SUNRISE TEAM
IMDB TV SHOW SEARCH ENGINE
THE NEXT BINGE

Raheel Bhimani, Starth Marie Daitol, Muizz Mullani, Lingwei Wu

University of Pennsylvania

Team: Sunrise
Title of Website: *The Next Binge*
List of Members: Raheel Bhimani, Starth Marie Daitol, Muizz Mullani, Lingwei Wu

1. INTRODUCTION

Many of us have felt the sense of emptiness (or perhaps fury) after binge-watching the final season of Game of Thrones, and wondered what's next. Our team members are of no exception. A common interest in solving this problem and the fact IMDB database offers many records for us to manipulate led us to settle on the function of our website.

IMDB database offers a wide range of information on different TV shows and movies. For the purpose of our website, we pre-processed the database using Google Colab by only selecting data that is related to TV shows and the functionality of our website.

Website *The Next Binge* will recommend TV shows based on user selections of various information such as Genre, Director, Rating, etc. A user is provided a wide range of search tools to explore a rich database of TV shows. It also provides a Watch-list function to allow a user to save the TV shows they are interested in watching.

2. ARCHITECTURE

List of technologies used: MySQL, React, Node.js, Pandas, Google Colab.

Description of system architecture/application: Utilizing the React platform, our system architecture follows that set-up and layout. A picture of that is enclosed below.

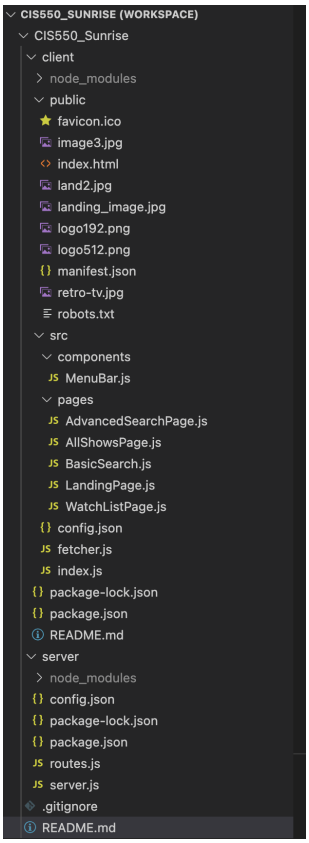

Fig. 1. An image of system architecture

3. DATA

We used these tables to extract information related to TV series. Then we built our database by importing processed data. Basic Name was used to look up any IMDb person's information. For example, when we want to look up TV shows directed by certain director, we joined the Basic Name table with the Director table using person Id (pid renamed from nconst. Similarly, the Titles table is used to tie information for Genres, Additional Title, and Principal Title tables by using title Id (tid renamed from tconst).

These datasets are related by Foreign Key referencing Primary key.
Datasets: https://www.imdb.com/interfaces/

Basic Name (name.basics.tsv):
This dataset contains basic information about the cast and crew including the following columns:

- nconst (string) - alphanumeric unique identifier of the name/person
- primaryName (string)– name by which the person is most often credited
- birthYear – in YYYY format
- deathYear – in YYYY format if applicable, else null
- primaryProfession (array of strings)– the top-3 professions of the person
- knownForTitles (array of tconsts) – titles the person is known for

	nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
0	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	tt0053137,tt0031983,tt0050419,tt0072308
1	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt0071877,tt0037382,tt0038355,tt0117057
2	nm0000003	Brigitte Bardot	1934	\N	actress,soundtrack,music_department	tt0057345,tt0056404,tt0054452,tt0049189
3	nm0000004	John Belushi	1949	1982	actor,soundtrack,writer	tt0077975,tt0072562,tt0080455,tt0078723
4	nm0000005	Ingmar Bergman	1918	2007	writer,director,actor	tt0083922,tt0050986,tt0069467,tt0060827

Fig. 2. An image of table Basic Name

	nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
count	11410090	11410090	11410090	11410090	9005439	11410090
unique	11410090	8910896	507	445	20664	4745293
top	nm0000001	David Smith	\N	\N	actor	\N
freq	1	344	10866261	11211635	2042431	2014013

Fig. 3. An image of table Basic Name Summary

Title Ratings (title_ratings.tsv):
This dataset contains the IMDb rating and votes information for titles including the following columns:

- tconst (string) - alphanumeric unique identifier of the title
- averageRating – weighted average of all the individual user ratings
- numVotes - number of votes the title has received

	tconst	averageRating	numVotes
0	tt0000001	5.7	1858
1	tt0000002	6.0	243
2	tt0000003	6.5	1628
3	tt0000004	6.0	158
4	tt0000005	6.2	2455

Fig. 4. An image of table Title Ratings

	averageRating	numVotes
count	1.213990e+06	1.213990e+06
mean	6.927401e+00	9.984573e+02
std	1.395220e+00	1.666289e+04
min	1.000000e+00	5.000000e+00
25%	6.200000e+00	1.100000e+01
50%	7.100000e+00	2.400000e+01
75%	7.900000e+00	9.400000e+01
max	1.000000e+01	2.543259e+06

Fig. 5. An image of table Title Ratings Summary

Principal Title (title_principals.tsv):
This dataset contains the principal cast/crew for titles including the following columns:

- tconst (string) - alphanumeric unique identifier of the title
- ordering (integer) – a number to uniquely identify rows for a given titleId
- nconst (string) - alphanumeric unique identifier of the name/person
- category (string) - the category of job that person was in
- job (string) - the specific job title if applicable, else null
- characters (string) - the name of the character played if applicable, else null

	tconst	ordering	nconst	category	job	characters
0	tt0000001	1	nm1588970	self	\N	["Self"]
1	tt0000001	2	nm0005690	director	\N	\N
2	tt0000001	3	nm0374658	cinematographer	director of photography	\N
3	tt0000002	1	nm0721526	director	\N	\N
4	tt0000002	2	nm1335271	composer	\N	\N

Fig. 6. An image of table Principal Title

	ordering
count	4.884690e+07
mean	4.579780e+00
std	2.776242e+00
min	1.000000e+00
25%	2.000000e+00
50%	4.000000e+00
75%	7.000000e+00
max	1.000000e+01

Fig. 7. An image of table Principal Title Summary

Episode Title (title_episode.tsv):
This dataset contains information about tv episodes including the following columns:

- tconst (string) - alphanumeric identifier of episode
- parentTconst (string) - alphanumeric identifier of the parent TV Series
- seasonNumber (integer) – season number the episode belongs to
- episodeNumber (integer) – episode number of the tconst in the TV series

	tconst	parentTconst	seasonNumber	episodeNumber
0	tt0020666	tt15180956	1	2
1	tt0020829	tt15180956	1	1
2	tt0021166	tt15180956	1	3
3	tt0021612	tt15180956	2	2
4	tt0021655	tt15180956	2	5

Fig. 8. An image of table Episode Title

	tconst	parentTconst	seasonNumber	episodeNumber
count	6501332	6501332	6501332	6501332
unique	6501332	169033	379	15717
top	tt0020666	tt12164062	1	\N
freq	1	17093	3209439	1385789

Fig. 9. An image of table Episode Title Summary

Crew Title (title_crew.tsv):
This dataset contains the director and writer information for all titles in IMDb including the following columns:

- tconst (string) - alphanumeric unique identifier of the title
- directors (array of nconsts) - director(s) of the given title
- writers (array of nconsts) – writer(s) of the given title

	tconst	directors	writers
0	tt0000001	nm0005690	\N
1	tt0000002	nm0721526	\N
2	tt0000003	nm0721526	\N
3	tt0000004	nm0721526	\N
4	tt0000005	nm0005690	\N

Fig. 10. An image of table Crew Title

	tconst	directors	writers
count	8691512	8691512	8691512
unique	8691512	841345	1163959
top	tt0000001	\N	\N
freq	1	3735601	4252259

Fig. 11. An image of table Crew Title Summary

Basic Title Data (title_basics.tsv):
This dataset contains basic information for all titles in IMDb including the following columns:

- tconst (string) - alphanumeric unique identifier of the title
- titleType (string) – the type/format of the title (e.g. movie, short, tvseries, tvepisode, video, etc)
- primaryTitle (string) – the more popular title / the title used by the filmmakers on promotional materials at the point of release
- originalTitle (string) - original title, in the original language
- isAdult (boolean) - 0: non-adult title; 1: adult title
- startYear (YYYY) – represents the release year of a title. In the case of TV Series, it is the series start year
- endYear (YYYY) – TV Series end year. null for all other title types
- runtimeMinutes (int)– primary runtime of the title, in minutes
- genres (string array) – includes up to three genres associated with the title

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
0	tt0000001	short	Carmencita	Carmencita	0	1894	\N	1	Documentary,Short
1	tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	5	Animation,Short
2	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	4	Animation,Comedy,Romance
3	tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	12	Animation,Short
4	tt0000005	short	Blacksmith Scene	Blacksmith Scene	0	1893	\N	1	Comedy,Short

Fig. 12. An image of table Basic Title

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
count	8691512	8691512	8691504	8691504	8691512	8691512	8691512	8691512	8691502
unique	8691512	11	4040222	4059774	11	256	97	864	2303
top	tt0000001	tvEpisode	Episode #1.1	Episode #1.1	0	\N	\N	\N	Drama
freq	1	6501575	42486	42486	8359905	1119907	8603934	6336113	979155

Fig. 13. An image of table Basic Title Summary

Additional Title Information (title_akas.tsv):
This dataset contains additional information about the titles in IMDb including the following columns:

- titleId (string) - a tconst, an alphanumeric unique identifier of the title
- ordering (integer) – a number to uniquely identify rows for a given titleId
- title (string) – the localized title
- region (string) - the region for this version of the title
- language (string) - the language of the title
- types (array) - Enumerated set of attributes for this alternative title. One or more of the following: ”alternative”, ”dvd”, ”festival”, ”tv”, ”video”, ”working”, ”original”, ”imdbDisplay”. New values may be added in the future without warning
- attributes (array) - Additional terms to describe this alternative title, not enumerated
- isOriginalTitle (boolean) – 0: not original title; 1: original title

	titleId	ordering	title	region	language	types	attributes	isOriginalTitle
0	tt0000001	1	Кармен/іта	UA	\N	imdbDisplay	\N	0
1	tt0000001	2	Carmencita	DE	\N	\N	literal title	0
2	tt0000001	3	Carmencita - spanyol tánc	HU	\N	imdbDisplay	\N	0
3	tt0000001	4	Καρμενσίτα	GR	\N	imdbDisplay	\N	0
4	tt0000001	5	Карменсита	RU	\N	imdbDisplay	\N	0

Fig. 14. An image of table Additional Title

	ordering
count	3.095145e+07
mean	4.025148e+00
std	3.486326e+00
min	1.000000e+00
25%	2.000000e+00
50%	4.000000e+00
75%	6.000000e+00
max	1.770000e+02

Fig. 15. An image of table Additional Title Summary

4. DATABASE

Explanation of data ingestion procedure and entity resolution efforts: The original files as described were too large for them to be efficiently brought into the database. In the processing steps, we first begin with Basic Title and then limit that dataset to include on ’tvseries’. This significantly curtails the number of tuples within each table. From the Basic Title, we obtain Genres and for ER resolution bring out this 0:N relationship into its own table using tid to tie the tables together. This also involved dropping any null values for genre as the primary keys are tid and genre

Next, we take the Title Ratings file and keep only titles (tid) that are contained in the Titles table. Basically, we keep only ratings for ’tvseries’. Similarly, we truncate the Episode Title file to keep only ’tvseries’ episodes.

Next, we take Title Crew file and from it we extract (lift/explode) Directors and Writers into their own respective tables, preserving 3NF. Similar to Genres, we drop any null values to preserve the joint primary keys. Further, we only keep titles that are ’tvseries’ as determined by the Titles table.

The next part is getting information about the people themselves. This first begins with taking the Title AKAS file and exploding out Characters while keeping the rest of the attributes in Principal Title. Both tables then are truncated to keep to ’tvseries’ that we obtained in the Titles table. Additionally on Cast In, the character array that is provided is further needed to be processed before exploding to account for character set differences, spacing between characters, and font case. This helps prevent two tuples being identified as unique when they really are duplicates.

Finally, we take the list of people we obtained from the Title AKAS file and match it to the Name Basics file, cutting down the people significantly. Further, we explode out Professions and Known For attributes into its own tables, once again to keep 3NF. Additionally, we drop any null values so that joint primary keys are valid. See the final statistics of ingested information below followed by Fig. 16 for ER diagram.

Statistics:

- AdditionalTitles: 620,601 instances
- CastIn: 1,237,392 instances
- Directors: 96,765 instances
- EpisodeTitles: 6,322,928 instances
- Genres: 278,483 instances
- IMDBPerson: 602,483 isntances
- KnownForTitles: 1,745,302 instances
- PrincipalTitle: 1,235,557 instances
- Professions: 839,674 instances
- Ratings: 80,078 instances
- Titles: 220,949 instances
- Writers: 210, 299 instances

5. WEB APP DESCRIPTION

Below are descriptions of individual pages of *The Next Binge* web app:

5.1. PAGE DESCRIPTION

5.1.1. The Next Binge Page

This page is our web app's landing page. You are invited to hit the binge button and start exploring our website. This page serves as a welcome page when user first comes to the website. Upon entering the site from this page, the watch list is cleared for the user to make his or her own.

5.1.2. Basic Search Page

This page displays all the TV shows in the database and gives the user the option to conduct some basic searches. For example, user can search TV shows written by certain writers or directed by certain directors.

5.1.3. Search By BirthYear Page

This page asks the user to provide their birth year and display TV shows that are related to this year. For example, this page can display TV shows that were actively running in the year user was born. This page also displays all the directors', writers' and actors' work who share user's birth year respectively.

5.1.4. Advanced Search Page

This page provides user the opportunity to mix and match different requests to generate search results. User is able to provide multiple search criteria at once. For example, user can ask the website to display all TV shows that are directed and written by Ricky Gervais that has a rating of 8 or above, the website will display all the shows that meet all of the user's search input.

5.1.5. Watch List Page

This page will display all the TV shows that user added in their personal list while browsing on the other pages. This page also allows user to delete unwanted TV shows from their watch list.

5.1.6. Commonality and Differences in functionality

Different pages offer different search criteria for user to help user identify their next binge. However, there are shared features amongst all pages. For example, user has the option to click on individual shows on all pages to add them into their personal watch list.

5.2. API SPECIFICATION

Below are the routes information for our web app.

5.2.1. Route 1 userChoice

Description: Returns TV shows based on a specific filter search (Rating, Director, Year, Actor) and its corresponding search attribute based on that filter.
Request Path: /advancedsearch → GET method
Request Parameters: **Query**: minRating (int), maxRating (int), director_name (string), actor_name(string), startYear(int), endYear(int)
Response parameters: JSON; results (JSON array of TV shows (string), Director (string), Actor (string), Rating (string), Start Year (int), End Year (int))

5.2.2. Route 2 getTvRunTime

Description: Returns TV shows within a specified runtime in minutes range. minRangeRunTime and maxRangeRunTime are user specified inputs.
Request Path: /rangeruntime → GET method
Request Parameters: **Query**: minRangeRunTime(int), maxRangeRunTime(int)
Response parameters: JSON; results (JSON array of TV_shows(string), Runtime (int))

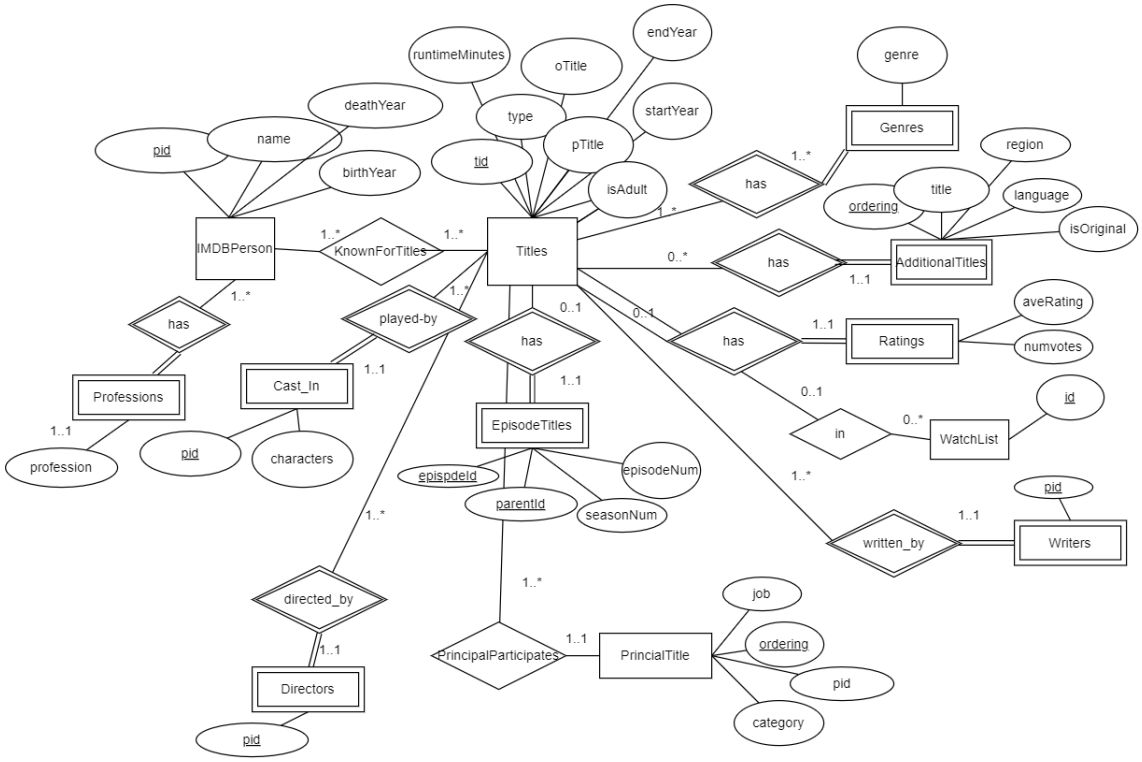

Fig. 16. ER Diagram of IMDb Database

5.2.3. Route 3 getLongestRunningTVShow

Description: Returns TV shows running within a specified year duration range. minYears and maxYears are user specified inputs.
Request Path: /longestRunning → GET method
Request Parameters: **Query**: minYears(int), maxYears(int)
Response parameters: JSON; results (JSON array of TV_shows(string), Years (int))

5.2.4. Route 4 getBirthYearDirectorTV

Description: Returns TV shows directed by a director who has the same user's birth year. userBirthyear is the birth year from user input. It can also filter results for TV shows that are in the English Language only.
Request Path: /birthyear → GET method
Request Parameters: **Query**: userBirthyear(int), isEnglish(boolean)
Response parameters: JSON; results (JSON array of TV_shows(string), Director (string))

5.2.5. Route 5 getBirthYearWriter

Description: Returns TV shows written by a writer who has the same user's birth year. userBirthyear is the birth year from user input. It can also filter results for TV shows that are in the English Language only.
Request Path: /birthyear/writers → GET method
Request Parameters: **Query**: userBirthyear(int), isEnglish(boolean)
Response parameters: JSON; results (JSON array of TV_shows(string))

5.2.6. Route 6 getBirthYearShows

Description: Returns TV shows that run between the user's birth year. userBirthyear is the birth year from user input. It can also filter results for TV shows that are in the English Language only.
Request Path: /birthyear/shows → GET method
Request Parameters: **Query**: userBirthyear(int), isEnglish(boolean)
Response parameters: JSON; results (JSON array of TV_shows(string), Start Year (int), End Year (int))

5.2.7. Route 7 getBirthYearActor

Description: Returns TV shows with a casted actor who has the same user's birth year. userBirthyear is the birth year from user input. It can also filter results for TV shows that are in the English Language only.
Request Path: /birthyear/actors → GET method **Request Parameters**: **Query**: userBirthyear(int), isEnglish(boolean)

Response parameters: JSON; results (JSON array of TV_shows(string), Actor (string))

5.2.8. Route 8 getGenreTVWriter

Description: Returns TV shows and writers of a particular genre where the genre is specified by the user.
Request Path: /writers → GET method
Request Parameters: **Query**: genre(string)
Response parameters: JSON; results (JSON array of TV_shows(string), Writer (string))

5.2.9. Route 9 getCastSize

Description: Returns TV shows that have cast sizes of at least minSize and at most maxSize. minSize and maxSize are both user inputs.
Request Path: /cast → GET method
Request Parameters: **Query**: minSize(int), maxSize(int)
Response parameters: JSON; results (JSON array of TV_shows(string), Cast Size (int))

5.2.10. Route 10 directorInXShows

Description: Returns the tv shows directed by a director involved in at least minSize number of different TV shows. User inputs are: director is the director's name, minSize is the minimum number of shows the director is involved in.
Request Path: /director → GET method
Request Parameters: **Query**: director(string), minSize (int)
Response parameters: JSON; results (JSON array of TV_shows(string), Director (string), Shows Directed (int))

5.2.11. Route 11 getAllShows

Description: Returns all the TV Show Titles.
Request Path: /basic/allshows→ GET method
Request Parameters: **Query**: none
Response parameters: JSON; results (JSON array of TV_shows(string))

5.2.12. Route 12 getShowsBasedOnRating

Description: Returns the tv shows of at least minRating and at most maxRating. minRating and maxRating are user inputs.
Request Path: /basic/ratings → GET method
Request Parameters: **Query**: director(string), minSize (int)
Response parameters: JSON; results (JSON array of TV_shows(string), Rating (int))

5.2.13. Route 13 getWatchList

Description: Returns the watchlist of the user. The watchlist contains all the TV Show Titles.
Request Path: /watchlist → GET method
Request Parameters: **Query**: none
Response parameters: JSON; results (JSON array of TV_shows(string))

5.2.14. Route 14 addToWatchList

Description: Adds a TV Show to the user's watchlist.
Request Path: /add → POST method
Request Parameters: **Query**: tid(string)
Response parameters: none

5.2.15. Route 15 removeFromWatchList

Description: Deletes a TV Show from the user's watchlist.
Request Path: /remove → DELETE method
Request Parameters: **Query**: tid(string)
Response parameters: none

5.2.16. Route 16 clearWatchList

Description: Clears / deletes all the entries in the user's watchlist.
Request Path: /clear → DELETE method
Request Parameters: **Query**: none
Response parameters: none

6. QUERIES

Here are some of the more complicated example queries we used on our web app:
Get tv show directed by a director involved in at least x-number (user input) of different TV shows within a specified rating range

```
WITH DirectorList AS (
    SELECT pid, COUNT(DISTINCT tid)
    AS numShows
    FROM Directors
    GROUP BY pid
    HAVING numShows >= ${minSize}
  ), DirectorsName AS (
    SELECT IP.name,
        DL.numShows, D.tid
    FROM IMDBPerson IP
        JOIN DirectorList DL
        ON IP.pid = DL.pid
        JOIN Directors D
        on IP.pid = D.pid
  )
```

```
SELECT DISTINCT T.tid AS tid,
    DN.name AS Director,
    T.pTitle as TV_Show,
    DN.numShows,
    "Add to WatchList" AS Watch
FROM Titles T
    JOIN DirectorsName
    DN ON T.tid=DN.tid
WHERE DN.name LIKE
    "%${director}%"
ORDER BY numShows DESC ,
    name ASC;
```

Recommend TV shows based on selected show's cast members AND genre

```
WITH Cast AS (
    SELECT CI.tid
    FROM Cast_In CI
        JOIN IMDBPerson P
        ON CI.pid = P.pid
    WHERE P.name LIKE '%${actor}%'
), Genre AS (
    SELECT tid
    FROM Genres
    WHERE Genre LIKE '%${genre}%'
)
SELECT pTitle as TV_Show
FROM Titles
WHERE tid IN
    (SELECT tid FROM Cast)
    AND
    tid IN
    (SELECT tid FROM Genre)
ORDER BY TV_Show ASC
```

Display long running TV shows within specified rating range

```
SELECT t.pTitle AS TV_Show,
    (t.endYear - t.startYear)
    AS runTime
FROM Titles t JOIN Ratings r
    ON t.tid = r.tid
WHERE r.aveRating >=
    ${minInputRating}
    AND r.aveRating <=
    ${maxInputRating}
ORDER BY runTime DESC;
```

Display all directors' work who share the user's birth year

```
WITH TVDirectors AS (
    SELECT t.pTitle, d.pid
    FROM Directors d
        JOIN Titles t ON d.tid = t.tid
    WHERE t.type LIKE '%tvseries%'
)
SELECT td.pTitle AS TV_Show
FROM TVDirectors td
    JOIN IMDBPerson ip
    ON td.pid = ip.pid
WHERE ip.birthYear = ${user_birthyear};
```

Recommend TV shows based on ratings, director name, TvShow Year, or actor name based on user selection

```
SELECT DISTINCT t.tid AS tid,
    pTitle AS TV_Show,
    aveRating AS Rating,
    startYear, endYear,
    "Add to WatchList" AS Watch
FROM Titles t
    JOIN Ratings r
    ON t.tid = r.tid
WHERE r.aveRating >= ${minRating}
    AND r.aveRating <= ${maxRating}
    AND t.startYear >= ${startYear}
    AND t.endYear <= ${endYear}
ORDER BY t.startYear, aveRating DESC;

WITH DirectorTitle AS (
    SELECT DISTINCT d.tid,
        ip.name AS Director
    FROM Directors d
        JOIN IMDBPerson ip
        ON d.pid = ip.pid
    WHERE ip.name LIKE
        "%${director_name}%"
),
    DirectorTVShows AS (
        SELECT d.tid,
            t.pTitle AS TV_Show,
            t.startYear,
            t.endYear, d.Director
        FROM DirectorTitle d
            JOIN Titles t
            ON d.tid = t.tid
    )
    SELECT dtv.tid AS tid,
        TV_Show, Director,
        aveRating AS Rating,
        startYear, endYear,
        "Add to WatchList" AS Watch
    FROM DirectorTVShows dtv
        JOIN Ratings r
        ON dtv.tid = r.tid
    WHERE r.aveRating >= ${minRating}
        AND r.aveRating <= ${maxRating}
        AND dtv.startYear >= ${startYear}
```

```
        AND dtv.endYear <= ${endYear}
    ORDER BY dtv.startYear,
        aveRating DESC,
        dtv.Director;

WITH ActorTitle AS (
    SELECT DISTINCT ci.tid,
        ip.name AS Actor
    FROM Cast_In ci
        JOIN IMDBPerson ip
        ON ci.pid = ip.pid
    WHERE ip.name LIKE "%${actor_name}%"
),
    ActorTVShows AS (
        SELECT a.tid,
            t.pTitle AS TV_Show,
            t.startYear,
            t.endYear, a.Actor
        FROM ActorTitle a
            JOIN Titles t
            ON a.tid = t.tid
    )
    SELECT atv.tid AS tid,
        TV_Show, Actor,
        aveRating AS Rating,
        startYear, endYear,
        "Add to WatchList" AS Watch
    FROM ActorTVShows atv
        JOIN Ratings r
        ON atv.tid = r.tid
    WHERE r.aveRating >= ${minRating}
        AND r.aveRating<=${maxRating}
        AND atv.startYear>=${startYear}
        AND atv.endYear<=${endYear}
    ORDER BY atv.startYear,
        aveRating DESC, atv.Actor;

WITH DirectorTitle AS (
    SELECT DISTINCT d.tid,
        ip.name AS Director
    FROM Directors d
        JOIN IMDBPerson ip
        ON d.pid = ip.pid
    WHERE ip.name LIKE
        "%${director_name}%"
),
    ActorTitle AS (
        SELECT DISTINCT d.tid,
            d.Director,
            ip.name AS Actor
        FROM DirectorTitle d
            JOIN Cast_In ci
            ON d.tid = ci.tid
            JOIN IMDBPerson ip
            ON ci.pid = ip.pid
        WHERE ip.name LIKE
            "%${actor_name}%"
    ),
        DirectorActorTVShows AS (
            SELECT a.tid,
                t.pTitle AS TV_Show,
                Director, Actor,
                t.startYear,
                t.endYear
            FROM ActorTitle a
                JOIN Titles t
                ON a.tid = t.tid
        )

SELECT datv.tid AS tid,
    TV_Show, Director,
    Actor, aveRating AS Rating,
    startYear, endYear,
    "Add to WatchList" AS Watch
FROM DirectorActorTVShows datv
    JOIN Ratings r
    ON datv.tid = r.tid
WHERE r.aveRating >= ${minRating}
    AND r.aveRating <= ${maxRating}
    AND datv.startYear >= ${startYear}
    AND datv.endYear <= ${endYear}
ORDER BY datv.startYear,
    aveRating DESC,
    datv.Director, datv.Actor;
```

(See Appendix in github repository for all queries.) These queries are utilized to join information from multiple tables in order to display requested TV show results based on user input.

The most complicated query is used for the function to recommend TV shows based on ratings, director name, TvShow Year, or actor name based on user selection. We used multiple joins and Common Table Expressions to simplify our query.

7. PERFORMANCE EVALUATION

We first preprocessed our data to eliminate records that are irrelevant to our website's functionality. For example, we are only interested in TV shows instead of movies and other types of media. By doing so we limit the number of records in Titles from 8,691,512 to 220,949. The dataset was originally too large to be successfully imported into our database. After eliminating unnecessary records, we were able to import completed data into our database.

We then changed the order of joins in queries to make sure the smaller sized tables are outer. For example, in Query 1 we initially used Genres as outer, recorded tim-

ings are as following: execution: 491 ms, fetching: 29 ms. After reordering the join, we used Title as outer, recorded timings are as following: execution: 472 ms, fetching: 24 ms

We also changed our queries to avoid unnecessary joins. For example, in function getAllShows, we previously used multiple joins to connect Title with other tables via tid attributes to obtain information. This query costs 4s 65ms to retrieve 500 records: execution: 3s 941ms, fetching: 124 ms. After we changed the query by only selecting from table Title, it only costs 225 ms to obtain 500 records: execution: 181 ms, fetching: 44 ms. In our most complex query where we select TV shows based on a combination of user input, we used multiple joins and multiple Common Table Expressions. When we changed the order of joins, it sped up the execution significantly. For example, when user provided rating range, show air year range and actor name, it took 2s 402 ms: execution 2s 318 ms, fetching: 84 ms. After optimization, it took 648 ms: execution 601 ms, fetching: 47 ms.

8. TECHNICAL CHALLENGES

After selecting the IMDB dataset, we quickly realized that the dataset has multiple tables with a significant amount of records in each. Initially we tried to load the raw files into the respective tables within Datagrip but due to the size of each file, we were unable to load the tables successfully.

We went back to our pre-processing workbook and challenged the information that we were bringing into each table (i.e. what was really needed for our web app vs what was a nice to have). Upon processing our data further, we were able to shrink the size of each raw file, which led to a successful import into the respective tables within Datagrip.

Another challenge we encountered was how to 'POST' and 'DELETE' data using API in React. This was needed for us to maintain a watchlist for the user. Additionally, we needed the list to be updated from any one of our search pages. This posed a significant challenge that required research, appropriate route directions, and updating render methods appropriately. On the same note, deleting the watchlist items either one-by-one or altogether while refreshing the result to show the removal(s) took a significant effort.

We also found query optimizations to pose a challenge. Several of our queries needed modifications on JOINS and the effective use and non-use of CTEs to speed up the results. This was especially true when loading large number of tuples in the initial load prior to user interaction and limiting of results.

Through this process, we learned additional skills on how to read documentation and apply API specs. Another interesting thing we were able to see first-hand was how the data ties together if foreign keys constraints are used appropriately. Lastly, importing and displaying the graphics and visualizations on each page was challenging yet enjoyable. Balancing what we wanted to show with how it would look on a web page was something that we learned through this process.