Kacper Wojdyla, Osama Al-Attia, Filip Sroka, Pedro Alvares Cabral da Camara

Compiling and Running the application

Inside the `FPgroup4` directory run: `stack run`

Using the application

The app, when started, welcomes the user to the app, after which it gives them three options (main menu):

1. Download data
2. Perform database queries

3. Quit

It then awaits input from the user. If the user inputs a different number than “1”, “2”, or “3”, the program will inform the user that it is an invalid input and ask the user for the input again. It will continue to do it until it receives valid input.

If the user inputs “1”, then the program downloads the data from the source. It then saves it to the database while providing the user with printouts informing what is happening in the background. After completion, it goes back to the main menu again and awaits further input from the user.

If the user inputs “2”, a new list of five options is printed out. The first four are the queries the user can do, and the last one, option five, will take the user back to the main menu. If the user chooses one of the queries, the program will ask for further input relating to that query. After, the program will display the results of the query and then display the same list of five options.

If the user inputs “3”, it will print a thank you message to the user and exit the program.

If the user at any point gives invalid input, the program will inform the user about it and print the list of options again.

Web source

The web source is a dataset of movies from 1900 to 2018. The data included the movie's title, year of release, genres and selection of most known cast members (https:// raw.githubusercontent.com/prust/wikipedia-movie-data/master/movies.json).

We had to change the dataset, as we wanted to reduce its size and make it more comfortable for parsing. With the original dataset, it would take ~5 minutes to parse and save all the data. We decided to reduce the dataset from ~55,000 to ~4,500 records (only keeping data from the year 2000 onwards). This massively reduced the waiting time to ~30 seconds. Additionally, we had to manually add a “records” key to the beginning of the dataset so it was easier to parse in Haskell.

The modified dataset is stored on our personal GitHub account and used in our code. (https://champagnekap.github.io/data/movies.json).

Camara **Extracting the information**

The data is obtained using code inside `Fetch.hs` file through an HTTP request to our GitHub repository. That data is then decoded from JSON into a list of records where the

`record` is our defined data type which stores title, year of release, cast members and genres. After which it stores the data in the database, which contains five tables: `actors`, `movies`, `movie_genre`, `cast` and `genres`.

It extracts information from the data by applying the `createRecord` function to each

record. Then from that record, we save the title and the year into appropriate tables. We also save cast and genres, but they are lists; therefore, there are more complex to save. To achieve this, we defined `iterateCast` and `iterateGenre` to iterate through each

recursively and add each value to the appropriate table in the database. For these, we also store two foreign keys inside the `movie_genre` and `cast` tables to allow for a “link”

between the tables.

Extra Features

1. Complex database design

Our database has an optimal design as it does not store duplicate data. It consists of five tables, two of which (`movies` and `genre` & `movie` and `actor`) have a many-to-many

relationship with each other, as one movie can belong to many genres, and each genre can belong to many movies (likewise with actors). This relationship requires additional tables to allow for a “link” between them by storing foreign keys from both tables.

2. Complex queries

Due to our optimal database design to perform three of the four queries, we need to use “JOIN” to make them possible. This is required as the information is spread across multiple tables.