Cianna Grama
Professor Peitzsch
COM 303: Database Systems
10 May 2025

Zara Database Write-Up

Introduction to Zara

Zara Overview

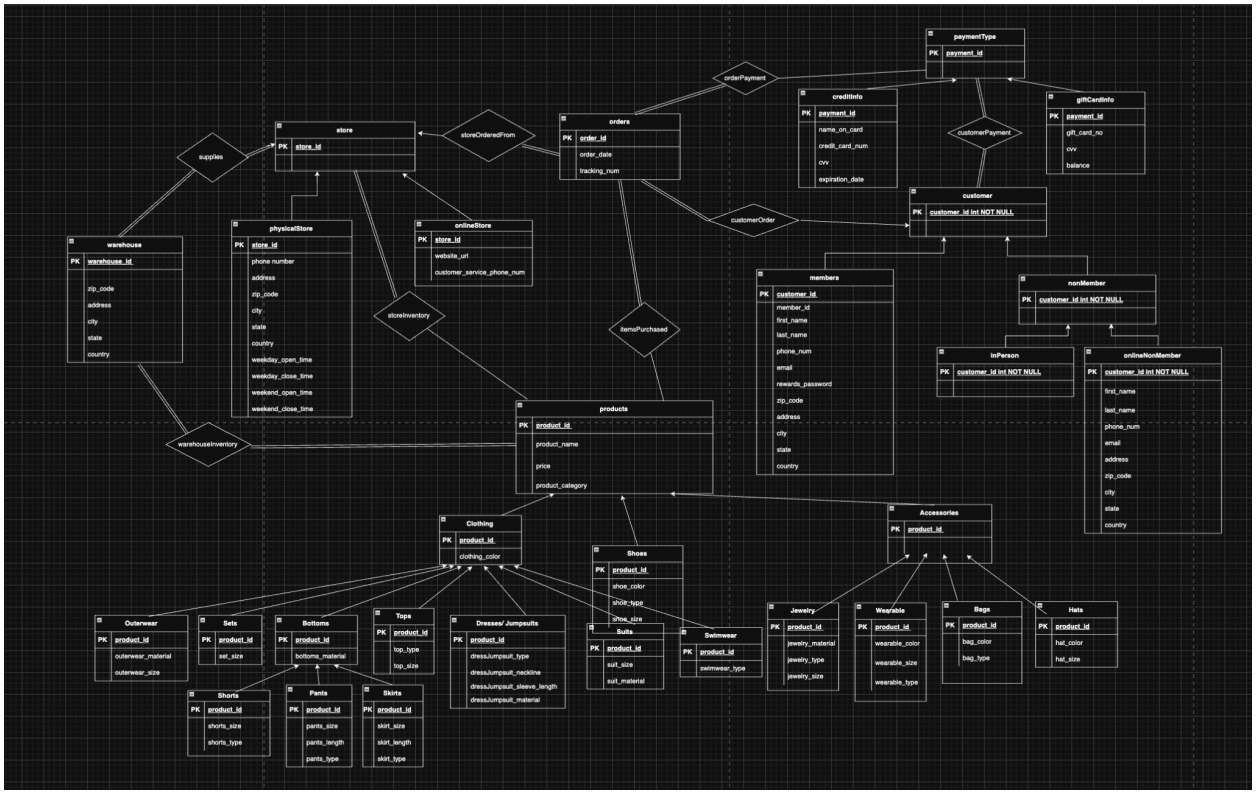
Zara is an international fashion retailer that follows the fast-fashion model of bringing fashion trends from runways to stores quickly. Zara was founded in 1975 by Amancio Ortega and Rosalia Mera in Galicia, Spain. Zara sells clothing, accessories, and shoes for men, women, and children that are stylish, affordable, and updated often for their customers. New items appear in Zara stores as often as twice a week. Zara is a large brand with thousands of stores in over 90 countries. Among the physical stores, Zara also has a large online selection of items.

Structures of Zara: Informing the Database Design

Zara has some unique structures that we had to keep in mind when designing the database. First, Zara is an international enterprise, so we needed to ensure the database could account for stores in different countries. Second, Zara has one online store for all countries, with slightly different interfaces for each country. There is technically only one website, so the online stores have the same backend for each country. The differences in the interfaces are in prices, currency types, and languages to account for those differences in different countries. The online stores use the same website and function the same in terms of usage and structure. Third, Zara has multiple locations in certain cities, so we ensured the database could handle that. Lastly, Zara only carries items that are their brand, so "brand" was not a part of our E-R model.

Entity-Relationship Model

The E-R model shows how the entities are connected in order to be able to access all the data in the database. For the purposes of organization, I will discuss the entities, their relationships, cardinality, and issues we encountered in creating the E-R in this section. I will go into detail about specific attributes within tables and their primary keys in the “Tables’ section. The full E-R model:



Entities

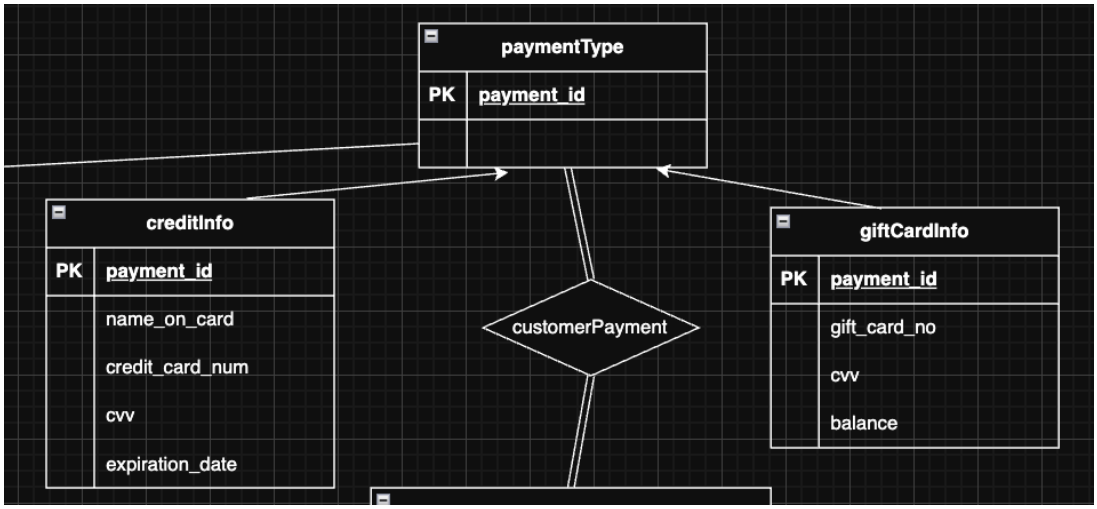
The main entities in the E-R model are payment type, customer, order, store, product, and warehouse. Payment type, customer, store, and product are all hierarchies to account for the complexities within each.

Payment Type

Payment type is an entity to account for a customer who has multiple forms of payment. Making payment type an entity rather than an attribute of customer allows for that possibility. Payment type is a hierarchy to account for different types of payments that a customer may use. The database allows for cards, gift cards, and cash.

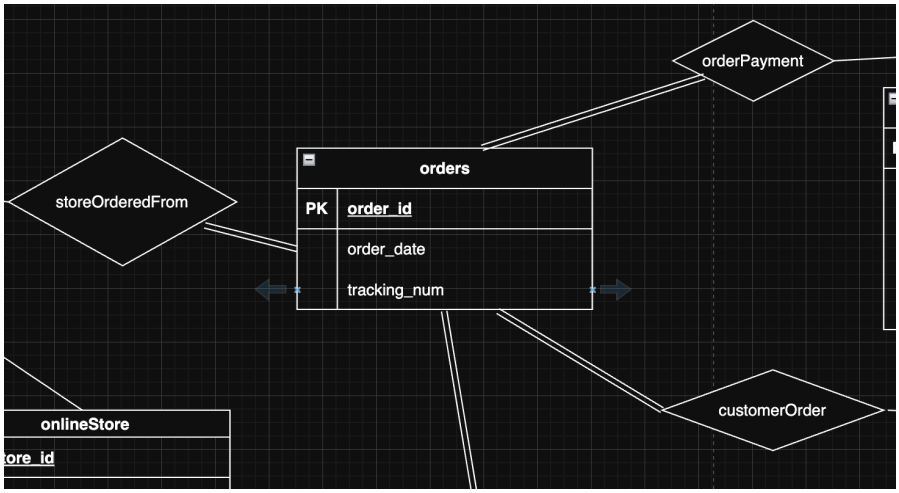
Payment type is a hierarchy with the superclass paymentType and two subclasses: creditInfo and giftCardInfo. PaymentType has one attribute payment_id, and each payment ID is either belonging to a credit card, a gift card, or cash. If the payment is cash, the payment ID would not be found in either of the subclasses and would only be in the paymentType table. If the payment is a card, then that payment ID would be in the paymentType table and the creditInfo table. So, the cardinality of paymentType to creditInfo is zero-to-one, meaning that not every entity in paymentType will be in creditInfo, but if an entity is, then there will only be one. The cardinality of creditInfo to paymentType is one-to-one, meaning that each entity in the creditInfo table must be associated with exactly one entity in paymentType. If the payment is a gift card, the payment ID would be found in the giftCardInfo table and the paymentType table.

So, the cardinality of paymentType to giftCardInfo is zero-to-one, meaning that not every entity in paymentType will be in giftCardInfo, but if an entity is, then there will only be one. The cardinality of giftCardInfo to paymentType is one-to-one, meaning that each entity in the giftCardInfo table must be associated with exactly one entity in paymentType.



Order

Order is an entity that contains every order that is placed within the database, distinguished by unique order IDs for every order. Orders include online purchases and purchases made in store, and multiple items purchased at once are considered to be part of the same order. Because orders are vital to Zara, this entity is very well connected in the database. Order has a relationship with payment type, customer, store, and products in the database, which I will go into further detail in the "Relationships" section. Initially, total_price was an attribute of the order entity, but it needed to be removed because it is a derived attribute of the total of the items purchased with the same order number. Without total_price as an attribute, the total price of an order can be accessed with a sum group by query of products purchased with the same order ID.

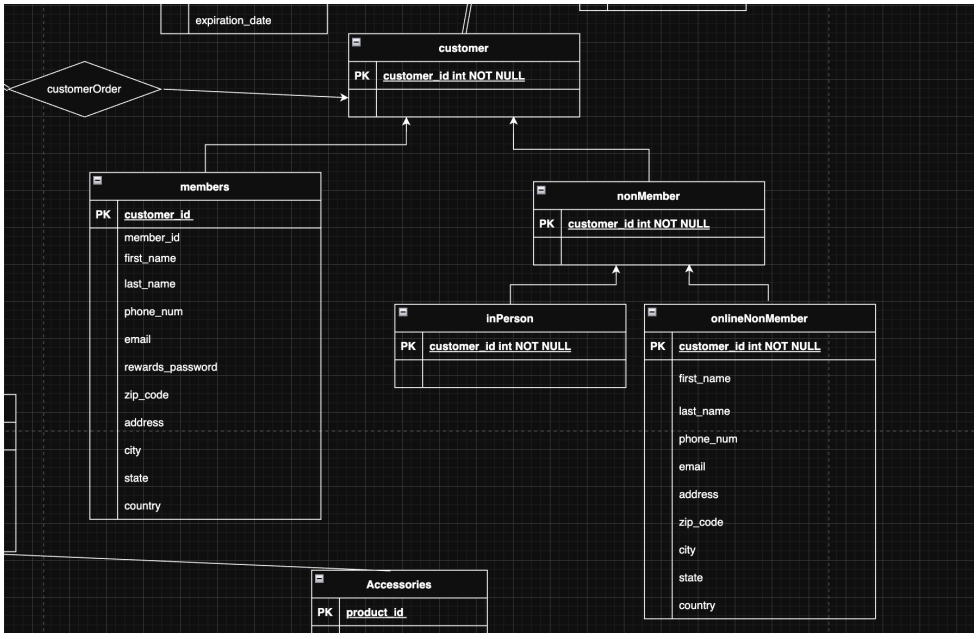


Customer

Customer is a hierarchy with the superclass customer. The next level of specification in the hierarchy is member or non-member, and the third level of specification is under non-member with two specifications: in person non-member and online non-member. Every single customer ID is also in a table in the specialization hierarchy. A customer is either a member, an in person non-member, or an online non-member. If a customer is a non-member, they belong to one of the specializations under non-member.

From the superclass, there is a disjoint and total specialization for member and non-member, meaning that an instance of customer can only belong to one subclass at a time- either member or non-member. The cardinality of customer to member and customer to nonMember is zero-to-one, meaning that a customer is either a member or not or a non-member or not. Under the subclass nonMember, there is a disjoint and total specialization for in person and online, meaning that an instance of nonMember will belong to either inPerson or onlineNonMember- only one, not both, and not neither. The cardinality of nonMember to onlineNonMember and nonMember to inPerson is zero-to-one, meaning that a non-member is either in person or online.

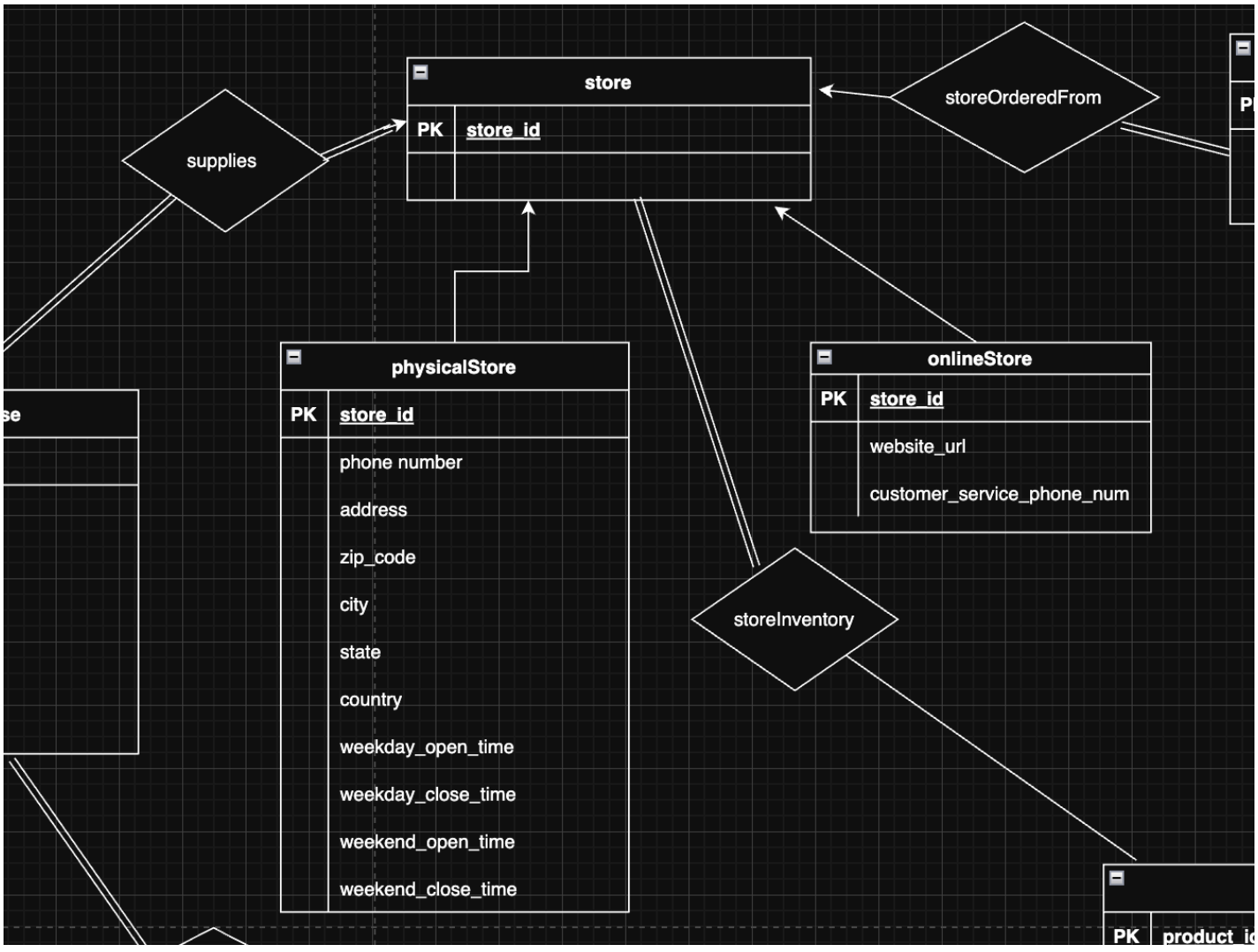
A difficulty we had was deciding how to identify customers who are non-members to do analytics. Since a person can have multiple payment methods and a payment method can be used for different customers, we could not use payment as an identifier. As we thought about this, we decided to not differentiate between customers who are non-members to protect their privacy. So, with the logic of the customer hierarchy, one non-member who uses two different cards would be listed as two separate customers in the database.



Store

Store is a hierarchy with the superclass store. The next level of specialization is physicalStore or onlineStore, and every single store is either a physical store or an online store. There is a disjoint and total specialization under store, meaning that an instance of store can either be physicalStore or OnlineStore, but not both, and not neither. The cardinality of store to physicalStore and store to onlineStore is zero-to-one, meaning that a store may be an online store or physical store, but only one. The cardinality of physicalStore to store is one-to-one, meaning that every physical store must be in the store table. The cardinality of onlineStore to store is also one-to-one, meaning that every online store must be in the store table.

The database technically only has one online store, since there is one central online store for Zara. In terms of functionality in our database, the online store and the relationships to store are the same for physical and online stores. So, the database treats online and physical stores as logically the same.

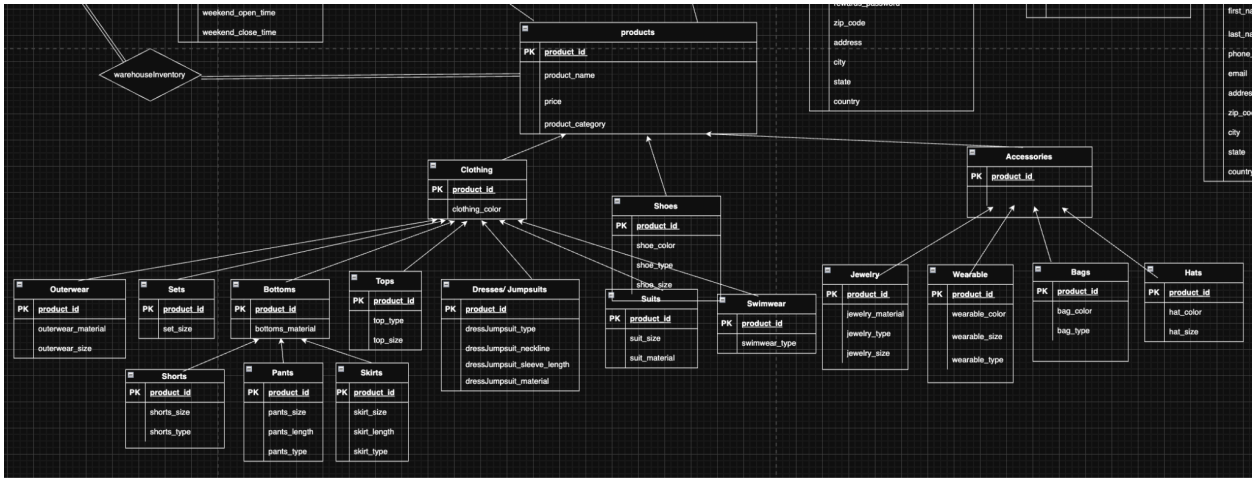


Product

Product is a hierarchy with the superclass products. The second level of specialization has three categories: clothing, shoes, and accessories. Within the clothing subclass on the second level, the next level has seven categories: outerwear, sets, bottoms, tops, dresses/jumpsuits, suits, and swimwear. Within the bottoms subclass on the third level, the bottoms are further specified as: shorts, pants, and skirts. Back to the second level, accessories have a further specialization of: jewelry, wearable, bags, and hats. Shoe on the second level does not have any further specifications. Every single product is also in at least one other table in the specialization hierarchy. A product is either clothing, accessories, or shoes. A clothing item is either outerwear, sets, bottoms, tops, dresses/jumpsuits, suits, or swimwear. Bottoms are either shorts, pants, or skirts. Accessories are either jewelry, wearable, bags, or hats.

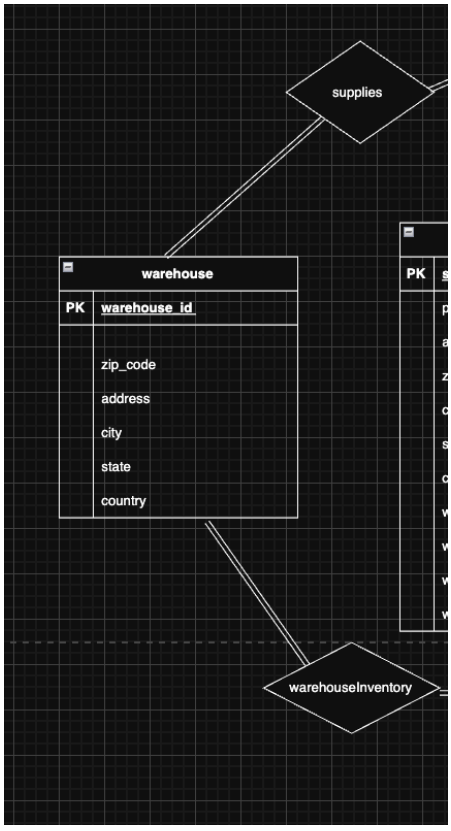
From the product superclass, there is a disjoint and total specialization for clothing, shoes, and accessories, meaning that an instance of a product can only belong to one of those subclasses at a time. The cardinality of product to clothing, product to shoes, and product to accessories is zero-to-one, meaning that a product is either clothing, shoes, or accessories, but not multiple and not none. Under the subclass clothing, there is a disjoint and total specialization for outerwear, sets, bottoms, tops, dresses/jumpsuits, suits, and swimwear, meaning that an instance of product will belong to one of those categories- only one, not multiple, and not none. The cardinality of clothing to the subclasses is zero-to-one, meaning that a clothing product fits into one of those specifications. The logic is the same for the bottoms specifications and the accessories specifications. The cardinality going from specialization to less specialized is one-to-one for all items in the product hierarchy, meaning that an instance of each specialized table must belong to a table in the level above.

The logic of products was difficult to determine, and we had a lot of drafts and changes throughout the process. We initially had product “category” (mens, womens, and childrens) as options on the second level of the hierarchy. This resulted in a very repetitive hierarchy where we had a different table for tops for men, women, and children. To reduce this repetition, we ultimately decided to make category an attribute of the superclass. So, the table specifications carry items for all three of the categories. For example, the shoes table contains shoes for men, women, and children. Another problem we struggled with was determining where to assign the size attribute. We initially had it as an attribute in the superclass, but because of the complexity and the size of each specialization, we made size an attribute in the lowest levels of the hierarchy.



Warehouse

Warehouse is an entity that contains the warehouses that supply the stores. We dealt with restocking through warehouses rather than through vendors. The vendors restock the warehouses, and then the warehouses restock the stores when necessary. The warehouse entity has a relationship with the store entity and with products entity, which I will go into further detail later. Logically, each store is supplied by one warehouse and one warehouse can supply multiple stores. It is also possible for one warehouse to only supply one store, but it is not possible for a warehouse to supply no stores.

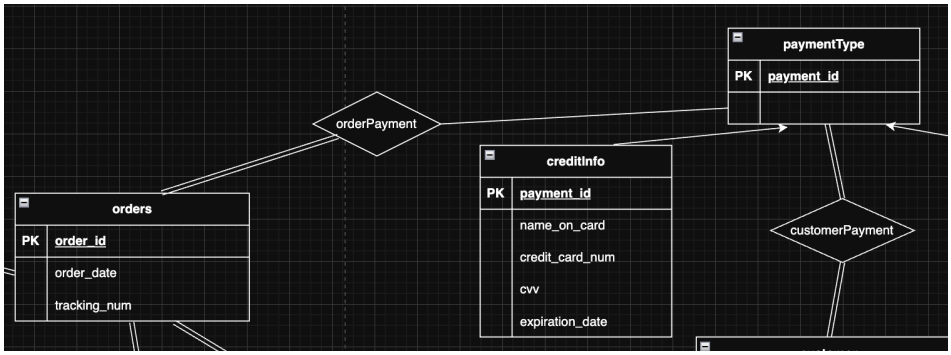


Relationships

The E-R diagram is reliant on relationships to connect the entities and allow for data to be accessible through queries. All relationships take the primary keys of the entities they are relating to and have them as attributes in the relationship. The specifics of primary keys for each relationship will be further discussed under the “Tables” section. This section will discuss the entities being related and their cardinality for each relationship.

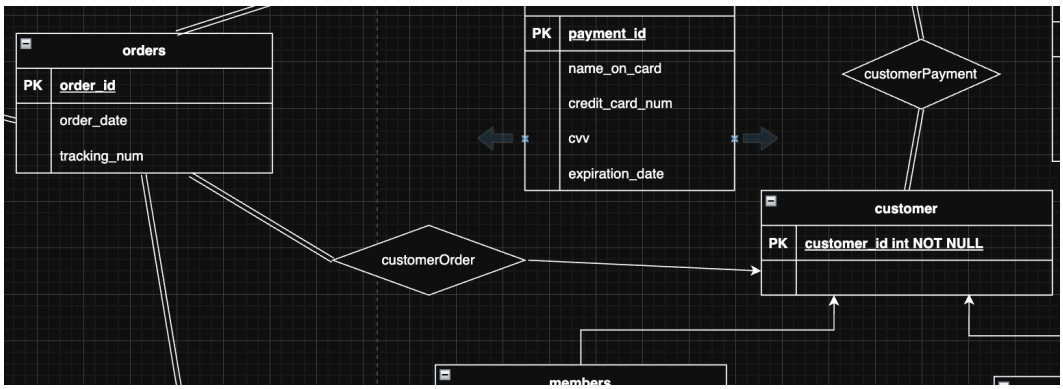
Order Payment

OrderPayment is a relationship between orders and paymentType. This relationship ensures that there is an associated payment for each order. The cardinality of orders to paymentType is many-to-one, meaning that each order must have one payment type, and a payment type can be associated with many orders or none at all in the orderPayment relationship. There is total participation on the orders side because every single order needs to have an associated payment.



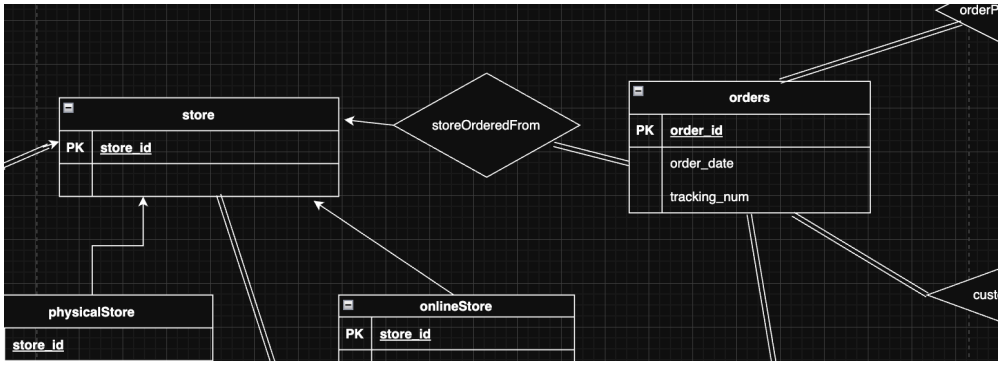
Customer Order

CustomerOrder is a relationship between customer and order. This relationship ensures that there is an associated customer for each order. The cardinality of orders to customer is many-to-one, meaning that each order must have one customer. A customer can be associated with many orders or none at all in the customerOrder relationship. There is total participation on the orders side because every single order needs to have an associated customer.



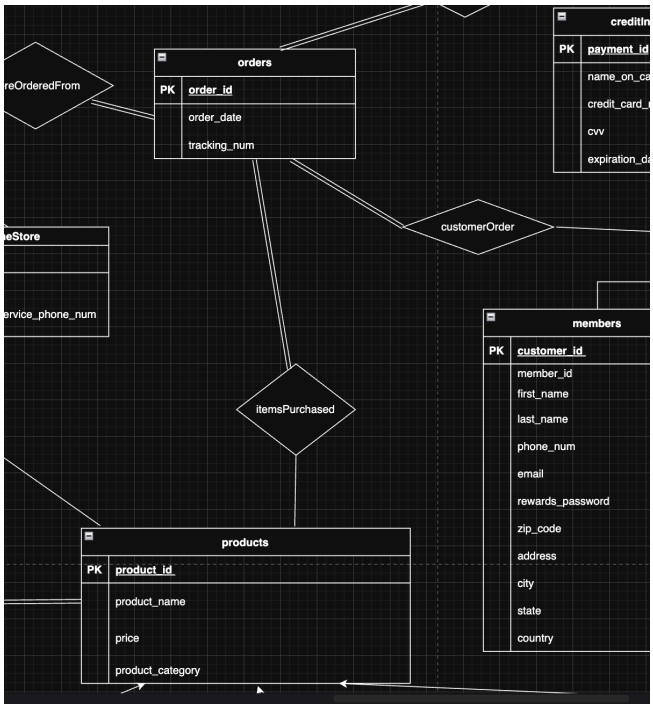
Store Ordered From

StoreOrderedFrom is a relationship between store and orders. This relationship ensures that there is an associated store for each order. The cardinality of orders to store is many-to-one, meaning that each order must have one store, and a store can be associated with many orders or none at all in the storeOrderedFrom relationship. There is total participation on the orders side because every single order needs to have an associated store.



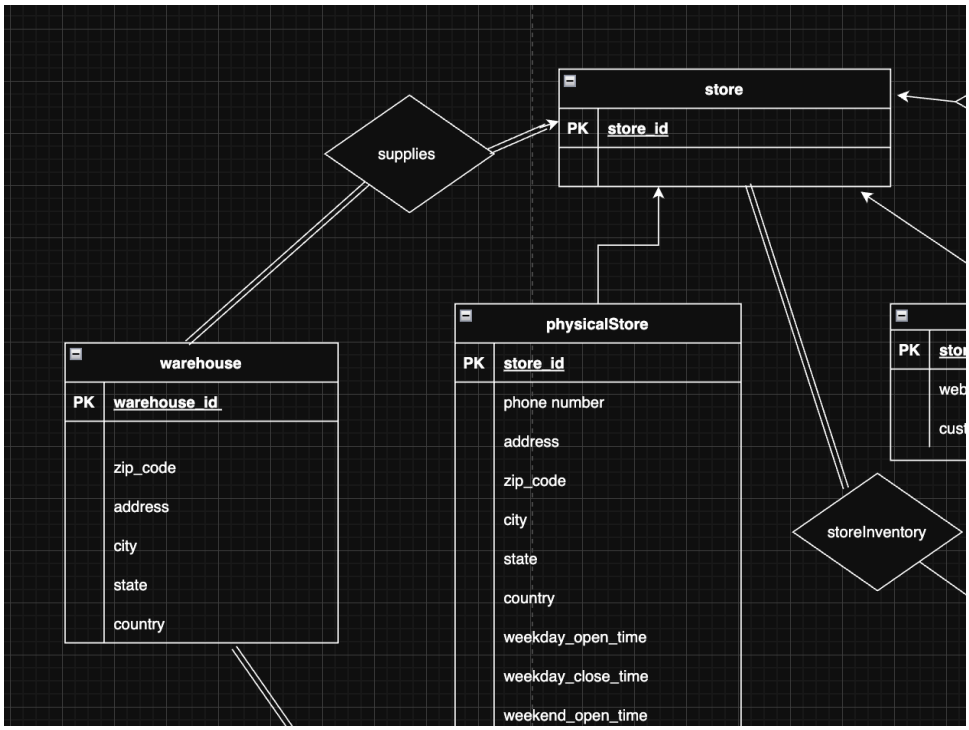
Items Purchased

ItemsPurchased is a relationship between orders and products. This relationship ensures that there is at least one associated product for each order. The cardinality of orders to products is many-to-many, meaning that each order can have multiple different products, and a product can be part of many different orders or none at all in the itemsPurchased relationship. There is total participation on the orders side because every single order needs to have at least one associated product.



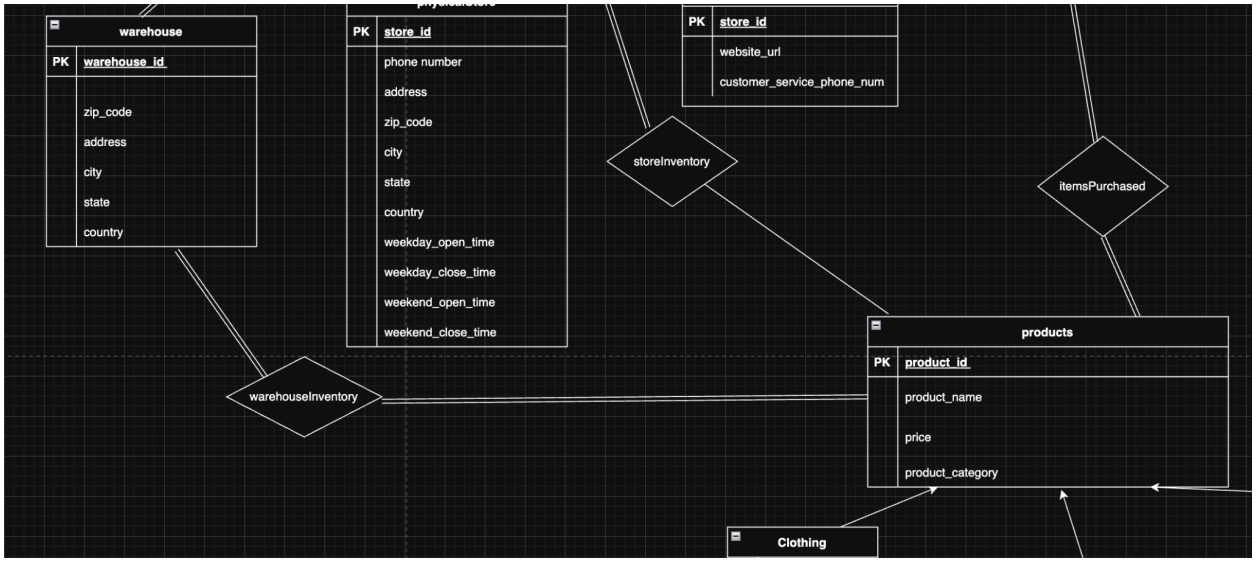
Supplies

Supplies is a relationship between warehouse and store. This relationship ensures that there is at least one associated warehouse for each store. The cardinality of warehouse to store is one-to-many with mandatory participation on the store side, meaning that each store is supplied by one warehouse, and warehouses can be supplied by many different stores in the supplies relationship. There is total participation on the warehouse side because every warehouse needs to have a store, and there is total participation on the store side because every store needs to have a warehouse.



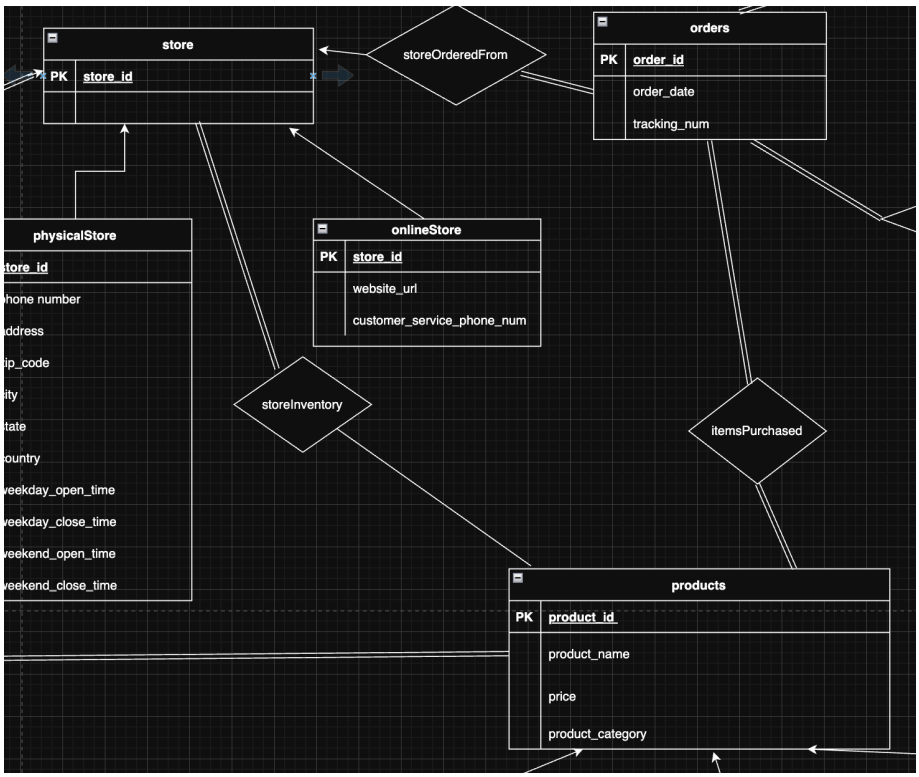
Warehouse Inventory

WarehouseInventory is a relationship between warehouse and products. This relationship ensures that the warehouses have inventory and contain the necessary products, and represents which products are held in inventory at each warehouse. The cardinality of warehouse to products is many-to-many, meaning that each warehouse can store every single product, and every product can be stored to every single warehouse. The design of our database is that every single warehouse has every single product so the warehouses can best supply the stores they are assigned to. There is total participation on the warehouse side because every warehouse needs to have all the products, and there is total participation on the products side because every product needs to be in all warehouses.



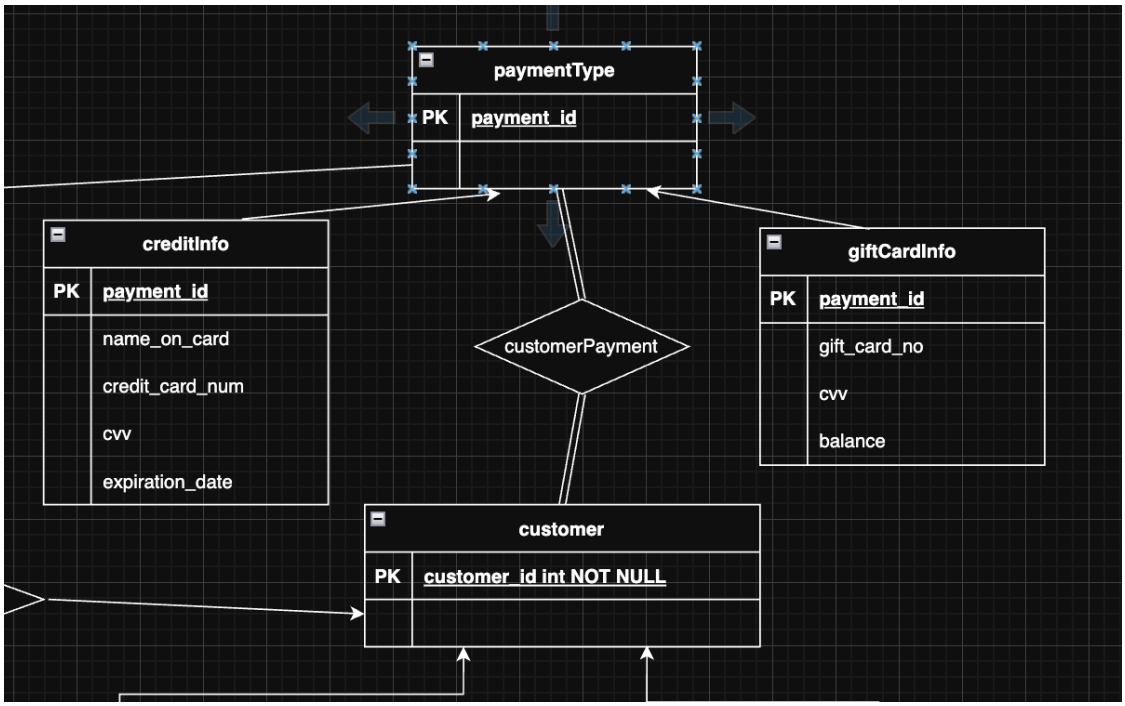
Store Inventory

StoreInventory is a relationship between store and products. This relationship ensures that the stores have inventory and contain the necessary products, and represents which products are held in inventory at each store. The cardinality of store to products is many-to-many, meaning that each store can have every single product, and every product can be in the inventory of every single store. There is total participation on the store side because every store needs to have a store inventory and contain some products.



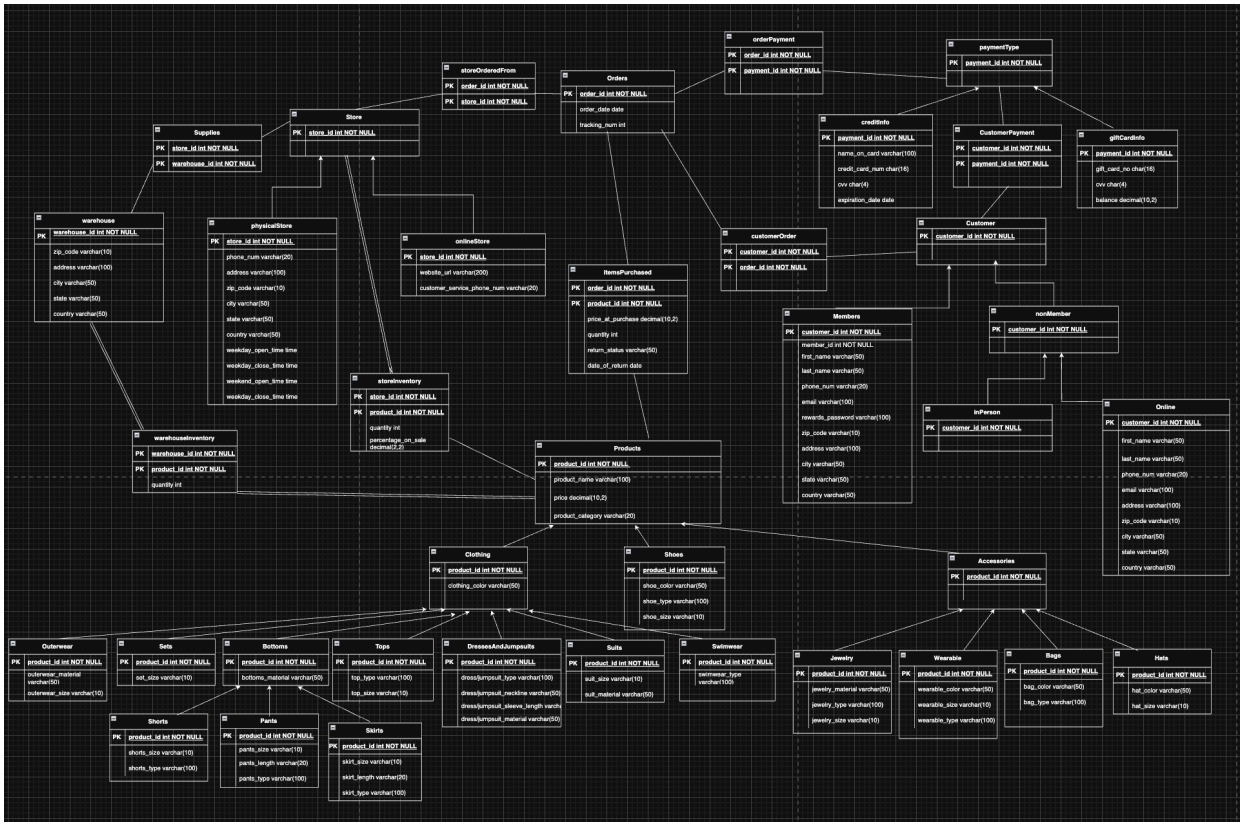
Customer Payment

CustomerPayment is a relationship between customer and paymentType. This relationship records which payment methods have been used by each customer. The cardinality of customer to paymentType is many-to-one, meaning that a customer can have multiple payment types, but each paymentType is associated with exactly one customer. Since some payment types are cash, there may not be much data on them and they will only be in the superclass, but they will still be stored in the paymentType table. There is total participation on the payment type side because every payment type needs to have a customer, and there is total participation on the customer side because every customer needs to have a payment type.



Tables

Overall table schema:

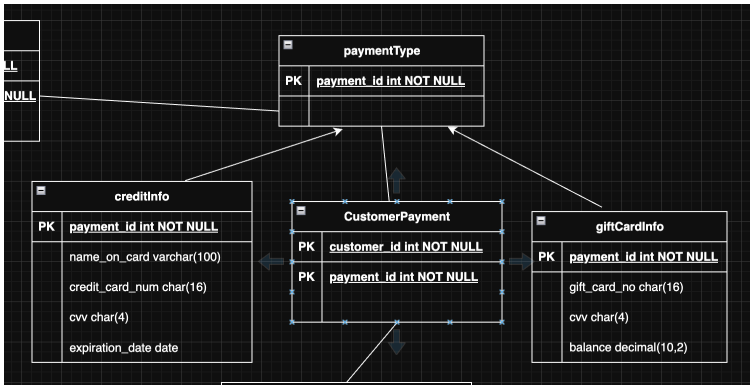


Entities

Each of the entities in the E-R model were directly turned into tables, and the hierarchies follow the logic described under the E-R Model section. Each hierarchy was implemented into tables by placing the primary key of the superclass as the primary key for all the subclasses.

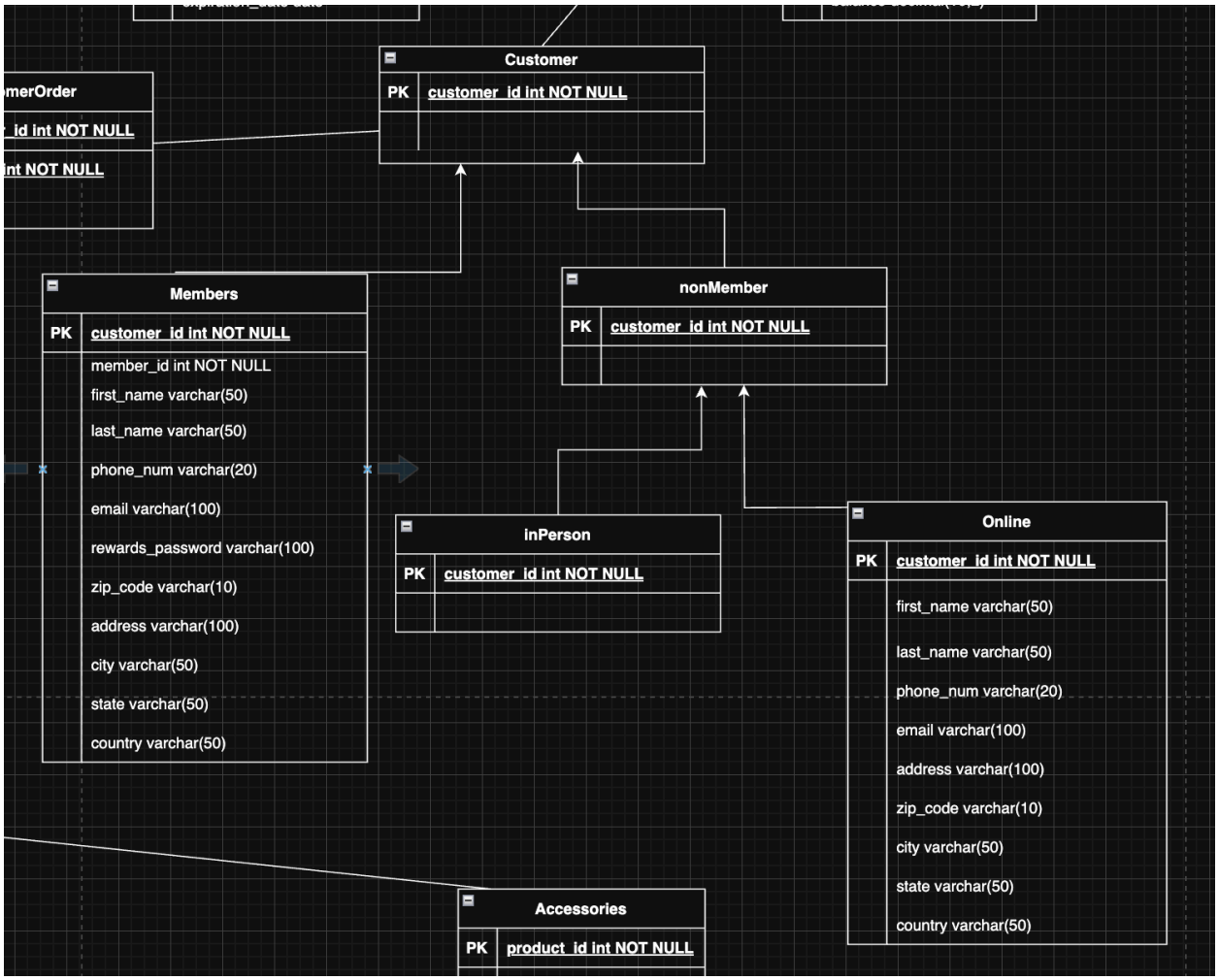
Payment Type

The superclass of payment type only has one attribute that is also its primary key: payment_id. Payment_id is the primary key of all the other tables in the hierarchy: creditInfo and giftCardInfo. CreditInfo contains the attributes name_on_card, credit_card_num, cvv, and the expiration_date. GiftCardInfo contains the attributes gift_card_no, cvv, and balance.



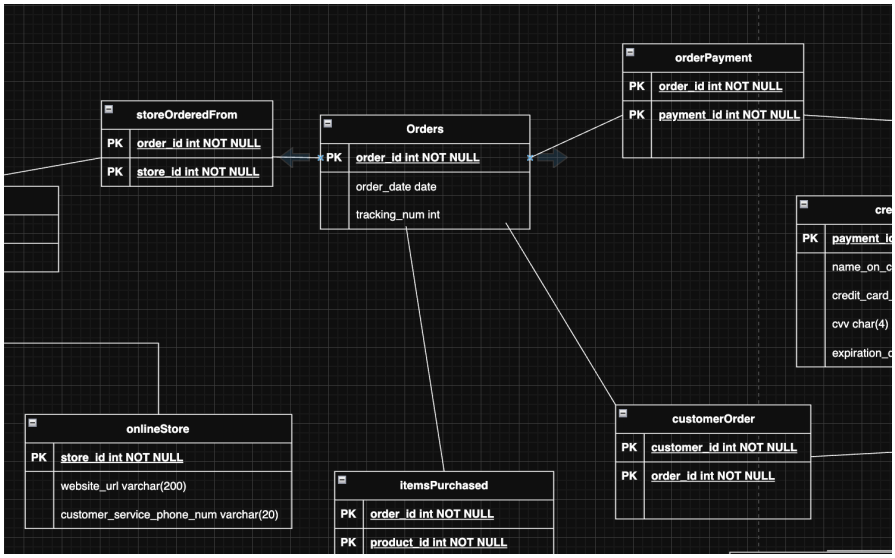
Customer

The superclass of customer only has one attribute that is also its primary key: customer_id. Customer_id is the primary key of all the other tables in the hierarchy: member, nonMember, inPerson, and online. Member contains the attributes member_id, first and last name, phone_num, email, rewards_password, and address information. The nonMember table only has one attribute that is also its primary key: customer_id. InPerson also only has one attribute that is also its primary key: customer_id. Online contains the attributes first and last name, phone_num, email, and address information, as well as customer_id as the primary key.



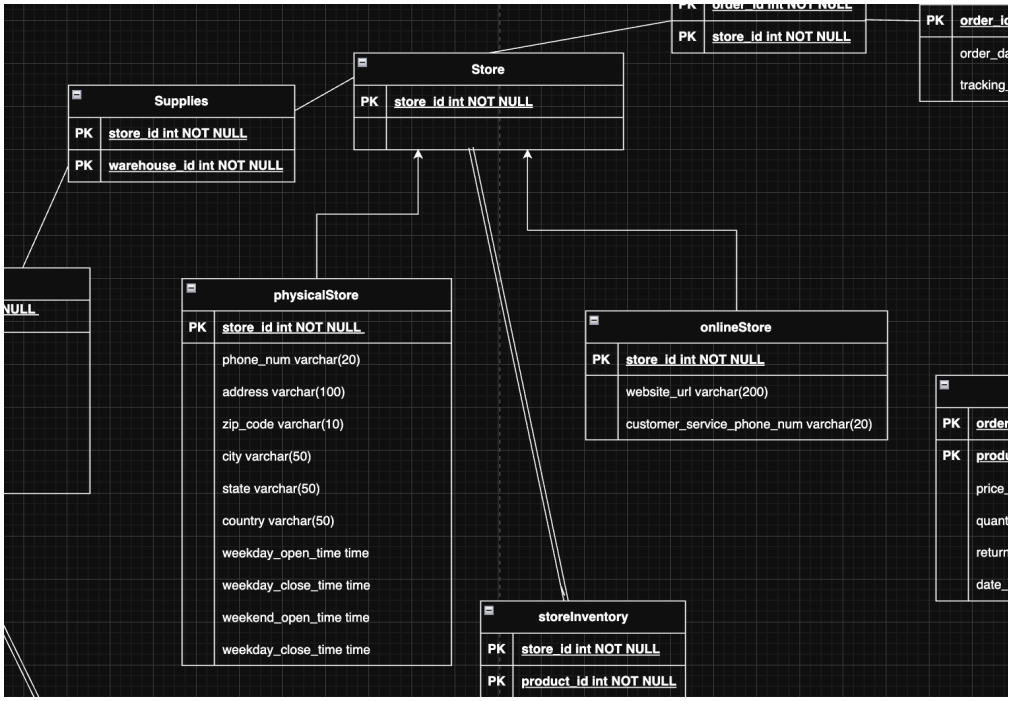
Order

The orders table contains three attributes. The primary key is order_id, which is an integer and cannot be null. This uniquely identifies each order in the table. The order_date attribute stores the date on which the order was placed. The tracking_num attribute is an integer representing the tracking number associated with the shipment of the order. This table is used to record basic order-level information such as when the order was placed and how it can be tracked.



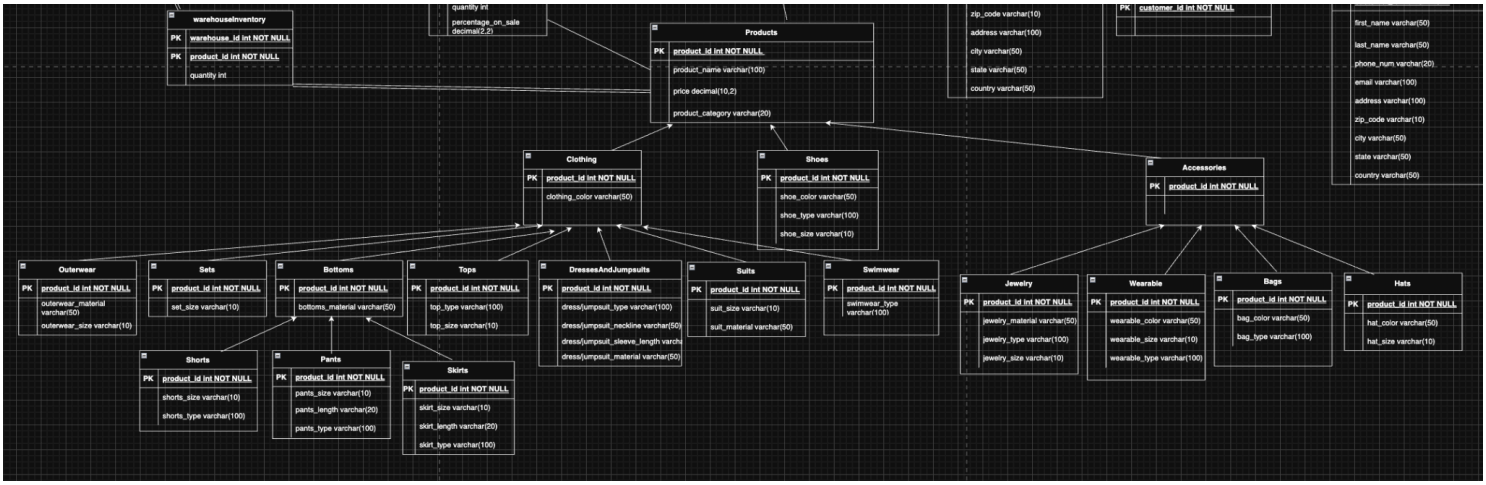
Store

The superclass of the store hierarchy is store, which only has one attribute that is also its primary key: store_id. Store_id is the primary key for all other tables in the hierarchy: physicalStore and onlineStore. The physicalStore table contains the attributes phone number, address, zip code, city, state, country, weekday open and close times, and weekend open and close times. The onlineStore table includes the attributes website URL and customer service phone number.



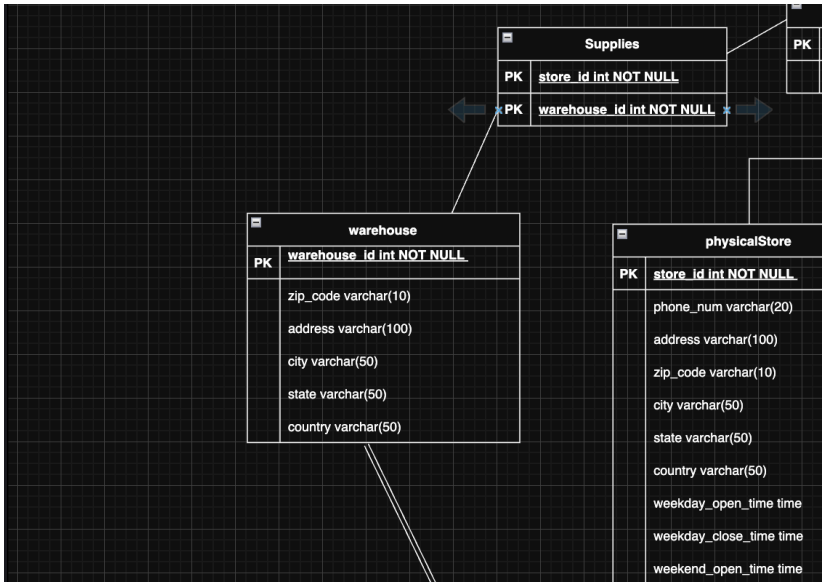
Product

The superclass of the product hierarchy is products, which contains three attributes: product_id (the primary key), product_name, price, and product_category. Product_id is the primary key of all the other tables in the hierarchy in order to maintain simplicity in the complex hierarchy. The next level contains the tables: clothing, shoes, and accessories. The clothing table contains the attribute clothing_color, and it serves as a superclass for further subcategories: outerwear, sets, bottoms, tops, dressesAndRompers, suits, swimwear, and skirts. Each of these subcategories includes specific attributes: outerwear_material and outerwear_size for outerwear, set_size for sets, bottoms_material for bottoms, and top_size and top_type for tops. Bottoms further branch into shorts, which includes shorts_size and shorts_type, and pants, which includes pants_size, pants_length, and pants_type. Skirt includes skirt_size, skirt_length, and skirt_type. DressesAndRompers includes attributes for neckline, sleeve length, and material. Suits include suit_size and suit_material, and swimwear includes swimwear_type. The shoes table contains the attributes shoe_color, shoe_type, and shoe_size. The accessories table contains no additional attributes but serves as a superclass for subcategories: jewelry, wearable, bags, and hats. Jewelry includes jewelry_material, jewelry_type, and jewelry_size. Wearable include wearable_color, wearable_size, and wearable_type. Bags include bag_color and bag_type, while hats include hat_color and hat_size.



Warehouse

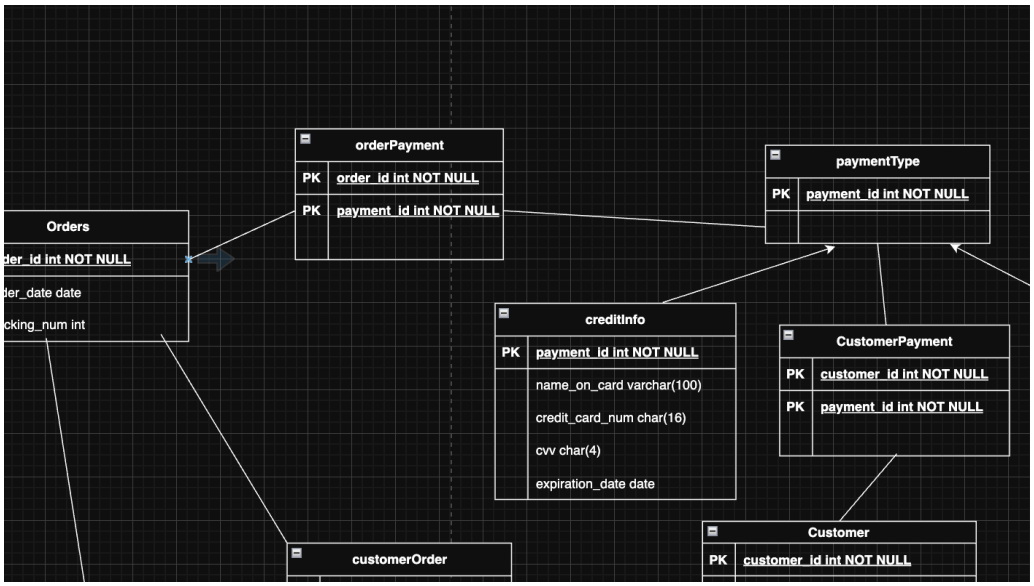
The warehouse table has the primary key warehouse_id. This uniquely identifies each warehouse. The warehouse table also contains the attributes zip_code, address, city, state, and country, which provide location and address details for each warehouse.



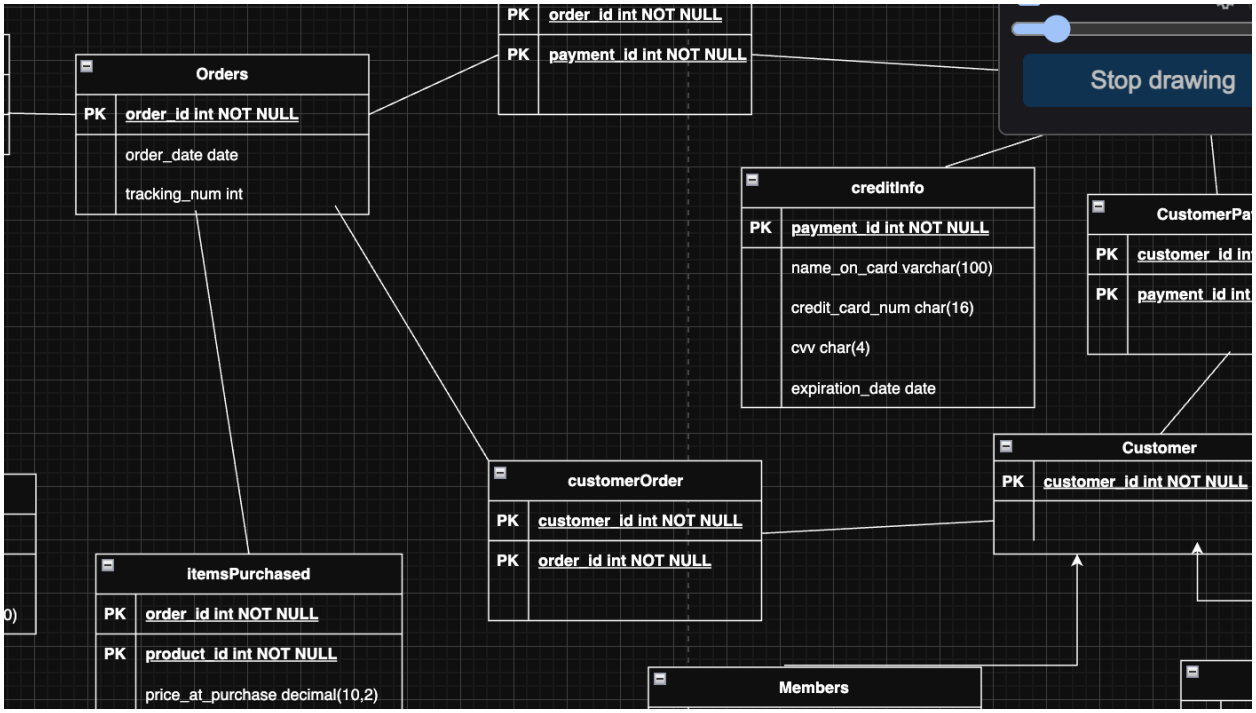
Relationships

All of our relationships were converted into tables, and the primary keys were assigned based on cardinality. The attributes in each relationship are the primary keys of the entities they relate together, but the primary key of the relationships depend on the cardinality of the relationships that were discussed in the "Entites" section of the E-R model. For the relationships that were one-to-many, the primary key of the many side became the primary key of the relationship. For the relationships that were many-to-many, then the primary key of the relationship became a combination of both of the primary keys. For the relationships that were one-to-one, one of the sides was chosen to be the primary key of the relationship.

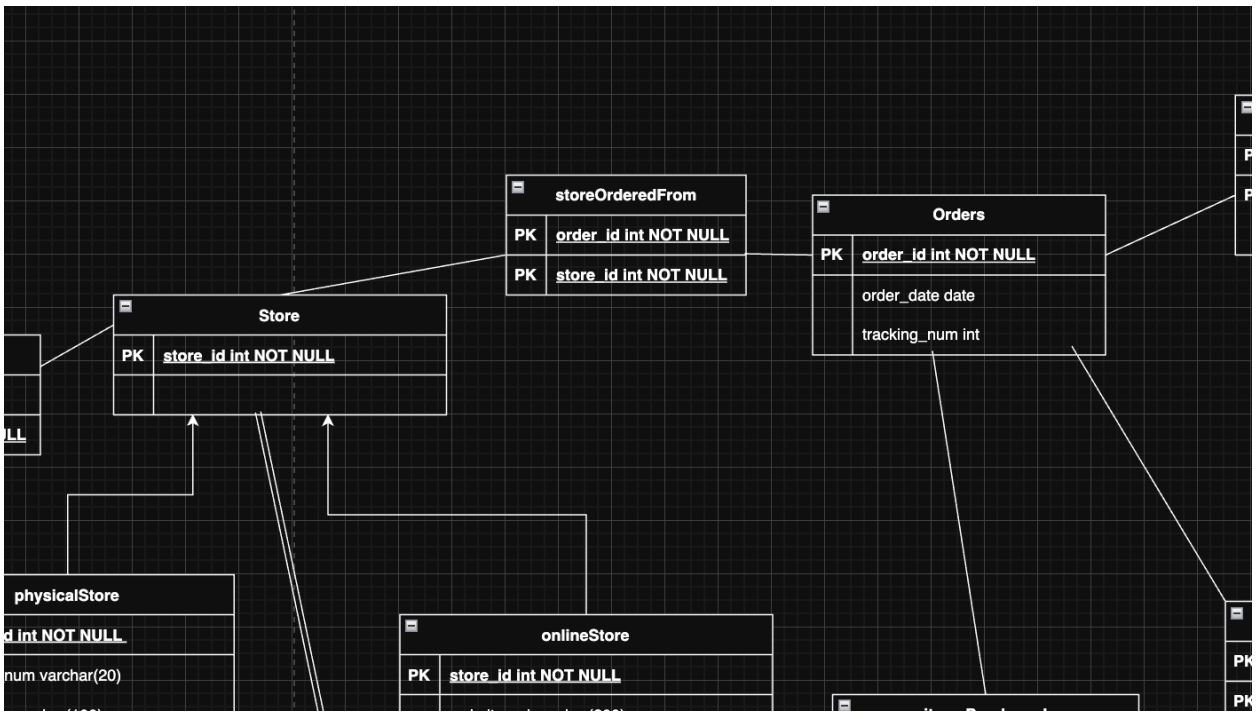
Order Payment



Customer Order

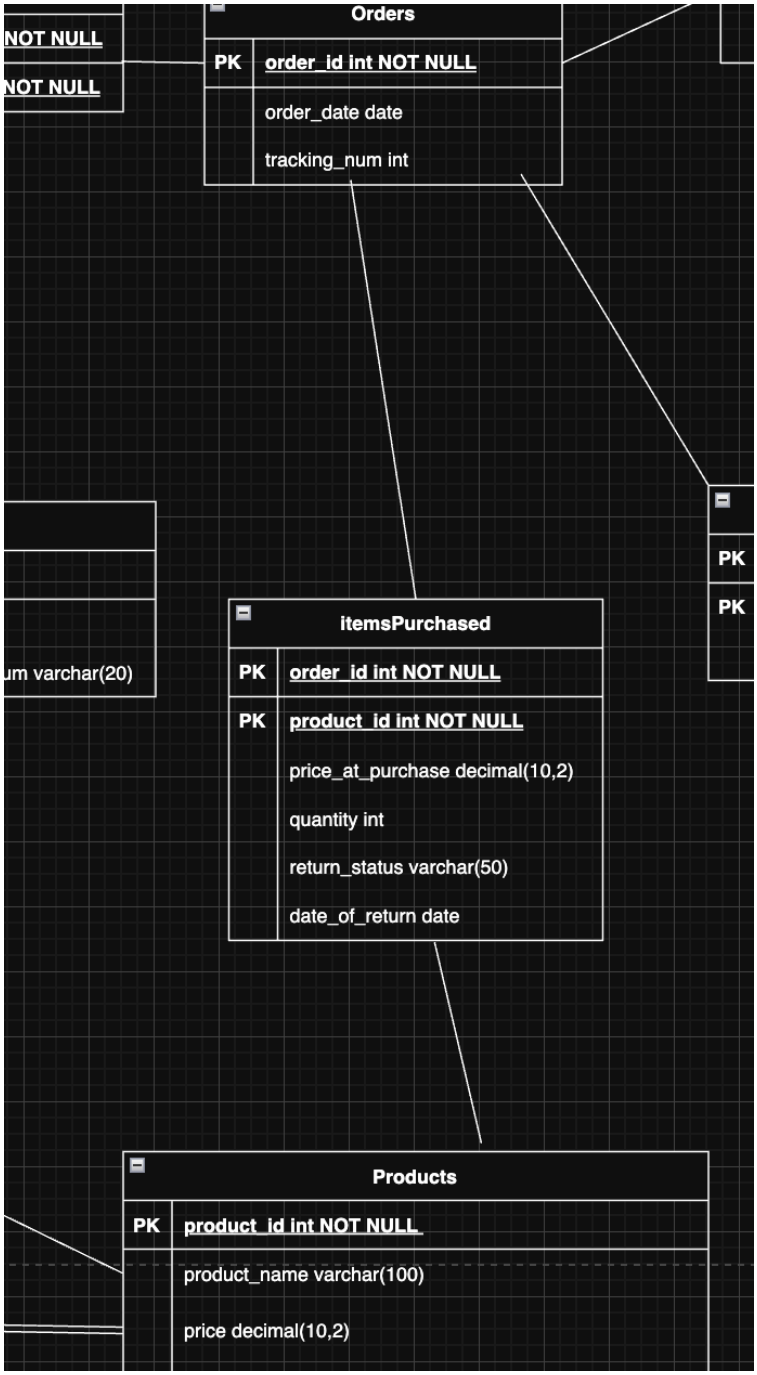


Store Ordered From

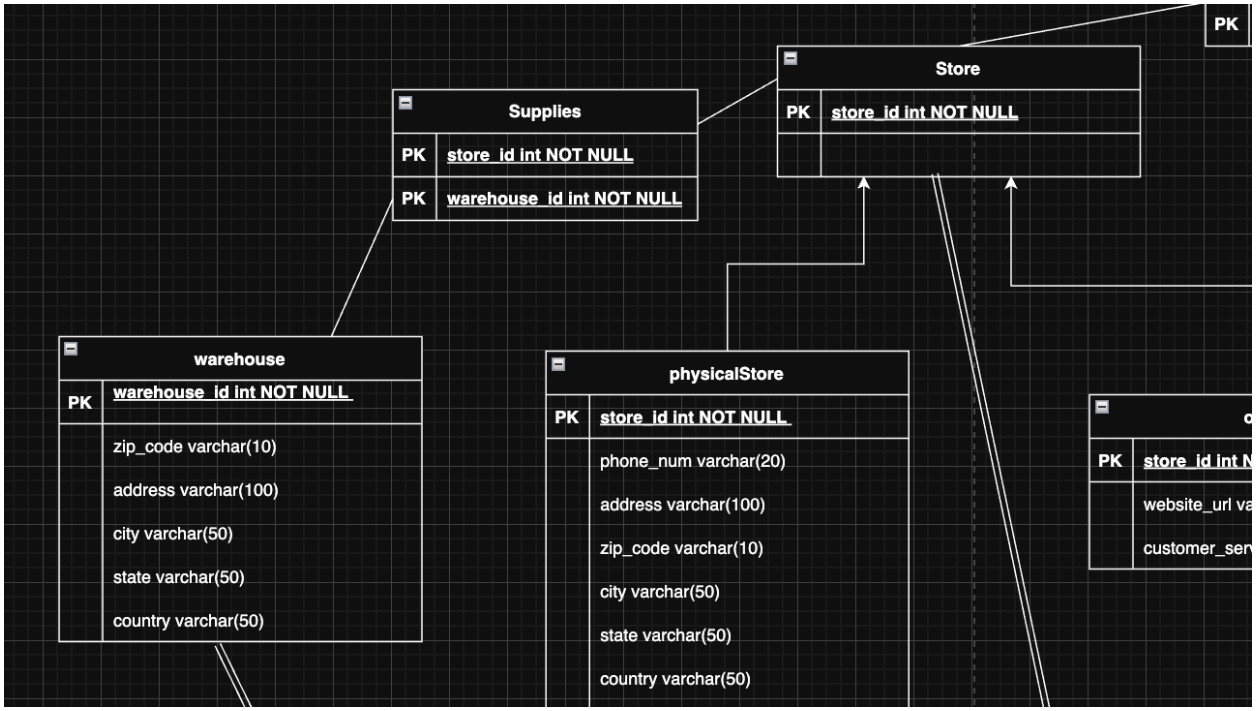


Items Purchased

Other than the attributes that are the primary keys of the tables itemsPurchased relates, itemsPurchased has some additional attributes. Price_at_purchase records the price of an item when it was purchased, calculated by the product price and the amount on sale an item is at the store it is being bought from. Quantity holds the quantity of a product purchased. Return_status holds a value that lets the system know if the item has been returned or not, and date_of_return holds the date that the item was returned, if applicable.

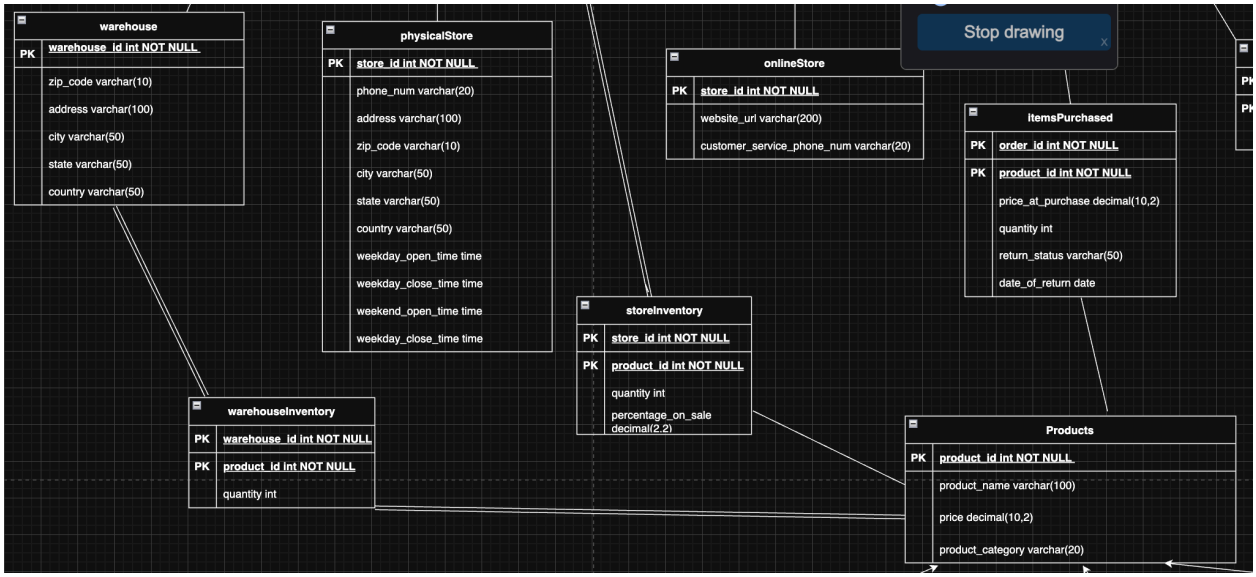


Supplies



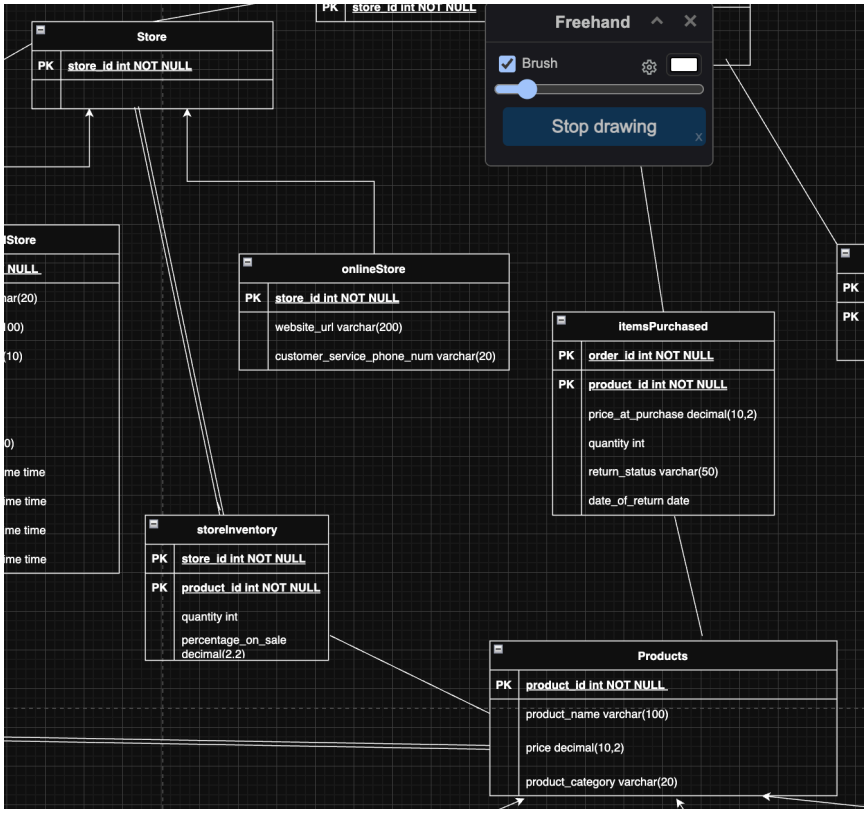
Warehouse Inventory

Other than warehouse_id and product-id, the warehouseInventory table also holds the attribute quantity that holds data on how much of an item is held at a specific warehouse.

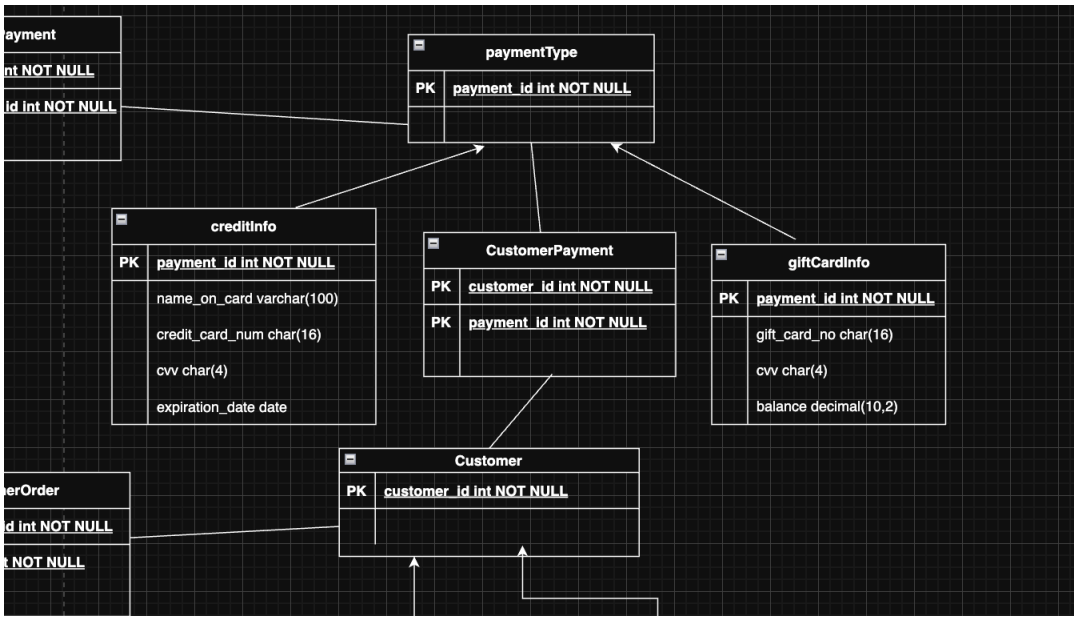


Store Inventory

Other than store_id and product-id, the storeInventory table also holds the attribute quantity that holds data on how much of an item is held at a specific store, as well as the sale percentage of each item.



Customer Payment



Difficulties Moving from E-R to Tables

When we moved over to tables, we realized that we needed to make orders its own entity. We originally had it as a relationship, but the transition made us realize that it functions as an entity and needs to be its own entity rather than a relationship.

Logical Structures of the Database

First, the online store inventory and physical store inventories are functionally the same in the database. The online store inventory has much higher quantities of items, but they act the same. In our database, there is technically a location that holds the online store inventory. It is technically a warehouse, but functionally a store, so it is logically considered a store in our database. This location is where all online orders are shipped out from. Second, the product ID in our database is functionally the same as a universal product code (UPC), but the UPC is externally associated outside of the database. This reduces the complications of UPCs in the database and allows the product ID to be incremented within the database.
The logic of purchases, restocks, and returns is as follows in our database: When a purchase is made, the quantity is an attribute in itemsPurchased, and that quantity is subtracted from the storeInventory quantity. Returns are handled by a decrease of the quantity value in itemsPurchased and an increase in the storeInventory quantity value of the store the item was purchased from. Store restocks are handled by a quantity checker. When the quantity in a store goes below 15 items, then a restock request is sent to the warehouse that supplies said store, and 100 new items are added to the inventory of the store and removed from the inventory of the warehouse.

Business Decisions

First, each product has a company-wide set price that is reflected in the price attribute of the products table. However, there are price differences for an item in different stores. This is the result of sales, which are an attribute in storeInventory. When an item is purchased, the price in product is multiplied by the amount on sale, and that purchase price is recorded in itemsPurchased under the attribute price_at_purchase. This ensures that different stores can have different product prices, and ensures that the purchase price is recorded if that data needs to be accessed. Second, physical store purchases and online purchases can be returned to any physical store. Once that item is returned, the store where the item was bought is found through a query and sent to that location. However, physical purchases cannot be returned online. Third, shipping is done through online orders only. Even if a customer orders a product online in a physical store, that goes into the system as an online order and is functionally the same. Fourth, our Zara database does not identify non-members to perform business analytic queries on specific people. Members have individualized data analytics because they agree to that as a part of our membership program, but non-members are not subject to the same analytics. Non-member purchases as a whole are still used in our analytics, but not individual analytics. Fifth, every

warehouse carries every product, as previously discussed. And lastly, when a new store is created, they are automatically assigned a warehouse to supply them.

Data Generation

We generated data by using the python Faker library to randomly generate entity data using realistic attribute values. Then, we used the entity data to populate the relationships, adding on some attributes with the Faker library generation when necessary. Then, we tested queries and altered our tables slightly based on our queries. The main alterations were removing derived attributes and removing unnecessary attributes. For example, we removed the total order price because it is derived and we removed warehouse_name as an attribute of warehouse because it is unnecessary.

We also altered the data generation to ensure logical accuracy. In customerPayment, we made sure the data populates so that every payment has a customer, and every member and online guest customer has assigned payments in the database. In orderPayment, we generated data so that order, payment, and customer are correctly corresponding. Each customer needs to have an order, each order needs to have a payment, but the payment of the order needs to be one that the customer has. We ensured this by populating all the data generation except for orderPayment, committing that data population, querying into the database to find the customer associated with each order, finding the payment associated with that customer, and then placing the associated payment and order into the orderPayment table. Lastly, in supplies, we ensured that every store is supplied by a warehouse, but that warehouses supply more than one store.

Significant Problems and Solutions

One major difficulty was determining how to create the logical structure of the E-R. We had many drafts and attempts at relating the data and went through different versions before landing on our official E-R. I discussed the problems within each entity and relationship in the E-R section. We solved this problem by asking lots of questions, working together, brainstorming separately, and then coming together to work to ensure our ideas were fresh.

Another major difficulty was data generation. Ensuring that the data populated correctly was vital to being able to test our database, and we ran into difficulties populating the relationship data. The entity data was easier to generate because we used the python Fake library to simulate real data, but the relationships could not be so easily generated. We solved this problem by making lists of the created IDs of the entities and then using those lists to populate the relationship data. Additionally, we had different approaches to hierarchy data populations, and it resulted in some confusion. I took the approach of generating bottom-up from the hierarchy to make sure that all the tables were evenly populated, and Carly took the approach of populating from top-down and randomly assigning tables in the hierarchy the generated IDs would fall under. We discovered that her method worked best for populating huge hierarchies,

like products, and that my method worked best for populating smaller hierarchies, like the members hierarchy.

Analytical Queries

We have implemented several analytical queries into our command line interface, falling under 3 categories: store trends, members trends, and overall Zara trends. Under store trends, the interface allows a manager to identify the top 20 selling products at a given store and the 5 stores with the highest revenue. Under member trends, the interface allows managers to see the products a given member has bought and what stores they bought them from, the order history for a given member, and frequent shopper data that shows the number of orders per customer. Under overall Zara trends, the interface allows a manager to identify the top 20 selling products overall, the top 20 selling products by state for each state, identify the stores with the most sales, identify the products that are most often bought together, and identify the products that outsell each other. In this section, I will be thoroughly explaining a select 5 of these queries that are the most complex and most representative of the databases's ability to interact with queries.

Although the interface for our database only includes the queries mentioned above, the success and functionality of those queries within the database demonstrate the database's solid structure and ability to handle complex queries. While it's impossible to anticipate every analytical query a manager might request, the queries we've included represent important analytical use cases and prove that our system is capable of supporting additional, similarly complex queries.

Top 20 Selling Products at a Given Store Location

One store trend query gathers the top 20 selling products at a given store location from the database. This query is useful in business analytics for Zara managers because it gives insights on what kind of products sell best in certain stores, allowing managers to increase the types of items that sell best for each store. The query is:

```
select p.product_id, p.product_name, sum(ip.quantity) as totalQuantitySold
    from storeOrderedFrom as s join orders as o on(s.order_id = o.order_id)
    join itemsPurchased as ip on(o.order_id = ip.order_id)
    join products as p on(ip.product_id = p.product_id)
    where s.store_id = STORE_ID
    group by p.product_id, p.product_name
    order by totalQuantitySold desc
    limit 20;
```

In practice, "STORE_id" in the where line is replaced with a store ID that is given by the manager in the interface. This query joins the storeOrderedFrom table (which is a relationship between store and order) with the itemsPurchased table (which is a relationship between orders and products) based on the order ID. Then, that output is joined with products based on product

ID. Although itemsPurchased is a relationship with products, the products table is brought in to get the product_name data into the output, since product_name is not an attribute of itemsPurchased. The where clause selects only the data that matches the store ID being queried for. The data is grouped by product ID and product name in order to get the sum of the quantities for each product, which is called totalQuantitySold in the output table. Then the data is ordered by the totalQuantitySold in a descending order so the top products are shown at the top. Lastly, the output is limited by 20 to reduce analytical complexity for the user.

An example output of this query for a store with store_id = 2 is as follows, showing the product ID, product name, and sum of the total quantity sold:

	product_...	product_name	totalQuantityS...
	6	Babys Accessory	27
	7	Girls Bottom	24
	8	Girls Set	17
	4	Womens Accessory	16
	1	Boys Top	15
	2	Babys Set	14
	3	Girls Accessory	12
	9	Girls Shoes	12
	5	Babys Set	8
	10	Mens Top	5

Note: a similar output can also be seen within the command line interface under the manager menu, but the output values may be slightly different due to differences in the randomized data generation.

Top 20 Selling Products Company-WIDe

One overall trend query gathers the top 20 selling products company-wIDe. This query is useful in business analytics for Zara managers because it gives insights on what kind of products sell best in Zara overall. The company can then focus on getting more of these types of their products in their stores, which could potentially boost sales. The query is:

```
select p.product_id, p.product_name, sum(ip.quantity) as total_quantity_sold
    from products as p join itemsPurchased as ip on(p.product_id = ip.product_id)
    group by p.product_id, p.product_name
    order by total_quantity_sold desc
    limit 20;
```

This query returns the product ID, product name, and total quantity sold for the top sold 20 products in the database overall. The query joins the products table with the itemsPurchased table (which is a relationship between orders and products) based on the product ID. Although itemsPurchased is a relationship with products, the products table is brought in to get the product_name data into the output, since product_name is not an attribute of itemsPurchased. The data is grouped by product ID and product name in order to get the sum of the quantities for each product, which is called total_quantity_sold in the output table. Then the data is ordered by

the total_quantity_sold in a descending order so the top sold products are shown at the top. Lastly, the output is limited by 20 to reduce analytical complexity for the user.

An example output of this query is shown here, showing the product ID, product name, and the total quantity sold:

product_...	product_name	total_quantity_s...
6	Babys Accessory	127
5	Babys Set	108
1	Boys Top	103
10	Mens Top	99
7	Girls Bottom	98
3	Girls Accessory	97
4	Womens Accessory	88
2	Babys Set	84
8	Girls Set	81
9	Girls Shoes	65

Note: a similar output can also be seen within the command line interface under the manager menu, but the output values may be slightly different due to differences in the randomized data generation.

Top 3 Products Most Purchased with a Given Product

Another overall trend query gathers the top three products that are most often purchased with a given product that is inputted in the command line. This query is useful in business analytics for Zara managers because it allows managers to discover what kinds of items sell best together. This data can be used to inform marketing because advertisements can be created to market communally bought products together, which could potentially boost sales. The data from this query can also be used to inform product placement in stores. Items that are often bought together can be located near each other in stores and listed together online, which also has the potential to boost sales. The query is:

```
with p1_orders as (
        select distinct ip.order_id
        from itemsPurchased ip
        where ip.product_id = {PRODUCT INPUT})
    select p.product_name, count(*) as times_bought_with_p1
    from itemsPurchased as ip join products as p on(ip.product_id = p.product_id)
    where ip.order_id in (select order_id from p1_orders)
        and ip.product_id != {PRODUCT INPUT}
    group by p.product_name
    order by times_bought_with_p1 desc
    limit 3;
```

In practice, "{PRODUCT INPUT}" is replaced with the product ID provIDed by the user through the interface. This query uses a with statement to create a temporary result called

p1_orders, which contains the distinct order IDs from the itemsPurchased table where the product ID matches the inputted product. The main query then finds the names of products and counts how many times each one was purchased in the same order as the inputted product. It joins the itemsPurchased table (which is a relationship between orders and products) with the products table using the product ID. In the where clause, the query filters for rows where the product ID does not match the inputted product (so the input itself isn't counted) and the order ID appears in the p1_orders result. The output is grouped by product_name and ordered by the count of times each product appears alongsIDe the inputted product. Lastly, the query limits the output to the top 3 items most commonly purchased with the given product ID. and finally limited to the top 3 most commonly co-purchased products.

An example output of this query is shown below for the product ID 1:

product_name	times_bought_with_p1
Babys Set	30
Girls Set	16
Babys Accessory	16

Note: a similar output can also be seen within the command line interface under the manager menu, but the output values may be slightly different due to differences in the randomized data generation.

Compare the Times 2 Given Items Outsell Each Other

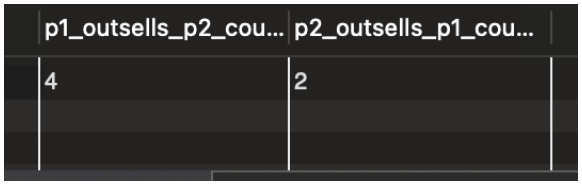
Another overall trend query compares the amount of times that two given items outsell each other. This query is useful in business analytics for Zara managers because it allows a manager to compare two different items when decIDing what products to stock stores with. The data can help boost sales by allowing managers to choose products that often outsell other items. The query is:

```
with sales_by_store as (
        select so.store_id, p.product_id,  p.product_name, sum(ip.price_at_purchase * ip.quantity) as total_sales
        from itemsPurchased as ip join products as p on(ip.product_id = p.product_id)
        join storeOrderedFrom as so on(ip.order_id = so.order_id)
        where p.product_id in ({PRODUCT 1}, {PRODUCT 2})
        group by so.store_id, p.product_id, p.product_name ),
    sales_comparison as (
        select
            store_id,
            max(case when product_id = {PRODUCT 1} then total_sales else 0 end) AS p1_sales,
            max(case when product_id = {PRODUCT 2} then total_sales else 0 end) AS p2_sales
        from sales_by_store
        group by store_id
    )
    select
        count(case when p1_sales > p2_sales then 1 end) AS p1_outsells_p2_count,
        count(case when p2_sales > p1_sales then 1 end) AS p2_outsells_p1_count
    from sales_comparison;
```

In practice “{PRODUCT 1}” is replaced with a product ID provIDed by the user through the interface and “{PRODUCT 2}” is replaced with another product ID provIDed by the user. The query uses a with statement to create a temporary result called sales_by_store, which contains the store ID, product ID, product name, and the total sales for each product at each store. Total sales are calculated by multiplying the price at purchase by the quantity of each item. To produce this result, the query joins the itemsPurchased table (which represents a relationship between orders and products) with the products table using product_id, and then joins the result with storeOrderedFrom (a relationship between stores and orders) using order_id. The query filters the data to only include entries where the product ID matches either of the two selected product IDs, and the results are grouped by store ID, product ID, and product name.

The second part of the query uses another with clause to create a temporary result called sales_comparison. This subquery takes the sales_by_store result and groups it by store, using case statements to pull out the total sales for each product indivIDually. Specifically, it computes p1_sales as the total sales of the first given product and p2_sales as the total sales of the second given product for each store. This is done using max case when statements to isolate the totals for each product. The last part of the query compares these sales across stores, counting how many stores had greater sales of the first given product over the second given product, and vice versa. The output includes two values: one showing how many stores the first given product outsold the second given product, and another showing how many stores the second given product outsold the first given product.

An example output of this query is shown below with product_id 2 and product_id 3 as the chosen products to compare.



p1_outsells_p2_cou...	p2_outsells_p1_cou...
4	2

Note: a similar output can also be seen within the command line interface under the manager menu, but the output values may be slightly different due to differences in the randomized data generation.

5 Stores with the Highest Revenue

Another store trend query finds the 5 stores with the highest revenue in the database.This is a useful analytical query because it gives Zara managers information about which stores they should invest in to continually bring in high revenue. The query is:

```
select s.store_id, sum(ip.price_at_purchase * ip.quantity) as total_sales
    from storeOrderedFrom as s join orders as o on(s.order_id = o.order_id)
    join itemsPurchased as ip on(o.order_id = ip.order_id)
    join products as p on(ip.product_id = p.product_id)
    group by s.store_id
    order by total_sales desc
    limit 5;
```

This query selects the store IDs along with their total sales, which are calculated by summing the product of price_at_purchase and quantity from the itemsPurchased table. The query starts by joining storeOrderedFrom (a relationship between stores and orders) with the orders table using the order_id. This result is then joined with itemsPurchased using order_id, and finally with the products table using product_id to retrieve product-level details. The results are grouped by store_id, and total sales are ordered in descending order so that the highest-revenue stores appear at the top. The output is limited to the top 5 stores by total sales.

An example output of this query is shown below.

store_id	total_sales
3	18018.00
4	15174.00
2	13700.00
7	13252.00
5	12729.00

Command Line Interface

Our command line interface has two main parts: 1) a menu for customers to place online orders, and 2) a menu for managers to run analytical queries, add new products, and add inventory to stores and warehouses.

Customer Interface

The customer interface allows customers: members and non-members, to place online orders through the interface. If a customer is a member, they are prompted to enter their customer ID. The system queries into the database, and if the customer ID inputted is not part of the member table, then a message alters the user that they are not a member. If the customer ID inputted is that of a member, then the member is welcomed into a member screen. The member menu allows members to place orders, view order details, add new payment methods, and change personal information. When a member places an order, a list of available products with details and quantities appears on the screen for the ease of the customer. The member is then prompted to select the product ID and quantity of the item they would like to purchase, and they can add multiple different items to their cart. Once they continue, the members' list of saved payments

appears on the screen and they are prompted to enter the payment ID of the payment type they would like to use for the purchase. Once they select one, their order is placed and added to the orders table as well as the tables that "orders" has a relationship with. The order ID is given to the member for future reference.
If a customer is not a member and is trying to place an online order, they are given the option to either check out as a guest or create an account. If they choose to create an account, their account is created as a new row in the members table, and they are taken to the member menu and can order through that interface. If they check out as a guest, the order placement follows the same steps as for members, but the guest is prompted to enter their payment information and their shipping information.

Each time an online order is placed, a function runs to check the quantity. If the quantity is below 20, a reorder is initiated: 100 products are taken from the warehouse that supplies the online store, and 100 products are removed from that warehouse.

Manager Interface

The manager interface allows Zara managers to conduct the analytical queries discussed above in the “Analytical Queries” section. The manager menu allows managers to view store, member, and overall trends. The queries query into the database and product printed outputs for the ease of the manager. Managers are also able to add inventory and products to the stores and warehouses.