [CustomerId] INT NOT NULL,
    [SpecialHandlingNotes] NVARCHAR(500) NULL,
    CONSTRAINT [PK_Sales_3NF] PRIMARY KEY (Id),
    CONSTRAINT [FK_Sales_Customer_3] FOREIGN KEY ([CustomerId]) REFERENCES
[Customer_3NF]([CustomerId])
)

CREATE TABLE [dbo].[Line_Item_3NF]
(
    [Id] INT NOT NULL IDENTITY(1,1),
```


Duncan Nisbett
Student ID: #000856539
C170: Data Management - Applications

```
[OrderId] INT NOT NULL,  
[ProductId] INT NOT NULL,  
[Quantity] INT NOT NULL  
CONSTRAINT [PK_Line_3NF] PRIMARY KEY (Id),  
CONSTRAINT [FK_Line_Product_3NF] FOREIGN KEY ([ProductId]) REFERENCES  
[Product_3NF]([ProductId]),  
CONSTRAINT [FK_Line_Sales_3NF] FOREIGN KEY ([OrderId]) REFERENCES  
[Sales_3NF]([Id])  
)
```

Screenshot Proof



The screenshot shows a SQL Server Enterprise Manager window titled "SQLQuery2.sql - (...NCAN\Duncan (54))*". The main pane displays the following T-SQL code:

```
CREATE TABLE [dbo].[Customer_3NF]  
(  
    [CustomerId] INT NOT NULL IDENTITY(1,1),  
    [LastName] NVARCHAR(50) NOT NULL,  
    [FirstName] NVARCHAR(50) NOT NULL,  
    [MiddleName] NVARCHAR(50) NULL,  
    [Address1] NVARCHAR(250) NOT NULL,  
    [Address2] NVARCHAR(250) NULL,  
    [City] NVARCHAR(50) NOT NULL,  
    [State] NCHAR(2) NOT NULL,  
    [Zip] NCHAR(6) NOT NULL,  
    [CustomerHomePhone] NCHAR(10) NULL,  
    [CustomerMobilePhone] NCHAR(10) NULL,  
    [CustomerOtherPhone] NCHAR(10) NULL,  
    CONSTRAINT [PK_Customer_3NF] PRIMARY KEY (CustomerId)  
)
```

The bottom pane, titled "Messages", shows the status: "Commands completed successfully."

```
SQLQuery2.sql - (...NCAN\Duncan (54))*  
CREATE TABLE [dbo].[Sales_3NF]  
(  
    [Id] INT NOT NULL IDENTITY(1,1),  
    [OrderId] INT NOT NULL,  
    [SaleDate] DATE NOT NULL,  
    [CustomerId] INT NOT NULL,  
    [SpecialHandlingNotes] NVARCHAR(500) NULL,  
    CONSTRAINT [PK_Sales_3NF] PRIMARY KEY (Id),  
    CONSTRAINT [FK_Sales_Customer_3] FOREIGN KEY ([CustomerId]) REFERENCES [Customer_3NF]([CustomerId]),  
)
```

100 %
Messages
Commands completed successfully.

```
SQLQuery2.sql - (...NCAN\Duncan (54))*  
CREATE TABLE [dbo].[Product_3NF]  
(  
    [ProductId] INT NOT NULL IDENTITY(1,1),  
    [Name] NVARCHAR(50) NOT NULL,  
    [Description] NVARCHAR(250) NOT NULL,  
    [UnitPrice] MONEY NOT NULL,  
    CONSTRAINT [PK_Product_3NF] PRIMARY KEY (ProductId)  
)
```

100 %
Messages
Commands completed successfully.

```
SQLQuery2.sql - (...NCAN\Duncan (54))*
CREATE TABLE [dbo].[Line_Item_3NF]
(
    [Id] INT NOT NULL IDENTITY(1,1),
    [OrderId] INT NOT NULL,
    [ProductId] INT NOT NULL,
    [Quantity] INT NOT NULL
CONSTRAINT [PK_Line_3NF] PRIMARY KEY (Id),
CONSTRAINT [FK_Line_Product_3NF] FOREIGN KEY ([ProductId]) REFERENCES [Product_3NF]([ProductId]),
CONSTRAINT [FK_Line_Sales_3NF] FOREIGN KEY ([OrderId]) REFERENCES [Sales_3NF]([Id])
)
100 %
Messages
Commands completed successfully.
```

Part D Customer View

SQL Code

```
CREATE VIEW [dbo].[vw_Customer] AS
SELECT CONCAT(FirstName, ' ', LastName) AS Name, LastName, Firstname, MiddleName,
Address1, Address2, City, State, Zip, CustomerHomePhone, CustomerMobilePhone,
CustomerOtherPhone
FROM [Customer_3NF]
```

Screenshot Proof

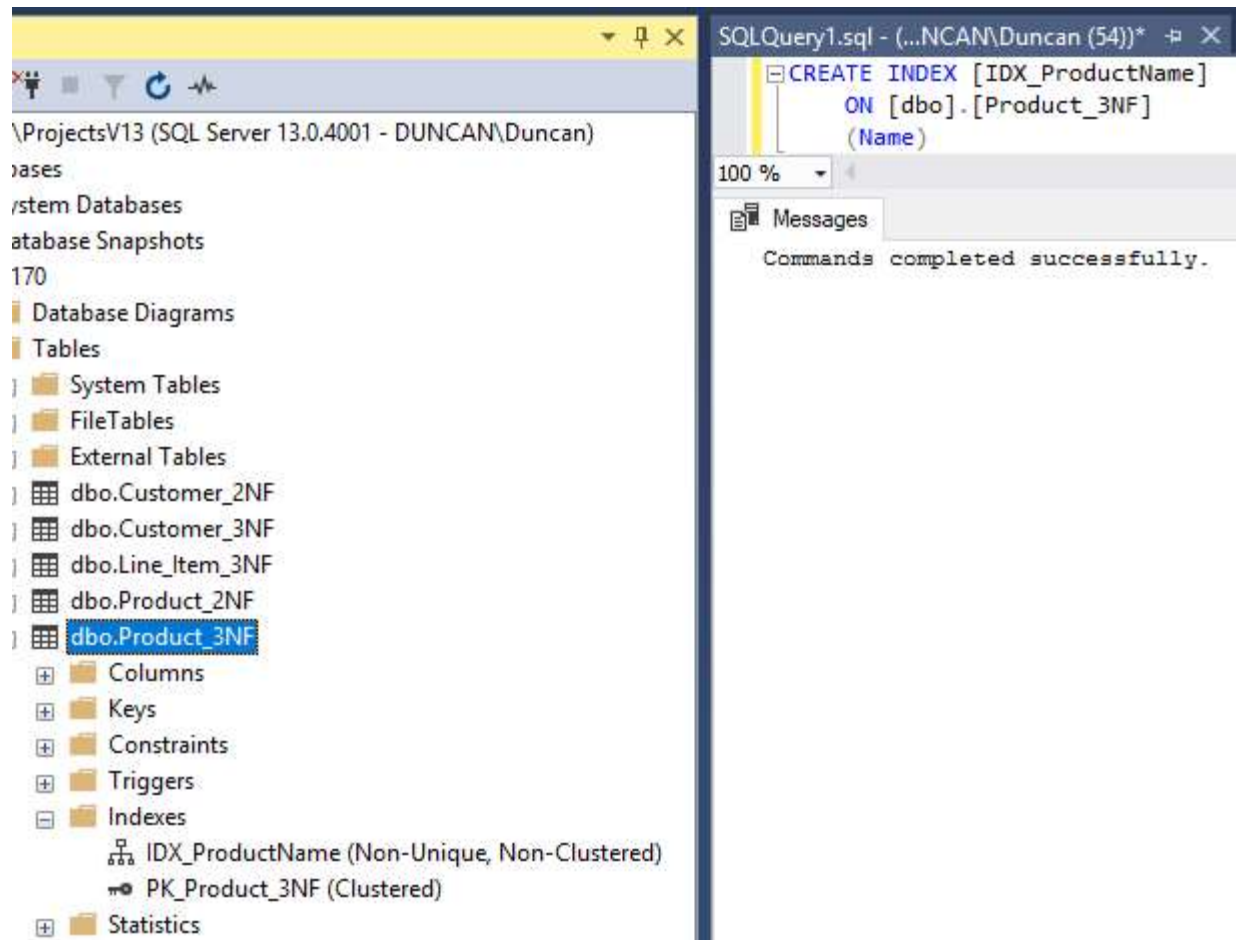
```
SQLQuery1.sql - (...NCAN\Duncan (54))*
CREATE VIEW [dbo].[vw_Customer] AS
SELECT CONCAT(FirstName, ' ', LastName) AS Name, LastName, Firstname,
MiddleName, Address1, Address2, City, State, Zip, CustomerHomePhone,
CustomerMobilePhone, CustomerOtherPhone
FROM [Customer_3NF]
100 %
Messages
Commands completed successfully.
```

Part E Create Product Name Index

SQL Code

```
CREATE INDEX [IDX_ProductName]
ON [dbo].[Product_3NF]
(Name)
```

Screenshot Proof



Part F Populate Tables

SQL Code

```
DECLARE @customerId int;
DECLARE @productId int;
DECLARE @saleId int;

INSERT INTO Customer_3NF (LastName, FirstName, MiddleName, Address1, Address2,
City, Zip, State, CustomerHomePhone, CustomerMobilePhone, CustomerOtherPhone)
VALUES (
    'Nisbett', 'Duncan', 'Edward', '123 Main Street', 'Apt #B4',
    'Denver', '49123', 'CO', '5551234567', '5553219966', '5559876543'
);

SELECT @customerId = SCOPE_IDENTITY();

INSERT INTO Product_3NF (Name, Description, UnitPrice)
VALUES (
    'Glazed', 'Glazed Donut', 1.75
);
```

Duncan Nisbett
Student ID: #000856539
C170: Data Management - Applications

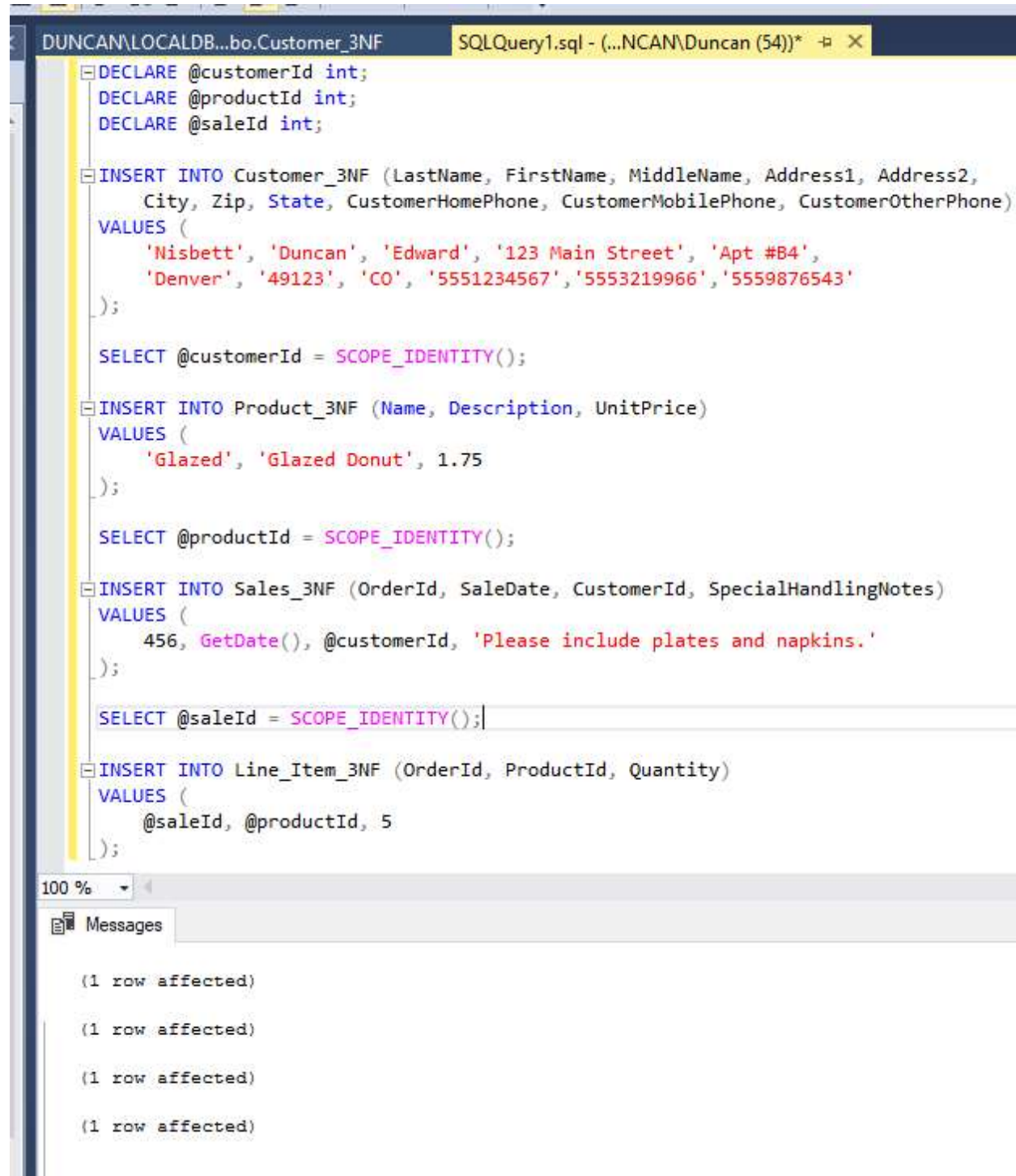
```
SELECT @productId = SCOPE_IDENTITY();

INSERT INTO Sales_3NF (OrderId, SaleDate, CustomerId, SpecialHandlingNotes)
VALUES (
    456, getDate(), @customerId, 'Please include plates and napkins.'
);

SELECT @saleId = SCOPE_IDENTITY();

INSERT INTO Line_Item_3NF (OrderId, ProductId, Quantity)
VALUES (
    @saleId, @productId, 5
);
```

Screenshot Proof



The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows a SQL query script named 'SQLQuery1.sql' with the following T-SQL code:

```
DECLARE @customerId int;
DECLARE @productId int;
DECLARE @saleId int;

INSERT INTO Customer_3NF (LastName, FirstName, MiddleName, Address1, Address2,
    City, Zip, State, CustomerHomePhone, CustomerMobilePhone, CustomerOtherPhone)
VALUES (
    'Nisbett', 'Duncan', 'Edward', '123 Main Street', 'Apt #B4',
    'Denver', '49123', 'CO', '5551234567', '5553219966', '5559876543'
);

SELECT @customerId = SCOPE_IDENTITY();

INSERT INTO Product_3NF (Name, Description, UnitPrice)
VALUES (
    'Glazed', 'Glazed Donut', 1.75
);

SELECT @productId = SCOPE_IDENTITY();

INSERT INTO Sales_3NF (OrderId, SaleDate, CustomerId, SpecialHandlingNotes)
VALUES (
    456, GetDate(), @customerId, 'Please include plates and napkins.'
);

SELECT @saleId = SCOPE_IDENTITY();

INSERT INTO Line_Item_3NF (OrderId, ProductId, Quantity)
VALUES (
    @saleId, @productId, 5
);
```

The bottom pane, titled 'Messages', shows the execution results for each statement:

```
(1 row affected)
(1 row affected)
(1 row affected)
(1 row affected)
```

Part G Display Values and Complex Join

Queries

```
SELECT * FROM Customer_3NF;
```

Duncan Nisbett
Student ID: #000856539
C170: Data Management - Applications

```
SELECT * FROM Product_3NF;  
SELECT * FROM Sales_3NF;  
SELECT * FROM Line_Item_3NF;
```

Screenshot Proof

The screenshot shows a SQL query window with the following text:

```
SELECT * FROM Customer_3NF;  
SELECT * FROM Product_3NF;  
SELECT * FROM Sales_3NF;  
SELECT * FROM Line_Item_3NF;
```

The results grid displays four tables:

CustomerId	LastName	FirstName	MiddleName	Address1	Address2	City	State	Zip	CustomerHomePhone	CustomerMobilePhone	CustomerOtherPhone
4	Nisbett	Duncan	Edward	123 Main Street	Apt #B4	Denver	CO	49123	5551234567	5553219966	5559876543

ProductId	Name	Description	UnitPrice
3	Glazed	Glazed Donut	1.75

Id	OrderId	SaleDate	CustomerId	SpecialHandlingNotes
3	456	2018-02-06	4	Please include plates and napkins.

Id	OrderId	ProductId	Quantity
3	3	3	5

Complex Join SQL

Quick

```
SELECT *  
FROM Line_Item_3NF l  
    INNER JOIN Product_3NF p ON p.ProductId = l.ProductId  
    INNER JOIN Sales_3NF s ON s.id = l.OrderId  
    INNER JOIN Customer_3NF c ON c.CustomerId = s.CustomerId
```

Proper

```
SELECT s.SaleDate, s.OrderId, c.CustomerId, c.FirstName, c.LastName, c.Address1,  
       c.Address2, c.City, c.State, c.Zip, c.CustomerHomePhone, c.CustomerMobilePhone,  
       c.CustomerOtherPhone,  
       l.Quantity, p.ProductId, p.Name, p.Description, p.UnitPrice,  
       s.SpecialHandlingNotes  
FROM Line_Item_3NF l  
    INNER JOIN Sales_3NF s ON s.Id = l.OrderId  
    INNER JOIN Product_3NF p ON p.ProductId = l.ProductId  
    INNER JOIN Customer_3NF c ON c.CustomerId = s.CustomerId
```

Complex Join Screenshot

Quick

The screenshot shows a SQL query window with the following text:

```
SELECT *  
FROM Line_Item_3NF l  
    INNER JOIN Product_3NF p ON p.ProductId = l.ProductId  
    INNER JOIN Sales_3NF s ON s.id = l.OrderId  
    INNER JOIN Customer_3NF c ON c.CustomerId = s.CustomerId
```

The results grid displays a single table with the following data:

Id	OrderId	ProductId	Quantity	ProductId	Name	Description	UnitPrice	Id	OrderId	SaleDate	CustomerId	SpecialHandlingNotes	CustomerId	LastName	FirstName	MiddleName	Address1
3	3	3	5	3	Glazed	Glazed Donut	1.75	3	456	2018-02-06	4	Please include plates and napkins.	4	Nisbett	Duncan	Edward	123 Main

Duncan Nisbett
Student ID: #000856539
C170: Data Management - Applications

Proper

SQLQuery1.sql - (...NCAN\Duncan (54))

```
SELECT s.SaleDate, s.OrderId, c.CustomerId, c.FirstName, c.LastName, c.Address1,
       c.Address2, c.City, c.State, c.Zip, c.CustomerHomePhone, c.CustomerMobilePhone, c.CustomerOtherPhone,
       1.Quantity, p.ProductId, p.Name, p.Description, p.UnitPrice, s.SpecialHandlingNotes
FROM Line_Item_3NF l
     INNER JOIN Sales_3NF s ON s.Id = l.OrderId
     INNER JOIN Product_3NF p ON p.ProductId = l.ProductId
     INNER JOIN Customer_3NF c ON c.CustomerId = s.CustomerId
```

100 %

	SaleDate	OrderId	CustomerId	FirstName	LastName	Address1	Address2	City	State	Zip	CustomerHomePhone	CustomerMobilePhone	CustomerOtherPhone	Quantity	ProductId	Name	Description	UnitPrice	SpecialHandlingNotes
1	2018-02-06	456	4	Duncan	Nisbett	123 Main Street	Apt PB4	Denver	CO	49123	5551234567	5553219966	5559876543	5	3	Glazed	Glazed Donut	1.75	Please include plates and napkins.