Portfolio 5 – Yuri Garcia Campos

A cada passo, procure se perguntar: Por quê o otimizador escolheu esse plano de execução para essa consulta?

Nos exercícios 1 e 2, crie tabelas comparativas contendo o tempo de execução das consultas.

1- Compare os planos de execução das consultas a seguir (fazendo a combinação de usar ou não índices em ambos os campos nome_empregado e depto):

i)

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

Sem index:

1	Gather (cost=1000.00..45381.80 rows=91108 width=103) (actual time=2.164..1274.761 rows=89480 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..35271.00 rows=37962 width=103) (actual time=1.179..1212.449 rows=29827 loops=3)
5	[...] Filter: (((nome_empregado)::text = 'Michael'::text) OR (depto = 10))
6	[...] Rows Removed by Filter: 636840
7	Planning Time: 0.163 ms
8	Execution Time: 1278.782 ms

Com index:

1	Gather (cost=1000.00..45381.80 rows=91108 width=103) (actual time=0.462..1208.442 rows=89480 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..35271.00 rows=37962 width=103) (actual time=0.468..1141.393 rows=29827 loops=3)
5	[...] Filter: (((nome_empregado)::text = 'Michael'::text) OR (depto = 10))
6	[...] Rows Removed by Filter: 636840
7	Planning Time: 0.068 ms
8	Execution Time: 1212.217 ms

Nesse caso pela pouca quantidade de dados mesmo com os indexs ativados o otimizador optou por não usa-lo.

ii)

```
select * from empregrado
where nome_empregado = 'Michael'
union
select * from empregado
where depto = 10
```

Sem index:

1	Unique (cost=108411.98..110462.24 rows=91123 width=566) (actual time=2736.726..2815.542 rows=89480 loops=1)
2	[...] -> Sort (cost=108411.98..108639.78 rows=91123 width=566) (actual time=2736.724..2789.812 rows=89525 loops=1)
3	[...] Sort Key: empregado.cpf, empregado.nome_empregado, empregado.salario, empregado.tipo_empregado, empregado.depto, empregad...
4	[...] Sort Method: external merge Disk: 4152kB
5	[...] -> Gather (cost=1000.00..77854.48 rows=91123 width=566) (actual time=73.049..2373.133 rows=89525 loops=1)
6	[...] Workers Planned: 2
7	[...] Workers Launched: 2
8	[...] -> Parallel Append (cost=0.00..67742.18 rows=91123 width=566) (actual time=30.739..2354.021 rows=29842 loops=3)
9	[...] -> Parallel Seq Scan on empregado (cost=0.00..33187.67 rows=37833 width=103) (actual time=3.537..1196.276 rows=29555 loops=3)
10	[...] Filter: (depto = 10)
11	[...] Rows Removed by Filter: 637112
12	[...] -> Parallel Seq Scan on empregado empregado_1 (cost=0.00..33187.67 rows=135 width=103) (actual time=41.117..1731.826 rows=43...
13	[...] Filter: ((nome_empregado)::text = 'Michael'::text)
14	[...] Rows Removed by Filter: 999570
15	Planning Time: 9.023 ms
16	JIT:
17	[...] Functions: 25
18	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true
19	[...] Timing: Generation 2.401 ms, Inlining 0.000 ms, Optimization 11.604 ms, Emission 73.270 ms, Total 87.275 ms
20	Execution Time: 3043.108 ms

Com index:

1	Unique (cost=76372.64..78422.91 rows=91123 width=566) (actual time=1748.293..1828.947 rows=89480 loops=1)
2	[...] -> Sort (cost=76372.64..76600.45 rows=91123 width=566) (actual time=1748.292..1800.307 rows=89525 loops=1)
3	[...] Sort Key: empregado.cpf, empregado.nome_empregado, empregado.salario, empregado.tipo_empregado, empregado.depto, empregad...
4	[...] Sort Method: external merge Disk: 4152kB
5	[...] -> Append (cost=6.93..45815.14 rows=91123 width=566) (actual time=1.944..1381.701 rows=89525 loops=1)
6	[...] -> Bitmap Heap Scan on empregado (cost=6.93..1180.63 rows=323 width=103) (actual time=1.943..300.474 rows=861 loops=1)
7	[...] Recheck Cond: ((nome_empregado)::text = 'Michael'::text)
8	[...] Heap Blocks: exact=861
9	[...] -> Bitmap Index Scan on index_nome_e_depto (cost=0.00..6.85 rows=323 width=0) (actual time=1.758..1.758 rows=861 loops=1)
10	[...] Index Cond: ((nome_empregado)::text = 'Michael'::text)
11	[...] -> Gather (cost=1000.00..43267.67 rows=90800 width=103) (actual time=0.289..1073.202 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=0.310..1130.650 rows=2...
15	[...] Filter: (depto = 10)
16	[...] Rows Removed by Filter: 637112
17	Planning Time: 0.183 ms
18	Execution Time: 1834.487 ms

Logo na primeira linha podemos ver que ao usar o index o custo reduziu. Sem o index foi feito um scan sequencial, ou seja, varreu tudo enquanto com o index foi usado o bitmap heap scan para diminuir o local de busca para 861 heap blocks e podemos ver que a quantidade das linhas removidas por filtro é bem maior na consulta com index e ela também é mais rápida.

iii)

```
select * from empregado
where nome_empregado='Michael' AND depto = 10
```

Sem index:

1	Gather (cost=1000.00..36272.50 rows=15 width=103) (actual time=104.843..1297.949 rows=45 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..35271.00 rows=6 width=103) (actual time=145.437..1216.396 rows=15 loops=3)
5	[...] Filter: (((nome_empregado)::text = 'Michael'::text) AND (depto = 10))
6	[...] Rows Removed by Filter: 666652
7	Planning Time: 0.084 ms
8	Execution Time: 1297.977 ms

Com index:

1	Bitmap Heap Scan on empregado (cost=4.58..63.65 rows=15 width=103) (actual time=0.028..0.083 rows=45 loops=1)
2	[...] Recheck Cond: (((nome_empregado)::text = 'Michael'::text) AND (depto = 10))
3	[...] Heap Blocks: exact=45
4	[...] -> Bitmap Index Scan on index_nome_e_depto (cost=0.00..4.58 rows=15 width=0) (actual time=0.019..0.019 rows=45 loops=1)
5	[...] Index Cond: (((nome_empregado)::text = 'Michael'::text) AND (depto = 10))
6	Planning Time: 0.077 ms
7	Execution Time: 0.103 ms

Nesse caso a consulta sem index faz uma varredura sequencial procurando pelo que foi pedido, já na consulta com index o otimizador percebe a existência de um index com os mesmos dados que ele precisa e vai direto nele, o que reduz drasticamente o tempo de execução.

2- Crie índices e observe os planos de execução das consultas antes e depois do índice ser criado. Compare os planos de consulta. Por que são diferentes? Acrescente os planos das consultas à resposta!

a)

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

Sem index:

1	Gather (cost=1000.00..34219.97 rows=323 width=103) (actual time=4.440..1108.510 rows=861 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..33187.67 rows=135 width=103) (actual time=2.326..1041.658 rows=287 loops=3)
5	[...] Filter: ((nome_empregado)::text = 'Michael'::text)
6	[...] Rows Removed by Filter: 666380
7	Planning Time: 0.087 ms
8	Execution Time: 1108.651 ms

Aqui o otimizador simplesmente vai na coluna do nome e faz uma varredura pela palavra “Michael”.

Com index:

1	Bitmap Heap Scan on empregado (cost=6.93..1180.63 rows=323 width=103) (actual time=0.178..0.768 rows=861 loops=1)
2	[...] Recheck Cond: ((nome_empregado)::text = 'Michael'::text)
3	[...] Heap Blocks: exact=861
4	[...] -> Bitmap Index Scan on index_nome_e_depto (cost=0.00..6.85 rows=323 width=0) (actual time=0.087..0.087 rows=861 loops=1)
5	[...] Index Cond: ((nome_empregado)::text = 'Michael'::text)
6	Planning Time: 0.068 ms
7	Execution Time: 0.811 ms

Aqui o otimizador utiliza do index da coluna nome o que reduz muito a busca e faz o tempo de execução diminuir muito.

b)

```
select * from empregado
where nome_empregado iLIKE 'Michael'
```

Sem index:

1	Gather (cost=1000.00..34204.67 rows=170 width=103) (actual time=10.063..1453.278 rows=861 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..33187.67 rows=71 width=103) (actual time=6.195..1390.505 rows=287 loops=3)
5	[...] Filter: ((nome_empregado)::text ~~* 'Michael'::text)
6	[...] Rows Removed by Filter: 666380
7	Planning Time: 0.239 ms
8	Execution Time: 1453.457 ms

Com index:

1	Gather (cost=1000.00..34204.67 rows=170 width=103) (actual time=4.855..1364.010 rows=861 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..33187.67 rows=71 width=103) (actual time=4.361..1304.332 rows=287 loops=3)
5	[...] Filter: ((nome_empregado)::text ~~* 'Michael'::text)
6	[...] Rows Removed by Filter: 666380
7	Planning Time: 0.148 ms
8	Execution Time: 1364.161 ms

Nesse caso foi pedido uma palavra “próxima” de “Michael” e não uma palavra IGUAL e o uso do index exige o igual, por isso o index não foi usado em nenhuma consulta.

c)

```
select * from empregado
where nome_empregado LIKE 'Michael%'
```

Sem index:

1	Gather (cost=1000.00..34204.67 rows=170 width=103) (actual time=2.601..1155.903 rows=2028 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..33187.67 rows=71 width=103) (actual time=1.904..1094.192 rows=676 loops=3)
5	[...] Filter: ((nome_empregado)::text ~~ 'Michael%'::text)
6	[...] Rows Removed by Filter: 665991
7	Planning Time: 0.056 ms
8	Execution Time: 1156.148 ms

Com index:

1	Gather (cost=1000.00..34204.67 rows=170 width=103) (actual time=2.737..1172.777 rows=2028 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..33187.67 rows=71 width=103) (actual time=1.795..1102.569 rows=676 loops=3)
5	[...] Filter: ((nome_empregado)::text ~~ 'Michael%'::text)
6	[...] Rows Removed by Filter: 665991
7	Planning Time: 0.053 ms
8	Execution Time: 1173.022 ms

Nesse caso foi pedido uma palavra “próxima” de “Michael%” e não uma palavra IGUAL e o uso do index exige o igual, por isso o index não foi usado em nenhuma consulta.

d)

```
select * from empregado
where nome_empregado LIKE '%Michael'
```

Sem index:

1	Gather (cost=1000.00..34204.67 rows=170 width=103) (actual time=3.079..1129.369 rows=861 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..33187.67 rows=71 width=103) (actual time=2.023..1069.104 rows=287 loops=3)
5	[...] Filter: ((nome_empregado)::text ~~ '%Michael'::text)
6	[...] Rows Removed by Filter: 666380
7	Planning Time: 0.057 ms
8	Execution Time: 1129.507 ms

Com index:

1	Gather (cost=1000.00..34204.67 rows=170 width=103) (actual time=3.860..1156.894 rows=861 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on empregado (cost=0.00..33187.67 rows=71 width=103) (actual time=2.294..1094.206 rows=287 loops=3)
5	[...] Filter: ((nome_empregado)::text ~~ '%Michael'::text)
6	[...] Rows Removed by Filter: 666380
7	Planning Time: 0.058 ms
8	Execution Time: 1157.031 ms

Nesse caso foi pedido uma palavra “próxima” de “%Michael” e não uma palavra IGUAL e o uso do index exige o igual, por isso o index não foi usado em nenhuma consulta.

3- Observe que na criação dos índices podemos modificar o valor do parâmetro FILLFACTOR. Procura na documentação e
inclua aqui suas referências.

3.1- O que significa esse parâmetro?

É um parâmetro para determinar o quão cheio vai estar as páginas do index.
https://www.postgresql.org/docs/13/sql-createindex.html

3.2- Qual o valor padrão deste parâmetro para índices BTree?

O valor padrão é 90.
https://www.postgresql.org/docs/current/sql-createindex.html#:~:text=B-trees%20use%20a%20default,to%20100%20can%20be%20selected.

3.3- Quais são as vantagens e desvantagens de configurar o FILLFACTOR como 30?

A vantagem é que quando inserido novos dados vai ter um espaço para eles, sem que precise ser criadas novas páginas, mas em contra ponto quando é usado um valor tão baixo você gasta muito espaço em disco para armazenar pouca coisa.

3.4- Quais são as vantagens e desvantagens de configurar o FILLFACTOR como 80 ou 90?

A desvantagem é que quando inserido novos dados não vai ter um espaço para eles, precisando ser criadas novas páginas, mas em contra ponto quando é usado um valor tão alto você gasta muito menos espaço em disco para armazenar os dados.

3.5- Em quais casos é positivo configurar o FILLFACTOR como 100?

Em casos em que a tabela é estática.