

MILESTONE 2

Github Link: <https://github.com/DarkHawk727/CS348-Project>

R1. Application high level description

We will be using the [Sakila](#) Movies Database to present an analysis dashboard for movies. Some features we look to implement include searching and recommendation of different movies. This would rely on setting up proper foreign key relationships in our database and determining a way to aggregate movie data to make recommendations. In addition, we would like to have a flexible interface for users to compare different movies based on genre, revenue, year released and other attributes. These pieces of information can be sorted and output tuple set size should be adjustable (limiting output to x values).

As per the outline recommendations, we will be using a locally hosted SQL based DB. Specifically, we believe DuckDB integrates well with our application written in Python. Additionally the total size of the database is well within the storage capacities of our laptops so using a remote server would introduce needless complexity.

We will be presenting the application to users via a web frontend. [The framework we have decided on in Milestone 2 is to use streamlit and numpy when setting up the common UI elements needed \(filters, inputs, buttons, tables\). Streamlit is markdown based and accepts pandas dataframes as the result of queries. Instructions on how to run the application can be found on the readme.](#)

R2. System Support Description

The project we are implementing will be hosted locally and version control would use GitHub. This includes the database which will be DuckDB and our application which will be written in Python. Some key frameworks that we will work with include Streamlit and Plotly for data tables/UI display and a Python DB connector provided by DuckDB for performing queries against our schema. In order to collaborate on application development, we have set up a GitHub repository so group members can contribute from their local machines. Specifically we would all have Python 10.x or later installed

locally along with DuckDB latest stable release of 1.2.0. [For the full list of dependencies, Requirements.txt](#) contains all the installations needed to run the application locally.

R3. Database with Sample Dataset

Sample Data

Our application is a movie review site (similar to IMDb) that draws on the Sakila sample SQL database provided by MySQL. Sakila simulates a DVD rental store containing tables for films, actors, languages, customer data, and rental transactions. It provides realistic foreign key relationships and built-in triggers (for events like returning a movie). Based on our application description, we will remove the data for store operation and focus on movies.

Data Generation/Cleaning/Importing

- **Generation:** The Sakila dataset is already curated by MySQL, so no additional data cleaning is required.
- **Scripts:** Two SQL files are provided—one for creating the schema (`schema.sql`) and another for inserting sample rows (`data.sql`).
- **Import Process(DuckDB):**
 1. `schema.sql` to define tables, views, and triggers.
 2. `data.sql` to populate the tables with sample records.

Using the Data to Populate the Database

1. Clone GitHub Repository for group project
2. Install DuckDB's [Python API](#)
3. Execute `schema.sql` and then `data.sql` (see [test.py](#)).
4. Confirm successful import by querying tables on either basic features or other simple tests. These tests should log when test.py is run.

R4. Database with Sample Dataset

Production Data

Beyond the sample data, we found that our review feature would better implemented as

- *Extended Data for Features: We extended Sakila's schema with additional functionality for user reviews and ratings.*
- *Made data modifications: We are using this DB as our base, but we are modifying it, adding extra tables and removing unnecessary tables.*

- *In addition, we sanitized the data to ensure reviews must all have a user and movie associated. In addition, we increased the variation in release year for the movies in our database to make the filtering feature realistic.*

Data Generation/Cleaning/Importing (same as R3)

- **Generation:** The Sakila dataset is already curated by MySQL, so no additional data cleaning is required.
- **Scripts:** Two SQL files are provided—one for creating the schema (`schema.sql`) and another for inserting sample rows (`data.sql`).
- **Import Process(DuckDB):**
 1. `schema.sql` to define tables, views, and triggers.
 2. `data.sql` to populate the tables with sample records.

Using the Data to Populate the Database

5. Clone GitHub Repository for group project
6. Install DuckDB's [Python API](#)
7. Execute `schema.sql` and then `data.sql` (see [test.py](#)).
8. Confirm successful import by querying tables on either basic features or other genetic tests.

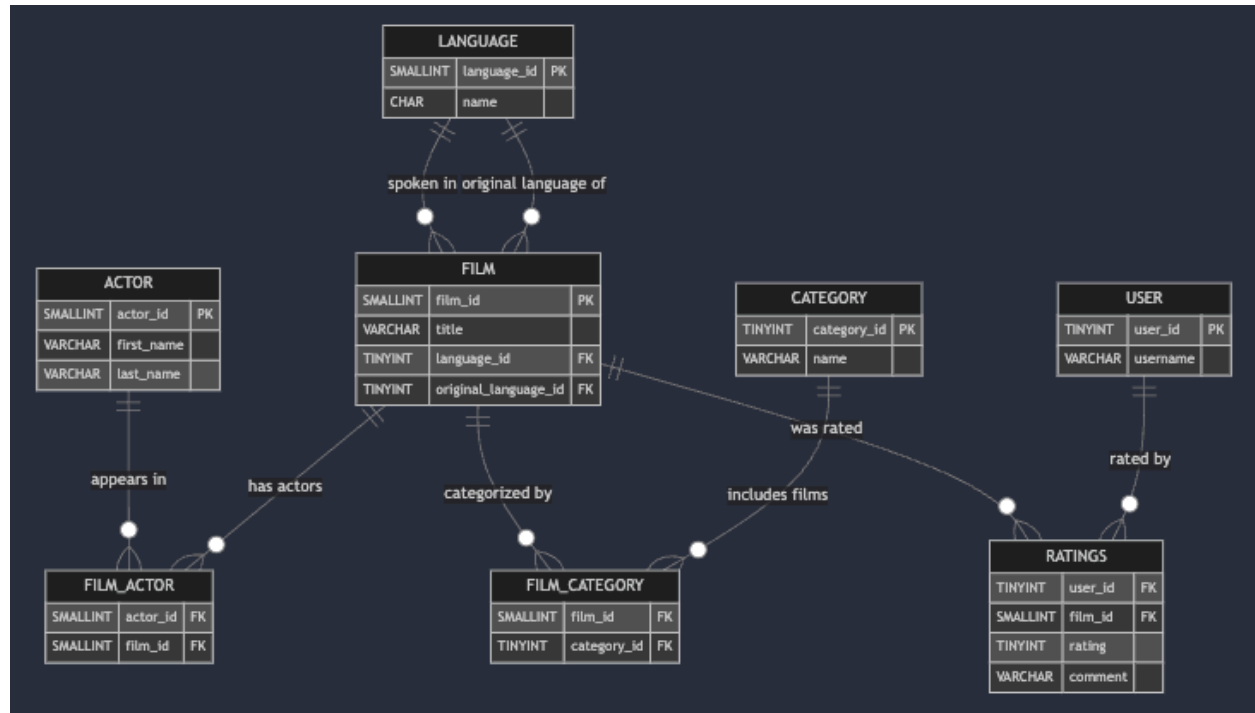
R5. Schema Design

R5a Assumptions

Many of the assumptions are defined in the schema with certain IDs being described as primary keys in their respective tables. We assume that there can be no duplicate movies (but can have the same name and language) as the movie id is a primary key. The same can be found for the other tables where there can be no duplicate users, languages, actors, and ratings.

When it comes to data quality, the dataset contains no null values for any entries of our table(almost every table is pre-generated to not have null values). For specific tables added by our group to extend the dataset, we also do not expect any null values. In particular, reviews could be “null” but in our case, we will implement a system that sets empty reviews as the empty string.

R5b E/R Diagram



This ER diagram was created using Mermaid.js to render the connections. If the quality of the image is poor, the raw file can be found on github.

R5c Relational Data Model

ACTOR

- actor_id (PK)

CATEGORY

- category_id (PK)

FILM

- film_id (PK)
- language_id (FK → LANGUAGE.language_id)
- original_language_id (FK → LANGUAGE.language_id)

FILM_ACTOR

- actor_id (PK, FK → ACTOR.actor_id)
- film_id (PK, FK → FILM.film_id)

FILM_CATEGORY

- film_id (PK, FK → FILM.film_id)
- category_id (PK, FK → CATEGORY.category_id)

LANGUAGE

- language_id (PK)

RATING

- user_id (PK, FK → USER.user_id)
- film_id (PK, FK → FILM.film_id)

USER

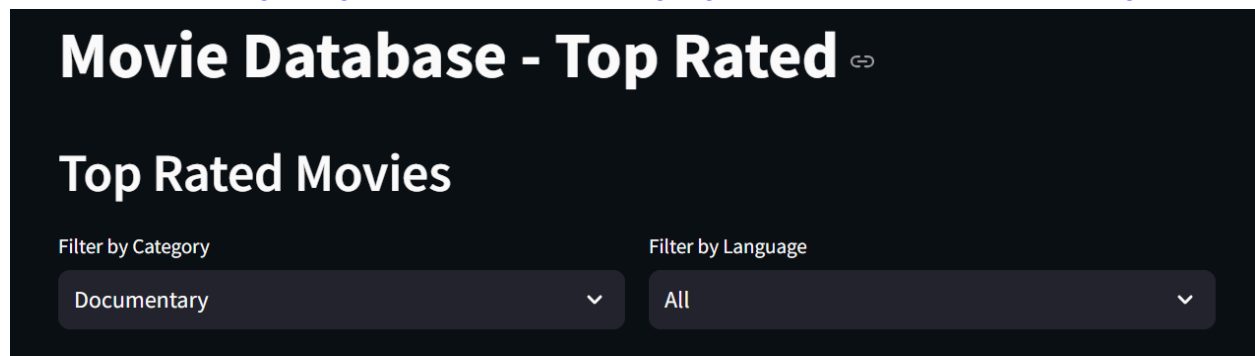
- user_id (PK)

R6. Basic Feature/Functionality 1

There will be a page to display the average ratings of each movie and it will display the title of the movie along with the average rating. In addition, [we will introduce filtering and review count to make the query more functional.](#)

R6-a Interface Design

The UI will depend on Streamlit's selection interface and is built by querying our dataset for languages and categories of films. We felt that having a side-by side layout is easy to use and the filtering immediately applies once the user makes a selection. The results will be listed out based on their rating along with the film name, language, and number of reviews through a table



R6-b SQL Query, testing with sample data

The query can be found under [tests/feature/inputs/test-sample-basic_feature_1.sql](#).

```

1  SELECT f.title, AVG(r.rating) AS average_rating
2  FROM FILM AS f
3  JOIN RATINGS AS r ON f.film_id = r.film_id
4  GROUP BY f.film_id, f.title;
5

```

The expected output should be (We have added more reviews for the production data and will also include reviews/ratings as an advanced feature):

```

CS348-Project > tests > feature > outputs > ≡ test-sample-basic_feature_1.out
1  |  |  | title  average_rating
2  |  |  | ACADEMY DINOSAUR  4.0

```

R6-c SQL Query, testing with production data

For our production data, we included more ratings to help set up our UI component. When it comes to optimizing the query, we rely on aggregation when calculating average ratings(which is generally a slower process) and we felt that by introducing indices on all our primary keys, we have optimized this query

The query can be found under [tests/feature/inputs/test-production-basic_feature_1.sql](#).

```

1  SELECT f.title, AVG(r.rating) AS average_rating
2  FROM FILM AS f
3  JOIN RATINGS AS r ON f.film_id = r.film_id
4  GROUP BY f.film_id, f.title;
5

```

The expected output should be:

```

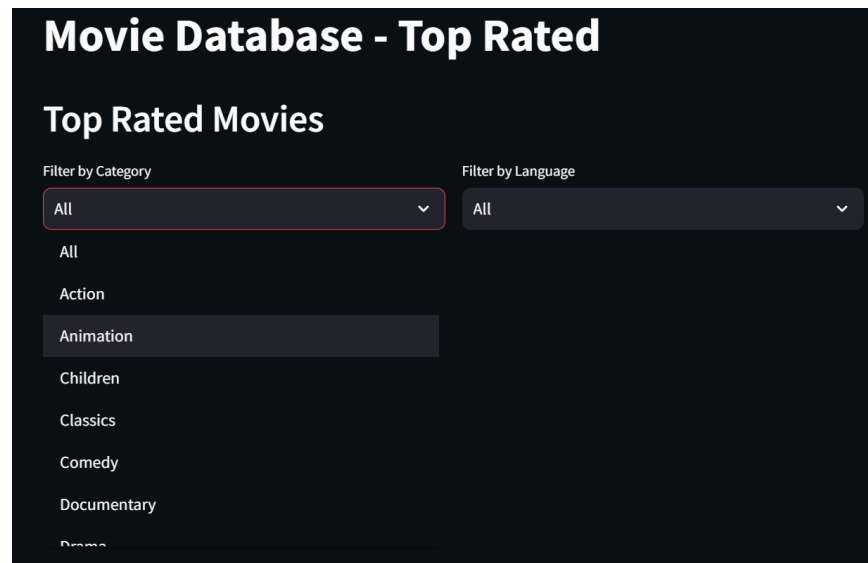
CS348-Project > tests > feature > outputs > ≡ test-production-basic_feature_1.out
1  |  |  | title  average_rating
2  |  |  | ACADEMY DINOSAUR  2.5
3  |  |  | AFFAIR PREJUDICE    5.0
4  |  |  | AFRICAN EGG        2.0

```

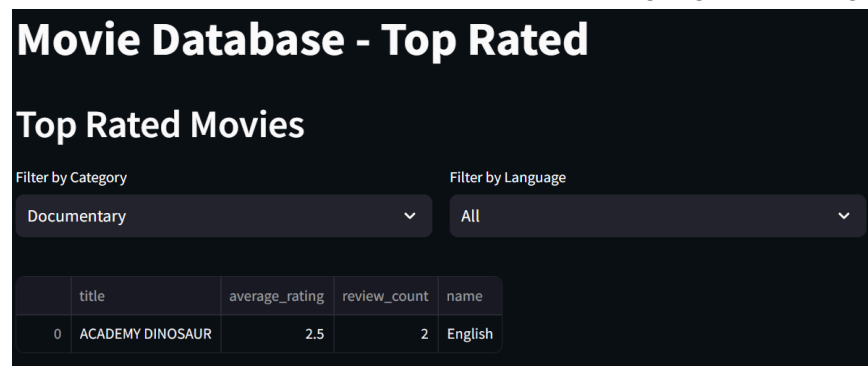
R6-d Implementation, SnapShot, Testing

For this feature, it's located on a separate page called "Top Rated". It gives a list of films which have user ratings and organizes them in descending order. The functionality of this feature combines the rating system with filters for both language and genre. In order to test, we used

the sample reviews made by users and tested the filtering properties and aggregation for average reviews. For this feature to be extended, we will implement advanced features for comments and review modification for the final milestone.



In the snapshot above, there are filters for both language and category



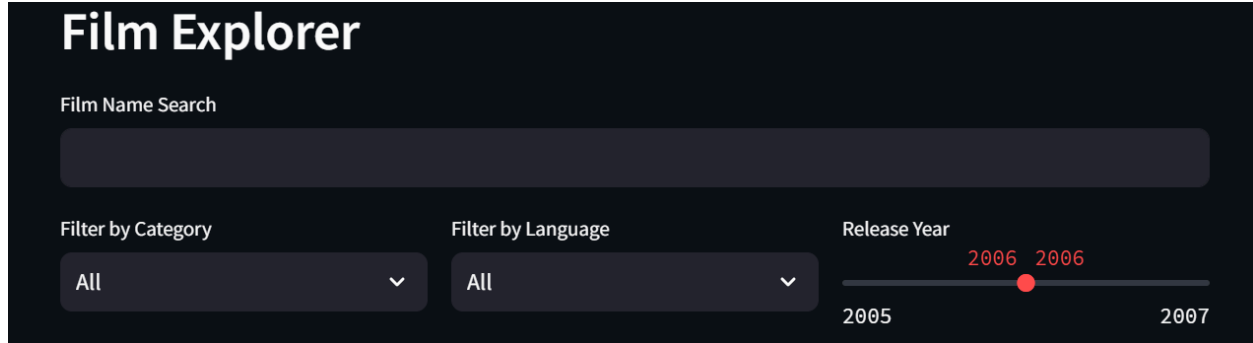
In the figure above, the only documentary that has reviews(in this case 2) is “ACADEMY DINOSAUR”

R7. Basic Feature/Functionality 2

This feature would work with the general search functionality to allow users to match their input with movies. Our goal includes adding a string match filter, category feature, language feature and release year filtering. On top of the basic search bar, we have implemented category/language filtering and release year filtering

R7-a

This would be an input and button widget combination for users to search up movies by name. This would combine as the user hits enter for the string search or confirm a selection. The UI has been updated to be much more comprehensive.



R7-b

Sample Data:

FILM table:

```
{{(1,Toy Story,Story about toys coming to life,1995,1,120,E),
(2,Avatar,Story of the Na'vi on Pandora,2009,2,190,PG-13),
(3,Toy Story 2, A sequel to toy story where one of the characters was stolen ,2005,1,142,E)
}}
```

LANGUAGE Data:

```
{ (1,English), (2,Italian), (3,Japanese) }
```

```
SELECT FILM.title, FILM.description,FILM.release_year, FILM.age_rating,
FILM.length, LANGUAGE.name, from FILM
INNER JOIN LANGUAGE ON FILM.language_id = LANGUAGE.language_id
where FILM.title like '%{user input}%';
```

The expected output for a search string “Toy” would be:

```
{
('Toy Story', 'Story about toys coming to life',1995, 'E`, 120, 'English'),
('Toy Story 2', 'A sequel to toy story where one of the characters was stolen', 2005, 'E`, 142,
'English')}
```

R7-c

For this query, instead of embedding variables into a conditional join statement, we have reduced the number of conditions based on the state of user selection from our form input. The generated query relies on specific attributes the user is searching up and does not have redundant predicates. When we actually reference the tables, indexing of the primary key is used to speed up query execution time.


```
CS348-Project > tests > feature > inputs > test-production-basic_feature_2.sql
1 SELECT FILM.title,
2   FILM.description,
3   FILM.release_year,
4   FILM.age_rating,
5   FILM.length,
6   LANGUAGE.name, from FILM
7 INNER JOIN LANGUAGE ON FILM.language_id = LANGUAGE.language_id
8 where FILM.title like '%ALA%';
```

For the sample above, only the search bar is in use while other filters are defaulted

	title	description	release_year
1	ALABAMA DEVIL	A Thoughtful Panorama of a Database Administrator And a Mad Scientist who must Outgun a Mad Scientist in A Jet Boat	20
2	ALADDIN CALENDAR	A Action-Packed Tale of a Man And a Lumberjack who must Reach a Feminist in Ancient China	20
3	ALAMO VIDEOTAPE	A Boring Epistle of a Butler And a Cat who must Fight a Pastry Chef in A MySQL Convention	20
4	ALASKA PHANTOM	A Fanciful Saga of a Hunter And a Pastry Chef who must Vanquish a Boy in Australia	20
5	GALAXY SWEETHEARTS	A Emotional Reflection of a Womanizer And a Pioneer who must Face a Squirrel in Berlin	20
6	HANOVER GALAXY	A Stunning Reflection of a Girl And a Secret Agent who must Succumb a Boy in A MySQL Convention	20
7	IMPACT ALADDIN	A Epic Character Study of a Frisbee And a Moose who must Outgun a Technical Writer in A Shark Tank	20
8	PRIDE ALAMO	A Thoughtful Drama of a A Shark And a Forensic Psychologist who must Vanquish a Student in Ancient India	20
9	SCALAWAG DUCK	A Fateful Reflection of a Car And a Teacher who must Confront a Waitress in A Monastery	20

The full output is [here](#), and will only return movies with name containing substring “ALA”

R7-d

Below is a sample of what the output should look like when you input name, language, and release year in the films tab. It demonstrates the ability to match string, release year, and language. Testing can involve picking out movies of a category, finding movies in English, those released in 2006 or later etc.

Film Explorer

Film Name Search

Gold

Filter by Category

All

Filter by Language

English

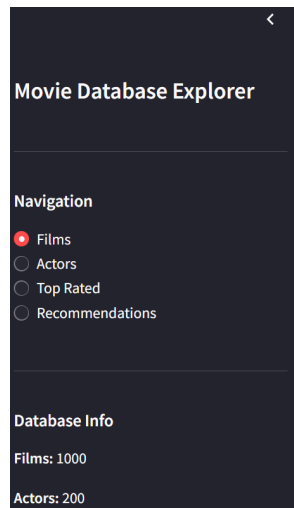
Release Year

200520062007

	film_id	title	release_year	length	age_rating	language
0	2	ACE GOLDFINGER	2,006	48	G	English
1	95	BREAKFAST GOLDFINGER	2,006	123	G	English
2	365	GOLD RIVER	2,006	154	R	English
3	366	GOLDFINGER SENSIBILI	2,006	93	G	English
4	367	GOLDMINE TYCOON	2,006	153	R	English
5	644	OSCAR GOLD	2,006	115	PG	English
6	798	SILVERADO GOLDFINGER	2,006	74	PG	English
7	870	SWARM GOLD	2,006	123	PG-13	English

R8. Basic Feature/Functionality 3

Note: The original feature for creating user reviews was moved to an advanced feature due to needing more advanced UI and complex input sanitation. This feature has now been switched to a sidebar with statistics on the database. Namely gives # of movies and actors currently in the database - inspired by other websites giving live time information on # of records/streams/videos active.



R8-a

This feature is automatically run by the init of the app and allows users to choose different categories to filter on and also have the ability to get recommendations or filter user ratings. In addition, it tracks the number of films and actors in the dataset.

R8-b

```
# Database info in sidebar (add more later)
st.sidebar.markdown("---")
st.sidebar.subheader("Database Info")
film_count = conn.execute("SELECT COUNT(*) FROM FILM").fetchone()[0]
actor_count = conn.execute("SELECT COUNT(*) FROM ACTOR").fetchone()[0]

st.sidebar.markdown(f"**Films:** {film_count}")
st.sidebar.markdown(f"**Actors:** {actor_count}")
```

Sample Data

- **FILM:** (film_id=106, title="Movie A"), (film_id=369, title="Movie B")
- **ACTORS:** (actor_id =1)

Expected Results

- Films: 2
- Actors: 1

R8-c

For the query, we rely on aggregation to count the number of actors and number of movies. Our production DB contains 1000 movies and 200 actors.

R8-d

When navigating the features, you can see all 4 pages and the movie count is up to date.

The screenshot displays the 'Movie Database Explorer' interface. On the left, a sidebar contains 'Navigation' options (Films, Actors, Top Rated, Recommendations) and 'Database Info' (Films: 1000, Actors: 200). The main area is titled 'Film Explorer' and includes a search bar, filters for Category and Language (both set to 'All'), and a 'Release Year' range slider from 2005 to 2007. Below these is a table of film details.

	film_id	title	release_year	length	age_rating	language
990	991	WORST BANGER	2,006	185	PG	English
871	872	SWEET BROTHERHOOD	2,006	185	R	English
816	817	SOLDIERS EVOLUTION	2,006	185	R	English
689	690	POND SEATTLE	2,006	185	PG-13	English
608	609	MUSCLE BRIGHT	2,006	185	G	English

R9. Basic Feature/Functionality 4

R9-a

There is a section in films which provides film details. When a user searches/picks a film from the dropdown, the information about the film including name, release year, category, actors, description, and genre.

Film Details

Select a film for details

ACADEMY DINOSAUR

Title: ACADEMY DINOSAUR

Release Year: 2006

Language: English

Length: 86 minutes

Age Rating: PG

Actors:
No actors listed

Categories:
No categories listed

Description: A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies

R9-b

Below is the primary query for locating the film and retrieving the general information about the film.

```

1  SELECT f.*, l.name as language
2  FROM FILM f
3  LEFT JOIN LANGUAGE l ON f.language_id = l.language_id
4  WHERE f.title = 'ACADEMY DINOSAUR'

```

film_id int16	title varchar	description varchar	...	length int32	age_rating varchar	language varchar
1	ACADEMY DINOSAUR	A Epic Drama of a	86	PG	English

In this case for sample input:

{id = 1, title = “ACADEMY DINOSAUR”, description = “A Epic Drama of a....”}, {id = 2, title = “star wars” description = “...”, ...}, {id = 3, title=“lord of the rings”, description = “...”, ...}}

The output would be the tuple for the film name “ACADEMY DINOSAUR”

CS348-Project > tests > feature > outputs > test-production-basic_feature_5.out

	film_id	title	description	release_year	language_id	length	age_rating	language
1								
2	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies	2006	1	86	PG	English

R9-c

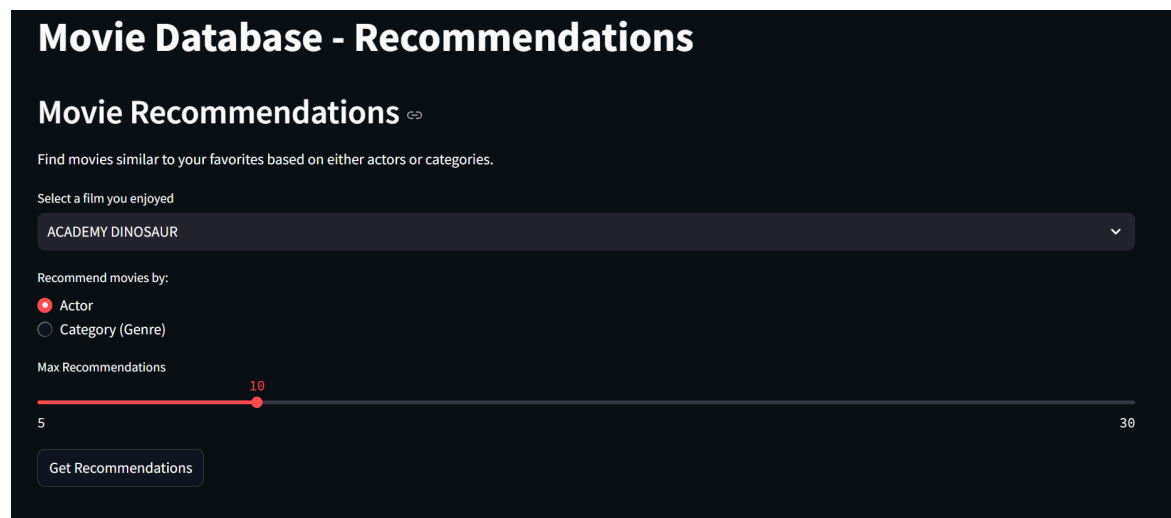
For the query, we rely on the indexing based on film ID to perform search(primary key of the film table). This is optimized given that we are filtering on the title attribute of films. In particular, we pre-compute parts of our query in Python to get actor and genre information.

R10. Basic Feature/Functionality 5

R10-a

For this feature, we will be creating a movie recommendation system for the user. This query will accept a movie and the user will be able to select either to recommend movies by common categories or common actors. The user will also be able to limit the amount of recommendations as in a large database the recommendations might be quite long. This query can be found under the “recommendation” section of the web app. This query will also not recommend the same movie.

This query can be found in the code running on the “recommendations.py” file under queryPages and the actual queries themselves in “queries/movie_recommendations.sql”



The screenshot shows a web interface titled "Movie Database - Recommendations". Below the title is a sub-header "Movie Recommendations" with a refresh icon. The main instruction is "Find movies similar to your favorites based on either actors or categories." There is a dropdown menu labeled "Select a film you enjoyed" with "ACADEMY DINOSAUR" selected. Below this, under "Recommend movies by:", there are two radio buttons: "Actor" (which is selected) and "Category (Genre)". At the bottom, there is a slider for "Max Recommendations" ranging from 5 to 30, with the current value set at 10. A "Get Recommendations" button is located at the bottom left of the form.

R10-b

```
WITH FilmActors AS (  
    SELECT fa.actor_id  
    FROM FILM_ACTOR fa  
    WHERE fa.film_id = {film_id}  
) ,  
  
FilmCategories AS (  
    SELECT fc.category_id  
    FROM FILM_CATEGORY fc  
    WHERE fc.film_id = {film_id}
```

```

)

SELECT DISTINCT f.film_id, f.title, f.release_year, f.age_rating
FROM FILM f
JOIN FILM_ACTOR fa ON f.film_id = fa.film_id
JOIN FILM_CATEGORY fc ON f.film_id = fc.film_id
WHERE
    f.film_id != {film_id}
    AND ({filter_condition})
ORDER BY f.title
LIMIT {limit};

```

Consider a sample Film table

[[id = 1, title = "matrix"], {id = 2, title = "star wars"}, {id = 3, title="lord of the rings"}]

Sample filmCategory table

[[film id = 1, category id = 1], {2,1}, {3,2}]

A similar movie query given star wars on category would return matrix (both sci fi movies) but not lord of the rings since it is fantasy.

R10-c

Using our production data set, we were able to implement a search based on a film user enjoyed and associated through actors in that film

Movie Recommendations

Find movies similar to your favorites based on either actors or categories.

Select a film you enjoyed

BANGER PINOCCHIO

Recommend movies by:

☒ Actor
 ☐ Category (Genre)

Max Recommendations

5

5

30

Get Recommendations

Recommended Movies Similar to 'BANGER PINOCCHIO' by Actor

	film_id	title	release_year	age_rating
0	1	ACADEMY DINOSAUR	2006	PG
1	5	AFRICAN EGG	2006	G
2	6	AGENT TRUMAN	2006	PG
3	8	AIRPORT POLLOCK	2006	R
4	19	AMADEUS HOLY	2006	PG

R16. Members.txt

See GitHub repository.

C1. ReadMe

Listed in Github with general setup instructions for Python Project. [It contains the application code, data, and current snapshot of development](https://github.com/DarkHawk727/CS348-Project)
<https://github.com/DarkHawk727/CS348-Project>

C2. SQL Code (Create table, etc)

See SQL files under the queries folder of GitHub. [The data is in CSV format under the data folder while configuration queries for creating/populating DB are under queries/configuration.](#)

C3. SQL Queries for features, test-sample.sql test-sample.out

[These files are under tests/feature/ the files specific to sample data have prefix test-sample-... They are named based on if they are related to sample or production.](#)
See GitHub Repository

C4. SQL Queries for features, test-production.sql test-production.out

These files are under tests/feature/ for input/output with prefix test-production-...
They are named based on if they are related to sample or production. The biggest change made is to ensure data consistency and create a number of predefined reviews which would be helpful for us when it comes to implementing UI functionalities.
See GitHub Repository

C5. Application Code

See GitHub Repository: <https://github.com/DarkHawk727/CS348-Project>