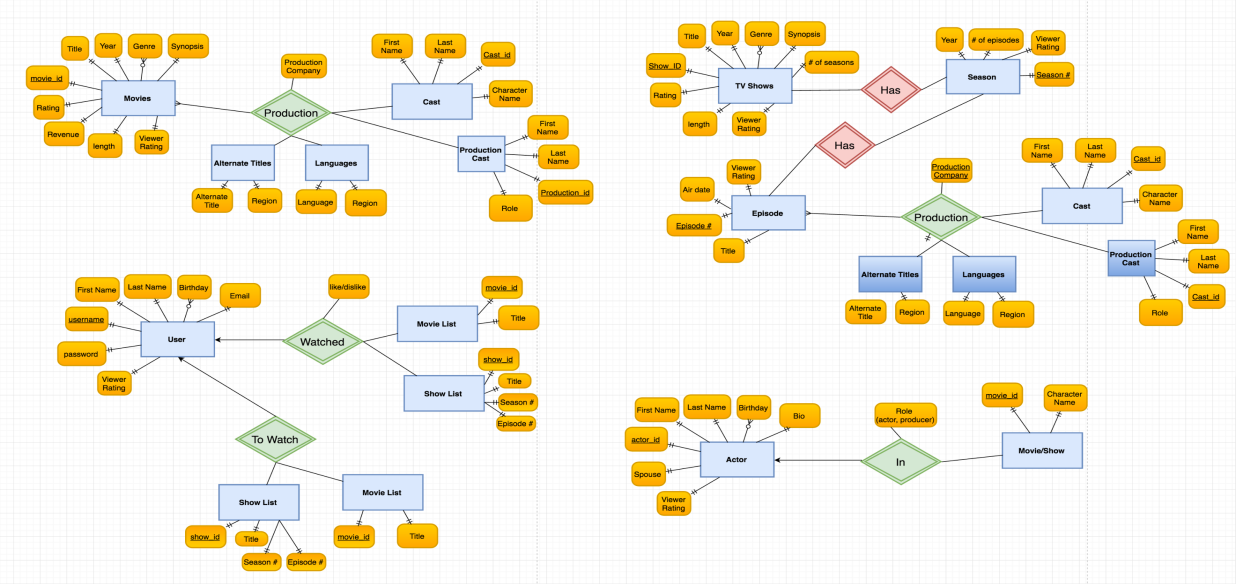
Team 22 - Final Report

Project Idea:

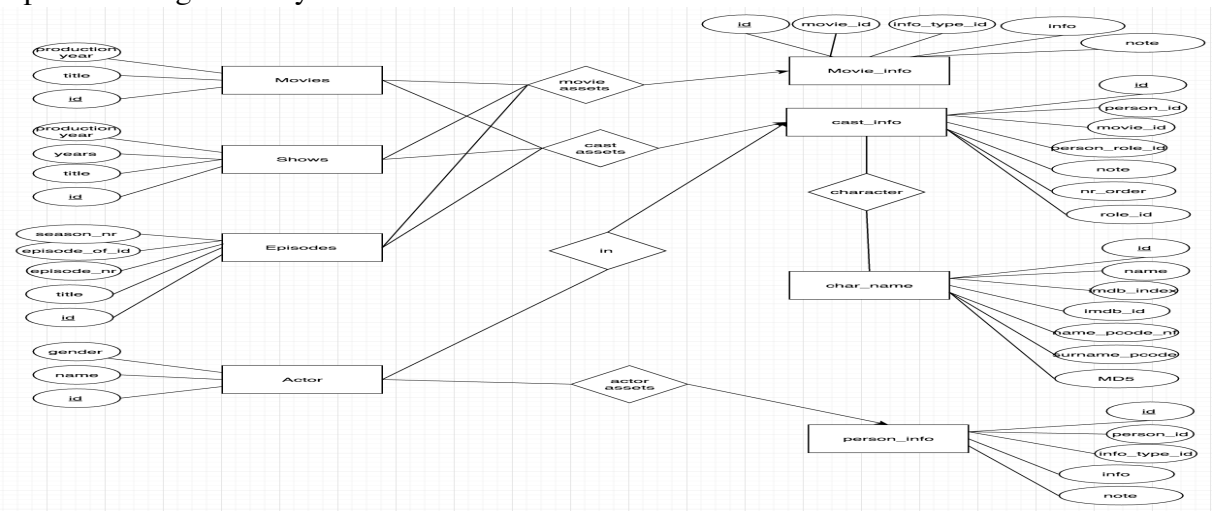
We will be creating a web application using the data provided by IMDB. Our potential target market for our application will be movie lovers and TV Show lovers. Many young adults will be using this application because it allows them to keep track of their favorite shows, movies, and actors along with their friends' favorite movies, shows, and actors as well as information about movies and chart ratings of tv shows. We had to remove the notification system because IMDBPY did not provide all the information we were looking for. IMDB has restrictions on the type of data they put out for the public to use.

ER Design:

Original:



Updated Design & Why:



We had to change our ER Design after we got all the data back from IMDBPY. When creating the original ER Design we thought the IMDB dataset would provide us much more information.

Our new relational model is the following:

Actor(id, name, gender)
MOVIES(title, production_year, id, likes)
TV(title, production_year, id, series_years)
movie_info(id, movie_id, info_type_id, info, note)
movie_info_idx(id, movie_id, info_type_id, info, note)
info_type(id, info)
role_type(id, role)
cast_info(id, person_id, movie_id, person_role_id, note, nr_order, role_id)
char_name(id, name, imdb_index, imdb_id, name_pcode_nf, surname_pcode, MD5)
person_info(id, person_id, info_type_id, info, note)
admin(id,username,passcode)
movies_watched(id,movie_id)
tv_watched(id,tv_id)
user_likes_tracker(user_id, movie_id)
actor_like(id,actor_id)

Data Import:

To import the dataset into our database we had to use IMDBPY. This was the program recommended to use. This program parsed the data and put all the useful information from imdb dataset into tables. From those tables, we used joins and create table to make tables that we needed for our application. We created a total of 16 tables. Our data is clean for the most part because of the parser we used.

MOVIES -Movies(id,title,production_year)
#of attributes: 3
#of rows: 1259796

TV-TV(id,title,years,production_year)
#of attributes: 4
#of rows: 137980

TV_episodes -TV_Episodes(id,title,episode_nr,episode_of_id,season_nr)
#of attributes: 4
#of rows: 2493749

Cast_info-cast_info(id,person_id,movie_id,person_role_id,note,nr_order,role_id)
#of attributes: 7
#of rows: 50154080

Cast_name-char_name(id,name,imdb_index,
imdb_id,name_pcode_nf,surname_pcode,MD5)
#of attributes: 7
#of rows: 4511140

Person_info - person_info(id,person_id,info_type_id,info,note)
#of attributes: 5
#of rows: 3756623

Actor - Actor(id,name,gender)
#of attributes: 3
#of rows: 5752380

movie_info(id, movie_id, info_type_id, info, note)
#of attributes: 5
#of rows: 23308520

movie_info_idx(id, movie_id, info_type_id, info, note)
#of attributes: 5
#of rows: 2038535

info_type(id, info)
#of attributes: 2
#of rows: 113

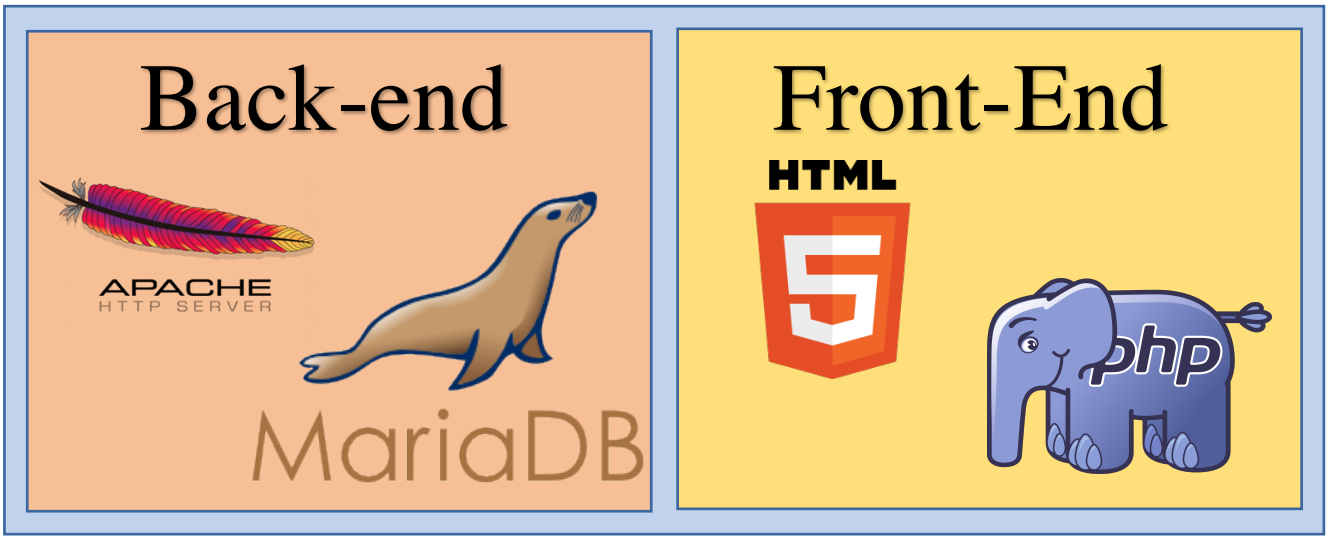
role_type(id, role)
#of attributes: 2
#of rows: 12

admin(id,username,passcode)
#of attributes: 3
#of rows: 3

movies_watched(id,movie_id)
#of attributes: 2
#of rows: 29

tv_watched(id,tv_id)
#of attributes: 2
#of rows: 21
user_likes_tracker(user_id, movie_id)
#of attributes: 2
#of rows: 7
actor_like(id,actor_id)
#of attributes: 2
#of rows: 9

Development Environment:



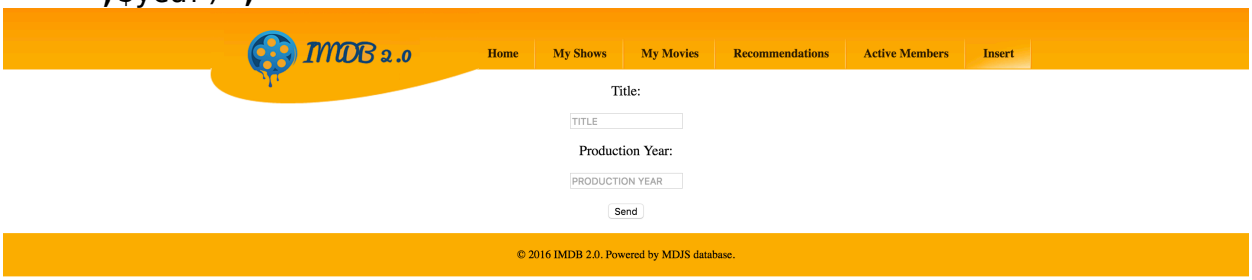
We chose MariaDB as our database server because Engineering IT provided us the VM with it already installed. Setting up everything for the first time was a challenge because this was new to use. Also for our backend webhosting we used the provided Apache webserver. We did not face any issues using both. Every time we saved something, we could immediately see our results on the webpage and had no issues with the webserver. For the frontend we used HTML and PHP. The php portion of the code was to connect to our database and run queries. The HTML was used to format the website and to display everything. We used one external API, Movie-Posters, to display an image related to the movie currently being viewed on the movies page. We chose this API because it served our needs perfectly and we were able to get it to work after a couple hours of tinkering. We also linked to another website in order to generate the ratings graph for a TV series.

Basic Functionalities:

Insert a Movie & TV Show:

This function allows users to add Movies and TV shows as new ones come out as our dataset is not updating from IMDB. This allows our database to be up to date.

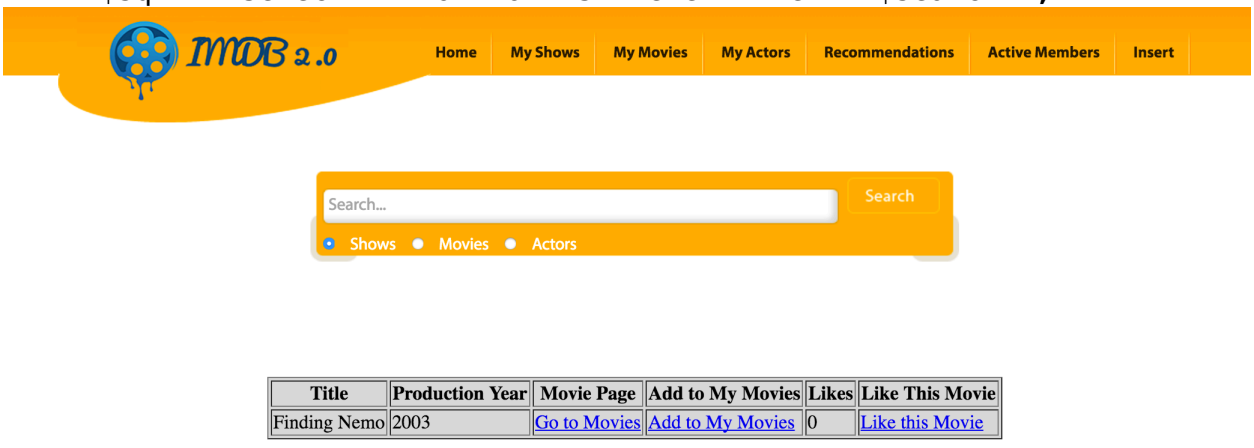
```
$sql = “Insert into MOVIES (title,production_year) VALUE ($title
,$year)”;
```



Search Movie:

Our main purpose is for all users to search our database for movies. This is a basic function but will be used by the users a lot. Also this search will display the picture of the movie poster. There are two parts to searching for a movie. First you search for a movie and it will bring up a list of all movies with that title. Then from there, you can navigate to the specific movie that you want, where it will show a lot of information regarding that movie (more on this last bit in the advanced functionality section).

```
$sql = "Select * FROM MOVIES where title = '$search'";
```

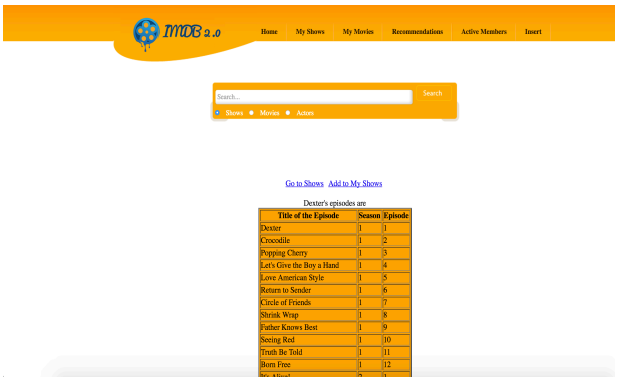


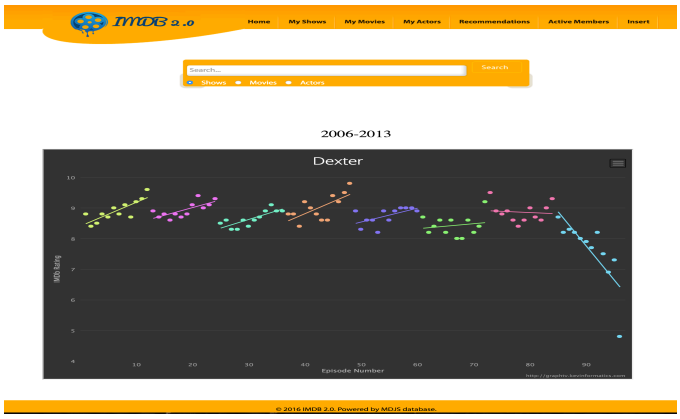
Title	Production Year	Movie Page	Add to My Movies	Likes	Like This Movie
Finding Nemo	2003	Go to Movies	Add to My Movies	0	Like this Movie

Search TV Show & Display its Episode:

This function allows the user to search for a TV series. Their result will be a list of all the episode we have for the given TV Show. This is important because it gives users useful information about the TV series. This also has a link to view the graph displaying the ratings.

```
$sql_rating = "Select series_years from TV where title ='$search'";
$sub = "Select id from TV where title ='$search'";
$sql_graph = "Select title, season_nr, episode_nr FROM
TV_episodes where episode_of_id = ($sub) ORDER BY season_nr,
episode_nr";
```

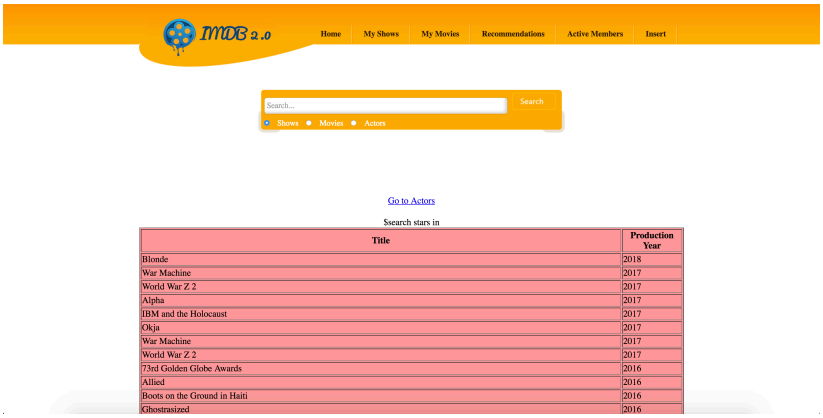




Search Movies by Particular Actor:

For this function users can search their favorite actor and get a table of all the movies the actor/actress has started in.

```
$sql = "select MOVIES.title, MOVIES.production_year from MOVIES inner join cast_info
on MOVIES.id = cast_info.movie_id and person_id = (SELECT id FROM ACTOR WHERE name =
'$actor_name') order by MOVIES.production_year desc";
```



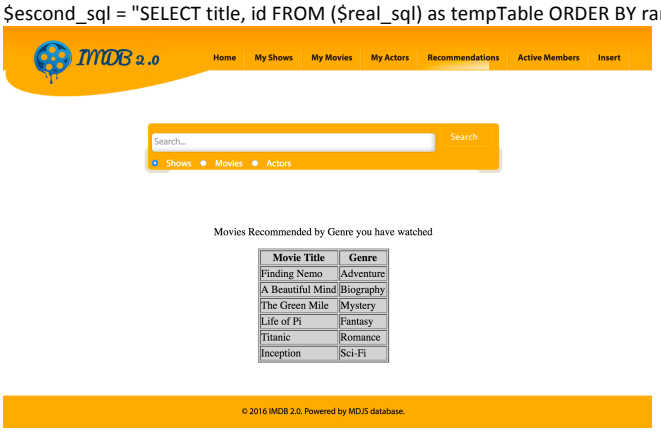
Advanced functionalities:

Our primary goal in creating this application is to make it something that users actually want to use. The main purpose of our application is to allow users to share what they're watching and what actors they enjoy with their friends and to see what and who their friends are watching. To accomplish this, we created a Netflix-Facebook hybrid that easily allows users to view what other users are doing. To that end, it was important that we keep the design simple and straightforward. We didn't want to overcomplicate and distract from the goal on hand, we wanted our application to be as intuitive and easy to use as Google. Finding a perfect balance between tricky and simple is something that has been at the forefront of our development process and we believe we have come up with a design that is both simple to use yet powerful enough to enable our users.

Recommendations List (Advanced Functionality 1):
A user is given a list of recommended movies based on the genres of the movies he has watched and the movies that have been watched by all other users. A sql query first finds the user id for the current logged in user. Then, with that id, a sql query finds a list of all genres that a user has watched and then finds the 50 highest rated movies in every other user's watched list that the user has not watched. Finally the query looks for any movies that have at least 1 like and then randomly selects a movie from that list. At the end, the user will be given a list of recommendations that other users have found to be good. If a user has not watched any movies, then genres will be chosen at random from other user's watched list.

```
$sql_genre= "SELECT DISTINCT info FROM movie_info, movies_watched WHERE info_type_id=3 and movie_info.movie_id =
movies_watched.movie_id and movies_watched.id = $usr_id";

$main_sql =  "SELECT DISTINCT title, MOVIES.id, movie_info_idx.info as rating
              FROM movie_info, movies_watched, movie_info_idx, MOVIES
               WHERE movie_info.info_type_id=3 and _info.movie_id = movies_watched.movie_id and
                movie_info.info = $mike and  movie_info_idx.info_type_id=101 and movie_info_idx.movie_id = movies_watched.movie_id and
                movies_watched.movie_id NOT IN (SELECT DISTINCT movies_watched.movie_id FROM movie_info, movies_watched WHERE
            info_type_id=3 and movie_info.movie_id = movies_watched.movie_id and movies_watched.id = $usr_id)
            and MOVIES.id = movie_info.movie_id
             and MOVIES.likes >= 1
             ORDER BY rating DESC
             LIMIT 50";
$escond_sql = "SELECT title, id FROM ($real_sql) as tempTable ORDER BY rand() LIMIT 1";
```



Personalized Watch Lists and Specialized Movie / TV View Pages (Advanced Functionality 2):

We designed a unique experience for each individual user. First, each user can add movies and / or tv shows to their own lists of watched movies or watched TV shows respectively. In addition, each user can view the watch lists of all other registered users on the web site. On a website like Netflix you cannot view what others have watched or view graphs charting the ratings.

My Movies:

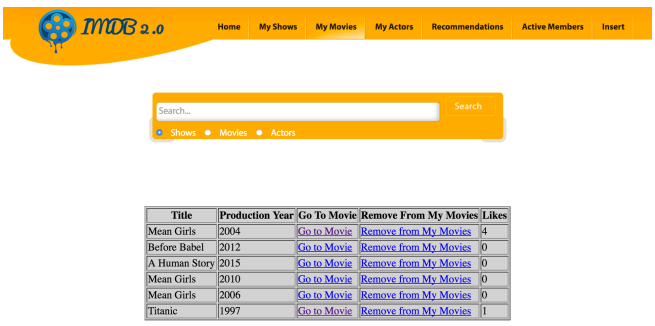
```
$sql1 = "Select * from MOVIES where id IN (select movie_id from movies_watched
where id = (select id from admin where username ='$u'))";
```

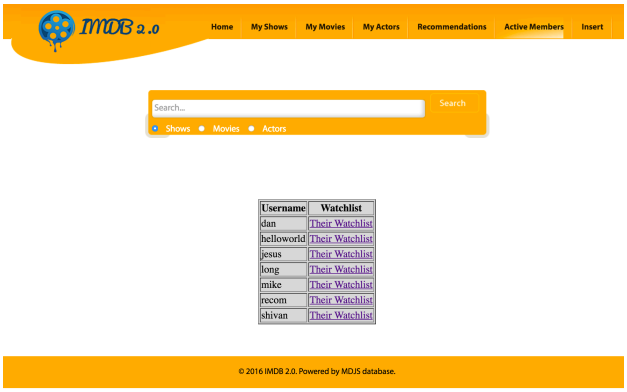
My Shows:

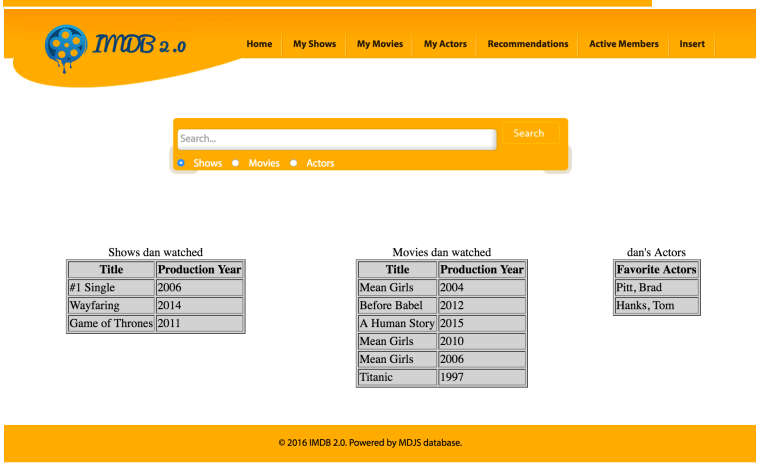
```
$sql1 = "Select * from TV where id IN (select tv_id from tv_watched where id =
(select id from admin where username ='$u'))";
```

My Actors:

```
$sql1 = "Select distinct name from ACTOR where id IN (select actor_id from
actor_like where id = (select id from admin where username ='$u'))";
```







When any user clicks on a particular movie in a list, this brings up a new web page that displays the name of the movie, a movie poster image, and information for that movie. This movie poster image is fetched using a special API. We had to modify the API in order to have it work seamlessly with our site and not require extra input from the user. Additionally, the movie page contains the movie's ratings, genre(s), complete cast list, and a plot summary.

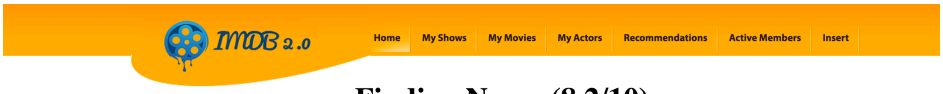
To interface with the API we automatically submit an input field on page load. $data is the id of the movie that is passed in when the page loads.

```
$sql = "SELECT title FROM MOVIES WHERE id=$data”
echo '<input type="test" value="'.$name . '" id="term" />';
```

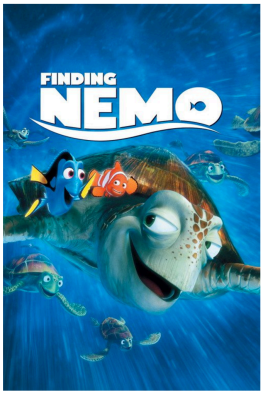
To generate the genres, cast table, and plot:

```
$sql_genre ="SELECT DISTINCT info FROM movie_info, MOVIES WHERE MOVIES.id =
movie_info.movie_id and info_type_id =3 and MOVIES.title = (SELECT title FROM
MOVIES WHERE id = $data)";
$genre = $row1["info"];

$sql_other = "select DISTINCT MOVIES.id, cast_info.person_id,
cast_info.person_role_id, cast_info.role_id, ACTOR.name FROM ACTOR, cast_info,
MOVIES where MOVIES.id = cast_info.movie_id AND cast_info.person_id = ACTOR.id AND
cast_info.movie_id = $data";
$actor = $row1["name"];
$sql_char_name = "SELECT name from char_name WHERE id = $person_role_id";
$char_name = $row_name["name"];
$sql_role_name = "SELECT role from role_type WHERE id = $role_id";
$role_name = $row_role["role"];
$sql_plot = "SELECT info FROM movie_info WHERE info_type_id=98 and
movie_id=$data";
$plot = $row["info"];
```



Finding Nemo (8.2/10)



Genres:

Adventure, Animation, Comedy, Family

Cast:

Actor	Character	Role
Bana, Eric	Anchor	actor
Bird, Nicholas	Squirt	actor
Brooks, Albert	Marlin	actor
Dafoe, Willem	Gill	actor
Garrett, Brad	Bloat	actor
Gould, Alexander	Nemo	actor
Humphries, Barry	Bruce	actor
Hunter, Bill	Dentist	actor
Pendleton, Austin	Gurgle	actor
Peterson, Bob	Mr. Ray	actor
Ranft, Joe	Jacques	actor
Ranft, Jordan	Tad	actor
Ratzenberger, John	Fish School	actor
Root, Stephen	Bubbles	actor
Rush, Geoffrey	Nigel	actor

Plot:

A clown fish named Marlin lives in the Great Barrier Reef loses his son, Nemo. After he ventures into the open sea, despite his father's constant warnings about many of the ocean's dangers. Nemo is abducted by a boat and netted up and sent to a dentist's office in Sydney. So, while Marlin ventures off to try to retrieve Nemo, Marlin meets a fish named Dory, a blue tang suffering from short-term memory loss. The companions travel a great distance, encountering various dangerous sea creatures such as sharks, anglerfish and jellyfish, in order to rescue Nemo from the dentist's office, which is situated by Sydney Harbor. While the two are doing this, Nemo and the other sea animals in the dentist's fish tank plot a way to return to Sydney Harbor to live their lives free again.

When any user clicks on a particular TV show in a list, it brings up a new web page that displays the name of that show along with the series years and a graph displaying IMDB viewer ratings for that show by episode for the length of the entire series. This graph is fetched using another special API.

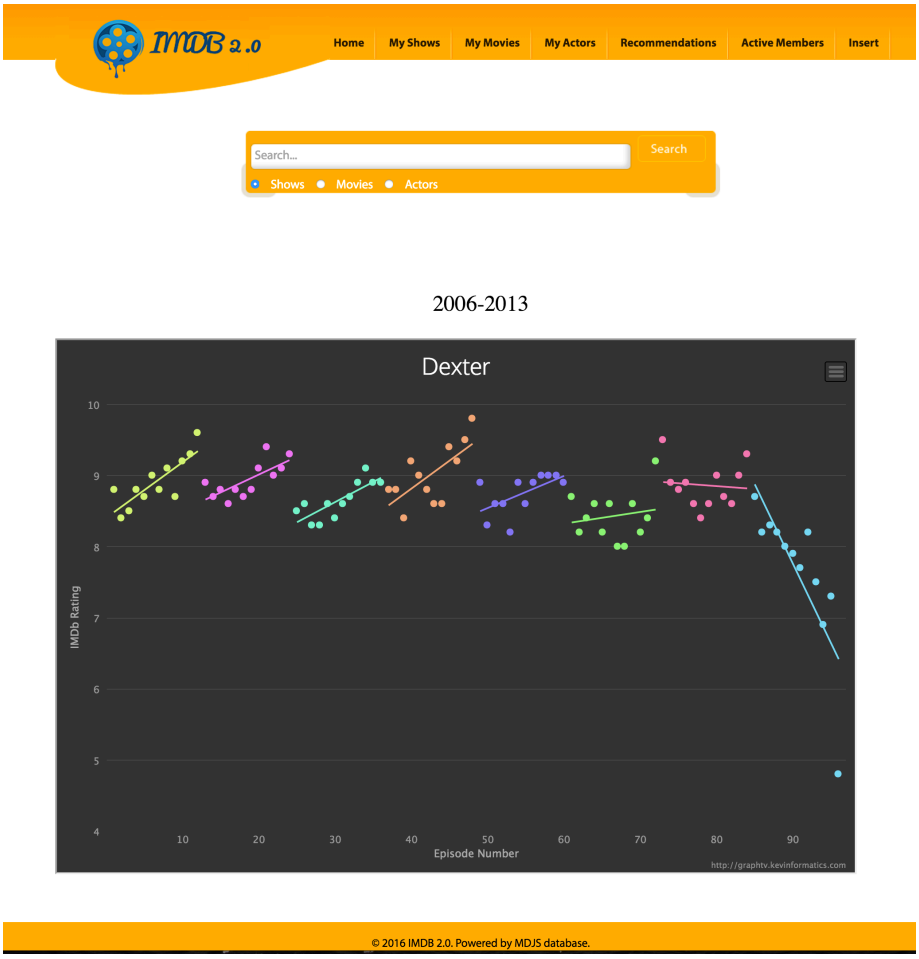
To generate the series years:

```
$sql = "SELECT series_years from TV where title=$title";

echo "<p align='center' style='font-size: 25px;'>$years</p>";
```

To interface with the graphing site:

```
$new = str_replace(' ', '%20', $data);
echo '<iframe src=http://graphtv.kevinformatics.com/?q="' .$new .' height=600px
width=900px scrolling="no"></iframe>';
```



Future work

Make our own parser to clean up the data parsed by IMDBPY. We would either create a parser to clean up and parse the data for initial import or we would use IMDBPY to import the data and then clean up the data once it's already been imported. This is important because the data from the IMDB dataset is filled with a lot of erroneous characters, which makes matching with user searches very difficult.

In addition to the above, we would add a parser to the search bar to help find content that is close to what the user entered. That way a user doesn't have to enter the content perfectly in the search bar (this would allow for a slight bit of user error, which would greatly improve usability).

Lastly, we implement some security measures. We would create a secure login page using encrypted passwords rather than storing everything in plain text. We would also implement SQL injection protection and XSS protection.