

UNIVERSIDADE DE SÃO PAULO

SCC5789 - BASES DE DADOS I

---

# Processamento e Otimização de Consultas com Indexação

---

*Alunos:*

Henrique G. Zanin

## Sumário

<b>1</b>	<b>Ambiente de testes - PostgreSQL</b>	<b>3</b>
<b>2</b>	<b>Exercício 1</b>	<b>3</b>
2.1	Item (a) . . . . .	4
2.2	Item (b) . . . . .	4
<b>3</b>	<b>Exercício 2</b>	<b>8</b>
3.1	Item (a) . . . . .	8
3.2	Item (b) . . . . .	9
3.3	Adicional . . . . .	15
3.4	Item (C) . . . . .	15
<b>4</b>	<b>Exercício 3</b>	<b>15</b>
<b>5</b>	<b>Exercício 4</b>	<b>16</b>
5.1	Item (a) . . . . .	16
5.2	Item (b) . . . . .	17
5.3	Item (c) . . . . .	22
<b>6</b>	<b>Exercício 5</b>	<b>23</b>
6.1	Item (a) . . . . .	23
6.2	Item (b) . . . . .	23
6.3	Item (b) . . . . .	25
6.4	Item (c) . . . . .	26
<b>7</b>	<b>Exercício 6</b>	<b>26</b>
7.1	Item (a) . . . . .	26
7.2	Item (b) . . . . .	26
7.3	Adicional . . . . .	27
7.4	Item (c) . . . . .	28
7.5	Item (d) . . . . .	28
<b>8</b>	<b>Exercício 7</b>	<b>31</b>
8.1	Item (a) . . . . .	31
8.2	Item (b) . . . . .	31
8.3	Item (c) . . . . .	33
8.4	Item (d) . . . . .	33
8.5	Item (e) . . . . .	33
<b>9</b>	<b>Exercício 8</b>	<b>34</b>

9.1	Item (a)	. . . . .	34
9.2	Item (b)	. . . . .	36
<b>10</b>	<b>Apêndice</b>		<b>36</b>

## 1 Ambiente de testes - PostgreSQL

A implementação foi realizada por meio de contêineres Docker. Utilizamos o Docker Compose para facilitar a criação do ambiente de testes. Todos os artefatos necessários para instanciar o ambiente encontram-se anexos a esse documento, assim como disponibilizamos o `compose.yaml` no Apêndice 10.

Os passos necessários para instalar o Docker em ambientes Linux, macOS e Windows podem ser encontrados no site do projeto Docker<sup>1</sup>. Após instalado o Docker Engine a inicialização do ambiente é dada pelo comando abaixo dentro do diretório que contém o arquivo `compose.yaml`.

```
1 docker compose up -d
```

O arquivo de configuração `compose.yaml`<sup>10</sup> mapeia a porta 5432 do PostgreSQL em execução no Docker para a porta local 5432, dessa forma é possível acessar o PostgreSQL por meio de qualquer ferramenta que estabeleça o socket de conexão com o banco no endereço `localhost:5432`.

As credenciais de acesso ao banco e o banco definido para a realização dos testes são:

```
1 usuario: postgres
2 senha: postgres
3 banco: bd1
4 ip: localhost
5 porta: 5432
```

Para desabilitar e parar os serviços em execução no Docker, execute:

```
1 docker compose down
```

## 2 Exercício 1

Ao especificar os comandos de inserção de dados listados nesta lista de exercícios, a chave primária das tabelas `clube` e `jogador` foi especificada de forma manual. Por exemplo, o primeiro clube tem chave primária 1, o segundo clube tem chave primária 2, e assim sucessivamente. Ao invés de especificar a chave primária de forma manual, é mais indicado utilizar um comando específico para que a chave primária seja definida de forma automática. Por exemplo, no SGBD Oracle, tem-se o comando `CREATE SEQUENCE`

---

<sup>1</sup><https://docs.docker.com/engine/install/>

## 2.1 Item (a)

De acordo com o SGBD escolhido para resolver a lista de exercícios, defina e explique qual o comando para a definição automática da chave primária.

**Resposta** O SGBD escolhido para a realização das atividades propostas foi o PostgreSQL. Para definição de uma chave primária sequencial utiliza-se o pseudotipo SERIAL associado a coluna identificada como PRIMARY KEY. Um exemplo de código SQL para PostgreSQL que define automaticamente uma chave primária de acordo com as inserções é dado abaixo:

```
1 CREATE TABLE exemplo_serial(  
2     id SERIAL PRIMARY KEY  
3 );  
4
```

Na prática, o PostgreSQL cria uma SEQUENCE automaticamente e a atribui à coluna indicada no tipo SERIAL. Devido a isso o tipo SERIAL é considerado um pseudotipo.

## 2.2 Item (b)

Explique quais as vantagens de se utilizar esse comando.

**Resposta** Esse comando/pseudotipo tem como vantagem operacional de evitar que o cliente controle os id's disponíveis para alocação. Caso não haja um controle interno o cliente deve sempre consultar na tabela qual é o último id alocado. O caso fica mais complicado se o cliente alocar aleatoriamente um identificador, exigindo a descoberta de quais são os id's disponíveis.

Do ponto de vista de desempenho podemos considerar dois casos base em duas tabelas distintas: uma com os id's inseridos de forma sequencial e outra de forma aleatória. O primeiro cenário envolve uma consulta com cláusula where buscando todos os id's menores que 10.000 em uma coluna sem índice. O segundo envolve a mesma consulta, porém com um índice na coluna id nas duas tabelas. Vamos às definições da tabela de testes:

```
1 CREATE SCHEMA testes;  
2 CREATE TABLE testes.t_test (id serial, name text);  
3 INSERT INTO testes.t_test (name) SELECT 'Nome sobrenome' FROM  
   generate_series(1, 4000000);  
4 CREATE TABLE testes.t_random AS SELECT * FROM testes.t_test ORDER  
   BY random();  
5
```

O comando abaixo executa um `SELECT *` para todas os id's menores que 10.000 na tabela onde os valores foram inseridos sequencialmente e exibe o plano de execução da consulta.

```
1  EXPLAIN (analyze true, buffers true, timing true)
2  select * from testes.t_test where id < 10000;
3
```

#### Consulta sem índice com valores sequenciais

```
1  Gather (cost=1000.00..44432.73 rows=9774 width=9) (actual
   time=0.694..513.493 rows=9999 loops=1)
2  Workers Planned: 2
3  Workers Launched: 2
4  Buffers: shared hit=12906 read=8716
5  -> Parallel Seq Scan on t_test (cost=0.00..42455.33 rows=4072
   width=9) (actual time=329.891..495.796 rows=3333 loops=3)
6      Filter: (id < 10000)
7      Rows Removed by Filter: 1330000
8      Buffers: shared hit=12906 read=8716
9  Planning Time: 0.088 ms
10 Execution Time: 163.413 ms
11
```

A próxima consulta possui o mesmo procedimento que o anterior, porém é executado na tabela onde os id's foram inseridos de forma aleatória.

```
1  EXPLAIN (analyze true, buffers true, timing true)
2  select * from testes.t_random where id < 10000
3
```

#### Consulta sem índice com valores aleatórios

```
1  Gather (cost=1000.00..44439.63 rows=9843 width=9) (actual
   time=0.964..136.490 rows=9999 loops=1)
2  Workers Planned: 2
3  Workers Launched: 2
4  Buffers: shared hit=3169 read=18453
5  -> Parallel Seq Scan on t_random (cost=0.00..42455.33 rows=4101
   width=9) (actual time=0.104..105.744 rows=3333 loops=3)
6      Filter: (id < 10000)
7      Rows Removed by Filter: 1330000
8      Buffers: shared hit=3169 read=18453
9  Planning Time: 0.079 ms
10 Execution Time: 136.838 ms
```

11

Os tempos de execução devem ser desconsiderados nos dois experimentos acima, para garantia de validade dos resultados deve ser definida uma amostra e calcular o tempo médio de execução. Nos concentraremos no custo de execução calculado pelo PostgreSQL. Nos dois cenários o plano de execução é o mesmo, um Scan sequencial paralelizado. Vemos também que o custo estimado é o mesmo, já que nesse caso não há indicação de que os dados são ordenados em nenhuma das tabelas. Conclui-se que não há mudanças de planos de execução nem de estimativa de custos nos dois cenários.

O próximo caso consiste na definição de um índice na coluna id nas duas tabelas. Será mantido a mesma consulta para as duas tabelas a fim de verificar se há impacto no custo estimado pelo otimizador. O próximo quadro exibe o plano de execução, os custos estimados e o uso de buffers para um SELECT no índice ordenado e no índice aleatório.

#### Índice sequencial

```

1  Index Scan using idx_id_seq on t_test (cost=0.43..337.26 rows=9819
width=9) (actual time=0.095..3.570 rows=9999 loops=1)
2      Index Cond: (id < 10000)
3      Buffers: shared hit=3 read=82
4  Planning:
5      Buffers: shared hit=16 read=4
6  Planning Time: 0.943 ms
7  Execution Time: 4.415 ms
8

```

#### Índice aleatoriamente ordenado

```

1  Bitmap Heap Scan on t_random (cost=189.29..17800.22 rows=9917
width=9) (actual time=5.471..30.030 rows=9999 loops=1)
2      Recheck Cond: (id < 10000)
3      Heap Blocks: exact=7983
4      Buffers: shared hit=1203 read=6810
5      -> Bitmap Index Scan on idx_id (cost=0.00..186.81 rows=9917
width=0) (actual time=4.486..4.486 rows=9999 loops=1)
6          Index Cond: (id < 10000)
7          Buffers: shared hit=3 read=27
8  Planning:
9      Buffers: shared hit=16 read=4

```

```
10 Planning Time: 0.950 ms
11 Execution Time: 30.435 ms
12
```

Os resultados obtidos quando a consulta envolve um índice são significativamente diferentes quando o índice é ordenado sequencialmente se comparados a um índice aleatoriamente distribuído. No índice ordenado observa-se que o otimizador estima um custo máximo de 337.26 unidades e utiliza apenas o Index Scan para realizar a consulta. No índice distribuído aleatoriamente o plano apresenta duas etapas: primeiro um Bitmap Index Scan e posteriormente um Bitmap Heap Scan. O custo estimado no segundo caso possui o valor de 17800.22 sendo 52,77 vezes superior ao do primeiro caso. O tempo de execução, quando comparados, também são significativamente diferentes, com o segundo caso sendo 6,89 vezes superior.

A discussão do resultado obtido deriva do entendimento de como os dados estão armazenados em disco. No primeiro caso, onde o índice é sequencial, os valores recuperados do disco estão dispostos sequencialmente, dispensando acessos aleatórios ao disco para recuperar valores contíguos. Quando os dados estão dispostos aleatoriamente exige-se o mapeamento dos blocos de disco necessários para recuperar as informações, etapa essa realizada no Bitmap Index Scan. O passo seguinte é acessar aleatoriamente os blocos mapeados na etapa anterior e verificar quais linhas atendem as condições da cláusula WHERE.

O otimizador do PostgreSQL mantém estatísticas acerca das colunas e tabelas existentes no banco. Para calcular o plano de execução são levadas em consideração algumas estatísticas, uma delas extremamente pertinente ao caso é a correlação entre os valores de uma coluna. Um valor próximo a 1 indica que os valores seguintes esperados são correlacionados com o atual. Por sua vez valores próximos a 0 indicam pouca correlação. Analisando as estatísticas armazenadas para o caso de estudo podemos observar a diferença entre os dois casos.

#### Consulta das estatísticas de correlação

```
1 SELECT tablename, attname, correlation
2 FROM pg_stats
3 WHERE tablename IN ('t_test', 't_random') and attname = 'id'
4 ORDER BY 1, 2;
5
```

Fica constatado, a partir da análise do plano de execução e das estatísticas armazenadas, que a definição de uma chave sequencial re-



Índice aleatoriamente ordenado

1	tablename		attname		correlation
2	-----	+	-----	+	-----
3	t_random		id		0.0026448716
4	t_test		id		1
5	(2 rows)				
6					

resolve não apenas questões operacionais como aprimoram o desempenho da consulta.

### 3 Exercício 2

Crie uma aplicação de banco de dados utilizando qualquer SGBD relacional, de forma que esse SGBD defina e implemente a aplicação DadosClubesJogadores.

#### 3.1 Item (a)

Crie as tabelas clube, jogador, joga

```

1 CREATE SCHEMA IF NOT EXISTS exercicio ;
2
3 CREATE TABLE if not exists "exercicio".clube (
4     idClube INTEGER NOT NULL,
5     cnpjClube VARCHAR(18),
6     nomeClube VARCHAR(50),
7     apelidoClube VARCHAR(50),
8
9     CONSTRAINT PK_CLUBE
10        PRIMARY KEY (idClube)
11 );
12
13 CREATE TABLE if not exists "exercicio".jogador (
14     idJogador INTEGER NOT NULL,
15     cpfJogador VARCHAR(14),
16     nomeJogador VARCHAR(50),
17     apelidoJogador VARCHAR(50),
18
19     CONSTRAINT PK_MEMBRO
20        PRIMARY KEY (idJogador)
21 );
22
23 CREATE table if not exists "exercicio".joga (

```

```

24 idJogador INTEGER NOT NULL,
25 idClube INTEGER NOT NULL,
26 dataInicioJoga DATE NOT NULL,
27 dataFimJoga DATE,
28 salario NUMERIC(10,2),
29 CONSTRAINT PK_JOGA
30 PRIMARY KEY (idClube, idJogador, dataInicioJoga),
31 CONSTRAINT FK_CLUBE
32 FOREIGN KEY (idClube)
33 REFERENCES "exercicio".clube(idClube)
34 ON DELETE CASCADE,
35 CONSTRAINT FK_JOGADOR
36 FOREIGN KEY (idJogador)
37 REFERENCES "exercicio".jogador(idJogador)
38 ON DELETE cascade
39 );
40

```

### 3.2 Item (b)

Povoe as tabelas clube, jogador, joga. Utilize, para tanto, os comandos de inserção de dados especificados nesta lista de exercícios. Porém, esses comandos devem ser adaptados para utilizarem o comando para a definição automática da chave primária especificado no Exercício 1. Cuidado ao copiar e colar: pode ser que aconteçam erros devido à cópia de caracteres e outros caracteres escondidos. Conserte os erros ao inserir os comandos no script.

```

1 — Insercao de "exercicio".clubes
2 INSERT INTO "exercicio".clube
3 VALUES (1, '60.517.984/0001-04', 'Sao Paulo Futebol Clube', 'Sao Paulo');
4 INSERT INTO "exercicio".clube
5 VALUES (2, '71.856.774/0001-67', 'Clube de Regatas do Flamengo', '
   Flamengo');
6 INSERT INTO "exercicio".clube
7 VALUES (3, '01.978.363/0001-69', 'Sport Club Corinthians Paulista', '
   Corinthians');
8 INSERT INTO "exercicio".clube
9 VALUES (4, '83.442.720/0001-34', 'Club de Regatas Vasco da Gama', 'Vasco'
   );
10 INSERT INTO "exercicio".clube
11 VALUES (5, '42.314.651/0001-04', 'Sociedade Esportiva Palmeiras', '
   Palmeiras');
12 INSERT INTO "exercicio".clube
13 VALUES (6, '42.254.727/0001-45', 'Sport Club Internacional', '
   Internacional');
14 INSERT INTO "exercicio".clube

```

```

15 VALUES (7, '57.535.577/0001-15', 'Gremio Foot-Ball Porto Alegre', '
    Gremio');
16
17 —Insercao de "exercicio".jogadores
18 INSERT INTO "exercicio".jogador
19 VALUES (1, '953.925.565-15', 'Ricardo Gomes Raymundo', 'Ricardo Gomes');
20 INSERT INTO "exercicio".jogador
21 VALUES (2, '335.386.603-52', 'Rogerio Mucke Ceni', 'Rogerio Ceni');
22 INSERT INTO "exercicio".jogador
23 VALUES (3, '852.940.513-70', 'Andre Goncalves Dias', 'Andre Dias');
24 INSERT INTO "exercicio".jogador
25 VALUES (4, '558.444.734-00', 'Jorge Wagner Goes Conceicao', 'Jorge Wagner'
    );
26 INSERT INTO "exercicio".jogador
27 VALUES (5, '164.528.522-72', 'Richarlyson Barbosa Felisbino', 'Richarlyson
    ');
28 INSERT INTO "exercicio".jogador
29 VALUES (6, '865.033.347-88', 'Washington Stecanela Cerqueira', 'Washington
    ');
30 INSERT INTO "exercicio".jogador
31 VALUES (7, '615.145.822-28', 'Dagoberto Pelentier', 'Dagoberto');
32 INSERT INTO "exercicio".jogador
33 VALUES (8, '053.888.708-71', 'Jorge Luis Andrade da Silva', 'Andrade');
34 INSERT INTO "exercicio".jogador
35 VALUES (9, '382.076.804-15', 'Bruno Fernandes das Dores de Souza', 'Bruno'
    );
36 INSERT INTO "exercicio".jogador
37 VALUES (10, '879.184.956-08', 'Ronaldo Simoes Angelim', 'Ronaldo');
38 INSERT INTO "exercicio".jogador
39 VALUES (11, '655.557.182-92', 'Leonardo da Silva Moura', 'Leo Moura');
40 INSERT INTO "exercicio".jogador
41 VALUES (12, '917.987.648-06', 'Juan Maldonado Jaimez Junior', 'Juan');
42 INSERT INTO "exercicio".jogador
43 VALUES (13, '767.273.180-77', 'Ailton Ribeiro Santos', 'Ailton');
44 INSERT INTO "exercicio".jogador
45 VALUES (14, '258.728.121-08', 'Adriano Leite Ribeiro', 'Adriano');
46 INSERT INTO "exercicio".jogador
47 VALUES (15, '959.714.886-27', 'Luis Antonio Venker Menezes', 'Mano Menezes
    ');
48 INSERT INTO "exercicio".jogador
49 VALUES (16, '845.014.566-04', 'Luiz Felipe Ventura dos Santos', 'Felipe');
50 INSERT INTO "exercicio".jogador
51 VALUES (17, '253.175.413-01', 'Anderson Sebastiao Cardoso', 'Chicao');
52 INSERT INTO "exercicio".jogador
53 VALUES (18, '615.881.355-94', 'William Machado de Oliveira', 'William');
54 INSERT INTO "exercicio".jogador
55 VALUES (19, '744.622.839-37', 'Elias Mendes Trindade', 'Elias');
56 INSERT INTO "exercicio".jogador
57 VALUES (20, '188.432.434-70', 'Bruno Ferreira Bonfim', 'Dentinho');

```

```

58 INSERT INTO "exercicio".jogador
59 VALUES (21, '341.538.856-57', 'Ronaldo Luis Nazario de Lima', 'Ronaldo');
60 INSERT INTO "exercicio".jogador
61 VALUES (22, '505.322.412-90', 'Dorival Silvestre Junior', 'Dorival Junior'
);
62 INSERT INTO "exercicio".jogador
63 VALUES (23, '123.736.300-44', 'Fernando Buttenbender Prass', 'Fernando
Prass');
64 INSERT INTO "exercicio".jogador
65 VALUES (24, '856.843.603-00', 'Gian Francesco Goncalves Mariano', 'Gian');
66 INSERT INTO "exercicio".jogador
67 VALUES (25, '530.610.186-03', 'Paulo Sergio Rocha', 'Paulo Sergio');
68 INSERT INTO "exercicio".jogador
69 VALUES (26, '870.788.636-58', 'Carlos Rafael do Amaral', 'Amaral');
70 INSERT INTO "exercicio".jogador
71 VALUES (27, '616.736.022-74', 'Jeferson Rodrigues Goncalves', 'Jeferson');
72 INSERT INTO "exercicio".jogador
73 VALUES (28, '192.616.143-29', 'Rodrigo Pimpao Vianna', 'Rodrigo Pimpao');
74 INSERT INTO "exercicio".jogador
75 VALUES (29, '481.228.706-51', 'Muricy Ramalho', 'Muricy Ramalho');
76 INSERT INTO "exercicio".jogador
77 VALUES (30, '415.324.323-05', 'Marcos Roberto Silveira Reis', 'Marcos');
78 INSERT INTO "exercicio".jogador
79 VALUES (31, '087.787.064-08', 'Danilo Larangera', 'Danilo');
80 INSERT INTO "exercicio".jogador
81 VALUES (32, '078.792.754-60', 'Sandro Laurindo da Silva', 'Sandro Silva');
82 INSERT INTO "exercicio".jogador
83 VALUES (33, '116.549.963-00', 'Jose Edmilson Gomes Moraes', 'Edmilson');
84 INSERT INTO "exercicio".jogador
85 VALUES (34, '106.598.386-72', 'Williams dos Santos Santana', 'Williams');
86 INSERT INTO "exercicio".jogador
87 VALUES (35, '757.494.184-05', 'Marcos Antonio da Silva Goncalves', '
Marquinhos');
88 INSERT INTO "exercicio".jogador
89 VALUES (36, '873.411.664-87', 'Adenor Leonardo Bacchi', 'Tite');
90 INSERT INTO "exercicio".jogador
91 VALUES (37, '222.722.710-90', 'Lauro Junior Batista da Cruz', 'Lauro');
92 INSERT INTO "exercicio".jogador
93 VALUES (38, '267.128.431-46', 'Fabian Guedes', 'Bolivar');
94 INSERT INTO "exercicio".jogador
95 VALUES (39, '504.862.542-08', 'Fabiano Eller dos Santos', 'Fabiano Eller'
);
96 INSERT INTO "exercicio".jogador
97 VALUES (40, '211.836.647-70', 'Andre Luiz Tavares', 'Andrezinho');
98 INSERT INTO "exercicio".jogador
99 VALUES (41, '243.285.875-10', 'Giuliano Victor de Paula', 'Giuliano');
100 INSERT INTO "exercicio".jogador
101 VALUES (42, '920.811.027-30', 'Alecsandro Barbosa Felisbino', 'Alecsandro'
);

```

```

102 INSERT INTO "exercicio".jogador
103 VALUES (43, '227.941.415-54', 'Paulo Autuori de Mello', 'Paulo Autuori');
104 INSERT INTO "exercicio".jogador
105 VALUES (44, '233.338.375-52', 'Victor Leandro Bagy', 'Victor');
106 INSERT INTO "exercicio".jogador
107 VALUES (45, '184.872.484-54', 'Leonardo Renan Simoes de Lacerda', 'Leo');
108 INSERT INTO "exercicio".jogador
109 VALUES (46, '253.788.331-40', 'Fabio Santos Romeu', 'Fabio Santos');
110 INSERT INTO "exercicio".jogador
111 VALUES (47, '143.688.572-83', 'Anderson Simas Luciano', 'Tcheco');
112 INSERT INTO "exercicio".jogador
113 VALUES (48, '888.342.876-57', 'Leandro dos Santos de Jesus', 'Makelele');
114 INSERT INTO "exercicio".jogador
115 VALUES (49, '180.836.554-22', 'Jonas Goncalves Oliveira', 'Jonas');
116
117 —Insercao de Joga
118 INSERT INTO "exercicio".joga
119 VALUES (2,1,TO_DATE('07.09.1990', 'DD.MM.YYYY'), NULL, 300000.00);
120 INSERT INTO "exercicio".joga
121 values (3,1,TO_DATE('01.02.2006', 'DD.MM.YYYY'), TO_DATE('03.07.2011', 'DD
122 .MM.YYYY'), 100000.00);
123 INSERT INTO "exercicio".joga
124 values (4,1,TO_DATE('01.12.2007', 'DD.MM.YYYY'), TO_DATE('03.08.2011', 'DD
125 .MM.YYYY'), 120000.00);
126 INSERT INTO "exercicio".joga
127 values (5,1,TO_DATE('01.07.2005', 'DD.MM.YYYY'), TO_DATE('13.02.2011', 'DD
128 .MM.YYYY'), 110000.00);
129 INSERT INTO "exercicio".joga
130 values (6,1,TO_DATE('01.12.2008', 'DD.MM.YYYY'), TO_DATE('18.07.2010', 'DD
131 .MM.YYYY'), 90000.00);
132 INSERT INTO "exercicio".joga
133 values (7,1,TO_DATE('01.04.2007', 'DD.MM.YYYY'), TO_DATE('23.06.2011', 'DD
134 .MM.YYYY'), 130000.00);
135 INSERT INTO "exercicio".joga
136 values (9,2,TO_DATE('07.09.2006', 'DD.MM.YYYY'), TO_DATE('23.08.2011', 'DD
137 .MM.YYYY'), 655000.00);
138 INSERT INTO "exercicio".joga
139 values (10,2,TO_DATE('07.04.2006', 'DD.MM.YYYY'), TO_DATE('04.02.2011', '
140 DD.MM.YYYY'), 614000.00);
141 INSERT INTO "exercicio".joga
142 values (11,2,TO_DATE('07.03.2005', 'DD.MM.YYYY'), TO_DATE('19.08.2010', '
143 DD.MM.YYYY'), 541000.00);
144 INSERT INTO "exercicio".joga
145 values (12,2,TO_DATE('07.02.2006', 'DD.MM.YYYY'), TO_DATE('01.07.2013', '
146 DD.MM.YYYY'), 581000.00);
147 INSERT INTO "exercicio".joga
148 values (13,2,TO_DATE('07.09.2008', 'DD.MM.YYYY'), TO_DATE('19.05.2013', '
149 DD.MM.YYYY'), 768000.00);
150 INSERT INTO "exercicio".joga

```

```
141 values (14,2,TO_DATE( '06.05.2009' , 'DD.MM.YYYY' ),TO_DATE( '16.12.2010' , '  
    DD.MM.YYYY' ),712000.00);  
142 INSERT INTO "exercicio".joga  
143 values (16,3,TO_DATE( '07.09.2007' , 'DD.MM.YYYY' ),TO_DATE( '21.06.2011' , '  
    DD.MM.YYYY' ),354000.00);  
144 INSERT INTO "exercicio".joga  
145 values (17,3,TO_DATE( '07.01.2008' , 'DD.MM.YYYY' ),TO_DATE( '04.03.2010' , '  
    DD.MM.YYYY' ),687000.00);  
146 INSERT INTO "exercicio".joga  
147 values (18,3,TO_DATE( '27.03.2008' , 'DD.MM.YYYY' ),TO_DATE( '20.11.2010' , '  
    DD.MM.YYYY' ),741000.00);  
148 INSERT INTO "exercicio".joga  
149 values (19,3,TO_DATE( '07.02.2006' , 'DD.MM.YYYY' ),TO_DATE( '09.06.2011' , '  
    DD.MM.YYYY' ),687000.00);  
150 INSERT INTO "exercicio".joga  
151 values (20,3,TO_DATE( '07.06.2007' , 'DD.MM.YYYY' ),TO_DATE( '25.06.2010' , '  
    DD.MM.YYYY' ),412000.00);  
152 INSERT INTO "exercicio".joga  
153 values (21,3,TO_DATE( '09.12.2008' , 'DD.MM.YYYY' ),TO_DATE( '05.09.2010' , '  
    DD.MM.YYYY' ),1268000.00);  
154 INSERT INTO "exercicio".joga  
155 values (23,4,TO_DATE( '27.03.2009' , 'DD.MM.YYYY' ),TO_DATE( '05.12.2010' , '  
    DD.MM.YYYY' ),768000.00);  
156 INSERT INTO "exercicio".joga  
157 values (24,4,TO_DATE( '12.04.2009' , 'DD.MM.YYYY' ),TO_DATE( '21.12.2010' , '  
    DD.MM.YYYY' ),748000.00);  
158 INSERT INTO "exercicio".joga  
159 values (25,4,TO_DATE( '16.12.2008' , 'DD.MM.YYYY' ),TO_DATE( '16.12.2010' , '  
    DD.MM.YYYY' ),517000.00);  
160 INSERT INTO "exercicio".joga  
161 values (26,4,TO_DATE( '08.12.2008' , 'DD.MM.YYYY' ),TO_DATE( '24.11.2010' , '  
    DD.MM.YYYY' ),871000.00);  
162 INSERT INTO "exercicio".joga  
163 values (27,4,TO_DATE( '09.01.2009' , 'DD.MM.YYYY' ),TO_DATE( '28.08.2011' , '  
    DD.MM.YYYY' ),873000.00);  
164 INSERT INTO "exercicio".joga  
165 values (28,4,TO_DATE( '06.03.2009' , 'DD.MM.YYYY' ),TO_DATE( '23.11.2010' , '  
    DD.MM.YYYY' ),682000.00);  
166 INSERT INTO "exercicio".joga  
167 values (30,5,TO_DATE( '14.03.1992' , 'DD.MM.YYYY' ),TO_DATE( '05.01.2011' , '  
    DD.MM.YYYY' ),768000.00);  
168 INSERT INTO "exercicio".joga  
169 values (31,5,TO_DATE( '07.04.2009' , 'DD.MM.YYYY' ),TO_DATE( '20.07.2010' , '  
    DD.MM.YYYY' ),748000.00);  
170 INSERT INTO "exercicio".joga  
171 values (32,5,TO_DATE( '07.03.2009' , 'DD.MM.YYYY' ),TO_DATE( '20.03.2011' , '  
    DD.MM.YYYY' ),517000.00);  
172 INSERT INTO "exercicio".joga
```

```
173 values (33,5,TO_DATE( '23.01.2009' , 'DD.MM.YYYY' ),TO_DATE( '11.07.2011' , '
    DD.MM.YYYY' ),871000.00);
174 INSERT INTO "exercicio".joga
175 values (34,5,TO_DATE( '05.01.2008' , 'DD.MM.YYYY' ),TO_DATE( '25.05.2011' , '
    DD.MM.YYYY' ),873000.00);
176 INSERT INTO "exercicio".joga
177 values (35,5,TO_DATE( '11.03.2009' , 'DD.MM.YYYY' ),TO_DATE( '12.11.2010' , '
    DD.MM.YYYY' ),682000.00);
178 INSERT INTO "exercicio".joga
179 values (37,6,TO_DATE( '07.09.2008' , 'DD.MM.YYYY' ),TO_DATE( '26.05.2011' , '
    DD.MM.YYYY' ),741000.00);
180 INSERT INTO "exercicio".joga
181 values (38,6,TO_DATE( '17.06.2009' , 'DD.MM.YYYY' ),TO_DATE( '09.02.2011' , '
    DD.MM.YYYY' ),358000.00);
182 INSERT INTO "exercicio".joga
183 values (39,6,TO_DATE( '08.08.2009' , 'DD.MM.YYYY' ),TO_DATE( '08.10.2010' , '
    DD.MM.YYYY' ),787000.00);
184 INSERT INTO "exercicio".joga
185 values (40,6,TO_DATE( '13.02.2008' , 'DD.MM.YYYY' ),TO_DATE( '21.03.2011' , '
    DD.MM.YYYY' ),186000.00);
186 INSERT INTO "exercicio".joga
187 values (41,6,TO_DATE( '12.04.2009' , 'DD.MM.YYYY' ),TO_DATE( '22.05.2011' , '
    DD.MM.YYYY' ),738000.00);
188 INSERT INTO "exercicio".joga
189 values (42,6,TO_DATE( '21.01.2009' , 'DD.MM.YYYY' ),TO_DATE( '03.03.2010' , '
    DD.MM.YYYY' ),617000.00);
190 INSERT INTO "exercicio".joga
191 values (44,7,TO_DATE( '02.02.2008' , 'DD.MM.YYYY' ),TO_DATE( '15.07.2010' , '
    DD.MM.YYYY' ),74000.00);
192 INSERT INTO "exercicio".joga
193 values (45,7,TO_DATE( '28.04.2009' , 'DD.MM.YYYY' ),TO_DATE( '09.05.2011' , '
    DD.MM.YYYY' ),351000.00);
194 INSERT INTO "exercicio".joga
195 VALUES(46,7,TO_DATE( '18.01.2009' , 'DD.MM.YYYY' ),TO_DATE( '07.04.2011' , 'DD
    .MM.YYYY' ),342000.00);
196 INSERT INTO "exercicio".joga
197 values (47,7,TO_DATE( '10.07.2008' , 'DD.MM.YYYY' ),TO_DATE( '01.06.2010' , '
    DD.MM.YYYY' ),536000.00);
198 INSERT INTO "exercicio".joga
199 values (48,7,TO_DATE( '26.02.2008' , 'DD.MM.YYYY' ),TO_DATE( '25.07.2011' , '
    DD.MM.YYYY' ),827000.00);
200 INSERT INTO "exercicio".joga
201 values (49,7,TO_DATE( '12.01.2009' , 'DD.MM.YYYY' ),TO_DATE( '27.04.2011' , '
    DD.MM.YYYY' ),863000.00);
202
203
```

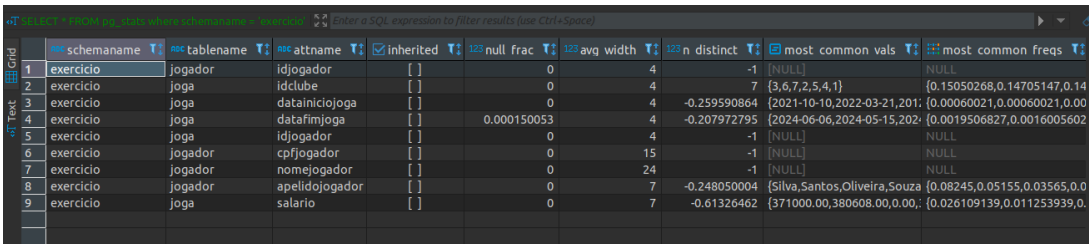
3.3 Adicional

A inserções realizadas no item B não são suficientes para a criação de estatísticas relacionadas às tabelas. A tabela clube, joga e jogador possuem respectivamente 7, 42 e 49 linhas. O PostgreSQL possui o parâmetro default\_statistics\_target que define o valor mínimo de 100 linhas a serem analisada.

3.4 Item (C)

Ao executar o script de povoamento de tabelas observamos que o PostgreSQL povoa a tabela pg\_stats para o schema "exercício". O povoamento pode ser visto na Figura 1

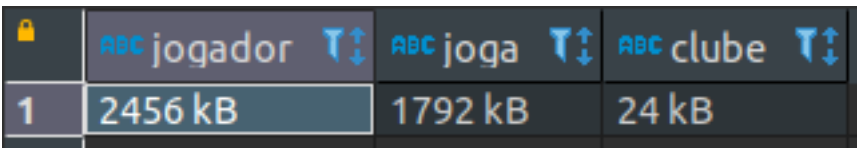
Figura 1: pg\_stats após o povoamento



schemaname	tablename	attname	inherited	null_frac	avg_width	n_distinct	most_common_vals	most_common_freqs
exercício	jogador	idjogador		0	4	-1	[NULL]	NULL
exercício	joga	idclube		0	4	7	{3,6,7,2,5,4,1}	{0.15050268,0.14705147,0.14
exercício	joga	datainiciojoga		0	4	-0.259590864	{2021-10-10,2022-03-21,201;	{0.00060021,0.00060021,0.00
exercício	joga	datafimjoga		0.000150053	4	-0.207972795	{2024-06-06,2024-05-15,202;	{0.0019506827,0.0016005602
exercício	joga	idjogador		0	4	-1	[NULL]	NULL
exercício	jogador	cpfjogador		0	15	-1	[NULL]	NULL
exercício	jogador	nomejogador		0	24	-1	[NULL]	NULL
exercício	jogador	apelidojogador		0	7	-0.248050004	{Silva,Santos,Oliveira,Souza	{0.08245,0.05155,0.03565,0.0
exercício	joga	salario		0	7	-0.61326462	{371000.00,380608.00,0.00,;	{0.026109139,0.011253939,0.

As novas linhas resultaram em relações com os seguintes tamanhos em kbytes:

Figura 2: pg\_total\_relation\_size após o povoamento



	jogador	joga	clube
1	2456 kB	1792 kB	24 kB

4 Exercício 3

Uma consulta pode ser realizada de diferentes formas, de acordo com o plano de execução escolhido pelo SGBD. Esse plano de execução pode ser obtido por meio de um comando específico. Por exemplo, no SGBD Oracle, tem-se o comando EXPLAIN PLAN. De acordo com o SGBD escolhido para resolver a lista de exercícios, defina e explique qual o comando para a visualização do plano de execução da consulta. Mais de um comando pode ser especificado e explicado, quando for o caso.



O comando em PostgreSQL para exibição do plano de execução é o EXPLAIN. Esse comando possui algumas variações que permitem aumentar o volume de informações a respeito do processamento da consulta.

```
1 Command: EXPLAIN
2 Description: show the execution plan of a statement
3 Syntax:
4 EXPLAIN [ ( option [, ...] ) ] statement
5 EXPLAIN [ ANALYZE ] [ VERBOSE ] statement
6 where option can be one of:
7 ANALYZE [ boolean ]
8 VERBOSE [ boolean ]
9 COSTS [ boolean ]
10 SETTINGS [ boolean ]
11 BUFFERS [ boolean ]
12 WAL [ boolean ]
13 TIMING [ boolean ]
14 SUMMARY [ boolean ]
15 FORMAT { TEXT | XML | JSON | YAML }
16
```

Os principais parâmetros definidos como opcionais habilitam as seguintes opções:

- ANALYZE: Exibe os tempos de execução reais e outras estatísticas.
- COSTS: Exibe o tempo de inicialização estimada e custo total de cada nó do plano
- BUFFERS: Exibe Informações sobre uso de buffer
- TIMING: Exibe o tempo real de inicialização e o tempo gasto em cada etapa.

## 5 Exercício 4

Considere a seguinte consulta. Liste, para cada clube e cada jogador, o nome do clube, o nome do jogador, a data de início, a data de fim e o salário.

### 5.1 Item (a)

Especifique o comando SQL que resolve essa consulta.

Join explícito

```

1      select clube.nomeclube, jogador.nomejogador, joga.
      datainiciojoga, joga.datafimjoga, joga.salario
2  from "exercicio".jogador jogador
3  join "exercicio".joga joga on jogador.idjogador = joga.idjogador
4  join exercicio.clube clube on clube.idclube = joga.idclube;
5

```

### Join implícito

```

1      select clube.nomeclube, jogador.nomejogador, joga.
      datainiciojoga, joga.datafimjoga, joga.salario
2  from "exercicio".jogador jogador, "exercicio".joga, "exercicio".clube
3  where jogador.idjogador = joga.idjogador and clube.idclube = joga.
      idclube;
4

```

### Produto cartesiano

```

1      select clube.nomeclube, jogador.nomejogador, joga.
      datainiciojoga, joga.datafimjoga, joga.salario
2  from "exercicio".jogador jogador
3  cross join "exercicio".joga joga
4  cross join "exercicio".clube clube
5  where jogador.idjogador = joga.idjogador and clube.idclube = joga.
      idclube;
6

```

## 5.2 Item (b)

Utilize o comando especificado no Exercício 3 para analisar o plano de execução da consulta. Faça um resumo explicando como a consulta foi realizada.

### Join explícito

```

1  explain (analyse true, buffers true, timing true, verbose true)
      select clube.nomeclube, jogador.nomejogador, joga.datainiciojoga,
      joga.datafimjoga, joga.salario
2  from "exercicio".jogador jogador
3  join "exercicio".joga joga on jogador.idjogador = joga.idjogador
4  join exercicio.clube clube on clube.idclube = joga.idclube;
5

```

```

1 Hash Join (cost=31.48..62.33 rows=1360 width=260) (actual
      time=0.243..0.354 rows=42 loops=1)
2   Output: clube.nomeclube, jogador.nomejogador, joga.datainiciojoga,
      joga.datafimjoga, joga.salario
3   Inner Unique: true

```

```

4   Hash Cond: (joga.idclube = clube.idclube)
5   Buffers: shared hit=3
6   -> Hash Join (cost=15.85..43.08 rows=1360 width=146) (actual
    time=0.136..0.187 rows=42 loops=1)
7     Output: jogador.nomejogador, joga.datainiciojoga,
    joga.datafimjoga, joga.salario, joga.idclube
8     Inner Unique: true
9     Hash Cond: (joga.idjogador = jogador.idjogador)
10    Buffers: shared hit=2
11    -> Seq Scan on exercicio.joga (cost=0.00..23.60 rows=1360
    width=32) (actual time=0.008..0.019 rows=42 loops=1)
12      Output: joga.idjogador, joga.idclube, joga.datainiciojoga,
    joga.datafimjoga, joga.salario
13      Buffers: shared hit=1
14    -> Hash (cost=12.60..12.60 rows=260 width=122) (actual
    time=0.094..0.095 rows=49 loops=1)
15      Output: jogador.nomejogador, jogador.idjogador
16      Buckets: 1024 Batches: 1 Memory Usage: 11kB
17      Buffers: shared hit=1
18    -> Seq Scan on exercicio.jogador (cost=0.00..12.60
    rows=260 width=122) (actual time=0.009..0.026 rows=49 loops=1)
19      Output: jogador.nomejogador, jogador.idjogador
20      Buffers: shared hit=1
21    -> Hash (cost=12.50..12.50 rows=250 width=122) (actual
    time=0.060..0.092 rows=7 loops=1)
22      Output: clube.nomeclube, clube.idclube
23      Buckets: 1024 Batches: 1 Memory Usage: 9kB
24      Buffers: shared hit=1
25    -> Seq Scan on exercicio.clube (cost=0.00..12.50 rows=250
    width=122) (actual time=0.020..0.025 rows=7 loops=1)
26      Output: clube.nomeclube, clube.idclube
27      Buffers: shared hit=1
28 Planning Time: 0.646 ms
29 Execution Time: 0.675 ms
30

```

## Join implícito

```

1   explain (analyse true, buffers true, timing true, verbose true)
    select clube.nomeclube, jogador.nomejogador, joga.datainiciojoga,
    joga.datafimjoga, joga.salario
2   from "exercicio".jogador jogador, "exercicio".joga, "exercicio".clube
3   where jogador.idjogador = joga.idjogador and clube.idclube = joga.

```

```
idclube;
```

```

1 Hash Join (cost=31.48..62.33 rows=1360 width=260) (actual
  time=0.258..0.358 rows=42 loops=1)
2   Output: clube.nomeclube, jogador.nomejogador, joga.datainiciojoga,
  joga.datafimjoga, joga.salario
3   Inner Unique: true
4   Hash Cond: (joga.idclube = clube.idclube)
5   Buffers: shared hit=3
6   -> Hash Join (cost=15.85..43.08 rows=1360 width=146) (actual
  time=0.148..0.198 rows=42 loops=1)
7     Output: jogador.nomejogador, joga.datainiciojoga,
  joga.datafimjoga, joga.salario, joga.idclube
8     Inner Unique: true
9     Hash Cond: (joga.idjogador = jogador.idjogador)
10    Buffers: shared hit=2
11    -> Seq Scan on exercicio.joga (cost=0.00..23.60 rows=1360
  width=32) (actual time=0.008..0.019 rows=42 loops=1)
12      Output: joga.idjogador, joga.idclube, joga.datainiciojoga,
  joga.datafimjoga, joga.salario
13      Buffers: shared hit=1
14      -> Hash (cost=12.60..12.60 rows=260 width=122) (actual
  time=0.099..0.100 rows=49 loops=1)
15        Output: jogador.nomejogador, jogador.idjogador
16        Buckets: 1024 Batches: 1 Memory Usage: 11kB
17        Buffers: shared hit=1
18        -> Seq Scan on exercicio.jogador (cost=0.00..12.60
  rows=260 width=122) (actual time=0.016..0.033 rows=49 loops=1)
19          Output: jogador.nomejogador, jogador.idjogador
20          Buffers: shared hit=1
21      -> Hash (cost=12.50..12.50 rows=250 width=122) (actual
  time=0.063..0.064 rows=7 loops=1)
22        Output: clube.nomeclube, clube.idclube
23        Buckets: 1024 Batches: 1 Memory Usage: 9kB
24        Buffers: shared hit=1
25        -> Seq Scan on exercicio.clube (cost=0.00..12.50 rows=250
  width=122) (actual time=0.017..0.022 rows=7 loops=1)
26          Output: clube.nomeclube, clube.idclube
27          Buffers: shared hit=1
28 Planning Time: 0.526 ms
29 Execution Time: 0.629 ms
30
```

## Produto cartesiano

```

1  explain (analyse true, buffers true, timing true, verbose true)
2  select clube.nomeclube, jogador.nomejogador, joga.datainiciojoga,
3  joga.datafimjoga, joga.salario
4  from "exercicio".jogador jogador
5  cross join "exercicio".joga joga
6  cross join "exercicio".clube clube
7  where jogador.idjogador = joga.idjogador and clube.idclube = joga.
8  idclube;

```

```

1 Hash Join (cost=31.48..62.33 rows=1360 width=260) (actual
2   time=0.232..0.319 rows=42 loops=1)
3   Output: clube.nomeclube, jogador.nomejogador, joga.datainiciojoga,
4   joga.datafimjoga, joga.salario
5   Inner Unique: true
6   Hash Cond: (joga.idclube = clube.idclube)
7   Buffers: shared hit=3
8   -> Hash Join (cost=15.85..43.08 rows=1360 width=146) (actual
9     time=0.130..0.180 rows=42 loops=1)
10    Output: jogador.nomejogador, joga.datainiciojoga,
11    joga.datafimjoga, joga.salario, joga.idclube
12    Inner Unique: true
13    Hash Cond: (joga.idjogador = jogador.idjogador)
14    Buffers: shared hit=2
15    -> Seq Scan on exercicio.joga (cost=0.00..23.60 rows=1360
16      width=32) (actual time=0.008..0.019 rows=42 loops=1)
17      Output: joga.idjogador, joga.idclube, joga.datainiciojoga,
18      joga.datafimjoga, joga.salario
19      Buffers: shared hit=1
20    -> Hash (cost=12.60..12.60 rows=260 width=122) (actual
21      time=0.087..0.088 rows=49 loops=1)
22      Output: jogador.nomejogador, jogador.idjogador
23      Buckets: 1024 Batches: 1 Memory Usage: 11kB
24      Buffers: shared hit=1
25      -> Seq Scan on exercicio.jogador (cost=0.00..12.60
26        rows=260 width=122) (actual time=0.009..0.026 rows=49 loops=1)
27        Output: jogador.nomejogador, jogador.idjogador
28        Buffers: shared hit=1
29    -> Hash (cost=12.50..12.50 rows=250 width=122) (actual
30      time=0.063..0.071 rows=7 loops=1)
31      Output: clube.nomeclube, clube.idclube
32      Buckets: 1024 Batches: 1 Memory Usage: 9kB
33      Buffers: shared hit=1

```

```

25      -> Seq Scan on exercicio.clube (cost=0.00..12.50 rows=250
      width=122) (actual time=0.018..0.023 rows=7 loops=1)
26          Output: clube.nomeclube, clube.idclube
27          Buffers: shared hit=1
28 Planning Time: 0.569 ms
29 Execution Time: 0.651 ms
30

```

Observa-se que diante das três possibilidades de execução da consulta o plano não é alterado, o otimizador realiza os passos necessários para converter o join implícito e o produto cartesiano para o mesmo plano de execução do join explícito.

O plano de execução em todos os casos se resume nas seguintes etapas, analisadas a partir da ordem de execução:

1. Leitura sequencial da tabela clube
2. Hash da tabela clube: Carrega a tabela na memória estruturando-a como uma tabela hash
3. Leitura sequencial da tabela jogador
4. Hash da tabela jogador: Carrega a tabela na memória estruturando-a como uma tabela hash
5. Leitura sequencial da tabela joga
6. Hash join envolvendo a condição joga e jogador por meio dos id's
7. Hash join envolvendo a condição joga e clube

O cenário anterior considera apenas os dados fornecidos previamente pela especificação do trabalho. Após a execução do script de povoamento as três consultas anteriores foram reexecutadas, mantendo o plano inalterado. O plano do join explícito esta representado no código abaixo.

```

1 Hash Join (cost=669.62..1112.42 rows=19993 width=157) (actual
      time=7.960..15.173 rows=19993 loops=1)
2   Output: clube.nomeclube, jogador.nomejogador, joga.datainiciojoga,
      joga.datafimjoga, joga.salario
3   Inner Unique: true
4   Hash Cond: (joga.idclube = clube.idclube)
5   Buffers: shared hit=342
6   -> Hash Join (cost=654.00..1043.42 rows=19993 width=43) (actual
      time=7.856..12.566 rows=19993 loops=1)

```

```

7      Output: jogador.nomejogador, joga.datainiciojoga,
      joga.datafimjoga, joga.salario, joga.idclube
8      Inner Unique: true
9      Hash Cond: (joga.idjogador = jogador.idjogador)
10     Buffers: shared hit=341
11     -> Seq Scan on exercicio.joga (cost=0.00..336.93 rows=19993
      width=23) (actual time=0.008..1.068 rows=19993 loops=1)
12     Output: joga.idjogador, joga.idclube, joga.datainiciojoga,
      joga.datafimjoga, joga.salario
13     Buffers: shared hit=137
14     -> Hash (cost=404.00..404.00 rows=20000 width=28) (actual
      time=7.245..7.245 rows=20000 loops=1)
15     Output: jogador.nomejogador, jogador.idjogador
16     Buckets: 32768 Batches: 1 Memory Usage: 1476kB
17     Buffers: shared hit=204
18     -> Seq Scan on exercicio.jogador (cost=0.00..404.00
      rows=20000 width=28) (actual time=0.010..2.454 rows=20000 loops=1)
19     Output: jogador.nomejogador, jogador.idjogador
20     Buffers: shared hit=204
21 -> Hash (cost=12.50..12.50 rows=250 width=122) (actual
      time=0.068..0.068 rows=7 loops=1)
22     Output: clube.nomeclube, clube.idclube
23     Buckets: 1024 Batches: 1 Memory Usage: 9kB
24     Buffers: shared hit=1
25     -> Seq Scan on exercicio.clube (cost=0.00..12.50 rows=250
      width=122) (actual time=0.023..0.028 rows=7 loops=1)
26     Output: clube.nomeclube, clube.idclube
27     Buffers: shared hit=1
28 Planning:
29     Buffers: shared hit=14
30 Planning Time: 1.279 ms
31 Execution Time: 16.104 ms
32

```

### 5.3 Item (c)

Especifique qual o tempo gasto para processar a consulta.

O tempo gasto para cada consulta, para os dados fornecidos na especificação, foram respectivamente: **0.675 ms, 0.629 ms, 0.651 ms**

Considerando os novos dados os tempos foram: **15.656 ms, 15.959 ms,**

15.584 ms

## 6 Exercício 5

Considere a seguinte consulta. Liste, para o clube de apelido Flamengo, o nome do clube, o nome do jogador, a data de início, a data de fim e o salário.

### 6.1 Item (a)

Especifique o comando SQL que resolve essa consulta.

```
1  select clube.nomeclube, jogador.nomejogador, joga.datainiciojoga ,
   joga.datafimjoga, joga.salario
2  from "exercicio".jogador jogador
3  join "exercicio".joga joga on jogador.idjogador = joga.idjogador
4  join exercicio.clube clube on clube.idclube = joga.idclube
5  where clube.apelidoclube = 'Flamengo';
6
```

### 6.2 Item (b)

Utilize o comando especificado no Exercício 3 para analisar o plano de execução da consulta. Faça um resumo explicando como a consulta foi realizada.

```
1 Nested Loop (cost=4.35..28.55 rows=5 width=260) (actual
   time=0.037..0.044 rows=6 loops=1)
2   Output: clube.nomeclube, jogador.nomejogador, joga.datainiciojoga,
   joga.datafimjoga, joga.salario
3   Inner Unique: true
4   Buffers: shared hit=15
5   -> Nested Loop (cost=4.21..27.55 rows=5 width=146) (actual
   time=0.028..0.031 rows=6 loops=1)
6     Output: joga.datainiciojoga, joga.datafimjoga, joga.salario,
   joga.idjogador, clube.nomeclube
7     Buffers: shared hit=3
8     -> Seq Scan on exercicio.clube (cost=0.00..13.12 rows=1
   width=122) (actual time=0.009..0.009 rows=1 loops=1)
9       Output: clube.idclube, clube.cnpjclube, clube.nomeclube,
   clube.apelidoclube
10      Filter: ((clube.apelidoclube)::text = 'Flamengo'::text)
11      Rows Removed by Filter: 6
```



```

12         Buffers: shared hit=1
13     -> Bitmap Heap Scan on exercicio.joga (cost=4.21..14.35 rows=7
width=32) (actual time=0.017..0.018 rows=6 loops=1)
14         Output: joga.idjogador, joga.idclube, joga.datainiciojoga,
joga.datafimjoga, joga.salario
15         Recheck Cond: (joga.idclube = clube.idclube)
16         Heap Blocks: exact=1
17         Buffers: shared hit=2
18     -> Bitmap Index Scan on pk_joga (cost=0.00..4.21 rows=7
width=0) (actual time=0.013..0.013 rows=6 loops=1)
19         Index Cond: (joga.idclube = clube.idclube)
20         Buffers: shared hit=1
21 -> Index Scan using pk_membro on exercicio.jogador (cost=0.15..0.20
rows=1 width=122) (actual time=0.002..0.002 rows=1 loops=6)
22     Output: jogador.idjogador, jogador.cpfjogador,
jogador.nomejogador, jogador.apelidojogador
23     Index Cond: (jogador.idjogador = joga.idjogador)
24     Buffers: shared hit=12
25 Planning:
26     Buffers: shared hit=244
27 Planning Time: 1.557 ms
28 Execution Time: 0.124 ms
29

```

1. Leitura do index da tabela jogador
2. Leitura do index da tabela joga com o objetivo de recuperar os blocos necessários para a etapa Bitmap Heap Scan. Note que não há indicação de armazenamento dos dados em memória (width=0), por meio do buffer é possível verificar que o bloco já está presente no buffer de memória do S.O (shared hit = 1)
3. Bitmap Heap Scan: leitura dos blocos mapeados pela etapa anterior, com o indicativo de recheck da condição(recuperar o dado em disco) joga.idclube = clube.idclube em uma etapa posterior
4. Scan sequencial na tabela clube com o objetivo de recuperar todas as colunas para o clube flamengo (rows=1 width=122) totalizando 122 bytes
5. Loop aninhado envolvendo as tabelas joga e clube com o objetivo de unir os atributos das relações
6. Loop aninhado envolvendo os atributos da tabela jogador e o loop aninhado

anterior

Para os novos valores inseridos, não há alteração no plano de execução. Apenas os valores relacionados aos usos de buffer, número de linhas estimadas, custo estimado, tempo de planejamento e tempo de execução variaram devido ao novo volume.

### 6.3 Item (b)

Utilize o comando especificado no Exercício 3 para analisar o plano de execução da consulta. Faça um resumo explicando como a consulta foi realizada.

```

1 Nested Loop (cost=70.71..313.38 rows=80 width=157) (actual
   time=0.509..4.241 rows=2812 loops=1)
2   Output: clube.nomeclube, jogador.nomejogador, joga.datainiciojoga,
   joga.datafimjoga, joga.salario
3   Inner Unique: true
4   Buffers: shared hit=8586
5   -> Nested Loop (cost=70.42..284.81 rows=80 width=137) (actual
   time=0.497..1.164 rows=2812 loops=1)
6     Output: joga.datainiciojoga, joga.datafimjoga, joga.salario,
   joga.idjogador, clube.nomeclube
7     Buffers: shared hit=150
8     -> Seq Scan on exercicio.clube (cost=0.00..13.12 rows=1
   width=122) (actual time=0.044..0.045 rows=1 loops=1)
9       Output: clube.idclube, clube.cnpjclube, clube.nomeclube,
   clube.apelidoclube, clube.numero_jogadores
10      Filter: ((clube.apelidoclube)::text = 'Flamengo'::text)
11      Rows Removed by Filter: 6
12      Buffers: shared hit=1
13     -> Bitmap Heap Scan on exercicio.joga (cost=70.42..243.12
   rows=2856 width=23) (actual time=0.445..0.861 rows=2812 loops=1)
14       Output: joga.idjogador, joga.idclube, joga.datainiciojoga,
   joga.datafimjoga, joga.salario
15       Recheck Cond: (joga.idclube = clube.idclube)
16       Heap Blocks: exact=137
17       Buffers: shared hit=149
18     -> Bitmap Index Scan on pk_joga (cost=0.00..69.71
   rows=2856 width=0) (actual time=0.428..0.429 rows=2812 loops=1)
19       Index Cond: (joga.idclube = clube.idclube)
20       Buffers: shared hit=12
21   -> Index Scan using pk_membro on exercicio.jogador (cost=0.29..0.36
   rows=1 width=28) (actual time=0.001..0.001 rows=1 loops=2812)

```

```

22      Output: jogador.idjogador, jogador.cpfjogador,
23      jogador.nomejogador, jogador.apelidojogador
24      Index Cond: (jogador.idjogador = joga.idjogador)
25      Buffers: shared hit=8436
26 Planning:
27 Buffers: shared hit=82
28 Planning Time: 3.457 ms
29 Execution Time: 4.503 ms

```

## 6.4 Item (c)

Especifique qual o tempo gasto para processar a consulta.

A consulta foi executada em **0.124 ms** com um tempo de planejamento de 1.557 ms. Com o novo volume de dados inseridos no Exercício 2(c) os valores são: 3.457 ms para o planejamento e 4.503 ms para a execução

## 7 Exercício 6

Considere a seguinte consulta. Liste todos os dados referentes ao jogador de apelido Danilo.

### 7.1 Item (a)

Especifique o comando SQL que resolve essa consulta.

```

1      select * from exercicio.jogador where apelidojogador = 'Danilo';
2

```

### 7.2 Item (b)

Utilize o comando especificado no Exercício 3 para analisar o plano de execução da consulta. Faça um resumo explicando como a consulta foi realizada.

```

1 Seq Scan on exercicio.jogador (cost=0.00..13.25 rows=1 width=286)
   (actual time=0.032..0.039 rows=1 loops=1)
2   Output: idjogador, cpfjogador, nomejogador, apelidojogador
3   Filter: ((jogador.apelidojogador)::text = 'Danilo'::text)
4   Rows Removed by Filter: 48

```

```

5   Buffers: shared hit=1
6 Planning Time: 0.097 ms
7 Execution Time: 0.064 ms
8

```

### 1. Recupera os dados sequencialmente com um filtro em apelidojogador

O plano de execução para um volume de dados maior reproduz os cenários anteriores onde o plano não é alterado. Isso deve-se ao baixo volume de dados inseridos, apesar de ser significativamente maior do que os fornecidos no trabalho ainda não são suficientes para a construção de um plano diferente.

```

1 Seq Scan on exercicio.jogador (cost=0.00..454.00 rows=2 width=50)
   (actual time=0.028..4.915 rows=1 loops=1)
2   Output: idjogador, cpfjogador, nomejogador, apelidojogador
3   Filter: ((jogador.apelidojogador)::text = 'Danilo'::text)
4   Rows Removed by Filter: 19999
5   Buffers: shared hit=204
6 Planning:
7   Buffers: shared hit=6 dirtied=1
8 Planning Time: 0.425 ms
9 Execution Time: 4.935 ms
10

```

## 7.3 Adicional

O seguinte comando em SQL desabilita a busca sequencial. Como essa é a única estratégia possível a ser utilizada pelo query planner espera-se que o plano não seja alterado.

```

1 SET enable_seqscan = OFF;
2

```

```

1 Seq Scan on exercicio.jogador (cost=10000000000.00..10000000454.00
   rows=2 width=50) (actual time=76.489..77.487 rows=1 loops=1)
2   Output: idjogador, cpfjogador, nomejogador, apelidojogador
3   Filter: ((jogador.apelidojogador)::text = 'Danilo'::text)
4   Rows Removed by Filter: 19999
5   Buffers: shared hit=204
6 Planning Time: 0.141 ms
7 JIT:

```

```

8 Functions: 2
9 Options: Inlining true, Optimization true, Expressions true, Deforming
  true
10 Timing: Generation 0.200 ms, Inlining 47.220 ms, Optimization 17.255
  ms, Emission 11.997 ms, Total 76.672 ms
11 Execution Time: 97.076 ms
12

```

O custo esperado ao desabilitar a única opção possível explode para o valor limite. Esse cenário obriga a utilização do Just-In-Time Compilation, definido para ser executado nos casos onde o custo estimado de execução é elevado (queries CPU Bound). Observa-se que o tempo total gasto é majoritariamente gasto no processo de compilação em tempo real do código SQL: Generation 0.200 ms, Inlining 47.220 ms, Optimization 17.255 ms, Emission 11.997 ms, Total 76.672 ms

Conclui-se que desabilitar atributos de otimização não é uma atividade impossível, ou seja, o PostgreSQL habilitará novamente o atributo caso o JIT verifique que essa é a única opção possível.

## 7.4 Item (c)

Crie um índice que indexe os apelidos dos jogadores.

```

1 CREATE INDEX jogador_apelidojogador_idx ON exercicio.jogador USING
  btree (apelidojogador);
2

```

## 7.5 Item (d)

Utilize o comando especificado no Exercício 3 para analisar o plano de execução da consulta. Faça um resumo explicando como a consulta foi realizada e compare com o plano de consulta do Exercício 6b.

```

1 Seq Scan on exercicio.jogador (cost=0.00..1.61 rows=1 width=286) (actual
  time=0.029..0.036 rows=1 loops=1)
2   Output: idjogador, cpfjogador, nomejogador, apelidojogador
3   Filter: ((jogador.apelidojogador)::text = 'Danilo'::text)
4   Rows Removed by Filter: 48
5   Buffers: shared hit=1
6 Planning Time: 0.124 ms
7 Execution Time: 0.057 ms
8

```

## 1. Recupera os dados sequencialmente com um filtro em apelidojogador

Não há alteração no plano de execução. O otimizador considera o custo de um scan sequencial mais vantajoso do que a utilização do índice. Ao desabilitar o scan sequencial podemos ver o custo de utilização do índice.

```

1 SET enable_seqscan = OFF;
2 explain (analyse true, buffers true, timing true, verbose true)
3 select * from exercicio.jogador where apelidojogador = 'Danilo';

```

```

1 Index Scan using jogador_apelidojogador_idx on exercicio.jogador
  (cost=0.14..8.16 rows=1 width=286) (actual time=0.148..0.152 rows=1
  loops=1)
2  Output: idjogador, cpfjogador, nomejogador, apelidojogador
3  Index Cond: ((jogador.apelidojogador)::text = 'Danilo'::text)
4  Buffers: shared hit=1 read=1
5  Planning Time: 0.155 ms
6  Execution Time: 0.210 ms
7

```

É possível observar que o tempo de execução é consideravelmente superior se comparado com a busca sequencial, cerca de 3.68 vezes maior. O custo estimado também é superior, na ordem de 5,06 vezes.

Para os novos valores inseridos no exercício 2c o plano sofre uma alteração na forma de recuperação dos valores do índice.

```

1 Bitmap Heap Scan on exercicio.jogador (cost=4.30..11.73 rows=2 width=50)
  (actual time=0.053..0.056 rows=1 loops=1)
2  Output: idjogador, cpfjogador, nomejogador, apelidojogador
3  Recheck Cond: ((jogador.apelidojogador)::text = 'Danilo'::text)
4  Heap Blocks: exact=1
5  Buffers: shared hit=1 read=2
6  -> Bitmap Index Scan on jogador_apelidojogador_idx (cost=0.00..4.30
  rows=2 width=0) (actual time=0.046..0.047 rows=1 loops=1)
7      Index Cond: ((jogador.apelidojogador)::text = 'Danilo'::text)
8      Buffers: shared read=2
9  Planning:
10     Buffers: shared hit=16 read=1
11  Planning Time: 0.817 ms
12  Execution Time: 0.161 ms
13

```

Nesse novo cenário o índice passa a ser acessado por meio do Bitmap Index Scan, mapeando os nós da Btree que contém os valores a serem analisados com a condição 'Danilo' para serem recuperados na etapa seguinte de Bitmap Heap Scan. O tamanho em bytes recuperados na primeira etapa é 0 (como pode ser visto em width=0). Os valores da tupla são recuperados apenas na etapa seguinte, como indicado na cláusula Recheck Cond.

Não está claro o exato motivo da aplicação da cláusula Recheck, pois a cláusula foi projetada para ser utilizada em cenários onde a memória utilizada para processar a consulta supera os 4Mb definidos como padrão no work\_mem. Para essa consulta o otimizador considera recuperar 2 linhas com um total de 50 bytes. Uma possível explicação, não encontrada na literatura, pode ser que a cláusula seja aplicada não relacionada ao volume de dados esperados a serem recuperados e sim em relação ao total de memória utilizada pela consulta.

Quando desabilitado o Bitmap Index scan o query planner indica a utilização do Index Scan. O otimizador está correto nesse caso, pois o custo do Index Scan é superior ao do Bitmap Index Scan, como visto a seguir:

```
1 Index Scan using jogador_apelidojogador_idx on exercicio.jogador
   (cost=0.29..12.32 rows=2 width=50) (actual time=0.073..0.075 rows=1
   loops=1)
2  Output: idjogador, cpfjogador, nomejogador, apelidojogador
3  Index Cond: ((jogador.apelidojogador)::text = 'Danilo'::text)
4  Buffers: shared hit=3
5  Planning Time: 0.168 ms
6  Execution Time: 0.151 ms
7
```

Uma breve olhada nas estatísticas da tabela pg\_stats indica o motivo pelo qual o Bitmap Index é preferível ao Index Scan. As páginas de disco a serem recuperadas não estão ordenadas de acordo com o índice em apelidojogador e sim em relação ao idjogador, dessa forma compensa mapear as páginas necessárias para recuperá-las posteriormente. O comando abaixo revela a correlação entre os valores das colunas da tabela jogador.

```
1 SELECT tablename, attname, correlation
2 FROM pg_stats
3 WHERE tablename IN ('jogador')
4 ORDER BY 1, 2;
5
```

Tabela 1

tablename	attname	correlation
jogador	apelidojogador	0.014959549
jogador	cpfjogador	-0.010097355
jogador	idjogador	0.99967754
jogador	nomejogador	0.985966

## 8 Exercício 7

Considere a seguinte consulta. Liste o número de jogadores de cada clube.

### 8.1 Item (a)

Especifique o comando SQL que resolve essa consulta

```

1  select clube.nomeclube , count(jogador.idjogador) as "numero de
   jogadores"
2  from exercicio.jogador jogador
3  join exercicio.joga joga on jogador.idjogador = joga.idjogador
4  join exercicio.clube clube on joga.idclube = clube.idclube
5  group by clube.nomeclube ;
6

```

### 8.2 Item (b)

Utilize o comando especificado no Exercício 3 para analisar o plano de execução da consulta. Faça um resumo explicando como a consulta foi realizada

```

1 HashAggregate (cost=151.13..153.13 rows=200 width=126) (actual
   time=0.562..0.573 rows=7 loops=1)
2   Output: clube.nomeclube, count(jogador.idjogador)
3   Group Key: clube.nomeclube
4   Batches: 1  Memory Usage: 40kB
5   Buffers: shared hit=6
6   -> Hash Join (cost=68.66..144.33 rows=1360 width=122) (actual
   time=0.430..0.515 rows=42 loops=1)
7     Output: clube.nomeclube, jogador.idjogador
8     Inner Unique: true
9     Hash Cond: (joga.idclube = clube.idclube)
10    Buffers: shared hit=6
11    -> Hash Join (cost=13.64..85.69 rows=1360 width=8) (actual
   time=0.288..0.347 rows=42 loops=1)

```



```

12      Output: jogador.idjogador, joga.idclube
13      Inner Unique: true
14      Hash Cond: (joga.idjogador = jogador.idjogador)
15      Buffers: shared hit=4
16      -> Index Only Scan using pk_joga on exercicio.joga
      (cost=0.15..68.55 rows=1360 width=8) (actual time=0.084..0.114
      rows=42 loops=1)
17      Output: joga.idclube, joga.idjogador,
      joga.datainiciojoga
18      Heap Fetches: 42
19      Buffers: shared hit=2
20      -> Hash (cost=12.88..12.88 rows=49 width=4) (actual
      time=0.161..0.162 rows=49 loops=1)
21      Output: jogador.idjogador
22      Buckets: 1024 Batches: 1 Memory Usage: 10kB
23      Buffers: shared hit=2
24      -> Index Only Scan using pk_membro on
      exercicio.jogador (cost=0.14..12.88 rows=49 width=4) (actual
      time=0.046..0.080 rows=49 loops=1)
25      Output: jogador.idjogador
26      Heap Fetches: 49
27      Buffers: shared hit=2
28      -> Hash (cost=51.90..51.90 rows=250 width=122) (actual
      time=0.074..0.075 rows=7 loops=1)
29      Output: clube.nomeclube, clube.idclube
30      Buckets: 1024 Batches: 1 Memory Usage: 9kB
31      Buffers: shared hit=2
32      -> Index Scan using pk_clube on exercicio.clube
      (cost=0.14..51.90 rows=250 width=122) (actual time=0.037..0.044
      rows=7 loops=1)
33      Output: clube.nomeclube, clube.idclube
34      Buffers: shared hit=2
35 Planning Time: 0.699 ms
36 Execution Time: 1.002 ms
37

```

1. Index Scan no índice de clube
2. Construção de uma tabela Hash com nomeclube e idclube
3. Index Only Scan no índice idjogador da tabela jogador
4. Construção de uma tabela Hash com idjogador

5. Index Only Scan no índice (idclube, idjogador) da tabela joga
6. Hash join envolvendo jogador.idjogador e joga.idclube
7. Hash join envolvendo joga.idclube e clube.idclube
8. HashAggregate com a cláusula de agrupamento nomeclube

### 8.3 Item (c)

Crie um novo atributo que armazene o número de jogadores de cada time. Em qual tabela esse novo atributo deve ser armazenado? Justifique. Note que a criação do atributo refere-se à alteração da tabela correspondente para incorporar esse atributo

O novo atributo deve ser armazenado na tabela clube. Cada time deve conter a contagem de jogadores que joga no time, portanto a tabela a ser alterada é a Clube.

```

1 alter table exercicio.clube add numero_jogadores integer;
2
3 update exercicio.clube set numero_jogadores=subquery.numero_jogadores
4 from (select clube.idclube , count(jogador.idjogador) as
      numero_jogadores
5  from exercicio.jogador jogador
6  join exercicio.joga joga on jogador.idjogador = joga.idjogador
7  join exercicio.clube clube on joga.idclube = clube.idclube
8  group by clube.idclube) as subquery
9 where clube.idclube = subquery.idclube;
10
```

### 8.4 Item (d)

Especifique o comando SQL que resolve a consulta do Exercício 7, porém usando o novo atributo criado no Exercício 7c.

```

1 select c.nomeclube, c.numero_jogadores from exercicio.clube c;
2
```

### 8.5 Item (e)

Utilize o comando especificado no Exercício 3 para analisar o plano de execução da consulta. Faça um resumo explicando como a consulta foi realizada e compare com o plano de consulta do Exercício 7b.

```

1 Seq Scan on exercicio.clube c  (cost=0.00..12.50 rows=250 width=122)
   (actual time=0.022..0.027 rows=7 loops=1)
```

```
2 Output: nomeclube, numero_jogadores
3 Buffers: shared hit=1
4 Planning Time: 0.073 ms
5 Execution Time: 0.064 ms
6
```

## 9 Exercício 8

Considere a consulta do Exercício 4. Faça testes de escalabilidade de volume de dados. Em um teste de escalabilidade, devem ser definidos volumes crescentes de dados considerando o mesmo intervalo de valores. Por exemplo, podem ser considerados volumes de dados de 100 MB, 200 MB, 300 MB e 400 MB. Outro exemplo, podem ser considerados volumes de dados de 5 MB, 10 MB, 15 MB e 20 MB.

### 9.1 Item (a)

Para cada volume de dados considerado, colete o tempo gasto para processar a consulta do Exercício 4.

Figura 3: Testes com o cache do SO limpo

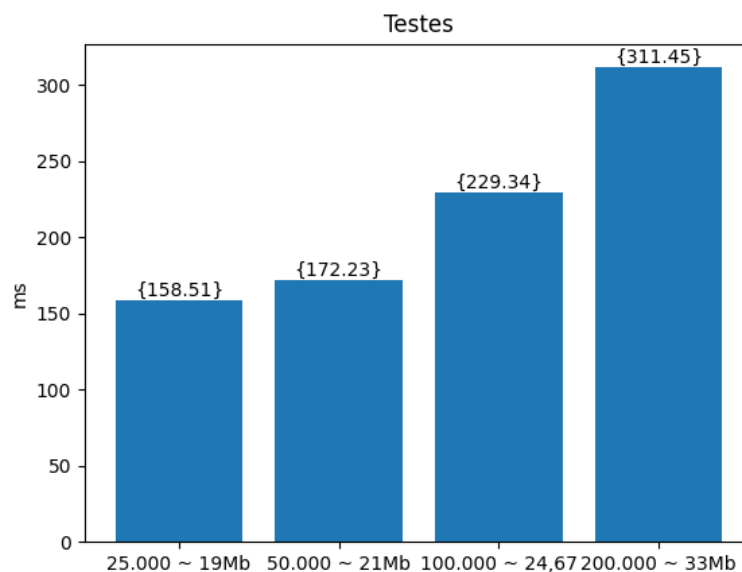


Figura 4: Testes com o cache do SO limpo e com estatísticas limpas

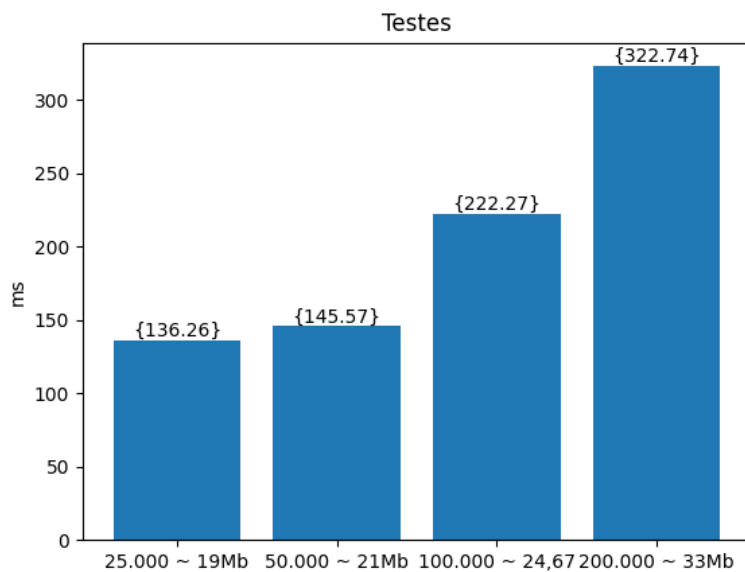


Figura 5: Testes com o cache pré-aquecido

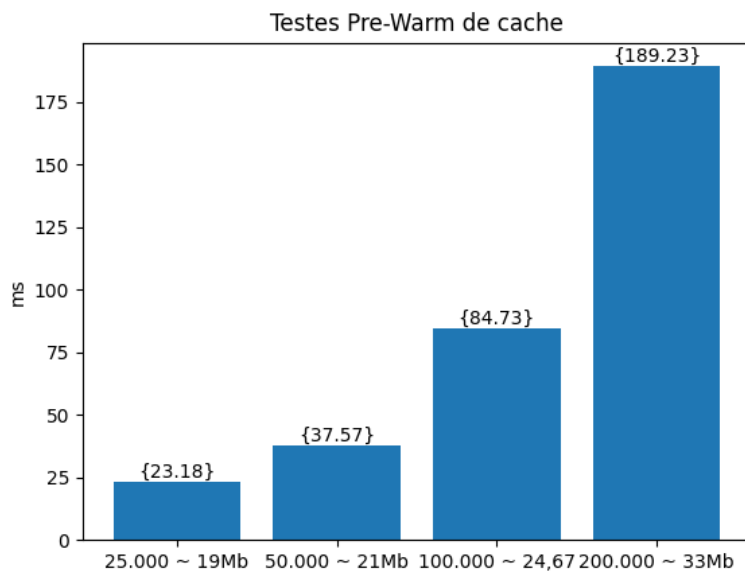
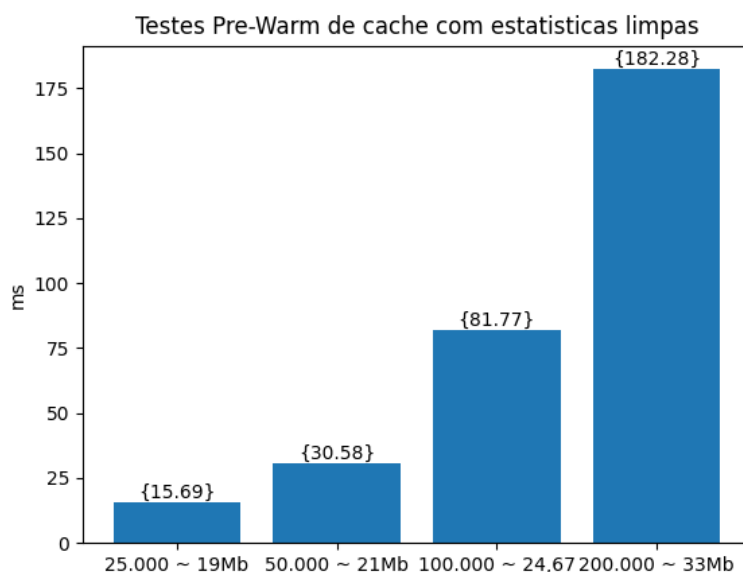


Figura 6: Testes com o cache pré-aquecido com as estatísticas limpas



## 9.2 Item (b)

**Realize uma análise dos dados coletados. Ou seja, discuta o que aconteceu com o tempo gasto para processar a consulta frente ao aumento do volume de dados.**

Para cada volume de dados, em todos os testes, foi tirada a média de 5 execuções da consulta. Como esperado, em todos os testes quando o volume de dados aumenta o tempo de consulta também aumenta. Observa-se a influência positiva do pré-aquecimento do buffer do SGBD, melhorando muito o tempo de resposta de todos os cenários.

Um caso particular encontrado nos testes foi o resultado apresentado quando as estatísticas do SGBD são limpas. Em todos os testes, com exceção do que contém o maior volume de dados, as consultas obtiveram um melhor tempo de resposta.

## 10 Apêndice

compose.yaml

```
1 services:
2 # -----Database
```

```
3 postgres:
4   image: postgres:14-alpine
5   container_name: postgres
6   volumes:
7     - postgres:/var/lib/postgresql/data
8     - ./sql:/sql
9   environment:
10    - POSTGRES_USER=postgres
11    - POSTGRES_PASSWORD=postgres
12    - POSTGRES_DB=db1
13   networks:
14     - bd-network
15   ports:
16     - "5432:5432"
17   deploy:
18     labels:
19       app.db1.description: "database service"
20     mode: replicated
21     replicas: 1
22     endpoint_mode: vip
23     resources:
24       limits:
25         # Revert cpu limit
26         cpus: '4'
27     restart_policy:
28       condition: on-failure
29       delay: 5s
30       max_attempts: 3
31       window: 120s
32 # ----- VOLUMES
33 volumes:
34   postgres:
35 # ----- NETWORKS
36 networks:
37   bd-network:
38     driver: bridge
39     labels:
40       app.db1.description: "db1 services network"
```