



Universidade Federal de Itajubá

# Relatório Final - Trabalho Prático

COM231 – Banco de Dados II

Fábio Piovani Viviani - 2017006774  
João Pedro Lopes Dias Ribeiro - 2017002176  
Rodrigo de Andrade Porto - 2019000500  
Thiago Geovane dos Santos - 2016014154  
Ygor Salles Aniceto Carvalho - 2017014382

Professora: Vanessa Cristina Oliveira de Souza

**Novembro**

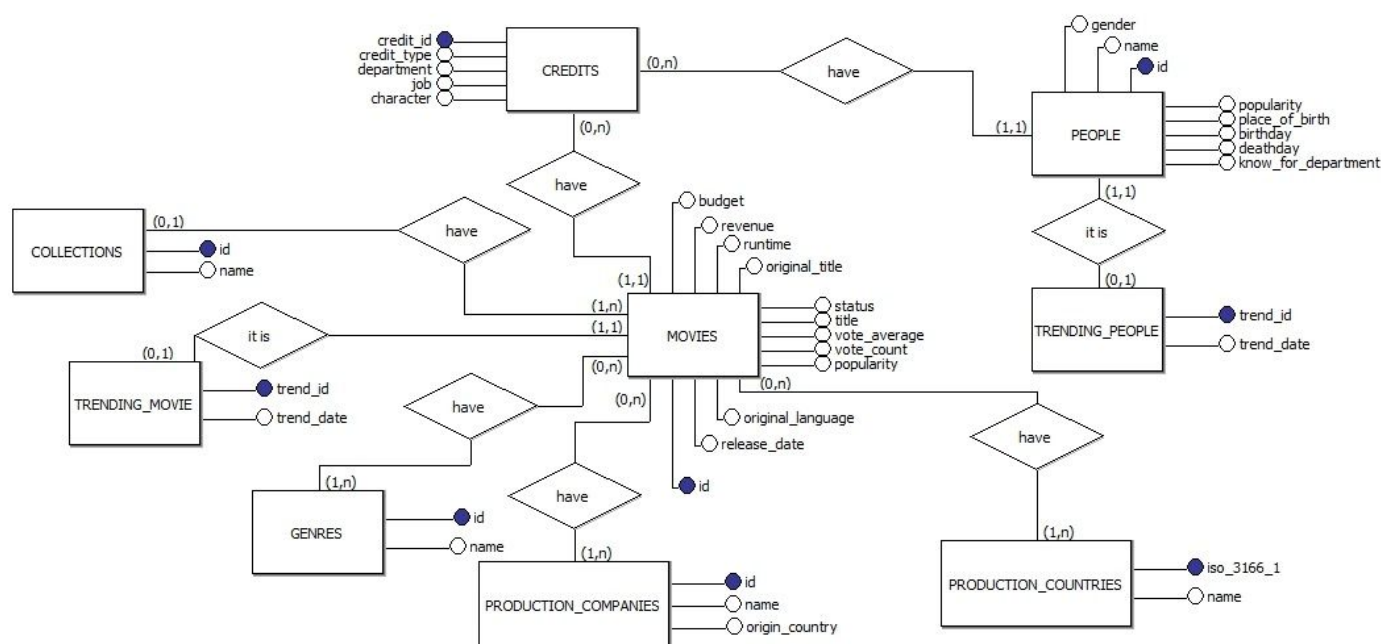
**2020**

## 1 - Definição da aplicação:

Nossa aplicação possui informações de diversos filmes e seus gêneros, coleções, produções, créditos, filmes em trending top do momento e outras informações adicionais.

Na busca pela utilização de uma API que atendesse nossas demandas esperadas, escolhemos a TheMovieDB (TMDb), que exige a criação de uma conta na plataforma para solicitação de uma key gratuita para utilização.

Com isso, realizamos a modelagem do banco de dados da nossa aplicação, que pode ser visto de acordo com a imagem abaixo.



**Figura 1 - Modelagem do banco de dados da nossa aplicação**

## 2 - Modelo do banco SQL:

O banco SQL é uma linguagem de pesquisa declarativa padrão para banco de dados relacional, no qual muitas características foram inspiradas na Álgebra Relacional. Neste trabalho foi utilizado o SGBD(Sistema Gerenciador de Banco de Dados) PostgreSQL para o banco de dados SQL.

Abaixo, pode-se consultar o modelo criado e utilizado para a funcionalidade da nossa aplicação:

```
CREATE TABLE genres (  
    id INTEGER PRIMARY KEY,  
    name VARCHAR(50)  
);
```

```
CREATE TABLE collections (  
    id BIGINT PRIMARY KEY,  
    name VARCHAR(150)  
);
```

```
CREATE TABLE production_companies (  
    id BIGINT PRIMARY KEY,  
    name VARCHAR(150),  
    origin_country VARCHAR(5)  
);
```

```
CREATE TABLE production_countries (  
    iso_3166_1 VARCHAR(5) PRIMARY KEY,  
    name VARCHAR(50)  
);
```

```
CREATE TABLE people (  
    id BIGINT PRIMARY KEY,  
    name VARCHAR(150),  
    gender INTEGER,  
    popularity FLOAT,  
    place_of_birth VARCHAR(150),  
    birthday DATE,  
    deathday DATE,  
    know_for_department VARCHAR(50)  
);
```

```
CREATE TABLE movies (  
    id BIGINT PRIMARY KEY,  
    original_language VARCHAR(30),  
    original_title VARCHAR(150),  
    popularity FLOAT,  
    status VARCHAR(30),  
    title VARCHAR(150),  
    vote_average FLOAT,  
    vote_count BIGINT,  
    release_date DATE,  
    budget FLOAT,
```

```
    revenue FLOAT,  
    runtime INTEGER,  
    id_collection BIGINT,  
        FOREIGN KEY (id_collection) REFERENCES collections(id)  
);
```

```
CREATE TABLE trending_movies (  
    trend_id INTEGER PRIMARY KEY,  
    trend_date DATE,  
    id_movie BIGINT UNIQUE,  
        FOREIGN KEY (id_movie) REFERENCES movies(id)  
);
```

```
CREATE TABLE movie_production_companies (  
    id_movie BIGINT,  
    id_production_company BIGINT,  
    PRIMARY KEY(id_movie, id_production_company),  
        FOREIGN KEY(id_movie) REFERENCES movies(id),  
        FOREIGN KEY(id_production_company) REFERENCES  
production_companies(id)  
);
```

```
CREATE TABLE movie_production_countries (  
    id_movie BIGINT,  
    iso_3166_1 VARCHAR(5) NOT NULL,  
    PRIMARY KEY (id_movie, iso_3166_1),  
        FOREIGN KEY (id_movie) REFERENCES movies(id),  
        FOREIGN KEY (iso_3166_1) REFERENCES  
production_countries(iso_3166_1)  
);
```

```
CREATE TABLE movie_genres (  
    id_movie BIGINT,  
    id_genre INTEGER NOT NULL,  
    PRIMARY KEY(id_movie, id_genre),  
        FOREIGN KEY (id_movie) REFERENCES movies(id),  
        FOREIGN KEY (id_genre) REFERENCES genres(id)  
);
```

```
CREATE TABLE credits (  
    credit_id VARCHAR(30) PRIMARY KEY,  
    credit_type VARCHAR(30),  
    department VARCHAR(30),  
    job VARCHAR(50),
```

```

character VARCHAR(150),
id_movie BIGINT,
id_people BIGINT,
    FOREIGN KEY(id_movie) REFERENCES movies(id),
    FOREIGN KEY(id_people) REFERENCES people(id)
);

CREATE TABLE trending_people (
    trend_id INTEGER PRIMARY KEY,
    trend_date DATE,
    id_people BIGINT UNIQUE,
    FOREIGN KEY (id_people) REFERENCES people(id)
);

```

### 3 - Permissões de grupo de usuários

Para acesso ao banco de dados da aplicação foi definido alguns perfis de usuário:

- **Perfil de desenvolvedor:** Os desenvolvedores terão acesso a somente CRUD na aplicação, não poderão alterar o Schema do banco. É como um usuário de aplicação que tem acesso CRUD a toda a aplicação somente, para poder fazer os testes necessários no desenvolvimento da aplicação.

```

CREATE ROLE desenvolvedores;
GRANT INSERT, SELECT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO desenvolvedores;

```

Figura  
de

```

CREATE ROLE comum;
GRANT SELECT ON TABLE movies TO comum;

```

2 - Criação do perfil  
desenvolvedor

- **Perfil de usuário comum:** Os usuários comuns são os visitantes da aplicação, e estes têm acesso somente para consultar os filmes.

Figura 3 - Criação do perfil de usuário comum

- **Perfil de usuário premium:** Os usuários premium são também visitantes da aplicação porém diferentemente dos usuários comuns, estes possuem acesso a consultar não somente aos filmes mais quais estão no top trending, os créditos, etc. Em resumo, podem consultar todo o dado disponibilizado pela aplicação

```
CREATE ROLE premium;  
GRANT SELECT ON ALL TABLES IN SCHEMA public TO premium;
```

**Figura 4 - Criação do perfil de usuário premium**

- **Perfil de usuário DBA - Administrador do Banco de Dados:** Os usuários dba\_public são os usuários de banco de dados tendo todo o privilégio no banco da aplicação

```
CREATE ROLE dba_public;  
GRANT ALL PRIVILEGES ON DATABASE "moviedb3" TO dba_public;  
GRANT ALL ON ALL TABLES IN SCHEMA public TO dba_public;  
GRANT USAGE ON SCHEMA public TO dba_public;
```

**Figura 5 - Criação do perfil de usuário DBA**

- **Perfil de usuário Aplicação:** Os usuários de aplicação são os que irão gerenciar aplicação, ou seja são aqueles que irão popular o banco, remover, alterar ou consultar os dados, os catálogos de filmes, etc. São os geradores de conteúdo. Em outras palavra são os funcionários da empresa que adquiriu o sistema. Eles possuem o mesmo acesso do desenvolvedor pois para o desenvolvedor construir a aplicação terá que assumir o papel desse perfil de usuário para poder realizar os testes necessários enquanto constroem a aplicação.

```
CREATE ROLE aplicacao;  
GRANT INSERT, SELECT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO aplicacao;
```

**Figura 6 - Criação do perfil de usuário Aplicação**

### 3 - Modelo do banco de dados orientado a documentos

Um banco de dados orientado a documentos é projetado para recuperar, gerenciar e armazenar informações orientadas a documentos, sendo uma das categorias principais de banco de dados, salientando sua forte relação com o modelo de bancos de dados relacionais.

Os bancos de dados orientados a documentos armazenam as informações em uma instância no banco, onde cada objeto pode ser diferente dos outros, tornando o mapeamento de objetos uma tarefa simplificada e com diferenciais ao mapeamento objeto-relacional.

Abaixo, pode-se consultar o modelo criado e utilizado para a funcionalidade da nossa aplicação.

```
movies: [{  
  _id: ,  
  belongs_to_collection: {  
    id: ,  
    name:  
  },  
  original_language: ,  
  original_title: ,  
  title: ,  
  popularity: ,  
  status: ,  
  vote_average: ,  
  vote_count: ,  
  genres: [  
    {  
      id: ,  
      name:  
    }  
  ],  
  release_date: ,  
  budget: ,  
  revenue: ,  
  runtime: ,  
  production_companies: [  
    {  
      id: ,  
      name: ,  
      origin_country:
```

```

        }
    ],
    production_countries: [
        {
            iso_3166_1: ,
            name: "
        }
    ]
}
}

```

```

credits: [{
    _id: ,
    credit_type: ,
    department: ",
    job: ",
    media: {
        id: ,
        vote_count: ,
        vote_average: ,
        title: ,
        release_date: ,
        original_language: ,
        original_title: ,
        genre_ids: [],
        popularity: ,
        character:
    },
    person: {
        id: ,
        name: ,
        gender: ,
        known_for_department: ,
        popularity:
    }
}
}

```

```

people: [{
    _id: ,
    name: ,
    gender: ,
    popularity: ,
    place_of_birth: ,
    birthday: ,
    deathday: ,
    known_for_department:
}
}

```



```
genres: [{  
  _id: ,  
  name:  
}]
```

```
collections: [{  
  _id: ,  
  name: ,  
  parts: {  
    id: ,  
    title: ,  
    release_date: ,  
    original_language: ,  
    original_title: ,  
    genre_ids: [],  
    popularity:  
  }  
}]
```

```
trending_movies: [{  
  _id: ,  
  movie: {  
    id: ,  
    title: "  
    release_date: "  
    original_language: "  
    original_title: "  
    genre_ids: [],  
    popularity:  
  },  
  trend_date: "  
}]
```

```
trending_people: [{  
  _id: ,  
  person: {  
    id: ,  
    name: ,  
    known_for_department: ,  
    popularity:  
  },  
  trend_date:
```

}}

#### **4 - Tabelas e counts**

Em nossa aplicação, realizamos a criação das seguintes tabelas em relação às informações utilizadas pela API. Abaixo é possível ver uma relação das tabelas utilizadas.

Tabelas:

1. collections
2. credits
3. genres
4. movie\_genres
5. movie\_production\_companies
6. movie\_production\_countries
7. movies
8. people
9. production\_companies
10. production\_countries
11. trending\_movies
12. trending\_people

Nas imagens abaixo, pode-se ver capturas de telas referentes aos counts das tabelas utilizadas em nossa aplicação.

Query Editor		Query History	
1	<code>select count(*) from collections</code>		
2	<code>select count(*) from credits</code>		
3	<code>select count(*) from genres</code>		
4	<code>select count(*) from movie_genres</code>		
5	<code>select count(*) from movie_production_companies</code>		
6	<code>select count(*) from movie_production_countries</code>		
7	<code>select count(*) from movies</code>		
8	<code>select count(*) from people</code>		
9	<code>select count(*) from production_companies</code>		
10	<code>select count(*) from production_countries</code>		
11	<code>select count(*) from trending_movies</code>		
12	<code>select count(*) from trending_people</code>		
13			
Data Output		Explain	Messages
	count bigint		
1	159		

Query Editor		Query History	
1	<code>select count(*) from collections</code>		
2	<code>select count(*) from credits</code>		
3	<code>select count(*) from genres</code>		
4	<code>select count(*) from movie_genres</code>		
5	<code>select count(*) from movie_production_companies</code>		
6	<code>select count(*) from movie_production_countries</code>		
7	<code>select count(*) from movies</code>		
8	<code>select count(*) from people</code>		
9	<code>select count(*) from production_companies</code>		
10	<code>select count(*) from production_countries</code>		
11	<code>select count(*) from trending_movies</code>		
12	<code>select count(*) from trending_people</code>		
13			
Data Output		Explain	Messages
	count bigint		
1	8916		

Query Editor		Query History	
1	<code>select count(*) from collections</code>		
2	<code>select count(*) from credits</code>		
3	<code>select count(*) from genres</code>		
4	<code>select count(*) from movie_genres</code>		
5	<code>select count(*) from movie_production_companies</code>		
6	<code>select count(*) from movie_production_countries</code>		
7	<code>select count(*) from movies</code>		
8	<code>select count(*) from people</code>		
9	<code>select count(*) from production_companies</code>		
10	<code>select count(*) from production_countries</code>		
11	<code>select count(*) from trending_movies</code>		
12	<code>select count(*) from trending_people</code>		
13			
Data Output		Explain	Messages
	count bigint		
1	19		

Query Editor		Query History	
1	<code>select count(*) from collections</code>		
2	<code>select count(*) from credits</code>		
3	<code>select count(*) from genres</code>		
4	<code>select count(*) from movie_genres</code>		
5	<code>select count(*) from movie_production_companies</code>		
6	<code>select count(*) from movie_production_countries</code>		
7	<code>select count(*) from movies</code>		
8	<code>select count(*) from people</code>		
9	<code>select count(*) from production_companies</code>		
10	<code>select count(*) from production_countries</code>		
11	<code>select count(*) from trending_movies</code>		
12	<code>select count(*) from trending_people</code>		
13			
Data Output		Explain	Messages
	count bigint		
1	2701		

**Figura 7 - Counts das tabelas (parte 1)**

moviedb/postgres@PostgreSQL 11

Query Editor

Query History

```
1 select count(*) from collections
2 select count(*) from credits
3 select count(*) from genres
4 select count(*) from movie_genres
5 select count(*) from movie_production_companies
6 select count(*) from movie_production_countries
7 select count(*) from movies
8 select count(*) from people
9 select count(*) from production_companies
10 select count(*) from production_countries
11 select count(*) from trending_movies
12 select count(*) from trending_people
13
```

Data Output

Explain

Messages

Notifications

	count	bigint
1	3003	

moviedb/postgres@PostgreSQL 11

Query Editor

Query History

```
1 select count(*) from collections
2 select count(*) from credits
3 select count(*) from genres
4 select count(*) from movie_genres
5 select count(*) from movie_production_companies
6 select count(*) from movie_production_countries
7 select count(*) from movies
8 select count(*) from people
9 select count(*) from production_companies
10 select count(*) from production_countries
11 select count(*) from trending_movies
12 select count(*) from trending_people
13
```

Data Output

Explain

Messages

Notifications

	count	bigint
1	1271	

moviedb/postgres@PostgreSQL 11

Query Editor

Query History

```
1 select count(*) from collections
2 select count(*) from credits
3 select count(*) from genres
4 select count(*) from movie_genres
5 select count(*) from movie_production_companies
6 select count(*) from movie_production_countries
7 select count(*) from movies
8 select count(*) from people
9 select count(*) from production_companies
10 select count(*) from production_countries
11 select count(*) from trending_movies
12 select count(*) from trending_people
13
```

Data Output

Explain

Messages

Notifications

	count	bigint
1	957	

moviedb/postgres@PostgreSQL 11

Query Editor

Query History

```
1 select count(*) from collections
2 select count(*) from credits
3 select count(*) from genres
4 select count(*) from movie_genres
5 select count(*) from movie_production_companies
6 select count(*) from movie_production_countries
7 select count(*) from movies
8 select count(*) from people
9 select count(*) from production_companies
10 select count(*) from production_countries
11 select count(*) from trending_movies
12 select count(*) from trending_people
13
```

Data Output

Explain

Messages

Notifications

	count	bigint
1	6074	

**Figura 8 - Counts das tabelas (parte 2)**

moviedb/postgres@PostgreSQL 11

Query Editor

Query History

```
1 select count(*) from collections
2 select count(*) from credits
3 select count(*) from genres
4 select count(*) from movie_genres
5 select count(*) from movie_production_companies
6 select count(*) from movie_production_countries
7 select count(*) from movies
8 select count(*) from people
9 select count(*) from production_companies
10 select count(*) from production_countries
11 select count(*) from trending_movies
12 select count(*) from trending_people
13
```

Data Output

Explain

Messages

Notifications

	count	bigint
1	1206	

moviedb/postgres@PostgreSQL 11

Query Editor

Query History

```
1 select count(*) from collections
2 select count(*) from credits
3 select count(*) from genres
4 select count(*) from movie_genres
5 select count(*) from movie_production_companies
6 select count(*) from movie_production_countries
7 select count(*) from movies
8 select count(*) from people
9 select count(*) from production_companies
10 select count(*) from production_countries
11 select count(*) from trending_movies
12 select count(*) from trending_people
13
```

Data Output

Explain

Messages

Notifications

	count	bigint
1	43	

moviedb/postgres@PostgreSQL 11

Query Editor

Query History

```
1 select count(*) from collections
2 select count(*) from credits
3 select count(*) from genres
4 select count(*) from movie_genres
5 select count(*) from movie_production_companies
6 select count(*) from movie_production_countries
7 select count(*) from movies
8 select count(*) from people
9 select count(*) from production_companies
10 select count(*) from production_countries
11 select count(*) from trending_movies
12 select count(*) from trending_people
13
```

Data Output

Explain

Messages

Notifications

	count	bigint
1	624	

moviedb/postgres@PostgreSQL 11

Query Editor

Query History

```
1 select count(*) from collections
2 select count(*) from credits
3 select count(*) from genres
4 select count(*) from movie_genres
5 select count(*) from movie_production_companies
6 select count(*) from movie_production_countries
7 select count(*) from movies
8 select count(*) from people
9 select count(*) from production_companies
10 select count(*) from production_countries
11 select count(*) from trending_movies
12 select count(*) from trending_people
13
```

Data Output

Explain

Messages

Notifications

	count	bigint
1	678	

**Figura 9 - Counts das tabelas (parte 3)**



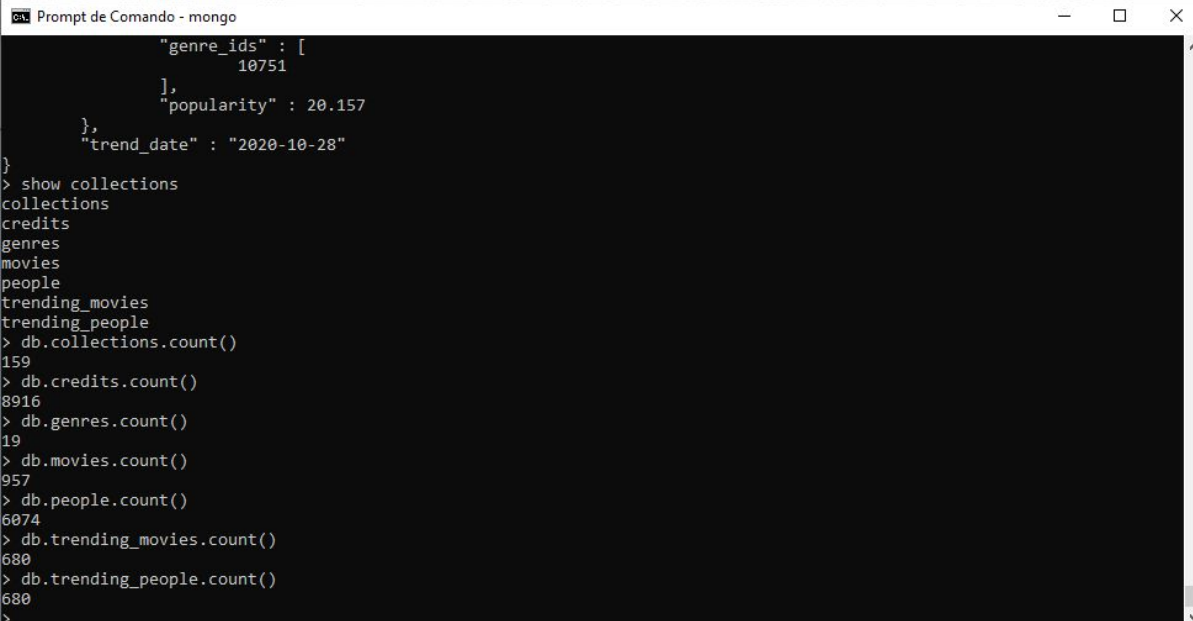
## 5 - Count em cada coleção

Também houveram a utilização de coleções para a aplicação em questão, são elas:

Coleções:

- 13.collections
- 14.credits
- 15.genres
- 16.movies
- 17.people
- 18.trending\_movies
- 19.trending\_people

Na imagem abaixo, pode-se ver capturas de telas referentes aos counts de cada coleção utilizada em nossa aplicação.



```
Prompt de Comando - mongo

    "genre_ids" : [
      10751
    ],
    "popularity" : 20.157
  },
  "trend_date" : "2020-10-28"
}
> show collections
collections
credits
genres
movies
people
trending_movies
trending_people
> db.collections.count()
159
> db.credits.count()
8916
> db.genres.count()
19
> db.movies.count()
957
> db.people.count()
6074
> db.trending_movies.count()
680
> db.trending_people.count()
680
>
```

**Figura 10 - Count das coleções**

## 6 - Resultados do teste de performance

### 6.1 - PostgreSQL

A realização do teste de performance do Jmeter com Postgres ocorreu com sucesso. Inicialmente, alteramos o valor da configuração max\_conections para 10.000 no banco de dados.

Após realizar a conexão do banco no Jmeter, realizamos os testes necessários na tabela people. Abaixo, pode-se analisar os resultados encontrados.

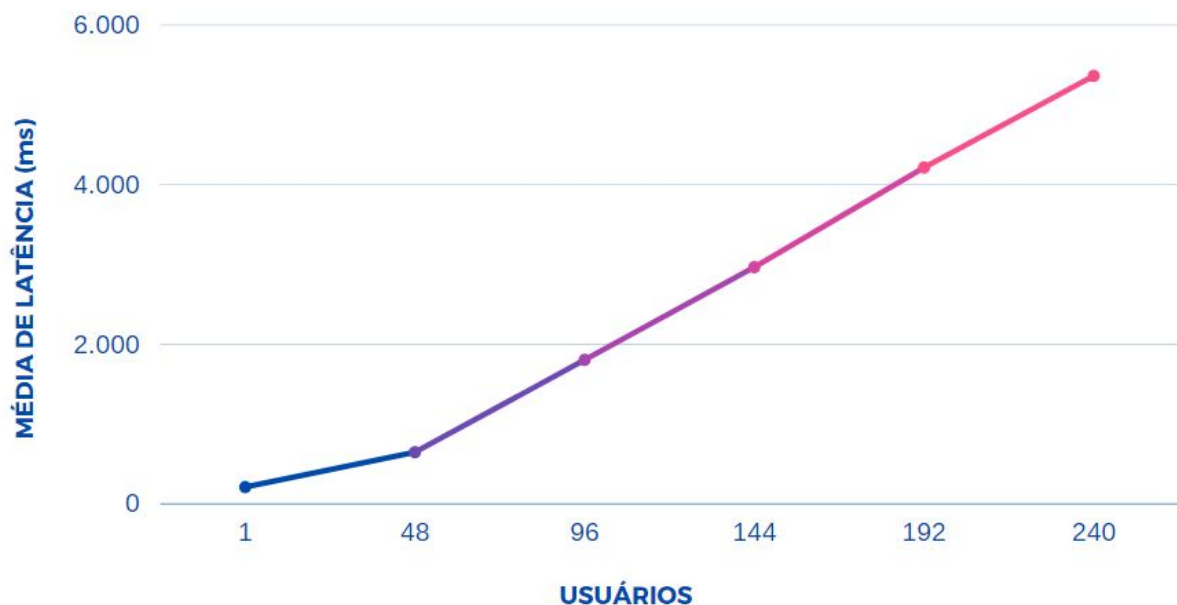
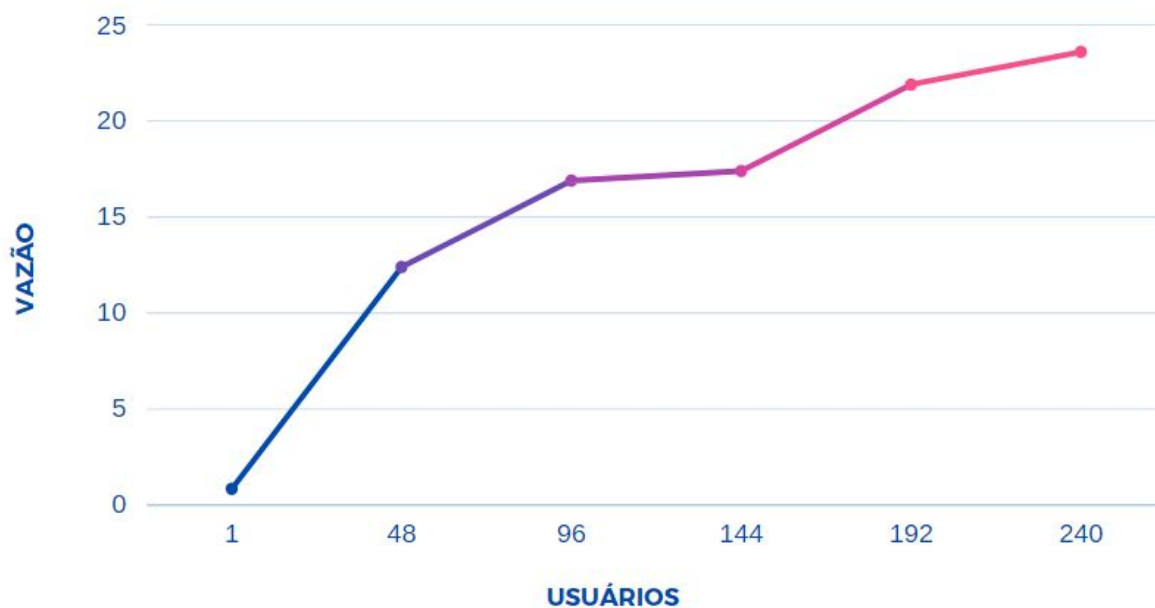


Gráfico 1 - Média de latência por usuários simultâneos realizando 1 requisição

USUÁRIOS	MÉDIA DE LATÊNCIA
1	211,43 ms
48	647,72 ms
96	1804,86 ms
1144	2967,21 ms
192	4213,59 ms
240	5363,45 ms

Tabela 1 - Média de latência por usuários simultâneos realizando 1 requisição

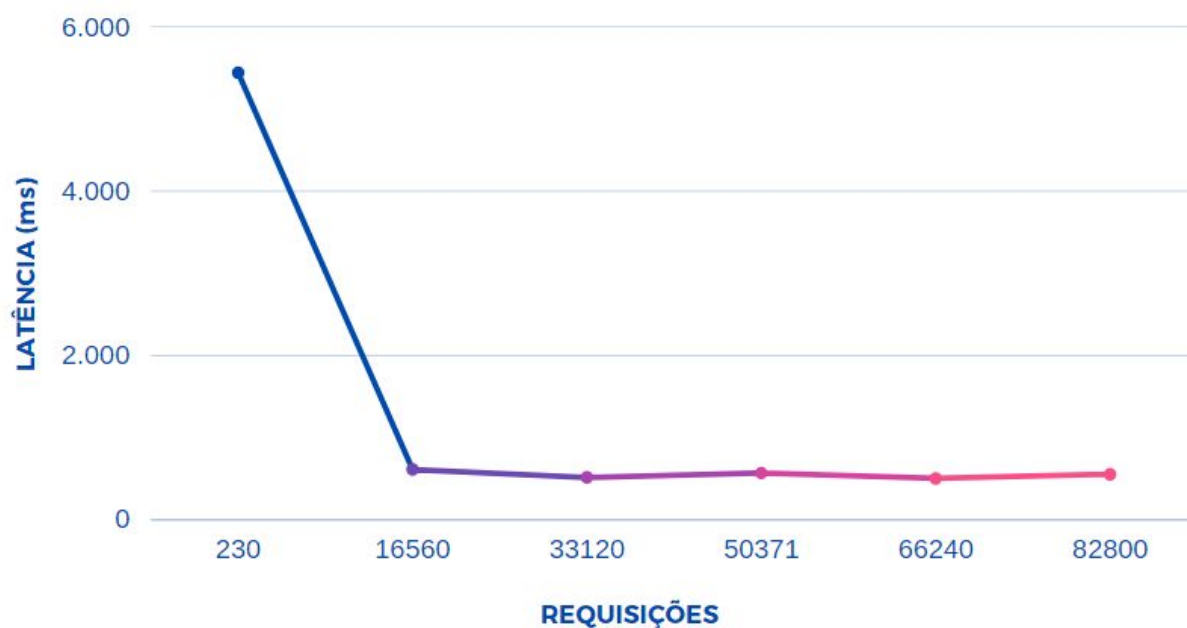


**Gráfico 2 - Usuários por vazão (vazão = número de requisições x tempo total)**

USUÁRIOS	VAZÃO
1	0,85/seg
48	12,4/seg
96	16,9/seg
1144	17,4/seg
192	21,9/seg
240	23,6/seg

**Tabela 2 - Usuários por vazão (vazão = número de requisições x tempo total)**

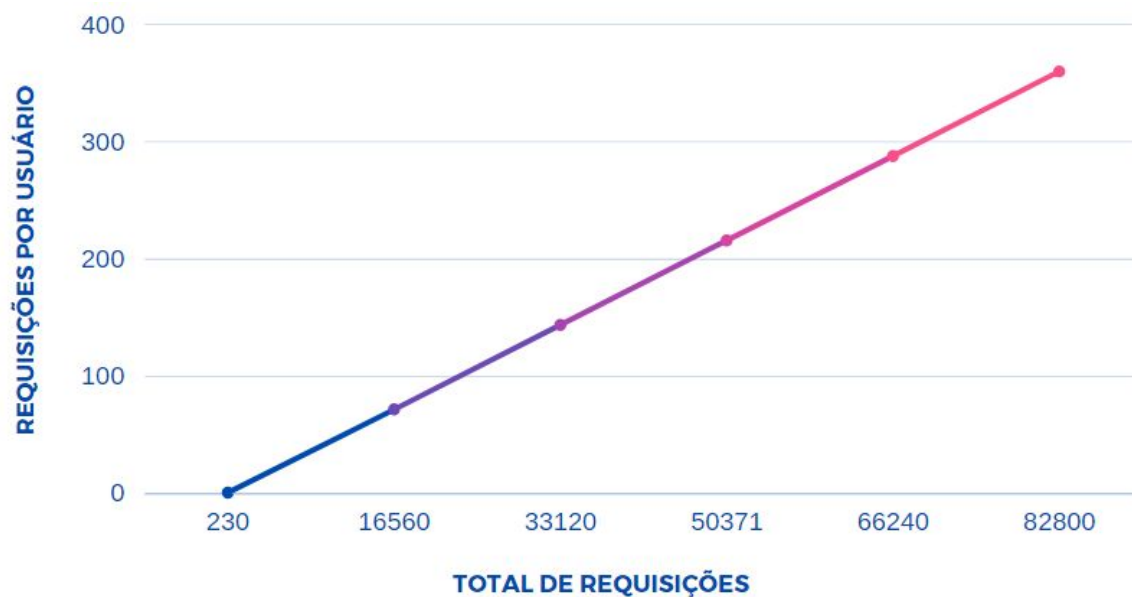




**Gráfico 3 - Quantidade de máxima requisições por latência com 230 usuários fixos**

REQUISIÇÕES	LATÊNCIA
230	5437,95 ms
16560	615,33 ms
33120	518,44 ms
50371	571,97 ms
66240	506,39 ms
82800	557,44 ms

**Tabela 3 - Quantidade de máxima requisições por latência com 230 usuários fixos**

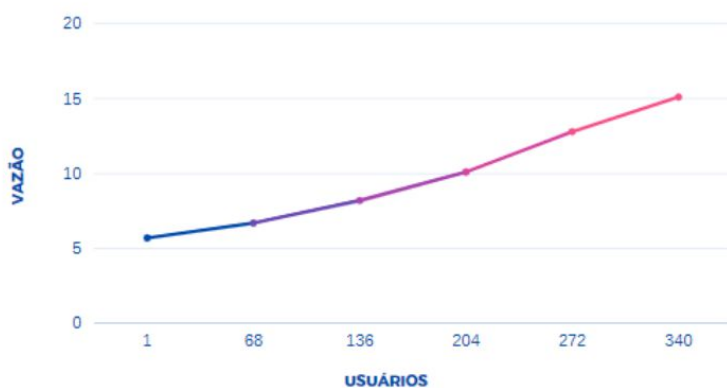


**Gráfico 4 - Quantidade de requisições por usuário (230 usuários fixos)**

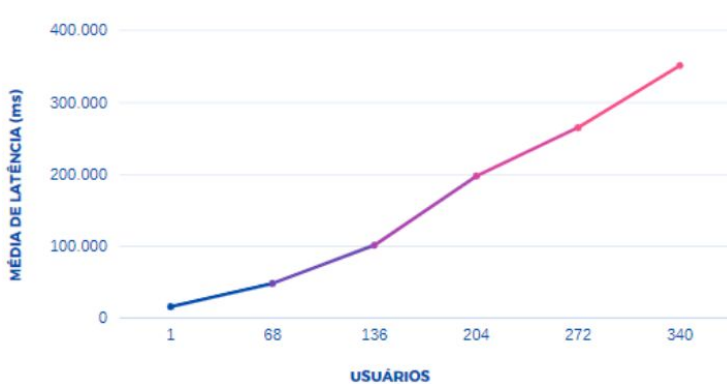
REQUISIÇÕES	LATÊNCIA
230	1
16560	72
33120	144
50371	216
66240	288
82800	360

**Tabela 4 - Quantidade de requisições por usuário (230 usuários fixos)**

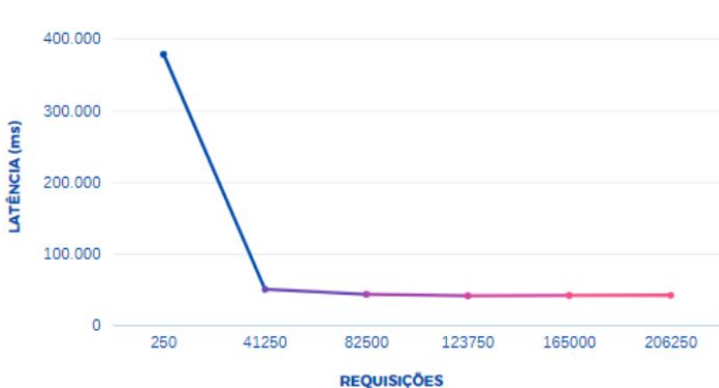
No MongoDB obtivemos os seguintes resultados que podem ser visualizados abaixo:



**Gráfico 5 - Usuários por vazão (vazão = número de requisições x tempo total)**



**Gráfico 6 - Média de latência por usuários simultâneos realizando 1 requisição**



**Gráfico 7 - Quantidade de máxima de requisições por latência com 250 usuários fixos**

**Conclusão:** O banco de dados orientado a documentos teve um desempenho superior ao banco de dados relacional, pois na medida em que a base de dados cresceu, o desempenho do mesmo foi mais rápido.

## 7 - Códigos do projeto

Primeira entrega (persistência dos dados da API no banco):

<https://github.com/fpviviani/python-moviedb>

Vídeo com explicações do código da primeira entrega:

[https://drive.google.com/file/d/1h9WqfBs7gzLLA-N4nTxnDH\\_7OE5LFq0P/view?usp=sharing](https://drive.google.com/file/d/1h9WqfBs7gzLLA-N4nTxnDH_7OE5LFq0P/view?usp=sharing)

Segunda entrega (front-end relatórios):

<https://github.com/fpviviani/laravel-moviedb>

## Referências

Apache Software Foundation. **Documentação Jmeter**. Disponível em: <https://jmeter.apache.org/index.html>. Acesso em: 25 nov. 2020.

BERWALDT, A. **JMeter Teste em Banco de Dados**. Disponível em: <https://medium.com/jmeter/jmeter-af2d4c00cae0>. Acesso em: 24 nov. 2020.

MONTEIRO, D. **Introdução para modelagem de dados para banco orientado a documentos**. Disponível em: <https://imasters.com.br/banco-de-dados/introducao-para-modelagem-de-dados-para-banco-orientado-documentos>. Acesso em: 12 nov. 2020.

TONKOV, K. **MongoDB Performance Testing with JMeter**. Disponível em: <https://www.blazemeter.com/blog/mongodb-performance-testing-with-jmeter>. Acesso em: 26 nov. 2020.