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): Anika Arora, Andy Dimnaku

Student email(s): aarora@caltech.edu, adimnaku@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 designed a streaming services database which included various information about Netflix, Hulu, Amazon Prime, and Disney+. We were motivated to create an application where it was easy to compare and contrast the various streaming services based on what and how much content they have, along with their prices, deals, and reviews from users. Finding consolidated information about streaming services in one place on the internet is hard, so we thought using a database to represent this was a good idea.

Data set (general or specific) Answer:

We found 4 separate Kaggle datasets representing the content on Netflix, Hulu, Amazon Prime, and Disney+. We used Python and Pandas to combine this into one big dataset that contained information about all four streaming services, while adding a few of our own attributes. We generated random data for specific movie/TV show ratings on each streaming service to create the content_ratings table. We created the deals and reviews table on our own by searching up information about each of the streaming services online. We filled the reviews table with actual reviews about each of the services that we found online.

Kaggle datasets:

- *Disney +:* ***https://www.kaggle.com/datasets/shivamb/disney-movies-and-tv-shows***
- *Netflix:* ***https://www.kaggle.com/datasets/shivamb/netflix-shows***
- *Hulu:* ***https://www.kaggle.com/datasets/shivamb/hulu-movies-and-tv-shows***
- *Amazon prime:* ***https://www.kaggle.com/datasets/shivamb/amazon-prime-movies-and-tv-shows***

Client user(s) Answer:

Clients are able to view any information from the database. They cannot make any changes or additions to any of the tables. They simply have SELECT privileges.

Admin user(s) Answer:

Admins can make any changes in the database. They are allowed to add, remove, and update values in tables. Admins are those who are managing the database and constantly keeping it up to date.

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.

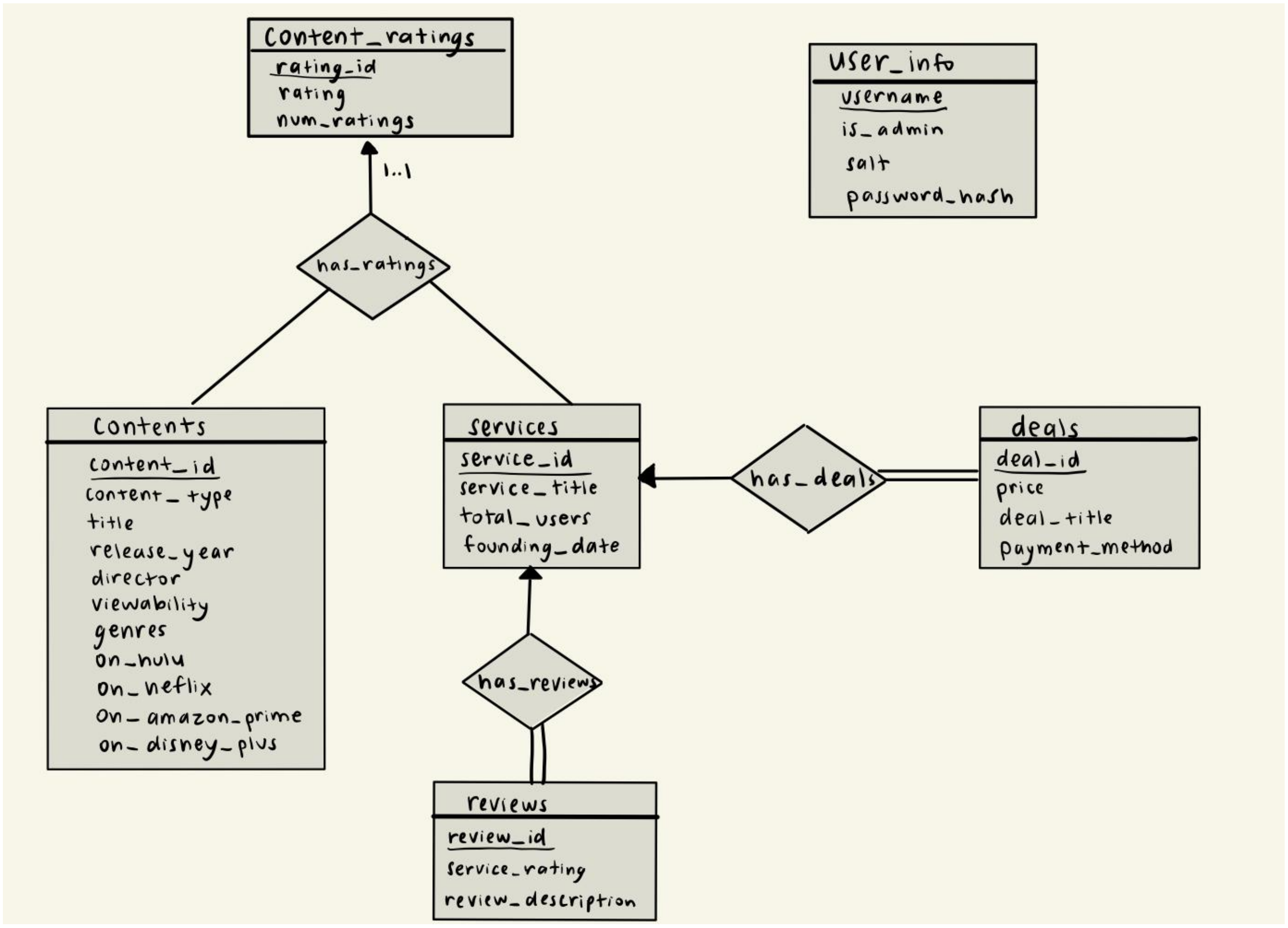
Notes: For this section **only**, we will allow (and encourage) students to share their diagrams on Discord (**#er-diagram-feedback**) to get feedback from other students on their ER diagrams given a brief summary of your dataset and domain requirements. This is offered as an opportunity to test your 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:

We have a ternary relationship with the entity sets content_ratings, contents, and services. This is because the combination of every piece of content and its corresponding service will always correspond to exactly one rating. This explains the 1..1 mapping as well. Furthermore, we have services and deals entity sets with a has_deals relationship set. There is total participation from the deals side, as every deal must be a part of a service, but only partial participation from the services side, as not every service has to have deals. Similarly, we have services and reviews entity sets with a has_reviews relationship set. There is total participation from the reviews side, as every review must be for a service, but only partial participation from the services side, as not every service has to have reviews. One deal can only point to one service, so we have a one-to-many relationship from deals to services. Again, one review can only correspond to one service, so we have a one-to-many relationship from reviews to services. User_info is an entity set that is not in any relationships.



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).

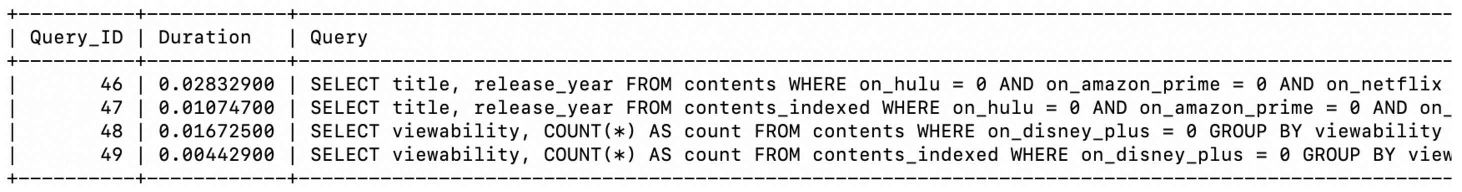
Here, describe your process for choosing your index(es) and show that it is used by at least one query, which speeds up the performance of the same query on a version of the same table without that index. You may find lec14-analysis.sql and Lecture 14 slides on indexes useful for strategies to choose 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 chose to index on each of the four boolean columns that represent whether the content is in the streaming service. So we added separate indices on the 'on_netflix', 'on_hulu', 'on_disney_plus', and 'on_amazon_prime' columns of the contents table.

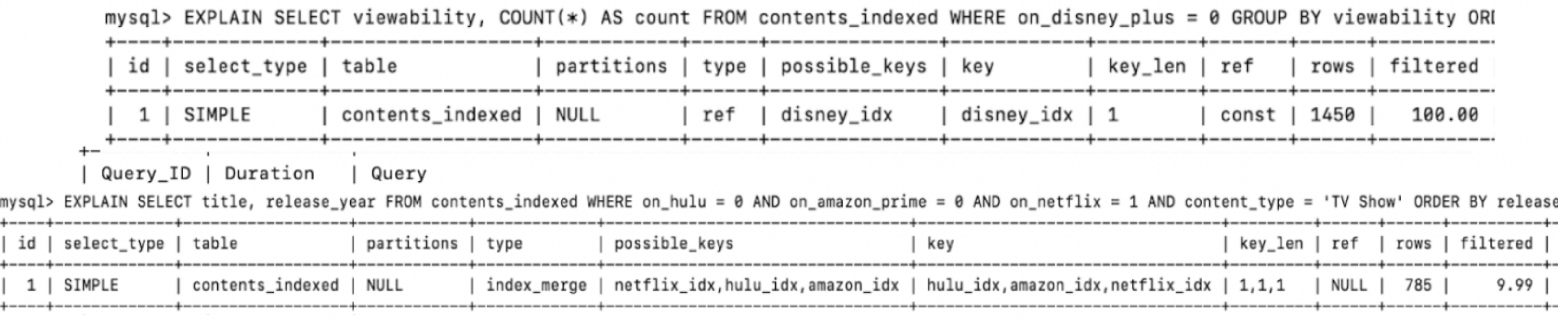
Justification and Performance Testing Results:

We thought that it would be a good idea to index on these columns because they may commonly be used in the WHERE conditions of queries when someone is trying to find information about which streaming services have certain content. We tested out two of our queries with and without the index. The first query gets all of the TV Shows that are both on Amazon Prime and Hulu, but not Netflix. This corresponds to 46 and 47 in the table below. The other query counts the number of movies on Disney Plus for each viewability category. This corresponds to 48 and 49 in the table below. We can see that both of the queries that are on the indexed tables perform significantly better.



```
+----------+------------+---------------------------------------------------------------------------------------------------
| Query_ID | Duration   | Query
+----------+------------+---------------------------------------------------------------------------------------------------
|       46 | 0.02832900 | SELECT title, release_year FROM contents WHERE on_hulu = 0 AND on_amazon_prime = 0 AND on_netflix
|       47 | 0.01074700 | SELECT title, release_year FROM contents_indexed WHERE on_hulu = 0 AND on_amazon_prime = 0 AND on_
|       48 | 0.01672500 | SELECT viewability, COUNT(*) AS count FROM contents WHERE on_disney_plus = 0 GROUP BY viewability
|       49 | 0.00442900 | SELECT viewability, COUNT(*) AS count FROM contents_indexed WHERE on_disney_plus = 0 GROUP BY view
+----------+------------+---------------------------------------------------------------------------------------------------
```

We also can confirm that our indices are actually being used by using the EXPLAIN command on our queries:



```
 mysql> EXPLAIN SELECT viewability, COUNT(*) AS count FROM contents_indexed WHERE on_disney_plus = 0 GROUP BY viewability ORI
 +----+-------------+------------------+------------+------+---------------+------------+---------+-------+------+----------
 | id | select_type | table            | partitions | type | possible_keys | key        | key_len | ref   | rows | filtered
 +----+-------------+------------------+------------+------+---------------+------------+---------+-------+------+----------
 |  1 | SIMPLE      | contents_indexed | NULL       | ref  | disney_idx    | disney_idx | 1       | const | 1450 |   100.00
 +----+-------------+------------------+------------+------+---------------+------------+---------+-------+------+----------
+-
| Query_ID | Duration   | Query
mysql> EXPLAIN SELECT title, release_year FROM contents_indexed WHERE on_hulu = 0 AND on_amazon_prime = 0 AND on_netflix = 1 AND content_type = 'TV Show' ORDER BY release
+----+-------------+------------------+------------+-------------+-------------------------------+-------------------------------+---------+------+------+----------+
| id | select_type | table            | partitions | type        | possible_keys                 | key                           | key_len | ref  | rows | filtered |
+----+-------------+------------------+------------+-------------+-------------------------------+-------------------------------+---------+------+------+----------+
|  1 | SIMPLE      | contents_indexed | NULL       | index_merge | netflix_idx,hulu_idx,amazon_idx | hulu_idx,amazon_idx,netflix_idx | 1,1,1   | NULL |  785 |     9.99 |
+----+-------------+------------------+------------+-------------+-------------------------------+-------------------------------+---------+------+------+----------+
```

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:

Functional dependency in contents table: {content_type, title, release_year} ⟶ {content_id}

Functional dependency in deals table: {service_id, price, deal_title} ⟶ {deal_id}

Normal Forms Used (with Justifications):

Reviews, deals, content_ratings, and services are all in 3NF. We reasoned that our dataset is not too large, as the biggest table content_ratings and contents contains around 20,000 tuples, while the other tables all contain less than 100 tuples. 3NF gives our schema the potential to reduce extra redundancy. It also preserves all functional dependencies and ensures the complete correctness of our database, which is very important in our case of streaming services since there is a lot of correlated information between content, streaming services and ratings.

NF Proof 1:

We can show that content_ratings is in 3NF.

We have the following non-trivial functional dependencies in the content_ratings relation:

{content_id, service_id} ⟶ rating_id,

{content_id, service_id, rating} ⟶ rating_id,

{content_id, service_id, rating, num_ratings} ⟶ rating_id,

rating_id ⟶ {content_id, service_id, rating, num_ratings}, and all of the other functional dependencies generated from this one using the decomposition rule.

We can show that these non-trivial functional dependencies α → β have α has a superkey for content_ratings. We can compute the attribute set closures for all α in the functional dependencies listed above.

1. $\{content\_id,\ service\_id\}^+\ -> \{content\_id,\ service\_id\}\ -> \{content\_id,\ service\_id,\ rating\_id\}\ -> \{content\_id,\ service\_id,\ rating\_id,\ rating,\ num\_ratings\}$

2. $\{content\_id,\ service\_id,\ rating\}^+\ -> \{content\_id,\ service\_id,\ rating\}\ -> \{content\_id,\ service\_id,\ rating,\ rating\_id\}\ -> \{content\_id,\ service\_id,\ rating,\ rating\_id,\ num\_ratings\}$

3. $\{content\_id,\ service\_id,\ rating,\ num\_ratings\}^+\ -> \{content\_id,\ service\_id,\ rating,\ num\_ratings\}\ -> \{content\_id,\ service\_id,\ rating,\ num\_ratings,\ rating\_id\}$

4. $rating\_id^+\ -> rating\_id\ -> \{rating\_id,\ content\_id,\ service\_id,\ rating,\ num\_ratings\}$

Therefore, we can see that the attribute set closures indicate that all α in the non trivial functional dependencies α → β are superkeys for the content_ratings relation, since they include all of the attributes.

However, we can also see that there are overlapping candidate keys. For example, for the functional dependency rating_id → {content_id, service_id, rating, num_ratings}, each attribute A in β – α is contained in a candidate key for R. Each of the attributes content_id, service_id, rating, and num_ratings is contained in a candidate key. Besides the primary key rating_id, we have a minimal candidate key of {content_id, service_id}, so {content_id, service_id, rating}, and {content_id, service_id, rating, num_ratings} are also candidate keys. Therefore, our relation content_ratings is in 3NF.

NF Proof 2:

We can show that the deals table is in 3NF..

We have the following non-trivial functional dependencies in the content_ratings relation:

{service_id, deal_title} → deal_id,

{service_id, price, deal_title} → deal_id,

{service_id, price, deal_title, payment_method} → deal_id, deal_id ⟶ {service_id, price, deal_title, payment_method}, and all of the other functional dependencies generated from this one using the decomposition rule.

We can show that these non-trivial functional dependencies α ⟶ β have α has a superkey for the deals relation. We can compute the attribute set closures for all α in the functional dependencies listed above.

1. $\{service\_id,\ deal\_title\}^{+} -> \{service\_id,\ deal\_title\} -> \{service\_id,\ deal\_title,\ deal\_id\} -> \{service\_id,\ deal\_title,\ deal\_id,\ price,\ payment\_method\}$

2. $\{service\_id,\ price,\ deal\_title\}^{+} -> \{service\_id,\ price,\ deal\_title\} -> \{service\_id,\ price,\ deal\_title,\ deal\_id\} -> \{service\_id,\ deal\_title,\ deal\_id,\ price,\ payment\_method\}$

3. $\{service\_id,\ price,\ deal\_title,\ payment\_method\}^{+} -> \{service\_id,\ price,\ deal\_title,\ payment\_method\} -> \{service\_id,\ price,\ deal\_title,\ payment\_method,\ deal\_id\}$

4. $\{deal\_id\}^{+} -> \{deal\_id\} -> \{deal\_id,\ service\_id,\ price,\ deal\_title,\ payment\_method\}$

Therefore, we can see that the attribute set closures indicate that all α in the non trivial functional dependencies α ⟶ β are superkeys for the reviews relation, since they include all of the attributes.

However, we can also see that there are overlapping candidate keys. For example, for the functional deal_id ⟶ {service_id, price, deal_title, payment_method}, each attribute A in β – α is contained in a candidate key for R. Each of the attributes service_id, price, deal_title, and payment_method is contained in a candidate key. Besides the primary key deal_id, we have a minimal candidate key of {service_id, deal_title}, so {service_id, deal_title, price}, and {service_id, deal_title, price, payment method} are also candidate keys. herefore, our relation reviews is in 3NF.

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.

1. Gets the service_title, service_rating, and review_description for all bad reviews across all services. We define a bad review as one that is less than 3 stars.

$$\Pi_{service\_title,\, service\_rating,\, review\_description}(\, \sigma_{service\_rating\, <\, 3}(reviews \bowtie services))$$

2. Displays all deals for the service with the highest average reviews including the columns service_title, price, deal_title, and payment_method. Using temporary tables here will help store information that we need. We first must find the service corresponding to the highest average service_ratings across all ratings, and then we must find the deals for this service.

$$t1 \leftarrow {}_{service\_id}G_{avg(service\_rating)\ as\ avg\_rating}(reviews)$$
$$t2 \leftarrow G_{max(avg\_rating)\ as\ avg\_rating}(t1)$$
$$\Pi_{service\_title,\, price,\, deal\_title,\, payment\_method}((t2 \bowtie t1) \bowtie services \bowtie deals)$$

3. Update the deals table such that it removes all deals from Hulu that require yearly payments. We assume that we don't know the service_id for Hulu, and only have the service_title.

$$deals \leftarrow deals\ -$$
$$\Pi_{deal\_id,\, service\_id,\, price,\, deal\_title,\, payment\_method}(\sigma_{payment\_method="yearly"\, \wedge\, service\_title\, =\, "Hulu"}(deals \bowtie services))$$

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:
One challenge that we faced was the initial data processing. We were using four separate raw data csvs, each representing all content held in one streaming service. We wanted to combine all of these datasets into one table that combined all of the content from the four streaming services. Our aim was also to include information about which content was on multiple streaming services by adding extra columns like (on_netflix, on_hulu, etc.). Doing this was challenging as we had to use Python and Pandas operations to create our final data table. Additionally, we originally wanted to combine our streaming service data with IMDB ratings data for specific movies and TV shows. While we found an IMDB dataset for ratings, it was very difficult to combine this with the Kaggle datasets we found, as the movies barely overlapped and IMDB had much older data. We ended up just generating our own random data for ratings. Our data generation described above is part of our submitted files under gen_data.py (originally from colab notebook https://colab.research.google.com/drive/1xrL0qcFQf6lxrP65ZoY26XL3tn0HiRph?usp=sharing).

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: It would have been nice to better scope out client and admin privileges, and provide more distinction between what each type of user can do. One possible stretch goal: Users can submit a request to add their own comments on streaming services based on the information that they see or prior experience. If approved, admins can update the comments table.
Additionally, eventually integrating this dataset with real ratings data, instead of randomly generated values, would be a good goal.

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: Anika Arora
Partner 1 Responsibilities and Reflection:
I spent around 15-20 hours working on this project. My responsibilities included the ER diagrams, functional dependency analysis, relational algebra, and SQL queries. Andy and I worked on the app.py together and helped each other throughout the project when we ran into issues. Overall, it was a good experience and we worked well together. It would have been much harder to do this project on my own. I really enjoyed seeing Python and MySQL work together, and this was actually a lot easier to do than I expected it to be. The app.py demo was very helpful!

Partner 2 Name: Andy Dimnaku
Partner 2 Responsibilities and Reflection:
I spend around 15-20 hours on the project as well with the parts I worked on were the DDL setup.sql file, load-data.sql file, the password management in the setup-passwords.sql file, mysql user permissions in the grant-permissions.sql file, and the procedure sql in the setup-routines.sql. Setting up the load-data file and editing the csvs took more time than expected. This was due to the different places that we were getting the data from and setting it up so it was better able to get integrated into the database. Also, I worked on the app.py file with Anika where we set up our python user interface for the database. I found that going through this process I was able to learn a lot how to use and create my own database. I found the python part interesting where we made it work with our database. I found the project good and it helped having Anika there to work with me and split the workload of creating the database .

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 think the project was a very good overview of what we learned in this class, and was very rewarding to develop. We would recommend that this is in all future classes. We think it maybe would have been helpful to have had at least 2 weeks of working on the final project without any overlapping assignments, as we felt a little bit overwhelmed with time.