



Comic Vine - Dados de Personagens

RELATÓRIO DO PROJETO BD

Equipe:

André Lucca Gomides de Lima - 2021005648

Ivan Leoni Vilas Boas - 2018009073

Matheus Robusti H. Marqui - 2019007055

Fernando Pereira Goulart - 2019017937



Unifei - Itajubá
12/06/2023



Índice

1. VISÃO GERAL DA APLICAÇÃO	4
1.1. ESTUDO DA API: COMIC VINE	4
1.2. FERRAMENTAS	5
2. MODELOS	6
2.1. ER	6
2.2. MODELO LÓGICO	6
2.3. FÍSICO	6
3. ROLES	8
3.1. ROLE ADMINISTRADOR	8
3.2. ROLES DOS CLIENTES	8
3.2.1. ROLE GEEKS	9
3.2.2. ROLE FILMES	9
3.2.3. ROLE PREMIUM	10
4. USUÁRIOS	10
4.1. DIREITOS	11
5. PRINT COM O COUNT DAS TABELAS.	13
6. DEFINIÇÃO DE ÍNDICES E JUSTIFICATIVAS.	15
6.1. ÍNDICES DA TABELA CHARACTERS	15
6.2. ÍNDICES DA TABELA MOVIES	16
6.3. ÍNDICES DA TABELA VOLUMES	17
6.4. ÍNDICES DA TABELA EDITORS	18
7. TESTE JMETER	20
7.1. REALIZAÇÃO DOS TESTES NO JMETER	20
7.1.1. TESTE PARA NÚMERO DE USUÁRIOS	22
7.1.2. TESTE PARA NÚMERO DE REQUISIÇÕES	23
8. LINK PARA CÓDIGO	27
9. LINK PARA VÍDEO	27
ANEXO I - ER	28
ANEXO II - LÓGICO	29



Índice de Ilustrações

FIGURA 1- DIREITOS DOS GRUPOS.....	11
FIGURA 2 - PARTE DOS DIREITOS DO GRUPO ADM	11
FIGURA 3 – DIREITOS DO GRUPO GEEKS	12
FIGURA 4 - DIREITOS DO GRUPO FILMES.....	12
FIGURA 5- DIREITOS DO GRUPO PREMIUM	13
FIGURA 6 - DIREITOS DOS USUÁRIOS.....	13
FIGURA 7 - QUANTIDADE DE PERSONAGENS	14
FIGURA 8 - QUANTIDADE DE EDITORES.....	14
FIGURA 9 - QUANTIDADE DE FILMES.....	14
FIGURA 10 - QUANTIDADE DE SUPER PODERES.....	14
FIGURA 11- QUANTIDADE DE HISTÓRIAS EM QUADRINHOS	15
FIGURA 12 - ANÁLISE DO ATRIBUTO NAME DA TABELA PERSONAGEM	16
FIGURA 13 - ANÁLISE DO ÍNDICE NAME DE PERSONAGEM.....	16
FIGURA 14 - ANÁLISE DO ATRIBUTO NAME DA TABELA FILMES	17
FIGURA 15 - ANÁLISE DO ÍNDICE NAME DE FILMES	17
FIGURA 16 - ANÁLISE DO ATRIBUTO NAME DA TABELA VOLUMES.....	18
FIGURA 17 - ANÁLISE DO ÍNDICE NAME DAS HISTÓRIAS EM QUADRINHOS	18
FIGURA 18 - ANÁLISE DO ATRIBUTO NAME DA TABELA EDITORES	19
FIGURA 19 - ANÁLISE DO ÍNDICE NAME DOS EDITORES.....	19
FIGURA 20 - ÍNDICES SECUNDÁRIOS DO BANCO	19
FIGURA 21 - GRÁFICO DE USUÁRIO X LATÊNCIA	23
FIGURA 22- CONFIGURAÇÃO DO TESTE PARA REQUISIÇÕES INFINITAS	23
FIGURA 23 - EXECUÇÃO DO TESTE PARA 1 USUÁRIO COM REQUISIÇÕES INFINITAS	24
FIGURA 24 - EXECUÇÃO DO TESTE PARA 1 USUÁRIO COM REQUISIÇÕES INFINITAS PARA 10 USUÁRIOS.....	24
FIGURA 25 - EXECUÇÃO DO TESTE PARA 1 USUÁRIO COM REQUISIÇÕES INFINITAS PARA 20 USUÁRIOS.....	24
FIGURA 26- REQUISIÇÃO E LATÊNCIA PARA 25 USUÁRIOS	25
FIGURA 27 - REQUISIÇÃO E LATÊNCIA PARA 30 USUÁRIOS.....	25
FIGURA 28 - TABELA DE REQUISIÇÕES X LATÊNCIA COM USUÁRIOS FIXOS	26
FIGURA 29 - GRÁFICO DE REQUISIÇÕES X LATÊNCIA	26
FIGURA 30- ER	28
FIGURA 31 - MODELO LÓGICO	29



1. Visão geral da aplicação

A aplicação visará disponibilizar serviços de dados de forma tratada aos clientes que são fãs do universo de quadrinhos e filmes através da utilização da **API COMIC VINE** (<https://comicvine.gamespot.com/api/documentation>) que irá disponibilizar os dados estruturado no formato JSON.

O aplicativo servirá como um guia de consulta aos seus usuários para pesquisarem e visualizarem as informações que giram em torno de seus personagens e heróis favoritos do mundo dos quadrinhos.

As principais tabelas utilizadas da API para ser utilizada na aplicação serão as seguintes: editors, volumes, characters (personagens), super_power (super poder) e movies (filmes).

1.1. Estudo da API: COMIC VINE

A Seguir serão apresentados os principais atributos das tabelas da API: editors, volumes, characters, super_power e movies que serão utilizados no banco:

Tabela characters			
Nome na API	Tipo	Tamanho	Descrição
id	bigint	-	ID exclusivo do personagem.
name	varchar	255	Nome conhecido do personagem.
real_name	varchar	255	Nome real do personagem
birth	varchar	20	Uma data, se houver, em que o personagem nasceu.
description	text	-	Descrição do personagem.
count_of_issue_appearances	integer	-	Número de edições em que esse personagem aparece.

Tabela movies			
Nome na API	Tipo	Tamanho	Descrição
id	bigint		ID exclusivo do filme.
name	varchar	255	Nome do filme.
description	text		Descrição do filme
release_date	timestamp		Data do filme.
date_added	timestamp		Data em que o filme foi adicionado ao Comic
runtime	varchar	50	A duração deste filme.
rating	varchar	30	A classificação deste filme.
budget	varchar	255	O custo de fazer este filme.
total_revenue	varchar	255	Receita total gerada por este filme.



Tabela super_power			
Nome na API	Tipo	Tamanho	Descrição
id	bigint		ID exclusivo do poder.
name	varchar	255	Nome do poder.
date_added	timestamp		Data em que o poder foi adicionado ao Comic Vine.
description	text		Descrição do poder.

Tabela volumes			
Nome na API	Tipo	Tamanho	Descrição
id	bigint		ID exclusivo do volume.
name	varchar	255	Nome do volume.
count_of_issues	integer		Número de edições incluídas neste volume
start_year	varchar	255	Ano em que o volume foi adicionado ao Comic
description	text		Descrição do volume.
date_added	timestamp		Data em que o volume foi adicionado ao Comic

Tabela editors			
Nome na API	Tipo	Tamanho	Descrição
id	bigint		ID exclusivo do editor
name	varchar	255	Nome do Editor
birth	timestamp		Data de aniversario
description	text		Descrição da editora.
date_added	timestamp		Data em que o editor foi adicionado ao Comic
gender	smallint		Sexo
hometown	varchar	255	Cidade em que a editor nasceu
country	varchar	255	Pais do editor

1.2. Ferramentas

Será utilizado a linguagem de programação **Typescript** para implementar o **TypeORM** e as ferramentas: **NodeJs**, o SGBD **postgres**, o **github** como repositório e do **Trello** para auxiliar no gerenciamento deste projeto.

A evolução do projeto pode ser gerenciada pela equipe com permissão em:

➤ <https://trello.com/b/EOSTPQai/banco-de-dados-2>



2. Modelos

A seguir será apresentado o Modelo conceitual (MER), o modelo lógico (DER) e o físico do banco de dados.

2.1. ER

O Modelo Entidade Relacionamento (também chamado Modelo ER, ou simplesmente MER) descreve as (entidades) envolvidos em um domínio de negócio de quadrinhos, com suas características (atributos) e como elas se relacionam entre si (relacionamentos) conforme observada na API da comic vine. O Mer será apresentado no anexo I.

2.2. Modelo Lógico

Enquanto o MER é um modelo conceitual, o Diagrama Entidade Relacionamento (Diagrama ER ou ainda DER) é a sua representação gráfica e principal ferramenta para a criação do banco físico. O Der será apresentado no anexo II.

2.3. FÍSICO

A seguir será apresentado a construção do base de dados da aplicação:

```
CREATE TABLE characters (  
  id bigint PRIMARY KEY,  
  name varchar (255),  
  birth varchar (20),  
  num_edicoes_aparece integer,  
  description text,  
  real_name varchar (255)  
);
```

```
CREATE TABLE movies (  
  id bigint PRIMARY KEY,  
  name varchar (255),  
  budget varchar (255),  
  description text,  
  data_added timestamp,  
  rating varchar (30),  
  total_revenue varchar (255),  
  runtime varchar (50),  
  data_comic date
```



);

```
CREATE TABLE super_powers(  
  id bigint PRIMARY KEY,  
  name varchar (255),  
  data_added timestamp,  
  description text  
);
```

```
CREATE TABLE volumes (  
  id bigint PRIMARY KEY,  
  name varchar (255),  
  count_of_issues integer,  
  start_year varchar(255),  
  description text,  
  date_added timestamp  
);
```

```
CREATE TABLE editors (  
  id bigint PRIMARY KEY,  
  name varchar(255),  
  hometown varchar (255),  
  country varchar(255),  
  data_added timestamp,  
  birth timestamp,  
  gender smallint  
);
```

```
CREATE TABLE editors_volumes (  
  id bigint PRIMARY KEY,  
  editor_id bigint NOT NULL,  
  volume_id bigint NOT NULL,  
  FOREIGN KEY (editor_id) REFERENCES editors(id),  
  FOREIGN KEY (volume_id) REFERENCES volumes(id)  
);
```

```
CREATE TABLE editors_Characters (  
  id bigint PRIMARY KEY,  
  editor_id bigint NOT NULL,  
  character_id bigint NOT NULL,  
  FOREIGN KEY (character_id) REFERENCES characters(id),  
  FOREIGN KEY (editor_id) REFERENCES editors(id)  
);
```



```
CREATE TABLE movies_characters (  
  id bigint PRIMARY KEY,  
  movie_id bigint NOT NULL,  
  character_id bigint NOT NULL,  
  FOREIGN KEY (movie_id) REFERENCES movies(id),  
  FOREIGN KEY (character_id) REFERENCES characters(id)  
);
```

```
CREATE TABLE characters_powers (  
  id bigint PRIMARY KEY,  
  character_id bigint NOT NULL,  
  power_id bigint NOT NULL,  
  FOREIGN KEY (character_id) REFERENCES characters(id),  
  FOREIGN KEY (power_id) REFERENCES super_powers(id)  
);
```

3. Roles

Os usuários que utilizarão o sistema farão parte de 5 grupos principais: sendo o primeiro o Administrador do banco e os demais serão os clientes da aplicação que desejam adquirir os serviços e informações acerca dos dados de filmes e histórias em quadrinhos dos seus personagens preferidos. Cada usuário cliente irá ter tipos de permissões diferentes nas tabelas e por conseguinte nos dados. A seguir será especificado cada grupo com as permissões a serem concedidas para uso dos dados das tabelas e do banco.

3.1. Role Administrador

O administrador será responsável pelo gerenciamento dos grupos (futura tabela de usuários) e pela carga de dados na API, esse usuário possuirá todos os privilégios de todas as tabelas do schema.

Assim temos:

```
CREATE ROLE adm;  
GRANT USAGE ON SCHEMA comicvine To adm;  
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA comicvine TO adm;
```

3.2. Roles dos Clientes

Os usuários que serão os clientes da aplicação possuirão acesso limitado aos recursos do banco de dados, sendo possível **apenas permissão de leitura nas tabelas** (visualização)



do schema. A seguir será especificado as leituras específicas das tabelas de cada tipo de cliente.

3.2.1. Role Geeks

Os usuários pertencentes ao grupo “Geeks” possuirão acesso limitado aos recursos do banco de dados, sendo possível **apenas permissão de leitura** somente nas tabelas referentes as histórias em quadrinho dos seus personagens favoritos, assim será permitida as visualizações dos dados das seguintes tabelas:

- **Volume, editor, personagem e super poder.**
- Não possuirão nenhum outro privilégio e nem acesso as demais tabelas do banco.

Assim temos:

```
CREATE ROLE geeks;  
GRANT USAGE ON SCHEMA comicvine TO geeks;  
GRANT SELECT ON TABLE comicvine.volume TO geeks;  
GRANT SELECT ON TABLE comicvine.editor TO geeks;  
GRANT SELECT ON TABLE comicvine.personagem TO geeks;  
GRANT SELECT ON TABLE comicvine.superpoder TO geeks;
```

3.2.2. Role Filmes

Os usuários pertencentes ao grupo “Pacote_Filmes” possuirão acesso limitado aos recursos do banco de dados, sendo possível **apenas permissão de leitura** somente nas tabelas referentes aos filmes dos seus personagens favoritos, assim será permitida as visualizações dos dados das seguintes tabelas:

- **Personagem, super poder e Filme.**
- Não possuirão nenhum outro privilégio no banco e **não terá direito de leitura nas tabelas: Volume e editor.**

Assim temos:

```
CREATE ROLE filmes;  
GRANT USAGE ON SCHEMA comicvine TO filmes;  
GRANT SELECT ON TABLE comicvine.personagem TO filmes;  
GRANT SELECT ON TABLE comicvine.superpoder TO filmes;  
GRANT SELECT ON TABLE comicvine.filme TO filmes;
```



3.2.3. Role Premium

Os usuários pertencentes ao grupo “Premium” **terão o direito de leitura em todas as tabelas**, tendo assim o somatório dos direitos atribuídos ao grupo “Geeks e Pacote_Filmes”. Assim será permitida as visualizações dos dados das seguintes tabelas:

- **Volume, editor, personagem, super poder e Filmes.**
- Não possuirão nenhum outro privilégio no banco e **não terão direito de escrita e atualização em nenhuma tabela.**

Assim temos:

```
CREATE ROLE premium;  
GRANT USAGE ON SCHEMA comicvine To premium;  
GRANT SELECT ON TABLE comicvine.volume To premium;  
GRANT SELECT ON TABLE comicvine.editor To premium;  
GRANT SELECT ON TABLE comicvine.personagem To premium;  
GRANT SELECT ON TABLE comicvine.superpoder To premium;  
GRANT SELECT ON TABLE comicvine.filme To premium;
```

4. Usuários

Para exemplificar iremos criar 4 usuários, uma para cada Role criada anteriormente:

```
CREATE USER useradm WITH PASSWORD 'admin';  
CREATE USER usergeeks WITH PASSWORD 'geeks';  
CREATE USER userfilmes WITH PASSWORD 'filmes';  
CREATE USER userpremium WITH PASSWORD 'premium';
```

Atribuindo os poderes aos usuários criados conforme os grupos:

```
GRANT adm TO useradm;  
GRANT geeks TO usergeeks;  
GRANT filmes TO userfilmes;  
GRANT premium TO userpremium;
```



4.1. Direitos

Verificando os poderes cedidos aos grupos do banco:

```
SELECT rolname, oid, rolsuper, rolinherit, rolcreatorole, rolcreatedb, rolcanlogin
FROM pg_roles
where rolname = 'adm' or rolname = 'geeks' or rolname = 'filmes' or rolname = 'premium';
```

Data Output Messages Notifications								
	rolname name	oid oid	rolsuper boolean	rolinherit boolean	rolcreatorole boolean	rolcreatedb boolean	rolcanlogin boolean	
1	adm	49881	false	true	false	false	false	
2	geeks	49882	false	true	false	false	false	
3	filmes	49883	false	true	false	false	false	
4	premium	49884	false	true	false	false	false	

Figura 1- Direitos dos Grupos

Verificando os poderes cedidos ao grupo adm:

```
SELECT table_name, privilege_type, is_grantable, with_hierarchy
FROM information_schema.role_table_grants
where grantee='adm';
```

Data Output Messages Notifications				
	table_name name	privilege_type character varying	is_grantable character varying (3)	with_hierarchy character varying (3)
1	producao	INSERT	NO	NO
2	producao	SELECT	NO	YES
3	producao	UPDATE	NO	NO
4	producao	DELETE	NO	NO
5	producao	TRUNCATE	NO	NO
6	producao	REFERENCES	NO	NO
7	producao	TRIGGER	NO	NO
8	personagem	INSERT	NO	NO
9	personagem	SELECT	NO	YES
10	personagem	UPDATE	NO	NO
11	personagem	DELETE	NO	NO
12	personagem	TRUNCATE	NO	NO
13	personagem	REFERENCES	NO	NO
14	personagem	TRIGGER	NO	NO
15	superpoder	INSERT	NO	NO

Figura 2 - Parte dos direitos do Grupo Adm



Verificando os poderes cedidos ao grupo geeks:

```
SELECT table_name, privilege_type, is_grantable, with_hierarchy
FROM information_schema.role_table_grants
where grantee = 'geeks';
```

Data Output Messages Notifications				
	table_name name	privilege_type character varying	is_grantable character varying (3)	with_hierarchy character varying (3)
1	personagem	SELECT	NO	YES
2	superpoder	SELECT	NO	YES
3	volume	SELECT	NO	YES
4	editor	SELECT	NO	YES

Figura 3 – Direitos do Grupo geeks

Verificando os poderes cedidos ao grupo filmes:

```
SELECT table_name, privilege_type, is_grantable, with_hierarchy
FROM information_schema.role_table_grants
where grantee='filmes';
```

Data Output Messages Notifications				
	table_name name	privilege_type character varying	is_grantable character varying (3)	with_hierarchy character varying (3)
1	personagem	SELECT	NO	YES
2	superpoder	SELECT	NO	YES
3	filme	SELECT	NO	YES

Figura 4 - Direitos do Grupo Filmes

Verificando os poderes cedidos ao grupo premium:

```
SELECT table_name, privilege_type, is_grantable, with_hierarchy
FROM information_schema.role_table_grants
where grantee='premium';
```



Data Output Messages Notifications				
	table_name name	privilege_type character varying	is_grantable character varying (3)	with_hierarchy character varying (3)
1	volume	SELECT	NO	YES
2	editor	SELECT	NO	YES
3	personagem	SELECT	NO	YES
4	superpoder	SELECT	NO	YES
5	filme	SELECT	NO	YES

Figura 5- Direitos do Grupo Premium

Verificando os poderes cedidos aos usuários do banco:

```
SELECT rolname, oid, rolsuper, rolinherit, rolcreaterole, rolcreatedb, rolcanlogin
```

```
FROM pg_roles
```

```
WHERE rolname = 'useradm' or rolname = 'usergeeks' or rolname = 'userfilmes' or rolname = 'userpremium';
```

Data Output Messages Notifications							
	rolname name	oid oid	rolsuper boolean	rolinherit boolean	rolcreaterole boolean	rolcreatedb boolean	rolcanlogin boolean
1	useradm	49885	false	true	false	false	true
2	usergeeks	49886	false	true	false	false	true
3	userfilmes	49887	false	true	false	false	true
4	userpremium	49888	false	true	false	false	true

Figura 6 - Direitos dos Usuários

5. Print com o count das tabelas.

A seguir será apresentado os números de registros de cada uma das principais tabelas:

```
SELECT COUNT(id) as "QTD de Personagens" FROM characters;
```



```
1 SELECT COUNT(id) as "Qtd de Personagens" FROM characters;
```

Data Output		Messages	Notifications
<div>Icons: list, copy, paste, delete, save, download, refresh</div>			
	Qtd de Personagens		
	bigint		
1	155407		

Figura 7 - Quantidade de Personagens

SELECT COUNT(id) as "QTD de Editores" FROM editors;

```
1 SELECT COUNT(id) as "QTD de Editores" FROM editors;
```

Data Output		Messages	Notifications
<div>Icons: list, copy, paste, delete, save, download, refresh</div>			
	QTD de Editores		
	bigint		
1	76469		

Figura 8 - Quantidade de Editores

SELECT COUNT(id) as "QTD de Filmes" FROM movies;

```
1 SELECT COUNT(id) as "QTD de Filmes" FROM movies;
```

Data Output		Messages	Notifications
<div>Icons: list, copy, paste, delete, save, download, refresh</div>			
	QTD de Filmes		
	bigint		
1	2640		

Figura 9 - Quantidade De Filmes

SELECT COUNT(id) as "QTD de Poderes" FROM super_powers;

```
1 SELECT COUNT(id) as "QTD de Poderes" FROM super_powers;
```

Data Output		Messages	Notifications
<div>Icons: list, copy, paste, delete, save, download, refresh</div>			
	QTD de Poderes		
	bigint		
1	128		

Figura 10 - Quantidade de Super Poderes



SELECT COUNT(id) as "QTD de Histórias em quadrinhos" FROM volumes;

	QTD de Histórias em quadrinhos
1	134916

Figura 11- Quantidade de histórias em quadrinhos

6. Definição de índices e justificativas.

Para facilitar as consultas do banco e melhorar o desempenho, além dos índices primários (id de cada tabela) será necessário a criação de índices em campos que são frequentemente usados em cláusulas de pesquisa ou nas junções de tabelas.

Cada uma das tabelas N: N possui um atributo "id" exclusivo e o PostgreSQL cria automaticamente um índice primário para cada um desses atributos. Esses índices permitem que o PostgreSQL acesse e relacione os dados de forma eficiente. Ao realizar consultas que envolvem junções entre essas tabelas, os índices correspondentes oferecem suporte às operações de junção, resultando em um desempenho aprimorado. Esses índices primários são fundamentais para otimizar o processo de junção, pois eles permitem que o banco de dados localize rapidamente os registros relacionados com base nos valores dos atributos "id". Com os índices, as consultas que envolvem junções entre as tabelas N: N se beneficiam de um tempo de resposta mais rápido e uma execução mais eficiente das operações de junção.

Além disso, no banco de dados, para o negócio deste projeto foram criados índices adicionais para atender às consultas mais frequentes e melhorar o desempenho em determinados atributos. A seguir, será apresentado os índices criados em cada uma das tabelas do banco:

6.1. Índices da Tabela Characters

Devido ao negócio da api serão realizadas frequentes consultas por nome de personagem, então criar um índice nesse campo melhora o desempenho das consultas, conforme analisado pelo EXPLAIN ANALYZE a seguir:

```
EXPLAIN ANALYZE SELECT * FROM characters WHERE NAME ='SUPERMAN';
```




```
18 EXPLAIN ANALYZE
19 SELECT *
20 FROM characters
21 WHERE NAME = 'SUPERMAN';
```

Data Output	Messages	Notifications
<div>QUERY PLAN text</div>		
1	Gather (cost=1000.00..6299.61 rows=2 width=658) (actual time=1828.985..1854.238 rows=0 loops=1)	
2	Workers Planned: 2	
3	Workers Launched: 2	
4	-> Parallel Seq Scan on characters (cost=0.00..5299.41 rows=1 width=658) (actual time=1615.654..1615.654 rows=0 loop...	
5	Filter: ((name)::text = 'SUPERMAN'::text)	
6	Rows Removed by Filter: 51802	
7	Planning Time: 16.643 ms	
8	Execution Time: 1854.262 ms	

Figura 12 - Análise do atributo name da tabela personagem

Com a criação do índice name, temos:

```
CREATE INDEX idx_characters_name ON characters (name);
```

```
1 -- Índice na tabela "characters"
2 CREATE INDEX idx_characters_name ON characters (name);
3 EXPLAIN ANALYZE SELECT * FROM characters WHERE NAME = 'SUPERMAN';
```

Data Output	Messages	Notifications
<div>QUERY PLAN text</div>		
1	Index Scan using idx_characters_name on characters (cost=0.42..12.45 rows=2 width=658) (actual time=0.018..0.019 rows=0 loops=1)	
2	Index Cond: ((name)::text = 'SUPERMAN'::text)	
3	Planning Time: 0.091 ms	
4	Execution Time: 0.031 ms	

Figura 13 - Análise do índice name de personagem

6.2. Índices da Tabela movies

Devido ao negócio da api serão realizadas frequentes consultas por nome de filme, criar um índice nesse campo melhorar o desempenho das consultas, conforme analisado pelo EXPLAIN ANALYZE a seguir:

```
EXPLAIN ANALYZE SELECT * FROM movies WHERE NAME = 'Superman';
```




```
19 EXPLAIN ANALYZE SELECT * FROM movies WHERE NAME = 'SUPERMAN';
```

	Data Output	Messages	Notifications
	<div>QUERY PLAN</div> <div>text</div>		
1	Seq Scan on movies (cost=0.00..121.00 rows=1 width=475) (actual time=0.753..0.753 rows=0 loops=...		
2	Filter: ((name)::text = 'SUPERMAN'::text)		
3	Rows Removed by Filter: 2640		
4	Planning Time: 13.855 ms		
5	Execution Time: 0.776 ms		

Figura 14 - Análise do atributo name da tabela filmes

Com a criação do índice name, temos:

```
CREATE INDEX idx_movies_name ON movies (name);
```

```
5 -- índice na tabela "movies"
6 CREATE INDEX idx_movies_name ON movies (name);
7 EXPLAIN ANALYZE SELECT * FROM movies WHERE NAME = 'SUPERMAN';
```

	Data Output	Messages	Notifications
	<div>QUERY PLAN</div> <div>text</div>		
1	Index Scan using idx_movies_name on movies (cost=0.28..8.30 rows=1 width=475) (actual time=0.017..0.017 rows=0 loop=...		
2	Index Cond: ((name)::text = 'SUPERMAN'::text)		
3	Planning Time: 0.089 ms		
4	Execution Time: 0.029 ms		

Figura 15 - Análise do índice name de filmes

6.3. Índices da Tabela volumes

Devido ao negócio da api serão realizadas frequentes consultas por nome de volume, criar um índice nesse campo melhora o desempenho das consultas, conforme analisado pelo EXPLAIN ANALYZE a seguir:

```
EXPLAIN ANALYZE SELECT * FROM volumes WHERE NAME = 'Superman';
```



```
11 EXPLAIN ANALYZE SELECT * FROM volumes WHERE NAME = 'SUPERMAN';
```

	Data Output	Messages	Notifications
	<div>QUERY PLAN</div> <div>text</div>		
1	Seq Scan on volumes (cost=0.00..5549.45 rows=1 width=237) (actual time=1379.758..1379.759 rows=0 loops=1)		
2	Filter: ((name)::text = 'SUPERMAN'::text)		
3	Rows Removed by Filter: 134916		
4	Planning Time: 53.346 ms		
5	Execution Time: 1379.774 ms		

Figura 16 - Análise do atributo name da tabela volumes

Com a criação do índice name, temos:

```
CREATE INDEX idx_volumes_name ON volumes (name);
```

```
9 -- Índices na tabela "volumes"
```

```
10 CREATE INDEX idx_volumes_name ON volumes (name);
```

```
11 EXPLAIN ANALYZE SELECT * FROM volumes WHERE NAME = 'Superman';
```

	Data Output	Messages	Notifications
	<div>QUERY PLAN</div> <div>text</div>		
1	Bitmap Heap Scan on volumes (cost=4.70..138.72 rows=36 width=237) (actual time=0.026..0.046 rows=34 loops=1)		
2	Recheck Cond: ((name)::text = 'Superman'::text)		
3	Heap Blocks: exact=28		
4	-> Bitmap Index Scan on idx_volumes_name (cost=0.00..4.69 rows=36 width=0) (actual time=0.021..0.021 rows=34 loops=1)		
5	Index Cond: ((name)::text = 'Superman'::text)		
6	Planning Time: 0.084 ms		
7	Execution Time: 0.065 ms		

Figura 17 - Análise do índice name das histórias em quadrinhos

6.4. Índices da Tabela editors

Devido ao negócio da api serão realizadas frequentes consultas por nome de editor, criar um índice nesse campo melhora o desempenho das consultas, conforme analisado pelo EXPLAIN ANALYZE a seguir:

```
EXPLAIN ANALYZE SELECT * FROM editors, movies WHERE movies.name = 'Superman';
```



```
13 -- Índice na tabela "editores"
14 CREATE INDEX idx_editors_name ON editors (name);
15 EXPLAIN ANALYZE SELECT * FROM editors WHERE name = 'Jae Lee' ;
```

Data Output Messages Notifications

QUERY PLAN

1	Seq Scan on editors (cost=0.00..1933.86 rows=1 width=498) (actual time=0.144..17.996 rows=1 loops...
2	Filter: ((name)::text = 'Jae Lee':text)
3	Rows Removed by Filter: 76468
4	Planning Time: 0.902 ms
5	Execution Time: 18.023 ms

Figura 18 - Análise do atributo name da tabela editores

Com a criação do índice name, temos:

CREATE INDEX idx_editors_name ON editors (name);

```
13 -- Índice na tabela "editores"
14 CREATE INDEX idx_editors_name ON editors (name);
15 EXPLAIN ANALYZE SELECT * FROM editors WHERE name = 'Jae Lee' ;
```

Data Output Messages Notifications

QUERY PLAN

1	Index Scan using idx_editors_name on editors (cost=0.42..8.44 rows=1 width=498) (actual time=0.021..0.021 rows=1 loop...
2	Index Cond: ((name)::text = 'Jae Lee':text)
3	Planning Time: 0.064 ms
4	Execution Time: 0.032 ms

Figura 19 - Análise do índice name dos editores

Os índices secundários criados para o banco de dados deste projeto estão todos representados na figura abaixo:

SELECT indexname, indexdef, tablename FROM pg_indexes WHERE schemaname = 'public'

```
1 SELECT indexname, indexdef, tablename FROM pg_indexes WHERE schemaname = 'public'
```

Data Output Messages Notifications

	indexname	indexdef	tablename
	name	text	name
1	idx_volumes_name	CREATE INDEX idx_volumes_name ON public.volumes USING btree (name)	volumes
2	idx_movies_name	CREATE INDEX idx_movies_name ON public.movies USING btree (name)	movies
3	idx_editors_name	CREATE INDEX idx_editors_name ON public.editors USING btree (name)	editors
4	idx_characters_name	CREATE INDEX idx_characters_name ON public.characters USING btree (na...	characters

Figura 20 - Índices Secundários do Banco



7. Teste JMeter

Os testes do Jmeter são importantes para avaliar o desempenho e a capacidade de um banco de dados em lidar com a carga de requisições e usuários. Os testes são fundamentais para entender o comportamento do banco de dados sob diferentes cargas e identificar problemas de desempenho antes que eles afetem os usuários finais. Eles ajudam a garantir que a infraestrutura seja dimensionada adequadamente e que os serviços sejam capazes de lidar com a carga esperada, proporcionando uma experiência satisfatória aos usuários e evitando interrupções indesejadas. Aqui estão alguns motivos pelos quais esses testes são importantes:

- **Identificação de limites de capacidade:** Determinar o ponto em que o desempenho do banco de dados começa a degradar sob uma carga crescente de requisições ou usuários, evitando sobrecargas e garantindo tempos de resposta adequados.
- **Otimização da configuração e infraestrutura:** Identificar gargalos no banco de dados ou na infraestrutura subjacente, permitindo ajustes de configuração, otimização de consultas e consideração de atualizações de hardware.
- **Dimensionamento da infraestrutura:** Determinar quantos usuários simultâneos o banco de dados pode suportar antes que o desempenho seja comprometido, auxiliando no dimensionamento correto da infraestrutura.
- **Prevenção de problemas de desempenho:** Identificar problemas potenciais antes que ocorram em ambiente de produção, melhorando a experiência do usuário e evitando interrupções não planejadas nos serviços.
- **Validação de atualizações e alterações:** Realizar testes de latência antes e depois de alterações significativas no banco de dados para avaliar o impacto dessas mudanças no desempenho do sistema.

Esses testes são fundamentais para entender e aprimorar o desempenho do banco de dados, garantir a escalabilidade adequada, prevenir problemas de desempenho e validar alterações e atualizações.

7.1. Realização dos testes no JMeter

Os Testes do JMeter foram realizados nas seguintes condições descritas a seguir:

- **Arquivo .Conf:** O arquivo "postgresql.conf" foi modificado para aumentar o valor de "max_connections" de 100 para 100 mil.
- **Ambiente de teste:** Os testes foram realizados no ambiente do Sistema Operacional Windows 10 Education de 64 bits, versão 22H2. O computador utilizado para os testes possui um processador Intel Core i7-7700 CPU com velocidade de 3.60 GHz e 8,00 GB de RAM.



- **Nº de execuções:** Executou-se o teste 30 repetições para averiguar a média de latência.
- **A consulta:** Escolheu para requisição dos testes a consulta a seguir:

SELECT

```
Mo.name AS NOME_FILME,  
Mo.rating AS Classificação,  
Ch.name AS NOME_PERSONAGEM,  
Mo.budget AS TOTAL_GASTO,  
Mo.total_revenue AS TOTAL_ARRECADADO,  
STRING_AGG(Sp.name, ', ') AS PODERES_PERSONAGEM
```

FROM

```
movies AS Mo  
INNER JOIN movies_characters AS Mc ON Mo.id = Mc.movie_id  
INNER JOIN characters AS Ch ON Mc.character_id = Ch.id  
INNER JOIN characters_powers AS Cp ON Ch.id = Cp.character_id  
INNER JOIN super_powers AS Sp ON Cp.power_id = Sp.id
```

WHERE

```
Mo.release_date < '2020-11-23'  
AND Ch.id IN (  
    SELECT ChInner.id  
    FROM characters AS ChInner  
    INNER JOIN characters_powers AS CpInner ON ChInner.id = CpInner.character_id  
    INNER JOIN super_powers AS SplInner ON CpInner.power_id = SplInner.id  
    GROUP BY ChInner.id  
    HAVING COUNT(SplInner.id) > 2  
)
```

GROUP BY

```
Mo.name,  
Mo.rating,  
Ch.name,  
Mo.budget,  
Mo.total_revenue  
order by Mo.name;
```

A consulta selecionada recupera informações relevantes da tabela "Movies", como o nome (NOME_FILME), rating (Classificação), o budget (TOTAL_GASTO) e total_revenue (TOTAL_ARRECADADO). Além disso, obtém o nome do personagem (PERSONAGEM) da tabela "characters" e o nome do super poder (PODER_PERSONAGEM) da tabela "super_power" para os filmes produzidos antes de 23/11/2020 e cujos personagens possuem mais de 2 poderes. O resultado final da consulta é ordenado em ordem alfabética pelo nome do filme.

Com o uso da ferramenta EXPLAIN ANALYZE percebe-se que a consulta acima é custosa para postgres e gasta **100.12 ms** para sua execução.



"QUERY PLAN"

```
"GroupAggregate (cost=3276.48..3291.15 rows=489 width=88) (actual time=65.653..69.789 rows=816 loops=1)"
" Group Key: mo.name, mo.rating, ch.name, mo.budget, mo.total_revenue"
" -> Sort (cost=3276.48..3277.70 rows=489 width=70) (actual time=65.636..66.108 rows=13449 loops=1)"
"   Sort Key: mo.name, mo.rating, ch.name, mo.budget, mo.total_revenue"
"   Sort Method: quicksort Memory: 1756kB"
"   -> Hash Join (cost=2907.95..3254.64 rows=489 width=70) (actual time=21.543..31.946 rows=13449 loops=1)"
"     Hash Cond: (cp.power_id = sp.id)"
"     -> Hash Join (cost=2903.07..3248.43 rows=489 width=64) (actual time=21.430..29.888 rows=13449 loops=1)"
"       Hash Cond: (mc.movie_id = mo.id)"
"       -> Hash Join (cost=2763.88..3106.91 rows=888 width=27) (actual time=19.282..24.944 rows=27509 loops=1)"
"         Hash Cond: (mc.character_id = ch.id)"
"         -> Seq Scan on movies_characters mc (cost=0.00..278.86 rows=16986 width=16) (actual time=0.021..0.786 rows=16986 loops=1)"
"           -> Hash (cost=2758.48..2758.48 rows=432 width=43) (actual time=19.237..19.241 rows=13432 loops=1)"
"             Buckets: 16384 (originally 1024) Batches: 1 (originally 1) Memory Usage: 1147kB"
"             -> Merge Join (cost=2467.36..2758.48 rows=432 width=43) (actual time=7.390..16.850 rows=13432 loops=1)"
"               Merge Cond: (ch.id = cp.character_id)"
"               -> Merge Join (cost=1255.78..10876.00 rows=4730 width=27) (actual time=4.865..12.225 rows=1331 loops=1)"
"                 Merge Cond: (ch.id = chinner.id)"
"                 -> Index Scan using ""PK_9d731e05758f26b9315dac5e378"" on characters ch (cost=0.42..8565.36 rows=155407 width=19) (actual time=0.039..2.716 rows=3465 loops=1)"
"                   -> GroupAggregate (cost=1255.36..1815.69 rows=4730 width=8) (actual time=4.824..9.082 rows=1331 loops=1)"
"                     Group Key: chinner.id"
"                     Filter: (count(spinner.id) > 2)"
"                     Rows Removed by Filter: 526"
"                     -> Merge Join (cost=1255.36..1567.35 rows=14191 width=16) (actual time=4.812..7.913 rows=14191 loops=1)"
"                       Merge Cond: (cpinner.character_id = chinner.id)"
"                       -> Sort (cost=1254.94..1290.41 rows=14191 width=16) (actual time=4.788..5.266 rows=14191 loops=1)"
"                         Sort Key: cpinner.character_id"
"                         Sort Method: quicksort Memory: 1161kB"
"                         -> Hash Join (cost=4.88..276.28 rows=14191 width=16) (actual time=0.065..3.086 rows=14191 loops=1)"
"                           Hash Cond: (cpinner.power_id = spinner.id)"
"                           -> Seq Scan on characters_powers cpinner (cost=0.00..232.91 rows=14191 width=16) (actual time=0.018..0.699 rows=14191 loops=1)"
"                             -> Hash (cost=3.28..3.28 rows=128 width=8) (actual time=0.035..0.035 rows=128 loops=1)"
"                               Buckets: 1024 Batches: 1 Memory Usage: 13kB"
"                             -> Seq Scan on super_powers spinner (cost=0.00..3.28 rows=128 width=8) (actual time=0.008..0.017 rows=128 loops=1)"
"                               -> Index Only Scan using ""PK_9d731e05758f26b9315dac5e378"" on characters chinner (cost=0.42..4047.52 rows=155407 width=8) (actual time=0.021..0.950 rows=3464 loops=1)"
"                                 Heap Fetches: 0"
"                               -> Sort (cost=1211.57..1247.05 rows=14191 width=16) (actual time=2.522..3.037 rows=14191 loops=1)"
"                                 Sort Key: cp.character_id"
"                                 Sort Method: quicksort Memory: 1161kB"
"                               -> Seq Scan on characters_powers cp (cost=0.00..232.91 rows=14191 width=16) (actual time=0.015..1.079 rows=14191 loops=1)"
"                                 -> Hash (cost=121.00..121.00 rows=1455 width=53) (actual time=2.093..2.093 rows=1464 loops=1)"
"                                   Buckets: 2048 Batches: 1 Memory Usage: 122kB"
"                                   -> Seq Scan on movies mo (cost=0.00..121.00 rows=1455 width=53) (actual time=0.027..1.672 rows=1464 loops=1)"
"                                     Filter: (release_date < '2020-11-23 00:00:00'::timestamp without time zone)"
"                                     Rows Removed by Filter: 1176"
"                                   -> Hash (cost=3.28..3.28 rows=128 width=22) (actual time=0.100..0.100 rows=128 loops=1)"
"                                     Buckets: 1024 Batches: 1 Memory Usage: 16kB"
"                                     -> Seq Scan on super_powers sp (cost=0.00..3.28 rows=128 width=22) (actual time=0.038..0.078 rows=128 loops=1)"
"Planning Time: 18.699 ms"
"Execution Time: 100.12 ms"
```

7.1.1. Teste para número de usuários

O teste no JMeter para averiguar o número máximo de usuário foi realizado da seguinte maneira:

Definiu-se uma quantidade fixa de requisições **igual a 1. (Interação igual a 1)** e aumentou gradualmente a quantidade de usuários de 10 em 10 até o teste retornar erro. Ao retornar erro (negação do serviço ao usuário) o teste foi regredindo de 1 em 1 até estabelecer o número de máximo de usuários.



O número máximo de usuários suportados pelo banco: Foi averiguado que de 1 até 97 usuários as requisições são realizadas perfeitamente sem erros, mas a partir de 98 usuários as começam a serem negadas, evidenciando que o número máximo de usuários suportados é **de 97 usuários**, conforme as condições definidas anteriormente.

Por fim, extraiu-se o valor da latência média de cada repetição e gerou-se o seguinte gráfico com os valores de latência em relação ao número de usuários:

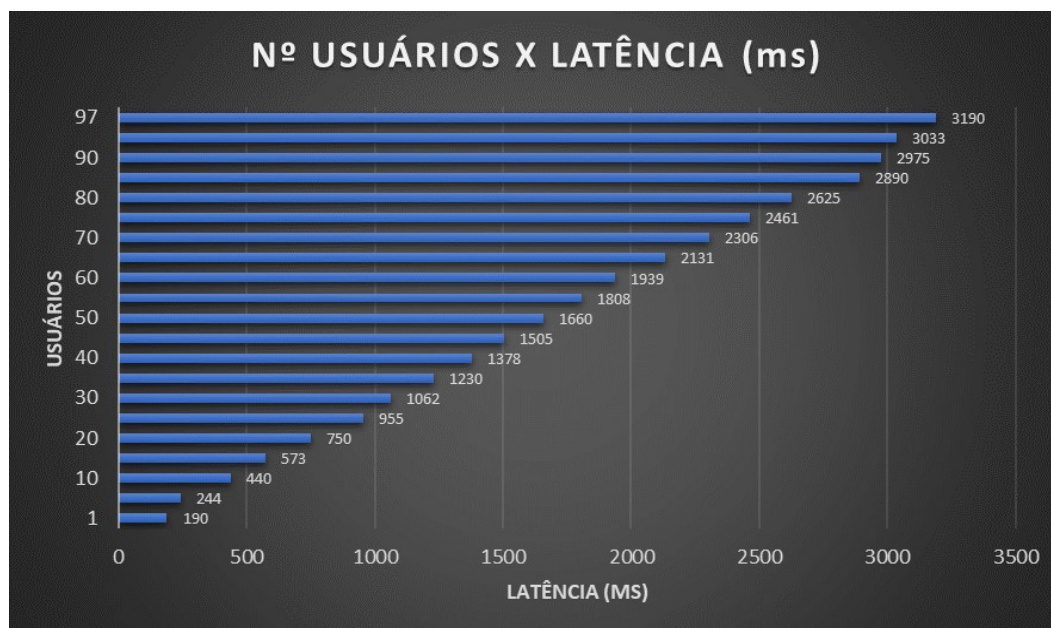


Figura 21 - Gráfico de usuário x Latência

7.1.2. Teste para número de requisições

O teste no JMeter para averiguar o número máximo de requisições foi realizado inicialmente da seguinte maneira:

Manteve constante a quantidade de usuários (threads) **como 1** e aumentou-se gradualmente a quantidade de requisições de 10 em 10 até que o teste retornasse erro. Como não houve erro na centena, optou-se por aumentar a requisição de 100 e 100 e mesmo assim, ainda, não se obteve erro. Então optou-se por deixar o número de requisições infinitas para um único usuário conforme é possível observar na figura a seguir:

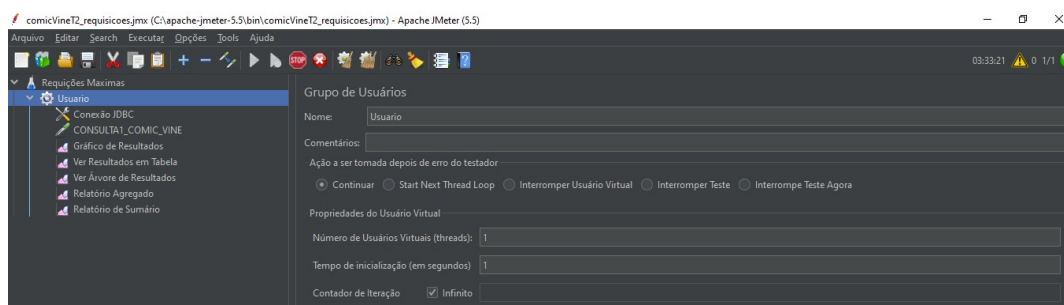
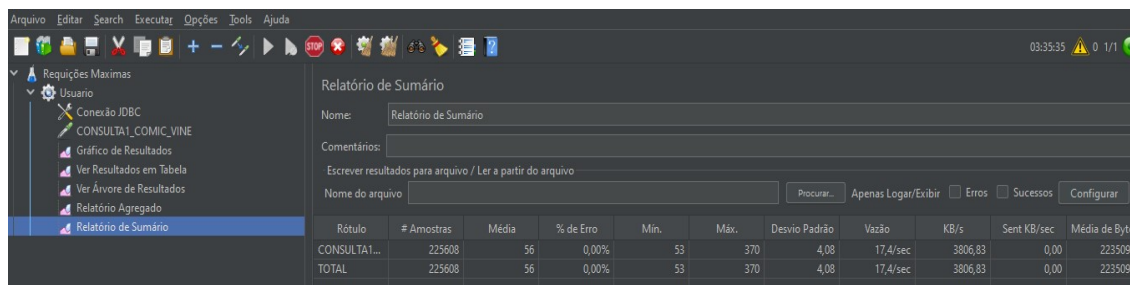


Figura 22- Configuração do teste para requisições infinitas



A latência das variações para 1 usuário com o número de interações (requisições) infinitas foi de **aproximadamente 56 ms**. O teste ficou executando mais de 3 horas mais de 200 mil requisições sem parar ou apresentar erros, conforme pode ser observado na figura a seguir:



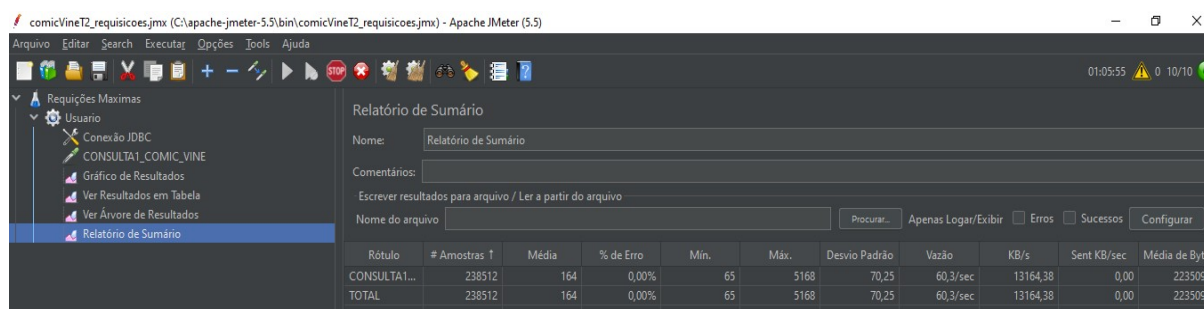
Rótulo	# Amostras	Média	% de Erro	Min.	Máx.	Desvio Padrão	Vazão	KB/s	Sent KB/sec	Média de Bytes
CONSULTA1...	225608	56	0,00%	53	370	4,08	17,4/sec	3806,83	0,00	223509,0
TOTAL	225608	56	0,00%	53	370	4,08	17,4/sec	3806,83	0,00	223509,0

Figura 23 - Execução do teste para 1 usuário com requisições infinitas

Então optou-se por testar o número de requisições com um número fixo diferente de usuários, inicialmente com 10, 20, 25, 30, 40 e 50 usuários.

Com 10 e 20 usuários fixos, assim como, como observado para apenas um único usuário, também não houve erros nas requisições “infinitas”.

A latência média observada para 10 usuários foi de aproximadamente **164 ms** como pode ser observado na figura a seguir:



Rótulo	# Amostras	Média	% de Erro	Min.	Máx.	Desvio Padrão	Vazão	KB/s	Sent KB/sec	Média de Bytes
CONSULTA1...	238512	164	0,00%	65	5168	70,25	60,3/sec	13164,38	0,00	223509,0
TOTAL	238512	164	0,00%	65	5168	70,25	60,3/sec	13164,38	0,00	223509,0

Figura 24 - Execução do teste para 1 usuário com requisições infinitas para 10 usuários

A latência média observada para **20 usuários** de aproximadamente **360 ms** como pode ser observado na figura a seguir:



Rótulo	# Amostras	Média	% de Erro	Min.	Máx.	Desvio Padrão	Vazão	KB/s	Sent KB/sec	Média de Bytes
CONSULTA1...	305491	360	0,00%	111	5720	216,26	55,2/sec	12059,37	0,00	223509,0
TOTAL	305491	360	0,00%	111	5720	216,26	55,2/sec	12059,37	0,00	223509,0

Figura 25 - Execução do teste para 1 usuário com requisições infinitas para 20 usuários



Para **25, 30, 40 e 50** usuários fixos as requisições dos usuários passaram a serem realizadas de 5 em 5 e caso não atendessem as requisições, a retornar erro (negação do serviço ao usuário) o teste foi regredindo de 1 em 1 até estabelecer o número de máximo de requisições para o nº fixo de usuários estabelecidos.

Para 25 usuários fixos a latência média total foi de aproximadamente **352 ms**, porém a partir de 28 requisições não é possível atender a todos os 25 usuários. Obtivemos o seguinte gráfico quanto requisições X latência:

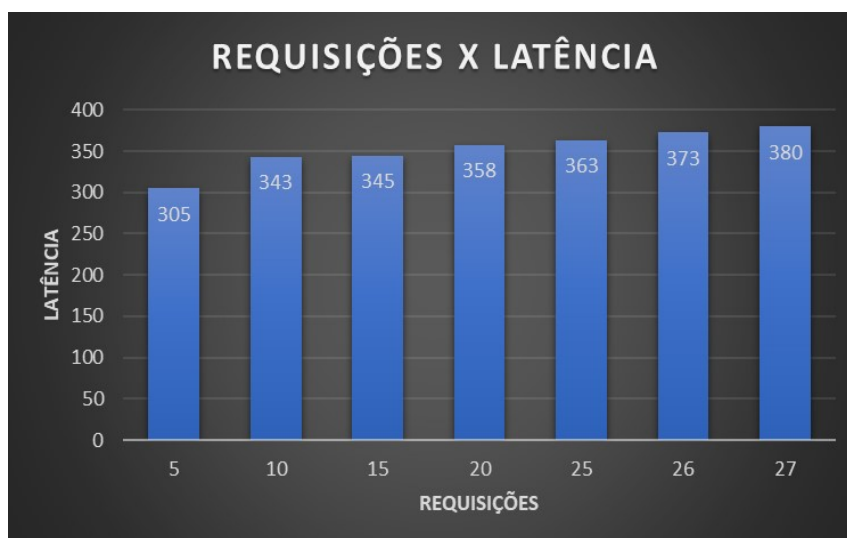


Figura 26- Requisição e latência para 25 usuários

Para 30 usuários fixos a latência média total foi de aproximadamente **456 ms**, porém a partir de 24 requisições não é possível atender a todos os 30 usuários. Obtivemos o seguinte gráfico quanto requisições x latência:



Figura 27 - Requisição e latência para 30 usuários



O teste com 40 usuários fixos a latência média foi de aproximadamente **704 ms**. A partir de 14 requisições não é possível atender a todos os 40 usuários.

Por fim, o teste com 50 usuários fixos a latência média foi de aproximadamente **1491 ms**. A partir de 6 requisições não é possível atender a todos os 50 usuários.

Assim obtermos os seguintes dados dos testes de requisições:

Números fixo de usuários	Número máximo de requisições	Latência Média (ms)
1	Infinita (+225 mil)	56
10	Infinita (+225 mil)	164
20	Infinita (+225 mil)	360
25	27	352
30	23	456
40	13	704
50	5	1491

Figura 28 - Tabela de requisições X latência com usuários fixos

Como para os testes de 1, 10 e 20 usuários fixos foram realizados com requisições infinitas o total de requisições de fato realizadas corresponderam no mínimo um total de 225 mil requisições em todas as repetições dos testes. Então assim é possível obter o seguinte gráfico de requisições X latência:

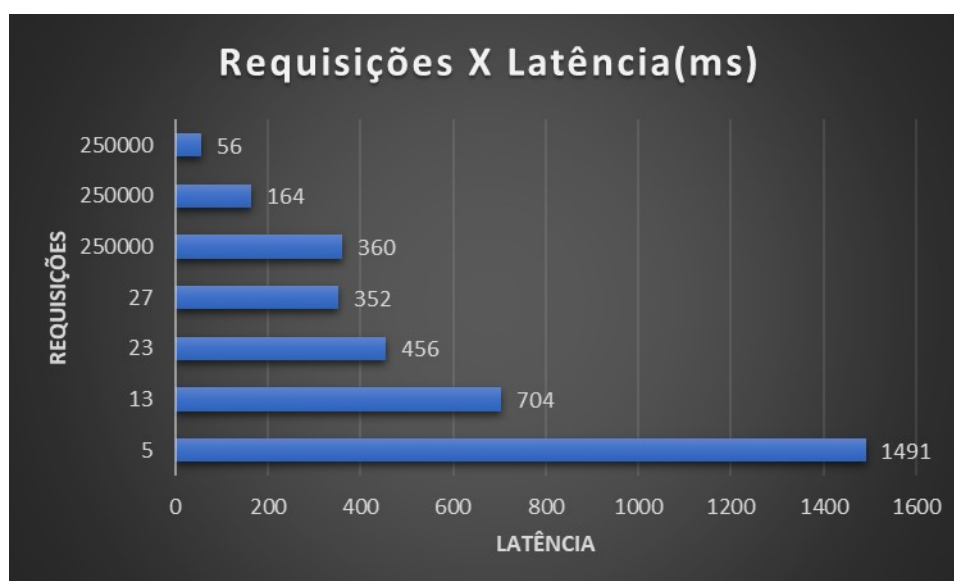


Figura 29 - Gráfico de Requisições x Latência



8. Link para código

O Repositório da aplicação deste projeto se encontra no Git em:

➤ <https://github.com/FernasG/comic-vine-api>

9. Link para vídeo

O vídeo mostrando a aplicação com as conexões com o banco se encontra em:

➤ <https://drive.google.com/file/d/1qN2HV8PN9Ifxybf6JmeCxW-n6iZbXL5x/view>

Anexo I - ER

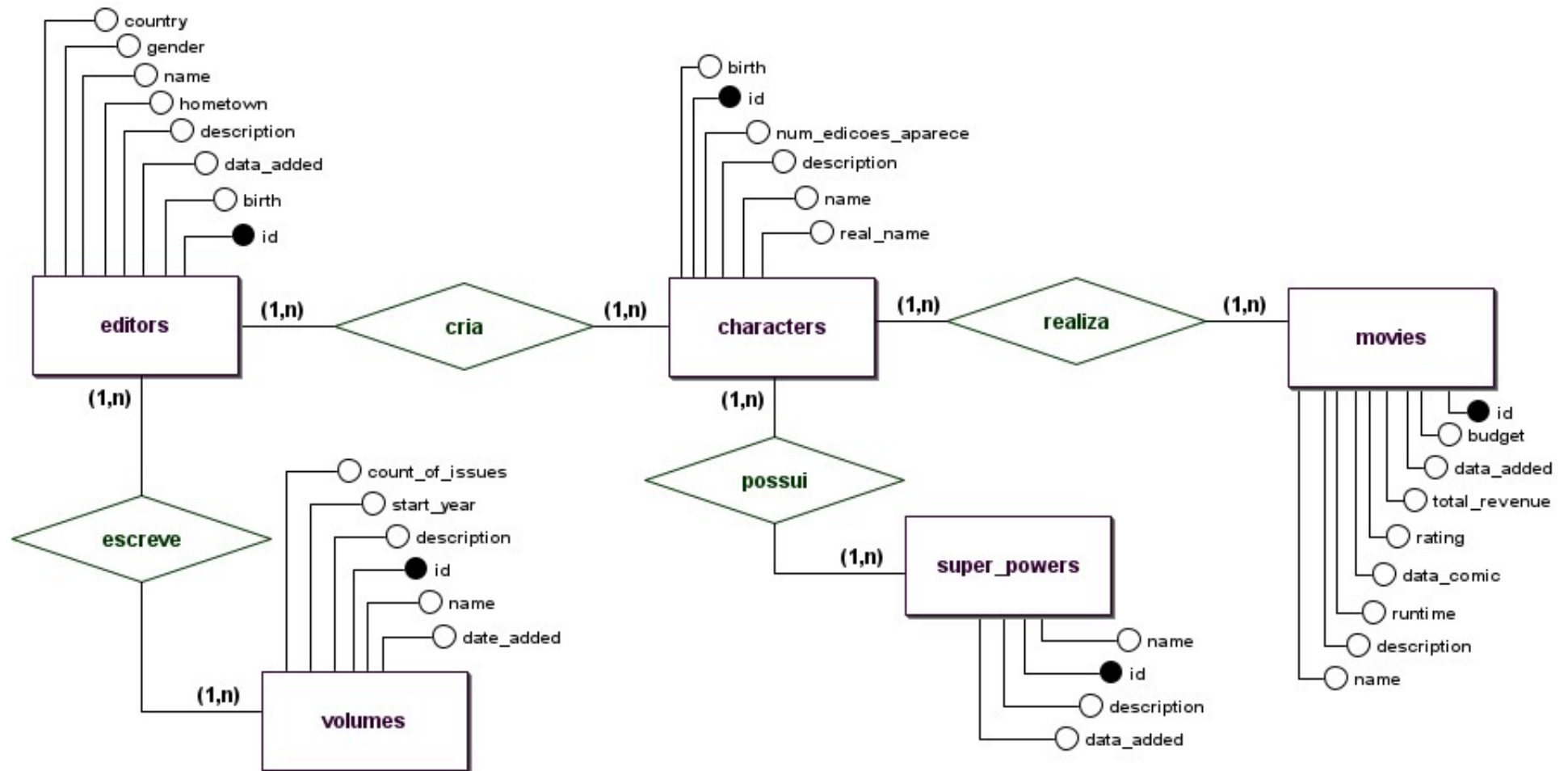


Figura 30- ER

Anexo II - Lógico

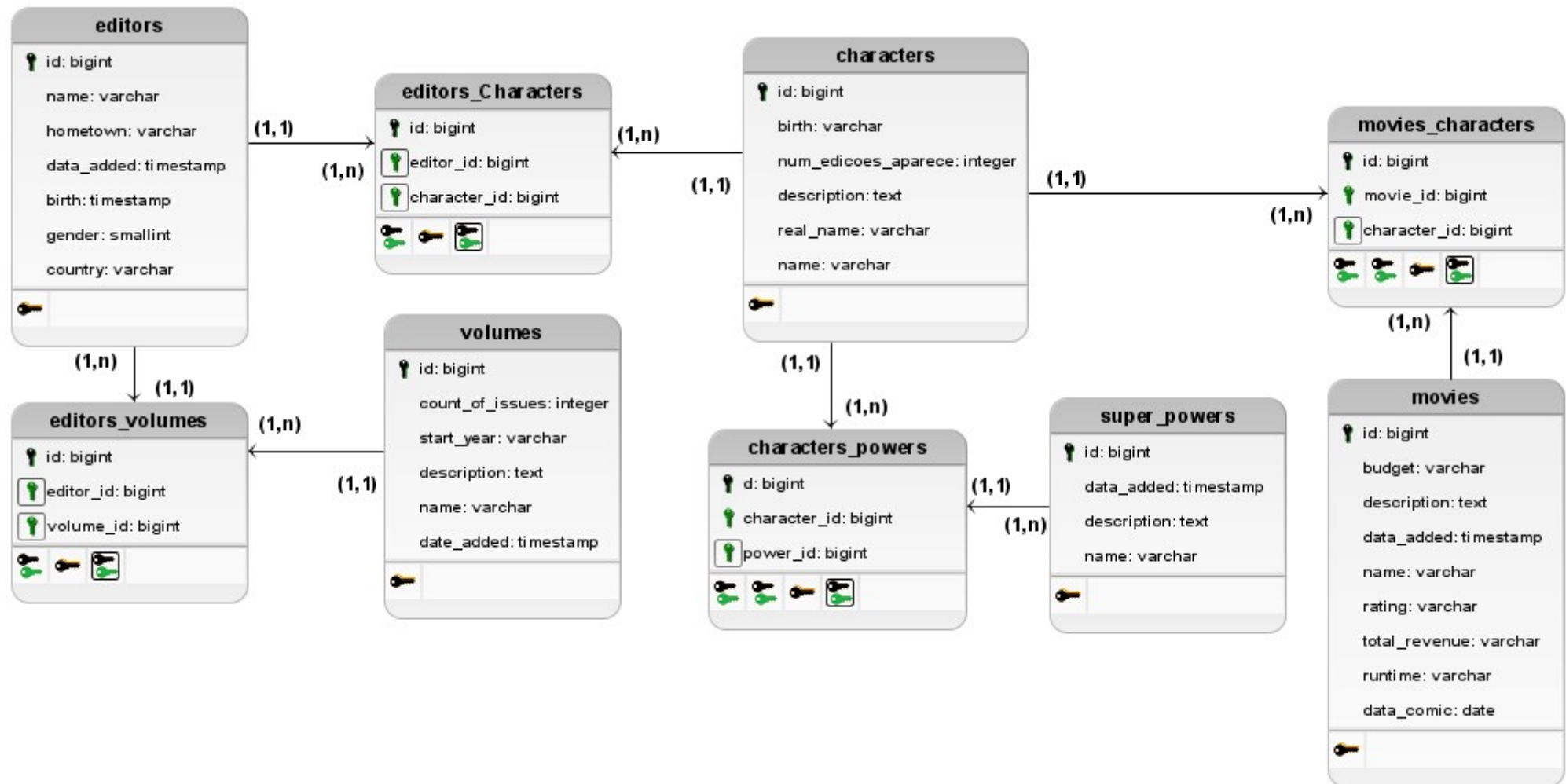


Figura 31 - Modelo Lógico