Name: Mahim Choudhury

**Table of Contents**

[Project Direction Overview 2](#_Toc115444560)

[Use Cases and Fields 2](#_Toc115444561)

[Structural Database Rules 5](#_Toc115444562)

[Conceptual Entity-Relationship Diagram 7](#_Toc115444563)

[Full DBMS Physical ERD 9](#_Toc115444564)

[Summary and Reflection 14](#_Toc115444565)

# Project Direction Overview

Update, if necessary, the overview that describes who the database will be for, what kind of data it will contain, how you envision it will be used, and most importantly, why you are interested in it.

I plan to design a database to keep track of the product inventory for my own business that I plan to open soon. The products will be related to printing services where customers can purchase printing materials i.e. printers, paper, ink cartridges, office supplies, shipping accessories, etc. They can also order printing of any documents or pictures etc. I think a well-maintained order management system is key run my business successfully. The database will be useful for the manager or the owner of the store to keep track of the inventory. It should be able to record that product information for inventory purposes, customer information for sales, and promos. A well-run inventory management system should help the manager to prevent any product shortages while not over-ordering products. It ensures good customer service as the customers will always find whatever they need every time they visit the store. Also, if a store has multiple branches, good inventory management should help the manager to order products and fill the inventory based on locations and demand. The manager can identify the most sold items or find out regular customers for future promos or sales predictions.

The system will consist of an app/website that the manager or the owner can use to manage the inventory of the company/store. The app/website should only allow authorized personnel(manager/owner) to log in and perform any inventory-related operation. The user of the app can store product information, track the number of items left in the inventory, record how many items are needed to fill the inventory, and order them. Additionally, the system can be used for the following purposes such as forecast future sales, track shipments, record transaction information, keep track of the returns by any customers, use customer addresses for promo and analysis purposes, etc. Overall, the database can be identified as a business order management system.

# Use Cases and Fields

Update, if necessary, the five or more use cases that enumerate steps of how the database will be typically used, and also identify significant database fields needed to support each use case.

1. The manager/owner signs up for the application and installs the application in the system

|  |  |
| --- | --- |
| *USER* | *SYSTEM* |
| *Visits the app or website* | *Displays sign up/log in page* |
| *Enters valid user account information* | *Checks the information and creates the account in the database or log in the user* |

Here, the database needs to securely store some crucial information about the user who is signing up or logging in.

|  |  |  |
| --- | --- | --- |
| Field Name | What is it? | What’s it used for? |
| Employee \_ID | A unique id is given to the user when they joined the company | To identify different employees or managers to avoid same-name confusion |
| First\_name | User’s legal first name | Used to display the user’s name on the screen |
| Last\_name | User’s legal last name | Used to display the user’s name on the screen |
| Store\_ID | A unique id to distinguish stores | Used to identify which store the operation should take place |
| Store\_location | The location of the store where the user will perform the tasks to perform business management. | This is important if the stores have multiple branches. This should avoid inventory conflicts between different branches of the business. |

1. Another usage would be to keep track of specific products that were sold every day in a specific location to help a manager understand the demand and maintain inventory

|  |  |
| --- | --- |
| *USER* | *SYSTEM* |
| *Enters the product names that were sold in one business day* | *Records the names of the products for that day* |
| *Enters the quantity of that product that was sold* | *Records the number of that specific product that was sold that day* |
| *Enters the date and transaction information* | *Keeps that date in the record as a business day and also the transaction id* |
| *Enters the location information* | *Makes sure the transaction is only for that location* |

Fields to be used in the database,

|  |  |  |
| --- | --- | --- |
| Field Name | What is it? | What’s it used for? |
| Product\_ID | A unique id for each batch of products that are coming into the warehouse. | This distinguishes different brands of products and categories of products |
| Product\_Name | The name of the product | Helps to identify what product it is |
| Product\_quantity | The number of products that were sold. | To keep track of the stock |
| Sales\_date | When the product was sold. | Helps to maintain inventory and organize restocking. |
| Transaction\_ID | A unique id was generated each time the item was sold | Helps to record the unique transaction id for future returns for example. |
| Store\_location | The name of the city whose store will be receiving the products. | This is important if the stores have multiple branches. This should make sure the inwards are going into the correct inventory location. |
| Store\_ID | A unique id to distinguish stores | Differentiate stores and avoid anomalies |

1. Next usage could be to store the inwards of products for a particular location’s warehouse. It should also record the distributor information. It would be useful for the user to understand the flow of product stock in a store and know which distributor the products are coming from.

|  |  |
| --- | --- |
| *USER* | *SYSTEM* |
| *Enters the product name and id* | *Displays the product itself* |
| *Enters the arrival date* | *Records the expected/actual date when the products should be in stock again.* |
| *Enters distributor Name and id* | *Records the distributor of the product and their information* |
| *Enters the location information* | *Makes sure the location information is valid and the inwards of products are only for that location’s inventory* |

Here are the fields needed in the database,

|  |  |  |
| --- | --- | --- |
| Field Name | What is it? | What’s it used for? |
| Product\_ID | A unique id for each batch of products that are coming into the warehouse. | This distinguishes different brands of products and categories of products |
| Product\_Name | The name of the product | Helps to identify what product it is |
| Distributor\_id | A unique id for each distributor | This distinguishes different distributors who will be supplying the products. |
| Distributor\_name | The name of the product distributor | Helps to identify what distributor distributed the product |
| Product\_expected arrival date | When the product is expected to arrive in the warehouse. | Users will know when the inventory would get restocked and can plan accordingly. |
| Product\_arrival date | The actual arrival dates. | This will record the actual time of the arrival and help the user predict future dates of product availability and so on. |
| Store\_location | The name of the city whose store will be receiving the products. | This is important if the stores have multiple branches. This should make sure the inwards are going into the correct inventory location. |

1. Another key usage of this database would be to keep track of the customer’s information for future promotions and restocking prediction. The address information is needed to send promo placards, verify customer identity in case of returns, etc.

|  |  |
| --- | --- |
| *USER* | *SYSTEM* |
| *Enters the customer’s name* | *Records the customer information* |
| *Enters the purchase date and products bought* | *Records the purchased date and the name of the product* |
| *Enters customer address if the purchase is online* | *Records the address of the customer for shipping purposes* |

Here are the fields for the database for this use case,

|  |  |  |
| --- | --- | --- |
| Field Name | What is it? | What’s it used for? |
| First\_name | Customer’s legal first name | Used to display the user’s name on the screen |
| Last\_name | Customer’s legal last name | Used to display the user’s name on the screen |
| Date of purchase | When they bought the product | Track the date of purchases to predict future purchases and maintain inventory to keep the customer returning. |
| Product\_name | What was purchased | For future stocks and promos. |
| Customer street address | Street address of the customer where they live | Shipping and promo purposes. |
| Customer city | City from which the customer from | Promos and future analysis purposes |

1. The next usage of this database would be to keep track of the customer returns (in-person or online returns) to maintain the stocking of specific products.

|  |  |
| --- | --- |
| *USER* | *SYSTEM* |
| *Enters the product ID that was returned and the quantity* | *Records the product and its quantity that came back to stock.* |
| *Enters the return reason and return date* | *Records the customer complaints and product restock date* |
| *Enters the customer information* | *Records the address, name, and phone number of the customer who is returning the product* |

Here are the fields to be used in the database for this use case,

|  |  |  |
| --- | --- | --- |
| Field Name | What is it? | What’s it used for? |
| First\_name | Customer’s legal first name | Used to display the user’s name on the screen |
| Last\_name | Customer’s legal last name | Used to display the user’s name on the screen |
| In\_person\_return\_date | When they returned the product in person | Track the date of return to update the inventory stock for those products. |
| Product\_name | What was returned | For restocking |
| Customer address | Where was the product returned from | Shipment tracking and promo purposes. |
| Shipment\_tracking\_id | If the product was shipped instead of in-person return, use the tracking information | To track when the return will arrive so the user can work with the inventory accordingly. |

# Structural Database Rules

Update, if necessary, your list of your structural database rules, along with supporting explanations.

1. Each store location must be associated with one store address; one store address must be associated with one store location.

This rule has been added during normalization. A store must be only associated with one address and vice versa. One cannot have the same store location in multiple addresses and similarly, multiple addresses cannot contain a single store location.

1. Each store address must have a store state; each Store state may have many store addresses

This one is also added as part of BCNF normalization. This helps us to distinguish state and address relation since each state can contain the same address. A state may have many store locations but one store location can only be located at one state.

1. Each store location may sell many products; each product may be sold from one store location.

From the 2nd use case, I found two useful entities, Product, and location.

This structural rule is very powerful in terms of covering the 2nd use case for my database. It can cover all the valuable information needed to record store-product inventory count and turnover information. We will be using the city name and store id to identify the store location. Here, since each product has unique ids, they are in one store location only, and thus, it can only be sold from one store. However, one store can sell more than one of the same or various products.

From the 3rd use case, I found product and distributor entities.

1. Each distributor may distribute many products; each product must be distributed by one distributor.

Here, we can identify the distributor and product relation. A distributor can supply different products but also may not do it and so I made it optional. However, a specific product cannot be distributed by many distributors in my business. For example, ’clear tape’ products can only be distributed by one distributor. This product will not be purchased from a different distributor and thus I made it mandatory.

Another key entity would-be customers from use case 4. Also, I would like to use the address as another entity that will be related to customers. This will again be connected to the product entity.

1. Each customer may purchase many products; each product may be purchased by many customers.

This is self-explanatory. Customers may or may not purchase an item and the same product can be purchased by many customers. Here, I made the customer purchase of products optional as a customer may end up not buying any product. Similarly, each product may or may not be purchased by many customers, thus this also is optional.

1. Each Customer must be from one address; many customers may have many addresses.

Here I made the first portion of the rule mandatory as the database should avoid wrong address information related to the customer. A customer cannot have multiple addresses at a time in the system. On the second portion of the rule, I specified that many or different customers will have different addresses or many addresses associated with their information.

From use case 5, we can see two main data points that are useful. Return and customer information.

1. Each Customer may initiate one to many returns; Each return must be initiated by one customer.

One customer can initiate one or more returns whereas, each return cannot be initiated by different customers. Therefore, the second portion of the rule is mandatory since one customer should have access to initiating a return of the specific product that they bought.

1. Each return can be consisting of many products; one product must have consisted of one return.

Many products can be returned at once, but each product can only be returned once. This rule is essential to avoid any data anomalies in the database regarding return information.

Based on the use cases described above, I came up with the following specialization-generalization rules,

1. A purchase is face-to-face, online, or both.

From the first few use cases, I decided to involve product purchase to come into play. A purchase can be face to face or in person both or none of those. Therefore, the constraints here are overlapping and totally complete since there is no other possible outcome of the purchase.

1. A product is a shipping supply or office supply or printing supply or none of those.

I think adding a product-based specialization-generalization rule would be useful. Since my business will sell shipping, copying/printing, and office supplies, I think adding them as a product subtype is very fitting. The rule depicts the product could be either of those or none of those and therefore it is disjoint and partially complete. The product must be one of those at a time and since there are other options mentioned with “none of those”, it is partially complete.

1. A return is an online return, an in-person return, or both.

The last rule is derived from the 5th use case. This rule defines the return can be done face to face, online, or both in the case of someone completing one portion of the return order online and the other in person. Therefore, the rule has overlapping and totally complete constraints in it since the return can either be complete by those three options or it is not possible to complete.

# Conceptual Entity-Relationship Diagram

The conceptual diagram was updated after normalization,

Chart

Description automatically generated with low confidence

Starting on the left side, a store can be associated with only one location and vice versa. A location can be in a state, but a state may have multiple locations. A store location may sell many products, but each product may only be sold from one store location. An Account entity is added later to store the account information. One account can associate with many stores but one store must be associated with one or more accounts. In the middle, we have multiple relationships related to the product, distributor, customer, return, and address. Starting with product and distributor, one product must be distributed from one distributor, but each distributor may distribute many products. Again, each product may be purchased by many customers and each customer may purchase many products. On the bottom right corner, each customer must have one address, but many customers may have various addresses. Each customer may initiate many returns, but each return must be initiated by one customer. Here, we can also relate the product-return relationship by the following, each product must have consisted of one return and each return may consist of multiple products. As for specialization-generalization, I added a purchase entity and connected it with customers. It has a mandatory and overlapping constraint. Similarly, the return entity now has mandatory and overlapping constraints as visualized in the ERD diagram. Lastly, the product entity now has partially complete and disjoint constraints as part of an added specialization.

# Full DBMS Physical ERD

The associative entity relationships in my ERD diagrams are store\_location/products, product/customer, product/return, customer/return, customer/purchase, and customer/address.

The store location and product is a 1:M relationship. Each store may sell many products but a product may be only sold from one store location.

The product and customer relationship is a many-to-many relationship. One Customer may purchase many products and many products can be purchased by one customer. Therefore, we need a bridging entity.

The product and return entity relationship is a one-to-many 1:M. Each return can have many products, but each product must be associated with one return only.

The Customer and return entity relationship is a 1:M. Each customer can initiate many returns, but each return can only be initiated by one customer.

The Customer and Purchase entity relationship is a 1:M. Each customer can make many purchases, but each purchase can only be made by one customer.

The Customer and Address entity relationship is 1:1. Each customer can have one address, and each address can only be associated with one customer.

Full DBMS ERD with Attributes:

Here, I will be adding a few useful attributes to my database entities and adding them to the initial physical DBMS ERD diagram. Before I do that, I am providing a table with all the attribute information and reasonings behind choosing the attribute.

|  |  |  |  |
| --- | --- | --- | --- |
| Table | Attribute | DataType | Reasoning |
| Store location | Store\_city | Varchar(255) | Store city will identify where the store is located and can be used for various purposes. I am allowing 255 characters just in case the city name is long. |
| Store location | Store\_state | Varchar(255) | The name of the state where the store is located. 255 characters are allowed for it. Information may be useful to the manager when locating the closest distributor is important. |
| Store location | Store\_postal | Decimal(5) | Postal zip code of the store. 5 digits should suffice for this piece of information. |
| Store location | Store\_street | Varchar(255) | The street address where the store is located exactly. Very useful to the manager for order management purposes. 255 characters are allowed here. |
| Product | Product\_name | Varchar(255) | A product name will identify a product. 255 characters will suffice the length of the product name for me. |
| Product | Product\_description | Varchar(1024) | Products should have a description attached to them to explain to the customer what it is for. I allowed a substantial number of characters to fulfill this attribute. |
| Distributor | Distributor\_name | Varchar(255) | The name of the distributor is used to identify a product distributor. I think 255 characters is long enough to tackle this. |
| Customer Product | Quantity | Decimal(5) | This is used to store the number of products a customer is purchasing. I think 5 digits will be more than safe to store this information. |
| Customer | First\_name | Varchar(64) | First name of the customer. The names are useful for sending promos, verifying returns, etc. Varchar 64 should be enough for this. |
| Customer | Last\_name | Varchar(64) | Last name of the customer. The names are useful for sending promos, verifying returns, etc. Varchar 64 should be enough for this. |
| Customer | Phone\_number | Decimal(11) | A phone number can be useful for return, shipping, or any promo purposes. 11 digits will be enough for it. |
| Purchase | Total\_price | Decimal(7,2) | The total price of the products purchased by a customer. 7 digits should be enough for this attribute |
| Purchase | Purchase\_date | Date | When was the purchase made. |
| Purchase | isOnline | Char(1) | ‘T’ if the purchase was made online. ‘F’ if the purchase was made face to face. This is useful for the manager to determine which stores are selling what type of products online vs which stores people are likely to go to in person so that they can organize the store to it’s the best fit. One char is needed for this attribute. |
| Address | Street\_address | Varchar(255) | Street address of the customer which has the number and the street name. 255 characters should be enough. |
| Address | City\_name | Varchar(255) | City name of the Customer address. Used for analysis and various other purposes. 255 characters should be enough. |
| Return | Product\_name | Varchar(255) | A product name will identify which product is being returned. 255 characters will suffice the length of the product name for me. |
| Return | Return\_date | Date | When was the return made. |
| Return | IsOnline | Char(1) | To determine of the return was made online or face to face. One char is enough for this. |
| Return | Return\_reason | Varchar(1024) | A brief description of the return reason. 1024 characters should be enough to write a small sentence. This is useful to determine what is wrong with the product. |

While there may be some other attributes that can be useful, I think those mentioned above should cover most of the needs of my database.

Here is the updated physical DBMS diagram(Without normalization),

Diagram

Description automatically generated

Normalization:

I noticed there is one area where data redundancy may occur in my database and that is the address information in the store location. Every time a store sells many products, the address information will be repeated. Thus, I broke it down into “Store Address” and “State”. When a store sells a product, it can reference its address to the Stores’ Address entity. Likewise, the store address entity can reference the Store State entity for the State Name.

The store Address entity can be further normalized by breaking it down to cities, but for my database purpose it is not important and so I left it as is. Here’s the updated full physical DBMS ERD,

Diagram

Description automatically generated

Notice at the top left corner, I added two new entities named “Store Address” and “Store State”. They serve a very important rule in terms of normalizing the tables. Store address information along with the store state names now won’t be repeated every time a store sells a product.

# Summary and Reflection

The database I am trying to design is inspired by my desire to maintain a good business order management system for my future business. Primarily, I want this database to make any manager’s/ business owner’s life easier when it comes to managing inventory. The system if used properly, should improve inventory turnover, keep returning customers happy, help plan and predict future sales, help with cost-cutting and timesaving, etc. I know a lot of the information would have to be inputted by the user i.e., daily sales information. I addressed the inventory management system comment from last week’s iteration. I changed it to an order management system as I agree with the consultant’s suggestion. I also specified store location will be identified by the city name, and store id where the store is located. Lastly, the customer address is needed to send promo placards, verify customer returns, etc. purposes.

I think the database taking a nice shape with the added specialization-generalization rules and normalization. The conceptual diagram helped me visualize the important data that I need to capture. Then the structural rules created a blueprint and how the data should relate to each other. Lastly, Physical DBMS with normalization exhibited the entire design with all the details and relations between the tables. Although there might be room for improvement, I believe the database is much more structured and optimized than before. Would there be any more normalization possible?

Apart from those minor concerns I have noticed so far, I think this database design is very practical and useful. Although there are existing databases about inventory or order management, nothing beats the pleasure of developing your own order management system for your future business.