Title of the Project: IMDB Management System

Name of the students: Jianbang Sun, Yibo Kong, Yuhan Zhang

I. Introduction:

The history of the film and TV shows is over 100 years. During these 100 years, countless movies and TV shows have been created. In order to find the movie and TV shows they want to watch among the huge number of movies and TV shows, viewers need to know some basic information, such as the cast list, description, rating, etc. In order to facilitate viewers to search and learn movie and TV shows information, we want to develop an application that can help viewers to find the movie and TV shows they want to watch.

Our application can implement basic management functions, such as searching movies or TV shows based on their names or ID, deleting or modifying information of existing movies or TV shows, and adding movies or TV shows. Similar to managing movies and TV shows, we also developed functions for managing actor information. In addition, we implemented some complex and useful functions, such as searching the top rated movies or TV shows, showing the cast list of a movie or TV show, and showing the characters an actor played. The applications we have now are developed more for the information management side, i.e. the company or the staff. If we can add properties such as permissions to our application and prevent users from changing the contents of the database, it may also be accessible to the users.

The three people in our group worked equally on the prep work such as identifying the domain, finding the data, and determining the database structure. During the development of the application, Jianbang Sun and Yuhan Zhang were responsible for developing the front-end and back-end, implementing the basic database management functions, and connecting the database to the front-end. Yibo Kong was responsible for processing the data, building the database, and writing the stored procedures. In the next report, we will describe the data we use and the structure of the database. We will also describe our implementation and evaluate it. Finally, we conclude with a summary of how the program performed and what we learned.

II. Our Implementation

We use java swing classes to build our GUI and use MySQL to manage and query the database. We also use mysql-connector-java-8.0 to connect the database with the java program.

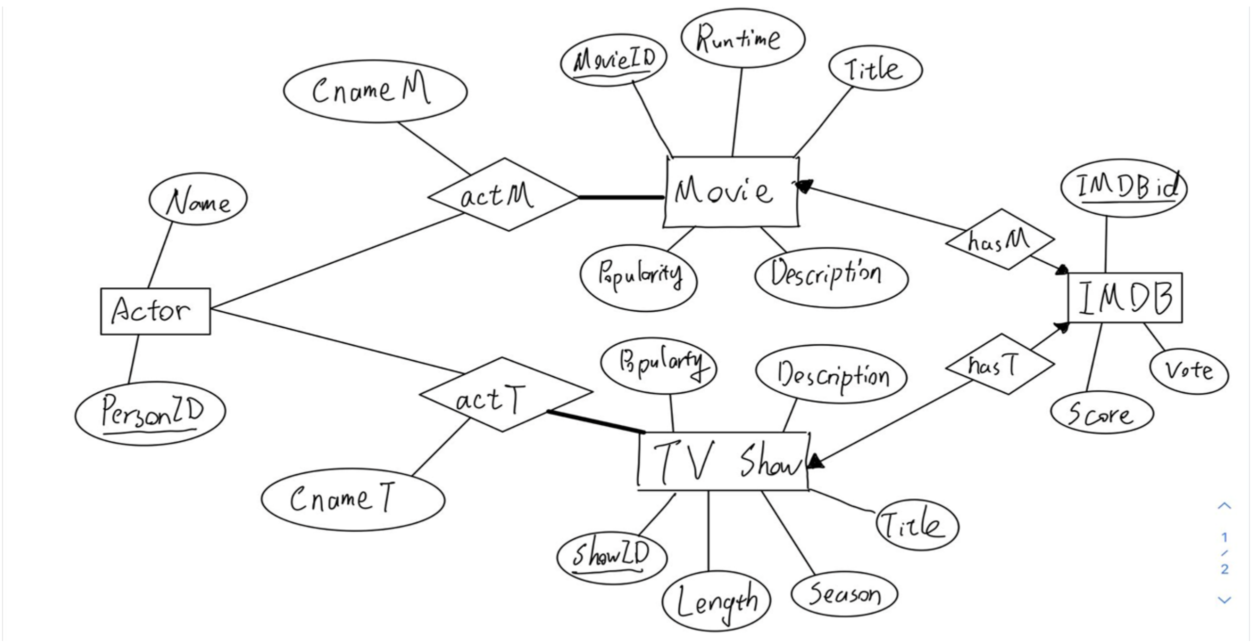
The main purpose of this software is to allow users to directly add, delete, or modify data in the database from the front end. And allow the user to easily view specific data by calling Stored Procedure.

We get the dataset from Kaggle, a website that contains a massive dataset. Here is the URL of the dataset: https://www.kaggle.com/datasets/victorsoeiro/netflix-tv-shows-and-movies?resource=download .

The dataset we have chosen contains all movies and TV shows on Netflix and cast lists up to May 2022. It contains basic information of the movie or TV series such as the description, year, length, title, ID, and IMDB score. It also contains the actor's personal ID, name, movie or TV series he/she was in and the character name. After our processing, we divided the dataset into 4 entity sets (actor, movie, TV show, IMDB) and 4 relationship sets (actM, actT, hasM, hasT). There are 3759 records of movies, 2047 records of TV shows, and 53957 records of actors. Since the relationships between movies or TVshows and IMDB are one-to-one, we do not use hasM or hasT in our database. In addition, there are some names that are not in English because actors come from all over the world. These names cause MySQL unable to handle queries about the actor's name. Thus, our application can only look up the roles an actor has played using their personal ID and look up the actor's name using their personal ID. And the cast lists of movies or TV shows contain only the person IDs of actors.

Here is the ER diagram and relational model of our database:

ER diagram:



Relational model:

(Since the relationships about IMDB are all one-to-one, we have merged hasM and hasT)

Actor(<u>*Person ID*: integer</u>, *Name*: string)
    Person ID is the primary key
actM(<u>*Person ID*: integer, *Movie ID:* string</u>, *CnameM:* string)
        (Person ID, movie ID) is the primary key
actT(<u>*Person ID*: integer, *Show ID:* string</u>, *CnameT:* string)
        (Person ID, show ID) is the primary key
movie(<u>*Movie ID*: string</u>, *TitleM*: string, *Description*: string, *Runtime*: integer, *Popularity:* double, *IMDB ID*)

Movie ID is the primary key

(TitleM, Description) is also a key

TV show(<u>*Show ID*: string</u>, *Title*: string, *Description*: string, *Length*: integer, *Season*: integer, *Popularity*: double, *IMDB ID*)

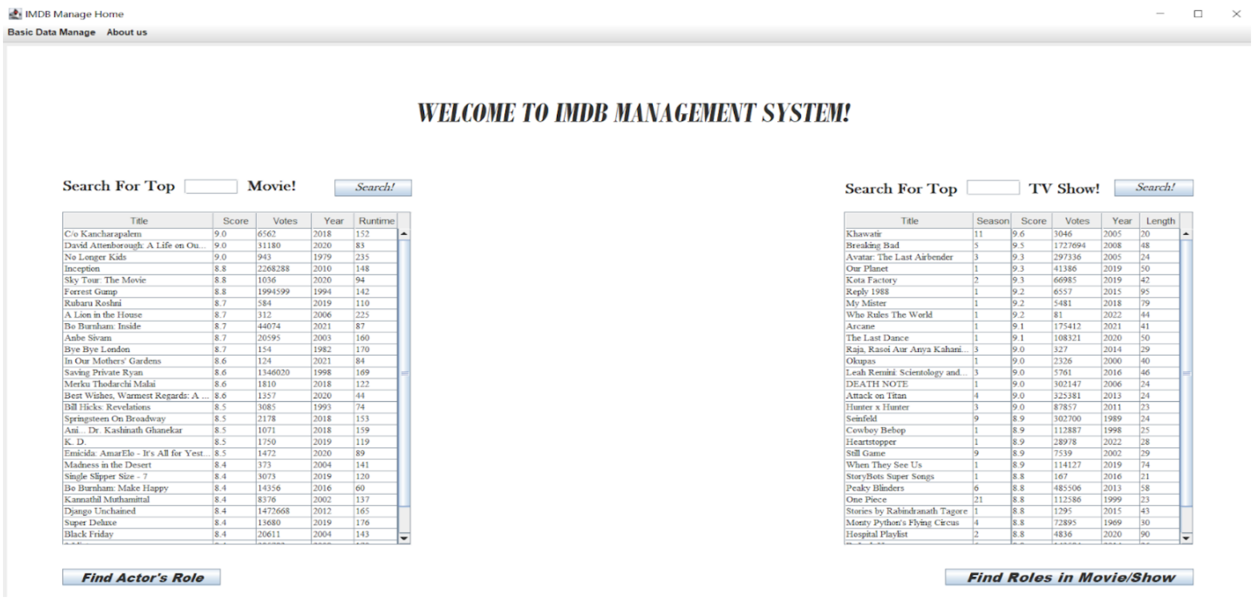
Title ID is the primary key

(Title, description) is also a key

IMDB(<u>*IMDB ID*: string</u>, *Score*: float, *Votes*: integer)

IMDB ID is the primary key

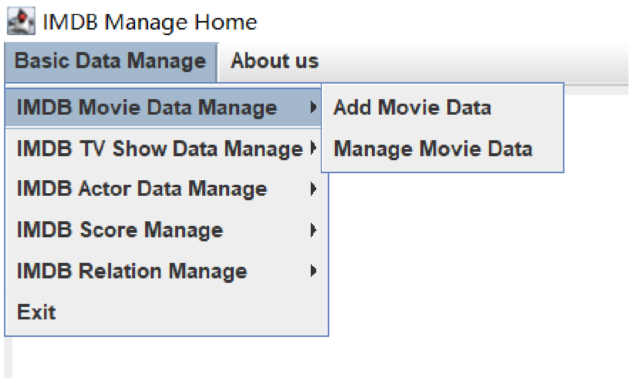
We have three main interfaces for our project, named MainLeftFrame, MainFrame, and MainRightFrame. The three main pages can be toggled with buttons at the bottom. The functions of our mainframes are designed for the stored procedures.



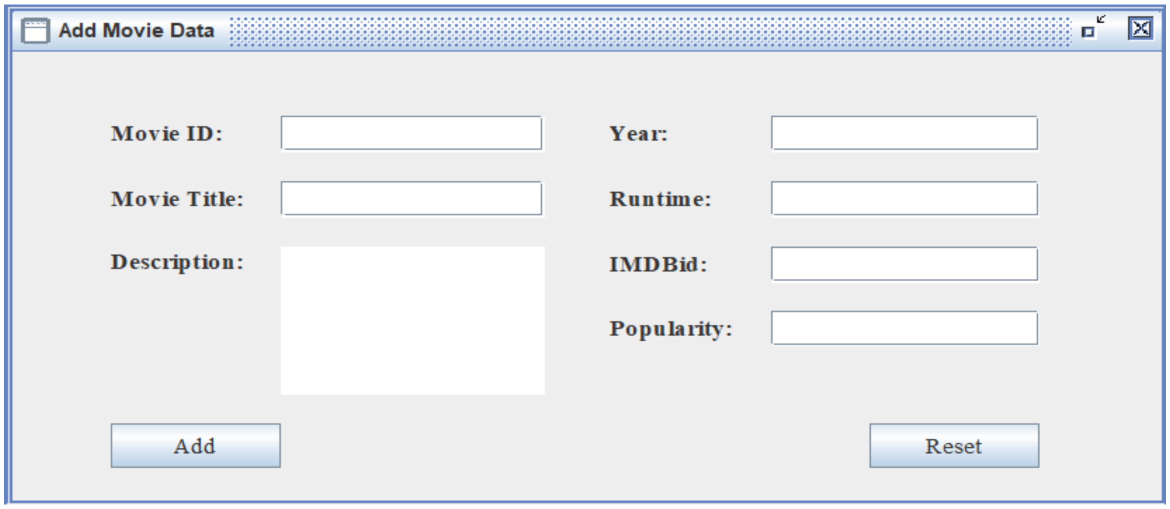
Title	Score	Votes	Year	Runtime
C/o Kancharapalem	9.0	6562	2018	152
David Attenborough: A Life on Ou...	9.0	31180	2020	83
No Longer Kids	9.0	943	1979	235
Inception	8.8	2268288	2010	148
Sky Tour: The Movie	8.8	1036	2020	94
Forrest Gump	8.8	1994599	1994	142
Rubaru Roshni	8.7	584	2019	110
A Lion in the House	8.7	312	2006	225
Bo Burnham: Inside	8.7	44074	2021	87
Anbe Sivam	8.7	20595	2003	160
Bye Bye London	8.7	154	1982	170
In Our Mothers' Gardens	8.6	124	2021	84
Saving Private Ryan	8.6	1346020	1998	169
Merku Thodarchi Malai	8.6	1810	2018	122
Best Wishes, Warmest Regards: A ...	8.6	1357	2020	44
Bill Hicks: Revelations	8.5	3085	1993	74
Springsteen On Broadway	8.5	2178	2018	153
Ani... Dr. Kashinath Ghanekar	8.5	1071	2018	159
K. D.	8.5	1750	2019	119
Emicida: AmarElo - It's All for Yest...	8.5	1472	2020	89
Madness in the Desert	8.4	373	2004	141
Single Slipper Size - 7	8.4	3073	2019	120
Bo Burnham: Make Happy	8.4	14356	2016	60
Kannathil Muthamittal	8.4	8376	2002	137
Django Unchained	8.4	1472668	2012	165
Super Deluxe	8.4	13680	2019	176
Black Friday	8.4	20611	2004	143

Title	Season	Score	Votes	Year	Length
Khawatir	11	9.6	3046	2005	20
Breaking Bad	5	9.5	1727694	2008	48
Avatar: The Last Airbender	3	9.3	297336	2005	24
Our Planet	1	9.3	41386	2019	50
Kota Factory	2	9.3	66985	2019	42
Reply 1988	1	9.2	6557	2015	95
My Mister	1	9.2	5481	2018	79
Who Rules The World	1	9.2	81	2022	44
Arcane	1	9.1	175412	2021	41
The Last Dance	1	9.1	108321	2020	50
Raja, Rasoi Aur Anya Kahani...	3	9.0	327	2014	29
Okupas	1	9.0	2326	2000	40
Leah Remini: Scientology and...	3	9.0	5761	2016	46
DEATH NOTE	1	9.0	302147	2006	24
Attack on Titan	4	9.0	325381	2013	24
Hunter x Hunter	3	9.0	87857	2011	23
Seinfeld	9	8.9	302700	1989	24
Cowboy Bebop	1	8.9	112887	1998	25
Heartstopper	1	8.9	28978	2022	28
Still Game	9	8.9	7539	2002	29
When They See Us	1	8.9	114127	2019	74
StoryBots Super Songs	1	8.8	167	2016	21
Peaky Blinders	6	8.8	485506	2013	58
One Piece	21	8.8	112586	1999	23
Stories by Rabindranath Tagore	1	8.8	1295	2015	43
Monty Python's Flying Circus	4	8.8	72895	1969	30
Hospital Playlist	2	8.8	4836	2020	90

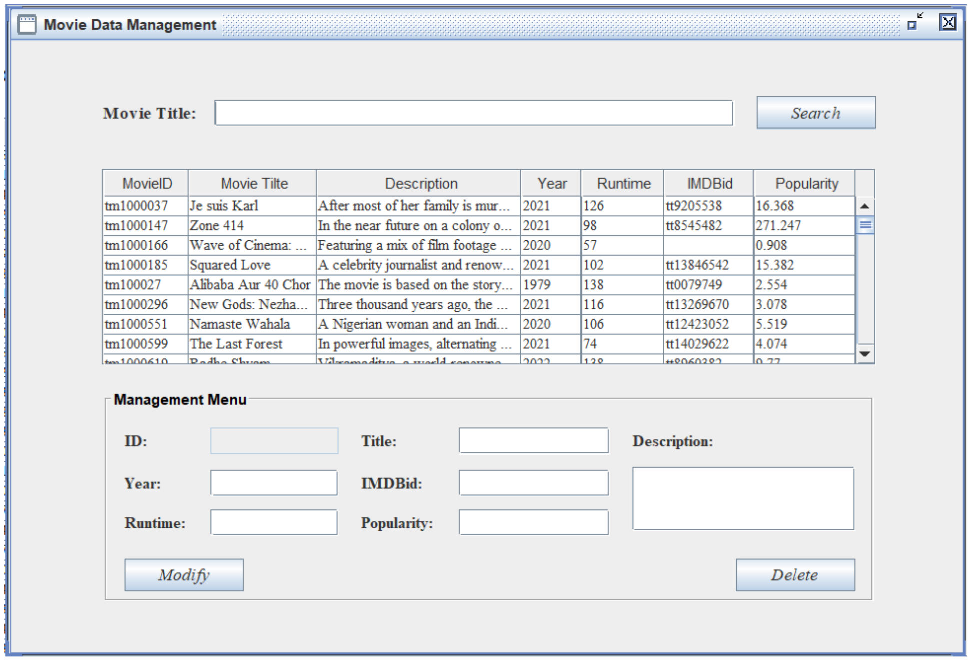
On the top left of the menu bar, we have a menu called Basic Data Manage. We can modify all the data at our database there.



These bars above allow us to add, delete and modify the contents of any of the search tables in our database.



For the add function, users only need to enter partial information and click Add. The program will then pop up a window to indicate whether the user's current action was successful or not.



MovieID	Movie Tilte	Description	Year	Runtime	IMDBid	Popularity
tm1000037	Je suis Karl	After most of her family is mur...	2021	126	tt9205538	16.368
tm1000147	Zone 414	In the near future on a colony o...	2021	98	tt8545482	271.247
tm1000166	Wave of Cinema: ...	Featuring a mix of film footage ...	2020	57		0.908
tm1000185	Squared Love	A celebrity journalist and renow...	2021	102	tt13846542	15.382
tm100027	Alibaba Aur 40 Chor	The movie is based on the story...	1979	138	tt0079749	2.554
tm1000296	New Gods: Nezha...	Three thousand years ago, the ...	2021	116	tt13269670	3.078
tm1000551	Namaste Wahala	A Nigerian woman and an Indi...	2020	106	tt12423052	5.519
tm1000599	The Last Forest	In powerful images, alternating ...	2021	74	tt14029622	4.074
tm1000610	Radha Shyam	Vikramaditya, a world-renowne	2022	138	tt8960382	0.77

For the edit function, the user needs to enter a keyword to search and select an entry. Each entry will then appear in the "Management menu" at the bottom of the window, where the user can modify the information and click the Modify button. If the user wants to delete an item, they need to select it and click on the Delete button. After that, a pop-up window will appear to ask the user to confirm again if they want to delete the data. When the user confirms again, the data will be deleted from the database.

To test our application, we assume scenarios when users use our application and check if the application works properly:

1. add movie/tv show/actor/imdb/relation

Running Time: < 0.1s
After a movie is successfully added, our application will pop up a window to tell the user that the addition was successful, and the user can clearly see that the movie has been added to the database.

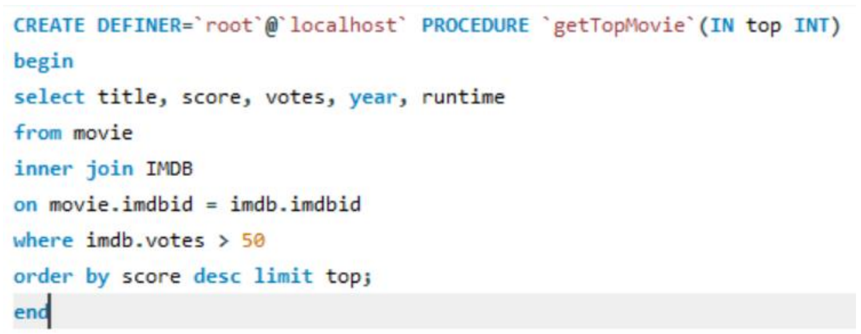
2. Delete movie/tv show/actor/imdb/relations
Running Time: < 0.1s
After successfully deleting a movie, our program will pop up a window to tell the user that the deletion was successful, and the user can clearly see that the movie has disappeared from the database.

3. Update movie/tv show/actor/imdb/relations
Running Time: < 0.1s
After successfully modifying a movie, our program will pop up a window to tell the user that the update was successful. When the user searches for the movie again, he/she will find that the data content has been updated.

4. Search movie/tv show/actor/imdb/relations
Running Time: < 0.1s
Our application provides a search function. The user enters a keyword and then clicks Search. The results will appear in the menu below.

5. Our application can display the top X rated movies (or TV show) with more than 50 votes. Users can choose whether they want to see the ranking of movies or TV shows, and X is an integer determined by the user.

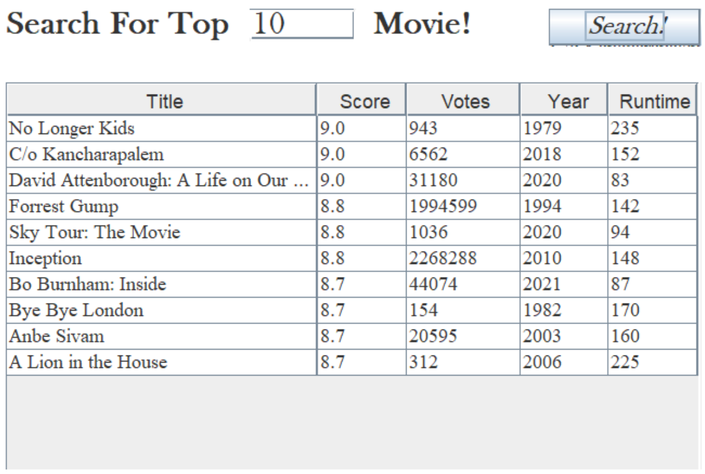
For evaluation purposes, we select the top 10 movies. It has the same process of top 10 TV shows.
Query:
This query is a stored procedure with one integer parameter.



```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `getTopMovie`(IN top INT)
begin
select title, score, votes, year, runtime
from movie
inner join IMDB
on movie.imdbid = imdb.imdbid
where imdb.votes > 50
order by score desc limit top;
end
```

And users can just fill in the spaces with the parameters they want.

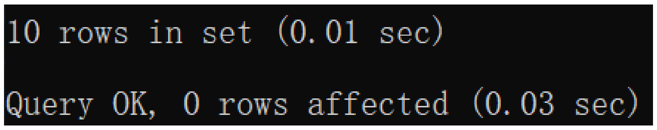
Result in our interface:


Search For Top 10 Movie!

Search!

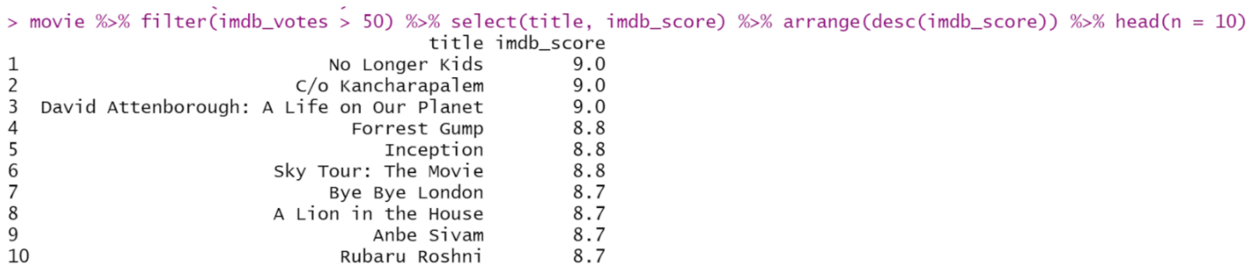
Title	Score	Votes	Year	Runtime
No Longer Kids	9.0	943	1979	235
C/o Kancharapalem	9.0	6562	2018	152
David Attenborough: A Life on Our ...	9.0	31180	2020	83
Forrest Gump	8.8	1994599	1994	142
Sky Tour: The Movie	8.8	1036	2020	94
Inception	8.8	2268288	2010	148
Bo Burnham: Inside	8.7	44074	2021	87
Bye Bye London	8.7	154	1982	170
Anbe Sivam	8.7	20595	2003	160
A Lion in the House	8.7	312	2006	225

Running time in mysql:

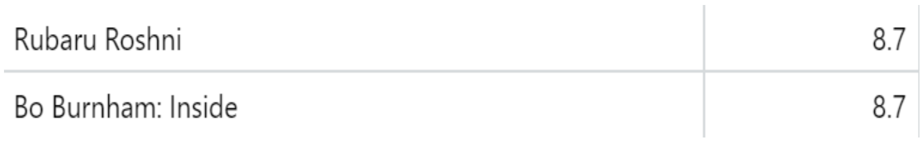

```
10 rows in set (0.01 sec)

Query OK, 0 rows affected (0.03 sec)
```

We can use R to process the raw data to determine if the results are correct.


```
> movie %>% filter(imdb_votes > 50) %>% select(title, imdb_score) %>% arrange(desc(imdb_score)) %>% head(n = 10)
                                   title imdb_score
1                         No Longer Kids        9.0
2                      C/o Kancharapalem        9.0
3  David Attenborough: A Life on Our Planet     9.0
4                           Forrest Gump        8.8
5                              Inception        8.8
6                    Sky Tour: The Movie        8.8
7                         Bye Bye London        8.7
8                    A Lion in the House        8.7
9                             Anbe Sivam        8.7
10                         Rubaru Roshni        8.7
```

We can see that the results after R processing are basically the same as those shown in our application. However, due to the different data alignment, our application shows Bo Burnham: Inside instead of Rubaru Roshni.



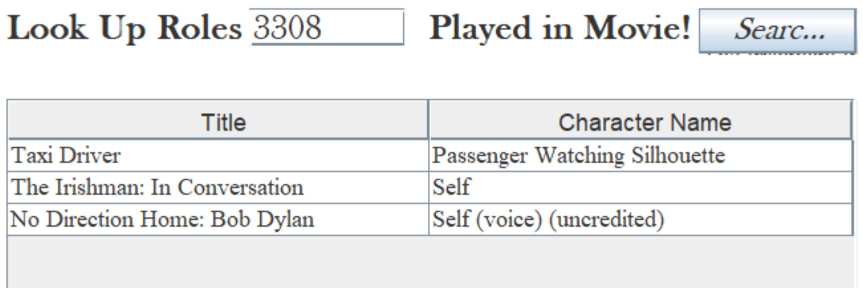
Rubaru Roshni	8.7
Bo Burnham: Inside	8.7

After we checked, both movies had the same score, and we only considered the score as an index, so it led to different results. In conclusion, the result of our application meets our expectations.

6. Our program is able to identify an actor's role in a movie/TV show by his/her personal ID.Users can choose whether they want to search in movies or TV shows.

We test to find out what role actor 3308 has played in the movie.
Query:
This query is a stored procedure with one variable character parameter.

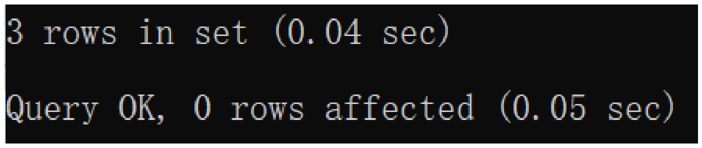
```
CREATE DEFINER=`root`@`localhost` PROCEDURE `getActorCharacterM`(IN actorID varchar(255))
begin
select  m.title, am.cnameM
from actor a
inner join actm am
on a.personid = am.personid
inner join movie m
on am.movieID = m.movieID
where a.personID = actorID ;
end
```
And users can just fill in the spaces with the parameters they want.
Result in our interface:



Look Up Roles 3308 Played in Movie! Searc...

Title	Character Name
Taxi Driver	Passenger Watching Silhouette
The Irishman: In Conversation	Self
No Direction Home: Bob Dylan	Self (voice) (uncredited)

Running time in mysql:



```
3 rows in set (0.04 sec)

Query OK, 0 rows affected (0.05 sec)
```
Use R to process the data to check the result.

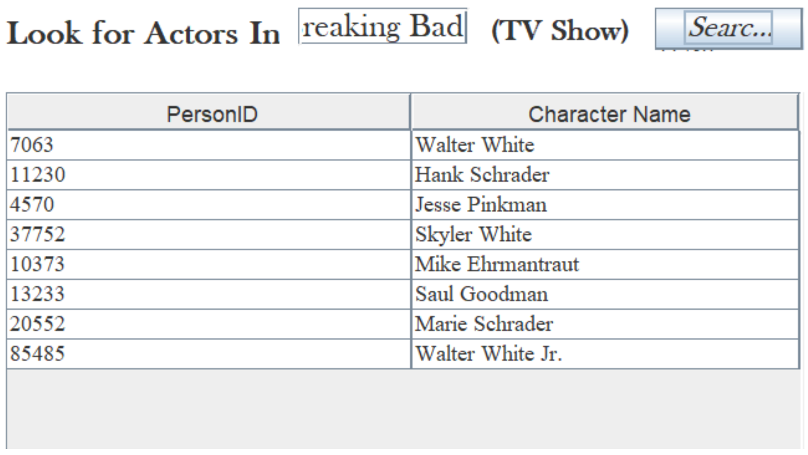
```
> credit %>% filter(person_id == 3308 & role == "ACTOR") %>% inner_join(movie, by = c("id" = "id"))
 %>% select(person_id, character, title)
  person_id                     character                        title
1      3308 Passenger Watching Silhouette                  Taxi Driver
2      3308     Self (voice) (uncredited)  No Direction Home: Bob Dylan
3      3308                          Self The Irishman: In Conversation
```
We can see that the results are exactly the same.

7. Users can search the cast list of a movie/TV show by its name. Users can choose whether they want to search for movies or TV shows.

We test to find the cast list of the TV show Breaking Bad.
This query is a stored procedure with one variable character parameter.

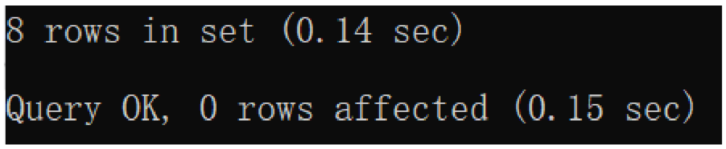
```
CREATE DEFINER=`root`@`localhost` PROCEDURE `getCastT`(IN Showname varchar(255))
BEGIN
select a.personID, at.CnameT
from actor a
inner join actt at
on a.personID = at.personID
where at.showID = (select s.showID from TVshow s where s.title = Showname);
END
```

And users can just fill in the spaces with the parameters they want.
Result in our interface:



PersonID	Character Name
7063	Walter White
11230	Hank Schrader
4570	Jesse Pinkman
37752	Skyler White
10373	Mike Ehrmantraut
13233	Saul Goodman
20552	Marie Schrader
85485	Walter White Jr.

Running time in mysql:



```
8 rows in set (0.14 sec)

Query OK, 0 rows affected (0.15 sec)
```

Use R to process the data to check the result.

```
> show %>% filter(title == "Breaking Bad") %>% select (id, title) %>% left_join(credit, by = c("id"
 = "id")) %>% select(title, person_id, character)
         title person_id        character
1 Breaking Bad      7063     Walter White
2 Breaking Bad      4570    Jesse Pinkman
3 Breaking Bad     37752     Skyler White
4 Breaking Bad     11230    Hank Schrader
5 Breaking Bad     10373 Mike Ehrmantraut
6 Breaking Bad     13233     Saul Goodman
7 Breaking Bad     20552   Marie Schrader
8 Breaking Bad     85485 Walter White Jr.
```

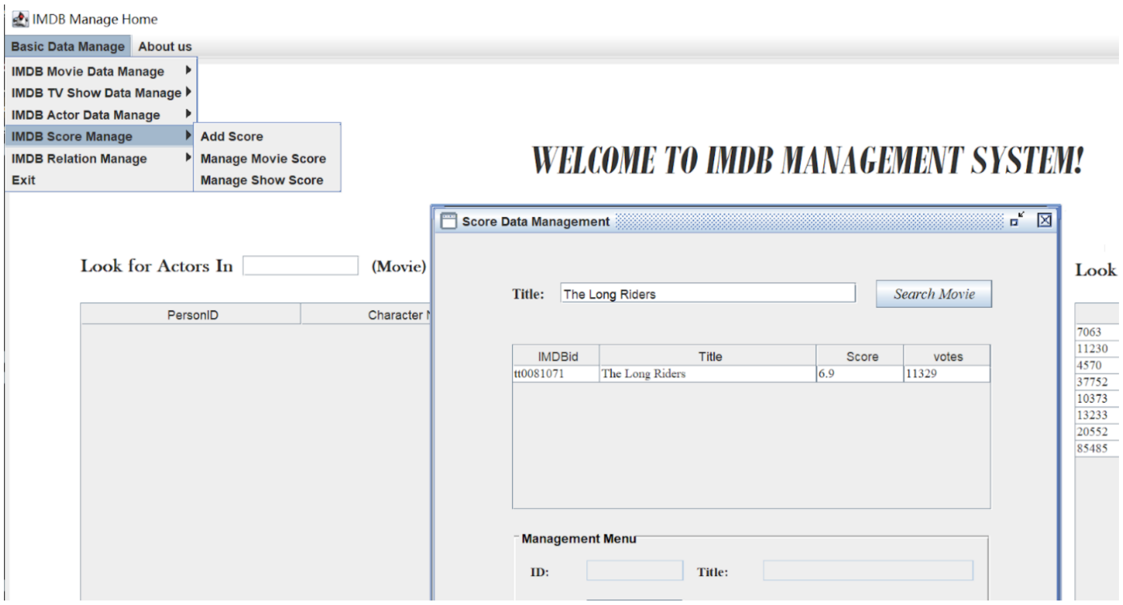
We can see that the results are exactly the same.

8. Our app allows users to find out the score of a movie/TV show by its title.Users can choose whether they want to search for movies or TV shows.
9.

We test to find the score of the movie The Long Riders.

This is a level 2 query since it involves more than one table:
SELECT i.`IMDBid`, m.`title`, i.`Score`, i.`Votes` FROM movie m, imdb i, hasm hm WHERE m.`MovieID` = hm.`MovieID` AND m.`IMDBid` = i.`IMDBid` AND m.`title` LIKE '%The Long Riders%';

Users can simply select Basic Data Management -> IMDB Score Manage -> Manage Movie Score in the upper left corner of the interface. Then fill in the top box with the title the user wants to search for.
Result:



WELCOME TO IMDB MANAGEMENT SYSTEM!

Score Data Management

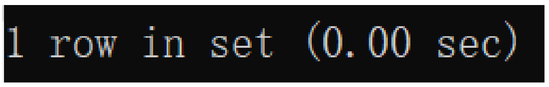
Title: The Long Riders

IMDBid	Title	Score	votes
tt0081071	The Long Riders	6.9	11329

ID: Title:

Look for Actors In (Movie)

Running time in mysql:



```
1 row in set (0.00 sec)
```

Use R to process the data to check the result.

```
movie %>% filter(title == "The Long Riders") %>% select (imdb_id, title, imdb_score, imdb_votes)
  imdb_id           title imdb_score imdb_votes
tt0081071 The Long Riders        6.9      11329
```

We can see that the results are exactly the same.

III. Conclusion

When building the database, we encountered many cases where we needed to modify the table attributes. With the help of DDL, we solved these problems very efficiently. In the process of data processing, we encountered unexpected problems. For example, when we saved the entity sets as csv files, we found that the commas in the movie or TV show descriptions mislead the MySQL data import. Therefore, we saved the entity sets as files separated by "|" symbols. Then, for example, we found that some descriptions had paragraphs, which caused MySQL to read "\n" and start reading the next line of data where it should not be separated. To solve this problem, we use the query and replace function in the word document to replace "\n " with " ". In addition, we learned how to use MySQL's workbench, which is very useful when we write stored procedures.

Our group learned how to connect java to a database, and how to use the swing class of the java language for interface design and implementation. We also learned how to export a Mysql database using Mysqldump, and how to import a Mysql database. We had a good practice with what we had learned before. In addition to the computer knowledge, we also learned to flexibly arrange work according to the distinct schedule of team members, which enhanced our teamwork ability. And gained experience in how the front end, back end, and Data Wrangler are interconnected.