Universidade Federal de Itajubá (UNIFEI)

The Movie Database
TMDB API

Grupo:

Adriano Lucas Ferreira
Antonio Gomes Xavier de Moura
Bruno José M. Vercelli
João Marcos Cucovia

Disciplina: Banco de Dados II (COM 231)
Professora: Dra Vanessa Cristina Oliveira de Souza

1. Introdução

A API escolhida foi a The Movie Database (TMDB), que disponibiliza informações sobre filmes, séries, programas de TV e outros dados relacionados a essas produções, como elenco, equipe técnica, créditos, trailers, reviews, temporadas, popularidade, trending de filmes, programas de TV e pessoas, etc.

Este trabalho utiliza somente as informações relacionadas aos filmes da API, como os gêneros, as empresas, reviews, elenco, equipe técnica e pessoas. Nesse contexto, analisamos e realizamos o tratamento dos dados fornecidos pela API em formato JSON antes de inseri-los no Postgresql, que foi o SGBD relacional escolhido para realizar esse trabalho devido à maior familiaridade do grupo com ele.

2. Diagrama Relacional

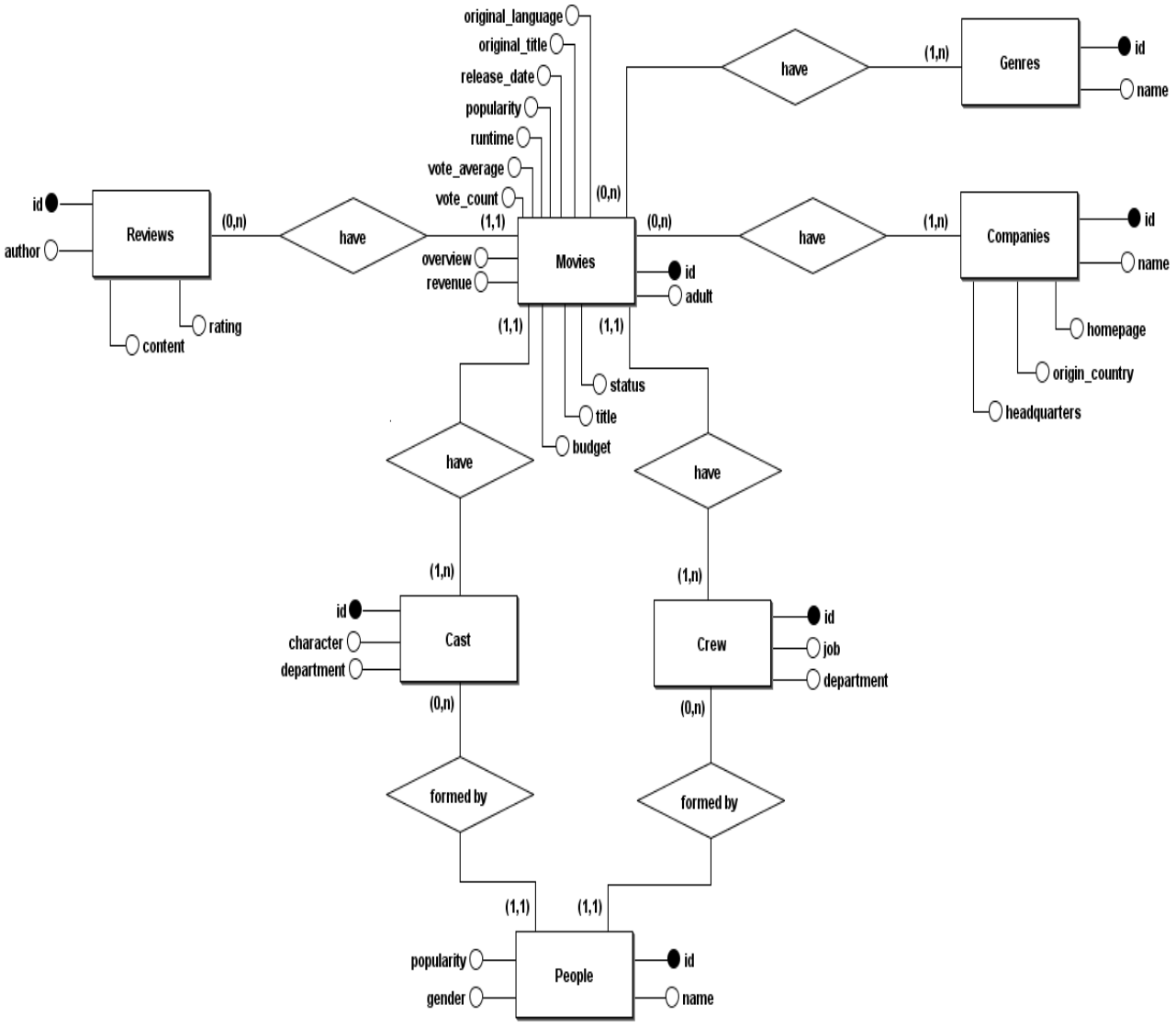

Figura 1 - Diagrama relacional

Segue abaixo os códigos SQL da criação das tabelas do diagrama relacional.

Query Editor | Query History | Data Output | Explain | Messages | Notifications

```sql
CREATE TABLE movies(
    id INT PRIMARY KEY,
    title VARCHAR(100),
    status VARCHAR(50),
    runtime INT,
    revenue BIGINT,
    release_date DATE,
    popularity DECIMAL,
    overview TEXT,
    original_title VARCHAR(100),
    original_language VARCHAR(50),
    budget BIGINT,
    adult BOOLEAN,
    vote_count INT,
    vote_average DECIMAL
);


CREATE TABLE reviews (
    id VARCHAR(50) PRIMARY KEY,
    author VARCHAR(100),
    content TEXT,
    rating INT CHECK (rating >= 0 AND rating <= 10),
    movieId INT NOT NULL,
    FOREIGN KEY (movieId) REFERENCES movies(id) ON DELETE CASCADE
);
```

Figura 2 - Criação das tabelas do diagrama relacional (parte 1)

Query Editor | Query History | Data Output | Explain | Messages | Notifications

```sql

CREATE TABLE genres (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);

CREATE TABLE movies_genres (
    movieId INT NOT NULL,
    genreId INT NOT NULL,
    FOREIGN KEY (movieId) REFERENCES movies(id) ON DELETE CASCADE,
    FOREIGN KEY (genreId) REFERENCES genres(id) ON DELETE CASCADE,
    PRIMARY KEY (movieId, genreId)
);

CREATE TABLE companies (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    homepage VARCHAR(100),
    origin_country VARCHAR(50),
    headquarters VARCHAR(100)
);

CREATE TABLE movies_companies (
    movieId INT NOT NULL,
    companyId INT NOT NULL,
    FOREIGN KEY (movieId) REFERENCES movies(id) ON DELETE CASCADE,
    FOREIGN KEY (companyId) REFERENCES companies(id) ON DELETE CASCADE,
    PRIMARY KEY (movieId, companyId)
);
```

Figura 3 - Criação das tabelas do diagrama relacional (parte 2)

Query Editor | Query History | Data Output | Explain | Messages | Notifications

```
56
57  CREATE TABLE people (
58      id INT PRIMARY KEY,
59      name VARCHAR(100),
60      gender INT CHECK(gender >=0 AND gender <= 3) DEFAULT 0,
61      popularity DECIMAL
62  );
63
64
65  CREATE TABLE tcast (
66      id VARCHAR(50) PRIMARY KEY,
67      character VARCHAR(100),
68      department VARCHAR(100),
69      personId INT NOT NULL,
70      movieId INT NOT NULL,
71      FOREIGN KEY (personId) REFERENCES people(id) ON DELETE CASCADE,
72      FOREIGN KEY (movieId) REFERENCES movies(id) ON DELETE CASCADE
73  );
74
75
76  CREATE TABLE crew (
77      id VARCHAR(50) PRIMARY KEY,
78      job VARCHAR(100),
79      department VARCHAR(100),
80      personId INT NOT NULL,
81      movieId INT NOT NULL,
82      FOREIGN KEY (personId) REFERENCES people(id) ON DELETE CASCADE,
83      FOREIGN KEY (movieId) REFERENCES movies(id) ON DELETE CASCADE
84  );
85
```

Figura 4 - Criação das tabelas do diagrama relacional (parte 3)

3. Grupos de usuários e suas permissões

Para realizar o acesso ao banco foram definidos os seguintes grupos de usuários:

- Role dba: Tem acesso total ao banco, pode realizar todas operações CRUD no esquema e em todas as tabelas. Este grupo pode conceder seus privilégios para outras roles ou usuários, e também pode criar novas roles ou usuários.
- Role programadores: Pode realizar as operações CRUD apenas para os registros de todas as tabelas do banco;
- Role usuarios_aplicação: Possui a permissão de select em todas as tabelas do banco

Query Query History

```
1  CREATE ROLE dba;
2  CREATE ROLE programadores;
3  CREATE ROLE usuarios_aplicacao;
4
5  CREATE USER adriano WITH PASSWORD 'adriano';
6  CREATE USER antonio WITH PASSWORD 'antonio';
7  CREATE USER bruno WITH PASSWORD 'bruno';
8  CREATE USER joao WITH PASSWORD 'joao';
9
10 GRANT ALL ON DATABASE "moviesDB" TO dba;
11 GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO programadores;
12 GRANT SELECT ON ALL TABLES IN SCHEMA public TO usuarios_aplicacao;
13
14 GRANT dba TO adriano;
15 GRANT programadores TO antonio;
16 GRANT programadores TO bruno;
17 GRANT usuarios_aplicacao TO joao;
```

Figura 5 - Definições dos grupos de usuários e suas permissões, além da criação dos usuários e atribuição das roles à eles

4. Índices

Os índices foram criados para facilitar as consultas em alguns dos atributos mais comuns das tabelas que são utilizados nos relatórios AD-HOC, e a seguir apresentaremos os índices usados em cada tabela.

- Tabela Tcast:
  - character: o elenco será mais consultado pelo nome do personagem no filme.
  - movieid e personid: esses atributos serão bastante utilizados para fazer joins com as suas respectivas tabelas, que são movies e people.

- Tabela Crew:
  - movieid e personid: assim como na tabela Tcast, esses atributos serão bastante utilizados para fazer joins com as suas respectivas tabelas, que são movies e people.

- Tabela People:
  - name e popularity: uma pessoa será bastante consultada pelo seu nome e popularidade.

- Tabela Movies:
    - title e popularity: um filme será consultado de forma mais fácil pelo seu título e popularidade.

Segue abaixo os códigos SQL da criação dos índices nas tabelas.

```sql
CREATE INDEX tcast_character_index ON tcast USING BTREE(character);
CREATE INDEX tcast_movieid_index ON tcast USING BTREE(movieid);
CREATE INDEX tcast_personid_index ON tcast USING BTREE(personid);

CREATE INDEX crew_movieid_index ON crew USING BTREE(movieid);
CREATE INDEX crew_personid_index ON crew USING BTREE(personid);

CREATE INDEX people_name_index ON people USING BTREE(name);
CREATE INDEX people_popularity_index ON people USING BTREE(popularity);

CREATE INDEX movies_title_index ON movies USING BTREE(title);
CREATE INDEX movies_popularity_index ON movies USING BTREE(popularity);
```

Figura 6 - Criação dos índices

5. Views

As views são utilizadas nos relatórios AD-HOC para não ser necessário fazer as consultas utilizando joins, pois já teremos a tabela com os joins necessários de cada relatório em sua respectiva view, e também para selecionar apenas as colunas que desejamos mostrar em cada relatório. Segue abaixo os códigos SQL da criação das views.

```sql
CREATE VIEW relatorio_movies AS
(
SELECT id, title, original_title, original_language, status, runtime, budget,
revenue, release_date, popularity, vote_count, vote_average
FROM movies
ORDER BY popularity DESC
);


CREATE VIEW relatorio_movies_tcast AS
(
SELECT m.id as movie_id, m.title, m.original_title, m.original_language, m.status,
m.runtime, m.budget, m.revenue, m.release_date, m.popularity as movie_popularity,
m.vote_count, m.vote_average, p.id as person_id, p.name, p.gender, p.popularity
as person_popularity, t.character, t.department
FROM movies m JOIN tcast t ON m.id = t.movieId JOIN people p ON p.id = t.personId
ORDER BY m.popularity DESC
);
```

Figura 7 - Criação das views (parte 1)

```sql
CREATE VIEW relatorio_movies_crew AS
(
SELECT m.id as movie_id, m.title, m.original_title, m.original_language, m.status,
m.runtime, m.budget, m.revenue, m.release_date, m.popularity, m.vote_count, m.vote_average,
p.id as person_id, p.name, p.gender, p.popularity as person_popularity, c.job, c.department
FROM movies m JOIN crew c ON m.id = c.movieId JOIN people p ON p.id = c.personId
ORDER BY m.popularity DESC
);


CREATE VIEW relatorio_movies_genres AS
(
SELECT m.id as movie_id, m.title, m.original_title, m.original_language, m.status,
m.runtime, m.budget, m.revenue, m.release_date, m.popularity, m.vote_count, m.vote_average,
g.id as genre_id, g.name as genre_name
FROM movies m JOIN movies_genres mg ON m.id = mg.movieId JOIN genres g ON g.id = mg.genreId
ORDER BY m.popularity DESC
);
```

Figura 8 - Criação das views (parte 2)

```sql
CREATE VIEW relatorio_movies_companies AS
(
SELECT m.id as movie_id, m.title, m.original_title, m.original_language, m.status, m.runtime,
m.budget, m.revenue, m.release_date, m.popularity, m.vote_count, m.vote_average,
c.id as companies_id, c.name as companies_name, c.homepage as companies_homepage,
c.origin_country as companies_origin_country, c.headquarters as companies_headquarters
FROM movies m JOIN movies_companies mc ON m.id = mc.movieId JOIN companies c ON c.id = mc.companyId
ORDER BY m.popularity DESC
);


CREATE VIEW relatorio_movies_reviews AS
(
SELECT m.id as movie_id, m.title, m.original_title, m.original_language, m.status, m.runtime,
m.budget, m.revenue, m.release_date, m.popularity, m.vote_count, m.vote_average, r.id as review_id,
r.author, r.content, r.rating
FROM movies m JOIN reviews r ON m.id = r.movieId
ORDER BY m.popularity DESC
);
```

Figura 9 - Criação das views (parte 3)

6. Count das tabelas

A figura a seguir mostra todas as tabelas criadas no nosso banco e a quantidade de registros em cada uma dessas tabelas.

moviesDB/postgres@PostgreSQL 14

Query Editor | Query History

```
1  CREATE OR REPLACE FUNCTION count_rows(tablename text) RETURNS integer AS
2  $$
3  DECLARE
4    result integer;
5  BEGIN
6    EXECUTE 'SELECT count(*) FROM ' || tablename INTO result;
7    RETURN result;
8  END;
9  $$ language plpgsql;
10
11 SELECT table_name, count_rows(table_name)
12 FROM information_schema.tables
13 WHERE table_schema NOT IN ('pg_catalog', 'information_schema')
14 ORDER by count_rows(table_name) DESC
```

Data Output | Explain | Messages | Notifications

	table_name name	count_rows integer
1	crew	155422
2	people	118636
3	tcast	62501
4	movies_companies	5994
5	movies_genres	5316
6	reviews	3661
7	companies	2562
8	movies	1945
9	genres	19

Figura 10 - Count das tabelas

7. Resultados dos testes de performance obtidos com JMeter

Para realizar os testes de performance no Jmeter escolhemos a seguinte consulta, que retornava 30916844 bytes no total.

```
1  SELECT * FROM movies m  JOIN tcast t ON m.id = t.movieId
2  JOIN people p ON p.id = t.personId
3  ORDER BY m.popularity DESC;
```

Figura 11 - Consulta utilizada nos testes de performance

Os testes de performance foram realizados utilizando um notebook com a seguinte arquitetura:

- Processador: Intel(R) Core(TM) i7 - 8750H CPU @ 2.20GHz, 2208 Mhz, 6 núcleos, 12 processadores lógicos
- Memória RAM: 16 GB
- SSD: 128GB
- HD: 1TB
- Sistema operacional: Windows 11 Pro

Os resultados dos testes de performance obtidos com o Jmeter serão apresentados a seguir.

7.1. Requisições x Latência com 30 usuários fixos

Requisições	Média de Latência
30	11491 ms
60	12535 ms
90	13246 ms
120	13675 ms
150	14035 ms

Tabela 1 - Média de latência por requisições com 30 usuários simultâneos realizando requisições

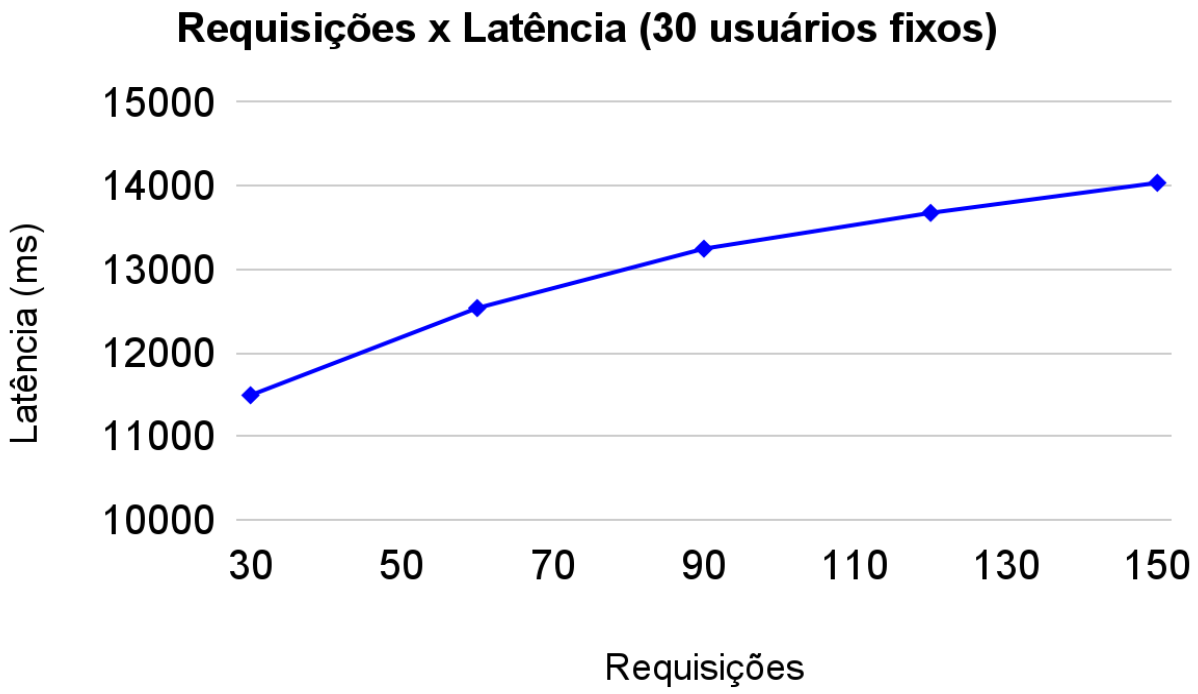

Gráfico 1 - Média de latência por requisições com 30 usuários simultâneos realizando requisições

7.2. Usuários x Latência com 1 requisição fixa

Usuários	Média de Latência
10	5514 ms
30	11420 ms
50	19247 ms
70	27080 ms
90	34111 ms
110	44738 ms

Tabela 2 - Média de latência por usuários simultâneos realizando uma requisição.

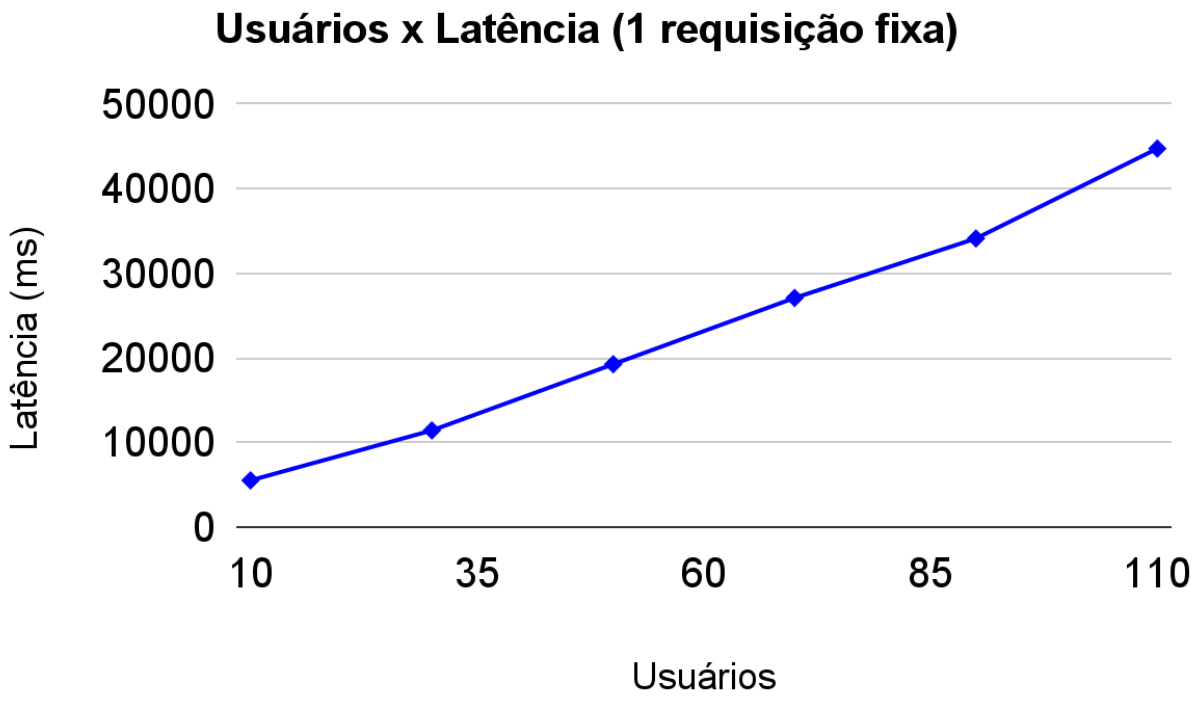

Gráfico 2 - Média de latência por usuários simultâneos realizando 1 requisição

8. Links para o código desenvolvido e para o vídeo mostrando a aplicação funcionando

8.1. Código:

- https://github.com/bjmvercelli/trabalho-pratico-bd2

8.2. Vídeo:

- https://drive.google.com/file/d/1Hvi3r41EDym9chmQWrjlB12L7lPdRfxa/view?usp=sharing