Databases: Modelling Exercise

ASOS

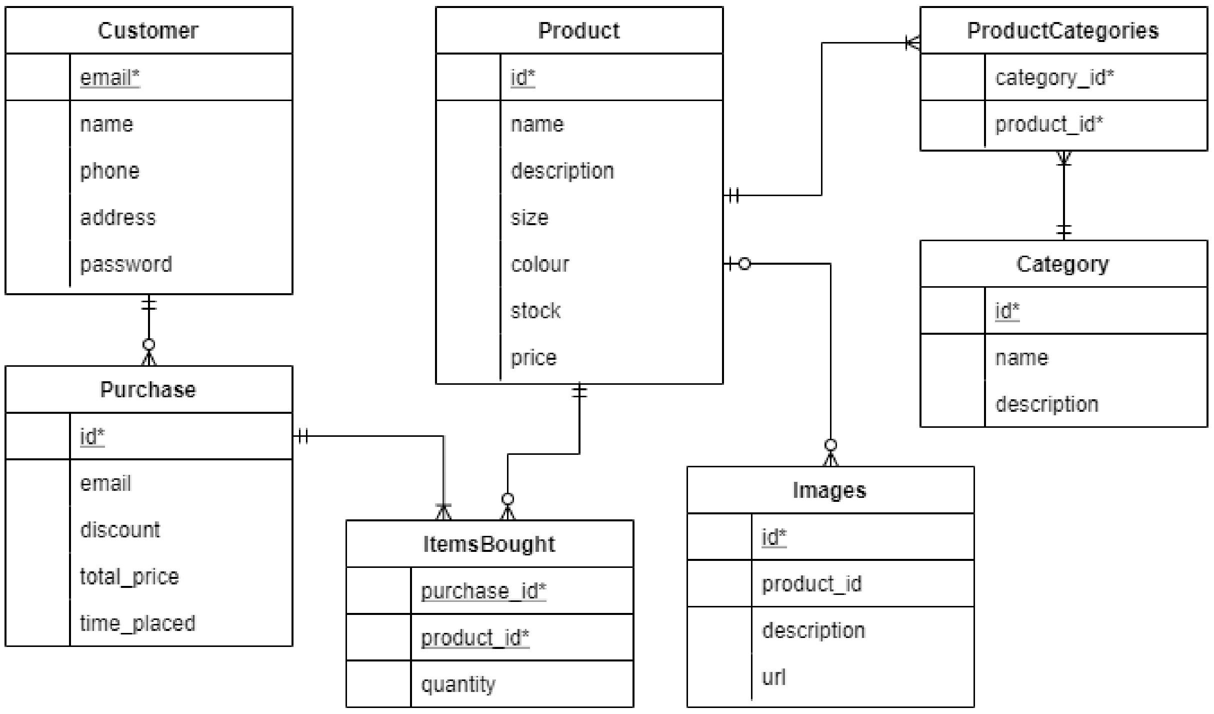
The website I have chosen to model is ASOS, which is an online clothing retailer on which users can browse clothes by categories (e.g. “newest”, “men”, “shirts”) or search them by name, then filtering by size, colour and/or price, with all these details shown on every product page along with an optional number of images of the product (optional 0 – many images).

Users can create accounts with an email and password, providing a name, address, and phone number (which are all required). Through their account, users can see the items they’ve bought through their order history.

The main entities I have modelled are the customer, product, category, and images, with a relational joining table allowing customers to purchase products, and another allowing products to belong to more than one category. The ProductCategories joining table can take one or more products being added to one or more categories, meaning that every product must belong to at least one category, and every category must have at least one product stored within it. The way the purchasing system is modelled allows a customer to purchase multiple different products through one purchase id, instead of a single relational purchases table between product and customer which would only allow one product to be bought at a time. This also follows a one-to-‘at least one’ relationship because a purchase_id must exist in at least one entry in the ItemsBought table to exist, but product_id doesn’t follow this and instead can have any number of entries (one-to-any number) in the ItemsBought table.

The images table doesn’t require a relational table because each product can have many images, but each image can only relate to a single product. Therefore, the table simply needs a foreign key displaying the product id that it belongs to. Product_id is also optional in this table because ASOS likely has a large database of images, not all of which are currently being used for products.

Modelling these as shown below gives the user the ability to create an account using their email address as their unique id, and by giving a password (which would be encrypted when stored in the database). They can then browse products by category or name, then adding them to their order and checking out, which produces a unique purchase id (which relates to a number of items in the ItemsBought table) detailing the total price including any discount, and the time the order was placed (this would likely feed in to another theoretical table which deals with deliveries).



A few different examples of user cases are:

1) Viewing all the items ordered by a certain user in the past 6 months

```
SELECT  Pur.email, Pur.id, Pro.name, Pur.time_placed FROM Purchase Pur
JOIN ItemsBought ON Pur.id=ItemsBought.purchase_id
JOIN Product Pro ON ItemsBought.product_id=Pro.id
WHERE Pur.email = '?' AND Pur.time_placed >= DATE_SUB(now(), INTERVAL 6 MONTH);
```

? = The email of the user

2) Browsing all men's medium white t-shirts that are in stock, sorting by price

```
SELECT Pro.id, Pro.name FROM Product Pro
JOIN ProductCategories PC ON PC.product_id=Pro.id
JOIN Category Cat ON Cat.id=PC.category_id
WHERE Cat.name = 'men' AND Cat.name='t-shirt'
AND size='medium' AND stock > 0
ORDER BY price ASC;
```

3) Selecting all the images of orange shirts to display on the website (this user is the person creating the website)

```
SELECT Im.url FROM Images Im
JOIN Product Pro ON Im.product_id=Pro.id
JOIN ProductCategories PC ON PC.product_id=Pro.id
JOIN Category Cat ON Cat.id=PC.category_id
WHERE Cat.name LIKE '%shirt%'
AND Pro.colour='orange';
```