“Up Next”: Movie Suggestion Application

Group 4: Logan Crooks, Allen Chang, Joel Foster

CS 564: Summer, 2020

## Table of Contents

[Introduction](#_heading=h.gdb4pmg9hatd)

[Motivation](#_heading=h.rfzxsoj4e5sr)

[Application Description](#_heading=h.qid23tkjjpax)

[Report Organization](#_heading=h.2u0dtmozbu6s)

[Task Assignment](#_heading=h.3zvwtdxw8cic)

[Our Implementation](#_heading=h.2bd6dror08vz)

[Description of the system architecture](#_heading=h.c7ba2typvztu)

[Description of the dataset](#_heading=h.f4b1fafwxcb5)

[Cleaning Data](#_heading=h.ag48tzasd0hu)

[ER diagram (final version from the previous checkpoint copied here)Note: this is not the one from checkpoint 4. This is a cleaned version with the addition of a disliked relation.](#_heading=h.md5fwc78e4xx)

[Relational model (final version from the previous checkpoint copied here)](#_heading=h.w2jyqmn709d3)

[Age(Birthday:char[10],Date of death:char[10],Age:int)   
  
and the Person relation will be the same as before minus the Age attribute: Person(ID: Char[9], Cast Name: String, Known for titles: comma delaminated string , Birthday: char[10], Height: int, Date of death: Char[10], Main Occupations: comma delimited string).](#_heading=h.iwcstmk3jlw1)

[Implementation](#_heading=h.d4oz5mkmbh4b)

[Evaluation: describe how you test your application (e.g, create testing scenarios or queries or something else, running your application through these scenarios/queries/etc.., checking the returned results and see how the results make sense or not).](#_heading=h.gjdgxs)

[Conclusion](#_heading=h.gat6nbrch76u)

[Important Mentions:](#_heading=h.1j9bvz9dku5k)

[References (if any)](#_heading=h.jomkcglpv2tp)

[Links:](#_heading=h.xikcu2kd8m7n)

## Introduction

*Motivation: why do you choose this problem and this domain? The importance of the application.*

*Describe the application.*

*Describe the overall organization of the report and task assignment for each team member*

#### Motivation

As a group of avid movie fans, we have noticed the need to curate a private and personal collection of movies, and have recommendations for more movies to add at your fingertips. This need isn’t just limited to the cataloging of movies, but having peripheral information such as actor and cast info, as well as ratings, geographic information, and so on readily available. Up Next is an attempt to gather all these types of information into features tied to a core movie database. Spanning the whole IMDB movie dataset, we hope to provide users the opportunity to curate their favorite movies across platforms, allowing them to save movies that appear on different streaming platforms such as Hulu or Netflix.

#### Application Description

The application is a user-based application, requiring account creation and username/password protection for individual users. Upon entering the application as a new user, a new user setup page is provided along with the option to immediately search for movies/actors that the user likes, and add them to the account. The user can go back to the login screen to use the credentials and login. Once the user logs in, immediately an interface displaying multiple recommendations for the users appears. These recommendations are based on the prior liked/favorited actors and movies. This interface allows users to select movies from the list, and learn more about each movie. This includes a synopsis of the movie, rating, actors list, and even a map displaying birth locations of related actors. An option to add the movie to a “liked” list also exists. Furthermore, a user settings page allows the user to view or alter these liked/favorited lists, modify account credentials, or even delete the account. The interface comes supplied with instructions for use on most frames as well.

#### Report Organization

We will format our report as similar to the template as possible for consistency. You can view the aforementioned table of contents for more information.

#### 

#### Task Assignment

|  |  |
| --- | --- |
| Person | Task |
| Allen | JDBC connection methods, querying methods, portion of GUI, population and loading data. |
| Logan | Bulk/Majority of the GUI Front-end. Interfacing JDBC elements into the front-end. |
| Joel | Smaller portions of GUI/JDBC Connections, Population and loading data, testing, documentation. |

## Our Implementation

### Description of the system architecture

The application is a Java Stand-alone application, utilizing JDBC and MySQL for the backend. Here is a diagram to visualize the project structure:



Specifics:

*mysql Ver 8.0.20 for Win64 on x86\_64 (MySQL Community Server - GPL)*

*openjdk version "11.0.6" 2020-01-14*

*OpenJDK Runtime Environment AdoptOpenJDK (build 11.0.6+10)*

*OpenJDK 64-Bit Server VM AdoptOpenJDK (build 11.0.6+10, mixed mode)*

### Description of the dataset

The datasets were retrieved from Kaggle, and include two main datasets:

1. [IMDB Movies dataset](https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset?select=IMDb+movies.csv)
   1. movies.csv
   2. names.csv
   3. places.csv
2. [World Cities Dataset](https://www.kaggle.com/max-mind/world-cities-database?select=worldcitiespop.csv)
   1. worldcitiespop.csv

Below is the number of columns and rows per each dataset

|  |  |  |
| --- | --- | --- |
| Dataset | Number of columns | Number of rows |
| IMDB Movies | 22 | 81,273 |
| IMDB Names | 20 | 175,715 |
| IMDB Ratings | 49 | 81,273 |
| World Cities | 7 | 2,375,760 |

Specifics on the raw data formatting can be found via the links above to the Kaggle site.

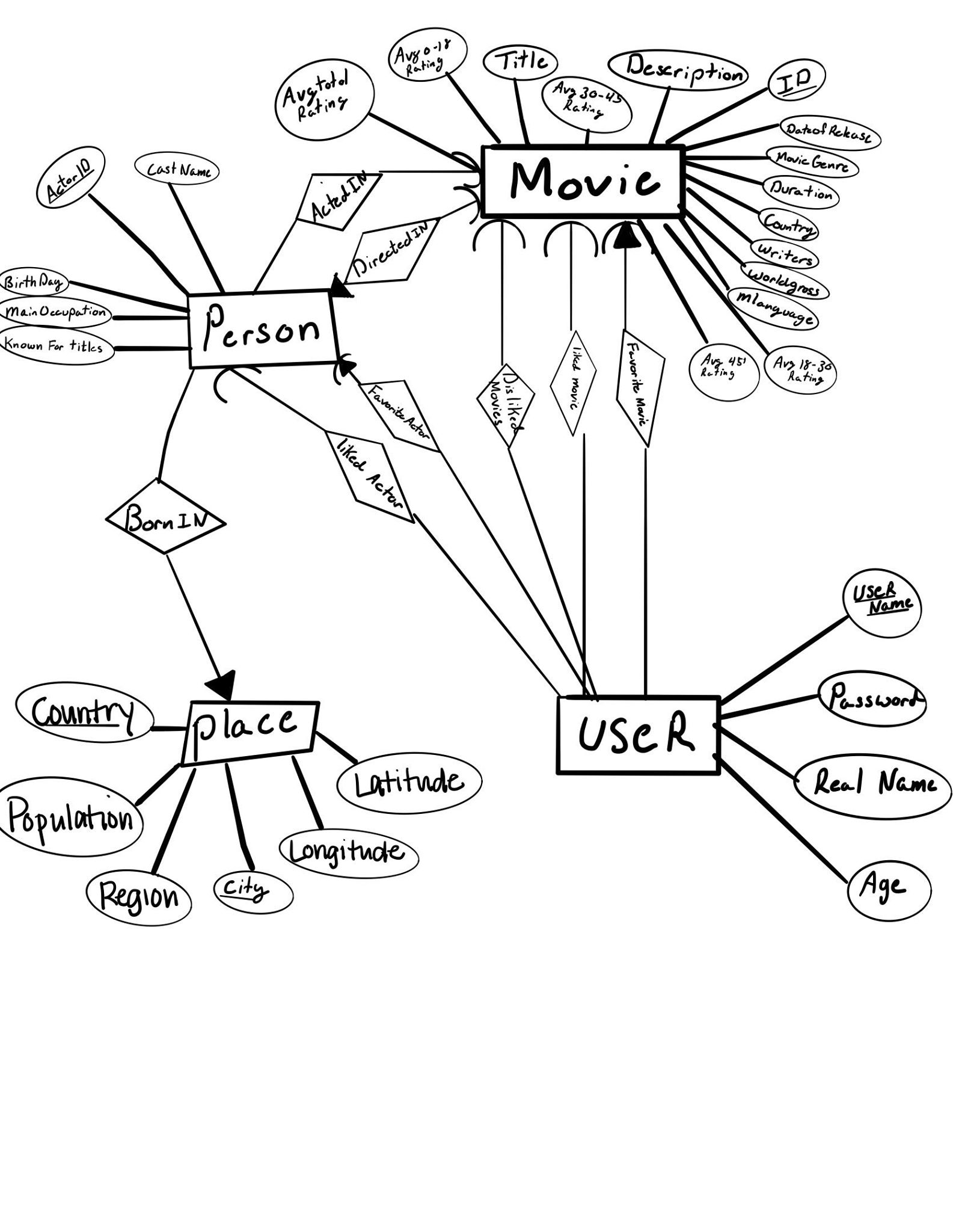
#### Cleaning Data

Out of the box, our data was far from ready to be loaded into tables. This included irrelevant columns, duplicates that would cause errors with Primary Keys, Unicode character issues, escape character issues that led to complications with table formatting and searching, data type conversion issues when loading, and so on.

We implemented the pandas library to generate cleaned datasets. We have provided those data sets in our submission. This was a major portion of the difficulty of populating, and was more time consuming than expected.

#### 

### ER diagram (final version from the previous checkpoint copied here)Note: this is not the one from checkpoint 4. This is a cleaned version with the addition of a disliked relation.



### 

### Relational model (final version from the previous checkpoint copied here)

**Relation Person**(ID: Char[9], Cast Name: String, Known for titles: comma delaminated string , Birthday: char[10], City:varchar(6),Country:varchar(16), Main Occupations: comma delaminated string)

**Relation Movie**:(ID: Char[9], Title: String, Date of Release: char[10], Movie Genre: comma delaminated string, Duration: int, Country: char[2], World gross income: int , Language: string)

Total Weighted Avg: double, avg0:double, avg18:double, avg30:double, avg45:double )

**Relation Place** ( City Name: string, Country: char[2], Longitude: Double, Latitude: Double)

**Relation ActedIn**(ID(movie): Char[9], ID(person): Char[9])

**Relation Directed**(ID(movie): Char[9], ID(person): Char[9])

**Relation Born in**(ID(person): Char[9], Country:Char[2], City name: String)

**Relation Likes(Movie)**(Username:varchar[15],ID(movie):char[9])

**Relation Dislikes(Movie)**(Username:varchar[15],ID(movie):char[9])

**Relation Favorite(Movie)**(Username:varchar[15],ID(movie):char[9])

*Note: Can only be a single instance of a unique username in this relation*

**Relation Likes(Person)**(Username:varchar[15],ID(person):char[9])

**Relation Favorite(Person)**(Username:varchar[15],ID(person):char[9])

*Note: Can only be a single instance of a unique username in this relation*

**Relation User**(Username:varchar[15], Real name:varchar[25], password:varchar[15], Age: int)

**Functional Dependencies:**

ID(Movie)->Title, Date of Release, Movie Genre, Duration, Country, World gross income, Language, Total Weighted Avg, Both Sex 0-18, Both sex 18-30, Both sex 30-45, Both sex less than 18

* All permutations of ID determination any set of the right

ID(person)->Cast Name, Known for titles, Birthday, Age, Height, Date of death, Main Occupation

-All permutations of ID determination any set of the right

Birthday, Date of death->Age

City Name, Country-> Longitude, Latitude

ID(person) -> Country, City name

ID(person)->City Name, Country, Longitude, Latitude

ID(Movie)->ID(Person)

ID(Movie)->ID(person), Cast Name, Known for titles, Birthday, Age, Height, Date of death, Main Occupation

Username-> Password, Username, Realname, age

Updated to 3NF(If not mention One above is already in 3NF):

#### Age(Birthday:char[10],Date of death:char[10],Age:int) and the Person relation will be the same as before minus the Age attribute: Person(ID: Char[9], Cast Name: String, Known for titles: comma delaminated string , Birthday: char[10], Height: int, Date of death: Char[10], Main Occupations: comma delimited string).

Note: We never use Age. Therefore, we Matian the age within the person because it is never actually used. It is left in to allow for further changes in the program.

### **Implementation**

The application starts at the Login screen, where users can enter their login credentials. Alternatively, they can press the Create New User button, which takes them to a new page. In this page, there are 5 sections for the user to fill out. First, their user information which is username, password, real name, and age. Next, there is a search bar where users can search for movies and add them to their list of likes. Then, the user can designate one as their favorite. Similarly, there are two more sections where the user can search for actors/directors and add them to their liked people lists and also designate one as a favorite. When the user is done, they press submit. All of the info is sent to the server side where tuples are added to the UserProgram, LikedMovies, FavoriteMovies, LikedPeople, and FavoritePerson tables.

Once the user logs in, they are taken to a new page where there are four lists. Each list recommends movies to the user based on the info they input such as their favorite movie’s genre, actors they like, and directors they like. Using SQL, each list is created from a query that utilizes subqueries to ensure no movies that the user already likes or dislikes appear and ordering to show the top 15 movies by average rating from people their age. We can do this because for each movie, we store its average rating based on age in the Movies table.

If the user clicks on any movie in any list, they have the option to see more about the movie. This takes the user to a Movie info page. Here, a query is sent to the server side to retrieve info about the movie such as Title, Description, Date, Gross, Writers, and more. The server side also retrieves information from the ActedIn and DirectedTables to display the actors and directors of the movie.

On the Movie info page, there are also buttons. If the user clicked “Like this movie?” then that movie will be added to the users liked list in the MySQL database. If the user clicks “Don’t like this movie”, then that movie will never show up on the recommendations again due to the use of the DislikedMovie table in our recommendation queries. If the User clicks on an actor or director, then the user will be able to add that actor or director to the users liked people list.

If the user clicks on “Wanna See A Cool Map?”, the user is taken to a new page where a world map displays with cyan dots all around the world. Each dot corresponds to the location that one of the actors was born in. The size of the dot represents the order that the actor was billed in the movie. The larger the dot, the higher the billing

When the user is back to the homepage where the movie lists are, there is another button called User settings. This takes the user to a new page where all of the user’s information will be displayed. User Name, Password, Age, Real Name can all be updated at this location. The user can also remove favorites and liked movies or people as well as add a new favorite movie or person from their liked list. Finally, if the User clicks “Delete user”, the user will be removed from the database and brought back to the Login screen.

Further examples of our prototype can be further seen in the video that will be included in the submission as well as the images added to the appendix.

**Query Performance**

The speed of our application was quite underwhelming, due to the complicated queries being run. The queries for recommending movies used too many tables and heavily slowed the process.

Our goal in recommending movies was that they should not be in the users LikedMovies list, FavoriteMovie, or DislikedMovie list. That is already the use of three tables unioned together. In addition, the movie must have at least one person from the users LikedPeople and FavoritePerson entries. There also cannot be any duplicates, meaning we must use the ‘DISTINCT’ keyword. Finally, the lists must be ordered by rating. Our most complicated query, which recommends movies that have a liked director and actor, utilizes 8 select statements and 8 tables.

Given more time to work on this project, this would have to be heavily optimized. The question is whether the query should be optimized or the tables should be reorganized. One solution is to add attributes for favorite movie and favorite person to the ProgramUser table itself, eliminating the need of two tables right off the bat. Furthermore, there is a question of whether the DislikedMovie table is really needed. It’s a matter of convenience for the user to not see movies they don’t like, but doesn’t have much use outside of that. One way to solve the problem of having to search through multiple tables for what movies to not recommend is to simply make one big DoNotRecommend table.

Alternatively, we could simplify the queries themselves. We could choose to only focus on Liked people in one list, then Favorite person in another list. We could have multiple lists for various different attributes, but limit the number of lists that use multiple attributes.

On a positive note, there were other query speed issues that we were able to successfully fix in our time constraint. First was the Movie info page. This page was originally quite slow, as we relied on the ‘listofactors’ attribute in the Movies table. We had to use functions to parse that string into an array of names, then match it to the Person table. After replacing this method with simply using the ActedIn table, loading Movie info time decreased dramatically.

Another issue was attempting to create a table using a join with a LIKE clause. This was extremely time consuming as MySQL was using the Cartesian product of the tables. To help speed up the process, we only joined two tables at a time, and on only one attribute. Therefore, we split our original WorkedOn table into ActedIn and Directed. This ended up very useful when we wanted to separate our queries based on liked directors and liked actors. In addition, instead of creating a table directly from join, the results of a join were exported to a csv file, then loaded in using LOAD INFILE, which ended up being a much faster process.

Overall, while we would like to see the application function faster in terms of listing recommendations, we are still happy with the results of it. We were able to successfully curate lists of movies that satisfy the criteria we wanted. No duplicate movies, no movies that have already been liked or disliked. The queries accomplished what we wanted them to.

### Evaluation: describe how you test your application (e.g, create testing scenarios or queries or something else, running your application through these scenarios/queries/etc.., checking the returned results and see how the results make sense or not).

We went through three rounds of testing. The initial test known as the alpha testers are the people who wrote the program itself, Logan, Allen, and Joel, the point of the alpha testers is to test the basics cases laid out in check point four.

Note:

Green Highlight = functionality/correct response

Yellow Highlight = functionality/correct response for all apparent cases

Red Highlight = not functional/incorrect response

The Cases that were evaluated by the Alpha Group:

1. **Functional querying for the movie suggestions**
   1. For suggesting movies by liked actors, do not suggest movies that already exist on the users liked movies/favorite movie lists
      1. To do this, navigate to user settings, look at list of liked content/favorited
      2. Then cross-check on the suggested content
   2. For suggesting movies by liked genre, do not suggest movies that already exist in the users liked movies/favorite movie lists
   3. For suggesting movies by liked directors/writers, do not suggest movies that already exist in the users liked movies/favorite movie lists
   4. Check that a user having **no entries** for “liked” relationships or “favorite” relationships (for genre, person, movie, director/writer) has query results that are completely empty.
   5. Check that the query taking in All three attributes (genre, actor, movie) also produces unique movie titles that have not already been liked/favorited by user.  
      (same process)
   6. Check that search results are further sorted by ranking, and only top x (x tbd) of the search results are shown \*if that specific number of is needed, we will specify later on
      1. Check the first entry. Open dialog for more info on the Movie. Check ranking
      2. Compare to second entry
      3. Ranking should be decreasing. Repeat down the list.
2. **Maintaining integrity of data tied to specific user, and allowing full ability to modify**
   1. Check that one user can only have one favorite actor/movie/other relevant movie attribute (this constraint is actually done inside the GUI instead of a Constant in the DBMS)
      1. Go to “don’t have a login” button
      2. Type in a batman in the movie label and hit search
      3. Click on batman a popup will ask if you would like to add batman to like list; hit yes
      4. Then Batman is in the list under the search button. Click it again
      5. A popup will say would you like to add batman to favorite movie
      6. Hit yes
      7. When you attempt to click on batman in the second list a popup should not appear

* 1. Check that a favorite item also can exists in like item for user but does not have to
     1. Log in to “User” and “Password”
     2. Hit log in
     3. Navigate to “User setting” button
     4. Look at liked movie and favorite movie list and make sure the favorite movie name is also in the list of liked movies
  2. Check that user can delete from all liked lists
     1. Do steps 1-3 above
     2. Click on movie in like movie list
     3. A popup will appear asking if you would like to remove “movie name” from movie list
     4. Hit yes
     5. Do steps 2-4 for all movie is liked list
     6. You should see an empty list
     7. Do steps 1-6 for liked actors
  3. Check that user can use search terms to add to every liked list
     1. Go to “don’t have a login” button
     2. Type in a batman in the movie label and hit search
     3. You should see batman appear in the list below
     4. Type in “Bill Murray” into the actor label hot search actor
     5. You should see bill Murry appear in the list below

1. **Login verification** 
   1. Test if invalid credentials during login return an error message
      1. Type in “UserFail” into username and “Passwordfail” into password
      2. A popup should appear saying the credentials are incorrect

See Fir.invalidLogin

* 1. Test that there only exists a single account for a single User/Pass pair
     1. Print all distaitice user tuples with select User and Password
     2. Intersect that with a list of all tuples of user and password
     3. The intersection should be the same as the distinct user and password
  2. Test that an already created login account has listings in tables for likedMovies, likedActors
     1. Log in
     2. navigate to user setting
     3. Look at liked people/movies list
  3. Test that a new user cannot register with already existing userName/Pass
     1. Create new account, login
     2. Log out, go to new user page, login with same credentials
     3. Should fail, otherwise we have a problem
  4. Test that new user doesn’t have matching credentials in any of the “liked” tables
     1. Set up new account credentials. Don’t set any liked/favorite preferences
     2. Now run a recommended query. Nothing should return.

1. **Interface properly interacts with Database**
   1. Add test users in the create new user page and ensure that database adds new tuples in the liked movies, liked people, program user tables
      1. Click “don’t have a log in”
      2. Type in “batman” in the movie tab and hit search movie button
      3. Click on the movie in the list under the location where you typed the movie

Hit yes to the popup

Then the movie name should be added to the list under the search button

* 1. Test that when users remove movies that they like or people that they like, the tuples are properly deleted from appropriate tables
     1. Login with “User” and “Password”
     2. then hit login
     3. navigate the “user setting” button.
     4. Click on a movie you wish to delete. A popup will appear asking you if you want to delete this movie.
     5. Once yes is pushed you should imeadly see that the movie disappeared from your liked list
  2. Try creating a user, password combination that is already in the database and make sure that no new tuples are added to any table when this is the case
     1. Click Don’t have a login” button
     2. Type in “User” and “Password” in the “ Username and password followed by “paul” and “ 8” and an error should popup

**SEE Fig.TakenUserName**

* 1. Make sure that user cannot be created when their age is not an integer
     1. Under the “don't have a login tab” attempt “User1” and “Password” and “Real Name” and “ten”; this will result in an error popup

**See Fig.ageError**

* 1. Test that user can change their age and name in the settings page and the proper tuples in tables are update
     1. Login with “User” and “Password”
     2. then hit log in navigate the
     3. “user setting” button
     4. replace age in the label and hit
     5. “Submit User age” button the age
     6. to the left of the button should immediately update to the the age you have typed in

The Second set was a set of beta testers, one person from each age demographic listed in the movie ratings category.

Nathan- in the range 0-18 (He is 16, Logan’s Brother

Beatrice - in range 18-30 (She is 21, Logan’s Friend)

Rosella -in the range 30-45(She is 39, Logan’s Mother )

Cindy- in the range 45+(She is 62, Logan’s Grandma)

Gracie- she is a cat(Cat, Logan’s Cat)

The purpose was to receive real feedback from a person not familiar with the program from each age demographic in the program. We gave them each the program and asked them to rate the program out of 5 and say one thing they would change about the program.

Here were the results:

|  |  |  |
| --- | --- | --- |
| Person/age | Rating out of 5 | Quote |
| Nathan/16 | 4 | “I gave it 4 out of 5 stars because the map was cool but I had some difficulty coming up with liked actors and Movies when creating a new account” -Nathan |
| Beatrice/21 | 3.8 | “I think the 3.8 is pretty good, because what does this have to offer over something like Netflixs Database?”  -Beatrice |
| Rosella/39 | 4.2 | “I gave it 4.2 because We couldn’t add more than one favorite? I was just repeatedly clicking trying to add another favorite movie and nothing happened” - Rosella |
| Cindy/62 | 3.1 | “I gave it a 3.1 because I Liked the shapes and the buttons, but for the life of Me I could not figure out what to do and also the text was kinda small”-Cindy |
| Gracie/cat | .6 | “This Program is biased against things without fingers! Wait,how am I speaking? Anyway, I say .6 stars!” - Gracie |

It was important that we took in the feedback from the beta test group and improved our program.

The younger groups had complaints in the vein of ease of use and questioned what the point of the program was. To solve these problems we added the ability to not add a large number of liked movies or people for the program to start suggesting Movies because over time the recommendations will learn and adapt to the users' likes and favorites. The point of this program is separate from platforms such as Netflix or Hulu, because our data set exceeds the movies on any one platform. Therefore, this program's purpose is to suggest movies if the user has more than one streaming platform. The program is not biased for one streaming platform. It also includes an interesting map feature.

The older age groups had complaints in the vein of how to use the program and how some interactions had a lack of feedback response. The old group had less intuition on how tech/software works. Therefore, to not alienate half of our users, we added instruction buttons along every step. We also made the buttons a universal color (yellow) to allow an increase in Intuition. To fix the lack of feedback response (ie. when you update the user password), we made it so a popup appears confirming that the user password change has been confirmed.

Once the changes to the program had been completed we had a third testing set. Unfortunately, due to the lack of people, it was the same people in the original beta group. We asked for a program rating of 0-5 and all demographics that are meant to use the program had an increased rating from the initial response due to our changes in the program.

Even though the beta testing was incredibly time consuming, and the time could arguably be put towards different aspects of the program, we believe that it allowed us to create a program, in particular the GUI, that can be used by a large variety of users. The alpha group can test the case that we think of. However, we have a bias on the usability, and therefore are blind to the program's faults. The beta group opened our eyes to these faults and allowed us to correct them.

## Conclusion

*What do you learn from this project (both interesting and uninteresting points)? Have you found any relevant database knowledge you have learned in this course helpful and have you encountered any database relevant issues that have been discussed in this course?*

We have learned many things over the duration of the program, many are regrets about the time constraints of this eight week course and wanting our program to have too many functions.

The first regret we had was having not clean data and particularly attempting to join two different data sets together. We had two data sets. The Movies data set that contains a person csv file and a Places data set. We had to join the data set consisting of 180K people with a Places data set consisting of over 3 million places. However, the Place data set was very dirty and once cleaned was about 2.6 million. To make it even more time consuming the person csv and the places csv had a completely different format of the countries so we had to include a third data set to merge the two larger data sets together.

The second regret is the raw amount of data we chose to insert into the program. We had over 80K movies, 170K people, and 2 million places. Many of these sets had attributes that were never used in the program itself. We just added it into the database. We were originally under the assumption that the more data(attributes), the better, so we might as well just place them inside the database. However, this just causes formatting errors when attempting to load the data and it took more memory to store each tuple. To not even use the data for any purpose, we learned more data is not always better. We didn’t use the attributes such as place population or person height or birthday. This can partially relate to issues discussed in class. The large amount of data would be very helpful if there is something that we could index off of to create a hash function or a B+ tree as it would solve the processing time issue. However, many of our relations are user-dependent; therefore, a constant adding and removing (like in a B+ tree) would require more processing time than it would to search through the limited number of liked people and movies.

We have learned the typical aspects, such as how to use a DBMS and how to use Mysql and how to connect a java language to a DBMS. However, we believe that learning what ‘not’ to do in the future was a far more important lesson.

#### Important Mentions:

Let’s talk about the places data set to clean. When it was all finished, it only corresponded to about 30K of our actors out of the 174K, so It is not the most influential data set. However the experience of finally seeing the work (even if it was a small percentage) was so satisfying. You can see in fig.map the dots are clearly displayed. I am sure if time allowed we could have better cleaned the data to have a larger overlap between the location actors were born in and a latitude and longitude position on a map. However, the functionality was only finished on 7/29 and 3p.m. Therefore, Further Cleaning was not possible. A problem also occurred with the correspondence between the country in the places database and Countries of the actors birthplace.

Also due to the time constraints our ER Diagram needed to be changed when we couldn’t complete everything. We successfully created the born table. However, we did not meet the constraints displayed on the ER Diagram. The places entity has places that an actor was not born in, and even though each actor only has one place they were born, not all actors have an entity place where they were born. Therefore see fig.newERDirgram.

*For a video of our final prototype, please follow the link provided in our References portion.*

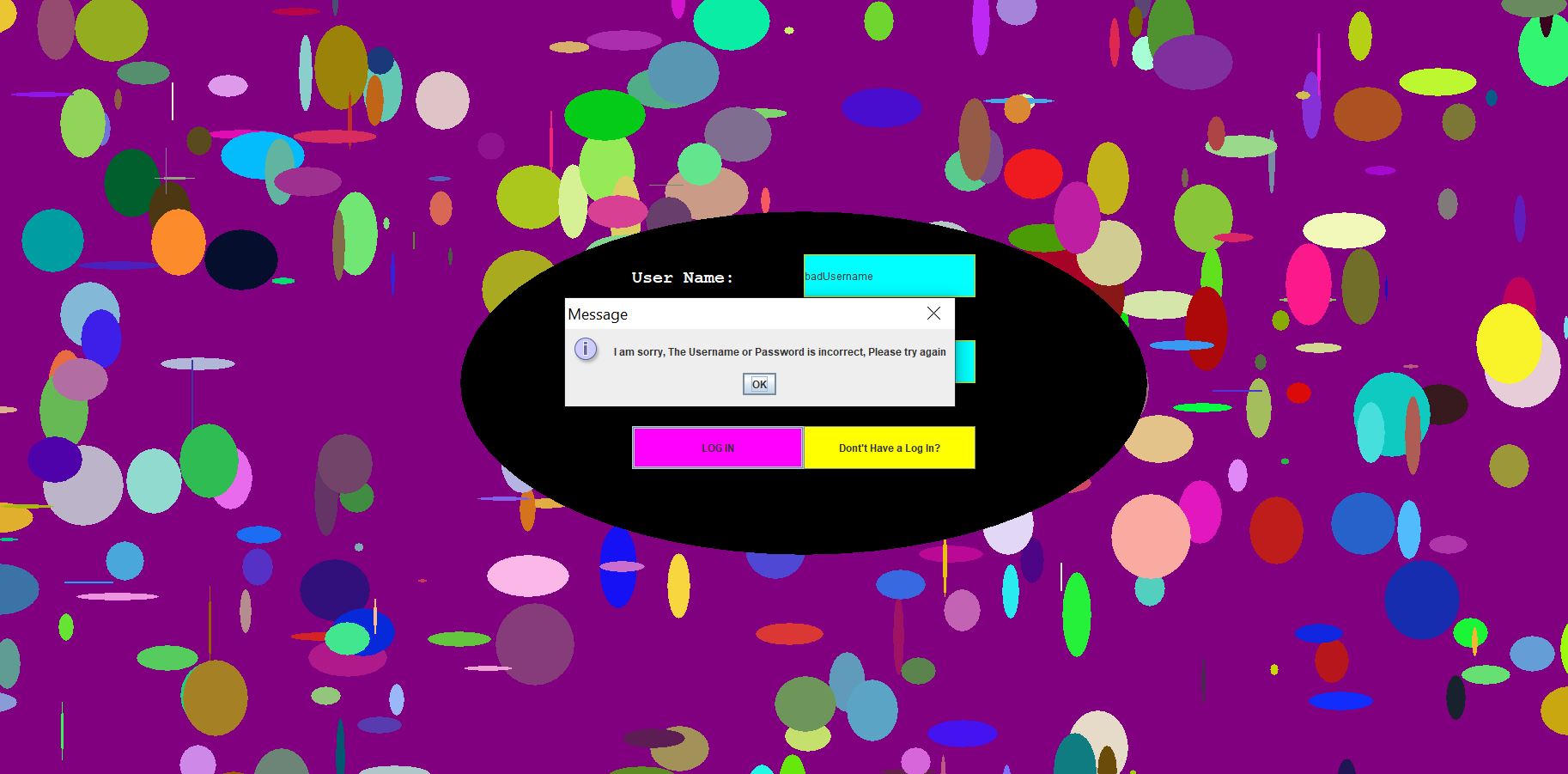
## References (if any)

### Links:

[Our GUI Walkthrough video](https://drive.google.com/file/d/1sWymTTiWD4jAARSLY6bIjeqyCeQQ5SQK/view?usp=sharing)

[Pandas Library](https://pandas.pydata.org/)

*Fig.invalidLogin*



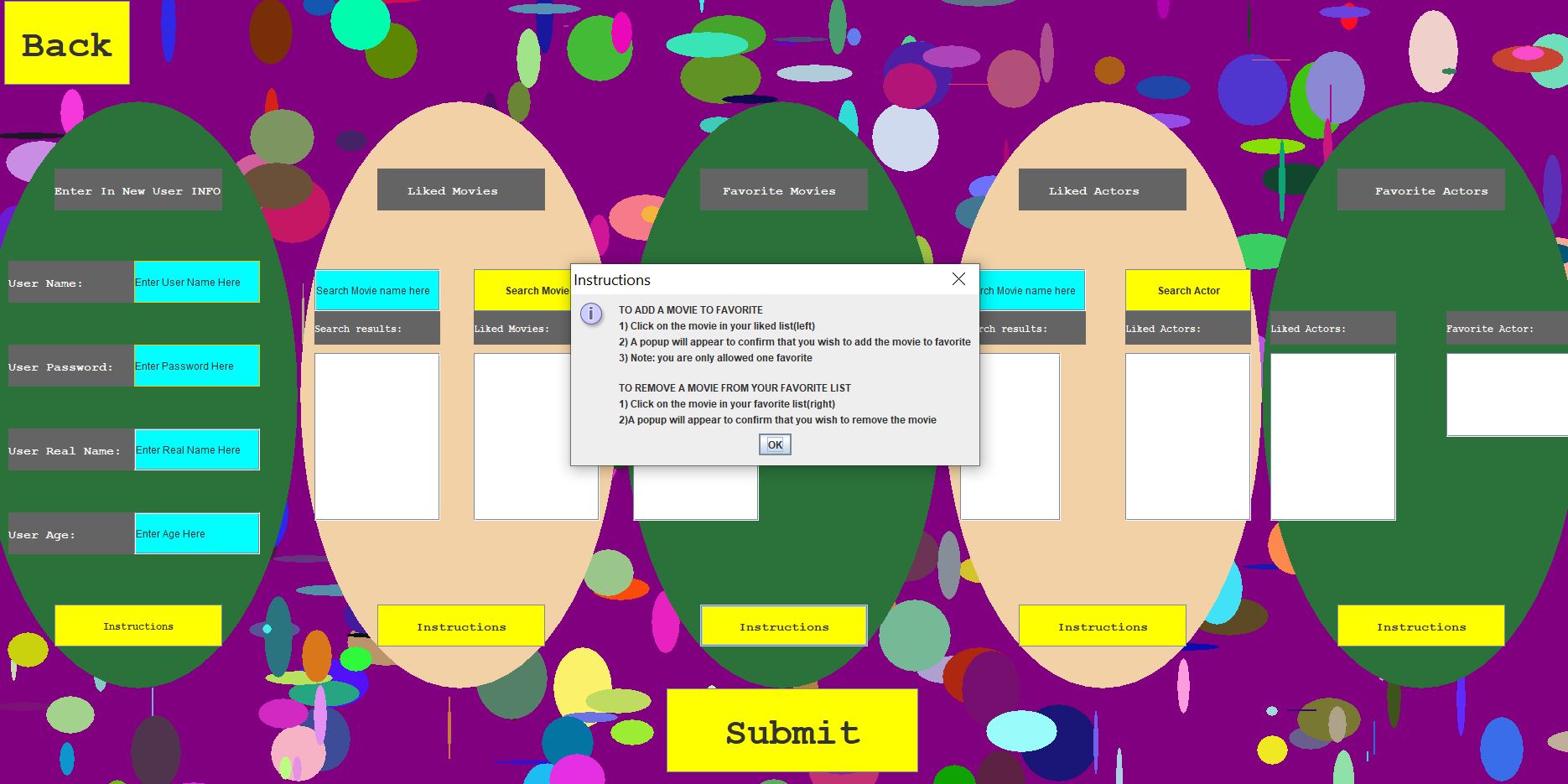
*Fig.TakenUserName*



*Fig.ageError*



*Fig.instructions*



*fig.newERDirgram.*

