Databases Final Project Phase II

William Xu, Kun Liu

Q1

The old Phase I file is attached at the end of this document for your convenience. Changes were made and the old document no longer accurately reflects our project. Please see Section 3 for details.

Q2

William Xu	wxu43@jhu.edu	615 section
Kun Liu	kliu56@jhu.edu	615 section

Our application domain is a searchable movie database with a user watch history and a recommendation system. The recommendation system will based on watch history using natural language processing. Users will be able to search and add movies to their watched history list, and then have our database with python backend output recommendations based on user watch history.

A video demo of our database running can be found at: https://drive.google.com/file/d/1bE4o6PrwKN4bbXzy-l4Vf7vbmZtcRD3p/view?usp=sharing. Screenshots of the results are also included in Section 11. Instructions on how to import and run our project code is included in Section 6. This project is quite finicky as it has various connections between Python packages, PHP, CSS, MySQL, and MacOS specific system paths. We strongly advise the user to use their best judgement to solve any issues that may arise during import.

Q3

Our project description and database design has changed since our Phase I submission. The old document no longer accurately reflects our project. Below is a list of our changes.

- Database tables
    - Please see Section 12 for new tables.
    - All of the new tables are in 3NF and are lossless.
    - We removed some tables as we did not end up supporting search functionality for them.
        - Crew removed
        - Cast removed
        - Keyword removed (we now search over movie title, tagline, and genre using keyword instead of having a dedicated keyword table)
    - We combined some tables and split up other tables.
        - Movies was split up into mDetails (movie details), mEcon (economic data), mRating (rating data), mType (other movie data such as runtime, adult, release date)
        - Genre was combined into mDetails
- Search functionality.
    - The type of English questions that our database answers is now different. They are listed below for details.
        1. What movies have the keyword ‘dog’ in their tagline?
        2. What is the runtime of ‘Iron Man’?
        3. Which war movies mention the keyword ‘gun’?
        4. Which adult movies mention the keyword ‘sex’?
        5. Which action movies under 200 minutes mention the keyword ‘love’?
        6. Suppose I know the movie id. What is the movie with the movie id 234?

7. What is the popularity of ‘Spider-man 2’?
8. What is the rating for ‘Toy story’?
9. How popular are movies with the word ‘Hero’ in the tagline?
10. How much money did ‘Jumanji’ make?
11. What was the budget for ‘Avengers’?
12. Add ‘Hulk’ to my watched list.
13. Delete ‘Hulk’ from my watched list.
14. What movies are on my watched list?
15. What movies do you recommend me based on my watch history?
16. If both ‘Cars’ and ‘Cars 2’ are on my recommended list, how much more would you recommend one over the other?

Q4

Data used for this database was obtained from Kaggle at https://www.kaggle.com/rounakbanik/the-movies-dataset?select=movies_metadata.csv. There are a total of 45,404 movie entries listed in a single CSV file.

We used the python script data_clean.ipynb in our attached zip file to preprocess the data to remove weird formatting. We then used excel to split the csv’s into the multiple tables described in Section 12. We imported the tables into our database using the Flat File import function provided by MySQL Workbench.

Q5

The database portion of the project was built locally on our Macs by downloading MySQL 8.0 from the Oracle website. Interaction with the local MySQL database was done through command line (like on the ugrad machine) and through the free version of the MySQL Workbench application provided by Oracle (for better visualization). With the exception of the visual tool provided by MySQL Workbench, operation of the MySQL database is otherwise identical to operation of dbase on ugrad machine.

Q6

NOTE: Due to the extremely finicky nature of the interaction of Python virtual environments, package dependencies, and version compatibility with MySQL and PHP, the user is advised to use their best judgement in running the database and troubleshooting (it took us forever to get it working)

System/package requirements:

- **Python**: 3.7/3.8
- **OS**: MacOS (created on MacOS, other platforms not tested)
- **Dependencies**: pandas, numpy, scikit-learn, mysql-connector-python
- **MySQL:** 8.0
- **VS Code:** 1.52 (not required, not sure which versions work)

Brief User Guide

1. Download and setup Python virtual environment.
    a. Make sure you have a Python package management system such as conda (anaconda is recommended) downloaded and setup.
    b. Create a virtual environment in Python 3.7 or 3.8.
    c. Install pandas, numpy, scikit-learn and mysql-connector-python in that virtual environment.
        i. All dependencies of the above packages must also be installed. The Python package management system should take care of this and any conflicts.
        ii. Conda might not be able to install mysql-connector-python correctly. In that case, use pip to install it.
    d. Activate the virtual environment.
    e. Make sure the environment is up and running and all necessary files are in the file path. You may also have to add something to the .bash file. Please use your best judgement to trouble shoot.
        i. Trouble shooting tips:
            1. Consult stack overflow
            2. Restart your computer
            3. Cry
2. Download and setup MySQL 8.0.
    a. It is important that it is MySQL 8.0 as support for previous versions of MySQL is not guaranteed in the above installed python packages.
3. Create and load the database with values.
    a. Load the data provided in the .zip file into the database.
        i. The tables are split into separate .csv files in /data.
        ii. Table schemas can be found below in Section 12.
    b. We used MySQL Workbench to help create the tables and import the data using flat file import.
4. Establish connection between the recommendation.py file and the database.
    a. This involves manually going into the recommendation.py file and changing the connection information in the get_recommendations() method. Detailed instructions can be found here: https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysql-connector-connect.html.
    b. Run the provided test_connection() function to ensure that the SQL queries in the python script are running from the correct database.
5. Run the login.php file. The project should be up and running!

Q7

Major area:

- Decision-support system using natural language processing
  - The database provides a movie recommendation system. This was accomplished by passing data from the database to Python to be processed, and then fed back into the database so it can be queried during the session.
  - The recommendation analysis was done over each movie's tagline. The tagline was analyzed using language processing methods from the scikit-learn package (word vectorizer and tfidf) to generate a similarity score between a user's watched history and other movies, and recommend the user movies with the highest score.
- Advanced GUI made with PHP and CSS
  - Fancy GUI was made using PHP and CSS.

Minor areas:

- Advanced SQL topics (cursors)
  - Cursors were used in the Python script to select the appropriate query when multiple queries were being executed on one connection.
- Embedded SQL
  - Embedded SQL statements were used in the Python scripts to extract relevant information into a pandas data frame and output to the recommendations table so it can be queried from PHP.
- Pagination
  - Search results are divided up into discrete pages with 15 entries per page for better user viewing experience. This was accomplished through a mix of PHP and SQL interactions.

Q8

- Our database provides user-specific specific recommendations based on their watch history.
  - Recommendations will change as the user adds more movies to their watched list.
  - Recommendations are made by using a python backend script that uses the TF-IDF (term frequency inverse document frequency) method from scikit-learn to analyze the importance of each word in a movie's tagline and compares it across movies to find similarity.
  - A recommendation score is attached for user reference. Higher score means more similarities with movies on the user's watched list.
- Our database provides functionality for multiple users to maintain their own watched list that is saved across sessions.
  - User can add to, and delete from their watched list.
  - User watched list is saved across sessions, so they can log back on and add to their list at a later date using their user_id.
- Our database provides allows users the options to select what information they want to see about the movies they are searching up.
  - Users can select to see movie economic data (budget, revenue), movie ratings (popularity, user ratings), or movie details (tagline, genre).
- The GUI of our database – although far from perfect – holds a few subtle refinements.
  - Movies results are returned in a manageable 15 entries per page, with the option of going onto the next page if you want more results, and the option of going back to the previous page.
  - Some search options involve dropdown menus and checkboxes. This design choice was a form of limiting user input to ensure valid inputs.
  - Tabs on top to quickly navigate to different pages.

Q9

- Due to time limitations, we decided to not include information such as actor, studio, credits, movie summaries, etc. As such, our search options are not as powerful as existing movie databases (not as many search and filtering options). If given additional time, the search options can be worked on to improve additional functionality such as search/filter by actor, movie summary, studio, etc.
- Our database does not currently support an order by function for user to order their results by. This functionality could be implemented if given more time.
- Our database does not currently support advanced user account management or security as it was not our focus. Currently, users must remember their user_id and there is no password required. Account management features and security would be a worthwhile extension with additional time.
- Another worthwhile extension would be to make recommendations based on other features of the movie, such as ratings, or economic data (budget, revenue). Currently our database only recommends based on similarity in tagline. Other methods of recommendations would involve different techniques and would be interesting to pursue. A further extension to this could be to use other user's data to recommend movies using machine learning prediction algorithms.

Q10

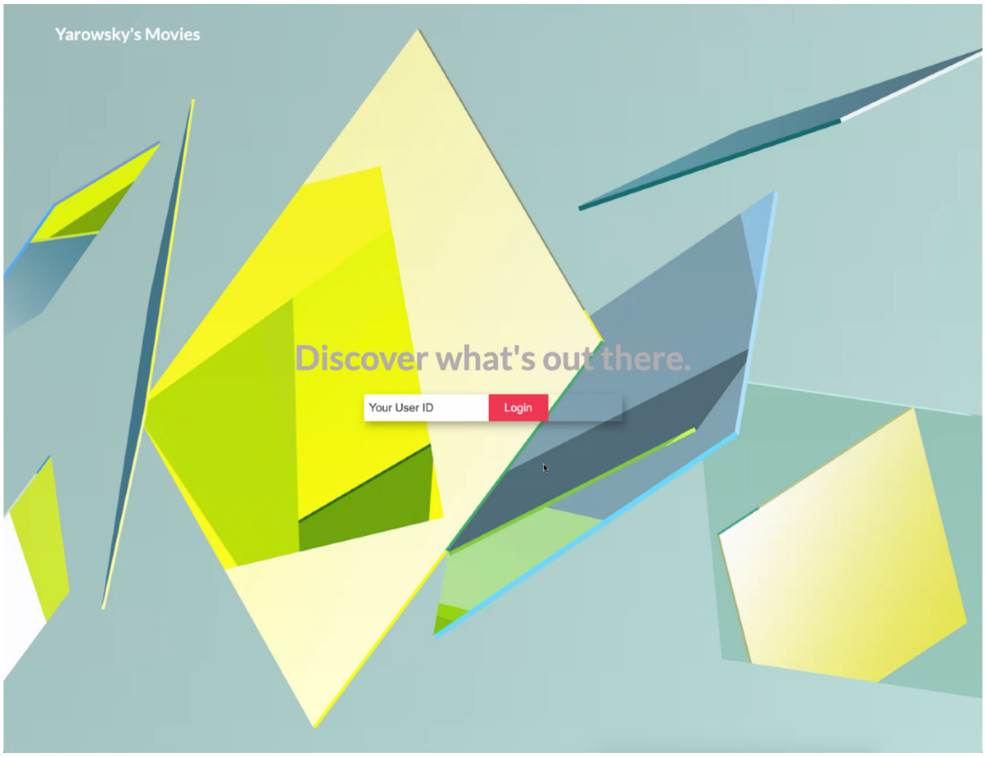
We used methods from the following Python packages: pandas, numpy, scikit-learn, mysql-connector-python. Excluding the python packages, the rest of the code were written by us (William and Kun) for this project.

Q11

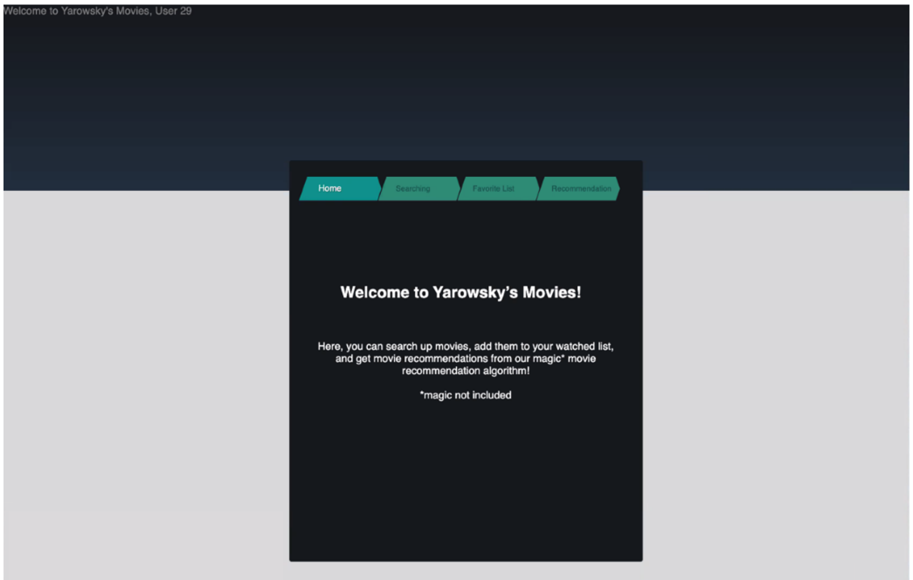
Please see the video demo at the link in Section 2 for a live demo. Below is a screenshot demonstration of the various pages and features.

Login page

This is the menu screen that greets the user. The name of the database is Yarowsky's Movies. The user can enter any user_id to start a session. The session starts when they click login.



Home page
Home page that greets the user once they login. User ID is displayed on the upper left corner. The four tabs on top is the way to access the various features this database supports.

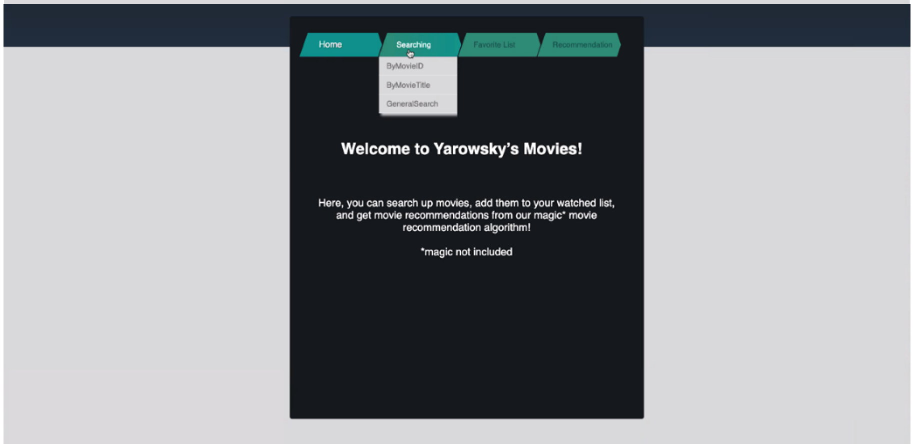

Welcome to Yarowsky's Movies, User 29

Welcome to Yarowsky's Movies!

Here, you can search up movies, add them to your watched list, and get movie recommendations from our magic* movie recommendation algorithm!

*magic not included

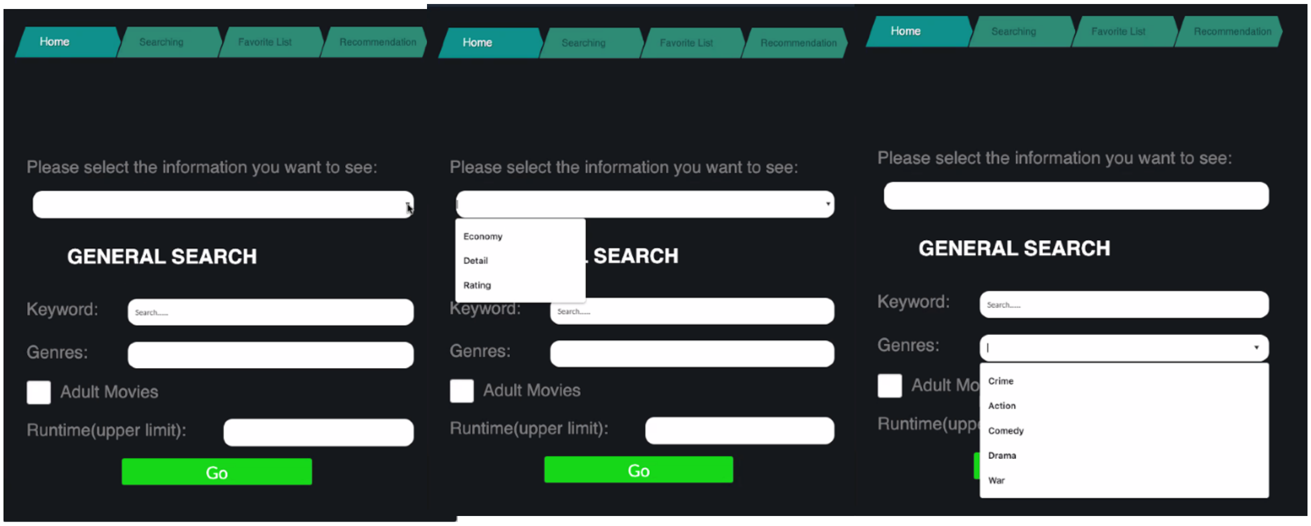
Search tab
Search tab provides three options for the user to search by. If they know the movie_id, they can search by movie id. They can search by title. They can also use the general search for keywords, genres, runtime, and other parameters.


Welcome to Yarowsky's Movies!

Here, you can search up movies, add them to your watched list, and get movie recommendations from our magic* movie recommendation algorithm!

*magic not included

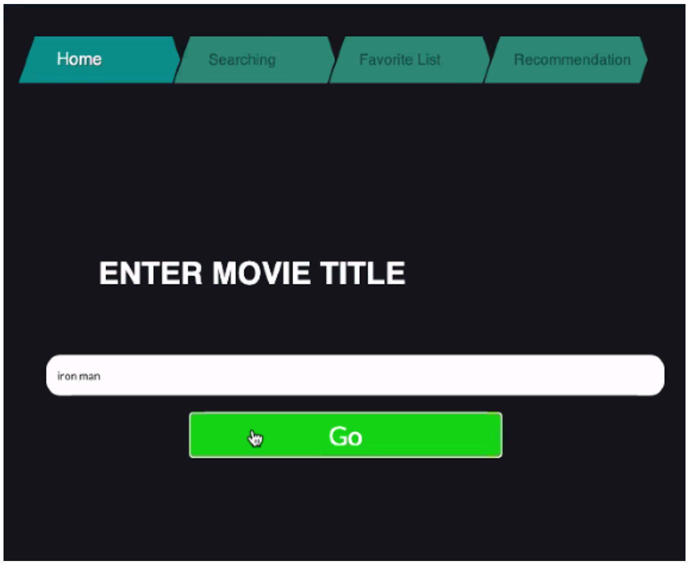
General search page
General search page provides the user the option to select what information about the movie they want to see (budget, tagline, ratings, etc.), the option to search by keyword, genres, and filter by runtime and adult movies. Below are images of the dropdown menus. The user can enter any keyword they want.


Please select the information you want to see:

GENERAL SEARCH

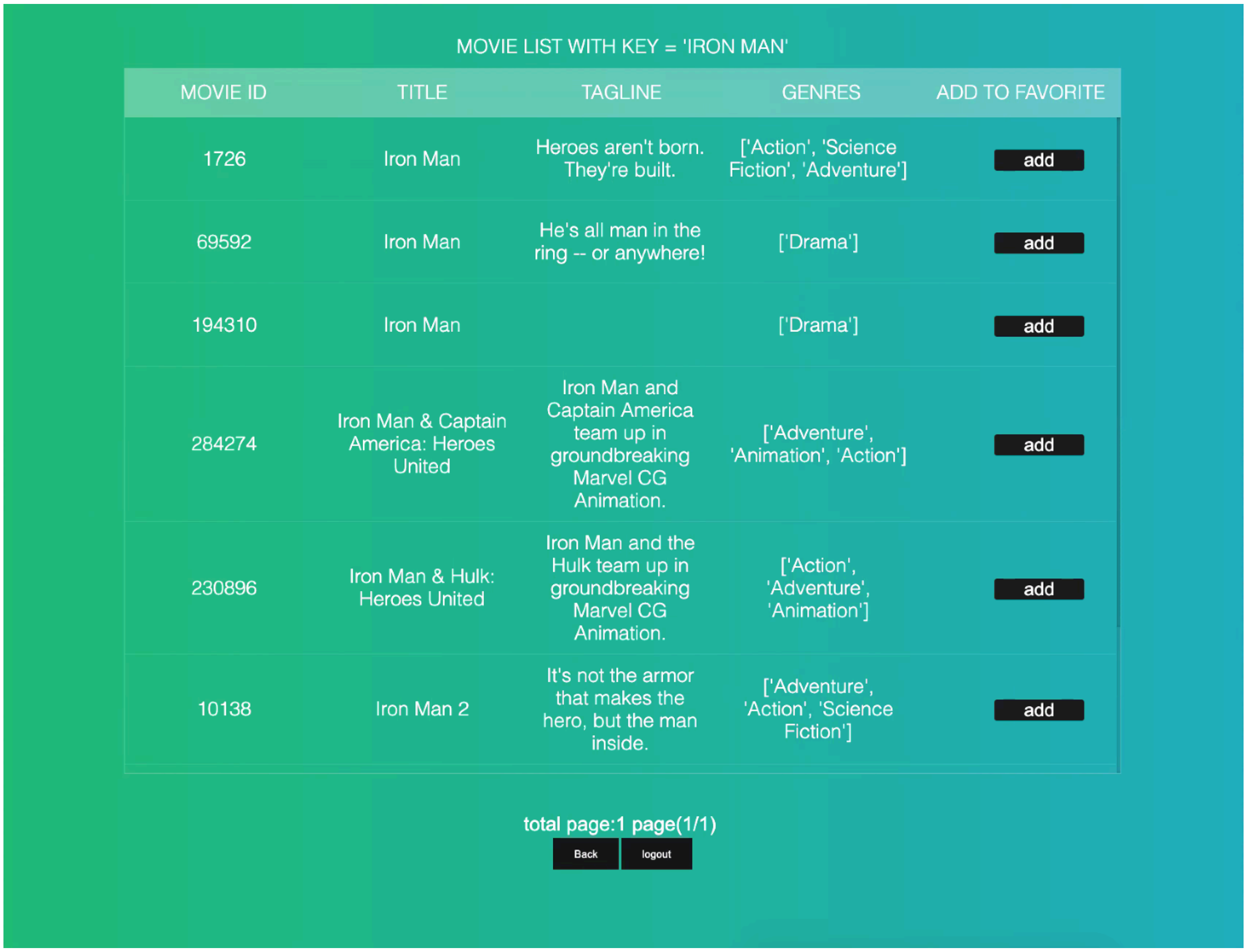
Example search
Let’s say we want to look up the movie ‘Iron Man’ because we have watched it and we want to add it to our watched list.

After we enter the movie title and click go, we are brought to the results page. By default, we list the tagline and genre of the movies (this can be changed in general search). On the right we can add the movie that we want into our watched/favorite list using the add button. We can navigate multiple pages of results at the bottom, but this result did not return multiple pages. We can also go back to the home page with the back button, and logout using the logout bottom.



ENTER MOVIE TITLE

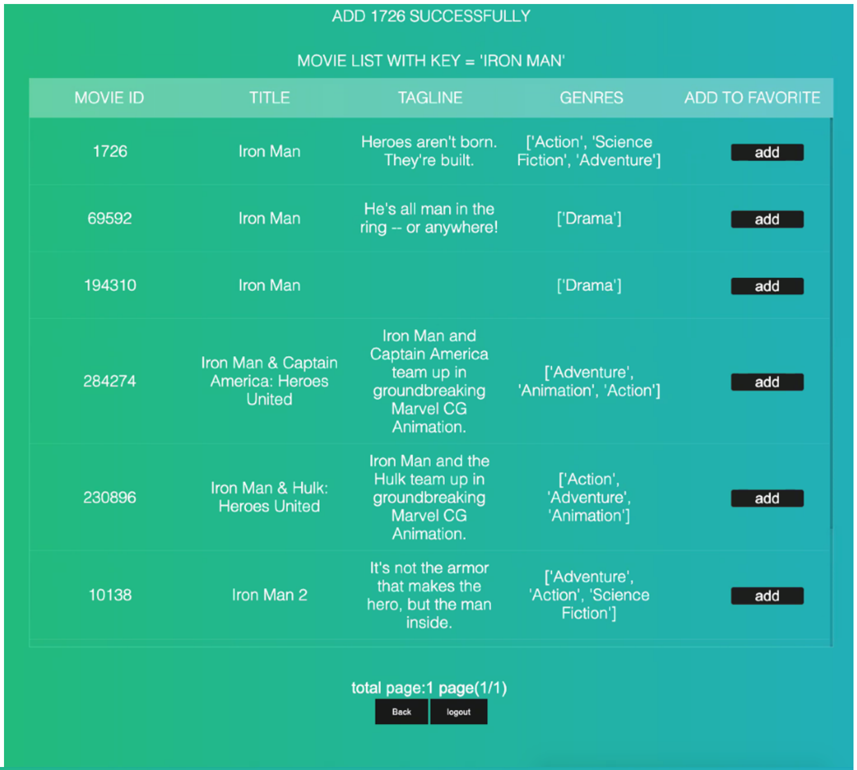
iron man


MOVIE LIST WITH KEY = 'IRON MAN'

MOVIE ID	TITLE	TAGLINE	GENRES	ADD TO FAVORITE
1726	Iron Man	Heroes aren't born. They're built.	['Action', 'Science Fiction', 'Adventure']	add
69592	Iron Man	He's all man in the ring -- or anywhere!	['Drama']	add
194310	Iron Man		['Drama']	add
284274	Iron Man & Captain America: Heroes United	Iron Man and Captain America team up in groundbreaking Marvel CG Animation.	['Adventure', 'Animation', 'Action']	add
230896	Iron Man & Hulk: Heroes United	Iron Man and the Hulk team up in groundbreaking Marvel CG Animation.	['Action', 'Adventure', 'Animation']	add
10138	Iron Man 2	It's not the armor that makes the hero, but the man inside.	['Adventure', 'Action', 'Science Fiction']	add

total page:1 page(1/1)

Add the first Iron Man movie to our favorite list
The top notifies the user that the movie was added successfully. If the user already has added it, it will say that the user has already added the movie.



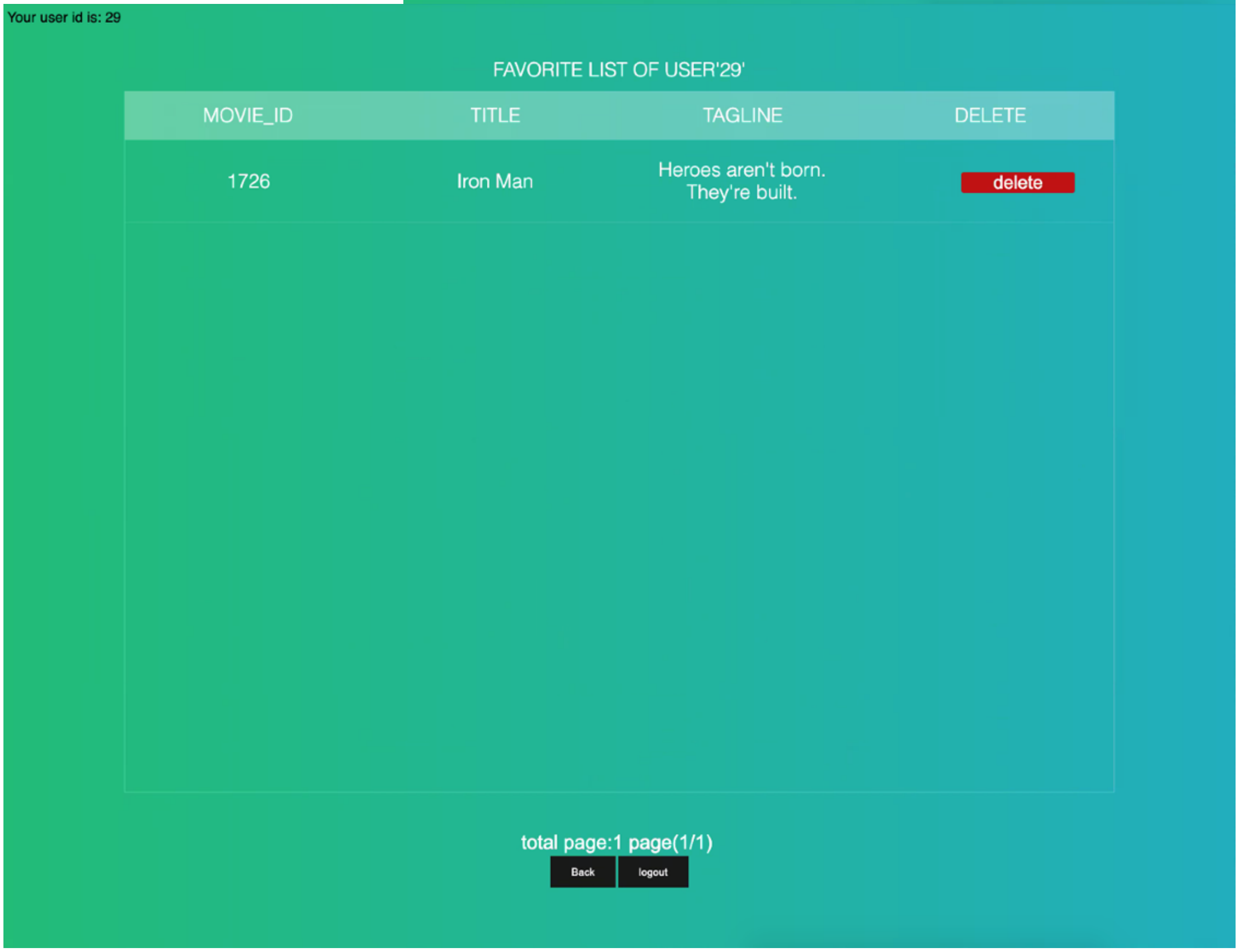
ADD 1726 SUCCESSFULLY

MOVIE LIST WITH KEY = 'IRON MAN'

MOVIE ID	TITLE	TAGLINE	GENRES	ADD TO FAVORITE
1726	Iron Man	Heroes aren't born. They're built.	['Action', 'Science Fiction', 'Adventure']	add
69592	Iron Man	He's all man in the ring -- or anywhere!	['Drama']	add
194310	Iron Man		['Drama']	add
284274	Iron Man & Captain America: Heroes United	Iron Man and Captain America team up in groundbreaking Marvel CG Animation.	['Adventure', 'Animation', 'Action']	add
230896	Iron Man & Hulk: Heroes United	Iron Man and the Hulk team up in groundbreaking Marvel CG Animation.	['Action', 'Adventure', 'Animation']	add
10138	Iron Man 2	It's not the armor that makes the hero, but the man inside.	['Adventure', 'Action', 'Science Fiction']	add

total page:1 page(1/1)

Check favorite list
We click back out of the results menu and click on the Favorite List tab to see which movies we have on our list. As seen, we have exactly the movie we just added. We also have the option to delete it from our watched list if a mistake was made.



Your user id is: 29

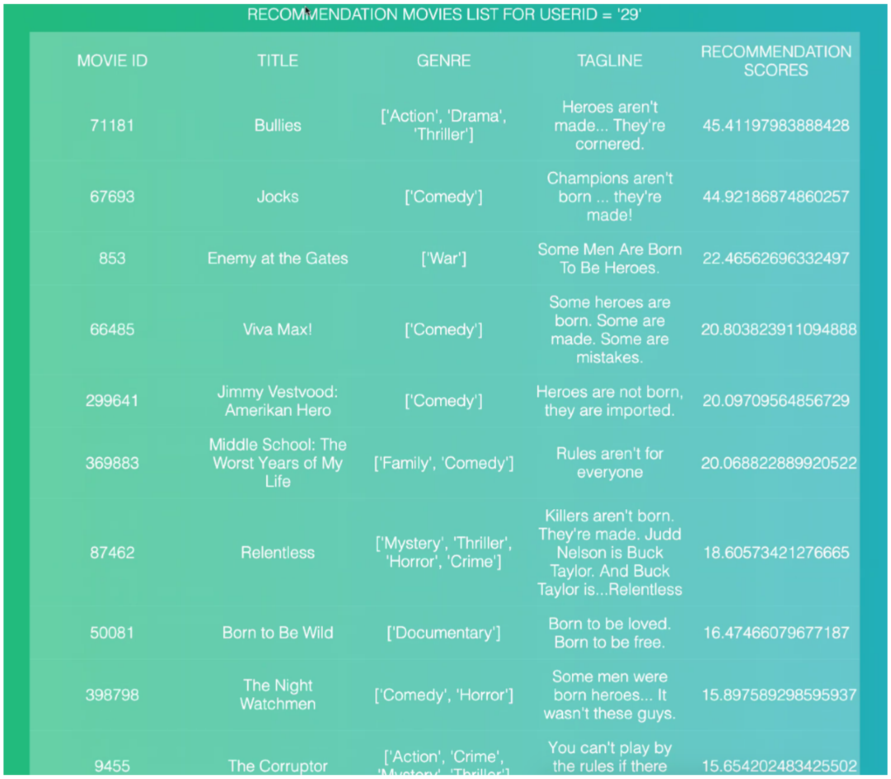
FAVORITE LIST OF USER'29'

MOVIE_ID	TITLE	TAGLINE	DELETE
1726	Iron Man	Heroes aren't born. They're built.	delete

total page:1 page(1/1)

Get movie recommendations
Now that we have a movie on our watched list, let’s see what other movies our algorithms recommends us. We go back onto the home page and click the recommendations tab.

We see that we are recommended movies with pretty similar taglines (intended since that is what our algorithm focuses on). On the right we have a recommendation score based on how similar the taglines are averaged over all of the movies in the user’s watched list.



RECOMMENDATION MOVIES LIST FOR USERID = '29'

MOVIE ID	TITLE	GENRE	TAGLINE	RECOMMENDATION SCORES
71181	Bullies	['Action', 'Drama', 'Thriller']	Heroes aren't made... They're cornered.	45.41197983888428
67693	Jocks	['Comedy']	Champions aren't born ... they're made!	44.92186874860257
853	Enemy at the Gates	['War']	Some Men Are Born To Be Heroes.	22.46562696332497
66485	Viva Max!	['Comedy']	Some heroes are born. Some are made. Some are mistakes.	20.803823911094888
299641	Jimmy Vestvood: Amerikan Hero	['Comedy']	Heroes are not born, they are imported.	20.09709564856729
369883	Middle School: The Worst Years of My Life	['Family', 'Comedy']	Rules aren't for everyone	20.068822889920522
87462	Relentless	['Mystery', 'Thriller', 'Horror', 'Crime']	Killers aren't born. They're made. Judd Nelson is Buck Taylor. And Buck Taylor is...Relentless	18.60573421276665
50081	Born to Be Wild	['Documentary']	Born to be loved. Born to be free.	16.47466079677187
398798	The Night Watchmen	['Comedy', 'Horror']	Some men were born heroes... It wasn't these guys.	15.897589298595937
9455	The Corruptor	['Action', 'Crime',	You can't play by the rules if there	15.654202483425502

More features of the database is demonstrated in the live demo linked in Section 2.

Q12

```
/*Example tuple mDetail(8844, ‘Jumanji’, ‘Roll the dice and unleash the
excitement!’, ‘['Adventure', 'Fantasy', 'Family']’)*/
CREATE TABLE mDetail(
  movie_id       int NOT NULL,
  title          text,
  tagline        text,
  genre          text,
  PRIMARY KEY(movie_id)
)

/*Example tuple mEcoBen(8844, 65000000, 262797249)*/
CREATE TABLE mEcoBen(
  movie_id       int NOT NULL,
  budget         int NULL,
  revenue        int NULL,
  PRIMARY KEY(movie_id)
)

/*Example tuple mType(8844, FALSE, 1995-12-15, 104)*/
CREATE TABLE mType(
  movie_id       int NOT NULL,
  adult          text,
  release_date   date,
  runtime        double,
  PRIMARY KEY(movie_id)
)

/*Example tuple mRating(8844, 17.015539, 6.9)*/
CREATE TABLE mRating(
  movie_id       int NOT NULL,
  popularity     double,
  vote_average   double,
  PRIMARY KEY(movie_id)
)

/*Example tuple watchedMovies(316, 8844)*/
CREATE TABLE watchedMovies(
  user_id              int NOT NULL,
  watched_movie_id     int NOT NULL,
  PRIMARY KEY(user_id, watched_movie_id)
  FOREIGN KEY(watched_movie_id) REFERENCES mDetail(movie_id)
)

/*Example tuple recommendation(8844, 26.123098)*/
CREATE TABLE recommendation(
  movie_id       int NOT NULL,
  scores         double,
  PRIMARY KEY(movie_id)
)
```

Q13

STORED PROCEDURES

```
/*Query to return movies by title*/
DELIMITER //
CREATE PROCEDURE SearchByTitle(IN input VARCHAR(20), IN pagesize INT, IN
numoffset INT)
BEGIN
SELECT movie_id, title, tagline, genre FROM mDetail WHERE title LIKE
CONCAT('%',input,'%') ORDER BY title ASC LIMIT pagesize OFFSET numoffset;
END; //


/*Query to return movies by id*/
DELIMITER //
CREATE PROCEDURE SearchByID(IN input INT)
BEGIN
SELECT movie_id, title, tagline, genre FROM mDetail WHERE movie_id = input;
END; //


/*Query to return movie details (tagline, genre) given keyword, genre, adult
rating and runtime*/
DELIMITER //
CREATE PROCEDURE SearchByGeneralDetail(IN keyword VARCHAR(50), IN mgenre
VARCHAR(20), IN adlt VARCHAR(5), IN runtime INT, IN pagesize INT, IN
numoffset INT)
BEGIN
SELECT D.movie_id, D.title, D.tagline, D.genre FROM mDetail AS D JOIN mType
AS T ON D.movie_id = T.movie_id WHERE (D.title LIKE CONCAT('%',keyword,'%')
OR D.tagline LIKE CONCAT('%',keyword,'%') OR D.genre LIKE CONCAT('%',
mgenre,'%')) AND T.adult = adlt AND T.runtime <= runtime ORDER BY title ASC
LIMIT pagesize OFFSET numoffset;
END; //


/*Query to return movie economic details (revenue, budget) given keyword,
genre, adult rating and runtime*/
DELIMITER //
CREATE PROCEDURE SearchByGeneralEcon(IN keyword VARCHAR(50), IN mgenre
VARCHAR(20), IN adlt VARCHAR(5), IN runtime INT, IN pagesize INT, IN
numoffset INT)
BEGIN
WITH mFilter AS (SELECT D.movie_id, D.title FROM mDetail AS D JOIN mType AS T
ON D.movie_id = T.movie_id WHERE (D.title LIKE CONCAT('%',keyword,'%') OR
D.tagline LIKE CONCAT('%',keyword,'%') OR D.genre LIKE CONCAT('%',
mgenre,'%')) AND T.adult = adlt AND T.runtime <= runtime)
SELECT F.movie_id, F.title, E.budget, E.revenue FROM mFilter AS F, mEcoBen AS
E WHERE F.movie_id = E.movie_id ORDER BY title ASC LIMIT pagesize OFFSET
numoffset;
END; //
```

```
/*Query to return movie rating details (popularity, voter average) given
keyword, genre, adult rating and runtime*/
DELIMITER //
CREATE PROCEDURE SearchByGeneralRating(IN keyword VARCHAR(50), IN mgenre
VARCHAR(20), IN adlt VARCHAR(5), IN runtime INT, IN pagesize INT, IN
numoffset INT)
BEGIN
WITH mFilter AS (SELECT D.movie_id, D.title FROM mDetail AS D JOIN mType AS T
ON D.movie_id = T.movie_id WHERE (D.title LIKE CONCAT('%',keyword,'%') OR
D.tagline LIKE CONCAT('%',keyword,'%') OR D.genre LIKE CONCAT('%',
mgenre,'%')) AND T.adult = adlt AND T.runtime <= runtime)
SELECT F.movie_id, F.title, R.popularity, R.vote_average FROM  mFilter AS F,
mRating AS R WHERE F.movie_id = R.movie_id ORDER BY title ASC LIMIT pagesize
OFFSET numoffset;
END; //


/*Query to return number of results to help divide up the result pages*/
DELIMITER //
CREATE PROCEDURE SearchByGeneralCount(IN keyword VARCHAR(50), IN mgenre
VARCHAR(20), IN adlt VARCHAR(5), IN runtime INT)
BEGIN
SELECT COUNT(*) FROM mDetail AS D JOIN mType AS T ON D.movie_id = T.movie_id
WHERE (D.title LIKE CONCAT('%',keyword,'%') OR D.tagline LIKE
CONCAT('%',keyword,'%') OR D.genre LIKE CONCAT('%', mgenre,'%')) AND T.adult
= adlt AND T.runtime <= runtime;
END; //


/*Query to add movie to user favorite list*/
DELIMITER //
CREATE PROCEDURE AddFavorite(IN userID INT, IN movieID INT)
BEGIN
INSERT INTO watchedMovies(user_id,watched_movie_id) VALUES(userID, movieID);
END; //


/*Query to remove movie from user favorite list*/
DELIMITER //
CREATE PROCEDURE RmvFavorite(IN userID INT, IN movieID INT)
BEGIN
DELETE FROM watchedMovies WHERE user_id = userID AND movie_id = movieID;
END; //

/*Query to get the recommended movies from the recommendation table. Used
because our python script direct adds to the recommendation table using
embedded SQL*/
DELIMITER //
CREATE PROCEDURE Recommend()
BEGIN
SELECT D.movie_id, D.title, D.genre, D.tagline, R.scores FROM mDetail AS D
JOIN recommendation AS R ON R.movie_id = D.movie_id ORDER BY R.scores DESC;
DELETE FROM recommendation;
END; //
```

```
DROP PROCEDURE SearchByTitle;
DROP PROCEDURE SearchByID;
DROP PROCEDURE SearchByKeyword;
DROP PROCEDURE AddFavorite;
DROP PROCEDURE RmvFavorite;
DROP PROCEDURE Recommend;
```

EMBEDDED SQL STATEMENTS in Python

```
#Embedded SQL statement in Python script to get user favorite movies
qryFav = 'SELECT D.movie_id, D.tagline FROM watchedMovies AS W, mDetail AS D
WHERE W.Watched_movie_id = D.movie_id AND W.user_id = ' + str(userID)

#Embedded SQL statement in Python script to get all movies for comparison
qryMov = 'SELECT D.movie_id, D.tagline FROM mDetail AS D'

#Embedded SQL statement to add our top 10 recommended movies to the
recommendations table
sql = 'INSERT INTO recommendation(movie_id, scores) VALUES (%s, %s)'
```

EMBEDDED SQL STATEMENTS in PHP

```
//Query to get user favorite list
$sql = sprintf("SELECT count(*) FROM db.watchedMovies where user_id
= %d;",$user_id);
```

The rest of the code that we have used to create our project can be found in the associated .zip file. Below is a short list of the files and what they were for.

- Python

data_clean.ipynb	data preprocessing
recommendation_draft.ipynb	Python analysis code testing draft
recommendation_clean.ipynb	Python analysis code testing clean copy
recommendation.py	Python function file for recommendations
test.py	Testing Python connection with PHP

- PHP (all for GUI)
    - assignPage.php
    - detailGeneralSearchCpy.php
    - economyGeneralSearchCpy.php
    - favorite.php
    - homeCpy.php
    - login.php
    - ratingGeneralSearchCpy.php
    - recommendations.php
    - SearchByIdCpy.php
    - SearchByTitlePagesCpy.php
    - Session.php
- CSS (all for GUI)
    - Favorites.css
    - Home.css
    - SearchByTitle.css
    - W2.css

Databases Final Project Phase I

1)
William Xu, Kun Liu

2)
Movie database with a user watch history and a recommendation system based on watch history using natural language processing. Users will be able to search and add movies to their watched history list, and then have our database + python backend output recommendations based on user given criteria.

3) Questions:

1. Find all of the comedy that have a budget of less than $600K ordered by movie id
2. Find all of the movies whose keyword contains “animal”
3. Find imdb_id based on movie id and original title contains “Head”
4. Find all of the movies whose budgets are over $1M and popularity are over 5 stars
5. Find all of the adult movies that are released in 2019 and runtime are less than 2 hours
6. Find all of the movies in English and user ratings are over 7 stars ordered by popularity
7. Find all of the movies whose budget are less than revenue
8. Find all of the movies starring “Tom Cruise” and user ratings are over 8 stars
9. From all of the movies starring “Tom Cruise”, rank them by how similar they are to movies that I have watched
10. Find the 10 most popular movies with a plot similar to “Toy Story”
11. Find the 10 most dissimilar movies to “Mission Impossible”
12. What movies have I watched?
13. Recommend me a movie with a rating above 3 stars with a similar plot to movies that I have watched
14. Recommend me a comedy movie with a similar plot to movies that I have watched
15. Recommend me 10 movies that I have not watched anything similar to (not similar plot) sorted by rating
16. Tell me which Romance movies I have watched and the average ratings of the movies
17. Search up “Monty Python and the Holy Grail” and tell me what is the probability that I will like this movie

4)

```
-- movie --
CREATE TABLE Movie(
        movie_id        int NOT NULL,
        title           varchar(100) NOT NULL,
        revenue         bigint DEFAULT NULL,
        release_date    date DEFAULT NULL,
        popularity      decimal(5,0) DEFAULT NULL,
        budget          bigint DEFAULT NULL,
        adult           boolean DEFAULT NULL,
        overview        text DEFAULT NULL,
        from_colleciton varchar(100)  DEFAULT NULL,
        language        varchar(5) DEFAULT NULL,
        runtime         int DEFAULT NULL,
        PRIMARY KEY (movie_id)
)

-- genre --
CREATE TABLE Genre(
        movie_id        int NOT NULL,
        genre           varchar(100) NOT NULL,
        PRIMARY KEY (movie_id)
)

-- cast --
CREATE TABLE Cast(
        movie_id        int NOT NULL,
        actor           varchar(100) NOT NULL,
        PRIMARY KEY (movie_id)
)

-- crew --
CREATE TABLE Crew(
        movie_id        int NOT NULL,
        member          varchar(100) NOT NULL,
        PRIMARY KEY (movie_id)
)

-- keyword--
CREATE TABLE Keyword(
        movie_id        int NOT NULL,
        keyword         varchar(100) NOT NULL
        PRIMARY KEY (movie_id)
)

-- rating --
CREATE TABLE Rating(
        user_id         int NOT NULL,
        movie_id        int NOT NULL,
        rating          decimal(2,0) NOT NULL,
        PRIMARY KEY (`user_id`)
)

-- user watched movies --
CREATE TABLE Watched(
        user_id         int NOT NULL,
        watched_id int NOT NULL,
        PRIMARY KEY (`user_id`)
)
```

5)

First, some simple search queries:

```
/*Find all of the adult movies that are released in 2019 and runtime are less than 2 hours*/
SELECT M.title FROM Movie as M
WHERE M.release_date LIKE “2019%” AND M.runtime < 120 AND M.adult = true
```

```
/*Find all of the movies whose keyword contains “animal”*/
SELECT  Movie.title FROM Movie as M NATURAL LEFT JOIN Keyword as K
WHERE K.keyword LIKE “animal”
```

Next, suppose a user wants to add movies to their watched list:

```
/*Users will add movies by clicking a button next to their searched movie after inputting their
user id */
INSERT INTO Watched VALUES(user_id, movie_id)
```

```
/*Users can see the movies they added to their watched history list*/
SELECT M.title FROM Movie as M NATURAL LEFT JOIN Watched as W
WHERE W.user_id = user_input
```

```
/*Users can delete movies from their watched history (maybe)*/
DELETE FROM Watched as W WHERE W.user_id = user_input1 AND W.watched_id =
user_input2
```

Next, onto the more interesting recommendation queries:

```
/*Recommend me a comedy movie with a similar plot to movies that I have watched*/
/*First, we need to select all of the comedy movies and the movies that the user have
watched and their associated plots from the overview attribute. With the two tables from the
two queries, we will send them to our python script. We will calculated the similarity between
each user watched movie and each comedy movie. We will average each similarity score
over each comedy movie. We will then need to output a single table Similarity(movie_id int,
sim_score decimal(3,2)). And then select from that table the desired movies*/

WITH MovInt(movie_id, overview) AS (SELECT M.movie_id, M.overview FROM Movie as M
NATURAL LEFT JOIN Genre as G WHERE G.genre = “comedy”)
```

```
WITH MovUser(movie_id, overview) AS (SELECT M.movie_id, M.overview FROM Movie as
M NATURAL LEFT JOIN Watched as W WHERE W.user_id = user_input)

---python script runs and returns Similarity(movie_id int, sim_score decimal(3,2))---

SELECT M.title FROM Movie as M NATURAL RIGHT JOIN Similarity as S
ORDER BY S.sim_score DESC
LIMIT 1

------------------------------------------------------------------------------------------------------------------------
/*Recommend me 10 movies that I have not watched anything similar to (not similar plot)
sorted by popularity*

WITH MovAll(movie_id, overview) AS (SELECT movie_id, overview FROM Movie)

WITH MovUser(movie_id, overview) AS (SELECT M.movie_id, M.overview FROM Movie as
M NATURAL LEFT JOIN Watched as W WHERE W.user_id = user_input)

---python script runs and returns Similarity(movie_id int, sim_score decimal(3,2))---

SELECT M.title FROM Movie as M NATURAL RIGHT JOIN Similarity as S
ORDER BY S.sim_score ASC, M.popularity DESC
LIMIT 10

------------------------------------------------------------------------------------------------------------------------
/*Search up “Monty Python and the Holy Grail” and tell me what is the probability that I will
like this movie */

WITH MovInt(movie_id, overview) AS (SELECT M.movie_id, M.overview FROM Movie as M
WHERE M.title = “Monty Python and the Holy Grail”)

WITH MovUser(movie_id, overview) AS (SELECT M.movie_id, M.overview FROM Movie as
M NATURAL LEFT JOIN Watched as W WHERE W.user_id = user_input)

---python script runs and returns Similarity(movie_id int, sim_score decimal(3,2))---

SELECT AVG(S.sim_score) FROM Similarity as S

------------------------------------------------------------------------------------------------------------------------
/*Find the 10 most popular movies with a plot similar to “Toy Story”*/

WITH MovAll(movie_id, overview) AS (SELECT movie_id, overview FROM Movie)

WITH MovInt(movie_id, overview) AS (SELECT M.movie_id, M.overview FROM Movie as M
WHERE M.title = “Monty Python and the Holy Grail”)

---python script runs and returns Similarity(movie_id int, sim_score decimal(3,2))---
```

```
SELECT M.title FROM Movie as M NATURAL RIGHT JOIN Similarity as S
ORDER BY S.sim_score DESC, M.popularity DESC
LIMIT 10
```

6)
Search through Kaggle.com for a sufficiently large movie dataset file in .csv. Currently https://www.kaggle.com/rounakbanik/the-movies-dataset?select=movies_metadata.csv looks promising. Maybe need to use python to split the single .csv table provided from kaggle into the multiple .csv's that matches each of our planned relations. From a quick google search, SQL seems to supporting import of .csv into relations in databases. Many database tools also have an option of importing a Flat File Source.

7)
We will have 2 types of outputs: Search results and Recommendation results.

Search results will be an output table listing the movie name and movie related information through a GUI or web interface.

Recommendation results will be an output table listing the movie name and attributes that the user wanted to filter by, outputted through a GUI or web interface.

8) Major: natural language processing
Minor: Object-oriented Implementation/ Advanced GUI form interface

9) Question:

1. Could we just implement our program like the way we did for homework3(php + mysql + better interface)?
2. Do we need to run our program in the ugrad machine?

```
        CREATE TABLE `link` (
  `movie_id` int NOT NULL,
  `imdb_id` varchar(45) DEFAULT NULL,
  `tmdb_id` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`movie_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
CREATE TABLE `mEcoBen` (
  `movie_id` int NOT NULL,
  `budget` int DEFAULT NULL,
  `revenue` int DEFAULT NULL,
  PRIMARY KEY (`movie_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `mDetail` (
  `movie_id` int DEFAULT NULL,
  `title` text,
  `tagline` text
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `mType` (
  `movie_id` int DEFAULT NULL,
  `adult` text,
  `release_date` text,
  `runtime` double DEFAULT NULL,
  `tagline` text,
  `title` text
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `mRating` (
  `movie_id` int DEFAULT NULL,
  `popularity` double DEFAULT NULL,
  `vote_average` double DEFAULT NULL,
  `vote_count` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `ratingDetail` (
  `rating_id` int NOT NULL AUTO_INCREMENT,
  `user_id` int DEFAULT NULL,
  `movie_id` int DEFAULT NULL,
  `rating` double DEFAULT NULL,
  `timestamp` int DEFAULT NULL,
  PRIMARY KEY (`rating_id`)
) ENGINE=InnoDB AUTO_INCREMENT=75182 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `top250` (
  `rank_id` int NOT NULL,
  `title` text,
  `rating` text,
  `genre` text,
  `ranking` double DEFAULT NULL,
  `director` text,
  `cast` text,
  PRIMARY KEY (`rank_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `watchedMovies` (
  `user_id` int DEFAULT NULL,
  `Watched_movie_id` int DEFAULT NULL,
  `record_id` int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`record_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```