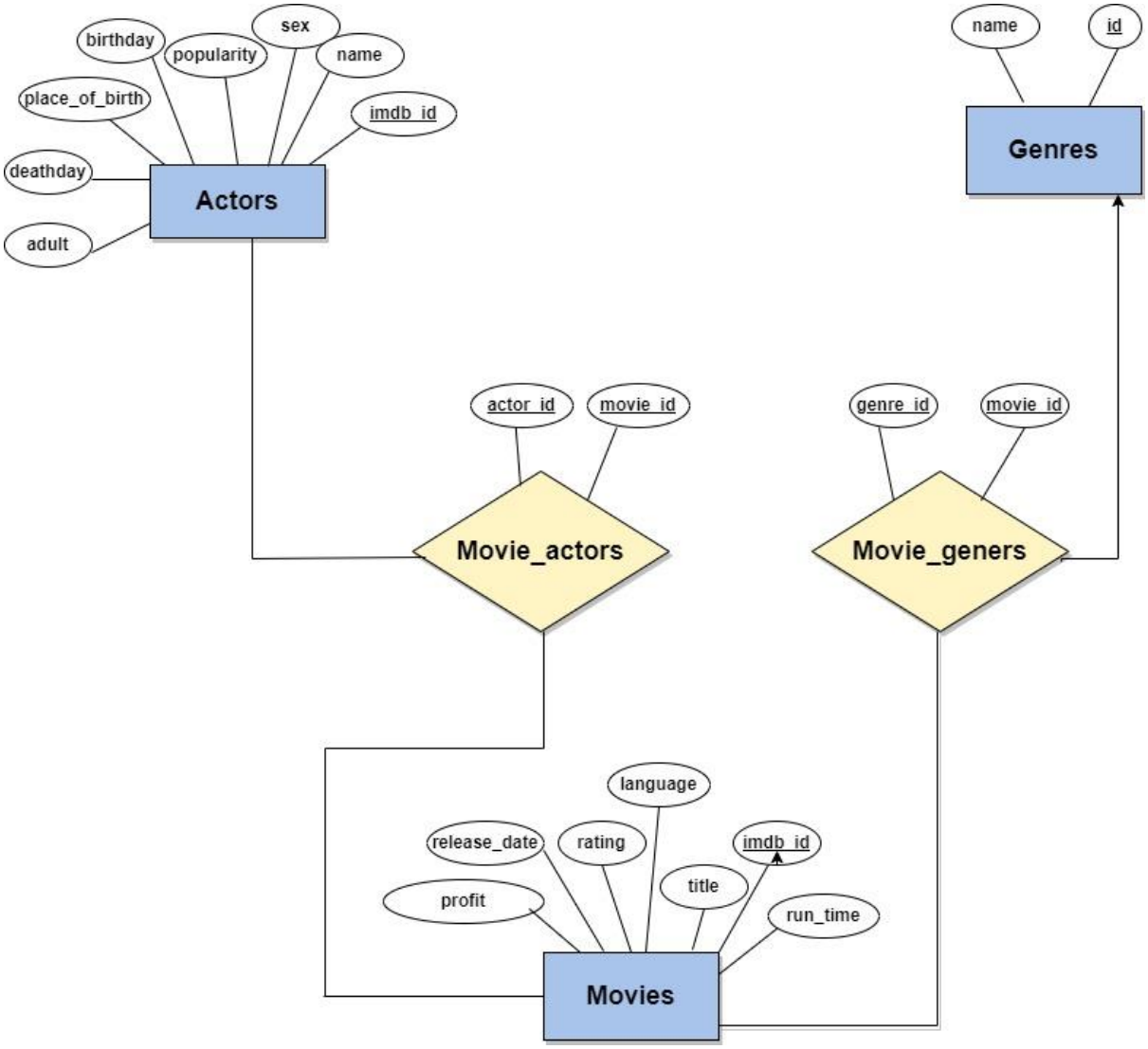
Software documentation

In our database, we implemented the following schema:



We loaded the “Movies” actor as this holds important data for investors- in addition to simple title information, we keep its runtime, its rating, and the movie profit, which are all crucial for an investment decision for given movies.

We needed data on actors as investors would probably like to know which actors bring the most profit or which actors will fit the investors views and values. Because there could be many actors in one movie, and one actor could be in many movies, there is a “Many-to-many” connection between these two tables. Hence, we need a relation to connect between them two, which is “Movie_actors”.

Data on genres could also be highly valuable, as they can teach the user what kind of movies are the most popular. For example, if comedies are the most profitable in several searches, one would want to invest in comedies. In here as well, a movie could have many genres and a genre fits by definition multiple movies. There is a “Many-to-many” connection between these two tables. Hence, we need a relation to connect between them two, which is “Movie_genres”.

We also added a view “Genres_Yearly_Revenues” so that the total profits of a genre in a specific year (or over a range of years). This view saves for every year and every genre the total revenues from this genre in this year.

We used three indices in our project, each is used to ease our queries. For example, searching movie titles in a very big database could be expensive. Thereby we added a full-text index on movie titles.

In addition, movie ratings and movies profits are searched for quite often in our queries, and could also ease the search for queries to be implemented in the future.

In our project, we have 7 different queries:

Please note- all the queries are written in the “utils.py” file

1 – A full-text query. If an investor wants to know how common a word in movie titles is, he/she inserts the word and gets the percentage (for example, in order to check a movie’s originality). We used a full-text index to ease the search of the query, which was simplified by using two separate queries to establish how many movies have this word in their title, and for the total amount of movies. The query works on the movie table

2 – Calculates the average profit for the specified genre. We used an aggregation in order to calculate this value for each genre and used the connections between the genres table and the movies table.

3 – We are also pursuing investors that have a strong ideology! We check in this query if there are more women than any other gender in the database. We did this by using two nested queries, and comparing the total amount of females to the entire case.

4 – This query uses the rating attribute in the Movies table, and its connecting table to “Actors”, to compare the average rating of each. We optimized this query by running two simpler queries, that require less joins each, and comparing their results in python. In addition, this query uses the index on rating.

5 – This query calculates the average run time in the year given as input- for the most profitable genre. This query uses the “Genres_Yearly_Revenues” view, as well as the genres table, the movies table, the connecting table between the movies and genres tables. By joining them all, and comparing each genre’s profit to the rest, we can get the result efficiently.

6 – In this query, we calculate for the year given as input the delta between the total profit of the year given, and the total of the year before that- for each genre. We do that with a nested query on “Genres_Yearly_Revenues” where we do a self-join on the view and join with the Genres table, to get the genres names and for comparing data from the input year and from the year before that.

7 – The users provide a name of a genre, and a threshold- the query looks for actosr that played in several movies from the genre that is greater than threshold- and returns the most profitable one. We do that by joining the five tables of the database, using aggregations and the index on “profit” for movies.

We have two python files that manage the database interface. “Main.py” is the file that interacts with the user and receives requests (the user inserts a number). This file also gets the needed inputs. “Utils.py” is a “server” file that stores and executes all the queries as requested from the main file. The main file prints the output of the queries calculated from utils’ request.

In the project- we used two APIs. One is tmdb (accessed with the python library tmdbv3api) and the other is imdb (accessed with the imdb python library and a Cinemagoer object). We used tmdb for loading movies, actors and genres was done from tmdb, and loading movie rating (a central attribute) was done from imdb.

The user runs the main.py file, inserts his/her requests and receives the query outputs to the shell.