

## Assumptions made

1. "*shop\_name*" is the key for the **shop** entity set since all the shops have unique names.
2. The *maker* needs to be recorded for a product. We assume there is only one maker for each product, and design it as an attribute.
3. Each **complaint** has a unique "*complaint\_id*" that can be used to identify complaints.
4. Users are not allowed to make a complaint about the product nor give a rating of the product unless they have made an order. Users are free to make complaints about any shops.
5. One order may contain many products which have different delivery dates.
6. Relevant timestamps are recorded upon the completion of the action it refers to.
7. An order may consist of different products from different shops. Thus, each product in the order may have different status (some shops have delivered while some have not).
8. A user can check the status of the complaints he has made using services backed by this database system.
9. A user makes an order of multiple products at one time instance. All the products in that orders are bought at the same timestamp.
10. Each **feedback** has class has a unique *feedback id* that can be used to identify it.
11. Each **complaint** is automatically assigned to an **employee** by Shiokee once it is generated by the user.
12. Each kind of **product** has unique *name*. For all iPhone 13 sold in all shops, their names will be the unified to a common one for all the products sold in shops. If there is duplication in name, specification is required to maintain this assumption (e.g. apple phone and apple fruit will be indicated with different names like 'apple(phone)' and 'apple(fruit)').

## Discussion outcome

**Product** has two subclass entity sets, namely **products in shop** and **products on order**. This is because these two share common attributes such as maker, category. However, they are essentially different as **product in shop** has unique attributes like quantity in stock and unique relationship to **price record** whereas **product on order** has unique attributes such as quantity in order, status and a relationship with **order**. Some of our group members propose using just one product entity set, but this alternative method cannot capture the unique attributes of the two subclasses mentioned above and may leave many unpleasant NULL values in implementation.

We choose to make **products in shop** a subclass of **product** and a weak entity set relating to the **shop**. Its key is  $\langle \text{product\_name}, \text{shop\_name} \rangle$ . Since each shop will get a shop-unique ID to each product it sells, we make *produce ID in shop* as an attribute of the product in shop. We assume each shop gives only 1 ID to an object of same name. If there are duplicate names, such as (apple as a fruit and apple phone), shop will need to specify the product name like 'apple(fruit)' and 'apple(phone)' to avoid such duplication in name. Hence, **product in shop** continue to use the superclass entity set key of *name*.

We choose to make **product on order** a subclass of **product** and a weak entity set relating to both the **shop** and **order**. Its key is  $\langle \text{product\_name}, \text{shop\_name}, \text{order\_id} \rangle$ . This is because a user may purchase an iPhoneX from shop A and an iPhoneX from shop B in the same order. Each **product on order** must belong to an order.

**Shop** does not have a generic relationship with the superclass entity set **product**. Instead, it has a fundamentally different relationship with its subclass entity sets. It **sells** a certain **product in shop** within a specific timeframe while it **supplies product on order** in an order. Thus, we do not create a relationship with the **product** superclass entity set directly. Instead, we design two relationship between **shop** and the two subclass entity sets of **product**.

**Price record** is a weak entity set of **product in shop** because we would like to keep track of all the changes in price and an attribute could not do the intended job. Some of our team members suggest using just an attribute. However, we cannot keep track of the all the changes in price in the history if we make only an attribute.


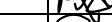
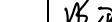
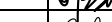

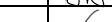
The **shop** can sell **product in shop** with a start date and an end date as it is of the shop's freedom to putaway and get items off the shelf at any time they want. If the shop is currently not selling that particular product in shop, the end date of *sell* relationship will not be NULL value, but the relationship will continue to be there to indicate that it used to sell that product for the purpose of tracing history of products sold on Shiokee.

Since the users can make complaints about both shops and products, we classify two subclass entity set from the general **complaint** superclass, **complaint about product** and **complaint about shop** respectively, both having unique relationship either to **product on order** or **shop**. This is also for easy tracing of the complaint target.

We decide not to include average rating and number of rating as attributes of **product in shop** as it can be computed.

We assume Shiokee does not allow a user to provide multiple feedbacks on one product in an order to avoid manipulation of rating score. A **feedback** must have one corresponding **product in order**, and a **product in order** can have one **feedback**, thus forming relationship indicator (referential integrity on the **product in order** side, and an arrow on the **feedback** side).

## APPENDIX C: INDIVIDUAL CONTRIBUTION FORM

Name	Individual Contribution to Submission 1 (Lab 1)	Percentage of Contribution	Signature
cheng zhengxing	join group discussion, suggest possible improvements, modify the details of ER graphics	16.66%	
zhang tianyu	join group discussion, suggest possible improvements, modify the details of ER graphics	16.66%	
zhou runbing	join group discussion, suggest possible improvements, modify the details of ER graphics	16.66%	
chirstopher arif setiadharma	double-checking the relationships and entity attributes	16.66%	
an ruyi	design the ER relation, create skeleton of design, create ER diagram, write report, join group discussion	16.66%	
peng wenxuan	design the ER relation, create skeleton of design, create ER diagram, write report, join group discussion	16.66%	

[illegible][illegible]