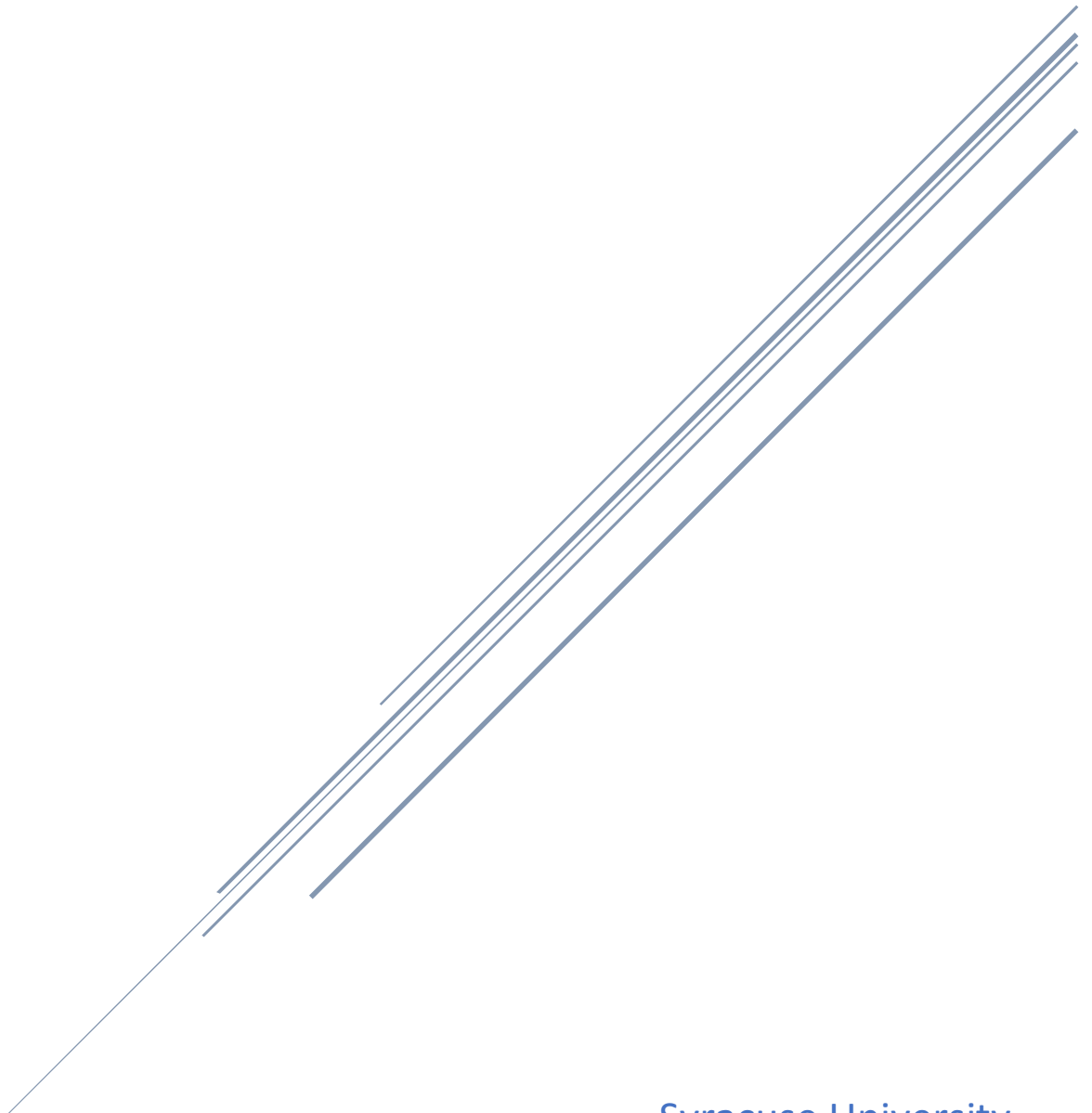
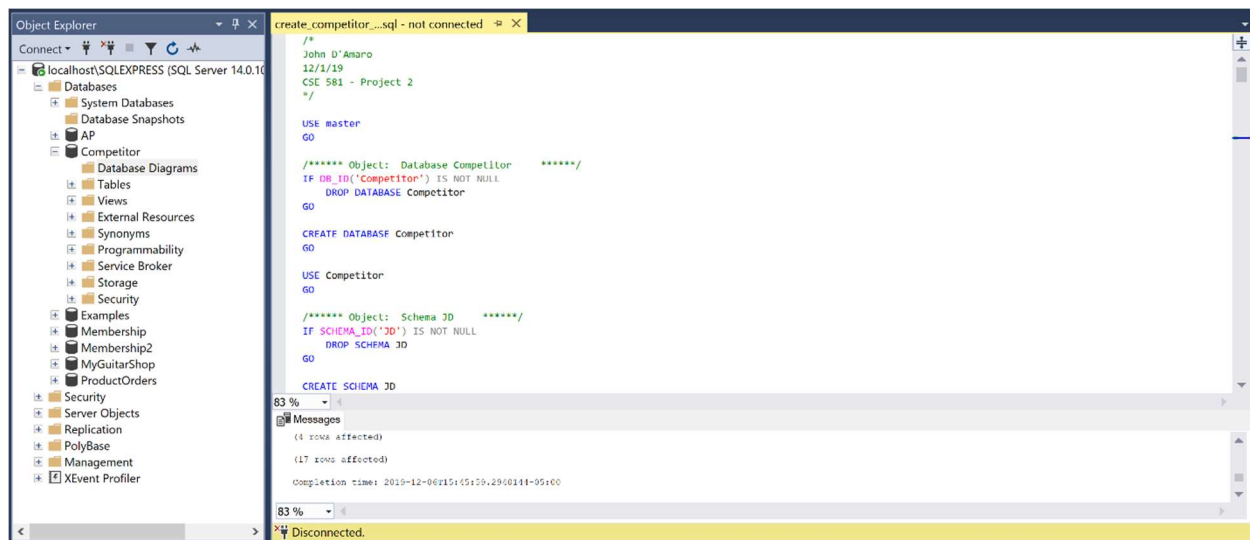


# PROJECT 2: AMAZON COMPETITOR DATABASE DESIGN

John D'Amaro, Master's in Engineering Management



Syracuse University  
Dr. Ercanli – CSE 581



## Abstract:

Project 2: Amazon Competitor Database Design amercers the student to go from the ground up in designing a database. Given a business problem, one analyses the structure of an Amazon-like company to format a database that would be used for said company. An enormous amount of consideration goes into how customers utilize the front end of the application and how those interactions influence how the back end is structured. Considerations such as: how a consumer buys a product, which warehouse delivers the chosen product and how that shipping cost is determined. Not only is project largely focused around the design of the actual database, but it tests the skills of how affective someone can create database objects such as views, procedures and transactions.

Some decisions that I made while designing this database include calculating shipping costs derived from ZIP code region and shipping service, customers can review the same product but only if it's ordered more than once and a customer can't have two of the same products on their wish list. I believe that these are accurate representations of how Amazon structure's their business model and how a competitor would structure theirs. Throughout the rest of this report, one will be guided through the reasoning behind these business decisions and how this is an optimal database design.

## Business Problem:

Your job is to create a database for an Amazon competitor. This database is to contain the following suggested information but more shall be added as you deem necessary. Please note that each order has a single shipping address, and can contain multiple products from one or more warehouses. Multiple warehouses may have the same product and the originating warehouse for a shipment is chosen based on the shipping cost, which can be determined by the zip codes of the originating warehouse and the shipping address. Suppliers may also have multiple products in various warehouses.

### Customers:

- Name
- Username
- Email
- Phone numbers (home, cell, business)
- Address book (several shipping and billing addresses)
  - City
  - State
  - Street
- Orders/Returns Information
- Wishlist
- Reviews
  - Date when the review is left
  - Which Order
  - Score
  - Text
  - Which product

### Products:

- ProductID
- ProductName
- Description
- Price
- Customer ratings:
  - Text
  - Score
  - Date when created
  - Customer who left it

### Suppliers:

- SupplierID
- Name
- Address
- Phone
- EmailAddress
- Products information:
  - ProductID
  - Number of Products Available

### Orders/Returns:

- OrderNumber
- Status (Ready to go, Shipped, Delivered, Return)
- OrderDate
- OrderItems:
  - ProductID
  - ProductName
  - Quantity
  - UnitPrice
- TotalPrice
- Shipping Service (USPS, FedEx, ...)
- Shipping Address
  - City
  - State
  - Street
- Shipping fare (depends on the address between the warehouse and the shipping address)
- Expected Shipping date
- Actual Shipping date
- Shipping Information

### Warehouses:

- WarehouseID
- Address
  - City
  - State
  - Street
- Stored Products:
  - ProductID
  - SupplierID
  - ProductName
  - Number in stock
  - Number on the way
  - Number in return

# Table of Contents

Abstract: .....	1
I. Database Design .....	4
Introduction.....	4
Design Consideration and Decisions .....	5
II. Implementation .....	9
III. Testing.....	11
IV. Conclusions.....	17
V. Appendix .....	18
A. Conceptual E/R Diagram .....	18
B. Entire Entity E/R Diagram .....	19
C. Product Portion of E/R Diagram.....	20
D. Orders Portion of E/R Diagram .....	22
E. Customers Portion of E/R Diagram .....	21

## I. Database Design

### Introduction

Amazon is the biggest online retailer in the world with over 175 warehouses within the US. On top of its extraordinary supply chain management, Amazon sells hundreds of millions of products annually which is a feat on its own. With all of this to factor in, how is it possible for this gargantuan company to keep track of all these transactions, shipments and product in stock? An extremely well-managed database that organizes every aspect of their operations. From customer and supplier information to shipping costs and optimal delivery routes. Every ounce their business is optimized to the second that your package is dropped off at your door and I believe that it is thanks to an efficient database management system.

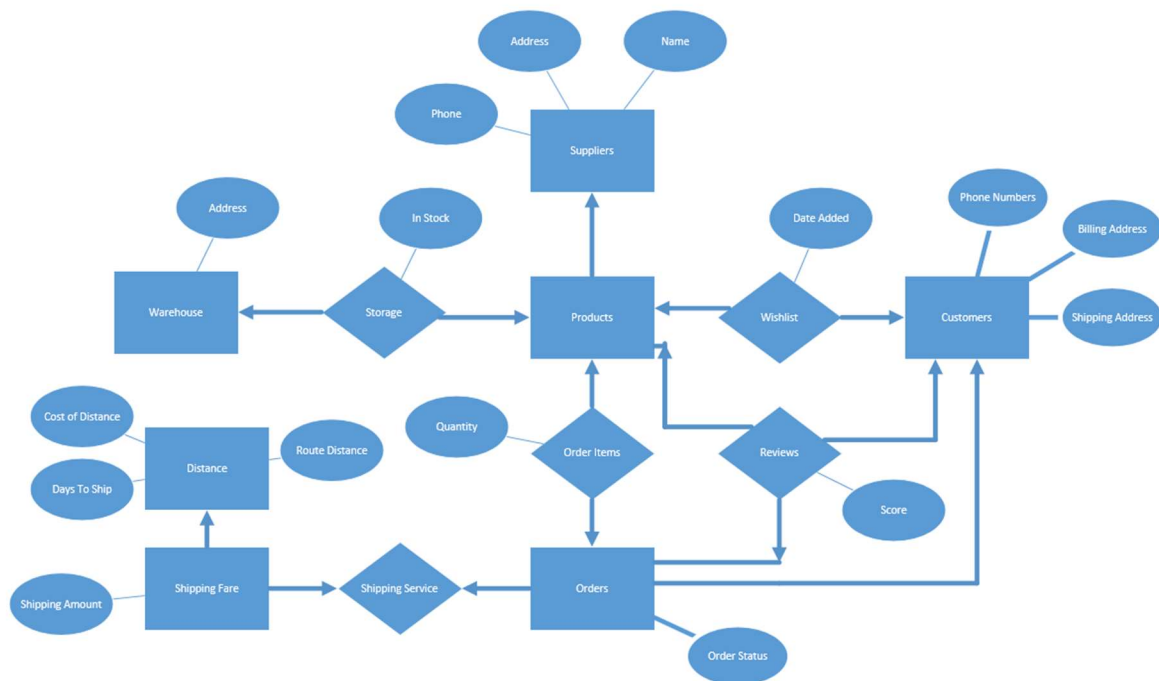
A database management system for a colossal company, such as Amazon, is categorized as a relational database. This type of database allows for relationships between tables with the help of primary keys. These keys are inserted into other tables, known as foreign keys, to relate the two tables together. For example, a relationship between a Customers table and a Products table is common. Further, there are three major relationships that tables can have but the most popular are one-to-many. Although not important now, this concept will be explained more in depth when explaining the design decisions of my database.

One of largest issues with a database of this size, that contains millions of rows of data, is performance. On average, Amazon processes 26.5M transactions a day. When working with datasets of a couple thousands of rows, SQL Server can process a query less than a second but with millions those transactions are going to slow down drastically. When success is measured on how fast one can get a consumer to purchase their product, a company needs to optimize how quickly their transactions are processed. Utilizing indexes, procedures and views will allow for these performance upgrades.

Although it doesn't seem like it, security goes together with performance for being equally as important to performance. One must manage which users can retrieve/modify data or create/alter database objects. Also, these users must be able to have concurrent access to this data which is a huge give and take relationship. If the database is the least restrictive on concurrency isolation, it may perform the highest but there are also the most amount of issues. However, if it's the most restrictive, there will be more server overhead to manage locks and cause severe performance problems.

All said and done, database design incorporates every aspect of performance, security and concurrency. Not only the actual design but also the queries that are used to retrieve data within. Using concise queries, procedures and trigger while having a fully functioning and efficient database is a formula for company success.

## Design Consideration and Decisions



**\*\*Figure 1: Conceptual E/R Diagram w/ Attributes<sup>1</sup>**

Depicted above, Figure 1 is the overall layout of the database design I have chosen. This diagram proves to be abstract enough for management personnel to understand key points while also technical enough for technical personnel follow. The three tables that make of most of the relationships within this database are the Customer, Order and Product tables. Starting at the top right of the figure with the Customers entity, it is possible for one customer to have multiple billing and shipping address as well as phone numbers. As mentioned before, this is one of the most powerful and useful functions of a relational database. Speaking in a general sense, a relationship is a line that is pointed to or from an entity and connects with another entity.

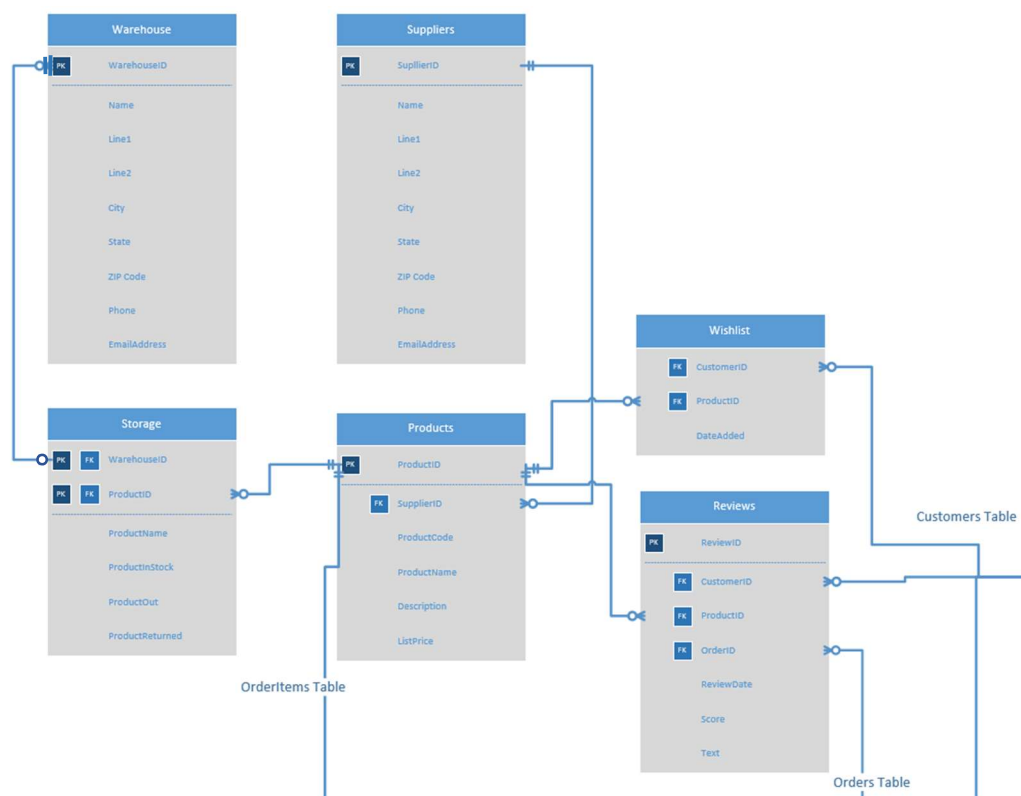
From right to left, starting at the Customers table, the database reads as such:

"One customer can have many orders, reviews and items on their Wishlist. On their Wishlist, there can be many products. Those reviews can be on many products from many orders. One supplier has many products, where one or many warehouses may store these products. One order may have many product items per order, which may come from one or multiple warehouses. One shipping service will ship these orders which will contribute in calculating the shipping cost. However, if the products are coming from different ZIP Code regions, there will be multiple shipping costs that calculate one total ship cost."

<sup>1</sup> All figures are also located in the Appendix.

Although this concept diagram shows generalities of the design, there are many specifics that it does not define. Such as, a customer can review the same product twice but only if that product is ordered again, they cannot leave multiple reviews from the same order. The same product cannot be on a customer's Wishlist twice. Two suppliers can have the same type of product, but not identical products. A warehouse could store a desired product but its currently out of stock, now a warehouse in a different region needs to ship that product which will increase the shipping cost. These business logic constraints cannot be depicted in Figure 1.

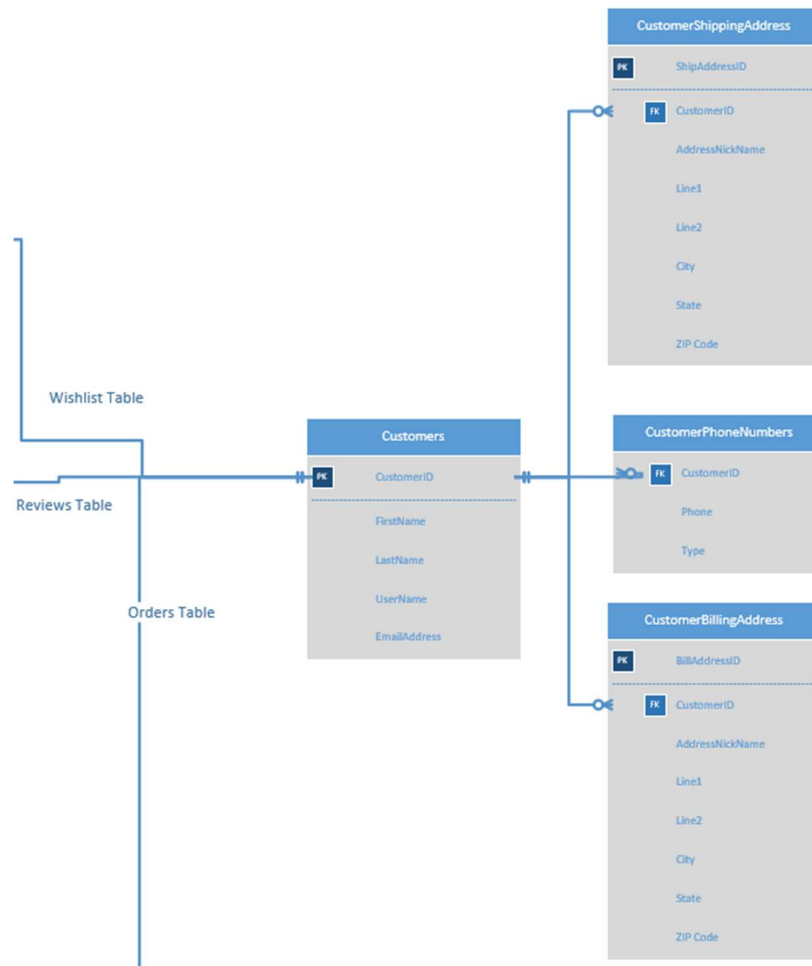
Turn your attention to **Section B in the Appendix** of the report for the E/R diagram that lists each entities attributes, primary/foreign keys and relationships between tables. This diagram was created to explore and showcase more of these elaborate relationships that the conceptual diagram could not show.



**\*\*Figure 2: Products Table Portion of the E/R Entity Diagram**

Let's look at the top left side of Figure 2. A warehouse has a singular address and contact information; however, it can store many products. Within the storage entity is a composite key constriction that will not allow the combination of WarehouseID and ProductID to appear twice because this combination must be unique. This constriction minimizes redundancy within the database. Since the SupplierID is a foreign key within the Products table, each product is directly assigned a supplier that manufactures that product. Also, the ProductCode that's assigned it each product is unique, meaning that no one product can be the same. The rest of the relationships state that many products can comprise of one order, saved

on a Wishlist or review. Most of the entities structured in this portion of the overall design were suggested in the problem statement of the project.

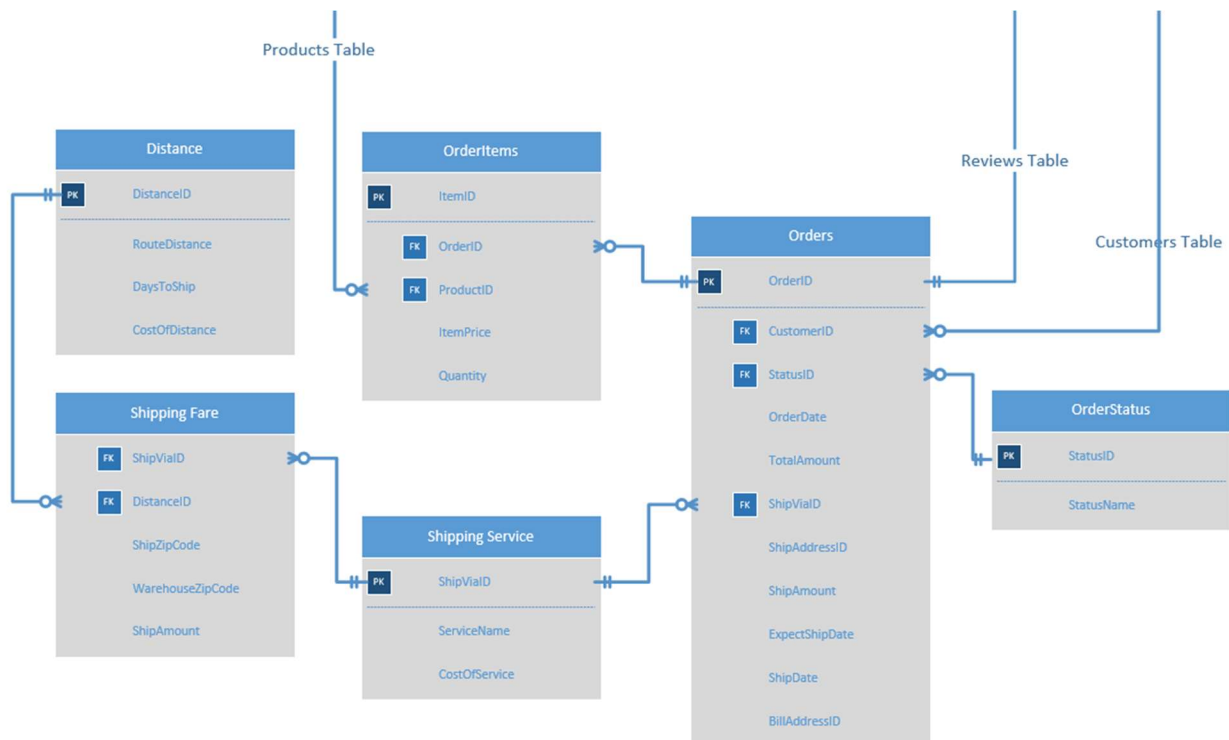


**\*\*Figure 3: Customers Table Portion of the E/R Entity Diagram**

Figure 3 shows the Customers portion of the total database diagram. The description in the conceptual diagram still applies; one customer can have many billing and shipping addresses, phone numbers, products on their Wishlist, reviews of products, and orders. The only intuitive part of this portion of the design is that customers directly interact with their products by reviewing them as well as placing them on their Wishlist's.

The most complicated part of the database design comes in the last section, how orders interact with customers and products. Below, Figure 4 shows the interactions with each order and how shipping information is calculated.





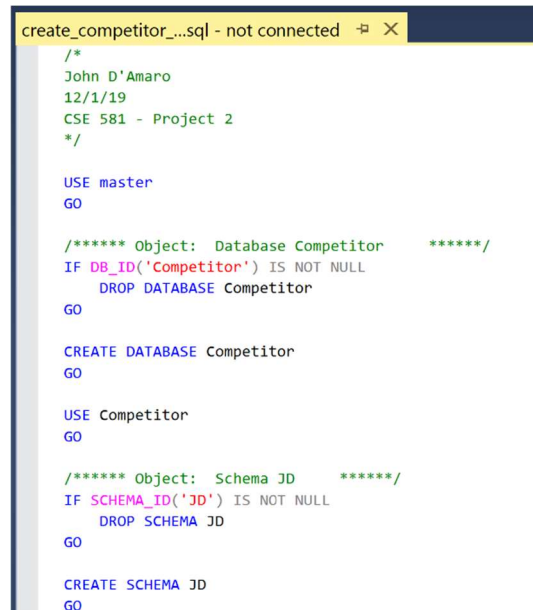
\*\*Figure 4: Orders Table Portion of the E/R Entity Diagram

A singular order has an enormous amount of information that follows it. A customer orders a couple products, that order has a status; whether it's ready, shipped, delivered or returned. Other information includes the date that order was processed, how much it costs, where it's going, total shipping cost etc. The issue is: which warehouse, that's the product that you order, is shipping it to you and what determines that cost? After some research, I determined that there are three shipping companies a customer could choose from: USPS, UPS, and FedEx. The base cost of shipping is derived from which company delivers your package, USPS costing the least while FedEx would be the most expensive.

From there, the ZIP Codes of the customer and each warehouse are considered. In the US, the first digit of a ZIP code determines a general region of a couple of states. For example, ZIP Codes starting with the number 9 consists of states along the west coast while ZIP Codes starting with the number 3 consists of Florida and it's bordering states. Inside the Distance entity I define four distances that have additional costs associated with them. If the warehouse that stores the desired product is within the same ZIP Code region as the customer, there is no additional charge to the base shipping rate. However, if the warehouse is in adjacent region, more than one away or on the opposite coast, the additional cost starts at \$5 and could reach up to \$15 dollars. I believe that this is a relatively accurate way to show how shipping costs are calculated. If a customer wants higher quality shipping, they must pay a price premium. If their desired product is across the country, they must also pay the traveling costs of shipping.

## II. Implementation

To start, my design creates the database “Competitor” and also a schema with my initials. The use of a schema for company of this size is to limit users to the database objects they are in control of but also for performance reasons. If all database objects were within the default schema, sifting and sorting through thousands of tables would take up unnecessary time.



```
create_competitor_...sql - not connected
/*
John D'Amaro
12/1/19
CSE 581 - Project 2
*/

USE master
GO

/***** Object: Database Competitor *****/
IF DB_ID('Competitor') IS NOT NULL
    DROP DATABASE Competitor
GO

CREATE DATABASE Competitor
GO

USE Competitor
GO

/***** Object: Schema JD *****/
IF SCHEMA_ID('JD') IS NOT NULL
    DROP SCHEMA JD
GO

CREATE SCHEMA JD
GO
```

Below is the most complex table throughout the Competitor database named “Orders”. Each order has a customer, a status, date ordered, total amount, shipping service, shipping address, shipping amount, expected shipping date, and actual shipping data and a billing address. This table allows for a customer to have many orders, these orders can be shipped to an address that is not necessarily the same as a billing address and tracks the cost of shipping that related from another table.

Notable points from this design are the foreign key constraints, data types of all columns and the check constraints on the columns that contain money data types. Thankfully, these foreign key references make future transactions less complicated since all tables are not related to each other. With this, one can find out which a customer buys frequently which could make them a primary marketing target. Depending on how elaborate the database is, a company could monitor the feedback that is retrieved from reviews on products. If the feedback comes back as positive and a specific geographic area buys a large quantity is buying a specific product, it would be privy for that company to stock that product in a warehouse that’s closest to that warehouse location.

Switching to database efficiency, the data types of each column hold specifically as much data as it needs to hold. For example, when coding string-holding columns, the number of bytes per column are optimized to only allow for the exact number of characters that column

needs to hold. As a visual, if you only need to hold a cup of water, there's no reason to use a bucket to hold that cup of water. Also notice that both Ship Date columns accept NULL values because the database is not going to know these values until a transaction is committed and processed.

Finally, the check constraints allow prevent potential integrity issues and drive business logic. For example, unless there was a special deal on a product that made it free, there shouldn't be a total amount that is free. The same goes for the shipping amount, explained earlier, the minimum price of a shipment is \$5. These types of constraints protect the integrity of the database in case a user is sloppy and inputs the wrong information.

```

/***** Object: Table Orders *****/
CREATE TABLE JD.Orders (
    OrderID          INT          PRIMARY KEY IDENTITY,
    CustomerID       INT          REFERENCES JD.Customers (CustomerID),
    StatusID         INT          REFERENCES JD.OrderStatus (StatusID),
    OrderDate        SMALLDATETIME NOT NULL,
    TotalAmount      MONEY        NOT NULL CHECK( Total > 0),
    ShipViaID        INT          REFERENCES JD.ShippingService (ShipViaID),
    ShipAddressID    INT          NOT NULL,
    ShipAmount       MONEY        NOT NULL CHECK( ShipAmount > 0),
    ExpectShipDate   SMALLDATETIME DEFAULT NULL,
    ShipDate         SMALLDATETIME DEFAULT NULL,
    BillAddressID    INT          NOT NULL
)
GO

```

The following code shows the data that was inserted within the Orders table. The following statements describe how to read the data:

"The first order in the database was ordered by Eric Pubins (CustomerID = 4). He was displeased with his order, so he returned it (StatusID = 2). Eric placed his order on February 2<sup>nd</sup>, 2019 that totaled to \$155. He chose that his package was shipped by USPS (ShipViaID = 1) to his home address (ShipAddress = 5). The shipping total came out to \$23 and was expected to ship on February 10<sup>th</sup>, 2019. That package actually shipped on the 11<sup>th</sup> of February and billed to his apartment address (BillAddressID = 5)."

With the help of relational databases, data redundancy is reduced to a minimum and specific values are stored in separate tables and called upon when needed.

```

SET IDENTITY_INSERT JD.Orders ON;

INSERT INTO JD.Orders (OrderID, CustomerID, StatusID, OrderDate, TotalAmount, ShipViaID,
(1, 4, 2, '2019-2-7', 155.00, 1, 5, 23.00, '2019-2-10', '2019-2-11', 5),
(2, 7, 1, '2019-3-10', 1400.00, 2, 9, 15.00, '2019-3-17', NULL, 8),
(3, 13, 4, '2019-5-16', 310.00, 3, 17, 35.00, '2019-5-16', '2019-5-19', 15),
(4, 9, 4, '2019-5-18', 200.00, 1, 11, 10.00, '2019-5-21', '2019-5-21', 10),
(5, 1, 3, '2019-5-22', 60.00, 2, 1, 7.00, '2019-5-24', '2019-5-24', 1),
(6, 15, 3, '2019-6-20', 950.00, 3, 20, 38.00, '2019-6-27', '2019-7-1', 17),
(7, 1, 1, '2019-8-27', 2150.00, 1, 1, 23.00, '2019-9-4', NULL, 1),
(8, 8, 4, '2019-9-25', 90.00, 2, 10, 15.00, '2019-10-2', '2019-10-5', 9),
(9, 12, 1, '2019-12-11', 160.00, 3, 16, 43.00, '2019-12-18', NULL, 14),
(10, 2, 1, '2019-12-19', 1460.00, 1, 3, 33.00, '2019-12-26', NULL, 12)

SET IDENTITY_INSERT JD.Orders OFF;

```

### III. Testing

Ultimate test for a database is seeing what smart business decision you can derive from the data you are given. The following report are some snapshots that I think are important for a company to understand and take advantage of.

```

/***** Business Reports *****/
IF OBJECT_ID('JD.TotalExpenditureReport') IS NOT NULL
    DROP VIEW JD.TotalExpenditureReport;
GO

CREATE VIEW JD.TotalExpenditureReport AS
SELECT JD.Customers.CustomerID,
       SUM(TotalAmount + ShipAmount) TotalExpenditure,
       SUM(Quantity) AS NumberOfProducts
FROM JD.Customers
JOIN JD.Orders ON JD.Customers.CustomerID = JD.Orders.CustomerID
JOIN JD.OrderItems ON JD.Orders.OrderID = JD.OrderItems.OrderID
GROUP BY (JD.Customers.CustomerID)

GO

SELECT * FROM JD.TotalExpenditureReport
GO

```

	CustomerID	TotalExpenditure	NumberOfProducts
1	1	4413.00	5
2	2	4479.00	5
3	4	356.00	3
4	7	1415.00	1
5	8	315.00	7
6	9	210.00	1
7	12	406.00	2
8	13	1035.00	5
9	15	1976.00	2

This report allows the supplier to understand how much money a customer has spent of products over the last year. It also counts the number of products which is valuable information towards marketing. Although a customer bought \$210 worth of products, that expenditure was only on one product, so it can be concluded that they infrequently go shopping for large ticketed items. However, customers 1 and 2 spent the most money this year by a vast difference than all the others. Companies can take other information about what type of products they buy, how frequently they buy them and how they buy them.

```

IF OBJECT_ID('JD.InventoryReport') IS NOT NULL
    DROP VIEW JD.InventoryReport;
GO

CREATE VIEW JD.InventoryReport AS
SELECT JD.Storage.ProductName, Name AS CompanyName, ProductInStock,
       JD.Warehouse.City + ', ' + JD.Warehouse.State + ' ' + JD.Warehouse.ZipCode AS WarehouseLocation
FROM JD.Suppliers
JOIN JD.Products ON JD.Suppliers.SupplierID = JD.Products.SupplierID
JOIN JD.Storage ON JD.Products.ProductID = JD.Storage.ProductID
JOIN JD.Warehouse ON JD.Storage.WarehouseID = JD.Warehouse.WarehouseID

GO

SELECT * FROM JD.InventoryReport
GO

```

	ProductName	CompanyName	ProductInStock	WarehouseLocation
1	iPhone X	Apple	4	Glastonbury, CT 07003
2	Mac	Apple	2	Glastonbury, CT 07003
3	windows	Microsoft	2	Glastonbury, CT 07003
4	Wide Mouth 32 oz	Hydro Flask	3	Canonsburg, PA 15317
5	20 oz Thermos	Hydro Flask	1	Canonsburg, PA 15317
6	Mini-Logo	Stussy	3	Mc Lean, VA 22101
7	Surfer Special	Stussy	5	Mc Lean, VA 22101
8	Box Logo	Supreme	1	Mc Lean, VA 22101
9	Skatewear	Supreme	8	Mc Lean, VA 22101
10	Coach Tote	Coach	5	Auburndale, FL 33823

This report shows which product is owned by what company and, how many products they currently hold in stock, and where the warehouse is located. With this, a warehouse can

notify a specific supplier that their stock is running low on a specific product and needs to be refilled for upcoming orders.

```

IF OBJECT_ID('JD.ProductReviews') IS NOT NULL
    DROP VIEW JD.ProductReviews;
GO

CREATE VIEW JD.ProductReviews AS
SELECT JD.Suppliers.Name, ProductCode, Score, ISNULL(Text, 'Review not added') AS Description
FROM JD.Suppliers
JOIN JD.Products ON JD.Suppliers.SupplierID = JD.Products.SupplierID
JOIN JD.Reviews ON JD.Products.ProductID = JD.Reviews.ProductID

GO

SELECT * FROM JD.ProductReviews
GO

```

	Name	ProductCode	Score	Description
1	Stussy	01274	5	Best T Shirt I have ever owned!
2	Coach	49205	3	Nice, but nothing special
3	Coach	49385	4	Review not added
4	Bic	32343	5	So great I got 4!
5	JanSport	86302	5	Light weight and Durable
6	Microsoft	66392	5	Best computer Ive ever owned

This report shows which customer reviewed what product and when that order was recorded. As I mentioned above, a customer can review a product multiple times but only if it was ordered at a different time. This information can be helpful to gather intel on how the public is reacting to a specific product/which demographic buys the most/any further modifications needed to improve its success.

```

IF OBJECT_ID('JD.OrdersPerRegion') IS NOT NULL
    DROP VIEW JD.OrdersPerRegion;
GO

CREATE VIEW JD.OrdersPerRegion AS
SELECT LEFT(ZipCode,1) + 'XXXX' AS ZIPCodeRegion, SUM(Quantity) QuantityOfProducts
FROM JD.OrderItems
JOIN JD.Orders ON JD.OrderItems.OrderID = JD.Orders.OrderID
JOIN JD.Customers ON JD.Orders.CustomerID = JD.Customers.CustomerID
JOIN JD.CustomerShippingAddress ON JD.Customers.CustomerID = JD.CustomerShippingAddress.CustomerID
GROUP BY LEFT(ZipCode,1) + 'XXXX'

GO

SELECT * FROM JD.OrdersPerRegion
GO

```

	ZIPCodeRegion	QuantityOfProducts
1	0XXXX	16
2	1XXXX	5
3	2XXXX	10
4	3XXXX	5
5	4XXXX	3
6	5XXXX	2

This report shows the number of products that have been bought in each specific geographical region. What a company can do with this report is see that ZIP Code regions start with 4 and 5 aren't buying a large quantity of items. However, regions starting with 0 and 2 are. Suppliers can shift their product to warehouses located in those regions. By doing this, those warehouses will be stocked up to handle large order demands and since the products will be within the same region, shipping costs will decrease for customers which should entice them to buy more product.

With these reports, I believe that companies will be able to strategize their goods to make shipments quicker, warehouse stock optimized while keeping customers happy. Drawing attention away from the business decisions, procedures and transactions are utilized so that these orders are automatic and quick to process. Below is a procedure that is ran whenever a new customer signs up to begin an order.

```

/***** Performance & Efficiency *****/
IF OBJECT_ID('JD.spNewCustomer') IS NOT NULL
    DROP PROC JD.spNewCustomer;
GO


CREATE PROC JD.spNewCustomer
    @FirstName VARCHAR(60) = NULL, @LastName VARCHAR(60) = NULL,
    @UserName VARCHAR(60) = NULL, @EmailAddress VARCHAR(255) = NULL
AS
    IF @FirstName IS NULL
        THROW 50001, 'Must include First Name', 1;
    IF @LastName IS NULL
        THROW 50001, 'Must include Last Name', 1;
    IF @UserName IS NULL
        THROW 50001, 'Must include User Name', 1;
    IF @EmailAddress IS NULL
        THROW 50001, 'Must include Email Address', 1;

    INSERT JD.Customers
    VALUES (@FirstName, @LastName, @UserName, @EmailAddress);
    RETURN @@IDENTITY;

BEGIN TRY
    DECLARE @CustomerID INT
    EXEC @CustomerID = JD.spNewCustomer
        @FirstName = 'Libby',
        @LastName = 'Kretzing',
        @UserName = 'LKretzing',
        @EmailAddress = 'LKretzing@gmail.com';
    PRINT 'New Customer Added';
    PRINT 'New Customer: ' + @FirstName + ' ' + @LastName;
END TRY
BEGIN CATCH
    PRINT 'New Customer information entered incorrectly';
END CATCH
GO

```

47 %

 Messages

Commands completed successfully.

Completion time: 2019-12-07 17:25:06.6889363-05:00

The mentioned procedure, spNewCustomer, takes four inputs which are general characteristics of an account. Before the customer's information is accepted, a couple of constraints must be followed. In this example, problems arise if the customer forgets to input any of their information. If everything is entered correctly, a message appears "New Customer Added. New Customer: Libby Kretzing". This solidifies to the customer that they have been added but further information will be needed to begin an order such as billing and shipping address.

```

IF OBJECT_ID('JD.spNewReview') IS NOT NULL
    DROP PROC JD.spNewReview;
GO

CREATE PROC JD.spNewReview
    @CustomerID INT, @ProductID INT, @OrderID INT,
    @ReviewDate SMALLDATETIME, @Score INT, @Text TEXT
AS
IF EXISTS(SELECT * FROM JD.Customers WHERE CustomerID = @CustomerID)
AND EXISTS(SELECT * FROM JD.Products WHERE ProductID = @ProductID)
AND EXISTS(SELECT * FROM JD.Orders WHERE OrderID = @OrderID)
    INSERT JD.Reviews
    VALUES(@CustomerID, @ProductID, @OrderID, @ReviewDate, @Score, @Text)
ELSE
    THROW 50001, 'Incorrect review information', 1;

BEGIN TRY
    EXEC spNewReview
        1, 17, 7, '2019-12-7', 5, 'Best computer I've ever owned';
END TRY
BEGIN CATCH
    IF ERROR_MESSAGE() > 50001
        PRINT CONVERT(varchar, ERROR_MESSAGE());
END CATCH
GO

SELECT * FROM JD.Reviews

```

	ReviewID	CustomerID	ProductID	OrderID	ReviewDate	Score	Text
1	1	13	12	3	2019-05-24 00:00:00	5	Best T Shirt I have ever owned!
2	2	13	6	3	2019-05-23 00:00:00	3	Nice, but nothing special
3	3	9	5	4	2019-05-22 00:00:00	4	NULL
4	4	8	7	8	2019-10-01 00:00:00	5	So great I got 4!
5	5	8	10	8	2019-10-02 00:00:00	5	Light weight and Durable
6	6	1	17	7	2019-12-07 00:00:00	5	Best computer I've ever owned

The next procedure allows a customer to input a review on a product. Unlike the last procedure that only checked whether the all inputs were NULL or not, this procedure makes sure that the product that is being bought was bought and received by a customer. This is accomplished by checking if the ID's that are input from the front end are stored in the back end. For example, this allows Ethan to review his iPhone that he bought on January 3<sup>rd</sup>. Since it was his order, and that was the product he bought with that order, he can review that item. If Ethan also bought the same item but years later, he can write another report. This constraint isn't met if Jane is trying to review the iPhone that Ethan bought. An error message would be returned if that were to happen or if Ethan were to review an item that wasn't in that specific order.

With these procedures, the data that the database is receiving is incredibly reliable and allows for the DBA to not worry if an incorrect value is input, since it will be returned. On the contrary, these procedures only let customers input information that they know, how are "unknown" variables calculated such as shipping cost or the expected shipping date? These handy blocks of code are known as transactions, which make automation faster and improve performance. Below is a transaction that starts a new order after a customer has placed it.



```

/***** New Order *****/
DECLARE @OrderID int
DECLARE @OrderItemID int

BEGIN TRY
    BEGIN TRAN;
    SET @OrderID = @@IDENTITY
    INSERT JD.Orders(OrderID, CustomerID, StatusID, OrderDate, TotalAmount,
        ShipViaID, ShipAddressID, ShipAmount, ExpectShipDate, ShipDate, BillAddressID)
        VALUES(@OrderID, 7, 1, '2019-12-7', 35.00, 1, 9, 10.00, '2019-12-18', NULL, 8);
    SET @OrderItemID = @@IDENTITY
    INSERT JD.OrderItems
        VALUES(@OrderItemID, @OrderID, 12, 35.00, 1);
    COMMIT TRAN;
END TRY
BEGIN CATCH
    ROLLBACK TRAN;
END CATCH

```

%

Messages

Msg 8101, Level 16, State 1, Line 155  
An explicit value for the identity column in table 'JD.OrderItems' can only be specified when a column list is used and IDENTITY\_INSERT is ON.

Completion time: 2019-12-07T17:26:58.4809375-05:00

Unfortunately, I believe the way I structured my database might be affected the performance of the transaction that I created but I can explain how its supposed to work. When a customer buys a product, that instantaneous creates a new order. A new order is comprised of many attributes such as the date it was ordered, where its being shipped to, how much it'll cost as well as how much shipping will be. On top of that, the order may consist of multiple products which I note as items. An order item has the same order ID as the base order since they are related.

```

/***** Shipping Cost *****/
DECLARE @ShipZip int
DECLARE @WarehouseZip int
DECLARE @DistanceID int
DECLARE @DaysToShip int
DECLARE @CostOfDistance money
DECLARE @CostOfService money

BEGIN TRY
    BEGIN TRAN
    SET @ShipZip = (SELECT ZIPCode FROM JD.CustomerShippingAddress WHERE CustomerID = 1)
    SET @WarehouseZip = (SELECT ZIPCode FROM JD.Warehouse WHERE WarehouseID = 5)
    IF ABS(CONVERT(int, (LEFT(@ShipZip, 1)) - CONVERT(int, LEFT(@WarehouseZip, 1)))) = 0
        SET @DistanceID = 1;
    ELSE
        IF ABS(CONVERT(int, (LEFT(@ShipZip, 1)) - CONVERT(int, LEFT(@WarehouseZip, 1)))) = 1
            SET @DistanceID = 2;
        ELSE
            IF ABS(CONVERT(int, (LEFT(@ShipZip, 1)) - CONVERT(int, LEFT(@WarehouseZip, 1)))) = 2
                SET @DistanceID = 3;
            ELSE
                IF ABS(CONVERT(int, (LEFT(@ShipZip, 1)) - CONVERT(int, LEFT(@WarehouseZip, 1)))) >= 3
                    SET @DistanceID = 4;

    SET @DaysToShip =
        (SELECT DaysToShip
         FROM JD.Distance
         WHERE DistanceID = @DistanceID)

    SET @CostOfDistance =
        (SELECT CostOfDistance
         FROM JD.Distance
         WHERE DistanceID = @DistanceID)

    SET @CostOfService =
        (SELECT CostOfService
         FROM JD.ShippingService
         WHERE ShipViaID = 2)

    INSERT JD.ShippingFare
        VALUES(2, @DistanceID, @ShipZip, @WarehouseZip, @CostOfDistance+@CostOfService)
    COMMIT TRAN
END TRY
BEGIN CATCH
    ROLLBACK TRAN
END CATCH

```

9 %

Messages

(0 rows affected)

Completion time: 2019-12-07T17:28:37.0509480-05:00



A more impressive transaction includes how shipping costs are calculated. The transaction accounts for three input variables: the shipping ZIP code, the warehouse zip code and the shipping service. With these three variables, the total cost of shipping is calculated. To determine the cost of distance, I split the US into each respective ZIP code region and monetized the distances. The further away a product is, the most expensive it is to ship. The difference of the first digits in the ZIP codes show how far on address is from another which correlates to a cost and expected shipping time. An additional shipping cost is found when the customer chooses which shipping service, they want to deliver their package. The following information was inserted into the Shipping Fare table to show that the new order should take X amount of days to ship and will cost X amount of money.

```

CREATE TRIGGER JD.CustomerShippingAddress_INSERT_UPDATE
ON JD.CustomerShippingAddress
AFTER INSERT, UPDATE
AS
    UPDATE JD.CustomerShippingAddress
    SET State = UPPER(State)
    WHERE ShipAddressID IN (SELECT ShipAddressID FROM Inserted);
GO

IF OBJECT_ID ('JD.Orders_INSERT') IS NOT NULL
    DROP TRIGGER JD.Orders_INSERT
GO

CREATE TRIGGER JD.Orders_INSERT
ON JD.Orders
AFTER INSERT
AS
    BEGIN
        IF EXISTS
            (SELECT *
             FROM JD.Orders JOIN
                (SELECT JD.OrderItems.OrderID, SUM(Quantity*ItemPrice) AS SumOfOrder
                 FROM JD.OrderItems
                 GROUP BY JD.OrderItems.OrderID) AS OrderTotal
             ON JD.Orders.OrderID = OrderTotal.OrderID
            WHERE (Orders.TotalAmount <> OrderTotal.OrderID))
        BEGIN
            THROW 50001, 'Order Totals differ in OrderItems and Orders table', 1;
            ROLLBACK TRAN;
        END;
    END;

/***** Security Section *****/
CREATE LOGIN JohnDamaro WITH PASSWORD = 'Cuse#2020',
    DEFAULT_DATABASE = Competitor,
    CHECK_EXPIRATION = ON;

```

Additional constraints and security measures must be made to keep a database under control without integrity breaches. Triggers were included in the data creation code to capitalize all states that are inserted or updated into any of the tables that contain state attributes. An integrity trigger was coded to maintain consistency with all insert statement in the Orders and Order Items tables. The quantity of the items multiplied by their individual costs must be equal to the total amount that's calculated in the orders table. Lastly, logins and users were created to make sure that specific people only have specific permissions to perform specific tasks. When I tried to grant roles for the users, it restricted myself from making changes or data retrieval, so I did not include them in my design.

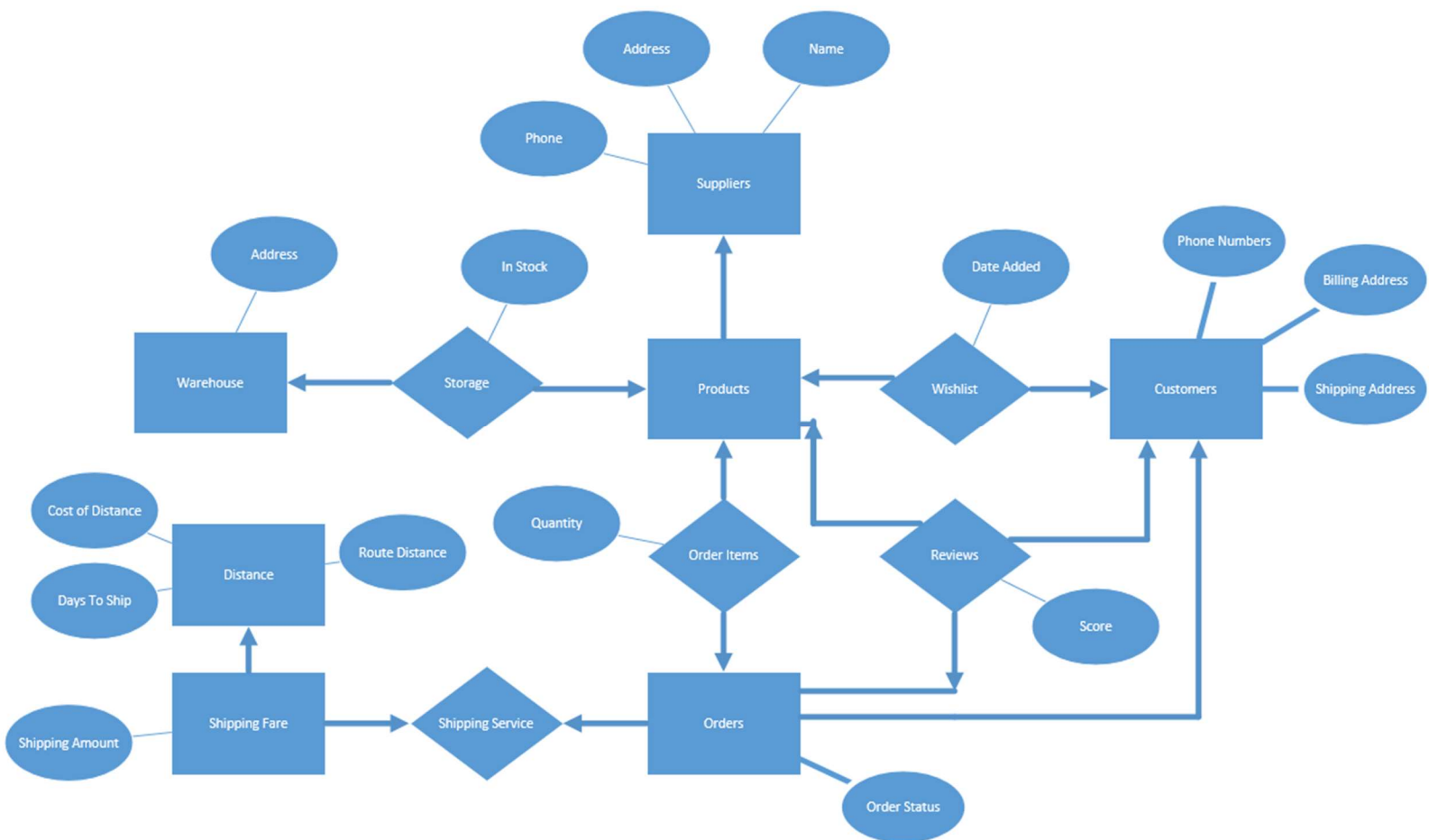
## IV. Conclusions

As a conclusion, relational databases are extremely resourceful when kept at a high standard, data constraints are placed, and performance is optimized. With SQL, analysts can retrieve extremely useful information that can pilot important business decisions that can give a company a market advantage. The data base objects I coded above will make for better processing performance but also need to be optimized. I feel like if I had more time to really sit down and work through each transaction , I could have created a transaction that takes in minimal amount of inputs and would've returned every little piece of information down to the minute that it leaves the door to the minute it touches a door step.

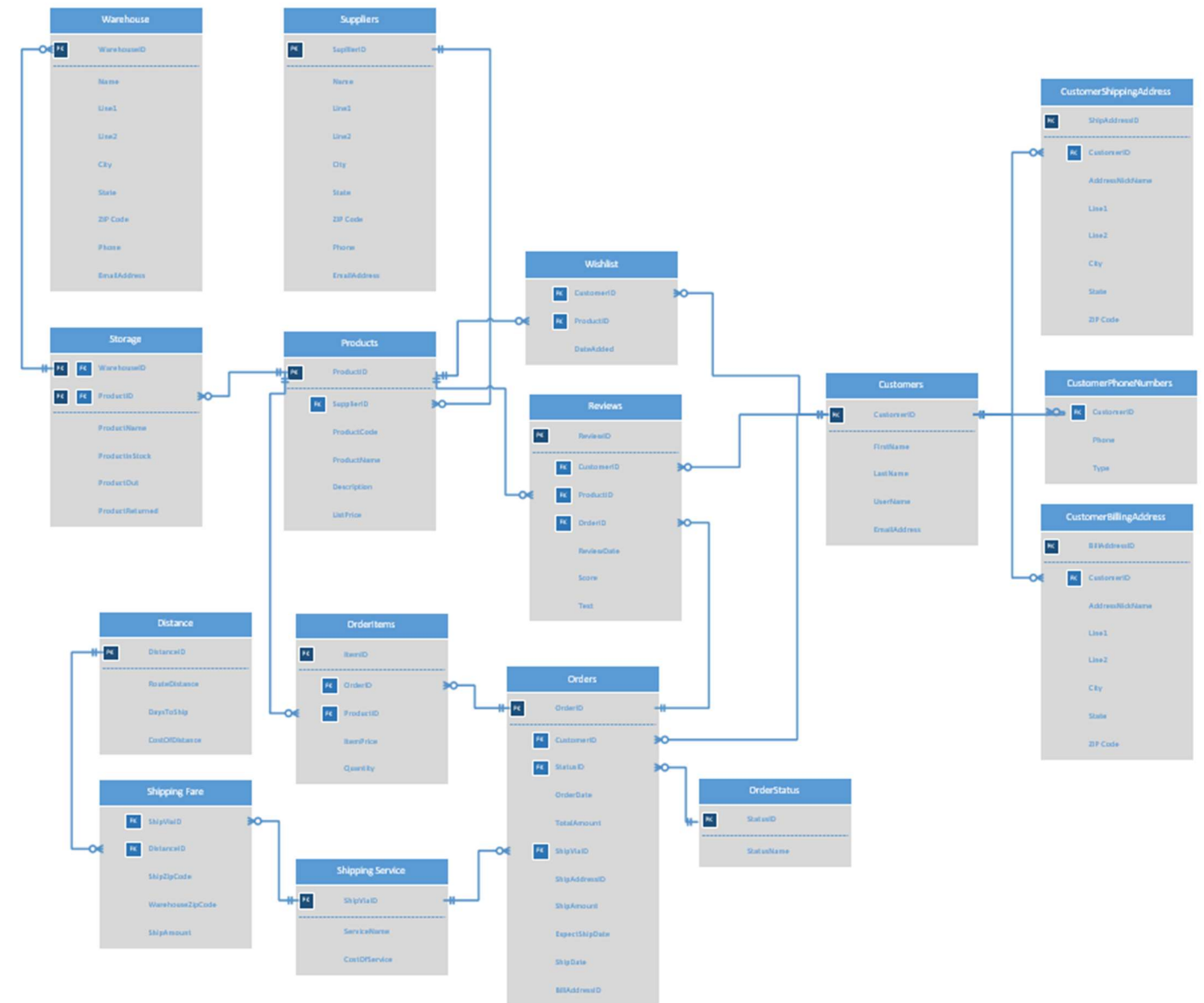
I believe that this project allowed me to build a relatively efficient database with enough data to test but not enough to make exquisite business decisions off it. I understand having to create our own data because if data was given to us, there wouldn't be a reason to design a database. However, because of the independent data creation, I feel as if my database is not to it's full potential since I spent multiple hours creating data and not on creating more complex procedures and transaction. All in all, I'm extremely appreciative of this project, this course, and my professor.

## V. Appendix

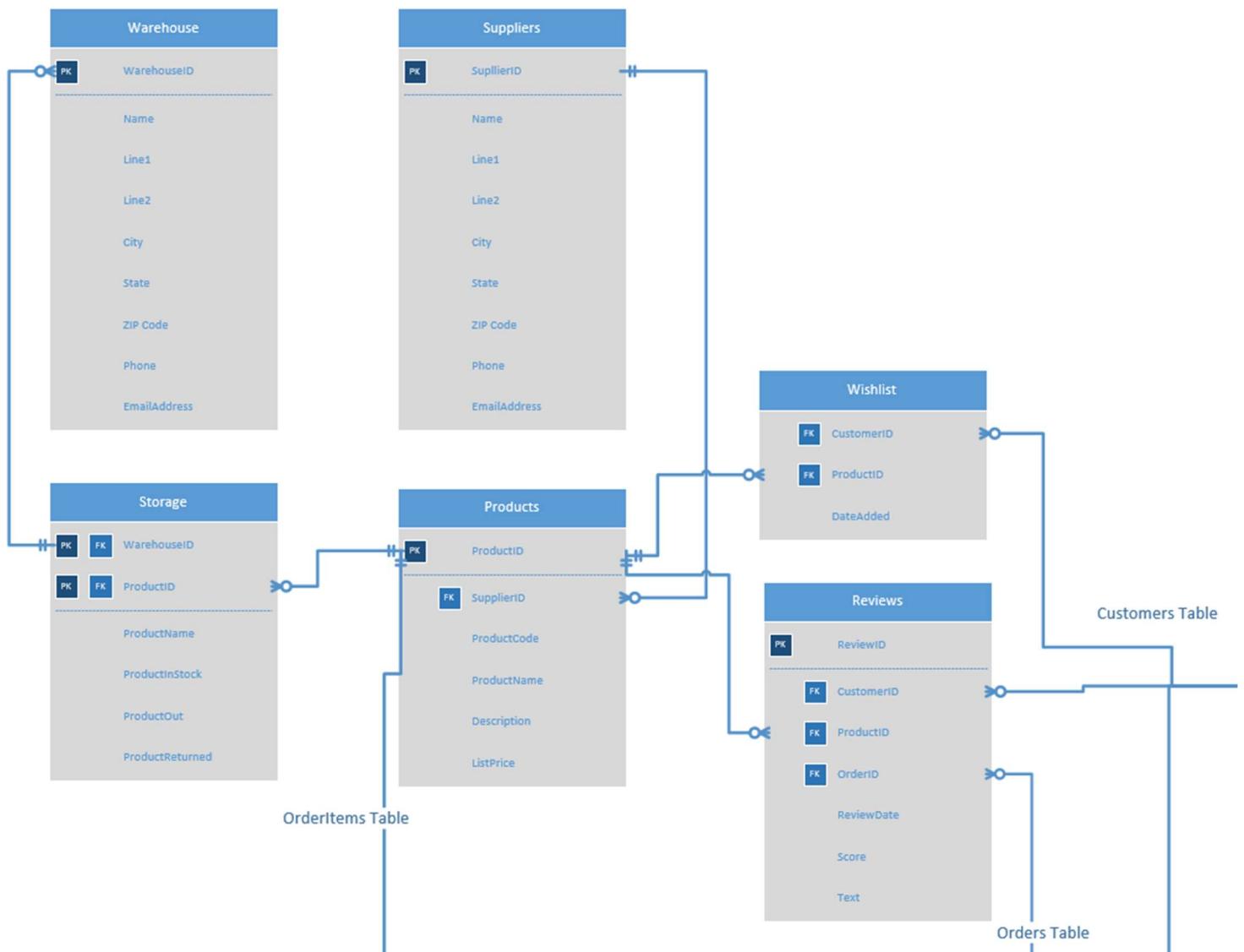
## A. Conceptual E/R Diagram



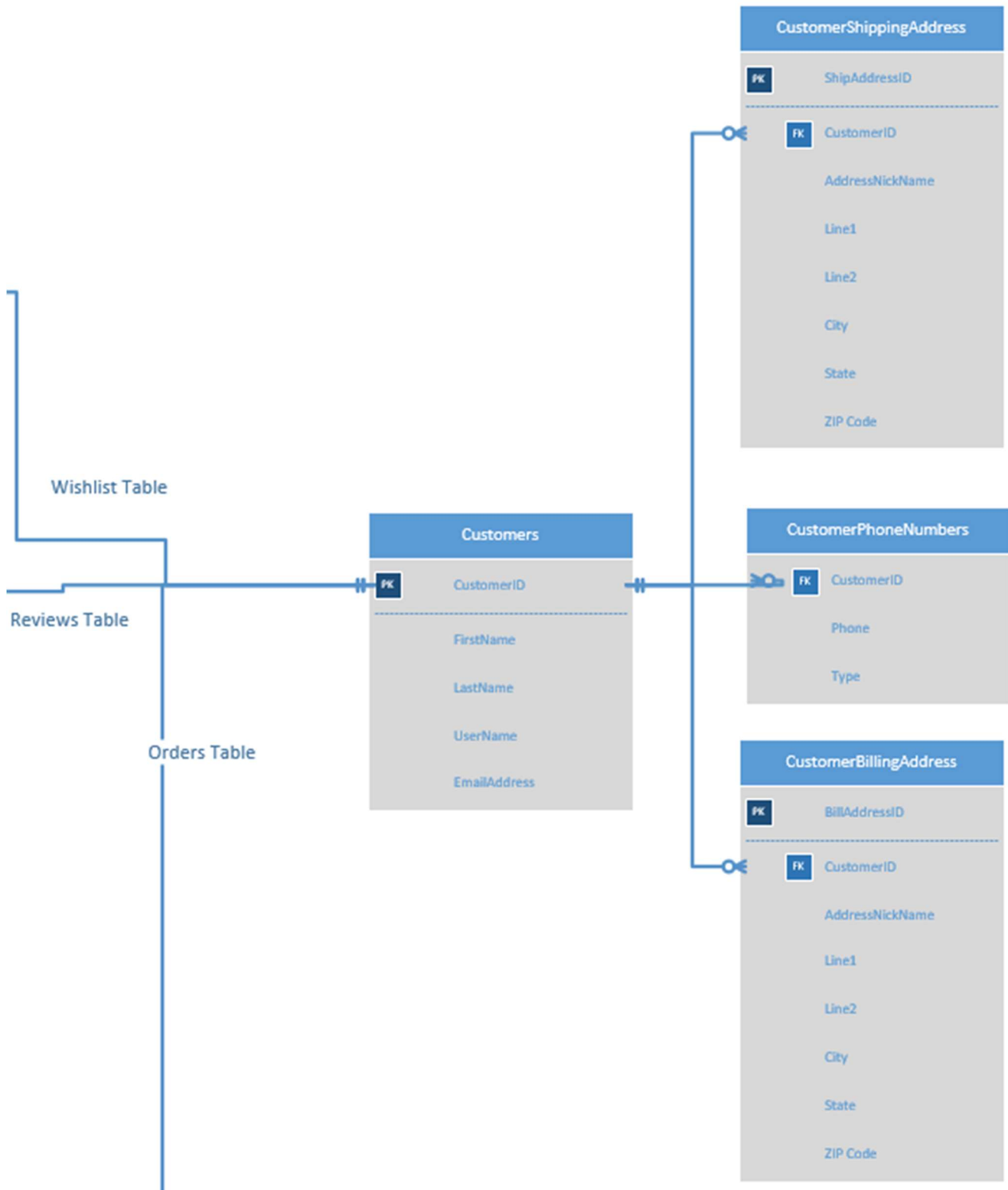
## B. Entire Entity E/R Diagram



## C. Product Portion of E/R Diagram



## D. Customers Portion of E/R Diagram



## A. Orders Portion of E/R Diagram

