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): Snigdha Saha, Sreemanti Dey

Student email(s): snigdha@caltech.edu, sdey@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 made our own version of Beli, a food-review app that allows users to find and rate restaurants among categories, get recommendations, and add friends to share ratings with. Our version will, similarly, allow users to rate restaurants, find generally good restaurants, and to choose restaurants based on the ratings of their friends, among other utilities. This is an application that requires keeping track of many different types of data, including users, restaurants, ratings, etc. It is therefore best to encapsulate the data into a well-designed SQL database and Python application.

Data set (general or specific) Answer:

CS121_Final_Project_Data_Generation.ipynb

We randomly generated our dataset in Python, since our database structure is pretty unique, and finding data in the desired format was difficult. We used the 'faker' library in Python for most of the generating work.

Faker provides a method to randomly generate names, and we generated usernames and emails based on the fake names. Passwords were completely randomly generated.

For restaurants, we hardcoded a set of cuisines and categories. Then, we used faker to generate names consistent with each cuisine, and randomly chose a category for the resulting restaurant (e.g. Akira's Bakery would be in the Bakery category and Japanese cuisine). We also had to use the

'deep_translator' library for this, since the faker library writes in the alphabet of the location given (so for Japanese cuisine the name would be in kanji).

To summarize, we generated user info and restaurant information with faker, hardcoded cuisine and category, and derived the content of the other tables from those. We also hardcoded some chain restaurants.

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

The intended clients are the users on the Beli app. They will create their profiles, rank restaurants, add descriptions, and connect with friends. They can search for restaurants and query for rankings, and to find suggested restaurants.

Admin user(s) Answer:

The admin is the only person who is able to adjust the restaurants and adjust the ranking formula. So, the admin will be the ones adding and updating restaurant data. We will decide a weighting formula to help create predicted rankings for users for each restaurant.

Part A. ER Diagrams

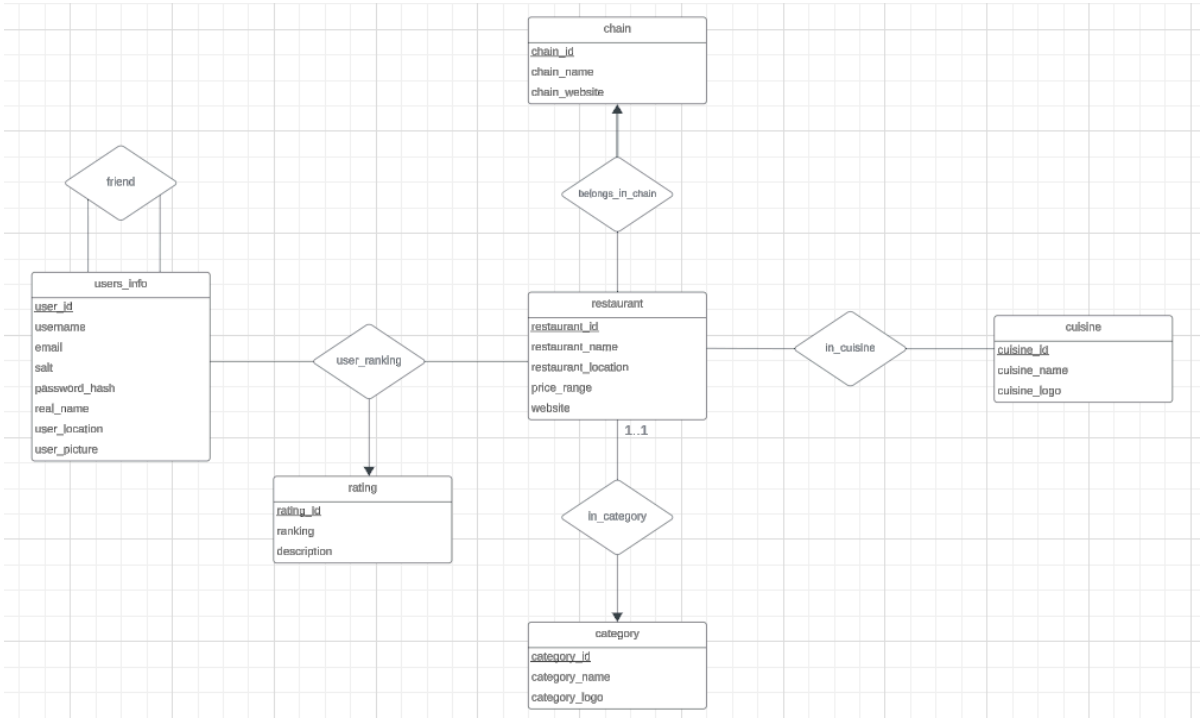
As we've practiced these past few weeks, the ER model is essential for designing your database application, and we expect you to iterate upon your design as you work through the ER and implementation steps. In this answer, you should provide a full ER diagram of your system. Your grade will be based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



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 created an index (called loc_idx) over restaurant_locations in the restaurant table.

Justification and Performance Testing Results:

We chose this index because we will often be searching for results from restaurant based on location, and hence it makes sense to speed up this process.

We use EXPLAIN to verify that the index is used:

EXPLAIN select * from restaurant where restaurant_location='Oakland';

```
+----+-------------+------------+------------+------+---------------+---------+---------+-------+------+--
--------+-------+
| id | select_type | table      | partitions | type | possible_keys | key     | key_len | ref   | rows | f
iltered | Extra |
+----+-------------+------------+------------+------+---------------+---------+---------+-------+------+--
--------+-------+
|  1 | SIMPLE      | restaurant | NULL       | ref  | loc_idx       | loc_idx | 803     | const |    4 |
 100.00 | NULL  |
+----+-------------+------------+------------+------+---------------+---------+---------+-------+------+--
--------+-------+
```

And finally, we test on a few queries with a non-indexed restaurant table, and we observe a good amount of performance improvement:

```
|     9497 | 0.00056475 | select * from restaurant_non_indexed where restaurant_location='Dallas'
|
|     9498 | 0.00052725 | select * from restaurant where restaurant_location='Dallas'
|
|     9499 | 0.01198925 | select * from restaurant_non_indexed where restaurant_location='Oakland'
|
|     9500 | 0.00042050 | select * from restaurant where restaurant_location='Oakland'
|
```

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.

Query 1:

$$_{username}G_{COUNT(rating)\ as\ num\_ratings,\ AVG(rating)\ as\ avg\_rating}\ (user\_rating \bowtie rating \bowtie users\_info)$$

Description: Gets the number of ratings and average rating for each user that has made at least one rating.

Query 2:

$$chain\_rests \leftarrow \Pi_{chain\_name,\ restaurant\_id}(chain \bowtie belongs\_in\_chain)$$

$$_{chain\_name}G_{AVG(rating)\ as\ avg\_rating}(chain\_rests \bowtie user\_rating \bowtie rating)$$

Description: Gets the average rating over all locations for each chain restaurant. That is, if a restaurant is a chain, then this query computes its average rating over all of its locations.

Query 3:

$$\Pi_{rating\_id,\, rating,\, REPLACE(rating\_description,\, "crap",\, "****")}(\sigma_{rating\_description\ LIKE\ "\%crap\%"}(rating)) \cup$$

$$\Pi_{rating\_id,\, rating,\, rating\_description}(\sigma_{rating\_description\ NOT\ LIKE\ "\%crap\%"}(rating))$$

Note: REPLACE is being used like the MySQL function REPLACE, not an aggregation. Similarly for LIKE and NOT LIKE.

Description: Censors out the word “crap” in the rating descriptions by selecting all the rows in rating with a rating description containing the word, then projecting to the other variables along with the rating description with the word replaced by ****. Then, these modified rows are unioned with the rows that did not include the word and are thus unchanged.

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:

Our database, on a logical level, has quite a few fields that would be most conveniently represented using non-atomic data types. One example of this is a user's friends; another is top restaurants by cuisine. It turned out that, rather than a straightforward replacement of all these fields with new tables for them (as discussed in the ER diagram slides), it was better in some cases to instead make materialized views or triggers.

Feedback from El and Peers

Significant feedback and implemented changes

Answer:

During our meeting with El, we got a lot of feedback regarding our UI. Firstly, we made our UI more user friendly by continuously looping the user back to the main menu after they did an action, instead of quitting the application (on client and user side). Then, we made sure to do error handling appropriately, so that we accounted for bad inputs from users or NULL results from the backend. Finally, we clarified descriptions of the actions associated with each menu choice (i.e. exports, changes to the backend, appropriate input options, etc.).

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:

One stretch goal that we have is being able to have functionality for restaurants with the same name in the same city. As of now, allowing that makes searching for chain restaurants very onerous, so to implement this change, we would likely have to either change some aspect of how we store restaurant data, or go through and write a very complicated chain searching function.

Another stretch goal would be to implement a recommendation system based on more granular location data than just cities. Currently, we have no way of implementing this, since we would need access to a very detailed geographic database to do so. However, if we had such data, our database design is well-equipped to take advantage of it, since we already do recommendations based on city and average rating.

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: Snigdha Saha
Partner 1 Responsibilities and Reflection:
Responsibilities: Made ER Diagram together, did Procedural SQL together, Index Performance, Password Management, User Passwords, Command-Line Python

Reflection: I spent roughly 12 hours total on this project throughout the term. The bulk of my time was focused on the command-line Python to ensure our menu interface worked with our database well. As a regular Beli user, I enjoyed working on this project and thinking about the design decisions made. I was grateful to work with a partner as we made a lot of decisions together through extensive brainstorming sessions, and we also identified a lot of good "scope vs over-scoping" areas, with the constraints of our knowledge and our time.

Partner 2 Name: Sreemanti Dey
Partner 2 Responsibilities and Reflection:
Responsibilities: Made ER Diagram together, DDL, Data Generation and Loading, Relational Algebra, SQL Queries, did Procedural SQL together
Reflection: I spent about 12 hours total on this project, counting individual coding and partner coding and debugging sessions. My responsibilities were as stated above, mostly falling into the categories of data, queries, and procedural work. Working with a partner was very helpful for this project, not only because it allowed us to implement more, but also because it was easier to brainstorm a good database structure and possible improvements as we went along.

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: We would recommend this for future terms, since it serves as a comprehensive test of what we learned in the course, while implementing an application that interests us. We found the testing sessions to be helpful (i.e. running through the UI and seeing the way our program will be tested), as well as getting peer and instructor feedback early on our ER diagrams.