CS3200

Final Report

Professor Durant

June 21, 2023

Elissa Alarmani
Pauline Liu
Sheena Kaw

FantasticFilms: IMDB's Top 250 Movies

READ ME

Step 1: Prerequisites

Make sure you have: Python 3. You can check by running “python3 --version” in your terminal window.

It is ideal to use a virtualenv or equivalent to manage the downloaded requirements.

Step 2: Required Python Modules

They are all included in the requirements.txt file. Direct into the project folder and you can download all the dependencies with the command: **pip install -r requirements.txt**

If you would like to rather install the requirements individually:

pandas: `` `pip install pandas` ``

pymysql: `` `pip install pymysql` ``

Step 3: Import Database Dump

Create a MySQL connection in MySQL workbench.

Download the fantasticfilmsdatabasedump.sql file

Do this in MySQL: **Server > Data Import > Import from Self-Contained File >** select the fantasticfilmsdatabasedump.sql file > **Dump Structure and Data > Start Import**

Once the import has completed, the tables in `imdb` should contain tuples.

Note: Database dump file contains stored procedures as well as the created schema and data.

Step 4: Running the project

If you decide to use an IDE, run the main.py file. Otherwise, you can run the application in the terminal by navigating into the project directory/folder. Then run:

python3 main.py

Enter your MySQL username and password and follow the instructions in the command line!

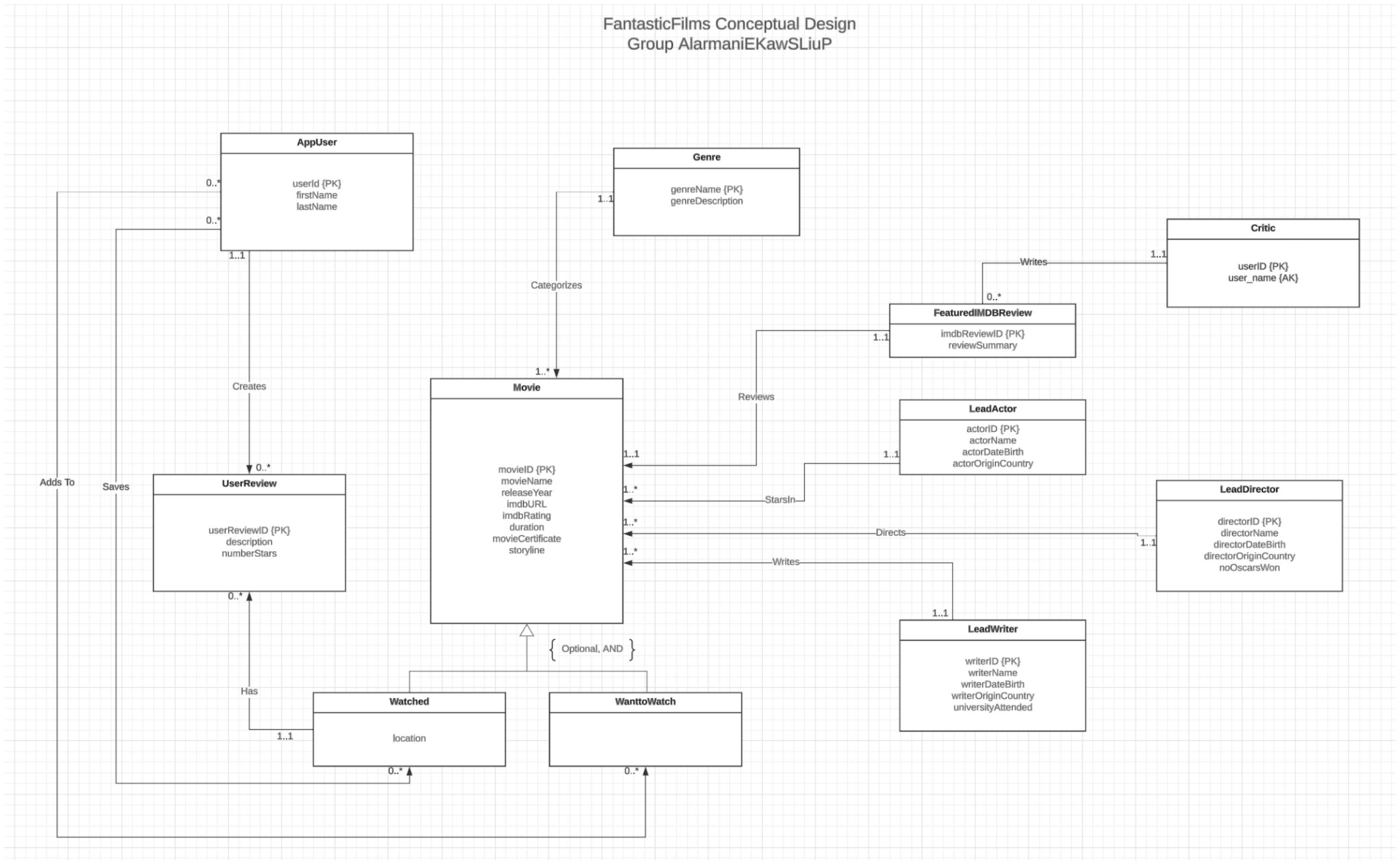
Technical Specifications

FantasticFilms is created using a Python script that interacts with a MySQL database to perform various search operations on movies stored in the database, as well as create, update, and delete user information and impressions of movies in the database.

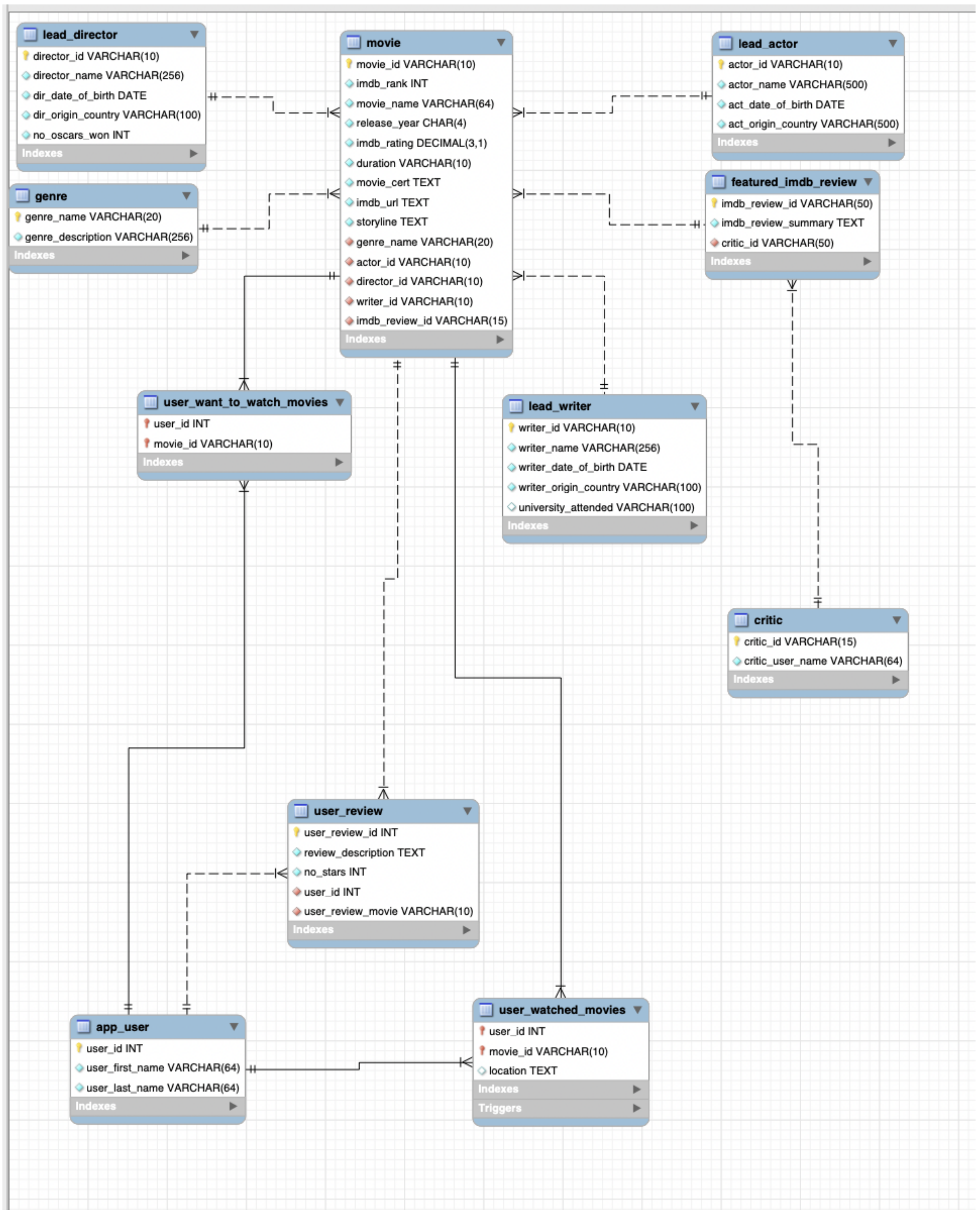
The script connects to a MySQL database using the pymysql library and prompts the user for a MySQL username and password. It assumes the existence of a database named "imdb" with relevant tables (the user is able to access the “imdb” database by importing the database dump).

Additionally, the data for the Top 250 IMDB was collected from a Kaggle dataset, “IMDB Top 250 Movies” (https://www.kaggle.com/datasets/karkavelrajaj/imdb-top-250-movies) which scraped the IMDB website.

Final Conceptual Design (UML)



Logical Design



User Flow

When a user first opens FantasticFilms, they are prompted for a MySQL username and password. The application then displays a welcome message and asks for the user's user ID, and if the user is new, asks them to type 'NEW'. If they are new user they are prompted for a first and last name and then given a user ID.

Then, the user is brought to the homepage, which has 5 options.

1. Search through the movies, command: **'search-movies'**.
2. View the list of all top 250 films, command: **'view-all-movies'**.
3. View/edit the user's watched list and their reviews for movies, command: **'view-watch-list'**.
4. View/edit the user's want to watch list, command: **'view-want-to-watch-list'**.
5. Quit the application; command: **'quit'**.

Command: **'search-movies'** → action: user is displayed the search options and additional information options

- Command: **'search-by-genre'**, enter genre name → action: display movie IDs and movie names of movies in that genre; if invalid requery
- Command: **'search-by-year'**, enter year → action: display movie IDs and movie names from that year, if invalid requery
- Command: **'search-by-movie-name'**, enter movie name → action: display the corresponding movie ID and movie name, if invalid requery
- Command: **'search-by-actor'**, enter actor name → action: display movie ID and movie names that star that actor, if invalid requery
- Command: **'search-by-director'**, enter director name → action: display movie ID and movie names with that director, if invalid requery
- Command: **'search-by-writer'**, enter writer name → action: display movie ID and movie names with that writer, if invalid requery
- Command: **'get-movie-info'**, enter movie ID → action: display IMDB review ID, writer ID, director ID, actor ID, genre name, movie ID, imdb rank (out of 250), movie name, release year, IMDB rating, duration, movie certification, IMDB URL, storyline, genre description, actor name, actor date of birth, actor origin country, director name, director date of birth, director origin country, number of oscars won by the director, writer name, writer date of birth, writer origin country, the university the writer attended, IMDB review summary, and critic ID for that movie, if invalid requery
- Command: **'get-genre-info'**, enter movie ID → action: display the genre name and genre description of the genre of that movie, if invalid requery
- Command: **'get-director-info'**, enter movie ID → action: display director ID, director name, director date of birth, director origin country, and number of oscars won by the director of that movie, if invalid requery

- Command: **‘get-actor-info’**, enter movie ID → action: display actor ID, actor name, actor date of birth, and actor origin country of the lead actor for that movie, if invalid requery
- Command: **‘get-writer-info’**, enter movie ID → action: display writer ID, name, date of birth, and university attended for the lead writer of that movie, if invalid requery
- Command: **‘get-top-review’**, enter movie ID → action: display critic ID, IMDB review ID, IMDB review summary, and critic username for the top IMDB review for that movie, if invalid requery
- Command: **‘help’** → action: display search & additional info options
- Command: **‘quit’** → action: return to homepage
- Command: **‘back’** → action: return to homepage

Command: **‘view-all-movies’** → action: display the movie ID and movie name of all movies in the database, automatically return to homepage

Command: **‘view-watch-list’** → action: display user’s watched list (movie ID, movie name, location where the user watched the movie (ex: ‘cinema’), user’s review of the movie, the number of stars the user gave the movie, offer the option to add/delete movies from the list

- Command: **‘add’**, enter movie name, enter location where the user watched the movie → action: adds movie to the user’s watched list, deletes movie from the user’s want to watch list if on the list, offers the option to add/delete additional movies
- Command: **‘delete’**, enter movie name → action: deletes movie from the user’s watched list, offers the option to add/delete additional movies
- Command: **‘none’** → action: displays watch list again, offers the option to write, update, delete, or read this user’s reviews
    - Command: **‘write’**, enter movie name, enter number of stars, enter review → action: saves this user’s review in the database
    - Command: **‘update’**, enter movie name, enter number of stars, enter review → action: update user’s most recent review for this movie
    - Command: **‘delete’**, enter movie name → action: deletes the user’s oldest review for this movie
    - Command: **‘read’**, enter movie name → action: displays all of the user’s reviews for this movie
    - Command: **‘read all’** → action: displays all of the user’s movie reviews
    - Command: **‘none’** → action: return to homepage

Command: **‘view-want-to-watch-list’** → action: displays user’s want to watch list (movie ID and movie name) and the option to add/delete movies from the list

- Command: **‘add’**, enter movie name → action: adds movie to the user’s want to watch list, offers the option to add/delete additional movies
- Command: **‘delete’**, enter movie name → action: deletes movie from the user’s want to watch list, offers the option to add/delete additional movies

- Command: **‘no’** → action: displays user’s want to watch list again, returns to the homepage

Command: **‘quit’** → action: closes the application

Lessons Learned

The creation of a database is a complex process that requires careful planning and attention to detail. Throughout this project, we have learned several lessons that allowed us to improve our models and the effectiveness of the project.

One important takeaway from this project is the significance of revision and editing when creating a database. After initially creating our database, we quickly learned the importance of continuous refinement. We revisited our models to modify them and ensure that they aligned well with our project’s requirements. This approach allowed us to identify any inefficiencies or shortcomings in the initial design. As a result, we were able to ensure our database was robust and streamlined. This experience also allowed us to become familiar with MySQL workbench. By creating a relational database for the top 250 movies we gained insight on storing and organizing data efficiently using relationships, attributes, primary keys, and foreign keys.

Another important lesson learned was the meticulous nature of the data collection process. Accuracy is important and we had to validate and cross-check our work during this process. We retrieved our data from the Kaggle dataset that scraped the Top 250 Movies from IMBD, however, the dataset was not comprehensive enough for our project so we had to manually search for additional information to populate certain columns in our database. This included finding information on the writers, directors, and actors of movies and recording their date of birth, country of origin, university attended by the writers, and the number of Oscars won by the directors. The formatting of the data had to be consistent throughout the process, for example, the formatting of dates and whether to numerically or alphabetically store the count of Oscars. By ensuring consistency and accuracy of our data we were able to build a reliable database.

Considering the user flow prior to the construction of the application was a vital step in our project, and we learned about the importance of planning for users’ needs. Understanding the potential actions and desires of the users helped us to determine which procedures and triggers were necessary to implement. By anticipating user requirements, we were able to build a database that better caters to users and allows for seamless application development. To continue, connecting the database to Python allowed us to become familiar with establishing a connection between Python and MySQL. We learned how to perform various operations on the database programmatically, and how to extract meaningful information from the collected data.

Another obstacle we encountered during the project was the difficulty in designing an application driven by user input. We had to account for the various issues that could arise when users were inputting data. Ensuring data entry was completed correctly by the user was a major concern and became a large focus of our front-end code. We had a substantial portion of our front-end code that was dedicated to handling and querying data to address errors caused by misspellings and other such user mistakes. This process taught us the importance of error handling and creating a user interface that minimizes the potential of errors, to begin with. For example, we gave very explicit directions in our user prompts to try and prevent user errors. Additionally, this allowed us to hone our Python skills by developing an input-driven script. Building the script that interacted with the user's input required careful consideration and the creation of intuitive prompts for users, as well as the ability to write clear and concise code. We gained valuable knowledge on developing scripts with a team that can be used in future projects that involve user interfaces and input processing.

Future Work

We feel that there are many areas where this project could be improved upon and expanded. One of our hopes going into this experience was to create an attractive, polished front end. Although we are proud of our front end, we hope to create a more complex and aesthetically pleasing design. We have looked into using the React.js framework to create a JavaScript front end as well. Also, we would like to add some additional information to the database tables such as more information about the rest of the cast, writers, and crew, as currently we only have information on the director, lead actor, and lead writer. Another piece of information we hope to add to the database is where a user can watch a movie, such as which streaming platforms offer it. It can be hard to find where to watch a movie, and so we believe adding this information to FantasticFilms would be very helpful to our users. However, as data collection was one of the more time-consuming aspects of this project, it was not possible to include all the information that we wanted to in the database for the scope of this class. The 'search' component of our project could also be expanded, as currently, users have to spell an actor or writer's name exactly as it is in the database to find the movies that they are in.

We also hope to scale this project up and allow users to read each other's reviews, instead of just their own. However, as we chose to host this project locally, it was not possible to implement this feature. Additionally, making this change to the application would require concurrency control and the use of more complex techniques to handle multiple users. Still, we see more potential for this project as a social and collaborative application instead of an individual movie tracker, which is what it is currently limited to. This is not to say that we do not plan to use the database for the latter purpose; we believe that being able to see one's past impressions of movies be very interesting, and we plan to record our thoughts with this application as we continue our goal of watching all 250 movies.

Lastly, we have the idea to add a 'recommendation' feature to the application, perhaps by incorporating machine learning techniques such as clustering to find movies within the database that are similar. When a user indicates they want to watch a certain movie, similar movies could be suggested to them. We could also apply machine learning techniques to the user and their movie lists and reviews, to try and predict what kind of movies they generally are interested in and what kind of movies they typically give high ratings. Additionally, natural language processing could be used to analyze user reviews to try and gain some insight into their impressions beyond their numerical star rating.