Victor Eduardo Requia 15/06/2022 BAN II

1)Recupere o CPF e o nome dos mecânicos que trabalham nos setores maiores que 100 e menores que 200.

1° Consulta sem otimização

```
explain analyse select m.cpf, m.nome, s.cods from mecanico m
join setor s using (cods)
where s.cods > 100 and s.cods < 200;
```

Resultado:

```
"Hash Join  (cost=11.50..254.76 rows=99 width=28) (actual time=0.041..1.272 rows=110
loops=1)"
"  Hash Cond: (m.cods = s.cods)"
"  ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=28) (actual
time=0.008..0.532 rows=10000 loops=1)"
"  ->  Hash  (cost=10.27..10.27 rows=99 width=4) (actual time=0.028..0.028 rows=99
loops=1)"
"        Buckets: 1024  Batches: 1  Memory Usage: 12kB"
"        ->  Index Only Scan using setor_pkey on setor s  (cost=0.29..10.27 rows=99 width=4)
(actual time=0.005..0.018 rows=99 loops=1)"
"              Index Cond: ((cods > 100) AND (cods < 200))"
"              Heap Fetches: 99"
"Planning Time: 0.114 ms"
"Execution Time: 1.287 ms"
```

2° Consulta com otimização usando índice (btree)

```
create index idx_mecanico_setor on mecanico using btree(cods);
```

```
explain analyse select m.cpf, m.nome, s.cods from mecanico m
join setor s using (cods)
where s.cods > 100 and s.cods < 200;
```

Resultado:

```
"Hash Join  (cost=11.50..254.76 rows=99 width=28) (actual time=0.067..1.380 rows=110
loops=1)"
"  Hash Cond: (m.cods = s.cods)"
"  ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=28) (actual
time=0.021..0.609 rows=10000 loops=1)"
"  ->  Hash  (cost=10.27..10.27 rows=99 width=4) (actual time=0.041..0.041 rows=99
loops=1)"
"        Buckets: 1024  Batches: 1  Memory Usage: 12kB"
"        ->  Index Only Scan using setor_pkey on setor s  (cost=0.29..10.27 rows=99 width=4)
(actual time=0.021..0.032 rows=99 loops=1)"
"              Index Cond: ((cods > 100) AND (cods < 200))"
"              Heap Fetches: 99"
"Planning Time: 0.128 ms"
"Execution Time: 1.395 ms"
```

3° Consulta com otimização usando índice (hash)

```
create index idx_mecanico_setor on mecanico using hash(cods);

explain analyse select m.cpf, m.nome, s.cods from mecanico m
join setor s using (cods)
where s.cods > 100 and s.cods < 200;
```

Resultado:

```
"Hash Join  (cost=11.50..254.76 rows=99 width=28) (actual time=0.038..1.173 rows=110
loops=1)"
"  Hash Cond: (m.cods = s.cods)"
"  ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=28) (actual
time=0.007..0.509 rows=10000 loops=1)"
"  ->  Hash  (cost=10.27..10.27 rows=99 width=4) (actual time=0.025..0.025 rows=99
loops=1)"
"        Buckets: 1024  Batches: 1  Memory Usage: 12kB"
"        ->  Index Only Scan using setor_pkey on setor s  (cost=0.29..10.27 rows=99 width=4)
(actual time=0.005..0.016 rows=99 loops=1)"
"              Index Cond: ((cods > 100) AND (cods < 200))"
"              Heap Fetches: 99"
"Planning Time: 0.115 ms"
"Execution Time: 1.189 ms"
```

Conclusão: em ambos os casos, a diferença é pequena. Caso a tabela seja escalável, seria mais interessante utilizar índice, pois, conforme
popular a tabela, a diferença vai ser maior

2) Recupere o CPF e nome dos mecânicos que atenderam no dia 13/06/2018.

1° Consulta sem otimização

```
explain analyse select m.cpf, m.nome from mecanico m
join conserto c using(codm)
where c.data = '13-06-2018';
```

Resultado:

```
"Nested Loop  (cost=0.29..368.87 rows=26 width=24) (actual time=0.017..0.624 rows=33
loops=1)"
"  ->  Seq Scan on conserto c  (cost=0.00..189.00 rows=26 width=4) (actual
time=0.011..0.548 rows=33 loops=1)"
"        Filter: (data = '2018-06-13'::date)"
"        Rows Removed by Filter: 9967"
"  ->  Index Scan using mecanico_pkey on mecanico m  (cost=0.29..6.92 rows=1 width=28)
(actual time=0.002..0.002 rows=1 loops=33)"
"        Index Cond: (codm = c.codm)"
"Planning Time: 0.137 ms"
"Execution Time: 0.636 ms"
```

2° Consulta com otimização usando índice (btree)

```
create index idx_conserto_data on conserto using btree(data);
explain analyse select m.cpf, m.nome from mecanico m
join conserto c using(codm)
where c.data = '13-06-2018';
```

Resultado:

```
"Nested Loop  (cost=4.77..233.98 rows=26 width=24) (actual time=0.035..0.102 rows=33
loops=1)"
"  ->  Bitmap Heap Scan on conserto c  (cost=4.49..54.12 rows=26 width=4) (actual
time=0.030..0.044 rows=33 loops=1)"
"        Recheck Cond: (data = '2018-06-13'::date)"
"        Heap Blocks: exact=27"
"        ->  Bitmap Index Scan on idx_conserto_data  (cost=0.00..4.48 rows=26 width=0)
(actual time=0.026..0.026 rows=33 loops=1)"
"              Index Cond: (data = '2018-06-13'::date)"
"  ->  Index Scan using mecanico_pkey on mecanico m  (cost=0.29..6.92 rows=1 width=28)
(actual time=0.001..0.001 rows=1 loops=33)"
"        Index Cond: (codm = c.codm)"
"Planning Time: 1.161 ms"
"Execution Time: 0.125 ms"
```

3° Consulta com otimização usando índice (hash)

```
create index idx_conserto_data on conserto using hash(data)
explain analyse select m.cpf, m.nome from mecanico m
join conserto c using(codm)
where c.data = '13-06-2018';
```

Resultado

```
"Nested Loop  (cost=4.49..233.70 rows=26 width=24) (actual time=0.020..0.081 rows=33
loops=1)"
"  ->  Bitmap Heap Scan on conserto c  (cost=4.20..53.83 rows=26 width=4) (actual
time=0.016..0.030 rows=33 loops=1)"
"        Recheck Cond: (data = '2018-06-13'::date)"
"        Heap Blocks: exact=27"
"        ->  Bitmap Index Scan on idx_conserto_data  (cost=0.00..4.20 rows=26 width=0)
(actual time=0.011..0.011 rows=33 loops=1)"
"              Index Cond: (data = '2018-06-13'::date)"
"  ->  Index Scan using mecanico_pkey on mecanico m  (cost=0.29..6.92 rows=1 width=28)
(actual time=0.001..0.001 rows=1 loops=33)"
"        Index Cond: (codm = c.codm)"
"Planning Time: 0.857 ms"
"Execution Time: 0.100 ms"
```

Conclusão: em ambos os casos, a diferença é pequena. Caso a tabela seja escalável, seria mais interessante utilizar índice, pois, conforme
popular a tabela, a diferença vai ser maior

3) Recupere o nome do mecânico, o nome do cliente e a hora do conserto para os consertos realizados de 12/06/2018 à 25/09/2018.

1° Consulta sem otimização

```
explain analyse select m.nome, cli.nome, con.hora, con.data from conserto con
join mecanico m using(codm)
join veiculo v using(codv)
join cliente cli using(codc)
where con.data between '12-06-2018' and '25-09-2018';
```

Resultado:

```
"Hash Join  (cost=923.76..1206.47 rows=3224 width=35) (actual time=5.705..8.082
rows=3216 loops=1)"
"  Hash Cond: (con.codm = m.codm)"
"  ->  Hash Join  (cost=581.76..856.00 rows=3224 width=27) (actual time=3.738..5.508
rows=3216 loops=1)"
"        Hash Cond: (cli.codc = v.codc)"
"        ->  Seq Scan on cliente cli  (cost=0.00..192.00 rows=10000 width=15) (actual
time=0.007..0.580 rows=10000 loops=1)"
"        ->  Hash  (cost=541.46..541.46 rows=3224 width=20) (actual time=3.720..3.720
rows=3216 loops=1)"
"              Buckets: 4096  Batches: 1  Memory Usage: 196kB"
"              ->  Hash Join  (cost=319.00..541.46 rows=3224 width=20) (actual
time=1.686..3.233 rows=3216 loops=1)"
"                    Hash Cond: (con.codv = v.codv)"
"                    ->  Seq Scan on conserto con  (cost=0.00..214.00 rows=3224 width=20)
(actual time=0.006..0.920 rows=3216 loops=1)"
"                          Filter: ((data >= '2018-06-12'::date) AND (data <= '2018-09-25'::date))"
"                          Rows Removed by Filter: 6784"
"                    ->  Hash  (cost=194.00..194.00 rows=10000 width=8) (actual
time=1.647..1.647 rows=10000 loops=1)"
"                          Buckets: 16384  Batches: 1  Memory Usage: 519kB"
"                          ->  Seq Scan on veiculo v  (cost=0.00..194.00 rows=10000 width=8) (actual
time=0.006..0.863 rows=10000 loops=1)"
"  ->  Hash  (cost=217.00..217.00 rows=10000 width=16) (actual time=1.938..1.938
rows=10000 loops=1)"
"        Buckets: 16384  Batches: 1  Memory Usage: 632kB"
"        ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=16) (actual
time=0.008..0.940 rows=10000 loops=1)"
"Planning Time: 0.296 ms"
"Execution Time: 8.292 ms"
```

2° Consulta com otimização usando índice (btree)

```
"Hash Join  (cost=869.31..1151.35 rows=3171 width=35) (actual time=5.451..7.531
rows=3161 loops=1)"
"  Hash Cond: (con.codm = m.codm)"
"  ->  Hash Join  (cost=527.31..801.02 rows=3171 width=27) (actual time=3.328..4.833
rows=3161 loops=1)"
"        Hash Cond: (cli.codc = v.codc)"
"        ->  Seq Scan on cliente cli  (cost=0.00..192.00 rows=10000 width=15) (actual
time=0.012..0.426 rows=10000 loops=1)"
"        ->  Hash  (cost=487.68..487.68 rows=3171 width=20) (actual time=3.297..3.299
rows=3161 loops=1)"
"              Buckets: 4096  Batches: 1  Memory Usage: 193kB"
"              ->  Hash Join  (cost=367.79..487.68 rows=3171 width=20) (actual
time=2.041..2.917 rows=3161 loops=1)"
"                    Hash Cond: (con.codv = v.codv)"
"                    ->  Bitmap Heap Scan on conserto con  (cost=48.79..160.35 rows=3171
width=20) (actual time=0.075..0.367 rows=3161 loops=1)"
"                          Recheck Cond: ((data >= '2018-06-12'::date) AND (data <=
'2018-09-25'::date))"
"                          Heap Blocks: exact=64"
"                          ->  Bitmap Index Scan on idx_conserto_data  (cost=0.00..47.99 rows=3171
width=0) (actual time=0.066..0.066 rows=3161 loops=1)"
"                                Index Cond: ((data >= '2018-06-12'::date) AND (data <=
'2018-09-25'::date))"
"                    ->  Hash  (cost=194.00..194.00 rows=10000 width=8) (actual
time=1.917..1.917 rows=10000 loops=1)"
"                          Buckets: 16384  Batches: 1  Memory Usage: 519kB"
"                          ->  Seq Scan on veiculo v  (cost=0.00..194.00 rows=10000 width=8) (actual
time=0.007..0.964 rows=10000 loops=1)"
"  ->  Hash  (cost=217.00..217.00 rows=10000 width=16) (actual time=2.067..2.068
rows=10000 loops=1)"
"        Buckets: 16384  Batches: 1  Memory Usage: 632kB"
"        ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=16) (actual
time=0.008..0.932 rows=10000 loops=1)"
"Planning Time: 0.338 ms"
"Execution Time: 7.951 ms"
```

3° Consulta com otimização usando índice (hash)

```
"Hash Join  (cost=922.96..1205.00 rows=3171 width=35) (actual time=6.042..8.467
rows=3161 loops=1)"
"  Hash Cond: (con.codm = m.codm)"
"  ->  Hash Join  (cost=580.96..854.67 rows=3171 width=27) (actual time=3.905..5.591
rows=3161 loops=1)"
"        Hash Cond: (cli.codc = v.codc)"
"        ->  Seq Scan on cliente cli  (cost=0.00..192.00 rows=10000 width=15) (actual
time=0.013..0.463 rows=10000 loops=1)"
"        ->  Hash  (cost=541.32..541.32 rows=3171 width=20) (actual time=3.872..3.876
rows=3161 loops=1)"
"              Buckets: 4096  Batches: 1  Memory Usage: 193kB"
"              ->  Hash Join  (cost=319.00..541.32 rows=3171 width=20) (actual
time=1.883..3.447 rows=3161 loops=1)"
"                    Hash Cond: (con.codv = v.codv)"
"                    ->  Seq Scan on conserto con  (cost=0.00..214.00 rows=3171 width=20)
(actual time=0.008..0.777 rows=3161 loops=1)"
"                          Filter: ((data >= '2018-06-12'::date) AND (data <= '2018-09-25'::date))"
"                          Rows Removed by Filter: 6839"
"                    ->  Hash  (cost=194.00..194.00 rows=10000 width=8) (actual
time=1.827..1.828 rows=10000 loops=1)"
"                          Buckets: 16384  Batches: 1  Memory Usage: 519kB"
"                          ->  Seq Scan on veiculo v  (cost=0.00..194.00 rows=10000 width=8) (actual
time=0.008..0.907 rows=10000 loops=1)"
"  ->  Hash  (cost=217.00..217.00 rows=10000 width=16) (actual time=2.080..2.081
rows=10000 loops=1)"
"        Buckets: 16384  Batches: 1  Memory Usage: 632kB"
"        ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=16) (actual
time=0.010..0.912 rows=10000 loops=1)"
"Planning Time: 0.341 ms"
"Execution Time: 8.769 ms"
```

Conclusão: em ambos os casos, a diferença é pequena. Caso a tabela seja escalável, seria mais interessante utilizar índice, pois, conforme
popular a tabela, a diferença vai ser maior

4) Recupere o nome e a função de todos os mecânicos, e o número e o nome dos setores para os mecânicos que tenham essa informação.

1° Consulta sem otimização

```
select m.nome, m.funcao, s.cods, s.nome from mecanico m
join setor s using(cods);
```

Resultado:

```
"Hash Join  (cost=280.00..523.26 rows=10000 width=35) (actual time=1.336..4.110
rows=10000 loops=1)"
"  Hash Cond: (m.cods = s.cods)"
"  ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=26) (actual
time=0.068..0.531 rows=10000 loops=1)"
"  ->  Hash  (cost=155.00..155.00 rows=10000 width=13) (actual time=1.214..1.215
rows=10000 loops=1)"
"        Buckets: 16384  Batches: 1  Memory Usage: 577kB"
"        ->  Seq Scan on setor s  (cost=0.00..155.00 rows=10000 width=13) (actual
time=0.006..0.400 rows=10000 loops=1)"
"Planning Time: 0.137 ms"
"Execution Time: 4.401 ms"
```

2° Consulta com otimização usando índice (btree)

```
create index idx_mecanico on mecanico using btree(nome);
create index idx_setor on setor using btree(cods);

explain analyse select m.nome, m.funcao, s.cods, s.nome from mecanico m
join setor s using(cods);
```

Resultado:

```
"Hash Join  (cost=280.00..523.26 rows=10000 width=35) (actual time=1.282..3.942
rows=10000 loops=1)"
"  Hash Cond: (m.cods = s.cods)"
"  ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=26) (actual
time=0.008..0.468 rows=10000 loops=1)"
"  ->  Hash  (cost=155.00..155.00 rows=10000 width=13) (actual time=1.221..1.222
rows=10000 loops=1)"
"        Buckets: 16384  Batches: 1  Memory Usage: 577kB"
"        ->  Seq Scan on setor s  (cost=0.00..155.00 rows=10000 width=13) (actual
time=0.006..0.400 rows=10000 loops=1)"
"Planning Time: 0.150 ms"
"Execution Time: 4.236 ms"
```

3° Consulta com otimização usando índice (hash)

```
create index idx_mecanico on mecanico using hash(nome);
create index idx_setor on setor using hash(cods);

explain analyse select m.nome, m.funcao, s.cods, s.nome from mecanico m
join setor s using(cods);
```

Resultados:

```
"Hash Join  (cost=280.00..523.26 rows=10000 width=35) (actual time=1.193..3.629
rows=10000 loops=1)"
"  Hash Cond: (m.cods = s.cods)"
"  ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=26) (actual
time=0.008..0.425 rows=10000 loops=1)"
"  ->  Hash  (cost=155.00..155.00 rows=10000 width=13) (actual time=1.135..1.135
rows=10000 loops=1)"
"        Buckets: 16384  Batches: 1  Memory Usage: 577kB"
"        ->  Seq Scan on setor s  (cost=0.00..155.00 rows=10000 width=13) (actual
time=0.005..0.389 rows=10000 loops=1)"
"Planning Time: 0.142 ms"
"Execution Time: 3.904 ms"
```

Conclusão: em ambos os casos, a diferença é pequena. Caso a tabela seja escalável, seria mais interessante utilizar índice, pois, conforme
popular a tabela, a diferença vai ser maior

5) Recupere o nome de todos os mecânicos, e as datas dos consertos para os mecânicos que têm consertos feitos (deve aparecer apenas um registro de nome de mecânico para cada data de conserto).

1° Consulta sem otimização

```
explain analyse select m.nome, c.data from mecanico m
join conserto c
on m.codm = c.codm;
```

Resultados:

```
"Hash Join  (cost=342.00..532.26 rows=10000 width=16) (actual time=1.967..3.980
rows=10000 loops=1)"
"  Hash Cond: (c.codm = m.codm)"
"  ->  Seq Scan on conserto c  (cost=0.00..164.00 rows=10000 width=8) (actual
time=0.011..0.399 rows=10000 loops=1)"
"  ->  Hash  (cost=217.00..217.00 rows=10000 width=16) (actual time=1.895..1.896
rows=10000 loops=1)"
"        Buckets: 16384  Batches: 1  Memory Usage: 632kB"
"        ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=16) (actual
time=0.008..0.845 rows=10000 loops=1)"
"Planning Time: 0.182 ms"
"Execution Time: 4.288 ms"
```

2° Consulta com otimização usando índice (btree)

```
create index idx_conserto_data on conserto using btree(data)

explain analyse select m.nome, c.data from mecanico m
join conserto c
on m.codm = c.codm;
```

Resultados:

```
"Hash Join  (cost=342.00..532.26 rows=10000 width=16) (actual time=1.880..3.885
rows=10000 loops=1)"
"  Hash Cond: (c.codm = m.codm)"
"  ->  Seq Scan on conserto c  (cost=0.00..164.00 rows=10000 width=8) (actual
time=0.010..0.392 rows=10000 loops=1)"
"  ->  Hash  (cost=217.00..217.00 rows=10000 width=16) (actual time=1.814..1.815
rows=10000 loops=1)"
"        Buckets: 16384  Batches: 1  Memory Usage: 632kB"
"        ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=16) (actual
time=0.006..0.802 rows=10000 loops=1)"
"Planning Time: 0.219 ms"
"Execution Time: 4.155 ms"
```

3° Consulta com otimização usando índice (hash)

```
create index idx_conserto_data on conserto using hash(data)
```

```
explain analyse select m.nome, c.data from mecanico m
join conserto c
on m.codm = c.codm;
```

Resultados:

```
"Hash Join  (cost=342.00..532.26 rows=10000 width=16) (actual time=1.845..3.782
rows=10000 loops=1)"
"  Hash Cond: (c.codm = m.codm)"
"  ->  Seq Scan on conserto c  (cost=0.00..164.00 rows=10000 width=8) (actual
time=0.009..0.389 rows=10000 loops=1)"
"  ->  Hash  (cost=217.00..217.00 rows=10000 width=16) (actual time=1.785..1.785
rows=10000 loops=1)"
"        Buckets: 16384  Batches: 1  Memory Usage: 632kB"
"        ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=16) (actual
time=0.006..0.807 rows=10000 loops=1)"
"Planning Time: 0.168 ms"
"Execution Time: 4.057 ms"
```

Conclusão: em ambos os casos, a diferença é pequena. Caso a tabela seja escalável, seria mais interessante utilizar índice, pois, conforme
popular a tabela, a diferença vai ser maior

6) Recupere a média da quilometragem de todos os veículos dos clientes.

1° Consulta sem otimização

```
explain analyse select avg(quilometragem) from veiculo;
```

Resultado:

```
"Aggregate  (cost=219.00..219.01 rows=1 width=8) (actual time=1.867..1.868 rows=1 loops=1)"
"  ->  Seq Scan on veiculo  (cost=0.00..194.00 rows=10000 width=8) (actual time=0.010..0.513 rows=10000 loops=1)"
"Planning Time: 0.048 ms"
"Execution Time: 1.890 ms"
```

2° Consulta com otimização usando índice (btree)

```
create index idx_quilometragem on veiculo using btree(quilometragem);
explain analyse select avg(quilometragem) from veiculo;
```

Resultado:

```
"Aggregate  (cost=219.00..219.01 rows=1 width=8) (actual time=1.262..1.262 rows=1 loops=1)"
"  ->  Seq Scan on veiculo  (cost=0.00..194.00 rows=10000 width=8) (actual time=0.011..0.405 rows=10000 loops=1)"
"Planning Time: 0.078 ms"
"Execution Time: 1.281 ms"
```

3° Consulta com otimização usando índice (hash)

```
create index idx_quilometragem on veiculo using hash(quilometragem);
explain analyse select avg(quilometragem) from veiculo;
```

Resultado:

```
"Aggregate  (cost=219.00..219.01 rows=1 width=8) (actual time=1.258..1.259 rows=1 loops=1)"
"  ->  Seq Scan on veiculo  (cost=0.00..194.00 rows=10000 width=8) (actual time=0.011..0.431 rows=10000 loops=1)"
"Planning Time: 0.071 ms"
"Execution Time: 1.277 ms"
```

Conclusão: em ambos os casos, a diferença é pequena. Caso a tabela seja escalável, seria mais interessante utilizar índice, pois, conforme
popular a tabela, a diferença vai ser maior

7) Recupere a soma da quilometragem dos veículos de cada cidade onde residem seus proprietários.

1° Consulta sem otimização

```
select sum(v.quilometragem), cli.cidade from veiculo v
join cliente cli using(codc)
group by cli.cidade;
```

Resultado:

```
"HashAggregate  (cost=587.26..587.32 rows=6 width=20) (actual time=6.012..6.014 rows=6
loops=1)"
"  Group Key: cli.cidade"
"  Batches: 1  Memory Usage: 24kB"
"  ->  Hash Join  (cost=317.00..537.26 rows=10000 width=20) (actual time=2.045..4.637
rows=10000 loops=1)"
"        Hash Cond: (v.codc = cli.codc)"
"        ->  Seq Scan on veiculo v  (cost=0.00..194.00 rows=10000 width=12) (actual
time=0.008..0.455 rows=10000 loops=1)"
"        ->  Hash  (cost=192.00..192.00 rows=10000 width=16) (actual time=1.980..1.980
rows=10000 loops=1)"
"              Buckets: 16384  Batches: 1  Memory Usage: 608kB"
"              ->  Seq Scan on cliente cli  (cost=0.00..192.00 rows=10000 width=16) (actual
time=0.006..0.889 rows=10000 loops=1)"
"Planning Time: 0.132 ms"
"Execution Time: 6.133 ms"
```

2° Consulta com otimização usando índice (btree)

```
create index idx_cliente_cidade on cliente using btree(cidade);
create index idx_veiculo_codc on veiculo using btree(codc);

explain analyse select sum(v.quilometragem), cli.cidade from veiculo v
join cliente cli using(codc)
group by cli.cidade;
```

Resultado:

```
"HashAggregate  (cost=587.26..587.32 rows=6 width=20) (actual time=6.272..6.274 rows=6
loops=1)"
"  Group Key: cli.cidade"
"  Batches: 1  Memory Usage: 24kB"
"  ->  Hash Join  (cost=317.00..537.26 rows=10000 width=20) (actual time=2.478..4.868
rows=10000 loops=1)"
"        Hash Cond: (v.codc = cli.codc)"
"        ->  Seq Scan on veiculo v  (cost=0.00..194.00 rows=10000 width=12) (actual
time=0.011..0.440 rows=10000 loops=1)"
```

```
"        ->  Hash  (cost=192.00..192.00 rows=10000 width=16) (actual time=2.384..2.385
rows=10000 loops=1)"
"              Buckets: 16384  Batches: 1  Memory Usage: 608kB"
"              ->  Seq Scan on cliente cli  (cost=0.00..192.00 rows=10000 width=16) (actual
time=0.010..1.071 rows=10000 loops=1)"
"Planning Time: 0.185 ms"
"Execution Time: 6.484 ms"
```

3° Consulta com otimização usando índice (hash)

```
create index idx_cliente_cidade on cliente using hash(cidade);
create index idx_veiculo_codc on veiculo using hash(codc);

explain analyse select sum(v.quilometragem), cli.cidade from veiculo v
join cliente cli using(codc)
group by cli.cidade;
```

Resultado:

```
"HashAggregate  (cost=587.26..587.32 rows=6 width=20) (actual time=5.275..5.276 rows=6
loops=1)"
"  Group Key: cli.cidade"
"  Batches: 1  Memory Usage: 24kB"
"  ->  Hash Join  (cost=317.00..537.26 rows=10000 width=20) (actual time=1.937..4.002
rows=10000 loops=1)"
"        Hash Cond: (v.codc = cli.codc)"
"        ->  Seq Scan on veiculo v  (cost=0.00..194.00 rows=10000 width=12) (actual
time=0.008..0.404 rows=10000 loops=1)"
"        ->  Hash  (cost=192.00..192.00 rows=10000 width=16) (actual time=1.874..1.874
rows=10000 loops=1)"
"              Buckets: 16384  Batches: 1  Memory Usage: 608kB"
"              ->  Seq Scan on cliente cli  (cost=0.00..192.00 rows=10000 width=16) (actual
time=0.005..0.835 rows=10000 loops=1)"
"Planning Time: 0.150 ms"
"Execution Time: 5.381 ms"
```

Conclusão: em ambos os casos, a diferença é pequena. Caso a tabela seja escalável, seria mais interessante utilizar índice, pois, conforme
popular a tabela, a diferença vai ser maior

8) Recupere a quantidade de consertos feitos por cada mecânico durante o período de 12/06/2018 até 19/010/2018

1° Consulta sem otimização

```
explain analyse select m.nome, count(1) from conserto c
join mecanico m using(codm)
where c.data between '12-06-2018' and '19-10-2018'
group by m.nome;
```

Resultado:

```
"HashAggregate  (cost=585.67..624.58 rows=3891 width=20) (actual time=4.106..4.408
rows=3225 loops=1)"
"  Group Key: m.nome"
"  Batches: 1  Memory Usage: 465kB"
"  ->  Hash Join  (cost=342.00..566.21 rows=3891 width=12) (actual time=1.782..3.320
rows=3850 loops=1)"
"        Hash Cond: (c.codm = m.codm)"
"        ->  Seq Scan on conserto c  (cost=0.00..214.00 rows=3891 width=4) (actual
time=0.009..0.776 rows=3850 loops=1)"
"              Filter: ((data >= '2018-06-12'::date) AND (data <= '2018-10-19'::date))"
"              Rows Removed by Filter: 6150"
"        ->  Hash  (cost=217.00..217.00 rows=10000 width=16) (actual time=1.727..1.728
rows=10000 loops=1)"
"              Buckets: 16384  Batches: 1  Memory Usage: 632kB"
"              ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=16) (actual
time=0.005..0.790 rows=10000 loops=1)"
"Planning Time: 0.152 ms"
"Execution Time: 4.666 ms"
```

2° Consulta com otimização usando índice (btree)

```
create index idx_conserto_data on conserto using btree(data);

explain analyse select m.nome, count(1) from conserto c
join mecanico m using(codm)
where c.data between '12-06-2018' and '19-10-2018'
group by m.nome;
```

Resultado:

```
"HashAggregate  (cost=554.20..593.11 rows=3891 width=20) (actual time=3.642..3.914
rows=3225 loops=1)"
"  Group Key: m.nome"
"  Batches: 1  Memory Usage: 465kB"
"  ->  Hash Join  (cost=402.17..534.75 rows=3891 width=12) (actual time=1.895..2.877
rows=3850 loops=1)"
"        Hash Cond: (c.codm = m.codm)"
"        ->  Bitmap Heap Scan on conserto c  (cost=60.17..182.53 rows=3891 width=4) (actual
time=0.080..0.413 rows=3850 loops=1)"
"              Recheck Cond: ((data >= '2018-06-12'::date) AND (data <= '2018-10-19'::date))"
"              Heap Blocks: exact=64"
"              ->  Bitmap Index Scan on idx_conserto_data  (cost=0.00..59.20 rows=3891
width=0) (actual time=0.071..0.071 rows=3850 loops=1)"
"                    Index Cond: ((data >= '2018-06-12'::date) AND (data <= '2018-10-19'::date))"
"        ->  Hash  (cost=217.00..217.00 rows=10000 width=16) (actual time=1.761..1.762
rows=10000 loops=1)"
"              Buckets: 16384  Batches: 1  Memory Usage: 632kB"
"              ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=16) (actual
time=0.008..0.772 rows=10000 loops=1)"
"Planning Time: 0.174 ms"
"Execution Time: 4.204 ms"
```

3° Consulta com otimização usando índice (hash)

```
create index idx_conserto_data on conserto using hash(data);

explain analyse select m.nome, count(1) from conserto c
join mecanico m using(codm)
where c.data between '12-06-2018' and '19-10-2018'
group by m.nome;
```

Resultado:

```
"HashAggregate  (cost=585.67..624.58 rows=3891 width=20) (actual time=4.534..4.893
rows=3225 loops=1)"
"  Group Key: m.nome"
"  Batches: 1  Memory Usage: 465kB"
"  ->  Hash Join  (cost=342.00..566.21 rows=3891 width=12) (actual time=2.054..3.649
rows=3850 loops=1)"
"        Hash Cond: (c.codm = m.codm)"
"        ->  Seq Scan on conserto c  (cost=0.00..214.00 rows=3891 width=4) (actual
time=0.010..0.824 rows=3850 loops=1)"
"              Filter: ((data >= '2018-06-12'::date) AND (data <= '2018-10-19'::date))"
"              Rows Removed by Filter: 6150"
"        ->  Hash  (cost=217.00..217.00 rows=10000 width=16) (actual time=2.035..2.036
rows=10000 loops=1)"
"              Buckets: 16384  Batches: 1  Memory Usage: 632kB"
"              ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=16) (actual
time=0.005..0.912 rows=10000 loops=1)"
"Planning Time: 0.168 ms"
"Execution Time: 5.197 ms"
```

Conclusão: em ambos os casos, a diferença é pequena. Caso a tabela seja escalável, seria mais interessante utilizar índice, pois, conforme
popular a tabela, a diferença vai ser maior

9) Recupere a quantidade de consertos feitos agrupada pela marca do veículo.

1° Consulta sem otimização

```
explain analyse select v.marca, sum(1) from conserto c
join veiculo v using(codv)
group by v.marca;
```

Resultado:

```
"HashAggregate  (cost=559.26..559.30 rows=4 width=11) (actual time=5.434..5.437 rows=4
loops=1)"
"  Group Key: v.marca"
"  Batches: 1  Memory Usage: 24kB"
"  ->  Hash Join  (cost=319.00..509.26 rows=10000 width=3) (actual time=1.946..4.162
rows=10000 loops=1)"
"        Hash Cond: (c.codv = v.codv)"
"        ->  Seq Scan on conserto c  (cost=0.00..164.00 rows=10000 width=4) (actual
time=0.008..0.455 rows=10000 loops=1)"
"        ->  Hash  (cost=194.00..194.00 rows=10000 width=7) (actual time=1.879..1.879
rows=10000 loops=1)"
"              Buckets: 16384  Batches: 1  Memory Usage: 538kB"
"              ->  Seq Scan on veiculo v  (cost=0.00..194.00 rows=10000 width=7) (actual
time=0.006..0.851 rows=10000 loops=1)"
"Planning Time: 0.226 ms"
"Execution Time: 5.524 ms"
```

2° Consulta com otimização usando índice (btree)

```
create index idx_veiculo_marca on veiculo using btree(marca);
create index idx_conserto_codv on conserto using btree(codv);

explain analyse select v.marca, sum(1) from conserto c
join veiculo v using(codv)
group by v.marca;
```

Resultado:

```
"HashAggregate  (cost=559.26..559.30 rows=4 width=11) (actual time=5.232..5.234 rows=4
loops=1)"
"  Group Key: v.marca"
"  Batches: 1  Memory Usage: 24kB"
"  ->  Hash Join  (cost=319.00..509.26 rows=10000 width=3) (actual time=1.930..3.977
rows=10000 loops=1)"
"        Hash Cond: (c.codv = v.codv)"
"        ->  Seq Scan on conserto c  (cost=0.00..164.00 rows=10000 width=4) (actual
time=0.008..0.433 rows=10000 loops=1)"
"        ->  Hash  (cost=194.00..194.00 rows=10000 width=7) (actual time=1.861..1.862
rows=10000 loops=1)"
"              Buckets: 16384  Batches: 1  Memory Usage: 538kB"
"             ->  Seq Scan on veiculo v  (cost=0.00..194.00 rows=10000 width=7) (actual time=0.006..0.827 rows=10000 loops=1)"
"Planning Time: 0.184 ms"
"Execution Time: 5.310 ms"
```

3° Consulta com otimização usando índice (hash)

```
create index idx_veiculo_marca on veiculo using hash(marca);
create index idx_conserto_codv on conserto using hash(codv);

explain analyse select v.marca, sum(1) from conserto c
join veiculo v using(codv)
group by v.marca;
```

Resultado:

```
"HashAggregate  (cost=559.26..559.30 rows=4 width=11) (actual time=5.466..5.468 rows=4 loops=1)"
"  Group Key: v.marca"
"  Batches: 1  Memory Usage: 24kB"
"  ->  Hash Join  (cost=319.00..509.26 rows=10000 width=3) (actual time=2.288..4.271 rows=10000 loops=1)"
"        Hash Cond: (c.codv = v.codv)"
"        ->  Seq Scan on conserto c  (cost=0.00..164.00 rows=10000 width=4) (actual time=0.008..0.428 rows=10000 loops=1)"
"        ->  Hash  (cost=194.00..194.00 rows=10000 width=7) (actual time=2.220..2.220 rows=10000 loops=1)"
"              Buckets: 16384  Batches: 1  Memory Usage: 538kB"
"              ->  Seq Scan on veiculo v  (cost=0.00..194.00 rows=10000 width=7) (actual time=0.005..1.024 rows=10000 loops=1)"
"Planning Time: 0.163 ms"
"Execution Time: 5.546 ms"
```

Conclusão: em ambos os casos, a diferença é pequena. Caso a tabela seja escalável, seria mais interessante utilizar índice, pois, conforme
popular a tabela, a diferença vai ser maior

10) Recupere o modelo, a marca e o ano dos veículos que têm quilometragem maior que a média de quilometragem de todos os veículos.

1° Consulta sem otimização

```
explain analyse select v.modelo, v.ano from veiculo v where quilometragem >
(select avg(quilometragem) from veiculo);
```

Resultado:

```
"Seq Scan on veiculo v  (cost=219.01..438.01 rows=3333 width=9) (actual
time=1.163..2.082 rows=4985 loops=1)"
"  Filter: (quilometragem > $0)"
"  Rows Removed by Filter: 5015"
"  InitPlan 1 (returns $0)"
"    ->  Aggregate  (cost=219.00..219.01 rows=1 width=8) (actual time=1.152..1.152 rows=1
loops=1)"
"          ->  Seq Scan on veiculo  (cost=0.00..194.00 rows=10000 width=8) (actual
time=0.004..0.399 rows=10000 loops=1)"
"Planning Time: 0.087 ms"
"Execution Time: 2.182 ms"
```

2° Consulta com otimização usando índice (btree)

```
create index idx_veiculo_quilometragem on veiculo using btree(quilometragem);

explain analyse select v.modelo, v.ano from veiculo v where quilometragem >
(select avg(quilometragem) from veiculo);
```

Resultado:

```
"Bitmap Heap Scan on veiculo v  (cost=285.13..420.79 rows=3333 width=9) (actual
time=1.362..1.840 rows=4985 loops=1)"
"  Recheck Cond: (quilometragem > $0)"
"  Heap Blocks: exact=94"
"  InitPlan 1 (returns $0)"
"    ->  Aggregate  (cost=219.00..219.01 rows=1 width=8) (actual time=1.175..1.175 rows=1
loops=1)"
"          ->  Seq Scan on veiculo  (cost=0.00..194.00 rows=10000 width=8) (actual
time=0.007..0.407 rows=10000 loops=1)"
"  ->  Bitmap Index Scan on idx_veiculo_quilometragem  (cost=0.00..65.28 rows=3333
width=0) (actual time=1.352..1.352 rows=4985 loops=1)"
"        Index Cond: (quilometragem > $0)"
"Planning Time: 0.114 ms"
"Execution Time: 1.950 ms"
```

3° Consulta com otimização usando índice (hash)

```
create index idx_veiculo_quilometragem on veiculo using hash(quilometragem);
```

```
explain analyse select v.modelo, v.ano from veiculo v where quilometragem >
(select avg(quilometragem) from veiculo);
```

Resultado:

```
"Seq Scan on veiculo v  (cost=219.01..438.01 rows=3333 width=9) (actual
time=1.526..2.467 rows=4985 loops=1)"
"  Filter: (quilometragem > $0)"
"  Rows Removed by Filter: 5015"
"  InitPlan 1 (returns $0)"
"    ->  Aggregate  (cost=219.00..219.01 rows=1 width=8) (actual time=1.514..1.514 rows=1
loops=1)"
"          ->  Seq Scan on veiculo  (cost=0.00..194.00 rows=10000 width=8) (actual
time=0.004..0.487 rows=10000 loops=1)"
"Planning Time: 0.099 ms"
"Execution Time: 2.577 ms"
```

11) Recupere o nome dos mecânicos que têm mais de um conserto marcado para o mesmo dia.

1° Consulta sem otimização

```
explain analyse select m.nome, count(codm)
from mecanico m
join conserto c using (codm)
group by m.nome, c.data
having count(codm) > 1;
```

Resultado:

```
"HashAggregate  (cost=607.26..732.26 rows=3333 width=24) (actual time=7.555..8.331
rows=17 loops=1)"
"  Group Key: m.nome, c.data"
"  Filter: (count(m.codm) > 1)"
"  Batches: 1  Memory Usage: 1681kB"
"  Rows Removed by Filter: 9966"
"  ->  Hash Join  (cost=342.00..532.26 rows=10000 width=20) (actual time=1.961..4.796
rows=10000 loops=1)"
"        Hash Cond: (c.codm = m.codm)"
"        ->  Seq Scan on conserto c  (cost=0.00..164.00 rows=10000 width=8) (actual
time=0.008..0.496 rows=10000 loops=1)"
"        ->  Hash  (cost=217.00..217.00 rows=10000 width=16) (actual time=1.903..1.903
rows=10000 loops=1)"
"              Buckets: 16384  Batches: 1  Memory Usage: 632kB"
"              ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=16) (actual
time=0.006..0.862 rows=10000 loops=1)"
"Planning Time: 0.171 ms"
"Execution Time: 8.733 ms"
```

2° Consulta com otimização usando índice (btree)

```
create index idx_conserto_data on conserto using btree(data);

explain analyse select m.nome, count(codm)
from mecanico m
join conserto c using (codm)
group by m.nome, c.data
having count(codm) > 1;
```

Resultado:

```
"HashAggregate  (cost=607.26..732.26 rows=3333 width=24) (actual time=7.761..8.669
rows=17 loops=1)"
"  Group Key: m.nome, c.data"
"  Filter: (count(m.codm) > 1)"
"  Batches: 1  Memory Usage: 1681kB"
"  Rows Removed by Filter: 9966"
"  ->  Hash Join  (cost=342.00..532.26 rows=10000 width=20) (actual time=2.001..4.936
rows=10000 loops=1)"
"        Hash Cond: (c.codm = m.codm)"
"        ->  Seq Scan on conserto c  (cost=0.00..164.00 rows=10000 width=8) (actual
time=0.079..0.587 rows=10000 loops=1)"
"        ->  Hash  (cost=217.00..217.00 rows=10000 width=16) (actual time=1.909..1.910
rows=10000 loops=1)"
"              Buckets: 16384  Batches: 1  Memory Usage: 632kB"
"              ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=16) (actual
time=0.006..0.901 rows=10000 loops=1)"
"Planning Time: 0.169 ms"
"Execution Time: 9.047 ms"
```

3° Consulta com otimização usando índice (hash)

```
create index idx_conserto_data on conserto using hash(data);

explain analyse select m.nome, count(codm)
from mecanico m
join conserto c using (codm)
group by m.nome, c.data
having count(codm) > 1;
```

Resultado:

```
"HashAggregate  (cost=607.26..732.26 rows=3333 width=24) (actual time=7.529..8.491
rows=17 loops=1)"
"  Group Key: m.nome, c.data"
"  Filter: (count(m.codm) > 1)"
"  Batches: 1  Memory Usage: 1681kB"
"  Rows Removed by Filter: 9966"
"  ->  Hash Join  (cost=342.00..532.26 rows=10000 width=20) (actual time=2.271..4.925
rows=10000 loops=1)"
"        Hash Cond: (c.codm = m.codm)"
"        ->  Seq Scan on conserto c  (cost=0.00..164.00 rows=10000 width=8) (actual
time=0.008..0.464 rows=10000 loops=1)"
"        ->  Hash  (cost=217.00..217.00 rows=10000 width=16) (actual time=2.253..2.253
rows=10000 loops=1)"
"              Buckets: 16384  Batches: 1  Memory Usage: 632kB"
"              ->  Seq Scan on mecanico m  (cost=0.00..217.00 rows=10000 width=16) (actual
time=0.006..1.039 rows=10000 loops=1)"
"Planning Time: 0.165 ms"
"Execution Time: 8.876 ms"
```