1) Please list out changes in your project directions if the final project is different from your original proposal (based on your stage 1 proposal submission).

Originally we were planning to scrap the product and price data from various major retailers' websites. However most of the major retailers like Walmart or Target don't have API support, and web scraping programs can take a while to write. Luckily we found a dataset of different products scrapped from Walmart, which contains everything we needed, so we used that for our initial data. Other than that, the general direction of our program stays the same.

2) Discuss what you think your application achieved or failed to achieve regarding its usefulness.

We developed a solid system that helps users compare prices and search or filter for products. There is also functionality that further helps the system with its data by letting users update and insert price and products. All the api on the backend and stored procedures on the SQL server really helps simplify things for further usage and development.

However, the original intent was to help a user decide which retailer offers the best price. We couldn't achieve this since we only got the Walmart dataset and there is not enough time for us to develop a web crawler for other websites. With that said, if a web crawler is developed in the future, having it insert or update our database is simple since we already implemented all the support for simple insert and update.

3) Discuss if you change the schema or source of the data for your application

We changes the followings about our schema:

Removal of entity Chain (a Chain can have a lot of Retailers, like Walmart or Target has a lot of shops at different locations): we decided to remove this entity since there is not enough data for it. Getting information about different locations' stock on Walmart websites or Target websites is hard and most of them offer the same price across all of their shops, so there is no point in separating different shops of one chain. Thus we removed it to simplify the project.

Retailer: Removal of retailerAddress because there is no data available for it.

Brand: Removal of productCount. We thought that this would be useful, however it is really excessive and can be found just by a simple query, which is not worth implementing all the triggers to update productCount everytime we insert/delete.

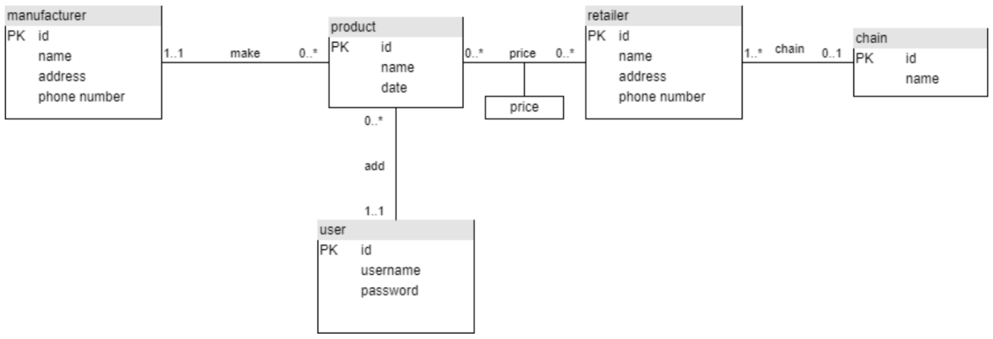
Product: We added in AUTO_INCREMENT property for productId, which saved us all the hassle of trying to generate a unique id.

Price: Added in another column username, which tells us which user inserts/updates the price.

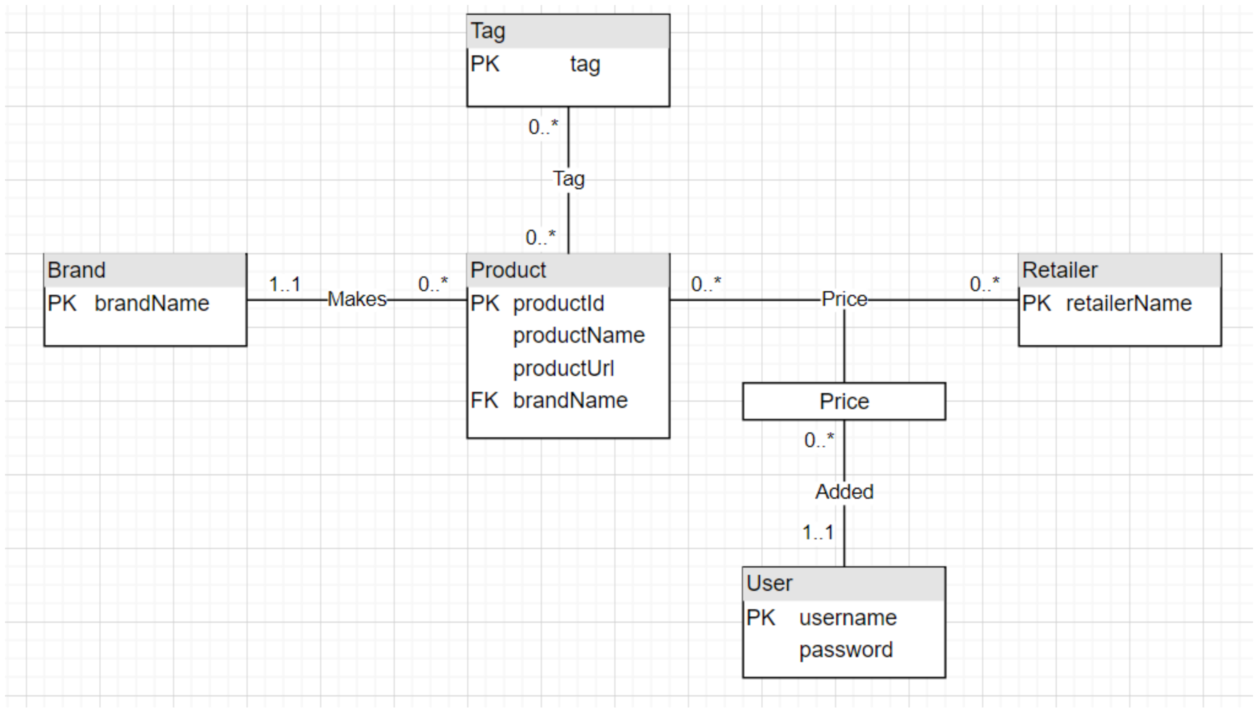
Added a whole new entity Tag, which has a many-many relation with Product. This will help us filter out products by tags, which we think is a really cool thing to have.

The source of data for our project is still the same, from a dataset of Walmart Product Data scrapped in 2019.

4) Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?



Proposed UML diagram



Final UML diagram

We added the new entity Tag, which forms a many-many relationship with Product. This helps the user of our programs because now they can filter out stuff they want. We also changed the User-Product relation into User-Price relation, since this is more accurate to the use case of our project because in reality, the user adds or updates the price of a product and it will create the product if it did not exist, not adds or updates a product. So, the user actually interacts with the Price and not the Product. We also removed the Chain entity since most Chain offers the same price across all of their stores, so there is no point in differentiating different Retailers of the same Chain. We also removed some unnecessary details like phone number, date and address. I think this is better suited for our application. Even though logically, the more details the better, it will flood our users' limited screen view with hundreds of unnecessary details, and make it harder for users to insert new products since they have to fill out phone numbers, addresses, date, etc. Thus, we think the new UML hits the perfect balance between having enough information and being convenient for the user, thus better suited for our application.

5) Discuss what functionalities you added or removed. Why?

To utilize the new Tag entity, we implemented Advanced Search feature, which features a UI that supports searching products name and filter for specific tags. We also display tags and who updated/inserted the price using the username field in our Price table.

6) Explain how you think your advanced database programs complement your application.

Having an advanced database really simplifies the process of storing, handling, reading and editing data. You can just simply call the SQL query and the database system will do the rest. It is really hard to imagine how we could implement the same application again without using a database system. The database system already takes care of all the nitty-gritty of data and helps us focus more on the actual functionality and user interaction of our application.

7) Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.

Ankushi: I was having trouble trying to generate unique ids for the Products when inserting a product. Doing COUNT(*) + 1 wouldn't work after some products are deleted and iterating through the id to find which one is unique is way too time consuming. Then after searching around, I realized I can just use AUTO_INCREMENT for productId when I declare the table. It is clean, saves a lot of time and convenient for both stored procedures and backend querying.

Everett: In some of our implemented functions, there are sequential queries that need to be executed in order. At first, I put all the queries one after another, but they kept running in parallel. Then I realized that in these requests they use a callbacks function that is called after an answer is recieved, thus if I want to execute a query after another, I have to put the second query inside the callback function of the first query, so only when the first query got an answer then the second query is executed. Thus I have all of them nested together. The code might not look clean but it works.

Khanh: When I tried to connect the backend with GCP's mySQL server, all the guides and documentation were telling me to connect using the UNIX path or download a client and I couldn't get them to connect. It took me way too long just to figure out that I could just use the IP address written on GCP of the SQL server, and the connection works flawlessly. All you need is the ip address, user and password.

Jie: For the advanced search in the frontend where we can filter stuff by tags, I find it really hard to create all the checkboxes and read inputs from them. Often, when the checkbox is clicked, a callback function is called to update the user input, however I couldn't pass the data to distinguish between the different checkboxes so that I can update the right variable. Then I found out about factories, which really makes reading from bunches of textboxes way easier.

8) Are there other things that changed comparing the final application with the original proposal?

All of the significant changes are already mentioned. The UI is different from originally proposed due to it being easier to implement. Also, since there is not much web scraping needed since we already have a dataset, the work distribution is shifted around. Also we decided to divide work by frontend, backend, SQL sections instead of individual functionality since one person can just focus on one platform only.

9) Describe future work that you think, other than the interface, that the application can improve on

Currently, anyone can delete a product or insert/update stuff that will directly affect the database. So if someone were to use the UI to delete all Products from the database or insert nonexistent products, we would have no way to protect against it. So we want to implement a feature so that a change to the database can only be executed if it is verified by an admin. We can have a table of pending modifications, and the admin can go through each entry and accept or reject the modification.

10) Describe the final division of labor and how well you managed teamwork.

Ankushi: Worked on SQL server side: Wrote the table schemas and the triggers. Imported the initial data and wrote triggers to insert the raw data into our table schemas. Wrote procedures to make insert/update easier to do on the backend.

Everett: Worked on the backend: Implemented all the available functionality to answer requests from the frontend. Wrote queries to send to mySQL and wrote the advanced queries.

Khanh: Put everything together. Established the frontend-backend connection and backend-mySQL connection. Debugged and solved errors arise. Wrote the SQL procedures that handle search with tags. Wrote dockerfiles to deploy everything on GCP.

Jie: Worked on the frontend: Designed the UI, the login feature and all the functionalities on the frontend to handle the user's input to send to the backend. Displayed results, feedback and error messages to users upon receiving the results from the backend.

11) Points we are trying to get back:

Stage 1 fixes:

- Renamed Readme.md

Stage 2 fixes:

- Removed translation of Make, Add, Chain (in CS411 PT1 S2 - UML Design (1).pdf)
- Added foreign key to Product and Retailer (in CS411 PT1 S2 - UML Design (1).pdf)
- Fixed TeamInfo.md

Stage 3 Fixes:

- Fixed Database Design file name
- Added screenshot of database (in Database Design.pdf)
- Changed query 2 which has subquery (in Database Design.pdf)
- Added Index Analyses for Query 1 and 2 (in Database Design.pdf)