

Portfolio 4 – Yuri Garcia Campos

2- Faça uma consulta que selecione os dados dos empregados de nome "Mary".

```
select * from empregado
where nome_empregado = 'Mary'
```

2.1 Guarde o tempo de processamento.

201.980ms

2.2 Quantas tuplas recuperou?

3539

2.3 Aplique a instrução EXPLAIN (ou melhor EXPLAIN ANALYZE) para observar o plano de execução da consulta.

1	Gather (cost=1000.00..34474.37 rows=2867 width=103) (actual time=2.483..990.172 rows=3539 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..33187.67 rows=1195 width=103) (actual time=3.666..941.501 rows=1180 loops=3)
5	[...] Filter: ((nome_empregado)::text = 'Mary'::text)
6	[...] Rows Removed by Filter: 665487
7	Planning Time: 0.078 ms

2.4 Quantas tuplas varreu?

665487+3539=669026

3- Faça uma consulta que selecione os dados dos empregados de nome "Mary" OU dos empregados do departamento 10 (use

operador lógico OR).

```
select * from empregado
where nome_empregado = 'Mary' OR depto = 10
```

3.1 Guarde o tempo de processamento.

1090.391ms

3.2 Aplique a instrução EXPLAIN (ou melhor EXPLAIN ANALYZE) para observar o plano de execução da consulta.

1	Gather (cost=1000.00..45624.70 rows=93537 width=103) (actual time=0.786..1018.241 rows=92072 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..35271.00 rows=38974 width=103) (actual time=0.660..963.936 rows=30691 loops=3)
5	[...] Filter: (((nome_empregado)::text = 'Mary'::text) OR (depto = 10))
6	[...] Rows Removed by Filter: 635976
7	Planning Time: 0.057 ms

3.3 Faça um AND das condições e repita o processo de observação em 3.1 e 3.2.

189,248ms

1	Gather (cost=1000.00..36284.00 rows=130 width=103) (actual time=132.287..968.787 rows=131 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..35271.00 rows=54 width=103) (actual time=75.400..915.053 rows=44 loops=3)
5	[...] Filter: (((nome_empregado)::text = 'Mary'::text) AND (depto = 10))
6	[...] Rows Removed by Filter: 666623
7	Planning Time: 0.081 ms

4- Crie um índice sobre o nome de empregado. (Observe o tempo de criação do índice!)

5000,330ms

4.1 Qual o método de indexação (ED) que você usou?

Método padrão do postgres: Btree

5- Repita 2 e 3.

8	<code>explain analyze</code>
9	<code>select * from empregado</code>
10	<code>where nome_empregado='Mary'</code>
11	
Query History Data Output Explain Messages Notifications	
QUERY PLAN	
text	
1	Bitmap Heap Scan on empregado (cost=34.65..8076.41 rows=2867 width=103) (actual time=2.407..5.158 rows=3539 loops=1)
2	[...] Recheck Cond: ((nome_empregado)::text = 'Mary'::text)
3	[...] Heap Blocks: exact=2099
4	[...] -> Bitmap Index Scan on index_nome (cost=0.00..33.93 rows=2867 width=0) (actual time=2.109..2.109 rows=3539 loops=1)
5	[...] Index Cond: ((nome_empregado)::text = 'Mary'::text)

56ms


3539tuplas

```

8  explain analyze
9  select * from empregado|
10 where nome_empregado='Mary' or depto = 10

```

Query History Data Output Explain Messages Notifications

QUERY PLAN		
	text	
1	Gather (cost=1000.00..45624.70 rows=93537 width=103) (actual time=0.486..1013.203 rows=92072 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..35271.00 rows=38974 width=103) (actual time=0.484..957.963 rows=30691 loops=3)	
5	[...] Filter: (((nome_empregado)::text = 'Mary'::text) OR (depto = 10))	
6	[...] Rows Removed by Filter: 635976	
7	Planning Time: 0.067 ms	

```

8  explain analyze
9  select * from empregado
10 where nome_empregado='Mary' and depto = 10

```

Query History Data Output Explain Messages Notifications

QUERY PLAN	
text	
1	Bitmap Heap Scan on empregado (cost=33.96..8082.89 rows=130 width=103) (actual time=0.404..1.839 rows=131 loops=1)
2	[...] Recheck Cond: ((nome_empregado)::text = 'Mary'::text)
3	[...] Filter: (depto = 10)
4	[...] Rows Removed by Filter: 3408
5	[...] Heap Blocks: exact=2099
6	[...] -> Bitmap Index Scan on index_nome (cost=0.00..33.93 rows=2867 width=0) (actual time=0.179..0.179 rows=3539 loops=1)
7	[...] Index Cond: ((nome_empregado)::text = 'Mary'::text)
8	Planning Time: 0.056 ms

6- Repita a consulta do item 3 agora usando UNION (ao invés de OR).

```

13 explain analyze
14 select * from empregado
15 where nome_empregado = 'Mary'
16 union
17 select * from empregado
18 where depto = 10

```

Query History Data Output Explain Messages Notifications

	QUERY PLAN	
	text	
1	Unique (cost=84175.25..86282.76 rows=93667 width=566) (actual time=1258.509..1342.871 rows=92072 loops=1)	
2	[...] -> Sort (cost=84175.25..84409.42 rows=93667 width=566) (actual time=1258.507..1313.474 rows=92203 loops=1)	
3	[...] Sort Key: empregado.cpf, empregado.nome_empregado, empregado.salario, empregado.tipo_empregado, empregado.depto, empreg...	
4	[...] Sort Method: external merge Disk: 4272kB	
5	[...] -> Append (cost=34.65..52749.08 rows=93667 width=566) (actual time=0.378..898.712 rows=92203 loops=1)	
6	[...] -> Bitmap Heap Scan on empregado (cost=34.65..8076.41 rows=2867 width=103) (actual time=0.378..2.726 rows=3539 loops=1)	
7	[...] Recheck Cond: ((nome_empregado)::text = 'Mary'::text)	
8	[...] Heap Blocks: exact=2099	
9	[...] -> Bitmap Index Scan on index_nome (cost=0.00..33.93 rows=2867 width=0) (actual time=0.191..0.192 rows=3539 loops=1)	
10	[...] Index Cond: ((nome_empregado)::text = 'Mary'::text)	
11	[...] -> Gather (cost=1000.00..43267.67 rows=90800 width=103) (actual time=0.453..888.126 rows=88664 loops=1)	
12	[...] Workers Planned: 2	
13	[...] Workers Launched: 2	
14	[...] -> Parallel Seq Scan on empregado empregado_1 (cost=0.00..33187.67 rows=37833 width=103) (actual time=0.536..888.625 rows=...	
15	[...] Filter: (depto = 10)	
16	[...] Rows Removed by Filter: 637112	
17	Planning Time: 0.157 ms	

6.1 O plano de execução usou o índice?

Sim

6.2 A varredura da segunda condição aconteceu encima do arquivo de dados ou do índice?

Arquivo de dados

6.3 Qual consulta foi executada de maneira mais eficiente? Explique.

A consulta pelo nome foi mais eficiente pois ela usou o índice.

6.4 Qual das duas consultas (3 ou 6) você escolheria (baseado no plano de execução com índices criados)?

A 3, já que ela gasta menos tempo e faz menos processos.

7- Consulte todos os empregados cujo nome começa com a letra M.

```

select * from empregado
where nome_empregado LIKE 'M%'

```

7.1- Observe se o plano de execução usou o índice.

Não usou

8- Consulte todos os empregados que tem "an" como substring do nome.

```
27 select * from empregado
28 where nome_empregado LIKE '%an%'
```

8.1- Observe se o plano de execução usou o índice. Por quê?

Não usou, porque para o índice ser usado precisa ter uma condição de igualdade.

8.2- Repita a consulta com o operador ILIKE. O que observou?

Agora como não tem distinção entre letras maiúsculas e minúsculas ele também selecionou nomes que iniciam com “an”.

9- Acrescente uma ordenação pelo nome do empregado às consultas dos itens 7 e 8.

9.1- O que acontece no plano de execução. Por quê?

Quando acrescentado o order by o otimizador gasta vários passos para fazer a ordenação e nesses passos acontece um external merge disk, já que são muitos dados.

10- Crie um índice composto BTree para os campos nome_empregado e depto (nessa ordem)

11- Execute as consultas nos itens 3.3 e 7.

11.1- Qual índice foi utilizado no plano de execução?

No item 7 nenhum e no item 3.3 foi utilizado o índice composto.

11.2- Observe as diferenças entre os planos de execução

Com o índice composto o item 3.3 foi capaz de procurar o nome e o depto diretamente pelo índice enquanto quando não usou o índice composto foi preciso fazer um filtro para encontrar as tuplas com o depto necessário.

11.3- Substitua o AND da condição pelo OR. O que acontece? Por quê? (considere na resposta os dois índices: o simples e o composto)

Quando usado o “OR” o otimizador opta por não usar nenhum dos dois índices já que ele quebra um pouco a condição de igualdade, diferente de quando usamos o “and” que adiciona na condição de igualdade, onde o otimizador prefere o índice composto.

12- Execute a consulta que retorna os nomes dos empregados do departamento de contabilidade (use junção).

```
select nome_empregado
from empregado inner join departamento
on empregado.depto = departamento.cod
where empregado.depto = 5
```

12.1- Você consegue melhorar o desempenho da consulta? Tente criar um índice no atributo depto (campo da junção) da tabela de empregados.

Sim, criando o índice depto o otimizador pode procurar diretamente nele o departamento igual a 5.

```

explain analyze
select nome_empregado
from empregado inner join departamento
on empregado.depto = departamento.cod
where empregado.depto = 5

```

ry History Data Output Explain Messages Notifications

QUERY PLAN	
text	
Nested Loop (cost=3130.05..32213.80 rows=280467 width=7) (actual time=39.625..2709.095 rows=282844 loops=1)	
[...] -> Seq Scan on departamento (cost=0.00..2.25 rows=1 width=4) (actual time=0.008..0.014 rows=1 loops=1)	
[...] Filter: (cod = 5)	
[...] Rows Removed by Filter: 99	
[...] -> Bitmap Heap Scan on empregado (cost=3130.05..29406.88 rows=280467 width=11) (actual time=39.613..2683.543 rows=282844 loops=1)	
[...] Recheck Cond: (depto = 5)	
[...] Heap Blocks: exact=10834	
[...] -> Bitmap Index Scan on index_dept (cost=0.00..3059.93 rows=280467 width=0) (actual time=38.304..38.304 rows=282844 loops=1)	
[...] Index Cond: (depto = 5)	
Planning Time: 2.087 ms	

13- Refaça a consulta do item 12 usando o operador de conjuntos IN.

```

39 explain analyze
40 select nome_empregado
41 from empregado inner join departamento
42 on empregado.depto = departamento.cod
43 where empregado.depto in ('5')
44

```

Query History Data Output Explain Messages Notifications

	QUERY PLAN	
	text	
1	Nested Loop (cost=3130.05..32213.80 rows=280467 width=7) (actual time=13.964..89.111 rows=282844 loops=1)	
2	[...] -> Seq Scan on departamento (cost=0.00..2.25 rows=1 width=4) (actual time=0.016..0.022 rows=1 loops=1)	
3	[...] Filter: (cod = 5)	
4	[...] Rows Removed by Filter: 99	
5	[...] -> Bitmap Heap Scan on empregado (cost=3130.05..29406.88 rows=280467 width=11) (actual time=13.943..63.396 rows=2828...	
6	[...] Recheck Cond: (depto = 5)	
7	[...] Heap Blocks: exact=10834	
8	[...] -> Bitmap Index Scan on index_dept (cost=0.00..3059.93 rows=280467 width=0) (actual time=12.277..12.277 rows=282844 loo...	
9	[...] Index Cond: (depto = 5)	
10	Planning Time: 0.187 ms	

13.1- Compare os planos de execução das duas consultas.

Quando usado o IN o tempo de execução diminui pois muda o método de varredura.

13.2- Refaça a consulta usando o operador = (ao invés do IN). Compare de novo. Tempo de execução?

Dessa vez o tempo de execução aumentou muito pouco.

14- Existe algum outro recurso que você possa estar utilizando para melhorar ainda mais o tempo de execução das consultas dos itens 12 e 13? Qual? Teste! Fez diferença? Por quê? Olhe o plano da consulta!

Na consulta 12 podemos melhorar o tempo de execução trocando o sinal de igual pelo operador LIKE.

15- Refaça a consulta do item 12 usando a cláusula inner join no FROM. Alguma diferença?

Nenhuma diferença nos passos de execução.

16- Acrescente mais uma condição à consulta do item 12: E tipo de empregado é contador

```
46 explain analyze
47 select nome_empregado
48 from empregado inner join departamento
49 on empregado.depto = departamento.cod
50 where empregado.depto = 5 and empregado.tipo_empregado = 10
51
52
```

Query History	Data Output	Explain	Messages	Notifications
	QUERY PLAN text			
1	Nested Loop (cost=3067.20..30338.30 rows=29084 width=7) (actual time=6.479..44.545 rows=29557 loops=1)			
2	[...] -> Seq Scan on departamento (cost=0.00..2.25 rows=1 width=4) (actual time=0.012..0.018 rows=1 loops=1)			
3	[...] Filter: (cod = 5)			
4	[...] Rows Removed by Filter: 99			
5	[...] -> Bitmap Heap Scan on empregado (cost=3067.20..30045.21 rows=29084 width=11) (actual time=6.465..42.034 rows=29557 loops=1)			
6	[...] Recheck Cond: (depto = 5)			
7	[...] Filter: (tipo_empregado = 10)			
8	[...] Rows Removed by Filter: 253287			
9	[...] Heap Blocks: exact=10834			
10	[...] -> Bitmap Index Scan on index_dept (cost=0.00..3059.93 rows=280467 width=0) (actual time=5.359..5.359 rows=282844 loops=1)			
11	[...] Index Cond: (depto = 5)			
12	Planning Time: 0.089 ms			

16.1- Leia com atenção e interprete o plano da consulta.

Varredura sequencial em departamento, filtrando pelo cod=5. Removeu 99 tuplas usando o filtro.

Faz um heap scan em empregado com a condição de depto=5. Filtra o tipo_empregado=10 e remove 253287 tuplas.

Sobram 10834 blocos.

Usa o index para achar o depto = 5.

17- Consulte os empregados com salário maior que 1000 reais.

54	<code>explain analyze</code>
55	<code>select nome_empregado</code>
56	<code>from empregado</code>
57	<code>where salario > 1000</code>
58	
Query History Data Output Explain Mess	
	nome_empregado character varying (200)
1	Vessie
2	Winifred
3	Marie
4	Jean

17.1- Tempo de processamento?

702,104ms

17.2- Melhora se criarmos um índice sobre o campo salário? O plano de consulta contém o índice criado? Por quê?

Não, pois o uso do índice se limita a igualdade e não ao ">".

17.3- Encontre uma explicação para a grande demora na construção do índice do item anterior. Sugira um caso em que demoraria menos.

Pois existem muitos dados para organizar no índice e isso causa colisões.

17.4- Faça agora a consulta para salários menores que 1000 reais. O plano de consulta contém o índice criado? Por quê?

Também não contem, pelo mesmo motivo de antes.

17.5- Avalie se a mesma explicação se aplica para as consultas: recupere os dados de empregados dos departamentos 10 ao 20. E do 10 ao 30?

Não, pois nesse caso é um intervalo de valores de índice e o otimizador consegue buscar os índices 10,11,12,...20 como se fossem várias igualdades.

18- Procure o mínimo e máximo salário dos empregados da empresa.

```
explain analyze
SELECT MAX(salario) FROM empregado
SELECT MIN(salario) FROM empregado
```

18.1- Observe o plano de consulta. Como o índice foi usado em ambos os casos (min e max)?

O otimizador simplesmente olhou qual era a primeira e a última posição do índice respectivamente.

19- Construa outras consultas que usem as cláusulas de ORDER BY, GROUP BY ou DISTINCT. Procure explicações relacionadas aos algoritmos e recursos utilizados nos planos de consulta.

Após uma varredura sequencial é criada uma tabela hash para agrupar as tuplas e depois ordena-las. Depois é feito um merge com os processos, agrupando tudo e por último é ordenado pela média de salários.

20- Criar consulta com duas soluções SQL (por exemplo: usando junção, usando aninhamento de consultas). Qual das duas consultas SQL você escolheria para implementação (baseado no plano de execução com índices)?

Primeiro ocorre a varredura, por ter muita coisa pra varrer não é usado o índice, e depois é feito a contagem.