Milestone 1

IMPROVEMENTS POSSIBLE AND WHAT I UNDERSTAND:

**PRODUCT:**

1. Change brand from an attribute to an entity and then say a **Product** has **Brand**(brandUUID,name).
2. Sku’s are the same for a product sold by the same supplier since it is a way of identifying a product in stock
3. I can add a stock attribute to the product entity to keep track of whether the product is in stock or not with possible values [inStock,outOfStock,preOrder]

**CUSTOMER:**

1. Both Supplier and Customer entities are subtypes of the User entity which should have the login credentials in tuples like (email,password,type) . NB email here which is the username has nothing to do with the business emails that users provide on their profile. I could as well just use a default userHandle for **username**.
2. **cardType** of a **PaymentCard** is the type of card eg Visa, MasterCard…..
3. **A Customer** reviews a **Product** and the **Customer** **Product** join here is called **Reviews** which has attributes [reviewID, author[fk(customerID)], about[fk(productID)], reviewBody, reviewRatingValue, dateCreated, dateModified]
4. **Carts** join table only needs fk constraints and a **dateCreated** field.

**ORDER:**

1. **orderStatus** has values (‘Pending’,’Fulfilled’,’Cancelled’)
2. **refund** has values [0,1]. 0=> false and 1=> true
3. **modifiedDate** on an **Order** is basically a particular date when either the **Product** was delivered or **Order** was cancelled. This then leads to the of question of whether to say every every **Customer** has a history where we capture the status of the **Order** along with dates? Like History[historyID,for[fk(oderID)],dateCreated, status], we can then log when the **Order** is ‘Pending’, and also log when it is ‘Fufilled’ and possibly ‘Cancelled’. As such, I can then remove dateModified on **Order** entity.
4. **\*\*** I have not yet accounted for the cost of the delivery on the amount. delivery cost should have a field\*\*

**DELIVERY SERVICE:**

1. **name** can be the primary key since all delivery services have different names
2. **Service** entity of **DeliverService** has a **description** field which basically just provides more information eg. **descrition**(‘You will receive your package within 2 days’) for maybe an express delivery , **descrition**(‘You will receive your package within a week’) for a regular delivery.
3. The **DeliveryService** and **Area** join will be called **AreaServed**.
4. **\*\* DeliveryService** delivers an **Order\*\*.**

**SUPPLIER:**

I can either assume that llamazon has their own **DeliveryService** which will then mean that a third type of **Supplier** can exist. The supplier who says that I want to sell a certain **Product** but I want llamazon to fulfill the **Order**. Which means llamazon will pick up the **Product**, process it and pack it in a **Warehouse** for delivery when ordered. Or assume that they don’t have their own **DeliveryService** and so it makes no sense to have a third type of Supplier.

1. Suppliers ship their inventory to fulfilment centers and these centers ship out orders whilst suppliers who prefer to deliver to customers directly, they do their own delivery.
2. \*\*I will assume these suppliers exist in the system: ~ Those that are **manufacturing companies** and **supply** your **warehouses** for you to sell. ~Those that **place** adverts on your platform and make their own **delivery.** ~There is also those that place adverts and say they want llamazon to fulfill their **Oders** for that **Product**.\*\*

**WAREHOUSE:**

1. Llamazon has fulfilment centers. Suppliers have different fulfilment options to choose from. They can choose whether they want to handle fulfilment or let llamazon sort, package and ship products through their own fulfilment centers.
2. Address of a Warehouse could be on the same table as the Warehuse entity and the same is applicable to other entities where a one-to-one relation exists but I can still choose to separate.

**RULES (for me only):**

1. A one-to-many relationships should be the norm in any database design
2. A one-to-one relationship should be rare, and they normally indicate that the two entities belong to the same table.

**Check this out ->**

“The retailer also leverages existing delivery routes via Fedex and UPS, too.”( <https://tinuiti.com/blog/amazon/amazon-supply-chain/>)

**REFERENCES**

1. <https://schema.org/>
2. <https://tinuiti.com/blog/amazon/amazon-supply-chain/>