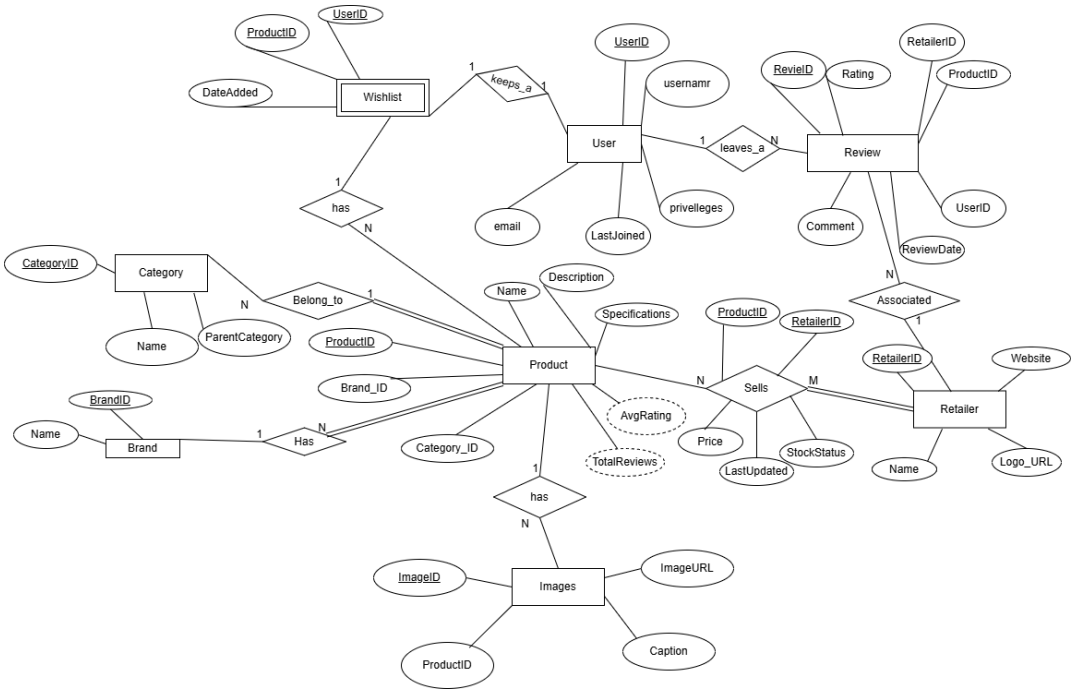
Macrohard: https://github.com/u24739163/Macrohard

Table of Contents

*** The research document(Task 1) is on the github repository.**

(E)ER-Diagram(Task 2):

The first ever diagram:



From here we added the relationship left_on from review to product to 1 to n where one product can have many reviews.

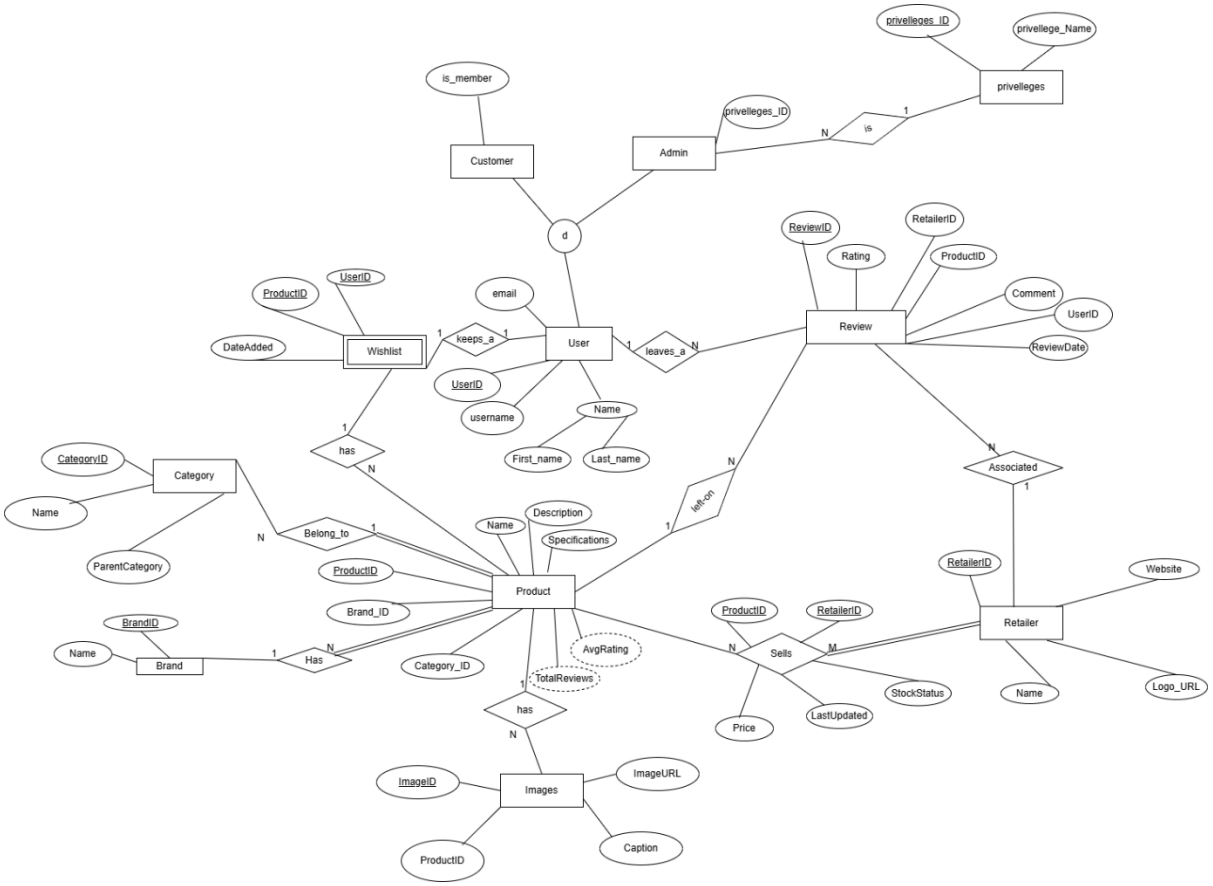
We also added composite attribute consisting of First_name and Last_name to User.

We also added a disjoint to User to differentiate between an admin and a Customer.

And Customer was given the attribute is_member.

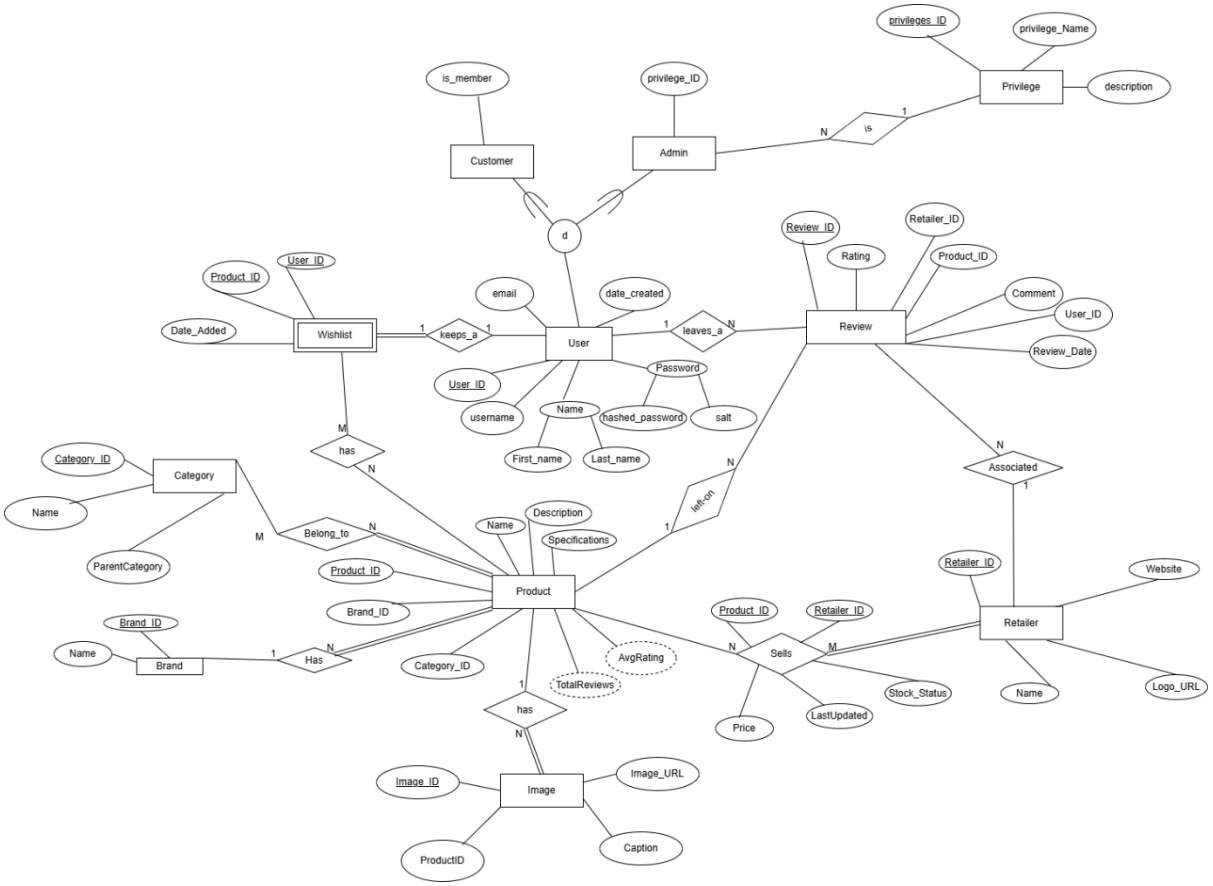
The admin is given the is attribute which connects to privileges and says what they can do.

The transition ER-Diagram



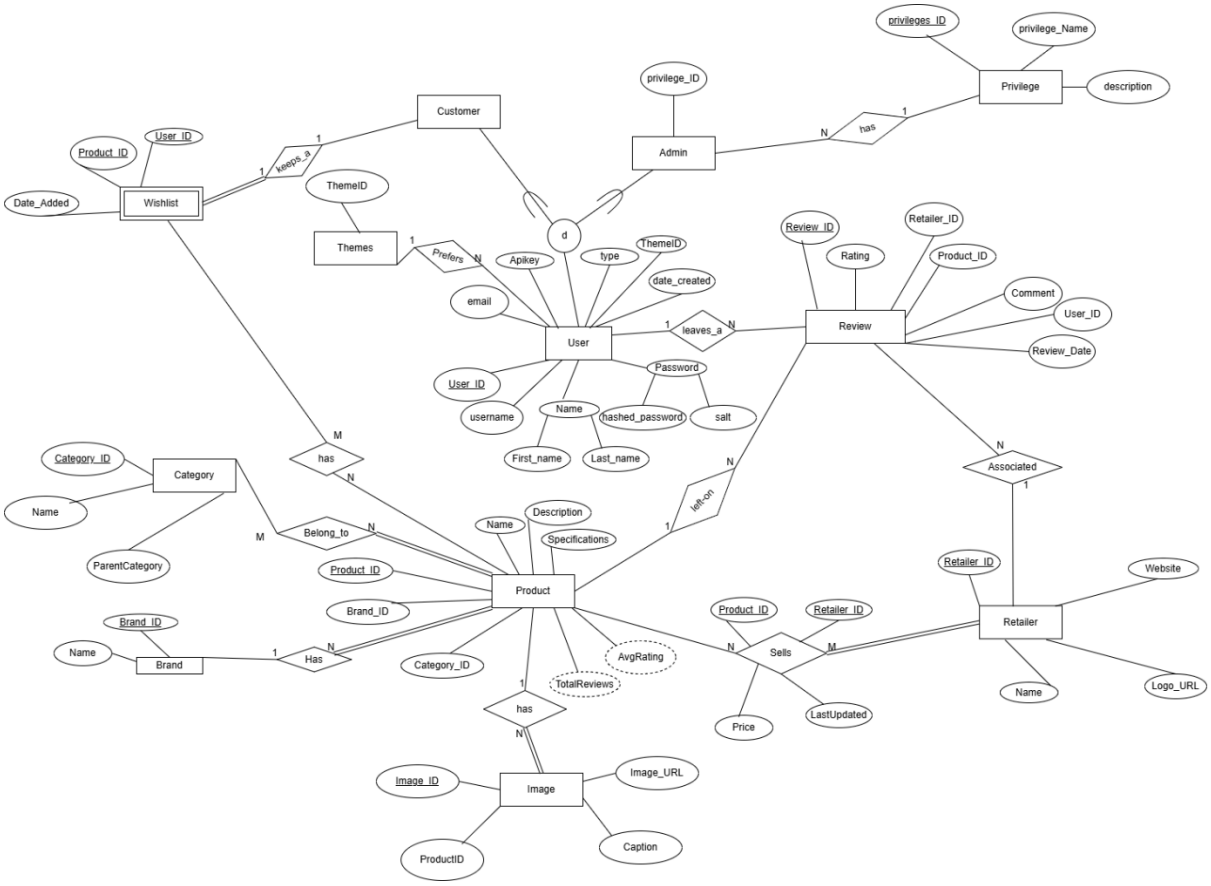
Added composite attribute Password to User consists of hashed_password and salt and added attribute date_created to User and added arrows to the disjoint.

The transition ER-Diagram



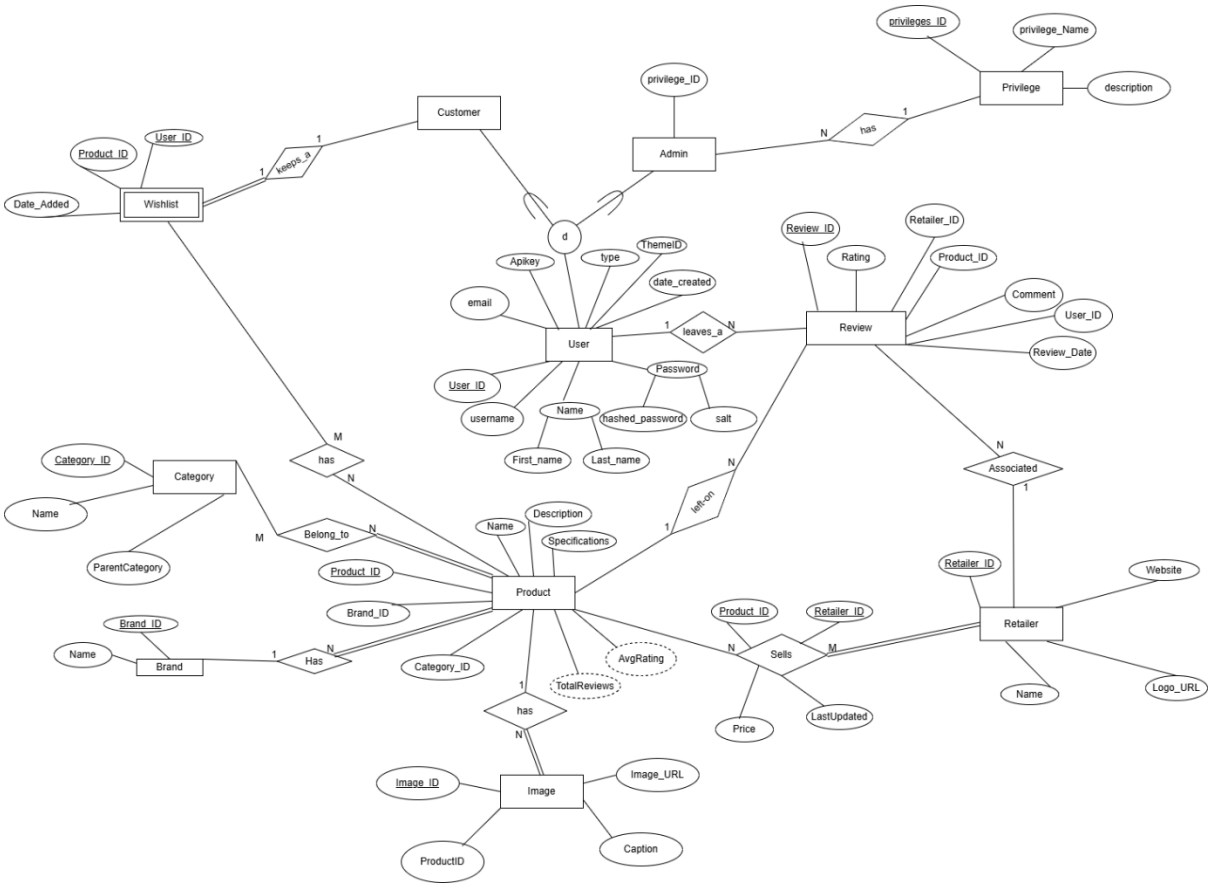
Moved the relationship (keeps_a)from User to Customer and removed is_member attribute from Customer.

The transition ER-Diagram



We removed the Themes table.

The Final ER Diagram



We made the assumption that the Admin cannot have a wishlist and that they would need to make a Customer account and therefore Customer is the only user that can have a wishlist and a wishlist cannot exist without the Customer thus the Wishlist is a weak entity because it depends on the UserID.

The derived attributes AvgRating and TotalReviews is assumed to able calculatable by using aggregate functions on the Review entity.

(E)ER-diagram to Relational Mapping(Task 3):

Step 1: Strong Attributes

User

User_ID	First_name	Last_name	Username	Email	Hashed_Password	Salt	Date_created	Apikey	Type

Review

Review_ID	Rating	Retailer_ID	Product_ID	Comment	User_ID	Review_Date

Retailer

Retailer_ID	Name	Website	Logo_URL

Product

Product_ID	Brand_ID	Category_ID	Name	Description	Specifications	Review_Date

Images

Image_ID	Product_ID	Image_URL	Caption

Category

Category_ID	Name	Parent_Category

Brand

Brand_ID	Name

Privileges

Privilege_ID	Privilege_Name	Description

Mapped all of the Strong Attributes

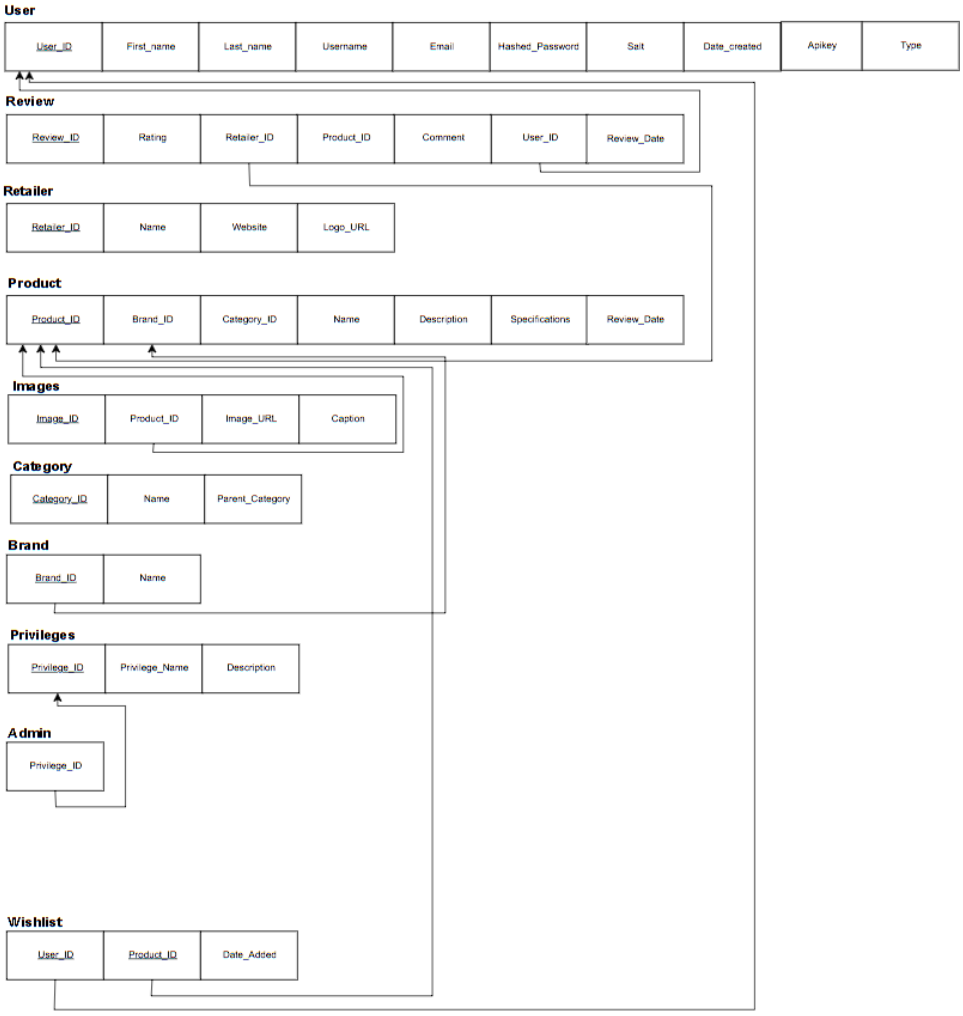
Step 2: Weak Attribute: Wishlist

Mapped the only weak entity Wishlist. Added as Foreign key the User_ID and Product_ID of it's owner entities.

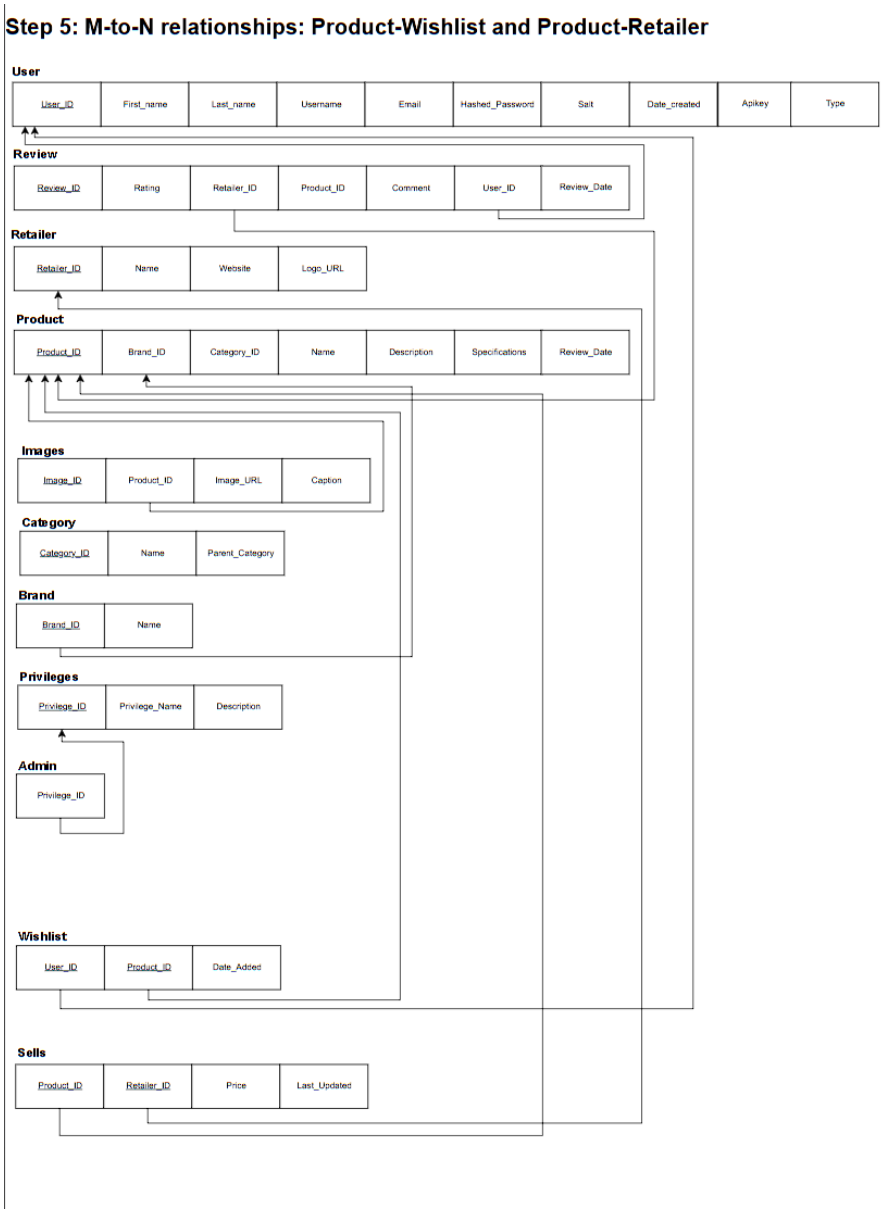
Step 3: 1-to-1 relationships: User-Wishlist

Mapped the 1-to-1 relationship between Wishlist and Customer. Added to Wishlist the Foreign primary key of User, User-ID which was already added in Step 2.

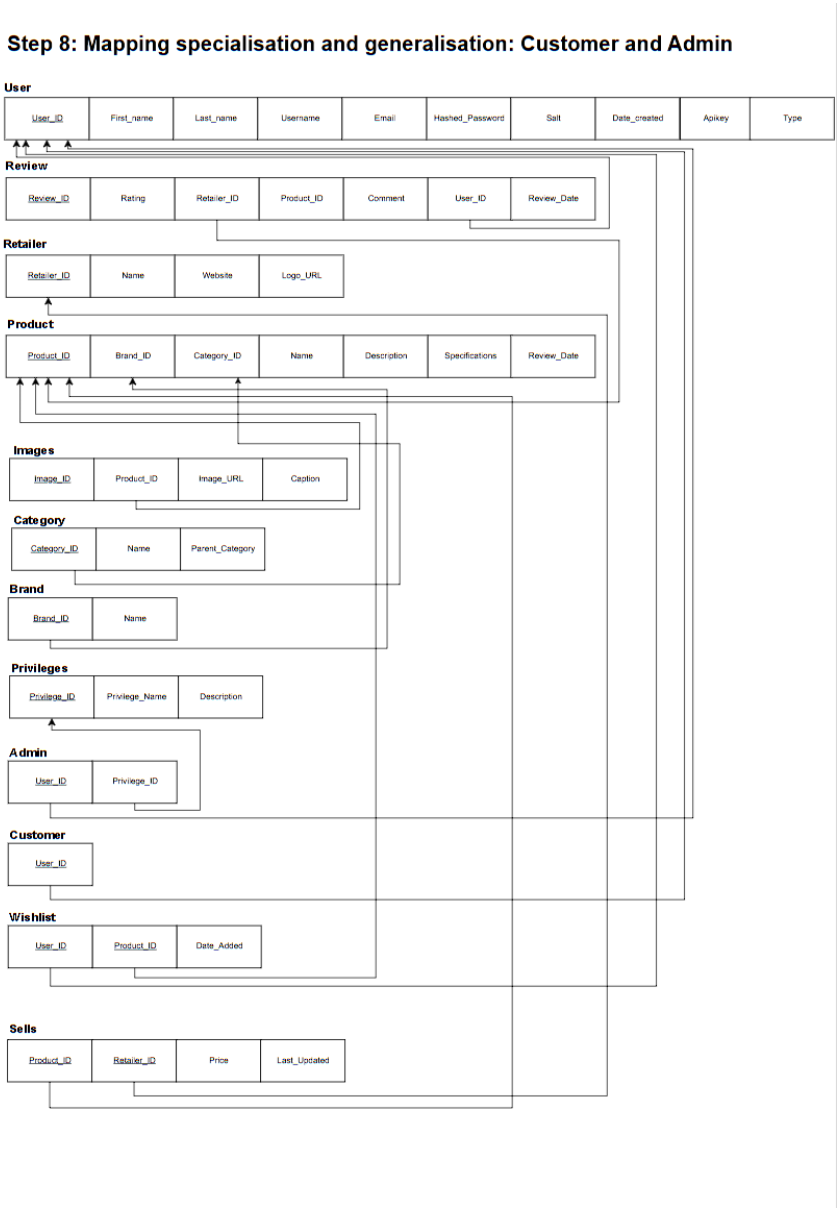
Step 4: 1-to-N relationships: User-Review, Product-Review, Product-Images, Product-Brand and Admin-Privilleges



Mapped the 1-to-N relationships. Added into the entities Review, Review, Images, Product and Admin as Foreign keys the primary keys of User, Product, Product, Brand and Privileges respectively.



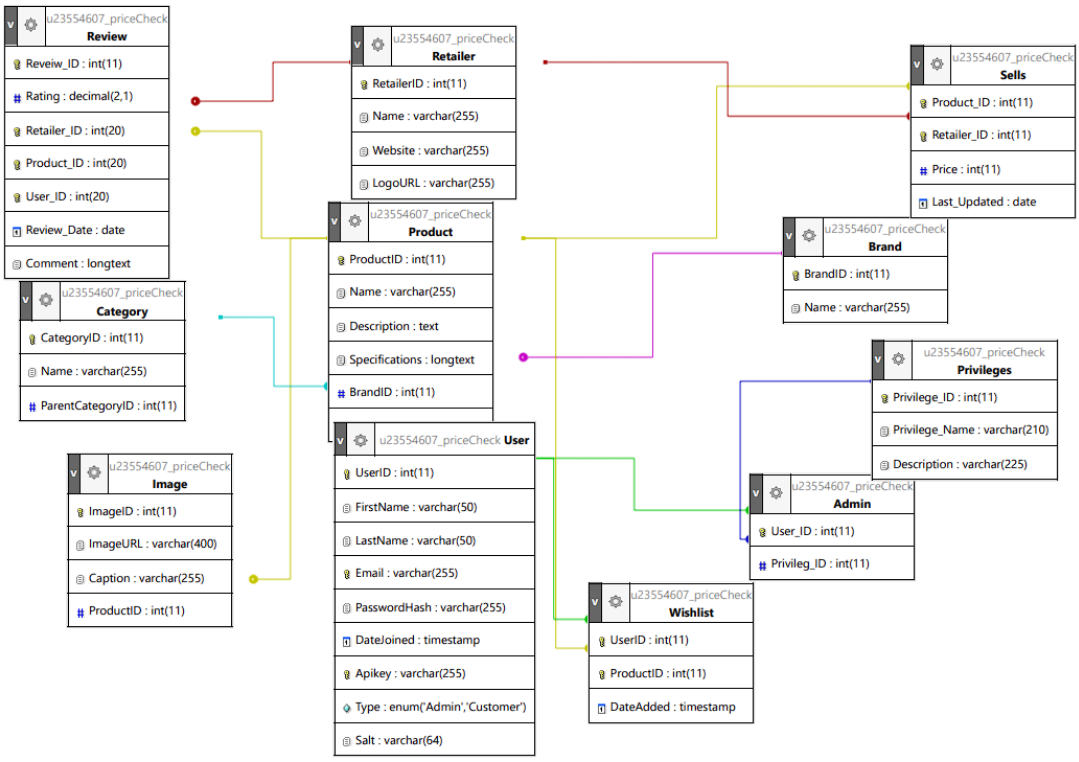
Mapped M-to-N relationships. Since Wishlist already has as Foreign keys the Primary keys of User and Product a new identifying entity would be redundant in this case. With Product-Retailer a new entity Sells was made which as Foreign keys has the Primary keys of both entities and both relational attributes.



Mapping specialisation and generalisation. Customer and Admin are subclasses of User with Admin having a privilege_ID unique to it.

Note: Steps 6, 7 and 9 were excluded since our ER-diagram did not include Multivalued attributes, N-ary relationships or Unions.

Relational Schema(Task 4):



Data Source (Task 6):

Since we decided to make our website mainly a tech price comparison website, we looked at prices from websites such as Evetech and Amazon to get a general idea on pricing and then created our own data and prices. We decided to use MyPHPAdmin as the tool to populate the database using SQL scripts and manually adjusting certain tables as it allowed for easy database scaling.

Analysing And Optimizing A Query(Task 7):

Using myPHPAdmin to explain the following query:

```
SELECT P.`ProductID`,P.`Name`,P.`Description`,
    P.`Specifications`,B.Name AS Brand,C.Name AS Category
    FROM Category AS C JOIN( Product AS P JOIN Brand AS B ON
    P.BrandID=B.BrandID) ON C.CategoryID=P.CategoryID
     WHERE `P`.`ProductID`=?
```

And all the operations of type const which is the best and are all select_type SIMPLE

And the next query:

SELECT `ImageURL`, `Caption` FROM `Image` WHERE `ProductID`=?

select_type is SIMPLE and the type is ref so it uses an index

And the next query:

SELECT R.RetailerID AS RID,R.Name AS Retailer ,R.Website AS link, R.LogoURL AS logo, `Price` FROM `Sells` AS S JOIN Retailer AS R ON `RetailerID`=`Retailer_ID` WHERE `Product_ID`=?

There are only two rows accessed and both are SIMPLE the first one is type ref and the second is a join that is type eq_ref which is the best for join

and the next query in the plan is:

SELECT ROUND(AVG(R.`Rating`), 1) AS Star FROM Review AS S JOIN (SELECT `Rating`,`Product_ID` FROM Review WHERE `Retailer_ID`=?) AS R ON R.`Product_ID`=S.`Product_ID` WHERE S.`Product_ID`=?;

The has two operations the first has a type index_merge and the second one has is type ref but I optimised the query to be:

SELECT ROUND(AVG(`Rating`), 1) AS Star FROM Review WHERE `Retailer_ID` = ? AND `Product_ID` = ?

And there was a performance gain as the number of operation went from 2 to 1 eliminating the second operation in the first query.

the last query is:

SELECT `Rating` AS STARS,User.`FirstName`, `Review_Date`,

`Comment` FROM `Review` JOIN `User` ON `User_ID`=`UserID` WHERE `Product_ID`=?

There are two operation making up this query they are both SIMPLE the first is in table Review and is a ref type which is quite optimised but the number of estimated rows is five which is smaller than what the actual number would be and the second operation is in table User and is of eq_ref and the PRIMARY key is used as the key and the rows are already optimised

To optimise add a composite index in table review but there was no gain at least not to a noticeable degree therefore the initial query is optimised

Member Contributions:

Arran Lamond u23655675 – Research document, Admin side of the Webpage, input on customer side of the webpage and completed the final PDF.

Michael Neto u24641342 – Frontend Designer, database population, GitHub branch manager and worked on the Admin side of the Webpage.

Keegan Lewis u24594522 – EER diagram, database finalisation, API, optimisation and slight database population.

Vele Ndamulelo u23554607 – original database designer, Wishlist implementation and the presentation slides creator.

Mulondi Makhado u24739163 – Relational Mapping, Products development, Implementation of Login, Registration and Homepage pages and setup the GitHub repository.

Tshepiso Makelana u23588579 – Frontend Designer, View page developer, README file and EER diagram input.

Bonus Marks:

Security:

We made our website more secure by using better hashing of passwords with sha512 to hash and then we salted the passwords. A salt is a unique string that is added to a password before it is hashed meaning that even if a password is used by multiple people, that password won't necessarily be exposed due to the salt making the hashing of each individual password different.