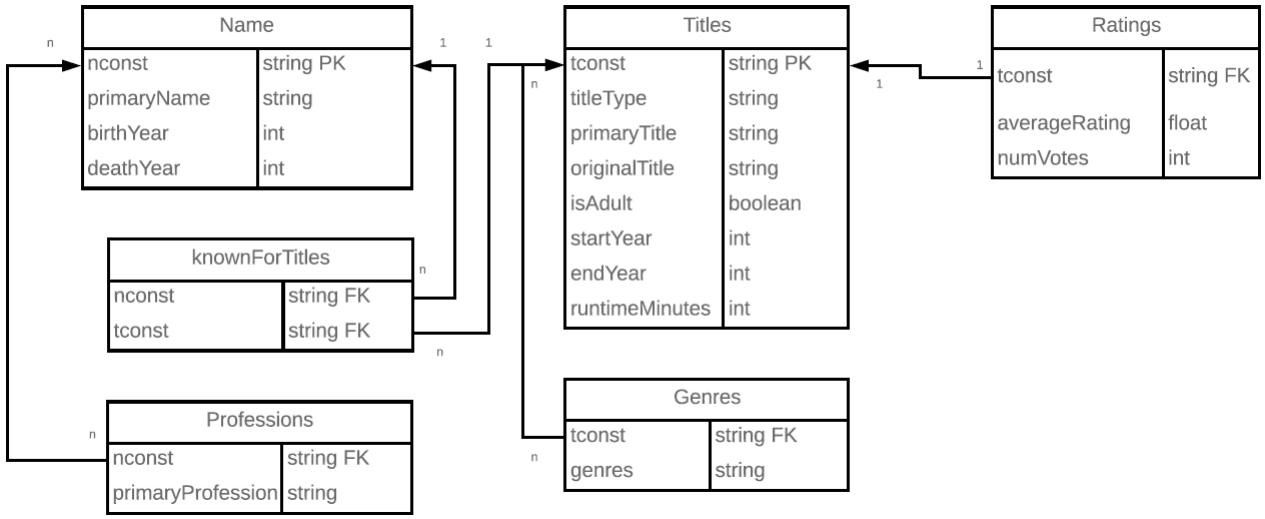
Decisions

I chose to use Python 3.8.1 as language to create the database and populate it.
The API also uses Python, plus Flask.
The database language was SQLite3.
The client application language chosen was Android (Java).

I assumed that the Step 2 of the challenge is referencing only to titles with the titleType = “movie”, since it only asks for movies. And all the ranking is done first by averageRating and second by numVotes.

Database

Analysing the data files, I built the following ER model of the database:



There was the possibility of reducing it to just 1 table (Title), because the challenge only asked for queries that used data from this table, to do so, the genres array could be treated as a string and the ratings data would be part of the titles table (since its 1 to 1 relationship), and in the search queries we could use the command "like" inside the “select” to search for the specific genres, however I decided not to go on this path, and created a more “complete” database.

createDB.py

The database is created and populated using this file. Since the path to the data files is hardcoded, it is necessary that they are in the same folder as the create.DB file, and named dataTitle.tsv, dataRatings.tsv and dataName.tsv.

There is a counterMax variable at the beginning of the code to define how many records will be added to the database at a time. So that it is not necessary to commit each line of the data file, which would make the process much slower.

At the end of the program's execution, a sidia.db file will be generated, which will be the database used by the other program: server.py.

requirements.txt

Contains the requirements necessary to install/run the API, output of a “pip freeze > requirements.txt” execution in the win10 cmd.

server.py

This program corresponds to the API that consumes the database generated in createDB.py. There are 3 methods:

topMovies: returns the result of a query, equivalent to a list of all titles ranked first by highest averageRating and second by numVotes.

topMoviesYear: returns the result of a query, equivalent to a list of the top 10 titles of that year (received as parameter), ranked first by highest averageRating and second by numVotes.

moviesCategory: returns the result of a query, equivalent to a list of all titles in a certain category (received as parameter).

In all cases, films with an average rating less than 6 or adult films will not be returned. And only titles with titleType = "movie" will be returned. (This also could have been treated at the time of the database creation, not inserting adult movies and/or with rating less than 6). Also I’m assuming that category is the same as genre.

The host used is 0.0.0.0 and the port is 5000.

Therefore to access the endpoints in the browser we can use the following example URLs:

http://localhost:5000/movies/Romance (lists all romance movies)

http://localhost:5000/ranking/ (lists all movies ranked by average rating and number of votes)

http://localhost:5000/ranking/2019 (lists top 10 movies of 2019 ranked by average rating and number of votes)

Client Application

The client application is an Android application with only one View. On this view there are 2 editTexts, 2 buttons and 1 recyclerView.

The “Search by category” button will send a request to the API requesting all films registered in the database with the genre equivalent to the text of the first editText (top to bottom). After receiving the result, it will be shown in the recyclerView below. If the bytton is pressed and there is no text in the editText, it will do nothing.

The “Search by year” button will send a request to the API requesting the top 10 films registered in the database with startYear equivalent to the text in the second editText (top to bottom), ordered first by averageRating and then by numVotes. After receiving the result, it will be shown in the recyclerView below. If the button is pressed and there is no text in the editText, it will request all the movies in the database ranked by averageRating and then by numVotes.

To improve

Add pagination. (client application)
Improve user interface. (client application)
Improve search queries because at the moment is somewhat slow. (API)
Sometimes the request gets a timeout or an unexpected end of stream error, still need to find the cause and solve the problem. (client application/API/???)