Software Documentation

Database Scheme Structure

Actors(actorId, actorName, gender)

- actorId – table's primary key
- actorName – the actor's name
- gender – the actor's gender

ActorsMovie(actorId, filmId)

- actorId, filmId – table's primary key

ActorsShow(actorId, showId)

- actorId, showId – table's primary key

Language(languageId, langName, os)

- languageId – table's primary key
- langName – the language's full name
- os – the language's shortened name

LanguageMovie(languageId, filmId)

- languageId, filmId – table's primary key

LanguageShow(languageId, showId)

- languageId, showId – table's primary key

Movie(apiId, title, langId, releaseDay, length, budget, revenue, collection, imdbId, homePage, status, popularity, voteCount, voteAvg, adult)

- apiId – table's primary key
- title – the movie's title
- langId – the movie's primary language
- releaseDay – the date of release
- length – the movie's length in minutes
- budget – the budget spent by the movie in USD
- revenue – the money the movie earned in USD
- collection – the collection the movie to which the movie is assigned
- imdbId – link to IMDB page
- homepage – link to the movie's homepage
- statues – the movie's status (released, rumored, planned, in production, post production)
- popularity – the movie's popularity rating
- voteCount – the number of people who rated the movie
- voteAvg – the movie's average rating
- adult – whether the film is adult only

Software Documentation

Shows(apiId, title, langId, releaseDay, length, homePage, status, popularity, voteCount, voteAvg, seasons, lastEpisodeId, nextEpisodeId)

- apiId – table's primary key
- title – the show's title
- langId – the show's primary language
- releaseDay – the date of release
- length – the length of a single episode in minutes
- homepage – link to the show's homepage
- statues – the show's status (ended, canceled, returning series, in production, planned, pilot)
- popularity – the show's popularity rating
- voteCount – the number of people who rated the show
- voteAvg – the show's average rating
- seasons – the number of the show's seasons
- lastEpisodeId – the id of the show's last episode
- nextEpisodeId – the id of the show's next episode

MovieOverview(filmId, overview)

- filmId – table's primary key
- overview – an overview of the movie

ShowsOverview(showId, overview)

- showId – table's primary key
- overview – an overview of the entire show

Genre(genreId, genreName)

- genreId – table's primary key
- genreName – the name of the genre

MoviesGenre(genreId, filmId)

- genreId, filmId – table's primary key

ShowGenre(genreId, showId)

- genreId, showId– table's primary key

PosterMovie(apiId, image)

- apiId – table's primary key
- image – an image of the movie's poster

PosterShow(apiId, image)

- apiId – table's primary key
- image – an image of the show's poster

Directors(<u>directorId</u>, directorName)

- directorId – table’s primary key
- directorName – the director’s name

Producers(<u>producerId</u>, producerName)

- producerI – table’s primary key
- producerName – the producer’s name

DirectorsMovie(<u>directorId</u>, <u>filmId</u>)

- directorId, filmId – table’s primary key

ProducersShow(<u>producerId</u>, <u>showId</u>)

- producerId, showId– table’s primary key

Tables’ Explanation

the project’s entities are movies, TV shows, actors, directors, producers, genres, and languages. Each of those entities has its own unique table which includes various details, both ones unique to this entity (such as budget for movies) and shared across several entities (such as title for movies and shows).

In addition, in order to connect two entities while avoiding redundancy, we created linked tables with two foreign keys from the entities’ main tables. For example, in order to link movies and genres we created the table MovieGenre.

In some cases, we separated data belonging to the same entity into two tables in order to avoid long loading times. For example, we delegated the movies’ overview field to a table separate from the movies’ general table since it was a large yet rarely used part of the data. Other tables created with this purpose in mind include ShowsOverview, PosterMovie, and PosterShow.

DB optimization

- We used indices to optimize and support quick joins between several tables:
    - In the case of Actors we added an “id” index.
    - In the case of ActorsMovie we added a “filmId” index.
    - In the case of ActorsShow we added a “showId” index.
    - In the case of DirectorsMovie we added a “filmId” index.
    - In the case of ProducersShow we added a “showId” index.
    - In the case of Movie we added “langID”, and “id” indices. We also added a full-text index “title”.
    - In the case of Shows we added “langID”, and “id” indices. We also added a full-text index “title”.
    - In the case of MoviesGenre we added an “apiId” index.
    - In the case of ShowGenre we add an “apiId” index.
- Actors, ActorsMovie, ActorsShow, Movie and Shows have their primary key “id” indexed to support fast joins between tables.
- Movie and Shows tables have their foreign key “langId” indexed as well to facilitate faster joins with the language table.
- DirectorsMovie and ProducersShow have their foreign key, “filmId” and “showId”, accordingly indexed. We did not index “actorId” as well since most searches are according to the indexed keys making this option faster.

Description of Main Queries

Main Queries

1. Similar Movie (Movie Page)
    - Given a movie, the query returns movies which share with it their main language, more than three actors, and more than two genres.
    - In order to optimize this query we did the following:
        - Added “filmId” index to ActorsMovie.
        - Added “apiId” index to MoviesGenre.
        - Added “id” index to Movie and Actors.
        - Added “langId” index to Movie.
    - In order to support this query our database includes:
        - MoviesGenre table, allowing for an easy counting of shared genres.
        - ActorsMovie table, allowing for an easy counting of shared actors.

2. Similar Show (Show Page)
    - Given a show, the query returns show which share with it their main language, at least one actor, and at least one genre.
    - In order to optimize this query we did the following:
        - Added “showId” index to ActorsShow.
        - Added “apiId” index to ShowGenre.
        - Added “id” index to Shows and Actors.
        - Added “langId” index to Shows.
    - In order to support this query our database includes:
        - ShowGenre table, allowing for an easy counting of shared genres.
        - ActorsShow table, allowing for an easy counting of shared actors.

3. Popular Movie (Front Page)
    - The query returns movies released between 1/1/2020 and 1/1/2021 whose number of votes, average rating, popularity, and revenue are above average for movies.
    - In order to optimize this query we added an “id” index to Movie.

4. Popular Show (Front Page)
    - The query returns shows released between 1/1/2020 and 1/1/2021 whose number of votes, average rating, and popularity are above average for shows.
    - In order to optimize this query we added an “id” index to Shows.

5. Recommended Actors by Genre (Actors page)
    - Given a genre’s name the query returns the names of actors who acted in a series or a movie which:

- Got an average rating higher than 8 (voteAvg).
- And, got more votes than is average for their genre (voteCount).

- In order to optimize this query we did the following:
    - We added “apiId” indices in the MoviesGenre and ShowGenre tables.
    - We added “filmId” and “showId” indices in the ActorsMovie and ActorsShow tables.
    - We added “id” indices in the Actors, Movie, and Shows tables.
- In order to support this query our database design includes:
    - ActorsMovie and ActorsShow tables allowing for linking between the actors and the movies/shows following the requirements.
    - MovieGenre and ShowGenre tables allowing for linking between a certain genre and movies/shows in this genre.
    - Genre table, allowing us to easily go over all the genres one-by-one.

6. Actor-Director Winning Combination (Credits page)
    - The query returns an actor-director pair who worked together in more than 7 movies which received a rating average higher than 5.
    - In order to optimize this query we did the following:
        - Added “id” index to Movie.
        - Added “filmId” index to ActorsMovie and DirectorsMovie.
    - In order to support this query our database includes:
        - ActorsMovie table, allowing for easy linking between actors and movies.
        - DirectorsMovie table, allowing for easy linking between directors and movies.
        - Together, those tables allow us to easily find movies following the requirements and connect them to actors and directors. We can then easily count the number of movies for each pair.

7. Actor-Producer Winning Combination (Credits Page)
    - The query returns an actor-producer pair who worked together in more than 7 shows which received a rating average higher than 5.
    - In order to optimize this query we did the following:
        - Added “id” index to Shows.
        - Added “showId” index to ActorsShow and ProducersShow.
    - In order to support this query our database includes:
        - ActorsShow table, allowing for easy linking between actors and shows.
        - ProducersShow table, allowing for easy linking between producers and shows.
        - Together, those tables allow us to easily find shows following the requirements and connect them to actors and producers. We can then easily count the number of shows for each pair.

8. Competing Movies (Movie Page)
    - Given a movie, this query other movies with the same main language, which share with it at least one genre and were released at most six months before or after it.
    - In order to support this query we did the following:
        - Added “apiId” index to MoviesGenre.
        - Added “id” index to Movie.
        - Added “langId” index to Movie.
    - In order to support this query our database includes:
        - MoviesGenre table, allowing for linking between movies and genres and making it easy to find movies with a shared genre.

9. Competing Shows (Show Page)
    - Given a show, this query other shows with the same main language, which share with it at least one genre and were released at most six months before or after it.
    - In order to support this query we did the following:
        - Added “apiId” index to ShowGenre.
        - Added “id” index to Shows.
        - Added “langId” index to Shows.
    - In order to support this query our database includes:
        - ShowGenre table, allowing for linking between shows and genres and making it easy to find shows with a shared genre.

10. Known Actors (Actors page)
    - Given a date range the query returns actors who acted in more than three movies or more than three shows during this date range.
    - In order to optimize this query we did the following:
        - We added “filmId” and “showId” indices in the ActorsMovie and ActorsShow tables.
        - We added “id” indices in the Actors, Movie, and Shows tables.
    - In order to support this query our database design includes:
        - ActorsMovie and ActorsShow tables allowing for linking between the actors and the movies/shows.

11. Successful Actor (Actors page)
    - Given a date range the query returns actors who acted in more than three movies in this date range, such that those movies’ revenue is at least 6.5 times their budget and their popularity is higher than 100, or in more than three TV series in this date range such that their popularity is higher than 100.
    - In order to optimize this query we did the following:

- We added “filmId” and “showId” indices in the ActorsMovie and ActorsShow tables.
- We added “id” indices in the Actors, Movie, and Shows tables.

- In order to support this query our database design includes:
    - ActorsMovie and ActorsShow tables allowing for linking between the actors and the movies/shows.

12. Active Directors (Credits page)
    - The query returns directors and the number of movies they directed, ordered by the number of movies.
    - In order to optimize the query we did the following:
        - Added “filmId” index in DirectorsMovie.
    - In order to support this query our database design includes:
        - DirectorsMovie table, allowing for easy counting of movies directed by each director.

13. Active Producers (Credits Page)
    - The query returns producers and the number of shows they produced, ordered by the number of shows.
    - In order to optimize the query we did the following:
        - Added “showId” index in ProducersShow.
    - In order to support this query our database design includes:
        - ProducersShow table, allowing for easy counting of shows produced by each producer.

14. Show Recommendations by Movie (Front Page)
    - Given the name of a movie the query returns shows that share at least one actor and at least one genre with the movie, as well as the main language.
    - In order to optimize this query we did the following:
        - Added “filmId” and “showId” indices to ActorsMovie and ActorsShow.
        - Added “id” indices to Actors, Movies, and Shows.
        - Added “langId” index to Movie and Shows.
        - Added “apiId” index to ShowGenre and MovieGenre.
        - In order to support this query our database includes:
        - ActorMovie and ActorShow tables, allowing for easy counting of shared actors.
        - MoviesGenre and ShowGenre tables, allowing for easy counting of shared genres.

15. Movie Recommendations by Show (Front Page)
    - Given the name of a show the query returns movies that share at least one actor and at least one genre with the show, as well as the main language.
    - In order to optimize this query we did the following:

    - Added “filmId” and “showId” indices to ActorsMovie and ActorsShow.
    - Added “id” indices to Actors, Movies, and Shows.
    - Added “langId” index to Movie and Shows.
    - Added “apiId” index to ShowGenre and MovieGenre.
- In order to support this query our database includes:
    - ActorMovie and ActorShow tables, allowing for easy counting of shared actors.
    - MoviesGenre and ShowGenre tables, allowing for easy counting of shared genres.

16. Popular Movie by Language (Language Page)

- Given a language, the query returns movies in this language ordered by their popularity.
- In order to optimize this query we did the following:
    - Added “langId” index to Movie.

17. Popular Show by Language (Language Page)

- Given a language, the query returns shows in this language ordered by their popularity.
- In order to optimize this query we did the following:
    - Added “langId” index to Shows.

18. Full Text Search

- Given a word or a phrase the query returns all the shows and movies whose title contains it.
- In order to optimize this query we did the following:
    - Added full-text index “title” to Shows and Movie.

Code structure

The app is divided into different modules which serve different purposes

API Data Retrieve Module
In this module we put everything that is related to the gathering of the data and the database creation itself. We first created the tables, then retrieved all the data using the API, and eventually poured all the data into the tables.

Application Source Code Module
This module handles all the logic behind the data pulling from the server and showing it to the client. This also handles all communication with the client.
Inside this module we have sub-modules:
1. **Client**
Includes “templates”, “static” folders and all html, js and css files for the app’s client. The client gets basic information from server using Flask’s render_template method and then further communicates with the server using HTTP requests such as GET and POST.
2. **Server**
In this sub-module, we have all the functionalities to process the different requests received from the client. We have files corresponding to our entities which include the different queries we have made beforehand. Those files are responsible for receiving the different parameters, processing them and then executing the queries and showing it using the HTTP templates.

Description of your API + how did you use it

We used "The Movie Database API" to get most of our data regarding movies, shows, directors, producers, languages and genres.
As mentioned above, the functionality of gathering the data has its own module.
We fetched the data in the following way:
- First created tables so we can put the data into
- Created Media class which was inherited by shows and movies classes.
- Used discover function which sends a page containing 20 different movies/shows.
We fetched 500 pages in total.
Before fetching the next page, we used the data from the current page to create instances of Movies and Shows classes and then inserted it to the relevant tables; different part of the data fetched was put into different tables. As we explained, we wanted to avoid stacking information in the movies/shows table and avoid redundancy so, for example, posters and overview are saved in different tables. Furthermore, data such as genre and language were saved in different tables as well. After we had the information regarding movies and TV shows, we extract more data related to them such as credits, directors and producers according to the movie/show ID.

We also fetched data from a csv file which is located in SRC/API_DATA_RETRIEVE folder. We made sure we didn't fetched movies we already fetched using the API.

External Libraries

Client – Tabulator, bootstrap
- Tabulator allow us to create interactive tables by automatically parsing jsons retrieved from the server into elegant tables with various UI/UX features we configured (such as linking, sort buttons, etc.)
- Bootstrap is for css
Server - Flask, flask_cors, pymysql, requests, blueprint, random, json

General Flow

When a user enters the website, the Front Page (index) template is displayed.
5 different movies and shows are presented at random.
This data is fetched from the **Popular Movie** and **Popular Show queries**.
The user can view 5 popular movies and shows in the current year (2020).

Furthermore, the user can write in the 2 input boxes a name of a movie/show and the corresponding query, **Similar Movie to Show/Similar Show to Movie** will be invoked and redirect the user to the movie/show page.
This page is also a template Movie/TV-Show which details regarding the movie/show are presented to the user.

In each page the header includes a menu and a search bar:
The search bar gives the user the ability to write a movie's/show's name and view its details in a separate page. If the name was incorrect, we will be redirected to an error page (that has its own template).
That page not only shows the movie's/show's details, it also presents 1 similar movie/show and 1 competing movie/show using the **Competing Movies/Competing Shows** queries if they exists. If not, no further details will be presented.

The menu gives the user the ability to navigate to different pages which different queries are used there.
- **Home**: the Front Page as explained above
- **Search Movies or TV Shows**: in this page a **full text query** is used.
The user can search for movies\shows that contains the word he typed in the input box and they will be presented to him in a table under it.
- **Actors**:
There are 4 tables.

1. Recommended Actor table. Uses **Recommended Actors According to Genre** query that uses client's input.

2. Known Actor table. Uses **Active Actor** query which receives client's input to view the known actors during the time period specified.

3. Successful Actor table. Uses **Successful Actor** query which receives client's input to view the known actors during the time period specified.

4. Winning Combo table. Uses **Winning Combo** query which receives actor name to view the winning combination of actor and directors.

- **Foreign Languages**:
There are 2 tables.

1. Movies table. Uses a simple query which shows movie according to the language given by the client.
2. Shows table. Uses a simple query which shows TV shows according to the language given by the client.

- **Credits**:
There are 4 tables.

1. Directors table. Uses **Active Directors** query that shows all the movies' directors.
2. Producers table. Uses **Active Producers** query that shows all the TV shows' producers.
3. Winning Combo (actors + directors) table. Uses **Winning Combo of Actors and Directors query**.
4. Winning Combo (actors + producers) table. Uses **Winning Combo of Actors and** Producers query.

Credits

We designed a large portion of our templates using nicepage.
Nicepage allows creating almost any modern web design in a simple and convenient way.