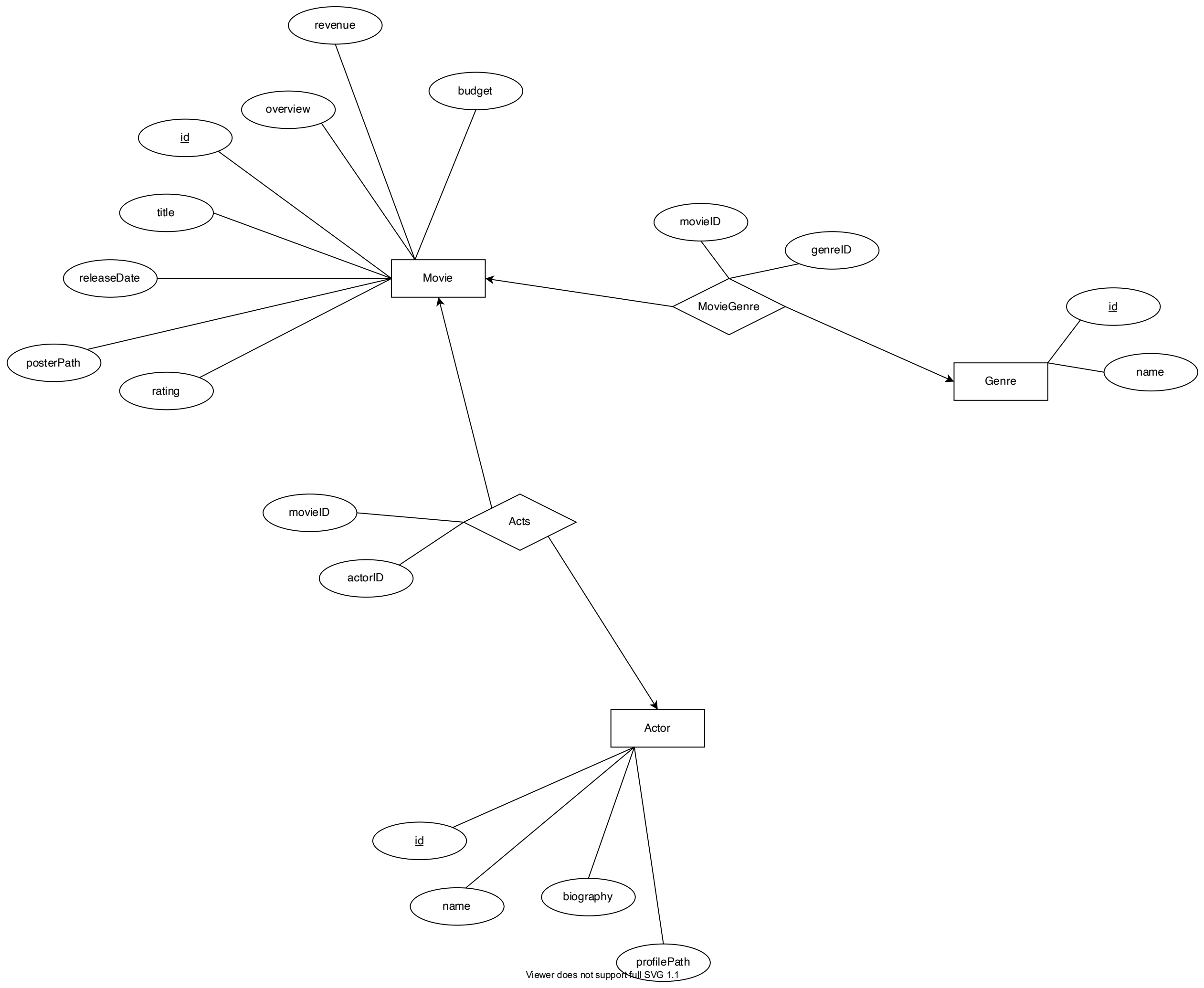
Software Documentation

DB structure:



There are 5 main tables in our DB schema:

1. Movie
2. Actor
3. Genre
4. MovieActor
5. MovieGenre

Movie, Actor and Genre are the three main 'classes' of data we work with.

Since each movie has many genres, and each genre has many movies in it, Movie and Genre should have a many to many connection. In a similar way, each actor plays in several movies, and each movie contains several actors. Therefor, here as well, there should be a many to many connection.

Those connections were done in our schema by the tables:

1. MovieActor
2. MovieGenre

The choice to create tables as connections was made to prevent duplications of data as we learned in class.

Main Quries:

1. Full text query for search:
```
query = ''' select  {}
            from    {}
            where   match({}) against('{}' in boolean mode);
'''.format(titles[table], table, titles[table], t)
```
This query is used for searching in specified table (one of: movie or actors) for a text(t) that we get from a user. It uses indexes that are defined for requested columns, allowing us to perform full text search.

2. Movies with biggest revenue:
```
query = ''' select      Movie.title, Movie.posterPath, (Movie.revenue -
Movie.budget) as pureRevenue,

                        Movie.overview, Movie.id

            from        Movie, MovieGenre, Genre

            where       Movie.id = MovieGenre.movieID and

                        Genre.id = MovieGenre.genreID

                        and Genre.name = '{}'

                        and Movie.revenue > 0

                        and Movie.budget > 0

            order by    pureRevenue desc

            limit       10;

        '''.format(genre)
```
This query is used for getting 10 movies with the biggest pure revenue. (Pure revenue is defined as revenue of movie minus budget of movie). Query joins 3 tables: Movie, MovieGenre and Genre. Searches for the movies with specified genre and returns top 10(their title, path to poster and pure revenue) films with the biggest pure revenue. Filter out movies without budget or revenue. For sufficiency first of all we choose movies from specified genre and only after that checking if movie has revenue and budget.

3. Most popular actors in specific genre:
```
query = ''' select      Actor.name, Actor.profilePath, Actor.biography,
count(*) AS amount, Actor.id

            from        Actor, Movie, MovieActor, MovieGenre, Genre

            where       Actor.id = MovieActor.actorID and

                        Movie.id = MovieActor.movieID and

                        MovieGenre.genreID = Genre.id and
                        MovieGenre.movieID = Movie.id and Genre.name =
'{}'
            group by    Actor.id
            order by    amount desc
            limit       10;
        '''.format(genre)
```

This query is used for getting 10 actors that played in the biggest amount of films in specified genre. It joins 5 tables: Actor, Movie, MovieActor, MovieGenre and Genre. And for each actor sums number of movies (in specified genre) that he played at. Orders results in descending order and returns top 10.

4. Actors of the movie:

```
query = ''' select      Actor.name, Actor.profilePath, Actor.id
            from        Actor, Movie, MovieActor
            where       Actor.id = MovieActor.actorID and
                        Movie.id = MovieActor.movieID and Movie.id =
'{}';
        '''.format(id)
```

This query is used for getting all the actors that played in specific movie. Query joins between 3 tables: Movie, Actor an MovieActor and returns all the actors that played in specified movie (their name, path to profile and id).

5. Movie position in Imdb rating:

```
query = ''' select  (count(*) + 1) as globalRating
            from    Movie
            where   Movie.rating > (    select Movie.rating
                                        from Movie
                                        where Movie.id = '{}' );


        '''.format(id)
```

This query is used for getting position of the movie in DB according to Imdb rating. Using sub query to get rating of the specified movie and then returning count of all movies with rating greater than that.

6. Recommendations for movies:

```
query = ''' select      M1.id, M1.title, M1.posterPath
            from        Movie as M1, Actor, MovieActor
            where       M1.id = MovieActor.movieID and
                        Actor.id = MovieActor.actorID and
                        Actor.id IN (   select A2.id
                                        from Movie as M2, Actor as A2,
MovieActor as MA2
                                        where M2.id = MA2.movieID and
                                        A2.id = MA2.actorID and M2.id =
'{}' and
                                        M1.id <> M2.id  )
            group by    M1.id
            order by    count(*) desc
            limit       5;
        '''.format(id)
```

This query is used for getting recommended movies based on specific movie id. Query joins 3 tables: Movie, Actor, MovieActor. In sub query we are searching for all actors that played in specified movie. Then using this for selecting movies with the same actors. Query returns top 5 movies (their id, title and path to poster) based on amount of common actors with specified movie.

7. Number of movies in genres:

```
query = ''' select      Genre.name, count(*) as totalMovies
            from       Movie, MovieGenre, Genre
            where      Movie.id = MovieGenre.movieID and
                         MovieGenre.genreID = Genre.id
            group by    Genre.id
            order by    totalMovies desc;
        '''
```

This query is used for getting number of movies in specified genre. Query joins 3 tables: Movie, MovieGenre, Genre. For each genre it returns name of genre and number of movies in it.

Data sources:

All the data used in DB originates from two sources – The Movie Database API and csv files containing information from IMDB. First, we filtered data in movies.csv to get only top 10,000 movies based on rating. Then based on these movies we got actors that played in these movies and this is the base of our DB. Based on the results of previous step we retrieved data from API (according to Imdb id's that were in csv files). For example, for each movie in the top movies that were chosen we send request to API to get more details about movie and also, it's relation for other tables, such as genre.

Code Structure:

/SRC:

- globe.py – file containing global variables used throughout the code.
- filter_csv.py – file with methods to filter the csv files we have
- database.py – our Database class from where we connect to the db and run our queries.
- API-DATA-RETRIEVE.py – from here we call the filter_csv methods, to get our movies, actors, and their connection. Then we use our chosen API to run requests and get details for each movie and each actor and insert them to our db. The genres are also fetched from the API.
- /APPLICATION-SOURCE-CODE:
  - Server.py – here we initialize our server, and also have the route handlers for each of the routes we have in our API, meaning from here we receive requests, access the Database class to run queries, and return results to the client.
  - /templates:
    - All the files found here are the html templates which we render to the user.
  - /static:
    - /css:
      - All the css files which style our html templates
    - /data:
      - data.zip – zip files with the 2 csv files we used, ratings.csv and movie_actors.csv. from ratings we took the top 10,000 movies, on which we ran http requests to our chosen API to get details. With movie_actors.csv we connected movies with actors, getting the actors on which we ran http requests to our chosen API.
    - /img:
      - All the static img files we have in our website
    - /js:
      - actor.js – logic for single actor details page
      - autocomplete.js – here we create all the logic which creates the autocomplete dropdown for the input field in the main page.
      - blocks.js – here we have the code generating the blocks page, which is the search results for either movies or actors.
      - genre.js – here we have to logic for generating a page for the top 10 movies and actors of a genre
      - indexUtils.js – here we have the main page logic, which includes the use of wordcloud2 api to generate a word cloud graph from our genres and the amount of movies we have for each one.
      - movie.js – logic for single movie details page

External Libraries:

Client:

All of our client code uses pure javascript code. We only used 1 external library of code (also pure js) which is called wordcloud2.js and is used to generate a word cloud graph with the given API:

- `WordCloud(elements, options);`

`elements – element into which we insert the word cloud graph`

`options – an object with different design values.`

`In our case we used:`

```
list: factorWeights(genres),
 wait: 500,
 gridSize: 40,
 backgroundColor: "transparent",
 color: '#DEF2F1',
 fontWeight: 'bold',
```

Server:

We used the following python modules:

pandas – for dealing with the csv files we have

requests – for sending http requests to the API we chose for retrieving data.

mysql.connector – for connecting to our database

datetime – for converting date strings we got from the api to datetime object to insert to our db.