COM231 - Relatório 1 do Trabalho Prático

Douglas Raimundo de Oliveira Silva - 2019018540,
Vítor Ramos de Carvalho - 2019004231,
Vitor Siqueira Lobao - 2018004809,
Webster Aurelio Carvalho Ramos - 2018014716,
Yerro Cândido Ferreira - 2019004142

[1]Instituto de Matemática e Computação – Universidade Federal de Itajubá (UNIFEI)
Itajubá – MG – Brazil

{d2019018540,d2019004231,d2018004809}@unifei.edu.br

{d2018014716,d2019004142}@unifei.edu.br

1. Introdução

Este relatório tem como propósito documentar as etapas relativas à implantação do Trabalho Prático da Disciplina COM231 - Banco de Dados II, ministrada pela professora Vanessa Cristina no segundo semestre de 2021 (2021.2).

Por meio deste, pretende-se demonstrar as atividades desenvolvidas ao longo das cinco etapas do trabalho. Inicialmente serão apresentados: a API utilizada, bem como uma breve descrição da mesma, contendo os dados que a mesma disponibiliza e a relevância encontrada pelo grupo na utilização desta API para o desenvolvimento do trabalho. Além disso será apresentada a modelagem do banco de dados relacional, bem como os resultados do desenvolvimento de uma aplicação que consome os dados da API e popula um banco de dados utilizando o PostgreSQL.

Por fim serão apresentados a aplicação desenvolvida para a geração dos relatórios *Ad-Hoc* e os resultados obtidos com os testes de performance do banco de dados, os quais foram feitos usando a ferramenta *Apache Jmeter*.

2. Definição da API

A API escolhida para o desenvolvimento do trabalho foi a **The *Rick and Morty* API**, que pode ser acessada através do link `rickandmortyapi.com`. Essa API, como o próprio nome indica, fornece dados a respeito do universo da série de televisão *Rick and Morty*. Através desta API é possível obter dados de todos os personagens, locais e episódios das quatro temporadas. Foi desenvolvida por *Axel Fuhrmann* que, segundo ele, é um fã incondicional da série criada por *Justin Roiland* e *Dan Harmon*.

A escolha dessa API se justifica pelo fato de ser de simples compreensão e manipulação. Uma característica que também foi considerada na sua escolha é pelo fato de ser bem documentada: todos os atributos foram explicados e isso é fundamental quando se trata de dados. Além disso, a API do *Rick and Morty* é bem otimizada, as requisições não demoram para serem respondidas e isso foi um outro ponto crucial ao optarmos por ela.

A API retorna dados em formato JSON (formato muito comum quando lida-se com APIs), então um trabalho necessário seria criar alguma aplicação que consumisse

estes dados, processasse a saída para preparar os dados para serem armazenados no banco de dados relacional. Este processo será detalhado mais a frente.

Esta API possui um total de 826 personagens dos mais variados tipos e raças, 126 locais entre planetas, cenários, cometas e outros astros cosmológicos separados em diversas dimensões e possui um total de 51 episódios contendo informações básicas como data de exibição e nome. Estes dados estão separados por páginas: os dados dos personagens estão contidos em 42 páginas, locais está armazenado em 7 páginas e os episódios possuem um total de 3 páginas.

Além dos esclarecimentos técnicos anteriormente, outro fator importante foi que o grupo todo é fã da série. Trabalhar com dados que envolvam algo de interesse da equipe traz uma melhor eficiência nos processos pois assim o projeto é mais divertido e interessante para os integrantes.

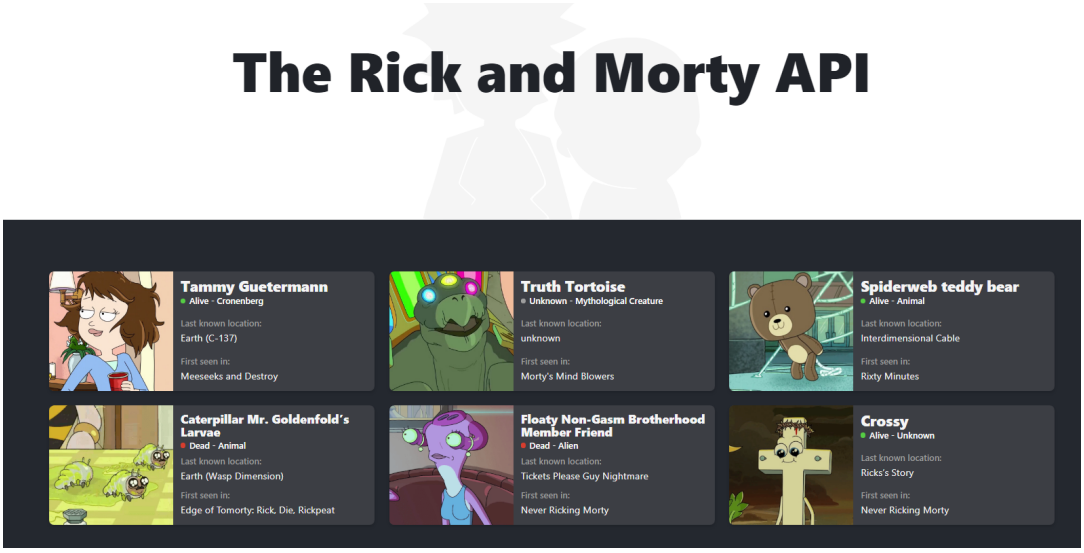


Figura 1. Homepage da API.

3. Engenharia Reversa dos Dados da API

Após a etapa de definição da API, coube ao grupo fazer um trabalho de engenharia reversa dos dados que eram fornecidos. Para isso, o grupo reuniu e analisou as responses da API para realizar o mapeamento dos atributos relevantes e interessantes. Após a análise, o grupo concluiu que iria utilizar todos os dados que a API entregava, exceto as imagens dos personagens pois esses dados não poderiam ser cruzados no momento da geração do relatório *Ad-Hoc*. O motivo pelo qual foi decidido capturar quase todos os dados é que assim poderia-se gerar relatórios mais flexíveis e completos, além de ser um desafio a nível de aplicação conjugar todas essas informações.

Uma informação importante foi que, durante todo o mapeamento dos atributos, a todo momento foi considerado o KPI principal do projeto: o relatório *Ad-Hoc*. Então todos os processos e seleção dos atributos foram considerados visando este objetivo final primordial do projeto.

4. Modelagem do Banco de Dados

Nesta seção foi iniciada a modelagem do banco de dados. Para isso, foi feito todo o processo de modelagem seguindo as etapas procedimentais aprendidas na disciplina de Banco de Dados I: foi construído o Modelo de Entidade e Relacionamento, seguido do Modelo Lógico e, por fim, a implementação do modelo físico.

O Sistema de Gerenciamento de Banco de Dados (SGBD) escolhido foi o PostgreSQL devido a maior familiaridade da equipe considerando todo o histórico das disciplinas de Banco de Dados. O PostgreSQL é um banco de dados que utiliza arquivos *HEAP*, isso significa que os registros nos arquivos de tabelas na memória física não são ordenados pois todas as inserções são feitas no final do arquivo de tabela. Além disso, o índice primário das tabelas são densos, sendo assim, é necessário um pouco mais da memória física para armazenar os arquivos de índice.

4.1. Modelo de Entidade e Relacionamento

O *MER* é uma das primeiras etapas na realização da modelagem do banco. Neste modelo é expresso uma ideia geral de como será a esquematização do banco. Com base nesse mapeamento é possível construir os próximos passos de um projeto de construção de um banco. Cada retângulo é uma tabela neste diagrama, já os losangos são as relações internas entre elas. Os círculos são os atributos e o preenchido é a chave primária da referida tabela. Todas as relações resultarão em chaves estrangeiras, por isso é importante estar atendo à cardinalidade das relações expressas pelos números que conecta a tabela à sua relação.

Foi decidido separarmos em três tabelas e duas relações: são elas *Character*, *Episode* e Location, as relações são Have e *Appear*. A tabela *Character* armazenará informações dos personagens como ID (identificador padrão do registro), nome, status, espécie, subespécie, gênero e um atributo referente à data de criação do registro no banco. A tabela Location armazenará dados dos lugares do universo de *Rick and Morty*, tais como ID, nome, tipo, dimensão e atributo de data de criação do registro. Além dessas, a tabela *Episode* persistirá os dados dos episódios da série como ID, nome, data da exibição, código identificador do episódio e criação do registro. Abaixo é possível ver o esquema do banco de forma superficial.

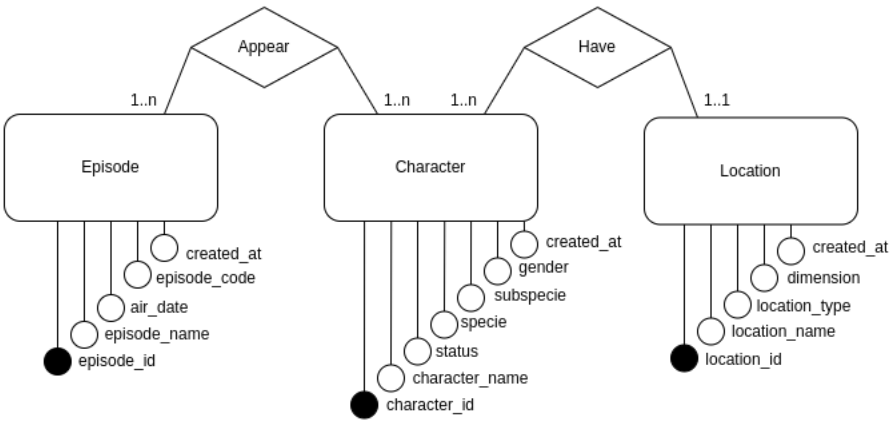


Figura 2. Modelo Entidade Relacionamento.

4.2. Modelo Lógico

A etapa do Modelo Lógico é quando detalha-se um pouco mais a esquematização do banco em relação a tipagem, restrições e nomeação dos atributos de cada tabela do banco de dados. Neste modelo, também é especificado as chaves primárias e estrangeiras das relações entre as tabelas, inclusive para relações *n* para *n*, agregando mais profundidade no processo de criação do banco.

Foram decididos nomes que não fossem muito grandes, mas que expressassem bem o significado do atributo. A fim de padronização, os campos referentes a texto foram tipados como VARCHAR(64), limitando assim a 64 caracteres. Os campos *status* e *gender* serão tipos especiais e qualquer valor inserido deve pertencer ao conjunto de valores listados por esse tipo de variável.

Na relação *Appear* foi necessária a criação de uma nova tabela para comportar o tipo de relação entre *Episode* e *Character*. Além disso, foi criada uma restrição UNIQUE na combinação das chaves estrangeiras *character_id* e *episode_id*. Existem várias maneiras de implementar uma tabela de relação *n* para *n*, a maneira optada foi para que evitasse chaves primárias compostas que também fossem chaves estrangeiras, esta combinação poderia trazer problemas em manutenções futuras, apesar de fazer sentido no contexto do projeto.

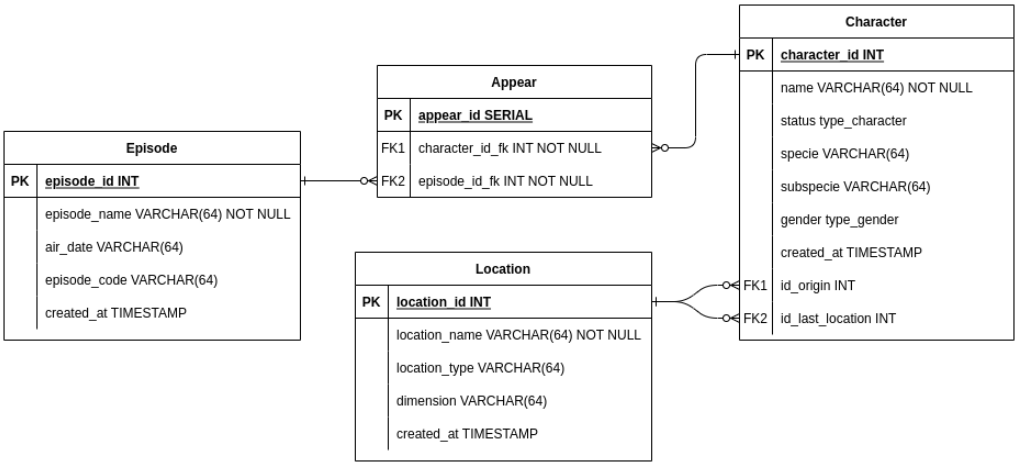


Figura 3. Modelo Lógico.

4.3. Usuários e Privilégios

Foi decidido implementar, junto do banco, alguns usuários e seus respectivos privilégios. Para exemplificar, optou-se por criar dois usuários *master* e *usr*. Além disso, foi criada uma ROLE denominada *superuser* que conteria um conjunto de privilégios, esta ROLE teria acesso total ao banco.

Para o *master* foi garantida a ROLE expressa no parágrafo anterior, este usuário teria todos os privilégios em todas as entidades do banco de dados. Já para o *usr*, foi garantida apenas o comando de consulta SELECT em todas as tabelas do banco de dados. Ambas as senhas foram para fins didáticos.

Essa etapa de criação de usuários e garantia de privilégios é extremamente delicada pois qualquer erro ou permissão excessiva pode gerar grandes transtornos para os

responsáveis do banco e, em quase todos os casos, para a empresa toda. Quando é requisitada alguma permissão de acesso por algum usuário, uma série de perguntas e análises devem ser feitas a fim de manter a segurança dos dados armazenados.

```
CREATE ROLE superuser;
GRANT ALL PRIVILEGES ON DATABASE "ricknmorty" TO superuser;
GRANT ALL ON ALL TABLES IN SCHEMA public TO superuser;

CREATE USER master WITH PASSWORD 'senhamaster123';
GRANT superuser TO master;

CREATE USER usr WITH PASSWORD 'senhausr123';
GRANT SELECT ON ALL TABLES IN SCHEMA public TO usr;
GRANT USAGE ON SCHEMA public TO usr;
```

Figura 4. Usuários e privilégios.

4.4. Índices

Pra fins didáticos, foi decidido a criação de um índice denominado *character_specie_index* na coluna relacionada a espécie dos personagens partindo da suposição de que tal coluna poderia ser muito acessada. Trata-se de um índice secundário em um arquivo *HEAP*, como consequência, o índice é denso, logo é criado um arquivo de índice com o campo indexado e um ponteiro referenciado registros. Além disso, a estrutura de dados utilizada foi para construção do índice foi a *B-tree* (por padrão do *PostreSQL*).

Consultas que utilizavam a coluna de espécies como filtro eram as mais demoradas na proporção do banco. Antes da criação do índice, registros que continham *specie* iguais a *'Humanoid'* levavam uma média de 0.240 ms. Utilizando o EXPLAIN ANALYSE podíamos concluir que era realizado um *Seq Scan* na tabela *Character* como única estratégia de busca.

```
1  EXPLAIN ANALYSE SELECT * FROM character WHERE specie = 'Humanoid'
```

Data Output | Explain | Messages | Notifications

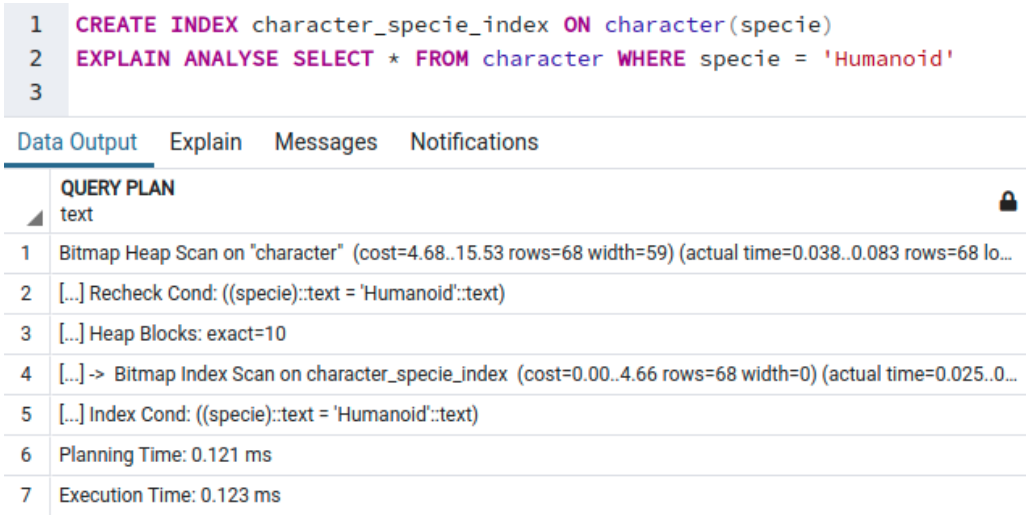
	QUERY PLAN text
1	Seq Scan on "character" (cost=0.00..20.33 rows=68 width=59) (actual time=0.025..0.294 rows=68 loops=1)
2	[...] Filter: ((specie)::text = 'Humanoid'::text)
3	[...] Rows Removed by Filter: 758
4	Planning Time: 0.090 ms
5	Execution Time: 0.333 ms

Figura 5. Estratégia de busca sem índice implementado.

Após a implementação do índice *character_specie_index*, houve um ganho de cerca de 0.210 ms. O tempo ganho, na prática, é insignificante, mas em termos percentuais, a implementação desse índice trouxe uma melhora próxima a 37%. Supondo

que a consulta necessitasse de 1000 segundos (ou 16 minutos e 40 segundos) para ser finalizada, com o índice o SGBD poderia realizar a mesma consulta em 630 segundos (ou 10 minutos e 30 segundos). Trata-se de um exemplo puramente teórico e que não condiz com a prática, pois este cálculo dependeria de muitos critérios, mas para efeitos de exemplificação pode-se notar a diferença de um índice em consultas no banco.

Abaixo podemos ver que o SGBD utiliza um *Bitmap Index Scan* utilizando o índice *character_specie_index*, retornando 10 blocos de memória que contêm tuplas que satisfazem a condição *specie* iguais a *'Humanoid'*. Finalizando então com um *Bitmap Heap Scan* nos blocos retornados.



```
1  CREATE INDEX character_specie_index ON character(specie)
2  EXPLAIN ANALYSE SELECT * FROM character WHERE specie = 'Humanoid'
3
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Bitmap Heap Scan on "character" (cost=4.68..15.53 rows=68 width=59) (actual time=0.038..0.083 rows=68 lo...
2	[...] Recheck Cond: ((specie)::text = 'Humanoid'::text)
3	[...] Heap Blocks: exact=10
4	[...] -> Bitmap Index Scan on character_specie_index (cost=0.00..4.66 rows=68 width=0) (actual time=0.025..0...
5	[...] Index Cond: ((specie)::text = 'Humanoid'::text)
6	Planning Time: 0.121 ms
7	Execution Time: 0.123 ms

Figura 6. Estratégia de busca com índice implementado.

Para a criação de índices, é necessário uma análise profunda acerca do banco de dados e sua utilização. Índices, assim como os dados, ocupam a memória principal e física de um computador e este é um dos motivos que fazem com que os administradores analisem se realmente há necessidade de um índice naquele atributo. Outro ponto importante é que nem todos os índices são utilizados pelo SGBD, a necessidade de utilização deles varia de consulta para consulta, e em alguns casos podem diminuir a performance do banco de dados, indo totalmente contrário a sua proposta.

5. Carga no Banco de Dados

Uma das tarefas necessárias é carregar o banco de dados, isso significa consumir a API via *request*, estruturar a *response* obtida e armazenar o resultado desse processo no banco de dados relacional especificado nas seções anteriores. Para isso, foi utilizada a linguagem *Python* por diversos motivos, como listados abaixo:

- Familiaridade dos membros;
- Linguagem dinâmica e poderosa o suficiente;
- Grande comunidade ativa;
- Matéria ter sido trabalhada em cima desta linguagem;
- Utilização da biblioteca *psycopg2* conforme disciplina;
- Linguagem multiparadigma, permitindo Orientação a Objetos, e;
- Fácil integração com o banco de dados.

A aplicação foi separada em 5 módulos: *character*, *episode*, *location*, *appear* e *main*, sendo que esta última atuava como um *controller* do sistema. Todos os módulos possuem uma estrutura padrão: a declaração do objeto a qual se refere e os métodos necessários para popular o banco.

Na declaração do objeto tem-se os atributos *page* responsável por armazenar o respectivo *link* necessário da API, um atributo *data* com a *response* em formato JSON e *__conn* onde há a *string* com os parâmetros de conexão com o banco.

Os métodos também são padronizados: *get_results* é responsável por tomar os resultados da *response*, *create_row* atua na estruturação do dado convertendo em JSON para uma lista preparada para inserção e, por último, *insert_row* trabalha conectando a aplicação ao banco, recebe os valores de inserção de emphcreate_row e insere-os na respectiva tabela.

No módulo *main* são instanciados os objetos e manipulados conforme necessário. A API retorna diversas páginas que possuem um campo (*next*) onde é possível acessar a próxima página, assim que este campo é nulo, significa que a página atual é a última. Foi executado um *while loop* inserindo os dados e pulando para a próxima página até ser encontrada um *next_page* nulo. Para cada iteração da estrutura de repetição, são tomados os resultados e então estes resultados são iterados através de um *for loop* onde, para cada iteração da segunda estrutura de repetição, o dado é estruturado e logo em seguida é inserido. Por último, é feita a validação da próxima página como explicado acima.

id [PK] integer	name character varying (64)	status type_character	specie character varying (64)	subspecie character varying (64)	gender type_gender	id_origin integer	id_last_location integer	created_at timestamp wi
1	Rick Sanchez	Alive	Human	[null]	Male	3	1	2021-11-22
2	Morty Smith	Alive	Human	[null]	Male	3	[null]	2021-11-22
3	Summer Smith	Alive	Human	[null]	Female	20	20	2021-11-22
4	Beth Smith	Alive	Human	[null]	Female	20	20	2021-11-22
5	Jerry Smith	Alive	Human	[null]	Male	20	20	2021-11-22
6	Abadango Cluster Princess	Alive	Alien	[null]	Female	2	2	2021-11-22
7	Abradolf Lincler	unknown	Human	Genetic experiment	Male	21	20	2021-11-22

Figura 7. Tabela *Character* populada.

	id [PK] integer	name character varying (64)	type character varying (64)	dimension character varying (64)	created_at timestamp without time zone
1	1	Earth (C-137)	Planet	Dimension C-137	2021-11-22 08:42:19.126925
2	2	Abadango	Cluster	unknown	2021-11-22 08:42:19.138458
3	3	Citadel of Ricks	Space station	unknown	2021-11-22 08:42:19.148366
4	4	Worldender's lair	Planet	unknown	2021-11-22 08:42:19.155489
5	5	Anatomy Park	Microverse	Dimension C-137	2021-11-22 08:42:19.164836
6	6	Interdimensional Cable	TV	unknown	2021-11-22 08:42:19.180098
7	7	Immortality Field Resort	Resort	unknown	2021-11-22 08:42:19.187169

Figura 8. Tabela *Location* populada.

	id [PK] integer	name character varying (64)	air_date character varying (64)	episode_code character varying (64)	created_at timestamp without time zone
1	1	Pilot	December 2, 2013	S01E01	2021-11-22 08:42:23.67812
2	2	Lawnmower Dog	December 9, 2013	S01E02	2021-11-22 08:42:23.696714
3	3	Anatomy Park	December 16, 2013	S01E03	2021-11-22 08:42:23.705649
4	4	M. Night Shaym-Aliens!	January 13, 2014	S01E04	2021-11-22 08:42:23.712674
5	5	Meeseeks and Destroy	January 20, 2014	S01E05	2021-11-22 08:42:23.71943
6	6	Rick Potion #9	January 27, 2014	S01E06	2021-11-22 08:42:23.725687
7	7	Raising Gazorpazorp	March 10, 2014	S01E07	2021-11-22 08:42:23.731331

Figura 9. Tabela *Episode* populada.

Na Figura 10 é possível enxergar algumas estatísticas do SCHEMA *public* do banco de dados. Foi utilizado uma tabela especial denominada *pg_stat_user_tables* que possui informações importantes acerca das entidades implementadas. Nela, pode-se verificar informações como nome e quantidade de registros (*n_live_tup*), entre outros.

```
1  SELECT
2      schemaname,
3      relname,
4      n_live_tup
5  FROM pg_stat_user_tables
6  ORDER BY n_live_tup;
```

Data Output | Explain | Messages | Notificat

	schemaname name	relname name	n_live_tup bigint
1	public	episode	51
2	public	location	126
3	public	character	826
4	public	appear	1267

Figura 10. Descrição de todas as tabelas.

Como esperado, a tabela *Appear* é a maior em número de registros pois relaciona a aparição de todos os personagens em relação aos episódios. Além disso, houve, na série, 826 personagens diferentes em 126 locais distintos ao longo dos 51 episódios diferentes.

Para acessar o repositório no Github contendo todos os códigos e *script* e demais documentações: `https://github.com/Dellonath/ricknmorty`

Para acessar o vídeo da apresentação da primeira parte: `https://www.youtube.com/watch?v=Qr-pSvwCL4c`.

6. Testes no Apache JMeter

O Apache Jmeter é um *software* que fornece ao usuário especializado algumas ferramentas para análise de desempenho de uma série de serviços diferentes. Com ele, é possível gerar relatório acerca da efetividade e velocidade, por exemplo, de sistemas implementados. O Apache JMeter foi implementado em Java pela Apache Software Foundation, sua versão mais recente, desde a confecção deste relatório, é a *5.4.1*.

Nesta seção do projeto foram realizados testes no banco de dados utilizando o *software* Apache JMeter, objetivando uma análise da *performance* do SGBD ao lidar com

múltiplas requisições, divididas em duas partes: múltiplas consultas ao banco por um único usuário e múltiplos usuários realizando uma única consulta. A cada disparo de consulta(s), será armazenada a latência em milissegundos, com o respectivo mapeamento do número do teste. Serão realizados 30 testes independentes para as duas etapas.

6.1. Testes com um usuário

O primeiro teste consiste em simular um usuário fazendo requisições ao banco, mas com uma progressão pré-estabelecida na quantidade de requisições. Primeiramente, o usuário fará 1 requisição, em seguida será armazenada a latência de resposta do banco, este procedimento irá se repetir 30 vezes. Depois de finalizado, será então feito disparos com 10 requisições oriundas de um usuário e, semelhante ao passo anterior, será armazenado a latência e repetido 30 vezes. Esse esquema de testes será feito, em seguida, para 100, 500, 600, 700 e 800 requisições. Para melhorar a visualização, o gráfico da Figura 11 foi transformado para uma escala logarítmica.

Na Figura 11, construída baseada na Tabela 6.1, podemos ver como se comportam as latências em diferentes testes. Podemos notar que a latência segue constante no primeiro teste e segue este padrão até para o teste de 800 requisições, apesar dos últimos terem uma latência média muito baixa. A que teve maior média (1558.7 ms) foi a latência referente a uma única requisição feita por um usuário, que também possui o maior desvio padrão (101.2 ms).

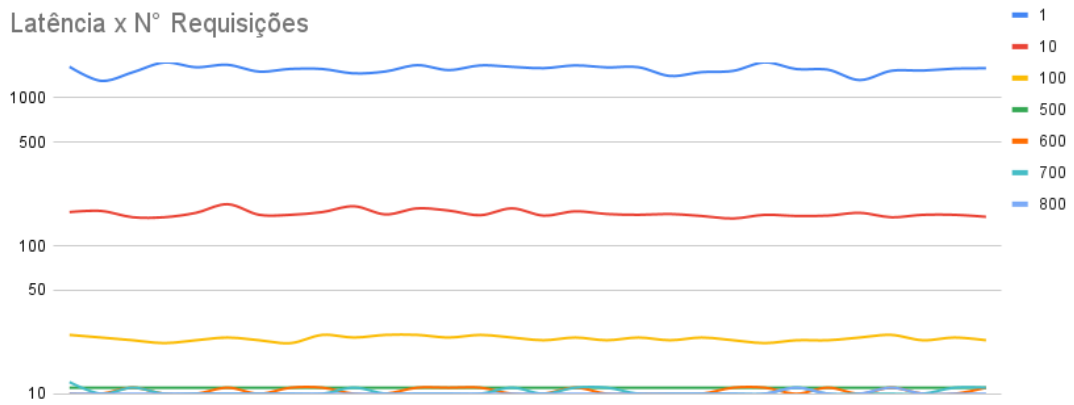


Figura 11. Latência por Número de Requisições em escala logarítmica.

Além disso, houve uma queda significativa no tempo a partir de 10 requisições, estabilizando em uma média de 165.4 ms e desvio padrão de 8.9 ms. Uma diminuição de cerca de 89% na média (e 91% no desvio padrão) em comparação com a primeira sessão de testes.

Para fins comparativos, foi utilizado o Teste t de Student que tem como hipótese nula (H_0) que ambos os grupos testados não possuem diferença estatística significativa entre si. Este teste compara as médias dos dois conjuntos através de uma manipulação algébrica, retornando o valor-ρ. Abaixo podemos ver a estrutura de hipóteses do teste aplicado, expressando matematicamente a diferença entre a hipótese nula e hipótese alternativa neste caso.

$$H_0 : \ \mu_0 - \mu_1 = 0$$
$$H_1 : \ \mu_0 - \mu_1 \neq 0$$

Na Tabela 6.1 é possível consultar, de forma mais detalhada, os resultados dos testes executados para um único usuário. Além da média (μ), do desvio padrão (σ) e do valor-ρ.

N° Teste	1	10	100	500	600	700	800
1	1622	169	25	11	10	12	10
2	1302	172	24	11	10	10	10
3	1488	156	23	11	11	11	10
4	1733	156	22	11	10	10	10
5	1610	167	23	11	10	10	10
6	1671	191	24	11	11	10	10
7	1504	162	23	11	10	10	10
8	1569	162	22	11	11	10	10
9	1566	169	25	11	11	10	10
10	1463	185	24	11	10	11	10
11	1507	163	25	11	10	10	10
12	1662	179	25	11	11	10	10
13	1540	173	24	11	11	10	10
14	1656	161	25	11	11	10	10
15	1619	179	24	11	10	11	10
16	1586	160	23	11	10	10	10
17	1656	171	24	11	11	11	10
18	1606	164	23	11	10	11	10
19	1610	162	24	11	10	10	10
20	1405	164	23	11	10	10	10
21	1491	159	24	11	10	10	10
22	1521	153	23	11	11	10	10
23	1736	162	22	11	11	10	10
24	1564	159	23	11	10	11	11
25	1549	160	23	11	11	10	10
26	1318	167	24	11	10	10	10
27	1522	156	25	11	11	10	11
28	1528	162	23	11	10	10	10
29	1575	162	24	11	10	11	10
30	1583	157	23	11	11	11	10
Média (μ)	1558.7	165.4	23.6	11.0	10.4	10.3	10.1
Desvio Padrão (σ)	101.2	8.9	0.9	0.0	0.5	0.5	0.3
Teste t de Student (valor-ρ)	-	0.0	0.0	0.0	0.0	0.5	0.0

Tabela 1. Testes de latência (ms) de requisições de um usuário.

O teste de hipótese foi aplicado par a par para os conjuntos de latências, sendo executado comparando o grupo 1 com 10, 10 com 100, 100 com 500 e assim sucessivamente. Para o teste em questão, foi utilizado o nível de significância convencional de 0.05 ($\alpha = 0.05$), resultando um nível de confiança estatística de 95%.

Com os esclarecimentos acima, podemos perceber que há sim diferenças estatísticas significativas entre quase todos os pares, apesar de possuírem médias extremamente próximas, mas em escalas de latência de tempo em ms, elas são consideráveis. Vale ressaltar que entre 600 requisições e 700 requisições é o único momento onde não há essa diferença considerável.

6.2. Testes com múltiplos usuários

Nesta seção será descrito o segundo teste realizado, que consiste em aumentar gradativamente o número de usuários fazendo uma única requisição, simulando um sistema em produção. Para ele, foi utilizado o mesmo *software* de teste, o Apache JMeter.

Foi feito, em primeira instância, uma única *thread* realizando uma única consulta, sendo armazenada a latência (em ms) da resposta do banco à requisição. Assim como no teste de várias requisições de um usuário, foram realizados 30 testes nesta primeira leva. Após os 30 testes, foram aumentados a quantidade de usuários para 10, e refeito os 30 testes armazenando a latência. Este procedimento foi repetido para novamente para 100, 500, 600 e 700 *threads*.

Na Figura 12 podemos acompanhar o desempenho do banco para diferentes requisições para níveis de escalonamento distintos. Podemos observar uma maior variância na latência quando a quantidade de usuários realizando consultas supera a margem dos 500. A volatilidade para testes acima de 100 usuários também é bem maior comparada com a volatilidade de 10 ou menos usuários. Outro ponto observável é que, de maneira geral, independente do nível de estresse aplicado ao banco, as variações tendem a se concentrar em um valor central (podemos afirmar com certo grau de segurança que o valor central é a média μ_i) para todos eles.

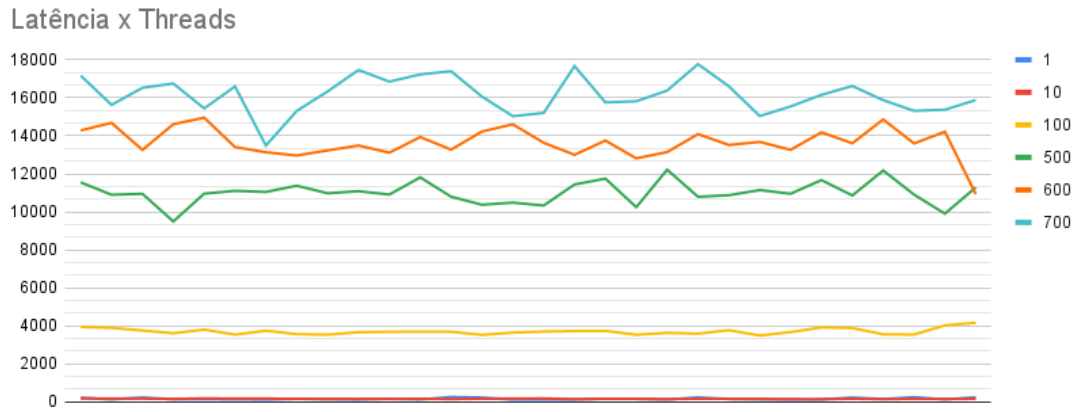


Figura 12. Latência por Número de Threads.

Na Tabela 6.2 é possível consultar, de forma mais detalhada, os resultados dos testes executados para um único usuário. Além da média (μ), do desvio padrão (σ) e do valor-ρ dos testes aplicados par a par.

N° Teste	1	10	100	500	600	700
1	213	181	3940	11549	14284	17168
2	124	166	3890	10899	14683	15629
3	225	167	3753	10950	13257	16528
4	136	152	3610	9494	14613	16753
5	131	180	3798	10959	14957	15444
6	128	167	3539	11105	13410	16598
7	115	167	3739	11047	13134	13489
8	151	160	3562	11373	12966	15313
9	134	152	3536	10978	13232	16338
10	118	151	3663	11085	13490	17465
11	157	151	3681	10910	13113	16853
12	123	160	3696	11812	13937	17232
13	250	142	3686	10796	13274	17399
14	225	163	3524	10374	14227	16067
15	112	168	3642	10485	14607	15035
16	116	176	3697	10332	13636	15203
17	120	140	3724	11440	13001	17671
18	153	163	3728	11746	13751	15760
19	150	156	3532	10245	12813	15819
20	121	141	3630	12218	13146	16382
21	230	162	3589	10793	14090	17775
22	148	156	3767	10874	13519	16608
23	130	152	3491	11145	13679	15040
24	117	140	3669	10954	13264	15548
25	129	134	3915	11669	14185	16157
26	222	162	3879	10862	13609	16623
27	145	139	3552	12174	14864	15876
28	242	154	3542	10906	13600	15316
29	126	148	4022	9904	14217	15376
30	237	159	4163	11287	10934	15884
Média (μ)	157.6	157.0	3705.3	11012.2	13649.7	16145.0
Desvio Padrão (σ)	46.6	12.1	162.1	603.9	792.7	949.9
Teste t de Student (valor-ρ)	-	0.94	0.0	0.0	0.0	0.0

Tabela 2. Tabela de testes de latência (ms) de requisições de um usuário.

7. Relatório AD-HOC

O relatório AD-HOC é um documento extremamente útil quando se trata de áreas como *Marketing* e *Business Intelligente* diretamente relacionadas à área de dados. Este tipo de relatório é realmente útil pois com ele se têm acesso a diversas informações compiladas em uma única folha e com isso é possível fazer comparações rápidas e precisas acerca da situação atual.

Uma das tarefas deste trabalho foi implementar o AD-HOC para que um determinado usuário pudesse gerar relatórios personalizados de maneira simples e intuitiva.

Sendo possível consultar registros e capturar informações acerca dos dados do banco e construir gráficos capazes de gerar *insights* que possam auxiliar diversos times e agregar valor ao negócio, por exemplo.

No relatório AD-HOC, enquanto no menu inicial, são listados todos os possíveis atributos que devem ser escolhidos e selecionados pelo usuário para a confecção do documento. Na Figura 13 há um exemplo de como é feita a seleção dos atributos. Os atributos são selecionados no canto esquerdo e os já escolhidos estão listados no canto direito. Neste caso e momento em específico, foi selecionado o campo ID, nome, gênero, status e espécie dos personagens.

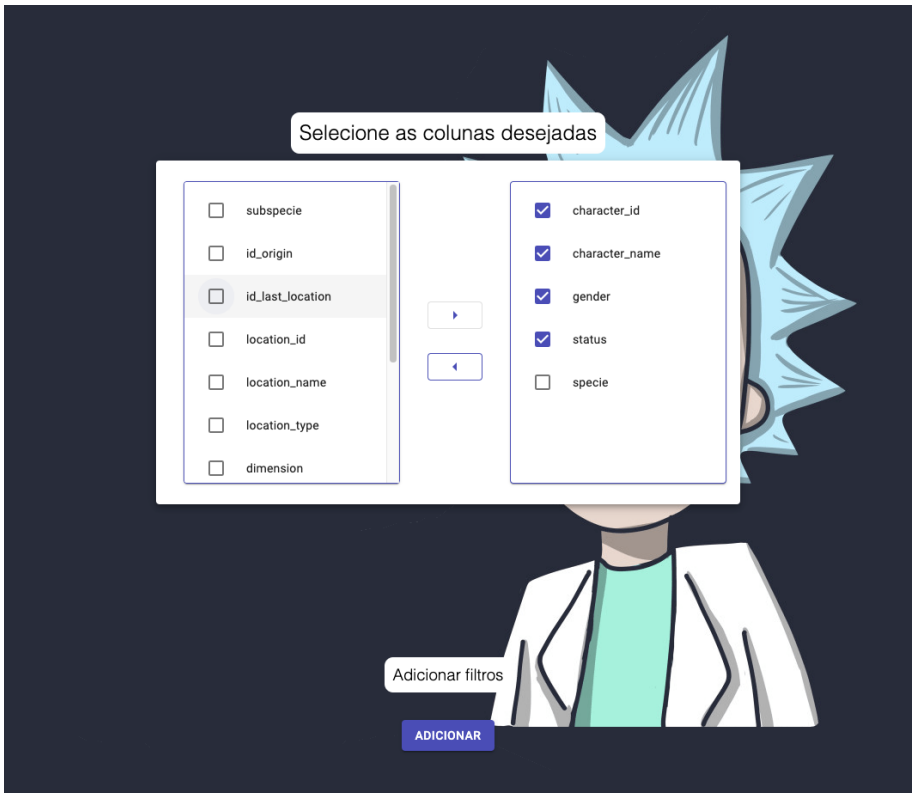


Figura 13. Menu de seleção de atributos.

Após esta etapa, o usuário deve realizar um filtro nos dados para que o relatório se adéque melhor à sua necessidade. O filtro também é intuitivo e de fácil aplicação. Na Figura 14, por exemplo, um determinado usuário exigiu que no relatório sejam retornados apenas os personagens cujo ID é igual a 20.

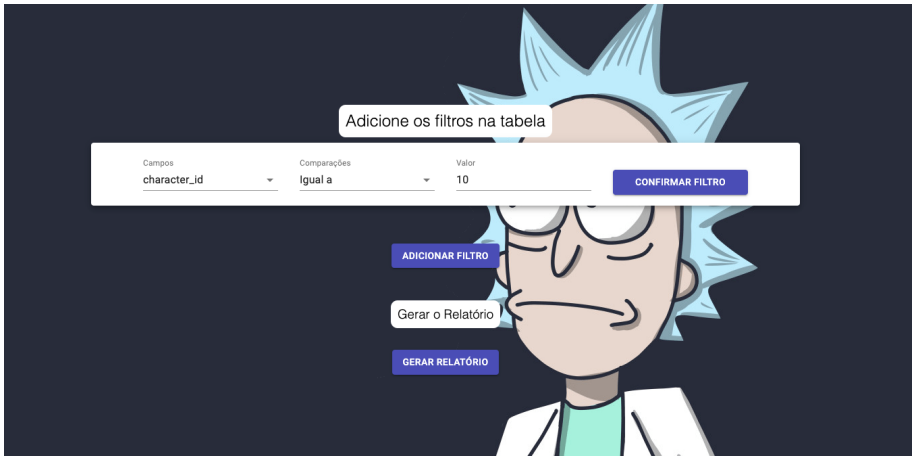


Figura 14. Aplicação do filtro.

Como resultado, é obtida uma relação de registros semelhante à Figura 15, onde é listado todos os registros que se encaixam dentro de todos campos selecionados e do filtro aplicado. Nesta Figura específica, foi feita uma seleção dos campos *character_id*, *gender* e *character_name* sem nenhum filtro aplicado.

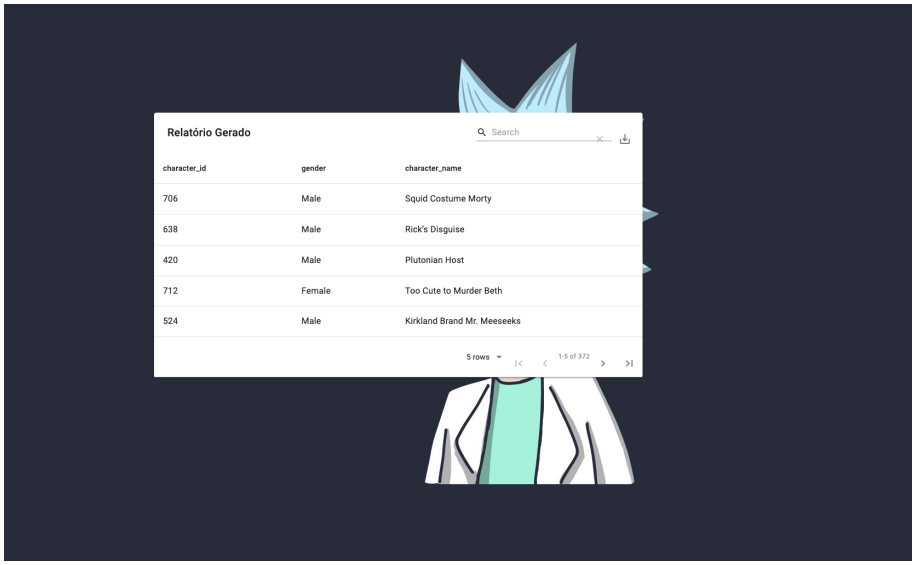


Figura 15. Relação do relatório AD-HOC.

Outra *feature* também implementada consiste em persistir o relatório através da exportação dos dados em CSV (*Comma Separated Value*) ou em arquivo formato PDF. Através desta ferramenta o usuário pode armazenar seu relatório para ser importado em outro sistema através do CSV ou para apresentação e envio de resultados para os tomadores de decisão do negócio, por exemplo, através do PDF. Nas Figuras *fig:relatorio* e *fig:csv* é possível ver exemplos das saídas em cada uma das opções fornecidas pelo sistema para o usuário.

Relatório Gerado

character_id	gender	character_name
706	Male	Squid Costume Morty
638	Male	þÿ R i c k s D i s g u i s e
420	Male	Plutonian Host
712	Female	Too Cute to Murder Beth
524	Male	Kirkland Brand Mr. Meeseeks
603	Genderless	Crossy
596	Female	Floaty Non-Gasm Brotherhood Member Friend
644	Male	Old Glorzo

Figura 16. Relatório persistido em PDF.

	A	B	C
1	"character_id"	"gender"	"character_name"
2	706	"Male"	"Squid Costume Morty"
3	638	"Male"	"Rick's Disguise"
4	420	"Male"	"Plutonian Host"
5	712	"Female"	"Too Cute to Murder Beth"
6	524	"Male"	"Kirkland Brand Mr. Meeseeks"
7	603	"Genderless"	"Crossy"
8	596	"Female"	"Floaty Non-Gasm Brotherhood Member Friend"
9	644	"Male"	"Old Glorzo"
10	411	"Male"	"Alien Mexican Armada"
11	746	"Male"	"Cenobite"
12	708	"Female"	"Squid Costume Summer"
13	686	"Male"	"Mr. Nimbus' Squid"
14	35	"unknown"	"Bepisian"
15	505	"Male"	"Fascist Shrimp Morty"
16	47	"Male"	"Birdperson"
17	447	"Male"	"Anchor Gear"
18	630	"Male"	"Morty Smith"
19	35	"unknown"	"Bepisian"
20	566	"Male"	"Debrah's Partner"

Figura 17. Relatório persistido em CSV.

Para acessar o repositório contendo os códigos do relatório AD-HOC, basta acessar o link abaixo: `https://github.com/vitorSiqueiraLobao/rickandmorty`.

8. Conclusão

Nesta primeira etapa do projeto foi possível aplicarmos muitos conceitos aprendidos na disciplina de Banco de Dados II, além de recapitularmos grande parte da disciplina de Banco de Dados I. As justificativas, os ajustes e modelagem voltado para o objetivo, entre outros fundamentos, foram a base para construção deste início de projeto. Foi possível vermos, na prática, como é feita o consumo de dados semi-estruturados de APIs e como eles devem ser processados para encaixarem na característica estruturada de um banco de dados relacional, sendo necessário aplicação dos fundamentos do paradigma Orientado a Objetos para implementação da aplicação para popular o banco.

Além disso, a reunião de todo o conhecimento adquirido durante ambas as disciplinas nos fizeram compreender ainda mais e recapitular todos os procedimentos de arquitetação e construção de um banco de dados, indo dos modelos mais superficiais

como o Modelo de Entidade-Relacionamento e Lógico até a implementação do Modelo Físico utilizando *SQL*.

Na segunda parte, através dos testes com o JMeter foi possível entender de forma mais profunda como é a *performance* de um banco de dados e como ele se comporta em diferentes situações de requisições. Os resultados mostraram uma certa estabilidade nos diferentes níveis de estresse do banco, mas com leve grau de volatilidade. Também, foi aplicado testes estatísticos para verificar se há uma diferença estatística significativa entre as latências dos testes e em boa parte deles aparente não seres significantes quando aplicados par a par.

No final, a implementação do relatório AD-HOC para que sejam feitas consultas de forma mais simples e direta, criando um encapsulamento em todo o sistema de consulta ao banco, democratizando a utilização da ferramenta e maior acesso à informações disponível por diversos usuários diferentes.