CS 121 Final Project Reflection

Due: March 18th, 11:59PM PST

This reflection document will include written portions of the Final Project. Submit this as **`reflection.pdf`** with the rest of the required files for your submission on CodePost.

For ease, we have highlighted any parts which require answers in **blue**.

Student name(s): Eshani Patel, Rachael Kim

Student email(s): ejpatel@caltech.edu, subinkim@caltech.edu

Part L0. Introduction

Answer the following questions to introduce us to your database and application; you may pull anything relevant from your Project Proposal and/or README.

Database/Application Overview

What application did you design and implement? What was the motivation for your application? What was the dataset and rationale behind finding your dataset?

Database and Application Overview Answer (3-4 sentences) :

We created an cosmetics e-commerce application (similar to Sephora or Ulta) powered by a database of products, brands, users, etc. Users will be able to filter products based on product type, brand, ingredients, rating, skin preference, and price. They can also create and login to an account to purchase them by adding the products to the cart and checking out and storing their purchase history information. They can compare products and see what their best option is given such filters.

The motivation for our application was to create a user-friendly app to shop for clean beauty products, which was building off the project one of us built in CS 132 (a clean beauty website). With this application, it provides a comprehensive backend and ability to integrate actual product and brand data for users to interact with. We wanted to allow a comprehensive and intuitive way for consumers to interact with an ecommerce platform, and since we both enjoy beauty/cosmetics products, we want to create a beauty store. Specifically, we allow users to filter products based on their ingredients, so that any users who may be allergic to certain ingredients or pursue vegan products are able to easily filter out products and narrow down their search results.

Our rationale for finding our dataset was to look for one that contains a lot of data regarding products and brands that could be formulated into a cosmetics store. Thus the kaggle set was an ideal base for our starting data.

Data set (general or specific) Answer:

Cosmetics dataset (https://www.kaggle.com/datasets/kingabzpro/cosmetics-datasets), which contains label (type of the product, e.g. moisturizer), brand, name, price, ranking (rating of the product on Sephora), ingredients, and skin preference (Combination, Dry, Oily, Normal).

***Client user(s) Answer*:**

The intended clients are beauty/cosmetics/skincare purchasers and consumers. This is all people for example who shop at online stores like Sephora or Ulta. Users may create and login to an account, add items to their shopping cart and browse items, purchase items and view their purchase history, but they may not modify anything other than their shopping cart and their user information. We have assigned privileges for client users to execute our UDFs/procedures/triggers but not privileges to make any direct modifications to the database.

Admin user(s) Answer:

Intended admins are beauty stores, such as Sephora and Ulta, or that want to launch an online platform to sell their beauty products. The beauty store themselves can have admin access, such that they can update their inventory of products, change features such as price, add new products/brands, and delete old products/brands. They are also able to analyze the performance of the store through various statistical measures that are provided through the admin application. Example statistics include: total amount of sale for the store, per brand, or per product type; best-selling products or brands; most recent sales per brand or per product type, and more.

Part A. ER Diagrams

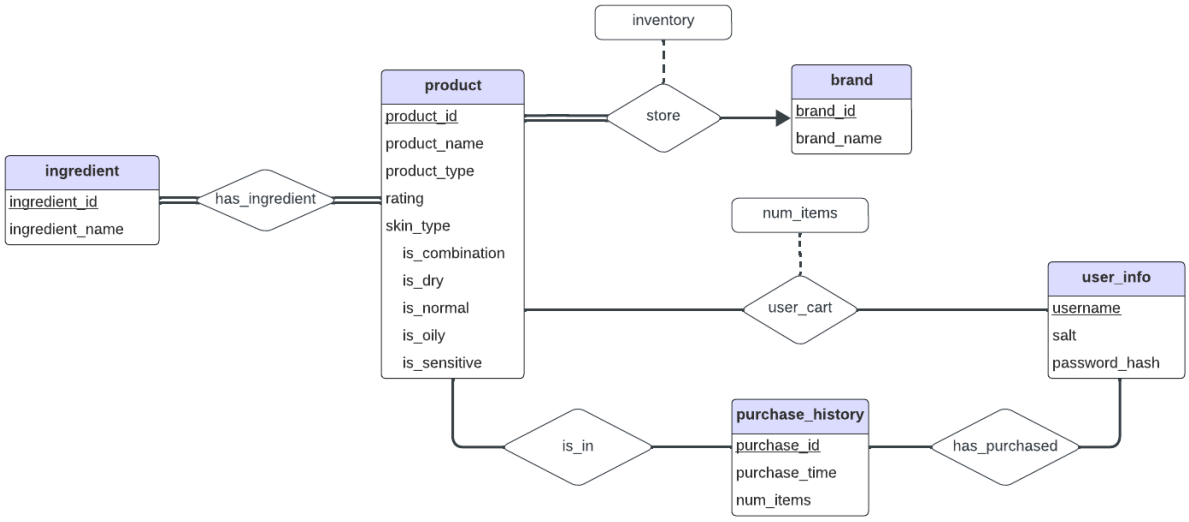
As we've practiced these past few weeks, the ER model is essential for designing your database application, and we expect you to iterate upon your design as you work through the ER and implementation steps. In this answer, you should provide a full ER diagram of your system. Your grade will be based on correct representation of the ER model as well as readability, consistency, and organization.

Notes: For this section **only**, we will allow (and encourage) students to share their diagrams on Discord (**#er-diagram-feedback**) to get feedback from other students on their ER diagrams given a brief summary of your dataset and domain requirements. This is offered as an opportunity to test your ER diagrams for accuracy and robustness, as another pair of eyes can sometimes catch constraints that are not satisfied or which are inconsistent with your specified domain requirements.

Requirements:

- Entity sets, relationship sets, and weak entity sets should be properly represented (also, do not use ER symbols not taught in class)
- Mapping cardinalities should be appropriate for your database schema, and in sync with your DDL
- Participation constraints should be appropriate and in sync with your DDL (total, partial, numeric)
- Use specialization where appropriate (e.g. *purchasers* and *travelers* inheriting from a *customers* specialization in A6)
- Do not use degrees greater than 3 in your relationships, do not use more than one arrow in ternary relationships.
- Use descriptive attributes appropriately
- Underline primary keys and dotted-underline discriminators
- Expectations from A6 still apply here
- Note: You do not need ER diagrams for views

ER Diagrams:



*** Note, we went through multiple iterations of our ER diagram, and got feedback on Discord and in OH.**

Part B. DDL (Indexes)

As mentioned in Part B, you will need to add at least one index at the bottom of your **`setup.sql`** and show that it makes a performance benefit for some query(s).

Here, describe your process for choosing your index(es) and show that it is used by at least one query, which speeds up the performance of the same query on a version of the same table without that index. You may find lec14-analysis.sql and Lecture 14 slides on indexes useful for strategies to choose and test your indexes. **Remember that indexes are already created in MySQL for PKs and FKs, so you should not be recreating these.**

Index(es):

We perform most of our queries using primary keys (such as unique ids associated with brand, product, purchase_history, etc) as our web interface allows us to easily assign primary keys as ids and utilize them in our queries.

We created an index on product type as it is a field that is not a primary key but is used commonly in our SELECT queries.

This was our process:

CREATE INDEX index_ptype ON product (product_type);

SET PROFILING=1

SELECT product_name, product_type, price FROM product WHERE product_type = 'Moisturizer';

DROP INDEX index_ptype ON product;

SELECT product_name, product_type, price FROM product WHERE product_type = 'Moisturizer';

SHOW PROFILES;

Justification and Performance Testing Results:

Creating an index on this helped improve our queries by 0.000414 seconds (first one is with index, third is one without index). Thus, for fields like product_type (price, rating, etc) which are commonly used in our SELECT queries to filter products could improve a little bit in performance by creating an index on them.

We believe that our index will further improve the performance of our application as the product table gets bigger with the growth of the store. Currently, our product table has about 1472 rows from the pre-loaded data. If our application grows to industry-scale (for instance, several tens of thousands of rows), we believe that the improvement in the performance from the index will be more significant.

```
mysql> SHOW PROFILES;
+----------+------------+-----------------------------------------------------------------------------------------------+
| Query_ID | Duration   | Query                                                                                         |
+----------+------------+-----------------------------------------------------------------------------------------------+
|        1 | 0.00151400 | SELECT product_name, product_type, price FROM product WHERE product_type = 'Moisturizer' |
|        2 | 0.02327900 | DROP INDEX index_ptype ON product                                                             |
|        3 | 0.00192800 | SELECT product_name, product_type, price FROM product WHERE product_type = 'Moisturizer' |
+----------+------------+-----------------------------------------------------------------------------------------------+
3 rows in set, 1 warning (0.00 sec)
```

Part C. Functional Dependencies and Normal Forms

Requirements (from Final Project specification):

- Identify *at least 2 non-trivial functional dependencies* in your database
- Choose and justify your decision for the normal form(s) used in your database for at least 4 tables (if you have more, we will not require extra work, but will be more lenient with small errors). BCNF and 3NF will be the more common NF's expected, 4NF is also fine (but not 1NF).
    - Your justification will be strengthened with a discussion of your dataset breakdown, which we expect you to run into trade-offs of redundancy and performance.
- For two of your relations having at least 3 attributes (each) and at least one functional dependency, prove that they are in your chosen NF, using similar methods from A7.
    - If you have identified functional dependencies which are not preserved under a BCNF decomposition, this is fine
- Expectations from A7 still apply here.

Functional Dependencies:

Normal Forms Used (with Justifications):

NF Proof 1:

NF Proof 2:

Part G. Relational Algebra

Requirements (from Final Project specification, Part G):

- Minimum of 3 non-trivial queries (e.g. no queries simply in the form **`SELECT <x> FROM <y>`**)
- At least 1 group by with aggregation
- At least 3 joins (across a minimum of 2 queries)
- At least 1 update, insert, and/or delete
    - This may be equivalent to said SQL statements elsewhere (e.g. queries or procedural code), but are not required to be; in other words, you can write these independent of other sections
- Appropriate projection/extended projection use
- Computed attributes should be renamed appropriately
- Part of your grade will come from overall demonstration of relational algebra in the context of your schemas; obviously minimal effort will be ineligible for full credit; it is difficult to formally define "obviously minimal", but refer to A1 and the midterm for examples of what we're looking for
- Above each query, briefly describe what it is computing; we will use this to grade for correctness based on what the query is supposed to compute; lack of descriptions will result in deductions, since we have no idea otherwise of what the query is intended to do.

Below, provide each of your RA queries following their respective description.

Relational Algebra 1

- Includes aggregation (SUM)
- Includes a JOIN
- The query is designed to compute the total inventory value (which is calculated by summing up the inventory * the price of each item under the brand) for each brand.
- This query is in our queries.sql and has been used in our admin application.

Query:

```
SELECT brand_name, SUM(price * inventory)
AS total_inventory_value FROM brand NATURAL JOIN store
NATURAL JOIN product GROUP BY brand_name;
```

Relational Algebra:

$$\Pi_{\text{brand\_name, total\_inventory\_value}}(\text{brand\_name } \mathcal{G}_{\text{SUM(price * inventory) AS total\_inventory\_value}}(\text{brand} \bowtie \text{store}))$$

Relational Algebra 2

- UPDATE query
- When a user tries to add one more item of the already-existing product to their cart, this query will be run. This adds 1 to num_items for the corresponding product in cart, thereby updating the user's cart.
- This query comes from a procedure we have written for our client application.

Query:

Assuming we are given p_product_id and p_username,

```
UPDATE cart
   SET num_items = num_items + 1
   WHERE product_id = p_product_id AND
   username = p_username;
```

Relational Algebra:

$$\text{cart} \leftarrow \Pi_{\text{username, product\_id, (num\_items + 1)}}(\sigma_{\text{username} = p\text{\_username} \wedge \text{product\_id} = p\text{-product\_id}}(\text{cart}))$$

Relational Algebra 3

- Includes JOINs
- The query returns product_id, brand_name, and product_name of products that contain ingredients that have “beta” in their names (e.g. beta-carotene)
- This query, with the regex as a variable, is used in our client application as one of the filter functionalities
- This query is also listed in our queries.sql file
- Here, we will assume that there is an RA function for ‘LIKE’
    - e.g. assume that we can use string_input LIKE regex in our RA

Query:

```
SELECT product_id, brand_name, product_name
    FROM brand NATURAL JOIN store NATURAL JOIN product
    NATURAL JOIN (SELECT product_id FROM
    has_ingredient NATURAL JOIN ingredient WHERE
    ingredient_name LIKE '%beta%') AS p;
```

Relational Algebra:

$$\Pi_{\text{product\_id, brand\_name, product\_name}} (\text{brand} \bowtie \text{store} \bowtie \text{product} \bowtie (\Pi_{\text{product\_id}} (\sigma_{\text{ingredient\_name LIKE '\%beta\%'}} (\text{has\_ingredient} \bowtie \text{ingredient}))))$$

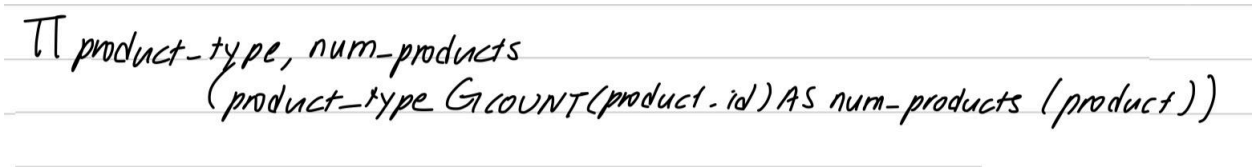
Relational Algebra 4

- Includes aggregation
- Though this query is not used in our application, it is included in our queries.sql file

Query:

```
SELECT product_type, COUNT(product_id) AS num_products
FROM product GROUP BY product_type;
```

Relational Algebra:



Part L1. Written Reflection Responses

Challenges and Limitations

List any problems (at least one) that came up in the design and implementation of your database/application (minimum 2-3 sentences)

Answer:

A challenge we encountered in the design of our database was figuring out how to reduce the bulkiness that ingredients, as a TEXT field, added to the product table. Based on feedback in OH, we decided to create a separate table for ingredients. While we initially thought this process was going to be simple, since the ingredients data appeared to be simply a list of comma-delimited ingredients in one string, we realized that this ingredients data is not consistent throughout the dataset we have downloaded. For instance, some of the tuples would have invalid ingredient data (e.g. empty data or strings like "Visit Dior store for more!").

Another problem was that the ingredients data was too bulky to be processed on excel or Google Sheets. While Google Sheets provides an easy tool to break down comma-delimited string data into multiple columns, because we had over 8000 rows of ingredients data (raw, before our data-processing), we were unable to process our data through this tool. Thus, we have decided to write a Python script, breaking these data down into a usable format. We then had to manually go through the dataset to clean up all the duplicates and invalid data. Due to the inconsistency, there were a lot of manual work involved: e.g. some rows may have an ingredient written in a different language (e.g. Miel/Honey), which is technically a duplicate of ingredient _name, since they are referring to the same ingredient.

We also had to remove any irrelevant strings that give us no information about the ingredients themselves. After several hours of manual data pre-processing, we were able to generate ingredients.csv, which contained a mapping of an ingredient_id to ingredient_name. Then, we wrote the second part of our script through which we create a mapping of `product_id` to `ingredient_id`. We went over each `ingredients` ***string, for which we looped through every ingredient to see if the ingredient exists within the string. This was surely an inefficient process, especially considering how bulky our ingredients dataset is. However, due to the nature of the data we had, we couldn't figure out a better way to tackle this.***

In the end, we were able to generate a has_ingredients table, which contains more than 122,000 rows. While parsing through ingredients data was an ordeal, this allowed us to add a nice feature to our website: filtering products based on their ingredients. This allowed us to better serve the purpose of our application and reminded us of our motivation to build a

website that is allergy and vegan-friendly (meaning we allow vegans or users with allergies to easily find the right product for them).

Another challenge we had was with our routines for when a user adds to cart and also checks out their cart. We brainstormed many ways to tackle this problem. Initially, we planned to check out the product immediately when the user adds to the cart (hence, trigger will remove the item from inventory upon the addition of item to the cart), which felt counter-intuitive considering that users shouldn't have to check out until they choose to do so.

Thus, we decided to keep a separate table `product_history` which will store all the checked-out items by the user. We also kept track of inventory in the store table, a cart associated with a username with products, and using this, maintained an up-to-date history of a user's purchases. We created a procedure to add an item to cart that would check if the product was not out of stock before the user added it to cart. For checkout, we had a procedure to update the inventory in the store table and then a trigger that would transfer the cart data associated with the user that checked out to the purchase history table where the time of the purchase was also logged. This allowed us to add another nice feature to our website, my page, where users can track their past order history.

In our project, we have also made an effort to improve client user experience by developing a website. Initially, we started off with a Python application for both admin and user. However, we thought that the textual nature of the application greatly limited the user experience, especially for the client. Due to the bulkiness of our data, it was often challenging to present it in a nice way so that the user can easily browse through. Clearly, it is not ideal for us to expect the user to maximize their terminal screen to scroll through thousands of rows of results. Thus, we decided to implement the website, which greatly eases the user experience and interactions with our service. The admin application was kept as a Python application, since its functionality was simple enough to be done through the command line interface.

With the development of the website came challenges. One major bug of our current client website is that the client doesn't automatically get logged out when they close the tab. Thus, the login info is maintained in our session variable. This may potentially create an issue if the admin decides to delete the user when the login info of the user in the session variable has not been flushed. This is a challenge that is unique to the Flask framework environment that we developed the website in. We didn't have enough time to research how to flush session variables automatically on close of the application. However, we have added error handling mechanisms to catch database errors and notify our users that a MySQL error has occurred, so further actions can be taken on their part. As of now, we recommend that you manually log out of the website before you close. Another issue that you may encounter is loading. It takes a bit of time for a Flask application to load for the first time. We have attached a demo video of our website in case you run into a problem with running the application.

Future work

If you are particularly eager for a certain application (have your own start-up in mind?), it is easy to over-scope a final project, especially one that isn't a term-long project. You can list any stretch goals you might have "if you had the time" which staff can help give feedback on prioritizing (2-3 sentences).

Answer:
If we had more time, we would have liked to incorporate a sale/coupon table that allows users to apply discounts to items of a certain brand or product type to reduce their price (check additional-ddl.sql for the DDL of sale table that we decided to take out for the sake of maintaining a manageable scope for the project).

Additionally, we would like to further develop the web application for clients, by potentially providing images associated with each product to ameliorate the user experience. We could not add this feature to our website for our submission, because we have over 1000 products on our website, and it would be too time consuming for us to manually download all the images from online for each of the products.

Furthermore, we would like to add a feature where clients/shoppers would be allowed to leave a review for specific products and brands. Currently, our application contains ratings for each product. With this new feature, we should be able to dynamically update this rating if there is any new review left on the product.

Collaboration (Required for Partners)

This section is required for projects which involved partner work. Each partner should include 2-3 sentences identifying the amount of time they spent working on the project, as well as their specific responsibilities and overall experience working with a partner in this project.

Partner 1 Name: *Eshani Patel*
Partner 1 Responsibilities and Reflection:
Responsibilities:

- *ER Diagram*
- *DDL*
- *SQL data loading into database*
- *Password management*
- *MySQL Users and Permission*
- *SQL queries and Procedural SQL used in command-line python application*
- *reflection.*

I spent about 20 hours on this project, working over the course of two weeks.

My experience working with Rachael was great, we communicated with each other well and met regularly to figure out our plan-of-action every few days, and we were both enthusiastic about developing this project and going above and beyonf.

Partner 2 Name: *Rachael Kim*
Partner 2 Responsibilities and Reflection:

Responsibilities:

- *Verifying/Editing ER diagram*
- *Editing/Writing DDL*
- *MySQL UDFs, Procedures, Triggers (collaborative work with Eshani, as we continued to develop these throughout our project to better suit the logic of our application)*
- *Developing client application (which is our website) and admin application*
- *Writing queries for the applications (as well as using queries written by Eshani)*
- *Data processing and cleaning*
- *Reflection*

I have spent about 25 hours on this project. I really enjoyed working with Eshani! Our project turned out to be better than what I expected because both of us were willing to put in the work. I was definitely over-ambitious, so I'm really grateful that Eshani put in the work to meet that ambition and produce amazing results together.

OTHER COMMENTS

This is the first time CS 121 has had a Final Project, and we would appreciate your feedback on whether you would recommend this in future terms, as well as what you found most helpful, and what you might find helpful to change.

Answer:
Having to finish the project during finals week was definitely painful, especially because of the time-consuming nature of it, but we really enjoyed working with real-life data and creating an application that can be useful.