The Big Picture

Kevin Bennett, David Daly-Coll, Giffin Davis

1. Introduction

1.1 Application Motivation

We all love movies. However, we all love different movies. Finding the right, next movie to watch can be a challenge. With so many different choices it can be hard to find a new movie to watch that interests you and is worth dedicating several hours to.

There are tens of thousands of movies. In fact, there were more than 9,000 movies released in 2020 alone[1]. With so many choices, and many of them streaming at one's fingertips, it can be a daunting task to select a movie to watch. Similarly, each movie can have hundreds, if not thousands, of reviews. Reading all of these reviews would likely take longer than watching the movie.

A movie-watcher needs the ability to define certain criteria they are looking for in a movie, such as an actor they would like to watch, or a film by a director they enjoy. In addition, the movie watcher wants to be able to quickly search for movies above a certain average rating because no one likes to waste time watching a bad movie (though there are those who intentionally look for bad movies and they will find this equally useful). Finally, the movie watcher would want to be able to include movies with reviews that contain certain keywords (e.g. "heart pounding" or "mediocre"). All of these features combined will save the movie watcher time in selecting a movie to watch and get them to the opening credits sooner.

1.2 Description of the Application

The Big Picture application allows the movie watcher to search for movies based on these criteria:

- Average aggregated review score
- Featuring a specific actor
- Directed by a specific director
- Containing reviews with certain keywords (e.g. "Mediocre")
- Year of Release
- Or some combination of the above

These criteria were selected because they were determined to be the most relevant when selecting a movie to watch. For example, it is very typical to want to watch movies above a certain rating because no one likes to feel like they wasted their time on a bad movie. Actors or actresses in the movies are equally important to one's decision to watch a movie because of one's personal preferences. The same is true of the film's director. Selecting movies that have

certain words in other reviews of the movie can be equally helpful to exclude movies that others describe as “boring” or if a parent is trying to avoid movies with “excessive violence.” Finally, the year is useful when one is trying to decide between a new movie or a classic.

The user of the application will be able to use any or all of the above criteria to perform a search to find the movie that fits their interests. And finally, after they have watched the movie, they will have the ability to add the movie to the database and their own review of the movie to help others in their movie choices.

1.3 Report Organization and Task Assignments

In the subsequent sections we will go into detail on the overall database implementation, broken out by the key elements of the design, and then the user interface, and descriptions around the functionality the user will see, and then finally some concluding remarks.

Over the course of the project, all three group members contributed to all elements save for the items called out below which saw the group member mentioned be the primary contributor to that area.

- Implementation of UI:
  - Griffin Davis owned the design and creation of the front-end, as well as all write-ups around its technical details and behavior.
- B+ tree:
  - File Scan:
    - Kevin Bennett tackled the first step needed for checkpoint 3, which was being able to successfully read in the ‘students’ file.
  - Insert:
    - Griffin Davis implemented the Insert portion of the B+ tree algorithm.
  - Search, Delete:
    - David Daly-Coll implemented both the search and deletion logic.
- Dataset Cleanup and Population of Database:
  - Griffin Davis and David Daly-Coll completed the tasks of retrieving the raw datasets, trimming them down and manipulating them as described later in the report, and populating the database that the user interacts with.
- Evaluation:
  - Kevin Bennett owned the creation of a robust evaluation plan for both the database and the UI.

2. Implementation

2.1 Description of the System Architecture

Our application uses the Electron[2] framework which allows us to create a cross platform application that’s easily bundled into distributables using Electron-Forge[3]. Using this framework also allows us to use web-based languages for our entire stack, including HTML, CSS, and

Javascript for both the front end and the backend. We are using a built in Node.js[4] server for handling the “back end” and the queries to our SQLite database. SQLite was chosen because it is very lightweight relative to other options and has good integration with the framework.

2.2 Description of the Dataset

Our database uses two datasets from Kaggle[5]. The first dataset contains a list of 17,000+ movies and key attributes. The second dataset contains over one million reviews for the movies in the first dataset.

Movie Dataset:

The movie dataset required various manipulation steps:

- Multiple columns were removed as extraneous (authors, streaming date, etc) or to cut down on the overall size (critics consensus)
- The directors and actors columns, which contained a delimited-list of values in each cell, were split into one row per actor/director per movie
- Leading and trailing whitespace was removed from the values that came from the split columns

This dataset then formed the basis for three of our entity sets- Actor, Director and Movie.

Review Dataset:

The review data required far less changes to the published dataset. What little was changed amounted to the removal of extraneous columns - publisher, review_type, review_date, critic_name and top_critic.

Both datasets have a rotten_tomatoes_link column, which is unique to each movie, which ensured we could join the two datasets.

2.3 ER Diagram

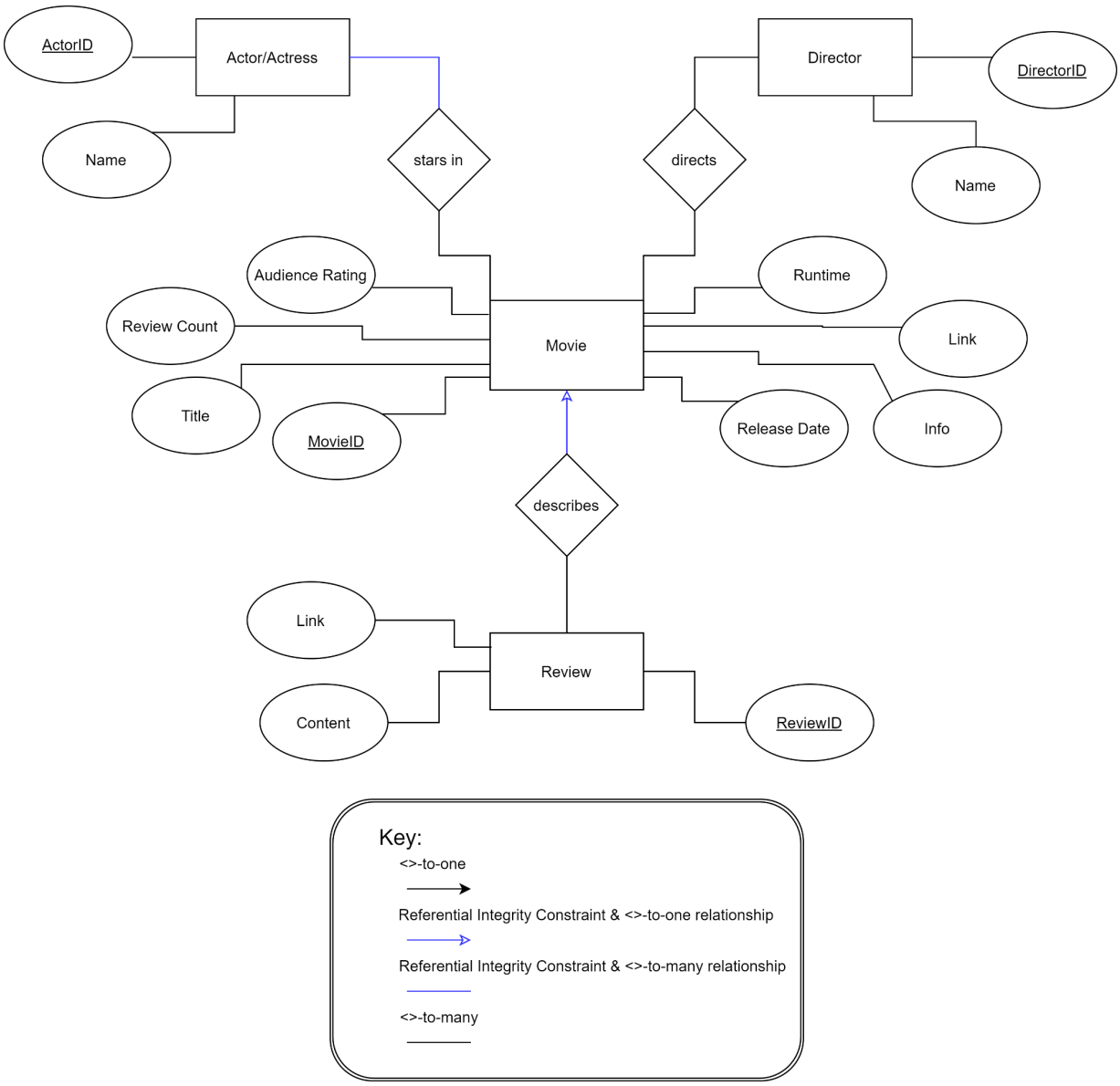

Figure 1. ER Diagram

2.4 Relational Model

1. Review (ReviewID, Link, Content)
    a. ReviewID is the primary key
    b. FDs: ReviewID -> Content, Link
    c. Normal Form: BCNF as the non-trivial FDs have a candidate key on the left-hand side (ReviewID).

2. Movie (MovieID, Title, Runtime, Rotten Tomatoes Rating, Review Count, Release Date, Link, Info)
    a. MovieID is the primary key
    b. FDs: MovieID -> Title, Runtime, Rotten Tomatoes Rating, Review Count, Release Date, Link, Info
    c. Normal Form: BCNF as the non-trivial FDs have a candidate key on the left-hand side (MovieID).

3. Actor/Actress (ActorID, Name)
    a. ActorID is the primary key
    b. FDs: ActorID -> Name
    c. Normal Form: BCNF as any two attribute relation is BCNF

4. Director (DirectorID, Director Name)
    a. DirectorID is the primary key
    b. FDs: DirectorID -> Name
    c. Normal Form: BCNF as any two attribute relation is BCNF

5. StarsIn (MovieID,ActorID)
    a. MovieID + ActorID is the composite key
    b. Normal Form: BCNF as any two attribute relation is BCNF

6. DirectedBy (MovieID,DirectorID)
    a. MovieID + DirectorID is the composite key
    b. Normal Form: BCNF as any two attribute relation is BCNF

7. Describes (ReviewID,MovieID)
    a. ReviewID is the primary key
    b. FDs: ReviewID -> MovieID
    c. Normal Form: BCNF as any two attribute relation is BCNF

2.5 Implementation: Description of the Prototype

The initial landing page of our application gives the user the ability to choose between finding a movie in the current database, or adding a movie to the database. By selecting either of these options, a similar screen will be shown with the relevant data points for either adding or finding a movie. On both of these screens, there are two buttons that give the user access to some of the high level information in the database. These buttons are the “Top Movies” and “Rating By Year” buttons.

Selecting the Top Movies button shows the user the top movies, based on rating, with over 1,000,000 users that have rated the movie. It also gives the user the ability to click on one of the movies and drill down into some more detailed information about that particular movie as well as the reviews of the movie.

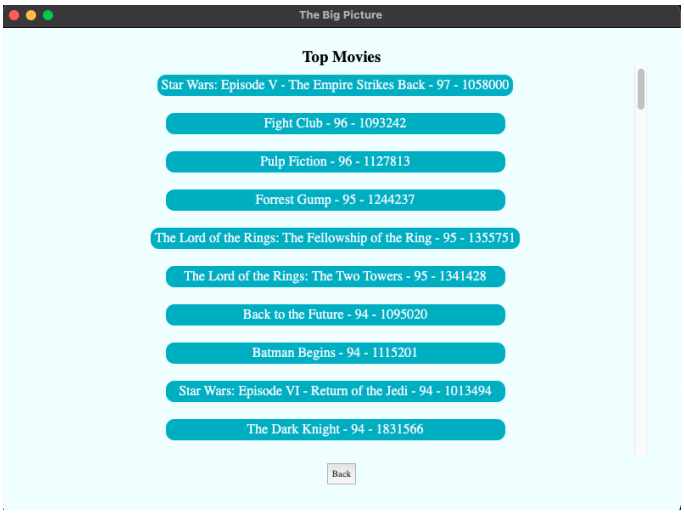

Figure 2. Top Movies button results

The Ratings By Year display will give a user insight into the average rating of movies for a particular year. We can see that in recent years, the ratings are quite low and the overall trend of movie ratings is going down over the last few years.

Figure 3. Rating By Year breakdown display

On the “Find a Movie” page, the user is able to search for movies by title, release year, runtime, average rating (equal to, greater than, or less than), actor, director, and of course by the content of the user reviews. The user is able to search by any combination of the information on this page. When searching for a string in the title of a movie, anything less than three characters is treated as the first three characters of the title. If there are more than three characters, any matching substring in the title will suffice. Upon selecting the relevant search criteria, the user will then be presented with a popup of the movies that match the criteria and the option to select any of those movies to view the relevant details.

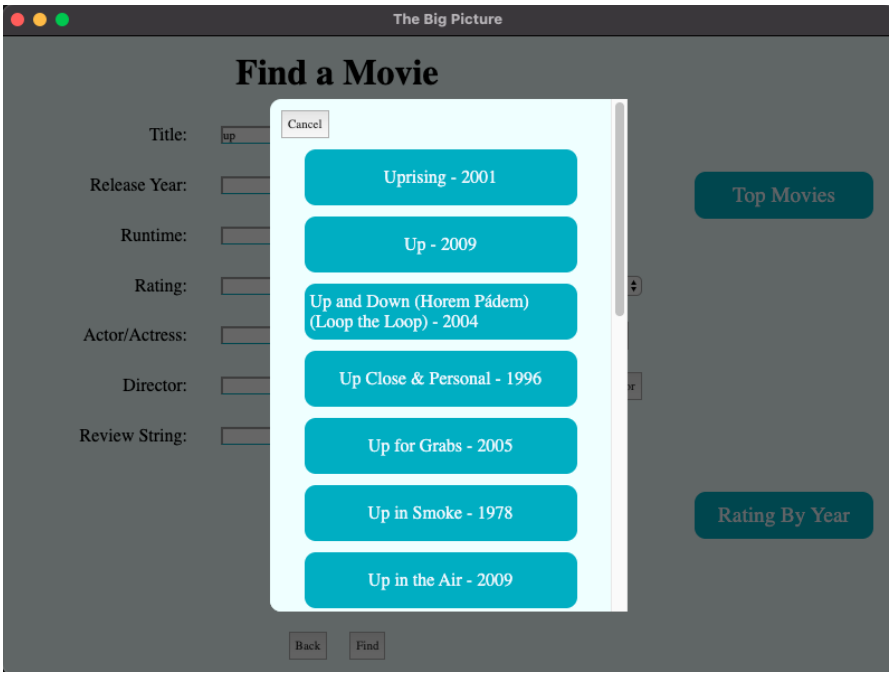

Figure 4. Find a Movie search results

Upon selecting the movie the user is most interested in, they will be able to see all of the relevant details appear on the screen. They’ll also see options for adding a review, adding an

actor, adding a director, and editing the other information relating to the movie. Selecting the Edit button will make the different fields editable in the view. Selecting the Add Actor or Add Director button will provide a popup to search for the actor/director that the user would like to add. The first result will always be an option to create a new actor/director with the exact name that was searched. A warning is thrown whenever this actor/director is selected to warn the user that they'll be creating a new record in the system. Selecting a review in the system will also allow the user to modify the content of the review.

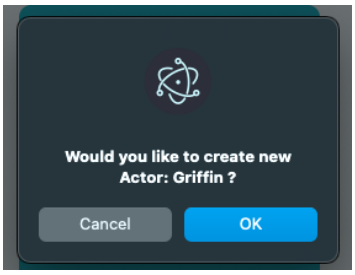

Figure 5. Display message to create a new Actor

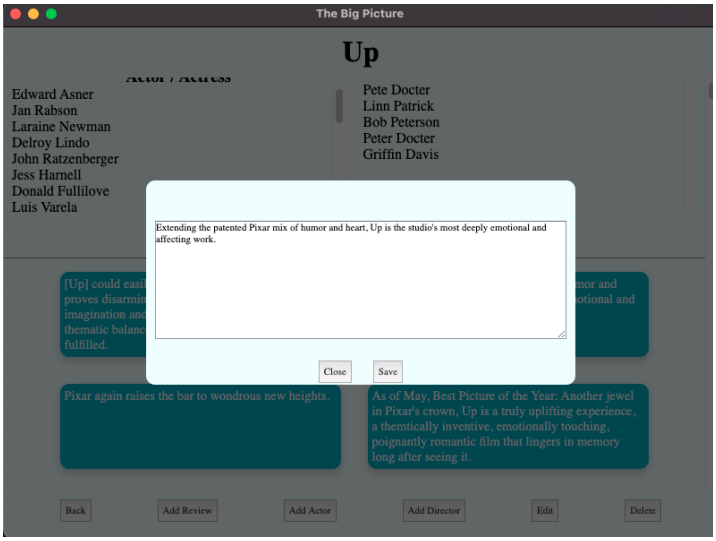

Figure 6. View information about a movie

Moving back to the main view, if the user chooses to add a movie, they can enter similar information such as the title, release date, runtime, rating, actor, and director. If they choose a title that already exists in the system, they will be stopped as it is a duplicate.

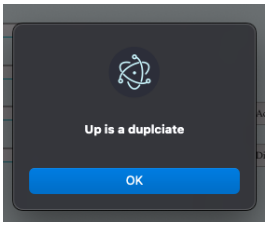

Figure 7. Duplicate movie display message

Once the user clicks add, they'll be brought to the main movie screen where they can edit any relevant information about the movie.

Figure 8. Edit movie details

Finally, the user has the option to delete a movie with the delete button at the bottom of the page, as well as delete an actor / director by using the right mouse button.

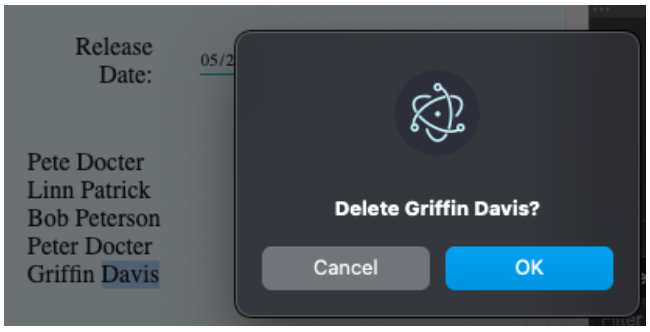

Figure 9. Delete actor / director confirmation prompt

2.6 Evaluation

Our approach to testing and evaluation of the application was split into two separate focuses. The first was to confirm the view and UI of the application functioned logically and as expected. After confirming that the UI of the application performed as expected, we verified that queries performed by the user returned the correct information (and did not return inaccurate information) and updates made by the user updated the dataset correctly and did not cause any data corruption or losses.

2.6.1 Testing the Functionality of the UI

The driving philosophy behind these tests was that if a button claimed to do “A”, only “A” should be performed. With that in mind, these specific tests were used to evaluate the UI of the application.

1. **Input fields should only accept appropriate values**
    a. Numeric fields - Rating, Runtime, Year
    b. String fields - Title, Actor/Actress/, Director, Review
    c. In a numeric input field attempt to enter a numeric value
        i. Success - numeric value is accepted
        ii. Failure - numeric value is not accepted
    d. In a number input field attempt to enter an alphabetic value
        i. Success - alphabetic value is not accepted
        ii. Failure - alphabetic value is accepted
    e. In a string input field attempt to enter a numeric or alphabetic values
        i. Success - all input values accepted
        ii. Failure - any input value not accepted

2. **Moving backward and forward from the main page should maintain user input in fields**
    a. In each input field, enter a value, then use the back button, and then return to the same page
        i. Success - input values being are maintained
        ii. Failure - Input values are cleared

3. **Sidebar buttons display when expected**
    a. On the initial loading page the sidebar buttons should not display; when searching for or adding a movie, the sidebar buttons should display
    b. Test loading that initial landing page and navigating to the “Find a Movie” and “Add a Movie” buttons. Use the back button on the “Find” and “Add” pages to move back to the landing page and move forward again
        i. Success - on landing page, sidebar buttons do not display
        ii. Failure - on landing page, sidebar buttons display
        iii. Success - when searching for/adding a movie, sidebar buttons display
        iv. Failure - when searching for/adding a movie, sidebar buttons display

4. **Sidebar button functionality**
    a. Using the button on the sidebar displays and groups data appropriately
        i. Select the “Top Movies” button
            1. Success - a breakdown of top movies is displayed
            2. Failure - a breakdown of top movies is not displayed OR movies that should not be included in the top movies list are displayed
        ii. Select the “Ratings by Year” button
            1. Success - a breakdown of the ratings of movies is displayed AND the ratings are grouped into years
            2. Failure - a breakdown of the movies is not displayed OR the ratings are not grouped by year

5. **Find a Movie versus Add a Movie**
    a. Using the “Find a Movie” button should *return* values. From landing page, use the “Find a Movie” button
        i. Search for any criteria (ie Year of 2004)
            1. Success - movie titles are displayed (note - no need to confirm that the correct movies are returned; that will be evaluated elsewhere)
            2. Failure - no movie titles are displayed OR values are added to the database
    b. Using the “Add a Movie” button should *add* values to the database. From the landing page, use the “Add a Movie” button
        i. Add the following movie:
            1. Title: “zzzMovie Titlezzz”
            2. Year: “9999”
            3. Actor/Actress: “zzzActor Namezzz”
            4. Director: “zzzDirector Namezzz”
        ii. Confirm the values are added to the appropriate tables
            1. Success - each table is updated with appropriate values
            2. Failure - Tables are not updated OR a search is performed instead of an addition

2.6.2 Testing the validity of queries and updates to the dataset

After verifying the UI of the application performed as designed, the next step was to verify the results of queries of or additions to the dataset. Outputs from query from the application should match directly with queries performed on the database and with manual review of the source CSV files.

1. **Given an input of a certain year, only movies of that year should be returned**
    a. Input a year (eg 2004) and confirm results are only movies from that year (compare results with CSV files or database tables)
        i. Success - all movies from input year are returned

ii. Failure - movies not from input year are returned
iii. Failure - at least one movie from input year is not returned

b. Similar tests should be performed on other numeric input values

2. **Adding a duplicate movie (ie a movie already in the database) should not add another entry to tables**
    a. Attempt to add a movie already in the database (eg “Pirates of the Caribbean: The Curse of the Black Pearl”)
        i. Success - warning indicating this movie already exists in the database and prompt to add another movie with the same title
        ii. Failure - no prompt to indicate this is a duplicate entry

3. **Searching for a movie containing an input string in the title should return all movies with that string as a substring in the title**
    a. Attempt to search for a substring in the title of a movie (eg “Heart” or “ing”)
        i. Success - all movies containing the input substring in the title are returned (compare with the results of a query on the tables directly eg WHERE Title like “%ing%”)
        ii. Failure - movies containing the string or substring in the title are missing from the results
        iii. Failure - movies not containing the string or substring in the title are returned by the query
    b. Similar tests should be performed on other string input values

4. **Adding a new movie to the database without the minimum required information will be rejected**
    a. Attempt to add a movie without entering a title
        i. Success - warning/error message stating that required info is missing from the input
        ii. Failure - the subset of information is added to the database
        iii. Failure - a movie with a NULL title is added to the database
        iv. Failure - no warning/error message is presented

5. **Adding a new director when one is not found**
    a. When searching for a director, if the director is not found in the database, a prompt will be displayed to add a new director
        i. Success - prompt displays asking if a new director should be added; also the database is updated (verify that the director is added to the database)
        ii. Failure - no prompt displays when adding a director not already in the database
        iii. Failure - prompt displays but the director’s information is not added to the database
    b. The same will be true for actors/actresses

3. Conclusion

With respect to our work on this project:

> Our biggest hurdle over the course of this project dealt with getting the two datasets into a clean enough form. The datasets we had planned to use in checkpoint 1 didn't contain a link between the primary movie dataset and the review dataset(s), making it impossible to actually implement the database.
>
> While the new Kaggle datasets did allow us to create our database as described above, it did not come without its drawbacks. Primary among them was that the actor and director names were not consistently formatted (e.g. first name of David also appearing as Dave), which means that the same actor/director can appear as multiple distinct strings. This makes it difficult to get full accuracy with queries centered around either name.
>
> The struggle to wrangle a target dataset helped illustrate what was mentioned during the very first lecture of this course and that was to be cautious about dataset selection.

As far as the assignment itself:

> We wound up spending far more time on the standalone B+ tree programming assignment than checkpoint 4 in the leadup to the week that both were due. One thing that might help further would be to add new checkpoints, with more discrete goals to help spread the workload out, or to move the B+ tree assignment further from the 4th checkpoint. We recognize that scheduling during the very short summer term can make this even more challenging to try and accommodate.
>
> Between selecting our data, planning the database, and implementation, this proved to be a great challenge that helped turn the abstract lessons learned in class into practice.

4. References

1. https://www.imdb.com/search/title/?year=2020&title_type=feature&
2. https://www.electronjs.org/
3. https://www.electronforge.io/
4. https://nodejs.org/en/
5. https://www.kaggle.com/stefanoleone992/rotten-tomatoes-movies-and-critic-reviews-dataset/version/1?select=rotten_tomatoes_movies.csv