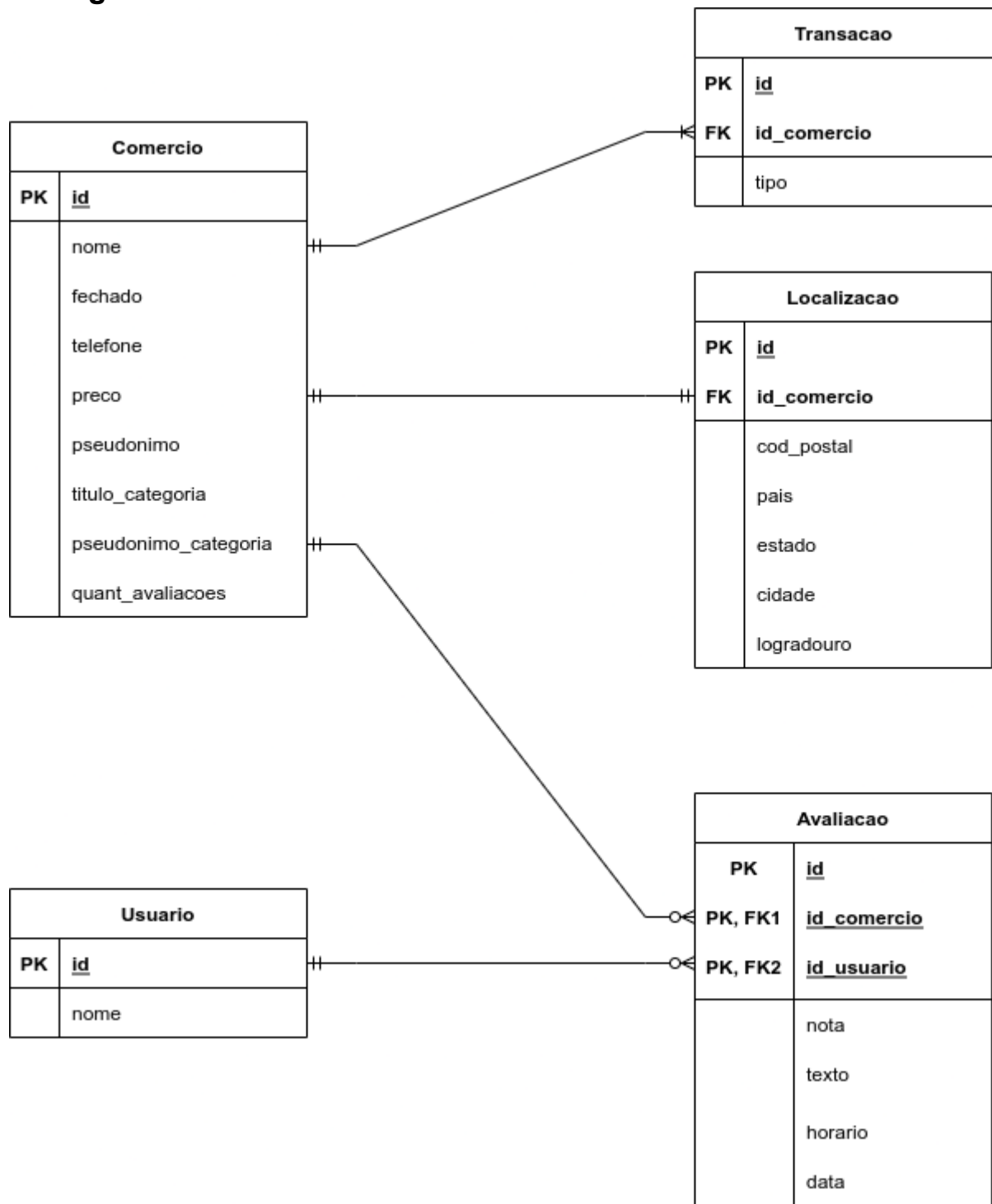


# **BANCO DE DADOS POPULADO COM YELP API**

## **RELATÓRIO**

<b>Andrew Enrique Oliveira</b>	<b>2017020746</b>
<b>Luana de Cássia Freitas</b>	<b>2019009541</b>
<b>Lucas Lima Lordello</b>	<b>2019015941</b>
<b>Rolandro Aparecido Corrêa</b>	<b>2020016530</b>

## 1. Diagrama relacional



## 2. Definição de grupos de usuários e suas permissões

- Criar cargo DBA

```
CREATE ROLE dba_yelp;  
GRANT ALL PRIVILEGES ON DATABASE "yelp" TO dba_yelp;  
GRANT ALL ON ALL TABLES IN SCHEMA public TO dba_yelp;
```

- Criar usuário DBA

```
CREATE USER dba WITH PASSWORD '12345';  
GRANT dba_yelp TO dba;
```

- Criar cargo Programador

```
CREATE ROLE programador_yelp;  
GRANT INSERT, SELECT, UPDATE ON TABLE public.avaliacao, public.comercio,  
public.localizacao, public.transacao, public.usuario TO programador_yelp;  
GRANT USAGE ON SCHEMA public TO programador_yelp;
```

- Criar usuários Programador

```
CREATE USER programador1 WITH PASSWORD '12345';  
GRANT programador_yelp TO programador1;  
CREATE USER programador2 WITH PASSWORD '12345';  
GRANT programador_yelp TO programador2;
```

## 3. Definição de índices e suas justificativas

```
CREATE INDEX cidadex ON public.localizacao USING btree (cidade);
```

Ao executar a consulta para buscar os usuários que fizeram avaliação de negócios em Los Angeles (código abaixo), foi observado um aumento de desempenho de aproximadamente 18% no tempo de execução, como demonstram os prints apresentados nesta seção.

```
Explain Analyze SELECT * FROM public.usuario WHERE id IN  
(SELECT id_usuario FROM public.avaliacao WHERE avaliacao.id_comercio IN  
(SELECT id_comercio FROM public.localizacao WHERE cidade = 'Los Angeles'))
```

- Antes da criação do índice – tempo de execução: 2.856 ms

[Data Output](#) [Notifications](#) [Explain](#) [Messages](#)

	QUERY PLAN	
	text	
1	Hash Semi Join (cost=209.84..282.17 rows=693 width=32) (actual time=1.847..2.784 rows=646 loops=1)	
2	[...] Hash Cond: ((usuario.id)::text = (avaliacao.id_usuario)::text)	
3	[...] -> Seq Scan on usuario (cost=0.00..53.43 rows=2943 width=32) (actual time=0.009..0.346 rows=2943 loops=1)	
4	[...] -> Hash (cost=201.18..201.18 rows=693 width=23) (actual time=1.783..1.785 rows=693 loops=1)	
5	[...] Buckets: 1024 Batches: 1 Memory Usage: 46kB	
6	[...] -> Hash Semi Join (cost=31.18..201.18 rows=693 width=23) (actual time=0.773..1.615 rows=693 loops=1)	
7	[...] Hash Cond: ((avaliacao.id_comercio)::text = (localizacao.id_comercio)::text)	
8	[...] -> Seq Scan on avaliacao (cost=0.00..153.29 rows=3429 width=46) (actual time=0.004..0.558 rows=3429 loops=1)	
9	[...] -> Hash (cost=28.29..28.29 rows=231 width=23) (actual time=0.296..0.296 rows=231 loops=1)	
10	[...] Buckets: 1024 Batches: 1 Memory Usage: 21kB	
11	[...] -> Seq Scan on localizacao (cost=0.00..28.29 rows=231 width=23) (actual time=0.084..0.239 rows=231 loops=1)	
12	[...] Filter: ((cidade)::text = 'Los Angeles'::text)	
13	[...] Rows Removed by Filter: 912	
14	Planning Time: 0.386 ms	
15	Execution Time: 2.856 ms	

- Depois da criação do índice – tempo de execução: 2.389 ms

[Query Editor](#) [Query History](#) [Data Output](#) [Notifications](#) [Explain](#) [Messages](#)

	QUERY PLAN	
	text	
1	Hash Semi Join (cost=208.51..280.84 rows=693 width=32) (actual time=1.539..2.328 rows=646 loops=1)	
2	[...] Hash Cond: ((usuario.id)::text = (avaliacao.id_usuario)::text)	
3	[...] -> Seq Scan on usuario (cost=0.00..53.43 rows=2943 width=32) (actual time=0.007..0.275 rows=2943 loops=1)	
4	[...] -> Hash (cost=199.84..199.84 rows=693 width=23) (actual time=1.486..1.488 rows=693 loops=1)	
5	[...] Buckets: 1024 Batches: 1 Memory Usage: 46kB	
6	[...] -> Hash Semi Join (cost=29.84..199.84 rows=693 width=23) (actual time=0.496..1.318 rows=693 loops=1)	
7	[...] Hash Cond: ((avaliacao.id_comercio)::text = (localizacao.id_comercio)::text)	
8	[...] -> Seq Scan on avaliacao (cost=0.00..153.29 rows=3429 width=46) (actual time=0.004..0.517 rows=3429 loops=1)	
9	[...] -> Hash (cost=26.96..26.96 rows=231 width=23) (actual time=0.154..0.155 rows=231 loops=1)	
10	[...] Buckets: 1024 Batches: 1 Memory Usage: 21kB	
11	[...] -> Bitmap Heap Scan on localizacao (cost=10.07..26.96 rows=231 width=23) (actual time=0.080..0.112 rows=231 l...	
12	[...] Recheck Cond: ((cidade)::text = 'Los Angeles'::text)	
13	[...] Heap Blocks: exact=4	
14	[...] -> Bitmap Index Scan on cidadex (cost=0.00..10.01 rows=231 width=0) (actual time=0.073..0.073 rows=231 loops=1)	
15	[...] Index Cond: ((cidade)::text = 'Los Angeles'::text)	
16	Planning Time: 0.694 ms	
17	Execution Time: 2.389 ms	

## 4. Definição de views

Foi criada uma view que exibe os dez melhores comércios de acordo com o valor médio de avaliações dos usuários, visto ser uma consulta pertinente para visualização em uma aplicação web em um contexto onde essa aplicação usaria o banco de dados.

```
CREATE OR REPLACE VIEW "10 Melhores Comercios por Avaliacao" AS
SELECT comercio.nome, comercio.preco, avg(avaliacao.nota)
FROM public.comercio INNER JOIN public.avaliacao
ON comercio.id = avaliacao.id_comercio
GROUP BY comercio.nome, comercio.preco
ORDER BY avg(avaliacao.nota) DESC
LIMIT 10;
```

Como resultado da consulta obtém-se o nome dos comércios, seu preço e a avaliação média.

```
SELECT * FROM "10 Melhores Comercios por Avaliacao";
```

	Data Output	Notifications	Messages	Explain
	nome character varying (255)	preco character varying (4)	avg numeric	
1	Fisherman's Outlet	\$\$	5.0000000000000000	
2	Eight Korean BBQ	\$\$	5.0000000000000000	
3	Nini's Deli	\$	5.0000000000000000	
4	Fat Ducks Deli & Bakery	\$	5.0000000000000000	
5	OBAO	\$\$	5.0000000000000000	
6	Flub A Dub Chub's	\$	5.0000000000000000	
7	Dirt Dog	\$\$	5.0000000000000000	
8	Fogo de Chão	\$\$\$	5.0000000000000000	
9	Lady Yum	\$\$	5.0000000000000000	
10	La Nonna	\$\$	5.0000000000000000	

## 5. Triggers

Esse trigger tem como propósito fazer uma chamada à função quant\_usuarios() após a remoção de algum usuário para informar a quantidade de usuários ainda registrados no banco de dados.

```
CREATE OR REPLACE FUNCTION quant_usuarios()
RETURNS TRIGGER LANGUAGE plpgsql
AS $$
BEGIN
    SELECT count(*) FROM usuario;
END;
```

```

$$;
CREATE TRIGGER confere_usuarios
    AFTER DELETE ON usuario
    FOR EACH ROW
    EXECUTE PROCEDURE quant_usuarios();

```

## 6. Procedures

Procedure criado com objetivo de inserir um usuário e uma avaliação do mesmo ao informar todos os valores necessários para tal nos parâmetros de `insereUsuario()`.

```

CREATE OR REPLACE PROCEDURE insereUsuario(id_com character varying(22),
id_user character varying(22), id_aval character varying(22), stars int,
comentario text, dia date, hora time, username character varying(22))
LANGUAGE SQL
AS $$
    INSERT INTO usuario VALUES (id_user, username);
    INSERT INTO avaliacao VALUES (id_com, id_user, id_aval, stars,
comentario, dia, hora);
$$;

```

## 7. Funções

Essa função realiza a inserção dos dados de um comércio nas tabelas comércio e localização, visto que as tabelas são relacionadas de modo que um registro de localização depende da existência do comércio correspondente no banco de dados.

```

CREATE OR REPLACE FUNCTION insereComercio(character varying(22), character
varying(255), boolean, character varying(20), character varying(4), character
varying(255), character varying(255), character varying(255), int, int,
character varying(20), character varying(255), character varying(255),
character varying(255), character varying(255))
RETURNS bool AS
$BODY$
    DECLARE
        id_comercio alias for $1;
        nome alias for $2;
        fechado alias for $3;
        telefone alias for $4;
        preco alias for $5;
        pseudo alias for $6;
        titulo_cat alias for $7;
        pseudo_cat alias for $8;
        quant_aval alias for $9;
        id_local alias for $10;
        cod_post alias for $11;
        pais alias for $12;
        estado alias for $13;
        city alias for $14;
        lograd alias for $15;
        ok bool;
BEGIN



```

```

        if lograd IN (SELECT logradouro FROM localizacao)
        AND city IN (SELECT cidade FROM localizacao WHERE logradouro =
lograd) then
            RAISE NOTICE 'Localização já existente.';
            ok = false;
        else
            INSERT INTO comercio VALUES (id_comercio, nome, fechado,
telefone, preco, pseudo, titulo_cat, pseudo_cat, quant_aval);
            INSERT INTO localizacao VALUES (id_comercio, id_local,
cod_post, pais, estado, city, lograd);
            RAISE NOTICE 'Comércio e localização inseridos com sucesso.';
            ok = true;
        END if;
    RETURN ok;
END;
$BODY$
LANGUAGE 'plpgsql' VOLATILE;

```

## 8. Print com o count das tabelas

Query Editor		Query History		
1	SELECT count(*) FROM public.comercio;			
2				
3				
Data Output		Notifications	Messages	Explain
	count bigint 			
1	1143			

Query Editor		Query History		
1	SELECT count(*) FROM public.localizacao;			
2				
3				
Data Output		Notifications	Messages	Explain
	count bigint			
1	1143			

Query Editor Query History

```
1 SELECT count(*) FROM public.transacao;  
2  
3
```

Data Output Notifications Messages Explain

	count bigint	
1	1813	

Query Editor Query History

```
1 SELECT count(*) FROM public.usuario;  
2  
3
```

Data Output Notifications Messages Explain

	count bigint	
1	2943	

Query Editor Query History

```
1 SELECT count(*) FROM public.avaliacao;  
2  
3
```

Data Output Notifications Messages Explain

	count bigint	
1	3429	



## 9. Resultados dos testes de performance obtidos com o JMeter

Foi verificado anteriormente a ausência de limite de acesso ao banco de dados, conforme abaixo:

Query Editor

Query History

1

2

3

**SELECT** rolname, rolconlimit  
**FROM** pg\_roles;

Data Output

Explain

Messages

Notifications

	rolname name	rolconlimit integer
1	pg_monitor	-1
2	pg_read_all_settings	-1
3	pg_read_all_stats	-1
4	pg_stat_scan_tables	-1
5	pg_read_server_files	-1
6	pg_write_server_files	-1
7	pg_execute_server_program	-1
8	pg_signal_backend	-1
9	postgres	-1

O valor de -1 indica que está ilimitada. O usuário utilizado para os testes foi o padrão postgres. O ideal é sempre limitar o acesso através de um comando. No exemplo, o limite é definido como 5 acessos.

```
ALTER USER nome_do_usuario WITH CONNECTION LIMIT 5;
```

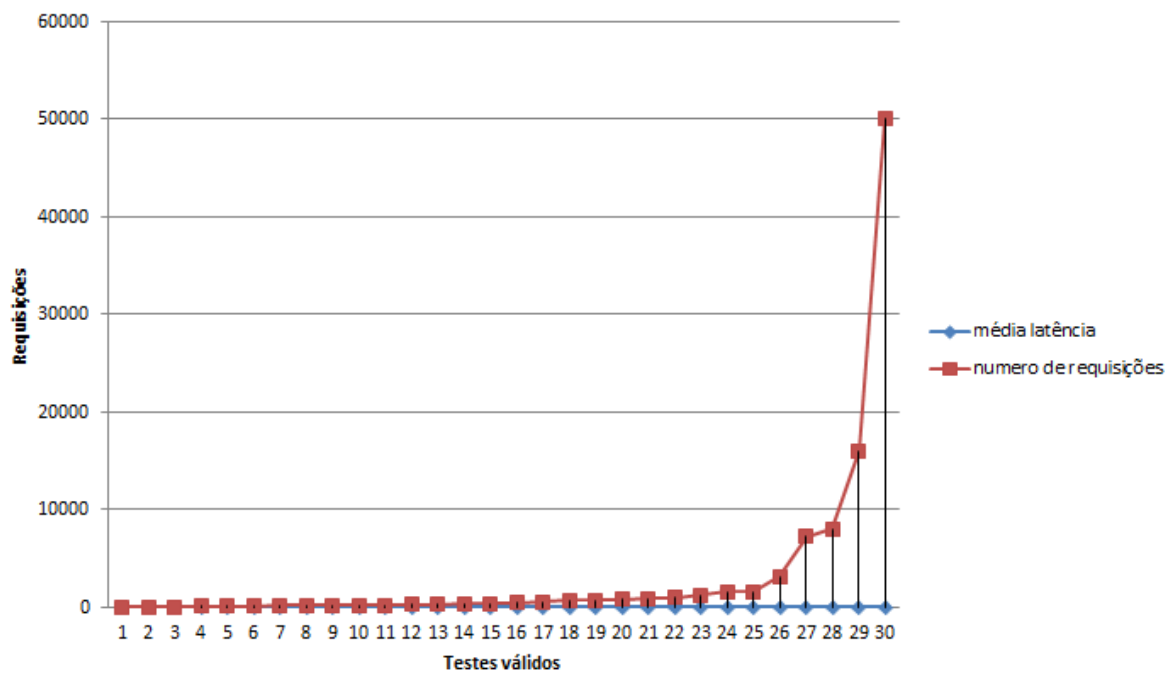
O primeiro teste realizado com JMeter, conforme solicitado, foi manter a quantidade de usuários (threads) como 1 e aumentar gradualmente as requisições até o teste retornar um erro.

teste com 1 usuário	média latência	numero de requisições
1	38,08	25
2	36,32	50
3	38,44	75
4	33,11	100
5	26,5	125
6	25,31	150
7	28,83	175
8	29,29	200
9	26,65	200
10	25,05	225
11	27,65	250
12	27,77	275
13	26,62	300
14	26,08	350
15	26,03	400
16	25,17	500
17	25,33	600
18	22,34	700
19	27,24	750
20	24,24	800
21	25,6	900
22	25,26	1000
23	26,08	1300
24	24,51	1600
25	26,3	1600
26	29,76	3200
27	24,21	7300
28	27,31	8000
29	23,87	16000
30	26,99	50000

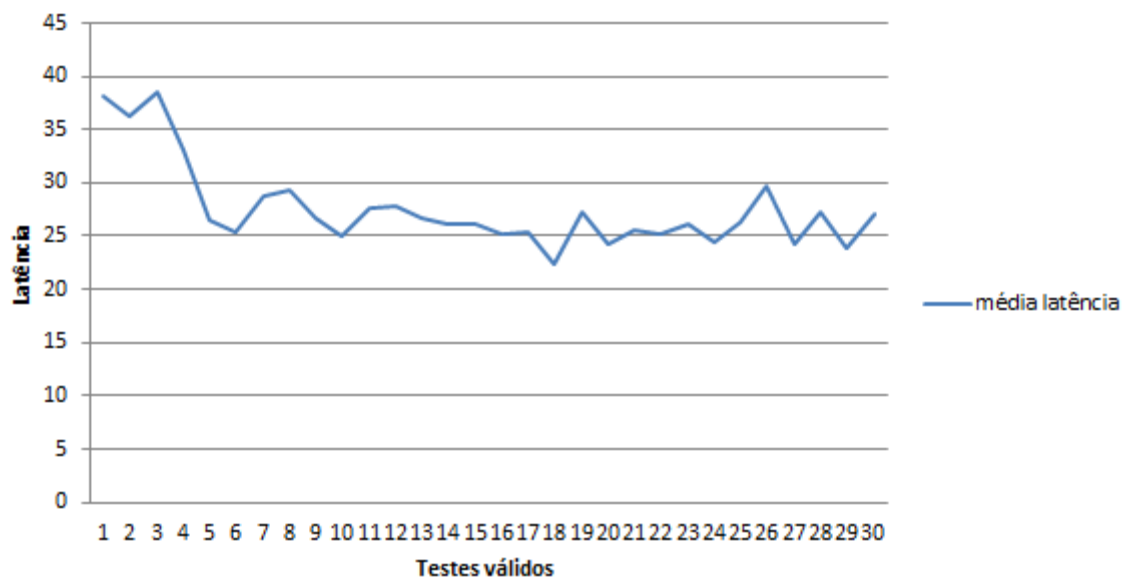
Foram executados 30 testes com um thread e aumento de requisições gradativo e manual. Pode-se constatar que quando se aumentam as requisições ao banco de dados, a latência – o tempo, em milissegundos, decorrido entre o momento em JMeter enviou o pedido e quando uma resposta inicial foi recebida – ficou estável entre 20 e 35 ms.

No gráfico 1, a média da latência aparece como uma linha muito próxima de zero, devido à disposição dos dados; já no gráfico 2, foi dada ênfase à latência alcançada em cada teste.

**Gráfico 1 - Latência por Número de Requisições**



**Gráfico 2 - Média de Latência**

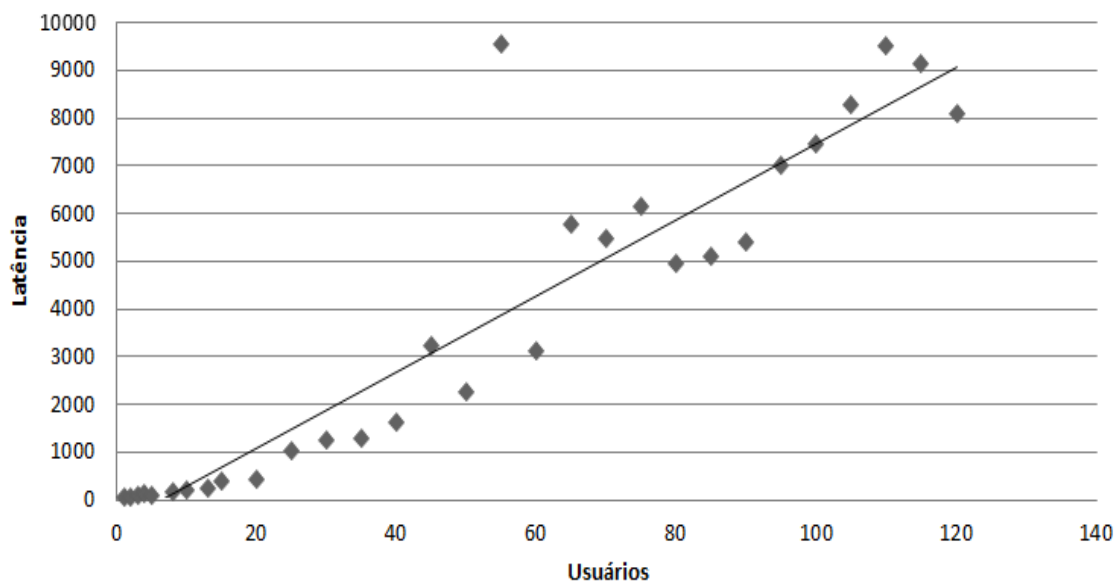


Muitas vezes, os erros ocorriam ao chegar próximo de 50.000 requisições, porém a máquina travava.

O segundo teste realizado foi definir uma quantidade de requisições fixa e aumentar a quantidade de usuários até o retorno de um erro. Esse teste determina o número máximo de usuários suportados pelo banco para essa consulta.

usuários	média latência	numero de requisições
1	24,21	5000
2	39,38	5000
3	60,81	5000
4	91,55	5000
5	80,59	5000
8	135,76	5000
10	165,9	5000
13	229,65	5000
15	356,13	5000
20	420,35	5000
25	997,06	5000
30	1209,15	5000
35	1244,58	5000
40	1616,57	5000
45	3192,4	5000
50	2229,01	5000
55	9508	5000
60	3104,56	5000
65	5742,43	5000
70	5438,09	5000
75	6127,06	5000
80	4928,43	5000
85	5078,98	5000
90	5365,13	5000
95	6987,56	5000
100	7412,56	5000
105	8263,98	5000
110	9491,19	5000
115	9127,98	5000
120	8064,69	5000

**Gráfico 3 - Latência por Número de Usuários**



Conforme os usuários aumentam, a latência aumenta proporcionalmente; no gráfico 3, nota-se que os dados acompanham uma linha de tendência.

**Gráfico 4 - Variação da Latência por Número de Requisições**

