Universidade Federal do Ceará
Campus de Quixadá

Construção de Sistemas de Gerência de Banco de Dados

Trabalho 1

Aluna: Bárbara Stéphanie Neves e Joyce Nayne Araújo
Professora: Lívia Almada

Maio
2019

Universidade Federal do Ceará
Campus de Quixadá

Construção de Sistemas de Gerência de Banco de Dados

Trabalho 1

Relatório do Trabalho 1 sobre conceitos de SQL, Indexação e Processamento de Consultas, da disciplina de Construção de Sistemas de Gerência de Banco de Dados.

Aluna: Bárbara Stéphanie Neves e Joyce Nayne Araújo

Matrículas: 388713 e 383868

Professora: Lívia Almada

Curso: Ciência da Computação

Maio
2019

Conteúdo

1 Escolha, Construção e Povoamento das Bases de Dados

1.1 Escolha

A base de dados escolhida foi **Eleições do Brasil – 2018** presentes no site do TSE que pode ser acessado através do **link**.

Como o do repositório do TSE possui dados eleitorais desde 1994, escolhemos fazer a análise de dados somente das eleições ocorridas em 2018. Portanto, a nossa base de dados tem cerca de **4.0 GB**.

A máquina onde foram feitos os processamentos das consultas possui as seguintes especificações:

	Componente	**Informações**
1	Processador	Intel Core i5 - 8th Gen
2	Memória RAM	8 GB
3	HD	1 TB
4	SO	Ubuntu 18.04.2 LTS
5	SGBD	PostgreSQL 10.8

Tabela 1: Especificações do computador utilizado para executar os testes

1.2 Construção e Povoamento

Escolhemos quatro tabelas para compor o modelo relacional. São elas:

	Tabela	**Informações**
1	**`bens_declarados`**	Contém os bens declarados dos(as) candidatos(as).
2	**`candidatos`**	Contém os dados pessoais dos(as) candidatos(as).
3	**`filiacoes`**	Contém as coligações dos partidos.
4	**`votacoes`**	Estão presentes todos os dados gerados das eleições.

Tabela 2: Tabelas do Banco de Dados **`eleicoes_brasil_2018`**.

Todos os dados citados acima presentes nas quatro tabelas estavam dispostos em arquivos *.csv*. Portanto, realizamos um processo de ETL (Extração, Transformação e Carga) utilizando o componente *Pentaho Data Integration* (PDI/*Kettle*).

As imagens abaixo correspondem as transformações feitas dos arquivos *.csv* para as suas respectivas tabelas.

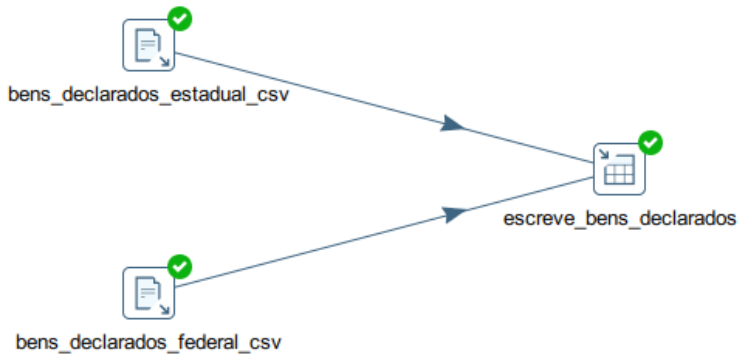

Figura 1: Transformação concluída da tabela **`bens_declarados`**

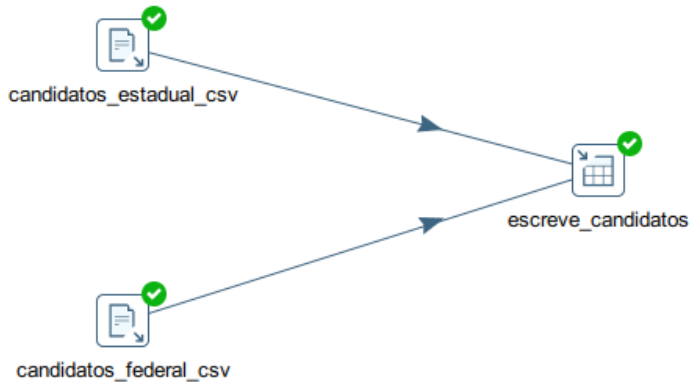

Figura 2: Transformação concluída da tabela **`candidatos`**

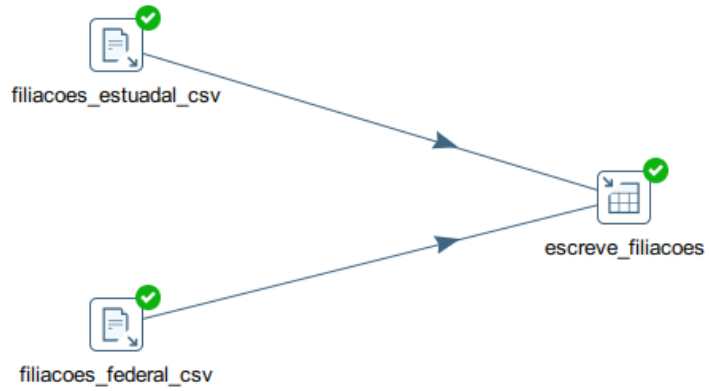

Figura 3: Transformação concluída da tabela **`filiados`**

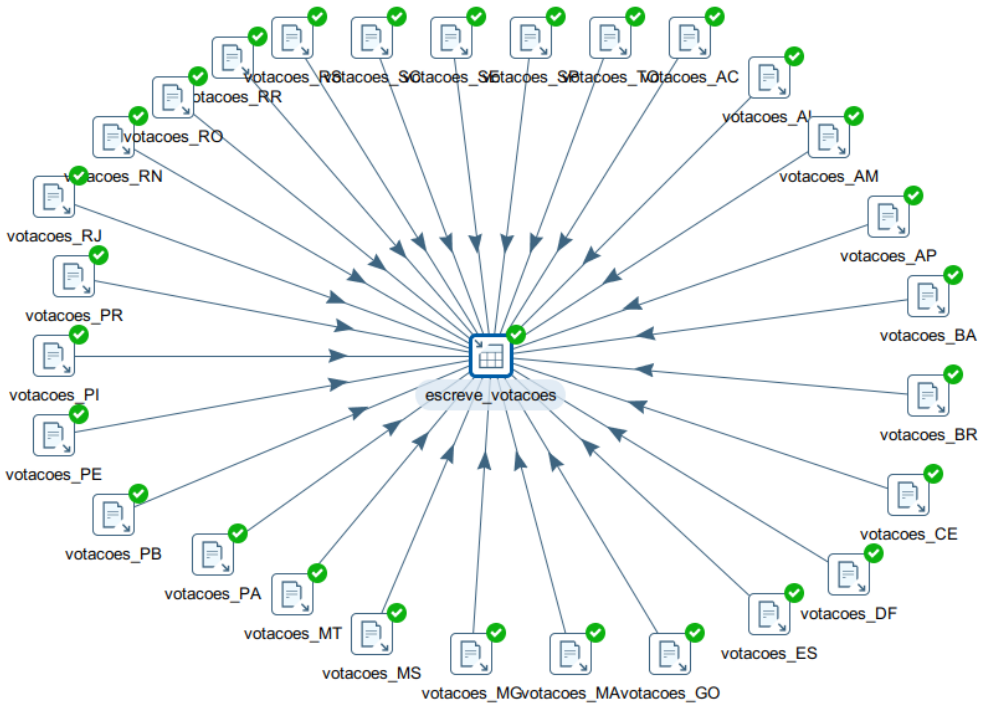

Figura 4: Transformação concluída da tabela **`votacoes`**

2 Execução e Análise das Consultas

2.1 Consulta 1

Esta consulta é do tipo *exact-match*.

1. **Tabela escolhida: `votacoes`**.
2. **Atributos escolhidos:** nome do candidato (`nm_candidato`), cargo (`ds_cargo`) e situação eleitoral (`ds_sit_tot_turno`).

A consulta abaixo mostra o nome, cargo e situação eleitoral dos(as) candidatos(as) do estado do Ceará.

```
SELECT A.nm_candidato, A.ds_cargo, A.ds_sit_tot_turno
FROM votacoes A
WHERE A.sg_uf = 'CE';
```

a) A consulta executou em **16 minutos e 11 segundos**.

b) O índice do tipo Hash foi criado em **2 horas e 10 minutos**.

```
CREATE INDEX index1
ON votacoes
USING hash (sg_uf);
```

c) Os comandos a seguir foram utilizados para limpar a *cache* do Sistema Operacional Ubuntu:

```
:/usr/bin$ sudo bash
:/usr/bin# sync
:/usr/bin# echo 3 > /proc/sys/vm/drop_caches
```

Também utilizamos as seguintes consultas para limpar a cache do **SGBD PostgreSQL** (primeiro desconectando o servidor e depois conectando novamente para poder executar cada consulta):

```
DISCARD ALL;
SET SESSION AUTHORIZATION DEFAULT;
RESET ALL;
DEALLOCATE ALL;
CLOSE ALL;
UNLISTEN *;
SELECT pg_advisory_unlock_all();
```

```
DISCARD PLANS;
DISCARD TEMP;
```

- ***Os comandos acima serão utilizados durante o decorrer do processamento das demais consultas presentes neste relatório.***

A consulta executou em **24 segundos e 694 milissegundos** utilizando o índice do tipo Hash.

	QUERY PLAN text
1	Bitmap Heap Scan on votacoes a (cost=5430.65..440367.22 rows=169632 width=...
2	Recheck Cond: ((sg_uf)::text = 'CE'::text)
3	Heap Blocks: exact=12050
4	-> Bitmap Index Scan on index1 (cost=0.00..5388.24 rows=169632 width=0) (act...
5	Index Cond: ((sg_uf)::text = 'CE'::text)
6	Planning time: 0.855 ms
7	Execution time: 24541.381 ms

Figura 5: Plano de execução da Consulta 1 utilizando índice do tipo Hash

d) O índice BTree foi criado em **2 minutos e 43 segundos**.
Consultas utilizadas:

```
DROP INDEX index1
ON votacoes;

CREATE INDEX index2
ON votacoes
USING btree (sg_uf);
```

e) A mesma consulta feita na **Alternativa c)** executou em **4 segundos e 630 milissegundos** com a utilização do índice do tipo BTree.

	QUERY PLAN text
1	Bitmap Heap Scan on votacoes a (cost=3324.12..452573.11 rows=177508 width=53) (actual time=304.467..4397.366 rows=175890 loops=1)
2	Recheck Cond: ((sg_uf)::text = 'CE'::text)
3	Heap Blocks: exact=12050
4	-> Bitmap Index Scan on index2 (cost=0.00..3279.74 rows=177508 width=0) (actual time=267.177..267.177 rows=175890 loops=1)
5	Index Cond: ((sg_uf)::text = 'CE'::text)
6	Planning time: 135.439 ms
7	Execution time: 4448.749 ms

Figura 6: Plano de execução da Consulta 1 utilizando índice do tipo BTree

f) De acordo com os planos de execução das consultas exibidos nas **Figuras 5** e **6**, podemos observar que o SGBD realizou uma varredura do tipo *Bitmap* levando em consideração o índice criado. Isso significa que o SGBD primeiro resgatou as linhas que satisfazem a condição de seleção e em seguida buscou-as na tabela. Além disso, com base no tempo de duração de cada consulta, podemos concluir que para este caso o índice do tipo BTree foi o que conseguiu tornar a consulta mais eficiente possível.

2.2 Consulta 2

Esta consulta é do tipo *select-range*.

1. **Tabela escolhida:** `votacoes`.
2. **Atributos escolhidos:** cargo do candidato (`ds_cargo`), nome (`nm_candidato`) e nome do partido (`nm_partido`).

A consulta abaixo mostra o cargo, nome e partido dos(as) candidatos(as) que tiveram acima de 50.000 votos.

```
SELECT A.ds_cargo, A.nm_candidato, A.nm_partido
FROM votacoes A
WHERE A.qt_votos_nominais > '50000';
```

a) A consulta acima executou em **2 minutos e 17 segundos**.

b) O índice do tipo BTree foi criado em **2 minutos e 48 segundos**.

```
CREATE INDEX index1
ON votacoes
USING btree (qt_votos_nominais);
```

c) A consulta executou em **963 milissegundos** utilizando o índice do tipo BTree.

	QUERY PLAN text
1	HashAggregate (cost=17446.58..17495.50 rows=4892 width=68) (actual time=10998.507..10998.629 rows=46 loops=1)
2	Group Key: ds_cargo, nm_candidato, nm_partido
3	-> Index Scan using index1 on votacoes a (cost=0.43..17409.80 rows=4904 width=68) (actual time=133.544..10991.635 rows=730 loops=1)
4	Index Cond: (qt_votos_nominais > '50000'::numeric)
5	Planning time: 141.552 ms
6	Execution time: 10999.042 ms

Figura 7: Plano de execução da Consulta 2 utilizando índice do tipo BTree

d) Ao observar o plano de execução acima da consulta com índice do tipo BTree, podemos verificar que não houveram problemas com a utilização do índice, que, pelo contrário, o SGBD faz uso do índice ao realizar o processamento da consulta.

e) O índice do tipo Hash foi criado em **6 horas e 8 minutos**. Consultas utilizadas:

```
DROP INDEX index1
ON votacoes;

CREATE INDEX index2
ON votacoes
USING hash (qt_votos_nominais);
```

f) A consulta executou em **2 minutos e 34 segundos**.

	QUERY PLAN text
1	Unique (cost=991032.50..991033.20 rows=70 width=68) (actual time=148458.083..148458.555 rows=46 loops=1)
2	-> Sort (cost=991032.50..991032.67 rows=70 width=68) (actual time=148458.079..148458.198 rows=730 loops=1)
3	Sort Key: ds_cargo, nm_candidato, nm_partido
4	Sort Method: quicksort Memory: 124kB
5	-> Gather (cost=1000.00..991030.35 rows=70 width=68) (actual time=681.507..148430.911 rows=730 loops=1)
6	Workers Planned: 2
7	Workers Launched: 2
8	-> Parallel Seq Scan on votacoes a (cost=0.00..990023.35 rows=29 width=68) (actual time=1012.883..147932.925 rows=243 loops=3)
9	Filter: (qt_votos_nominais > '50000'::numeric)
10	Rows Removed by Filter: 3028899
11	Planning time: 829.122 ms
12	Execution time: 148458.868 ms

Figura 8: Plano de execução da Consulta 2 “utilizando” índice do tipo Hash

f) Ao analisarmos a **Figura 8**, percebemos claramente que o SGBD não faz uso do índice do tipo Hash.

g) Como não tivemos problemas com a utilização do índice BTree, podemos concordar que a criação desse índice foi extremamente mais rápida do que a criação do índice Hash.

h) Concluímos que a ordem física e a ordem lógica, devido a grande quantidade de tuplas da tabela usada pela consulta, não favoreceram o processamento da mesma, podendo ser um dos motivos pelo qual o SGBD não utilizou o índice Hash criado. Sendo assim, ele preferiu ordenar o arquivo e em seguida realizou uma varredura buscando as tuplas que satisfaziam o filtro.

2.3 Consulta 3

Esta consulta é do tipo junção.

1. **Tabelas escolhidas:** **`votacoes`** e **`bens_declarados`**.
2. **Atributos escolhidos:** nome do candidato (`nm_candidato`), tido do bem do candidato (`ds_tipo_bem_candidato`) e descrição do bem do candidato (`ds_bem_candidato`).
3. **Chave primária de `votacoes`:** `sq_candidato`.
4. **Chave estrangeira de `bens_declarados`:** `sq_candidato`.

A consulta abaixo mostra o nome do(a) candidato(a), tipo do bem e o bem declarado dos(as) candidatos(as) que concorreram às eleições estaduais e federais de 2018 no Brasil.

```
SELECT DISTINCT B.nm_candidato, A.ds_tipo_bem_candidato,
A.ds_bem_candidato
FROM bens_declarados A, votacoes B
WHERE A.sq_candidato = B.sq_candidato;
```

a) A consulta acima executou em **17 minutos e 7 segundos**.

b) Criando o índice:

```
CREATE INDEX index1
ON votacoes
USING hash (sq_candidato);
```

c) A consulta executou em **9 minutos e 11 segundos**.

	QUERY PLAN text
1	Unique (cost=16352548.74..16812453.88 rows=45990514 width=96) (actual time=740675.046..798173.856 rows=65299 loops=1)
2	-> Sort (cost=16352548.74..16467525.02 rows=45990514 width=96) (actual time=740675.042..792780.643 rows=34056538 loops=1)
3	Sort Key: a.nm_candidato, b.ds_tipo_bem_candidato, b.ds_bem_candidato
4	Sort Method: external merge Disk: 3391928kB
5	-> Merge Join (cost=2597145.05..3425399.47 rows=45990514 width=96) (actual time=202063.535..229054.450 rows=34056538 loops=1)
6	Merge Cond: (((b.sq_candidato)::numeric) = a.sq_candidato)
7	-> Sort (cost=16306.26..16540.21 rows=93581 width=78) (actual time=823.974..2355.494 rows=93581 loops=1)
8	Sort Key: ((b.sq_candidato)::numeric)
9	Sort Method: external merge Disk: 9264kB
10	-> Seq Scan on bens_declarados b (cost=0.00..4417.81 rows=93581 width=78) (actual time=47.041..680.956 rows=93581 loops=1)
11	-> Materialize (cost=2580838.79..2626275.93 rows=9087428 width=34) (actual time=201239.551..215879.299 rows=37289691 loops=1)
12	-> Sort (cost=2580838.79..2603557.36 rows=9087428 width=34) (actual time=201239.546..213351.711 rows=9087428 loops=1)
13	Sort Key: a.sq_candidato
14	Sort Method: external merge Disk: 401592kB
15	-> Seq Scan on votacoes a (cost=0.00..1033567.28 rows=9087428 width=34) (actual time=53.459..146769.068 rows=9087428 loops=1)
16	Planning time: 910.380 ms
17	Execution time: 800446.754 ms

Figura 9: Plano de execução da Consulta 3 “utilizando” índice do tipo Hash em **`sq_candidato`** da tabela **`votacoes`**

De acordo com a **Figura 9**, a consulta não faz uso do índice Hash criado. Mas, o SGBD usa o método de ordenação *Quicksort* e compara as tuplas que satisfazem a condição de seleção.

d) Criando o índice:

```
CREATE INDEX index2
ON bens_declarados
USING hash (sq_candidato);
```

e) A consulta executou em **16 minutos e 7 segundos**.

	QUERY PLAN text
1	Unique (cost=16352548.74..16812453.88 rows=45990514 width=96) (actual time=943953.931..965827.114 rows=65299 loops=1)
2	-> Sort (cost=16352548.74..16467525.02 rows=45990514 width=96) (actual time=943953.929..960526.222 rows=34056538 loops=1)
3	Sort Key: a.nm_candidato, b.ds_tipo_bem_candidato, b.ds_bem_candidato
4	Sort Method: external merge Disk: 3391928kB
5	-> Merge Join (cost=2597145.05..3425399.47 rows=45990514 width=96) (actual time=204924.259..244095.913 rows=34056538 loops=1)
6	Merge Cond: (((b.sq_candidato)::numeric) = a.sq_candidato)
7	-> Sort (cost=16306.26..16540.21 rows=93581 width=78) (actual time=243.756..3123.127 rows=93581 loops=1)
8	Sort Key: ((b.sq_candidato)::numeric)
9	Sort Method: external merge Disk: 9264kB
10	-> Seq Scan on bens_declarados b (cost=0.00..4417.81 rows=93581 width=78) (actual time=0.010..35.572 rows=93581 loops=1)
11	-> Materialize (cost=2580838.79..2626275.93 rows=9087428 width=34) (actual time=204680.495..231224.624 rows=37289691 loops=1)
12	-> Sort (cost=2580838.79..2603557.36 rows=9087428 width=34) (actual time=204680.492..228914.693 rows=9087428 loops=1)
13	Sort Key: a.sq_candidato
14	Sort Method: external merge Disk: 401592kB
15	-> Seq Scan on votacoes a (cost=0.00..1033567.28 rows=9087428 width=34) (actual time=526.504..145265.009 rows=9087428 loops=1)
16	Planning time: 0.575 ms
17	Execution time: 966200.748 ms

Figura 10: Plano de execução da Consulta 3 utilizando índice do tipo Hash em **`sq_candidato`** da tabela **`bens_declarados`**

Com as informações contidas do plano de execução da consulta contida na **Figura 10**, o SGBD não utiliza o índice criado e faz uso do método de ordenação *Merge External*.

f) As respostas dos processamento de consultas acima são praticamente iguais, diferenciando apenas na escolha dos métodos para executar cada consulta, já que o SGBD não fez uso de nenhum dos índices criados.

2.4 Consulta 4

Esta consulta é do tipo agregação.

1. **Tabela escolhida:** **`votacoes`**.
2. **Atributos escolhidos:** nome do partido (`nm_partido`) e nome do candidato (`nm_candidato`).

A consulta abaixo mostra o nome do partido e a quantidade de candidatos(as) que pertencem a cada partido presente nas eleições estaduais e federais de 2018 no Brasil.

```
SELECT A.nm_partido, COUNT(A.nm_candidato)
FROM votacoes A
GROUP BY A.nm_partido;
```

a) A consulta executou em **3 minutos e 53 segundos**.

b) O índice do tipo Hash foi criado em **17 minutos e 11 segundos**.

```
CREATE INDEX index1
ON votacoes
USING hash (nm_partido);
```

c) A consulta executou em **2 minutos e 22 segundos**.

	QUERY PLAN text
1	Finalize GroupAggregate (cost=1000498.92..1000499.79 rows=35 width=33) (actual time=141973.697..141973.753 rows=35 loops=1)
2	Group Key: nm_partido
3	-> Sort (cost=1000498.92..1000499.09 rows=70 width=33) (actual time=141973.683..141973.697 rows=105 loops=1)
4	Sort Key: nm_partido
5	Sort Method: quicksort Memory: 33kB
6	-> Gather (cost=1000489.42..1000496.77 rows=70 width=33) (actual time=141947.256..141947.560 rows=105 loops=1)
7	Workers Planned: 2
8	Workers Launched: 2
9	-> Partial HashAggregate (cost=999489.42..999489.77 rows=35 width=33) (actual time=141728.502..141728.519 rows=35 loops=3)
10	Group Key: nm_partido
11	-> Parallel Seq Scan on votacoes a (cost=0.00..980557.28 rows=3786428 width=51) (actual time=66.927..138788.360 rows=3029143 loops=3)
12	Planning time: 383.280 ms

Figura 11: Plano de execução da Consulta 4 “utilizando” índice do tipo Hash

De acordo com a **Figura 11** acima, o SGBD não faz uso do índice criado. Ele faz um agrupamento dos dados, utiliza o método *Quicksort*, e, no final, faz um agrupamento Hash com o atributo.

2.5 Consulta 5

As consultas a seguir possuem uma junção, um comando de seleção e um operador especial.

Ambas criam índices do tipo BTree, já que, aparentemente, para esta base de dados, índices deste tipo funcionam mais eficientemente.

2.5.1 Consulta 5.1

```
SELECT A.nm_coligacao, SUM(B.qt_votos_nominais)
FROM filiacoes A, votacoes B
WHERE A.sq_coligacao = B.sq_coligacao
GROUP BY A.nm_coligacao;
```

a) A consulta executou em **2 minutos e 53 segundos**.

b) O tempo de criação do índice BTree **`index1`** sobre o atributo **`A.sq_coligacao`** foi de **197 milissegundos**.

c) O tempo de execução com o índice `index1` foi de **2 minutos e 51 segundos**.

d) O tempo de criação do índice BTree **`index2`** sobre o atributo **`B.sq_coligacao`** foi de **3 minutos e 51 segundos**.

e) O tempo de execução com os índices foi de 2 minutos e 59 segundos.

	QUERY PLAN text
1	Finalize GroupAggregate (cost=2427083.15..2427097.19 rows=432 width=54) (actual time=170482.568..170483.489 rows=408 loops=1)
2	Group Key: a.nm_coligacao
3	-> Sort (cost=2427083.15..2427085.31 rows=864 width=54) (actual time=170482.548..170482.633 rows=1224 loops=1)
4	Sort Key: a.nm_coligacao
5	Sort Method: quicksort Memory: 220kB
6	-> Gather (cost=2426949.21..2427041.01 rows=864 width=54) (actual time=170354.921..170470.122 rows=1224 loops=1)
7	Workers Planned: 2
8	Workers Launched: 2
9	-> Partial HashAggregate (cost=2425949.21..2425954.61 rows=432 width=54) (actual time=170067.921..170068.180 rows=408 loops=3)
10	Group Key: a.nm_coligacao
11	-> Merge Join (cost=1524410.61..2231918.00 rows=38806242 width=25) (actual time=154643.587..165505.662 rows=15290551 loops=3)
12	Merge Cond: (b.sq_coligacao = ((a.sq_coligacao)::numeric))
13	-> Sort (cost=1523693.09..1533159.16 rows=3786428 width=11) (actual time=154637.023..159308.580 rows=3029143 loops=3)
14	Sort Key: b.sq_coligacao
15	Sort Method: external merge Disk: 67848kB
16	-> Parallel Seq Scan on votacoes b (cost=0.00..980557.28 rows=3786428 width=11) (actual time=231.632..133694.154 rows=3029143 l...
17	-> Sort (cost=717.51..733.17 rows=6262 width=30) (actual time=6.554..903.785 rows=15286623 loops=3)
18	Sort Key: ((a.sq_coligacao)::numeric)
19	Sort Method: quicksort Memory: 842kB
20	-> Seq Scan on filiacoes a (cost=0.00..322.62 rows=6262 width=30) (actual time=0.027..2.620 rows=6262 loops=3)
21	Planning time: 159.917 ms
22	Execution time: 170493.655 ms

Figura 12: Plano de execução após criar o índice **`index1`**

	QUERY PLAN text
1	Finalize GroupAggregate (cost=2427083.15..2427097.19 rows=432 width=54) (actual time=177543.618..177547.026 rows=408 loops=1)
2	Group Key: a.nm_coligacao
3	-> Sort (cost=2427083.15..2427085.31 rows=864 width=54) (actual time=177543.566..177543.895 rows=1224 loops=1)
4	Sort Key: a.nm_coligacao
5	Sort Method: quicksort Memory: 220kB
6	-> Gather (cost=2426949.21..2427041.01 rows=864 width=54) (actual time=177373.563..177433.062 rows=1224 loops=1)
7	Workers Planned: 2
8	Workers Launched: 2
9	-> Partial HashAggregate (cost=2425949.21..2425954.61 rows=432 width=54) (actual time=176984.141..176984.377 rows=408 loops=3)
10	Group Key: a.nm_coligacao
11	-> Merge Join (cost=1524410.61..2231918.00 rows=38806242 width=25) (actual time=159998.392..172495.357 rows=15290551 loops=3)
12	Merge Cond: (b.sq_coligacao = ((a.sq_coligacao)::numeric))
13	-> Sort (cost=1523693.09..1533159.16 rows=3786428 width=11) (actual time=159991.561..166406.891 rows=3029143 loops=3)
14	Sort Key: b.sq_coligacao
15	Sort Method: external merge Disk: 67680kB
16	-> Parallel Seq Scan on votacoes b (cost=0.00..980557.28 rows=3786428 width=11) (actual time=273.924..139259.754 rows=3029143 loops=3)
17	-> Sort (cost=717.51..733.17 rows=6262 width=30) (actual time=6.821..887.729 rows=15286623 loops=3)
18	Sort Key: ((a.sq_coligacao)::numeric)
19	Sort Method: quicksort Memory: 842kB
20	-> Seq Scan on filiacoes a (cost=0.00..322.62 rows=6262 width=30) (actual time=0.028..2.918 rows=6262 loops=3)
21	Planning time: 170.090 ms
22	Execution time: 177557.048 ms

Figura 13: Plano de execução após criar ambos os índices

2.5.2 Consulta 5.2

```
SELECT A.sg_uf, B.nm_candidato, SUM(A.qt_votos_nominais)
FROM votacoes A, candidatos B
WHERE A.sq_candidato = B.sq_candidato
GROUP BY A.sg_uf, B.nm_candidato;
```

a) O tempo de execução sem os índices foi de **3 minutos e 40 segundos**.

b) O tempo de criação do índice BTree **`index1`** sobre o atributo **`A.sq_candidato`** foi de **3 minutos e 11 segundos**.

c) O tempo de execução com o índice **`index1`** foi de **3 minutos e 12 segundos**.

d) O tempo de criação do índice BTree **`index2`** sobre o atributo **`B.sq_candidato`** foi de **238 milissegundos**.

e) O tempo de execução com os índices foi de **2 minutos e 56 segundos**.

	QUERY PLAN text
1	Finalize GroupAggregate (cost=2621112.12..2896103.74 rows=780300 width=61) (actual time=173755.260..176138.968 rows=26170 loops=1)
2	Group Key: a.sg_uf, b.nm_candidato
3	-> Gather Merge (cost=2621112.12..2870743.99 rows=1560600 width=61) (actual time=173755.243..179582.769 rows=78502 loops=1)
4	Workers Planned: 2
5	Workers Launched: 2
6	-> Partial GroupAggregate (cost=2620112.10..2689612.01 rows=780300 width=61) (actual time=171612.673..173842.296 rows=26167 loops=3)
7	Group Key: a.sg_uf, b.nm_candidato
8	-> Sort (cost=2620112.10..2635048.64 rows=5974616 width=32) (actual time=171612.648..173013.039 rows=3081490 loops=3)
9	Sort Key: a.sg_uf, b.nm_candidato
10	Sort Method: external merge Disk: 129616kB
11	-> Merge Join (cost=1528801.71..1661755.68 rows=5974616 width=32) (actual time=159660.066..166324.024 rows=3081490 loops=3)
12	Merge Cond: (a.sq_candidato = ((b.sq_candidato)::numeric))
13	-> Sort (cost=1523693.09..1533159.16 rows=3786428 width=14) (actual time=159628.975..164517.178 rows=3029143 loops=3)
14	Sort Key: a.sq_candidato
15	Sort Method: external merge Disk: 76288kB
16	-> Parallel Seq Scan on votacoes a (cost=0.00..980557.28 rows=3786428 width=14) (actual time=238.331..135607.422 rows=3029143 loops=3)
17	-> Sort (cost=5108.62..5181.56 rows=29177 width=34) (actual time=31.081..199.444 rows=3080616 loops=3)
18	Sort Key: ((b.sq_candidato)::numeric)
19	Sort Method: quicksort Memory: 4079kB
20	-> Seq Scan on candidatos b (cost=0.00..2944.77 rows=29177 width=34) (actual time=0.028..15.474 rows=29177 loops=3)
21	Planning time: 91.053 ms
22	Execution time: 179705.768 ms

Figura 14: Plano de execução após criar o índice **`index1`**

	QUERY PLAN text
1	Finalize GroupAggregate (cost=2621112.12..2896103.74 rows=780300 width=61) (actual time=168935.235..171013.234 rows=26170 loops=1)
2	Group Key: a.sg_uf, b.nm_candidato
3	-> Gather Merge (cost=2621112.12..2870743.99 rows=1560600 width=61) (actual time=168935.217..174664.150 rows=78501 loops=1)
4	Workers Planned: 2
5	Workers Launched: 2
6	-> Partial GroupAggregate (cost=2620112.10..2689612.01 rows=780300 width=61) (actual time=166675.682..168640.903 rows=26167 loops=3)
7	Group Key: a.sg_uf, b.nm_candidato
8	-> Sort (cost=2620112.10..2635048.64 rows=5974616 width=32) (actual time=166675.653..167894.167 rows=3081490 loops=3)
9	Sort Key: a.sg_uf, b.nm_candidato
10	Sort Method: external merge Disk: 129344kB
11	-> Merge Join (cost=1528801.71..1661755.68 rows=5974616 width=32) (actual time=155500.637..162177.007 rows=3081490 loops=3)
12	Merge Cond: (a.sq_candidato = ((b.sq_candidato)::numeric))
13	-> Sort (cost=1523693.09..1533159.16 rows=3786428 width=14) (actual time=155472.270..160582.592 rows=3029143 loops=3)
14	Sort Key: a.sq_candidato
15	Sort Method: external merge Disk: 76200kB
16	-> Parallel Seq Scan on votacoes a (cost=0.00..980557.28 rows=3786428 width=14) (actual time=215.708..133890.281 rows=3029143 loops=3)
17	-> Sort (cost=5108.62..5181.56 rows=29177 width=34) (actual time=28.351..174.274 rows=3080616 loops=3)
18	Sort Key: ((b.sq_candidato)::numeric)
19	Sort Method: quicksort Memory: 4079kB
20	-> Seq Scan on candidatos b (cost=0.00..2944.77 rows=29177 width=34) (actual time=0.047..13.798 rows=29177 loops=3)
21	Planning time: 10.960 ms
22	Execution time: 175284.371 ms

Figura 15: Plano de execução após criar ambos os índices

2.5.3 Análise

As consultas executam na mesma proporção de tempo já que o SGBD não faz uso dos índices criados. Tendo em média, 4 minutos de execução cada.