Projeto de Banco de Dados usando IMDb

Giulia Bonaspetti Martins

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil.

Abstract. *This paper presents a project that analyzes and makes more understandable various data about movies and series. For this we will use a graphical web interface where it will be possible to access data such as which are the highest rated and most produced genres, see movies that contain two or more actors together or even years and actors that have produced more "quality" movies (highest IMDb ratings).*

Resumo. *Este artigo apresenta o trabalho que analisa e torna mais compreensível diversos dados sobre filmes. Para isso utilizaremos uma interface gráfica web onde será possível acessar a dados como quais são os gêneros mais bem classificados e os mais produzidos, consultar filmes que contêm dois ou mais atores juntos ou até mesmo anos e atores que produziram mais filmes de "qualidades" (notas mais altas no IMDb).*

1. Introdução

Atualmente, quando queremos ver um filme, nós passamos mais tempo procurando algo para assistir do que de fato vendo o filme. Esse trabalho tem como objetivo facilitar a pesquisa de filmes e o acesso à informações sobre os mesmos.

Para isso irei utilizar as informações encontradas na base de dados do IMDb (*Internet Movie Database*) [1] [2] para criar uma aplicação web onde será possível realizar algumas consultas predefinidas com os parâmetros escolhidos pelo usuário.

Por exemplo, caso uma pessoas esteja com dificuldade de achar filmes e essa pessoa é uma grande fã de Angelina Jolie e do Denzel Washington ela pode usar a aplicação para procurar filmes em que os dois estrelaram, nisso a pessoa achará o filme *O Colecionador de Ossos*.

A aplicação é realizada em JavaScript, HTML e CSS e um banco de dados relacional *SQLite* [3]. O servidor é implementado com a ajuda de *Express.js* [4] e a parte visual da aplicação utilizará a biblioteca React.js [5].

O script de coleta e armazenamento de dados foi feito em *Ruby* [6]. A ferramenta usada para organização pessoal é o *GitHub* [7].

Ao total foram implementadas 11 funcionalidades diferentes e 5 consultas extras para ajudar na composição da interface.

O restante desse artigo irá se apresentar da seguinte maneira:

- Descrição da base de dados: Apresentação do banco de dados e como ele foi construído. Além de alguns dados sobre o mesmo.
- Ambiente de desenvolvimento: Apresentação das tecnologias utilizadas, já faladas nessa introdução, com mais detalhes.

- Metodologia: Apresentação da divisão e organização do trabalho.
- Funcionalidades: Apresentação das funcionalidades com seus respectivos códigos e resultados.
- Conclusão: Apresentação dos principais resultados, aprendizados e dificuldades do projeto.
- Referências

2. Descrição da base de dados

O banco de dados utilizado é relacional em SQLite. As informações e os arquivos de onde elas foram retiradas podem ser encontrados nos bancos de dados IMDb.

O banco de dados IMDb é constituído por 7 arquivos com diversos dados. Usamos um script escrito em Ruby para retirar desses arquivos as informações e criar tabelas mais direcionadas à nossa necessidade.

Explicando um pouco o script e forma como foram montadas as tabelas do nosso banco de dados:

1. Criamos 5 tabelas: genres, movies, classifications, actors e castings, que estão detalhadas mais abaixo
2. Criamos 6 arquivos do tipo .sql: insert_genres.sql, insert_movies.sql, insert_classifications.sql, insert_actors.sql, insert_castings.sql e update_movies.sql
3. No arquivo name.basics.tsv do IMDb nós filtramos somente os atores e atrizes e para cada entrada do .tsv criamos uma entrada em um arquivo insert_actors.sql do tipo `INSERT INTO actors(id,name,birth_year,death_year) VALUES(%d,'%s',%s,%s);` onde id = nconst, name = primaryName, birth_year = birthYear e death_year = deathYear
4. No arquivo title.principals.tsv nós também filtramos somente os atores e atrizes e para cada entrada do .tsv criamos uma entrada em um arquivo insert_castings.sql do tipo `INSERT INTO castings(actor_id,movie_id) VALUES(%d,%d);` onde actor_id=nconst e movie_id=tconst
5. No arquivo title.basics.tsv nós filtramos somente os filmes e para cada entrada do .tsv criamos uma entrada em um arquivo insert_movies.sql do tipo `INSERT INTO movies(id,title,year) VALUES(%d,'%s',%s);` onde id=tconst, title=originalTitle e year=startYear. Após dividimos genres em uma lista onde cada elemento é um gênero para o qual criamos uma entrada em um arquivo insert_genres.sql do tipo `INSERT INTO genres(id,name) VALUES(%s,'%s');` onde id=(número do index do gênero na lista) e name=(o nome do genêro)
6. Ainda arquivo title.basics.tsv para cada entrada filtrada do .tsv criamos uma entrada em um arquivo insert_classifications.sql do tipo `INSERT INTO classifications(genre_id,movie_id) VALUES(%s,%d);` onde genre_id=(número do index do gênero na lista) e movie_id=tconst
7. No arquivo title.ratings.tsv para cada entrada do .tsv criamos uma entrada em um arquivo update_movies.sql do tipo `UPDATE movies SET rating=%s,votes=%s WHERE id=%d;` onde o primeiro parâmetro é averageRating, o segundo numVotes e o último tconst
8. Enfim, executamos cada um dos arquivos sql para construir e atualizar as tabelas

Sendo assim, no fim temos as seguintes tabelas:

- Atores (actors):
  - Id (id): chave-primária, do tipo *INTEGER*, identificador único por pessoa
  - Nome (name): do tipo *TEXT*
  - Ano de nascimento (birth_year): do tipo *INTEGER* no formato AAAA ou *null*
  - Ano de morte (death_year): do tipo *INTEGER* no formato AAAA ou *null*
- Elenco (castings):
  - Id (id): chave-primária, do tipo *INTEGER*, identificador único dupla(autor, filme)
  - Id do ator (actor_id): do tipo *INTEGER* não nula, chave-estrangeira referenciando a chave id da tabela actors
  - Id do filme (movie_id): do tipo *INTEGER* não nula, chave-estrangeira referenciando a chave id da tabela movies
- Filmes (movies):
  - Id (id): chave-primária, do tipo *INTEGER*, identificador único por filme
  - Título (title): do tipo *TEXT*
  - Ano de lançamento (year): do tipo *INTEGER* no formato AAAA
  - Pontuação, a nota IMDb média que o filme recebeu pelos usuários (rating): do tipo *FLOAT*
  - Quantidade de votos (votes): do tipo *INTEGER*
- Gêneros (genres):
  - Id (id): chave-primária, do tipo *INTEGER*
  - Nome (name): o nome do gênero, do tipo *TEXT*
- Classificações (classifications):
  - Id (id): chave-primária, do tipo *INTEGER*, identificador único dupla(gênero, filme)
  - Id do Gênero (genre_id): do tipo *INTEGER*, não nula, chave-estrangeira referenciando a chave id da tabela genres
  - Id do Filme (movie_id): do tipo *INTEGER*, não nula, chave-estrangeira referenciando a chave id da tabela movies

Utilizando o mesmo exemplo citado na introdução, nós teremos na tabela atores (Tabela 1), filmes (Tabela 2), elenco (Tabela 3), gêneros (Tabela 4), e classificações: (Tabela 5).

Tabela 1. Exemplo da tabela de atores

id	name	birth_year	death_year
1401	Angelina Jolie	1975	NULL
243	Denzel Washington	1954	NULL

Tabela 2. Exemplo da tabela de filmes

id	title	Year	rating	votes
145681	The Bone Collector	1999	6.7	156925

Tabela 3. Exemplo da tabela de elenco

id	actor_id	movie_id
581786	1401	145681
581785	243	145681

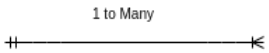
Tabela 4. Exemplo da tabela de gêneros

id	name
3	Drama
8	Crime
16	Mystery

Após aplicar o script, o número de entradas em cada tabela foi:

- Atores (actors): 4014853
- Elenco (castings): 17682799
- Filmes (movies): 574978
- Gêneros (genres): 28
- Classificações (classifications): 775338

Outro dado interessante de ver é o diagrama entidade-relação das tabelas. Considerando PK:Primay Key, FK:Foreing Key,



Nós temos:

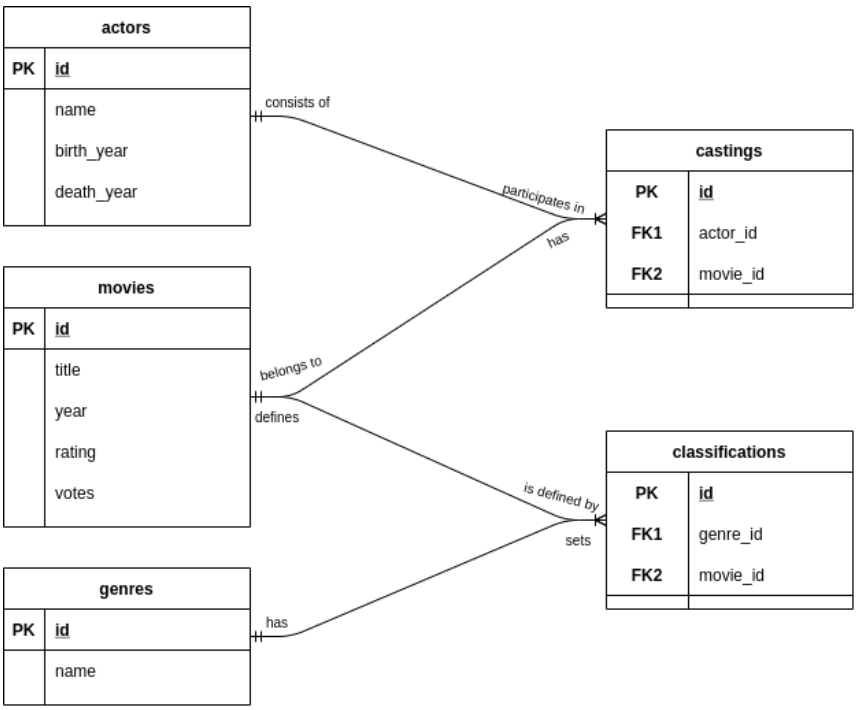


Tabela 5. Exemplo da tabela de classificações

id	movie_id	genre_id
152407	145681	8
152408	145681	3
152409	145681	16

3. Ambiente de desenvolvimento

Como dito anteriormente, o ambiente de desenvolvimento será JavaScript, HTML e CSS para a aplicação, Ruby para o preenchimento do banco de dados e SQLite para o banco em si.

Para a organização do desenvolvimento, eu utilizarei um repositório público no GitHub para o projeto, ou seja, todo o projeto é acessível no repositório *gbonaspetti/pbdd_movies* [8].

Inicialmente a ideia era alimentar manualmente o banco de dados, porém levaria mais tempo para escolher os filmes e criar as entradas do que fazer um script que pegasse automaticamente esses dados. O script de criação e preenchimento da base de dados foi feito em Ruby.

O servidor da aplicação foi feito em JavaScript, utilizando o framework para aplicações web Express.js, assim como a parte visual, também conhecido como *frontend*, que foi feita em JavaScript utilizando a biblioteca React.js.

Uma outra ferramenta muito usada durante o trabalho para facilitar a visualização dos dados foi *DB Browser for SQLite* [9].

4. Metodologia

O trabalho foi dividido em sete etapas principais, das quais as quatro primeiras já foram realizadas:

1. Decisão do tema e busca de informações
2. Decisões técnicas
3. Preenchimento do banco de dados
4. Desenvolvimento das consultas básicas
5. Desenvolvimento das consultas avançadas
6. Desenvolvimento da interface web
7. Melhorias

A decisão do tema foi tomada em dupla, infelizmente o segundo integrante do grupo não pode continuar o trabalho.

Para esta decisão e busca de informações cada integrante fez uma proposta e então discutimos qual dentre elas se encaixava mais na disciplina e agradava aos dois.

Para que pudéssemos tomar essa decisão, cada um procurou ter o máximo de dados possíveis nessa primeira etapa, então foi realizada uma pesquisa de onde encontrar os dados que nos permitiriam fazer o que queríamos.

Depois passamos para a etapa de decisões técnicas. Agora que tínhamos os dados e o que fazer com eles, precisávamos decidir como fazer. Optamos por tecnologias simples e conhecidas para que não fosse necessário realizar nenhuma formação e focarmos somente no objetivo do trabalho.

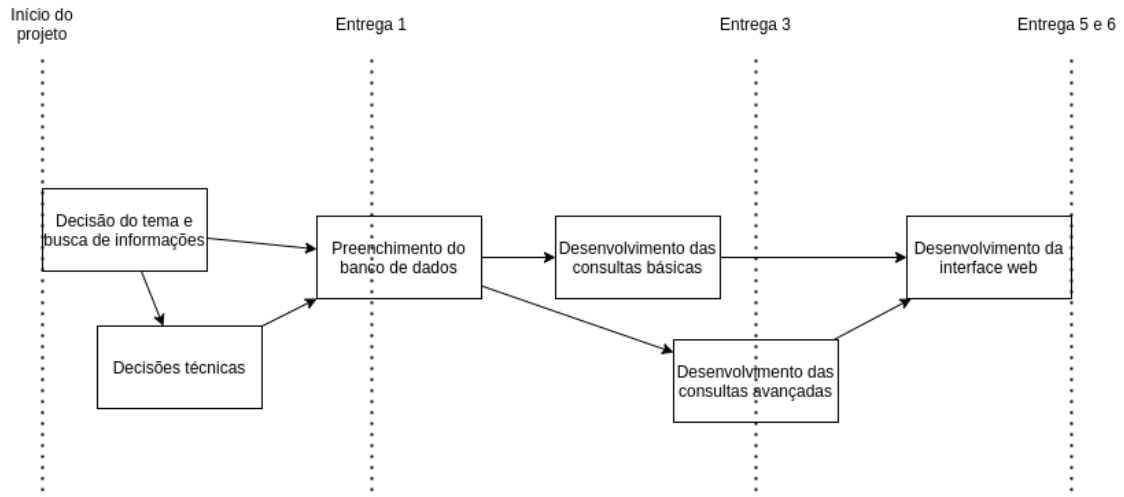
Após a saída do segundo integrante do grupo, algumas das tecnologias foram mudadas para facilitar o trabalho.

O preenchimento do banco de dados foi realizado através de um script escrito em Ruby, para mais detalhes consulte a seção 2: Descrição da base de dados.

O banco de dados é preenchido utilizando diretamente os dados das tabelas do IMDb filtrados e organizados por nós.

Nós decidimos filtras as tabelas, e não utilizar direto as do IMDb para não termos um monte de dados inúteis para as nossas funcionalidades. Dado a grande quantidade de dados, consideramos que isso foi uma boa ideia, porém a divisão das tabelas foi feita de acordo com o que nós achávamos que seria importante somente analisando as funcionalidades, sem implementar, e ao desenvolvê-las vimos que algumas decisões não foram ótimas (como por exemplo ter uma tabela para os gêneros no lugar de ter o nome direto na tabela de classificações).

O fluxograma abaixo serve a facilitar a visualização de quando cada etapa acontece e quais etapas puderam ser realizadas em paralelo, ele foi feito de maneira pessimista, ou seja, a última etapa termina ao fim do prazo de entrega, não havendo tempo para melhorias:



Por fim, conforme o tempo que restará, pretendo fazer algumas melhorias no trabalho. Essas melhorias não estão definidas ainda, podem ser por exemplo melhorias na apresentação de dados ou adicionar novas funcionalidades.

5. Funcionalidades

Assim que definimos nossas funcionalidades nós atribuímos um valor de complexidade de zero a cinco, sendo 0 nem um pouco complexo e 5 muito complexo. Para este relatório as funcionalidades de complexidade 0 a 3 foram consideradas básicas e as funcionalidades de complexidade 4 e 5 consideradas avançadas.

Além disso, algumas consultas extras foram feitas para obter dados que ajudariam a montar a interface.

5.1. Consultas extras

Abaixo estão listadas todas as consultas extras feitas. Junto com cada consulta é apresentado o objetivo da operação, a implementação e o resultado obtido.

1. Todos os anos de lançamento possíveis
    - Objetivo: Obter todos os anos de lançamento de filmes possíveis.
    - Consulta:

```
SELECT year
FROM movies
WHERE year IS NOT NULL
GROUP BY year
```

    - Resultado: O resultado obtido foi uma lista de objetos cuja única propriedade é o ano de lançamento. Abaixo, um exemplo de um objeto da lista.

```
{"year":2020}
```

    - A lista possui 133 entradas.
2. Todos os anos de nascimento possíveis
    - Objetivo: Obter todos os anos de nascimento de atores possíveis.
    - Consulta:

```
SELECT birth_year as year
FROM actors
WHERE birth_year IS NOT NULL
GROUP BY birth_year
```

    - Resultado: O resultado obtido foi uma lista de objetos cuja única propriedade é o ano de nascimento. Abaixo, um exemplo de um objeto da lista.

```
{"year":1993}
```

    - A lista possui 207 entradas.
3. Todos os anos de falecimento possíveis
    - Objetivo: Obter todos os anos de falecimento de atores possíveis.
    - Consulta:

```
SELECT death_year as year
FROM actors
WHERE death_year IS NOT NULL
GROUP BY death_year
```

    - Resultado: O resultado obtido foi uma lista de objetos cuja única propriedade é o ano de falecimento. Abaixo, um exemplo de um objeto da lista.

```
{"year":2012}
```

    - A lista possui 136 entradas.
4. 3000 atores
    - Objetivo: Obter 3000 nomes de atores possíveis (ordenados por id).
    - Consulta:

```
SELECT name
FROM actors
WHERE name IS NOT NULL
GROUP BY id
LIMIT 3000
```

- Resultado: O resultado obtido foi uma lista de objetos cuja única propriedade é o nome dos atores. Abaixo, um exemplo de um objeto da lista.

{"name":"Marilyn Monroe"}

5. Maior número de votos
   - Objetivo: Obter o maior número de votos.
   - Consulta:

```
SELECT MAX(votes) as vote
FROM movies
```

   - Resultado: O resultado obtido foi uma lista com um único objeto cuja única propriedade é o número de votos. Abaixo, um exemplo do objeto da lista.

{"vote":2379513}

5.2. Funcionalidades Básicas

Abaixo estão listadas todas as consultas básicas. Junto com cada consulta é apresentado o objetivo da operação, a implementação e o resultado obtido.

1. Idade dos atores
   - Complexidade: 2
   - Objetivo: Identificar facilmente a idade dos atores.
   - Consulta:

```
SELECT GROUP_CONCAT(name) as names, (2021 - birth_year)
as age
FROM actors
WHERE death_year IS NULL
AND birth_year IS NOT NULL
GROUP BY birth_year
```

   - Resultado: O resultado obtido foi uma lista de objetos cujas propriedades são o conjunto dos nomes e a idade correspondente. Abaixo, um exemplo de um objeto da lista.

{"names":"Gabi The Dog,Tucker Wilkerson,Melody Anita Pop,Kinga Orsolya Eröss,Ailen Cortes Morales,Blue Ivy Carter,Baste Granfon,Elijah J. Angelo,Caroline Jennings,Zoe the Dog,Matilda Wilks,Tian-Lin Chen,Davídek Krízek,Ko Aoki,Sôma Torigoe,Amy Parajuli,Demir Birinci,Sunday Curry,Indiana Arnold,Roger Dale Floyd,JJ Pantano,Elias Richard Siegmann,Godis Noir,Makayla Rose Hilli,Aria Goodson,Anzhelika Eshbaeva,Miroslav Pentsov,Quinn Copeland,Mila Efimova,Angelina Timofeeva,Mark Doronin,Zuzu Ion,Bentley Storteboom,Taisiya Kotova,Nicolás Lucas,Jimmy Gutzeit,Eva Smirnova,Tenta Banka,Yuno Ôta,Otone Maeda,Manon Maindivide,Max Günther","age":9}

   - A lista possui 178 entradas.
     - Adversidade 1 - Tamanho da lista: Visto que no começo havíamos previsto de preencher o banco de dados manualmente, ele não possuiria tantas entradas. Após usar o script a situação mudou e se

tornou bem mais complicado de visualizar os dados devido a quantidade.
Como solução foi decidido de fazer uma interface que permitisse filtrar esses dados segundo o que fosse digitado.
- Adversidade 2 - Falta de dados: O banco de dados IMDb não está 100% completo e algumas vezes faltam dados que pareceriam lógicos serem não-nulos e que são essenciais para a pesquisa. Foi o caso da data de nascimento nessa funcionalidade.
Como solução foi feita uma filtragem na própria consulta SQL.
- Adversidade 3 - Dados errôneos: Outro problema possivelmente ligado ao descrito acima foram alguns dados da nossa consulta que parecem errados porém são apenas erros no banco de dados IMDb dois quais não temos controle.
O exemplo abaixo mostra uma pessoa com 2015 anos. Ao checar a base de dados ela estava coerente com a disponibilizada pelo IMDb. Foi decidido então deixar esses resultados entre os demais e adicionar um aviso ao usuário.

{"names":"Guillaume Pelletier","age":2015}

2. Filmes produzidos em um ano especifico
    - Complexidade: 2
    - Objetivo: Listar a quantidade de filmes produzidas no ano.
    - Consulta: Onde `?` é o valor escolhido pelo usuário

```
SELECT id, title, rating, votes
FROM movies
WHERE year = ?
ORDER BY rating DESC
LIMIT 3000
```

    - Resultado: O resultado obtido foi uma lista de objetos cujas propriedades são o titulo, a note média e a quantidade de votos, ordenada por melhores notas. Abaixo, um exemplo de um objeto da lista passando como parâmetro o ano 2020.


```
{
"id":1070874,
"title":"The Trial of the Chicago 7",
"rating":7.8,
"votes":133056
}
```


    - O número de entradas da lista vai ser relativo ao ano selecionado. No exemplo acima nós temos 14428 entradas para 2020, restringidas a 3000.
        - Adversidade 1 - Tamanho da lista: Assim como dito anteriormente, as listas podem conter muitos resultados. Para facilitar a visualização eles foram ordenados por melhores notas e, para evitar uma falha do navegador, a consulta foi limitada em 3000 respostas.
        - Adversidade 2 - Filmes não avaliados: para filmes não avaliados, ou seja com rating e votes nulos, foi decidido mostrar uma mensagem no lugar dos valores nulos.
3. A quantidade de filmes produzida em cada ano

- Complexidade: 2
- Objetivo: Listar a quantidade de filmes produzidas no ano.
- Consulta:

```
SELECT COUNT(id) as quantity, year
FROM movies
WHERE year IS NOT NULL
GROUP BY year
ORDER BY year DESC
```

- Resultado: O resultado obtido foi uma lista de objetos cujas propriedades são a quantidade de filmes e o ano correspondente. Abaixo, um exemplo de um objeto da lista.

```
{"quantity":1330,"year":1943}
```

- A lista possui 133 entradas.
  - Observação 1 - Quantidade de filmes: Inicialmente a quantidade de filmes produzidas por ano foi chocante, mas coerente com o banco de dados.
    Após analise, esse numero passa a ser mais aceitável, visto que somente a industria de Bollywood faz em torno de 2000 filmes por ano (segundo a enciclopédia Britannica [10]).
    Além disso, a base de dados IMDb não inclui somente filmes populares, logo os pequenos filmes de empresas independentes também são contados.
  - Observação 2 - Lançamentos: O banco de dados IMDb inclui também os filmes que estão previstos para um ano especifico, sendo assim a lista de resultados possui objetos como o seguinte.

```
{"quantity":3,"year":2026}
```

4. Anos que produziram mais filmes de "qualidades" (notas mais altas no IMDb)
   - Complexidade: 2
   - Objetivo: Listar a média das notas de cada ano.
   - Consulta:

```
SELECT year, AVG(rating) as rating
FROM movies
WHERE rating IS NOT NULL
AND year IS NOT NULL
GROUP BY year
ORDER BY rating DESC
```

   - Resultado: O resultado obtido foi uma lista de objetos cujas propriedades são a média das notas médias dos filmes e o ano correspondente. Abaixo, um exemplo de um objeto da lista.

```
{"year":2014,"rating":6.260972300734885}
```

   - A lista possui 125 entradas.
     - Adversidade - Falta de dados: Assim como anteriormente, algum dados que pareceriam lógicos serem não-nulos e que são essenciais para a pesquisa estavam faltando.
       Como solução foi feita uma filtragem na própria consulta SQL.

5. Expectativa de vida de atores
    - Complexidade: 2
    - Objetivo: Possuir dados suficientes para dois tipos de visualizações, número correspondente à média das idades dos atores já falecidos e um gráfico dessas idades.
    - Consulta 1: Para a média foi feito

```
SELECT AVG(age) as deathAvg
FROM (
SELECT (death_year - birth_year) as age
FROM actors
WHERE death_year IS NOT NULL
AND birth_year IS NOT NULL
)
```

    - Resultado: O resultado obtido foi uma lista com um único objeto cuja a única propriedade é a média das idades dos atores.

```
{"deathAvg":70.52406354071498}
```

    - Consulta 2: Para a quantidade de falecidos com cada idade foi feito

```
SELECT COUNT(id) as actors, (death_year - birth_year) as
age
FROM actors
WHERE death_year IS NOT NULL
AND birth_year IS NOT NULL
GROUP BY age
```

    - Resultado: O resultado obtido foi uma lista de objetos cujas propriedades são o número de atores e a idade correspondente. Abaixo, um exemplo de um objeto da lista.

```
{"actors":208,"age":27}
```

    - A lista possui 117 entradas.
        - Adversidade 1 - Falta de dados: Novamente, filtramos a consulta SQL para alguns dados que estavam faltando.
        - Adversidade 2 - Dados errôneos: Outro problema também encotrado aqui foram alguns dados da nossa consulta que parecem errados porém são apenas erros no banco de dados IMDb dois quais não temos controle.
          O exemplo abaixo mostra uma pessoa que morreu com -1 anos de idade. Ao checar a base de dados essa pessoa seria Grace Golden que como podemos na entrada apresentada pela Tabela 6 ela teria nascido em 1904 e morrido em 1903.
          O erro provavelmente foi um erro de digitação na tabela, pois segundo o Wikipédia [11] ela faleceu em 1993.
          Como o erro era ligado aos dados disponibilizados pelo IMDb, foi decidido então deixar esses resultados entre os demais e adicionar um aviso ao usuário.

```
{"actors":1,"age":-1}
```

6. Quais são os gêneros mais bem classificados
    - Complexidade: 3

Tabela 6. Entrada da tabela referente a Grace Golden

id	name	birth_year	death_year
993447	Grace Golden	1904	1903

- Objetivo: Listar os gêneros em ordem de melhor classificação
- Consulta:

```
SELECT name, AVG(rating) as rating
FROM classifications
INNER JOIN movies ON movies.id = classifications.movie_id
INNER JOIN genres ON genres.id = classifications.genre_id
GROUP BY genre_id
ORDER BY rating DESC
```

- Resultado: O resultado obtido foi uma lista de objetos cujas propriedades são o nome do gênero e a média da notas dos filmes desse gênero. Abaixo, um exemplo de um objeto da lista.

```
{"name":"Comedy", "rating":5.90322271128095}
```

- A lista possui 28 entradas.

7. Atores que produziram mais filmes de "qualidades" (notas mais altas no IMDb)
   - Complexidade: 3
   - Objetivo: Listar os atores em ordem de mais filmes melhor qualificados.
   - Consulta:

```
SELECT *
FROM (
SELECT name, AVG(rating) as rating, COUNT(movie_id) as mo-
vies
FROM actors
INNER JOIN castings ON castings.actor_id = actors.id
INNER JOIN movies ON castings.movie_id = movies.id
WHERE rating IS NOT NULL
GROUP BY actors.id
ORDER BY rating DESC
)
WHERE movies > 10
LIMIT 3000
```

   - Resultado: O resultado obtido foi uma lista de objetos cujas propriedades são o nome do artista, a média da notas dos filmes que ele realizou e a quantidade de filmes correspondente. Abaixo, um exemplo de um objeto da lista.

```
{
"id":658,
"name":"Meryl Streep",
"rating":6.591228070175437,
"movies":57
}
```

- A lista possui 347077 entradas originalmente, restringidas a 3000.
  - Adversidade 1 - Tamanho da lista: Outro caso em que uma filtragem será necessária para auxiliar o usuário. Nesse caso há também uma pequena demora para gerar os dados, para resolver o problema e para evitar uma falha do navegador, a consulta foi limitada em 3000 respostas.
  - Adversidade 2 - Ator de um só filme: A maioria dos atores da lista possuíam pouquíssimos filmes (grande maioria só havia participado de um filme). Para tornar a lista um pouco mais interessante, eu decidi considerar apenas atores que participaram de mais de 10 filmes.

8. Quais atores fizeram mais filmes
   - Complexidade: 3
   - Objetivo: Listar os atores que produziram mais filmes.
   - Consulta:

```
SELECT actors.id as id, name, COUNT(movie_id) as quantity,
GROUP_CONCAT(title) as titles
FROM castings
INNER JOIN movies ON movies.id = castings.movie_id
INNER JOIN actors ON actors.id = castings.actor_id
GROUP BY actor_id
ORDER BY quantity DESC
LIMIT 3000
```

   - Resultado: O resultado obtido foi uma lista de objetos cujas propriedades são o nome do ator, quantidade de filmes que ele participou e os respectivos títulos. Abaixo, um exemplo de um objeto da lista.

```
{
"id":609944,
"name":"Wagner Moura",
"quantity":17,
"titles":"Carandiru,Deus é Brasileiro,O Caminho das Nuvens,Cidade Baixa,Tropa de Elite,Ó Paí, Ó,Saneamento Básico, O Filme,They Killed Sister Dorothy,VIPs,Tropa de Elite 2: O Inimigo Agora é Outro,Exodus Where I Come from Is Disappearing,O Homem do Futuro,A Busca,Serra Pelada,Praia do Futuro,Sweet Vengeance,Sergio"
}
```

   - A lista possui 576431 entradas originalmente, restringidas a 3000.
     - Adversidade - Tamanho da lista: Novamente uma filtragem será necessária para auxiliar o usuário e, para evitar uma falha do navegador, a consulta foi limitada em 3000 respostas.

9. Quais são os gêneros mais produzidos
   - Complexidade: 3
   - Objetivo: Listar os gêneros em ordem de mais produzidos
   - Consulta:

```
SELECT name, COUNT(movie_id) as quantity
FROM classifications
INNER JOIN movies ON movies.id = classifications.movie_id
INNER JOIN genres ON genres.id = classifications.genre_id
GROUP BY genre_id
ORDER BY quantity DESC
```

- Resultado: O resultado obtido foi uma lista de objetos cujas propriedades são o nome do gênero e a quantidade de filmes correspondente. Abaixo, um exemplo de um objeto da lista.

```
{"name":"Drama","quantity":206647}
```

- A lista possui 28 entradas.

5.3. Funcionalidades Avançadas

Abaixo estão listadas todas as consultas avançadas implementadas ou que estavam previstas e não puderem ser feitas. Junto com cada consulta é apresentado o objetivo da operação, a implementação feita/prevista e o resultado obtido/previsto.

1. Consultar por filmes que contêm dois ou mais atores juntos
    - Complexidade: 4
    - Objetivo: Listar os filmes que possuem a combinação de 2 ou mais atores selecionados
    - Consulta: onde `${req.query.actorList}` é a lista dos atores selecionados separados por vírgulas

```
SELECT id, title, actorsNames
FROM (
SELECT movies.id as id, title, COUNT(name) as actorsNum-
ber, GROUP_CONCAT(name) as actorsNames
FROM movies
INNER JOIN castings ON castings.movie_id = movies.id
INNER JOIN actors ON actors.id = castings.actor_id
WHERE name IN (${req.query.actorList})
GROUP BY movies.id
)
WHERE actorsNumber > 1
```

    - Resultado: O resultado obtido foi uma lista de objetos cujas propriedades são o identificador, o título do filme e quais dos atores selecionados participaram do filme. Abaixo, um exemplo de um objeto da lista passando como parâmetro a lista de atores ('Angelina Jolie', 'Brad Pitt', 'Denzel Washington', 'Adam Brody').

```
{
"id":356910,
"title":"Mr.  & Mrs.  Smith",
"actorsNames":"Brad Pitt,Angelina Jolie,Adam Brody"
}
```

    - O número de entradas da lista vai ser relativo ao ano selecionado. No exemplo acima nós temos 3 entradas para os quatro atores.

2. Consulta personalizada (usuário escolhe quais e quantos parâmetros ele quer para filtrar a pesquisa)
    - Complexidade: 4
    - Objetivo: Permitir que o usuário explore o banco de dados fazendo as consultas que lhe convém
    - Consulta: onde para cada fator existente de `req.query` a condição equivalente é concatenada à consulta sql, exceto `req.query.from` que define a tabela que vai ser consultada

```
let conditions = 'WHERE id IS NOT NULL'

if(req.query.birth) conditions = conditions.concat(' AND
birth_year = ${req.query.birth}')
if(req.query.death) conditions = conditions.concat(' AND
death_year = ${req.query.death}')
if(req.query.rate) conditions = conditions.concat(' AND
rating >= ${req.query.rate}')
if(req.query.votes) conditions = conditions.concat(' AND
votes >= ${req.query.votes}')
if(req.query.year) conditions = conditions.concat(' AND
year = ${req.query.year}')

const sql = `SELECT *
FROM ${req.query.from}
${conditions}
LIMIT 3000`
```

    - Resultado: O resultado obtido foi uma lista de objetos cujas propriedades são todos os atributos da tabela escolhida. Abaixo, um exemplo de um objeto da lista passando como parâmetro o tipo atores, o ano de nascimento 1907 e o ano de falecimento 2003.

```
{
"id":31,
"name":"Katharine Hepburn",
"birth_year":1907,
"death_year":2003
}
```

3. Qual é a menor distância (em filmes) entre dois atores, consulta no estilo "teorema do Kevin Bacon" [12]
    - Complexidade: 5
    - Objetivo: Calcular a menor distância entre dois atores dados pelo usuário
    - Esta funcionalidade não foi implementada devido a sua dificuldade. Ao tentar implementar, vi que SQLite3 não possuía laços de iteração[13], então seria muito complicado realizar o algoritmo.
      Contudo, o algoritmo que havia sido planejado era uma busca em largura[14] com a ajuda de uma tabela temporária T, em uma versão pseudo-código a distancia entre os atores X e Y:

(a) Dado o nome dos atores, procura-se seus id na tabela actors
(b) Na tabela castings, seleciona-se a lista dos id dos filmes feitos pelo ator X
(c) Para cada id da lista, procura-se na mesma tabela todos id dos atores que participaram do filme e adiciona na tabela T a dupla (actor_id, distance) onde distance é 1
(d) Após, checa-se se o id de Y esta na tabela, se sim retorna distance, se não executa-se os passos (e) e (f)
(e) Para cada actor_id de T, seleciona-se a lista dos id dos filmes feitos por ele em castings
(f) Para cada id da lista, procura-se na mesma tabela todos id dos atores que participaram do filme e caso ele não esteja ainda em T, adiciona na tabela dupla (actor_id, distance) onde distance é a distance do ator que fez o filme com ele somado de um e repete o passo (d)
(g) Caso o algoritmo terminasse o último filme do último ator e Y ainda não estivesse na lista, ele retornaria distance igual a -1

- Resultado esperado: O resultado esperado seria uma lista com um único objeto cuja a única propriedade é a distancia entre os dois atores.

```
{"distance":  2}
```

4. Consultar filmes similares
   - Complexidade: 5
   - Objetivo: Calcular um índice de similaridade entre filmes e retornar a lista dos filmes com maior similaridade.
   - Esta funcionalidade não foi implementada devido à uma má gestão do tempo e a falta de loopings.
     Contudo, o algoritmo que havia sido planejado era uma ponderação de elementos com a ajuda de uma tabela temporária T, em uma versão pseudo-código dado o filme X:
     (a) Dado o título do filme, procura-se seus dados na tabela movies
     (b) Realizando um inner join com a tabela classifications, seleciona-se somente os filmes que possuem o mesmo gênero e adiciona-se esses filmes na tabela T
     (c) Então adiciona-se uma nova coluna à T chamada similarity, inicialmente com todos valores iguais a zero.
     (d) Após, para cada filme de T, se ele possui o número de votos maior ao número de votos de X, adiciona-se 1 à similarity, se igual adiciona-se 2
     (e) Se ele possui o valor de rating (nota IMDb) maior ao de X, adiciona-se 4 à similarity, se o valor for igual adiciona-se 6
     (f) Caso o ano de lançamento for igual ao ano de lançamento de X adiciona-se 5, se for até 3 anos antes ou depois, adiciona-se 3
     (g) Filtra-se então a tabela para todos os valores de similarity maiores que 0 e ordenados em ordem decrescente de similarity

     Se sobrasse tempo, uma das melhorias previstas era incluir os atores que fizeram o filme no calculo de similaridade. O peso seria somar um para cada ator em comum entre os filmes.

- Resultado esperado: Uma lista de objetos cujas propriedades são todos os atributos da tabela movies mais a similaridade. Abaixo, um exemplo de um objeto da lista passando como parâmetro o filme The Lion King de 2019 (similaridade calculada com base no algoritmo sem melhorias).


```
{
"id":2948372,
"title":"Soul",
"year":2020,
"rating":8.1,
"votes":225143,
"similarity":8
}
```


6. Conclusão

Nesse trabalho foram realizadas consultas e analises do banco de dados IMDb com o objetivo de tornar mais fácil a escolha de filmes para assistir e sanar possíveis curiosidades que o usuário possa ter sobre o meio artístico do cinema.

Ao total foram implementadas 11 funcionalidades diferentes e 5 consultas extras para ajudar na composição da interface.

Dentre tais funcionalidades destacam-se a consulta por filmes que contêm dois ou mais atores juntos e curiosidades como quais são os gêneros mais produzidos.

Acredito que meus maiores aprendizados estão diretamente ligados as minhas maiores dificuldades nesse trabalho, foi a primeira vez que lidei com um banco de dados tão grande e tive que achar soluções criativas de como contornar isso. Tenho certeza de que existem diversas outras otimizações que poderiam ser feitas, como paginação da API [15]. Contudo, devido ao tempo disponível, eu optei por soluções mais simples.

Outra dificuldade que tive foi a falta de laços de iteração na linguagem escolhida, isso me ensinou muito sobre a linguagem, apesar de não ter conseguido contornar o problema dentro do prazo.

A última dificuldade que posso destacar foi o fato de ter planejado o projeto para sr feito em grupo e ter que realizá-lo em grande parte sozinha. Sendo assim, sao identificados inumeras continuaçoes e melhorias para o trabalho, dentre elas:

- atualização frequente do banco de dados (atualmente ele esta sempre na versão que o banco estava na primeira utilização do algoritmo)
- melhoras na interface, em termos de design e de experiencia do usuário
- melhoras de performance e de acesso aos dados
- mais funcionalidades

Referências

1 - Interfaces dos dados do IMDb
Último acesso: abril de 2021
https://www.imdb.com/interfaces/

2 - Conjunto de dados do IMDb
Último acesso: março de 2021
https://datasets.imdbws.com/

3 - Site oficial SQLite
Último acesso: maio de 2021
https://www.sqlite.org/index.html

4 - Site oficial Express.js
Último acesso: abril de 2021
https://expressjs.com/

5 - Site oficial React.js
Último acesso: maio de 2021
https://reactjs.org/

6 - Site oficial Ruby
Último acesso: abril de 2021
http://www.ruby-lang.org/

7 - Homepage GitHub
Último acesso: maio de 2021
https://github.com/

8 - Repositório GitHub do projeto
Último acesso: maio de 2021
https://github.com/gbonaspetti/pbdd_movies

9 - Site oficial DB Browser for SQLite
Ultimo acesso: maio de 2021
https://sqlitebrowser.org/

10 - Artigo sobre Bollywood na enciclopédia Britannica
Ultimo acesso: maio de 2021
https://www.britannica.com/topic/Bollywood-film-industry-India

11 - Página Wikipédia sobre Grace Golden
Ultimo acesso: maio de 2021
https://en.wikipedia.org/wiki/Grace_Golden

12 - Six Degrees of Kevin Bacon
Último acesso: março de 2021
https://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon

13 - SQLite language
Último acesso: maio de 2021
https://www.sqlite.org/lang.html

14 - Página Wikipédia sobre a busca em largura
Último acesso: maio de 2021
https://pt.wikipedia.org/wiki/Busca_em_largura

15 - Artigo sobre paginação API
Último acesso: maio de 2021
https://thiagolima.blog.br/parte-5-pagina\
%C3\%A7\%C3\%A3o-ordena\%C3\%A7\%C3\
%A3o-e-filtros-em-apis-restful-3045d88b4114