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): Ishaan Mantripragada, Rohan Jha, Olivia Xu

Student email(s): imantrip@caltech.edu, rjha@caltech.edu, oxu@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) :

The application we designed was a movie recommendation system. Netflix, Hulu, Max, and other video streaming services are used by millions of people every day including the three of us. We wanted to work on something that would actually prove useful and interesting for our final project. We chose the specific movie dataset from Kaggle because it had unique features such as genre and cast that we specifically wanted to incorporate into our recommendation system.

Data set (general or specific) Answer:

Kaggle Dataset: https://www.kaggle.com/datasets/thedevastator/imdb-movie-ratings-dataset.

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

A client in this database has the ability to view different movies and rate their own movies. They can search for a list of movies based on specific preferences such as genre.

Admin user(s) Answer:

An admin in this database has the ability to add new movies to the database as they come out for clients to search for and rate. Admins also have the ability to look at how users have rated movies which may provide information about the types of movies current users enjoy.

Part A. ER Diagrams

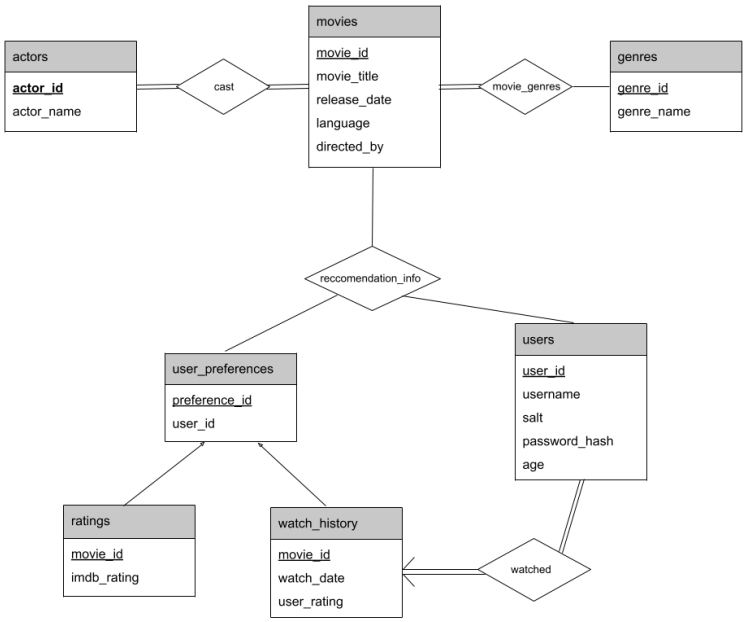
As we've practiced these past few weeks, the ER model is essential for designing your database application, and we expect you to iterate upon your design as you work through the ER and implementation steps. In this answer, you should provide a full ER diagram of your system. Your grade will be based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



Entities and Attributes

1. User Entity:
    a. Attributes: `user_id`, `username`, `password`, `age`
    b. The `user_id` is the primary key.
    c. Consider hashing the `password` for security, which is not depicted in the ER diagram but will be in the implementation.
2. Movie Entity:
    a. Attributes: `movie_id`, `title`, `release_date`, `language`, `directed_by`, `overview`
    b. The `movie_id` is the primary key.
3. Genre Entity:
    a. Attributes: `genre_id`, `name`
    b. The `genre_id` is the primary key.
4. Actor Entity:
    a. Attributes: `actor_id`, `name`
    b. The `actor_id` is the primary key.
5. Cast Entity (Association Entity linking Movies and Actors):
    a. Attributes: `movie_id`, `actor_id`
    b. Composite primary key (`movie_id`, `actor_id`), both of which are foreign keys.
6. Watch History Entity:
    a. Attributes: `user_id`, `movie_id`, `watch_date`, `user_rating`

    b. Composite primary key (`user_id`, `movie_id`), both are foreign keys.
7. User Preferences Entity:
    a. Attributes: `preference_id`, `user_id`, `movie_id`, `actor_id`
    b. The `preference_id` is the primary key, and `user_id`, `movie_id`, and `actor_id` are foreign keys.
8. Ratings Entity:
    a. Attributes: `movie_id`, `user_rating`, `imdb_rating`
    b. The `movie_id` is the primary key and a foreign key.

Relationships

- A User can have multiple entries in Watch History (one-to-many).
- A User can have multiple User Preferences (one-to-many).
- A Movie can have multiple Genres (many-to-many, which implies a junction table between Movies and Genres).
- A Movie can have multiple Actors through the Cast (many-to-many).
- A Movie has multiple Ratings (one-to-many).

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):

CREATE INDEX idx_movies_title ON movies(movie_title);

Justification and Performance Testing Results:

This index improves the performance because we may want to search for movies based on the movie title.

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:

1. `users` table:
    a. `user_id` → `username, hashed_password, age`
    b. The `user_id` uniquely identifies a user's username, hashed password, and age.
2. `movies` table:
    a. `movie_id` → `movie_title, release_date, language, directed_by, overview`
    b. The `movie_id` uniquely identifies all the other attributes of a movie.
3. `genres` table:
    a. `genre_id` → `genre_name`
    b. The `genre_id` uniquely identifies the genre name.
4. `actors` table:
    a. `actor_id` → `name`
    b. The `actor_id` uniquely identifies the actor's name.
5. `watch_history` table:
    a. `user_id, movie_id` → `watch_date, user_rating`
    b. The combination of `user_id` and `movie_id` uniquely identifies the watch date and user rating.
6. `ratings` table:
    a. `movie_id` → `imdb_rating`
    b. The `movie_id` uniquely identifies the IMDb rating for a movie.

Normal Forms Used (with Justifications):

Users (3NF):

- It's in 2NF because all non-primary-key attributes are fully functionally dependent on the primary key user_id.
- It's in 3NF because there are no transitive dependencies between non-primary-key attributes. Each attribute is directly dependent on the primary key (user_id) and not on any other non-primary key attribute.

Movies (3NF):

- It's in 2NF because every attribute is fully functionally dependent on the primary key, movie_id.
- It's in 3NF as there are no transitive dependencies; movie_title, release_date, language, directed_by, and imdb_movie_link all directly depend on movie_id and not on each other.

Watch_history (BCNF):

- It is in 3NF for the same reasons as above.
- Every determinant in the table is a candidate key. The composite primary key (user_id, movie_id) fully determines every other non-key attribute (watch_date, user_rating). There's no attribute that can determine any part of the primary key, ensuring all dependencies preserve the BCNF condition.

Movie_genres (BCNF):

- It is in 3NF for the same reasons as above..
- The primary key is the composite key (movie_id, genre_id), and there are no non-prime attributes. Hence, all attributes are fully functionally dependent on the primary key, and there are no dependencies on non-candidate keys.

NF Proof 1:

Proof for Users:

- Assuming a single primary key user_id, there are no attributes functionally dependent on another non-primary key attribute, thus satisfying the requirement for 3NF.
- See justification for more depth.

NF Proof 2:

Proof for Movies:

- Given the primary key movie_id, there's a direct functional dependency of all other attributes on this key without any transitive dependency, aligning with the criteria for 3NF.
- See justification for more depth.

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. -- This query first identifies the movie_id of "Example Movie" from the movies table and
   -- then uses that movie_id to update the IMDb rating in the ratings table.

   ```
   UPDATE ratings
   SET imdb_rating = new_rating
   WHERE movie_id = (SELECT movie_id FROM movies WHERE movie_title = 'Specific Movie Title');
   ```

2. -- This query retrieves movies that a specific actor has been in, which also belong to a
   -- specific genre. This might be useful for users interested in exploring movies of a particular
   -- genre that feature their favorite actor.

   ```
   SELECT DISTINCT m.movie_title, m.release_date
   FROM movies m
   JOIN cast c ON m.movie_id = c.movie_id
   JOIN actors a ON c.actor_id = a.actor_id
   JOIN movie_genres mg ON m.movie_id = mg.movie_id
   JOIN genres g ON mg.genre_id = g.genre_id
   WHERE a.name = 'Actor Name' AND g.genre_name = 'Genre Name'
   ORDER BY m.release_date DESC;
   ```

3. -- This query groups results by genre and calculates the average IMDb rating. It also counts
   -- the number of movies in each genre.

   ```
   SELECT g.genre_name, AVG(r.imdb_rating) AS average_imdb_rating, COUNT(m.movie_id) AS
   number_of_movies
   FROM genres g
   JOIN movie_genres mg ON g.genre_id = mg.genre_id
   JOIN movies m ON mg.movie_id = m.movie_id
   LEFT JOIN ratings r ON m.movie_id = r.movie_id
   GROUP BY g.genre_name
   ORDER BY average_imdb_rating DESC, number_of_movies DESC;
   ```

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 that we had during the design of this application was finding a dataset with all of our desired columns. Additionally, cleaning the dataset up and writing our load-data.sql to get our data into the database was also challenging. This was because we had many fields that we not in the right format as well as many unnecessary fields.

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 the time, we would have liked to explore more recommendation features such as creating an algorithm that took into account how the user rated certain actors, directors, and genres for different movies. Then, we could recommend movies that are more likely to be better fits for the user.

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: Ishaan Mantripragada
Partner 1 Responsibilities and Reflection:

- My specific responsibilities were to make sure the user interface was functioning properly., which included showing the right movies based on what the user was looking for. I also focused on the implementation regarding browsing by movies, genres, directors, etc. Overall 8-9 hours were spent working on this project.

Partner 2 Name: Olivia Xu
Partner 2 Responsibilities and Reflection:

- My specific responsibilities had to do with data cleanup and loading the dataset into the database. Additionally, I focused on the schema designs and user flowcharts, reflection.pdf and README.txt. Overall, 8-9 hours were spent in total working on this project.

Partner 3 Name: Rohan Jha
Partner 3 Responsibilities and Reflection:

- My specific responsibilities were to implement a majority of the coding. This included the DDL implementation, app.py, queries.sql, etc. Overall, 8-9 hours were spent working on the project.

Regardless of our individual responsibilities, we all worked together on all parts of the project so that each member was aware and understood every part of the project. We also collaborated with each other whenever one of us were stuck or had debugging issues.

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:
The final project overall was very helpful in solidifying the concepts we learned in class into a real world application. In the future, it might be helpful to students if we had more checkpoints throughout the term so that there isn't a huge amount of work left to do for the final project when it is finals week.