

Gerenciamento de Banco de Dados

Arthur do Prado Labaki – 11821BCC017

Atividade 4 - Otimização de consultas

a) Execute o script de criação de dados para a atividade de otimização de consultas no Postgres

The screenshot displays the PostgreSQL Query Editor interface. On the left, the 'Catalogs' pane shows the database structure, including the 'novovendas' schema and its tables: 'cliente', 'itemped', 'pedido', 'produto', 'vendedor', 'vnomef', 'vnomem', and 'vsobrenome'. The main 'Query Editor' pane contains the following SQL script:

```
1 CREATE SCHEMA novovendas;
2 SET search_path TO novovendas;
3
4 create sequence codcli_seq start with 1;
5
6 CREATE TABLE CLIENTE (
7     CODCLI INTEGER NOT NULL,
8     NOME VARCHAR(50) NOT NULL,
9     SEXO CHAR(1),
10    ENDERECO VARCHAR(60),
11    CIDADE CHAR (20) NOT NULL,
12    CEP CHAR(10),
13    UF CHAR (2) NOT NULL,
14    CGC CHAR(14) NOT NULL,
15    PRIMARY KEY (CODCLI)
16 );
17
18 CREATE TABLE VENDEDOR (
19     CODVEND INTEGER NOT NULL,
20     NOME VARCHAR(50) NOT NULL,
21     SALFIXO FLOAT,
22     FAIXACOMIS CHAR NOT NULL,
23     PRIMARY KEY (CODVEND)
24 );
25
26 CREATE TABLE PRODUTO (
```

At the bottom, the 'Data Output' pane shows a message: '1 Verifique as mensagens de execução do script na aba Messages abaixo'.

b) Ao executar o script, verifique na aba Messages do pgAdmin o andamento do processamento, ou execute no psql. Analise o tempo para execução dos 10%, 20%, 30%... até 100% do processo. O tempo cresce linearmente?

Analisando o tempo de execução do script, comparando 10%, temos que:

- O início da execução aconteceu em 15:18:32.82.
- Em 10%, o tempo foi de 15:19:10.27, apenas 39 segundos do início.
- Em 20%, o tempo foi de 15:19:47.72, cerca de 38 segundos dos 10%.
- Em 30%, o tempo foi de 15:20:27.99, aproximadamente 40 segundos.
- Em 40%, o tempo foi de 15:21:09.60, 42 segundos dos 30%.
- Em 50%, o tempo foi de 15:21:49.28, cerca de 40 segundos.
- Em 60%, o tempo foi de 15:22:29.22, aproximadamente 40 segundos.
- Em 70%, o tempo foi de 15:23:08.86, cerca de 40 segundos.
- Em 80%, o tempo foi de 15:23:48.92, 40 segundos dos 70%.
- Em 90%, o tempo foi de 15:24:28.22, 39 segundos aproximadamente.
- Concluindo a execução com 100%, o tempo final foi de 15:25:14.47, 46 segundos de 90% e cerca de 7 minutos do início.

Com isso, concluímos que o tempo de execução do script cresce linearmente, sendo 40 segundos sua taxa de crescimento.

c) Resolva as consultas respondendo às perguntas em um arquivo texto

Consulta 1:

- Analise o plano de consulta do comando UPDATE. Verifique se os índices de chave primária são empregados.

Analisando o plano de consulta do comando UPDATE, podemos ver que os índices de chave primária não são empregados.

Update on pedido p1 (cost=0.00..48012411.35 rows=0 width=0) (actual time=194415.301..194415.303 rows=0 loops=1)
-> Seq Scan on pedido p1 (cost=0.00..48012411.35 rows=1000000 width=22) (actual time=0.635..174781.814 rows=1000000 loops=1)
SubPlan 1
-> Aggregate (cost=47.97..47.98 rows=1 width=8) (actual time=0.167..0.167 rows=1 loops=1000000)
-> Nested Loop (cost=14.52..47.94 rows=4 width=12) (actual time=0.100..0.165 rows=4 loops=1000000)
-> Nested Loop (cost=0.72..12.75 rows=1 width=4) (actual time=0.009..0.009 rows=1 loops=1000000)
-> Index Scan using pedido_pkey on pedido p (cost=0.42..8.44 rows=1 width=8) (actual time=0.003..0.004 rows=1 loops=1)
Index Cond: (numped = p1.numped)
-> Index Only Scan using cliente_pkey on cliente c (cost=0.29..4.31 rows=1 width=4) (actual time=0.004..0.004 rows=1 loops=1)
Index Cond: (codcli = p.codcli)
Heap Fetches: 0
-> Hash Join (cost=13.80..35.15 rows=4 width=16) (actual time=0.078..0.141 rows=4 loops=1000000)
Hash Cond: (pr.codprod = i.codprod)
-> Seq Scan on produto pr (cost=0.00..20.02 rows=502 width=18) (actual time=0.003..0.042 rows=502 loops=1000000)
-> Hash (cost=13.75..13.75 rows=4 width=18) (actual time=0.010..0.010 rows=4 loops=1000000)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Index Scan using itemped_pkey on itemped i (cost=0.43..13.75 rows=4 width=18) (actual time=0.005..0.007 rows=4 loops=1)
Index Cond: (numped = p1.numped)
Planning Time: 1.334 ms
Execution Time: 194419.784 ms

Update on pedido p1 (cost=0.00..48012411.35 rows=0 width=0)
-> Seq Scan on pedido p1 (cost=0.00..48012411.35 rows=1000000 width=22)
SubPlan 1
-> Aggregate (cost=47.97..47.98 rows=1 width=8)
-> Nested Loop (cost=14.52..47.94 rows=4 width=12)
-> Nested Loop (cost=0.72..12.75 rows=1 width=4)
-> Index Scan using pedido_pkey on pedido p (cost=0.42..8.44 rows=1 width=8)
Index Cond: (numped = p1.numped)
-> Index Only Scan using cliente_pkey on cliente c (cost=0.29..4.31 rows=1 width=4)
Index Cond: (codcli = p.codcli)
-> Hash Join (cost=13.80..35.15 rows=4 width=16)
Hash Cond: (pr.codprod = i.codprod)
-> Seq Scan on produto pr (cost=0.00..20.02 rows=502 width=18)
-> Hash (cost=13.75..13.75 rows=4 width=18)
-> Index Scan using itemped_pkey on itemped i (cost=0.43..13.75 rows=4 width=18)
Index Cond: (numped = p1.numped)

- Verifique se a criação dos índices melhora o desempenho dessa atualização. Qual o tempo (em segundos) para executar a atualização antes e depois de criar os índices?

Antes:

Custo total: 48012411.35

Tempo de execução: 1 min 54 secs.

Depois:

Custo total: 48013519.35

Tempo de execução: 2 min 35 secs

Logo a criação dos índices piorou o desempenho dessa atualização.

Update on pedido p1 (cost=0.00..48013519.35 rows=0 width=0) (actual time=183586.261..183586.263 rows=0 loops=1)
-> Seq Scan on pedido p1 (cost=0.00..48013519.35 rows=1000000 width=22) (actual time=0.431..161016.593 rows=1000000 loops=1)
SubPlan 1
-> Aggregate (cost=47.97..47.98 rows=1 width=8) (actual time=0.154..0.155 rows=1 loops=1000000)
-> Nested Loop (cost=14.52..47.94 rows=4 width=12) (actual time=0.092..0.152 rows=4 loops=1000000)
-> Nested Loop (cost=0.72..12.75 rows=1 width=4) (actual time=0.008..0.008 rows=1 loops=1000000)
-> Index Scan using pedido_pkey on pedido p (cost=0.42..8.44 rows=1 width=8) (actual time=0.003..0.004 rows=1 loops=1000000)
Index Cond: (numped = p1.numped)
-> Index Only Scan using cliente_pkey on cliente c (cost=0.29..4.31 rows=1 width=4) (actual time=0.003..0.003 rows=1 loops=1000000)
Index Cond: (codcli = p.codcli)
Heap Fetches: 0
-> Hash Join (cost=13.80..35.15 rows=4 width=16) (actual time=0.072..0.131 rows=4 loops=1000000)
Hash Cond: (pr.codprod = i.codprod)
-> Seq Scan on produto pr (cost=0.00..20.02 rows=502 width=18) (actual time=0.002..0.039 rows=502 loops=1000000)
-> Hash (cost=13.75..13.75 rows=4 width=18) (actual time=0.009..0.009 rows=4 loops=1000000)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Index Scan using itemped_pkey on itemped i (cost=0.43..13.75 rows=4 width=18) (actual time=0.005..0.006 rows=4 loops=1000000)
Index Cond: (numped = p1.numped)
Planning Time: 1.375 ms
Execution Time: 183590.081 ms

Update on pedido p1 (cost=0.00..48013519.35 rows=0 width=0)
-> Seq Scan on pedido p1 (cost=0.00..48013519.35 rows=1000000 width=22)
SubPlan 1
-> Aggregate (cost=47.97..47.98 rows=1 width=8)
-> Nested Loop (cost=14.52..47.94 rows=4 width=12)
-> Nested Loop (cost=0.72..12.75 rows=1 width=4)
-> Index Scan using pedido_pkey on pedido p (cost=0.42..8.44 rows=1 width=8)
Index Cond: (numped = p1.numped)
-> Index Only Scan using cliente_pkey on cliente c (cost=0.29..4.31 rows=1 width=4)
Index Cond: (codcli = p.codcli)
-> Hash Join (cost=13.80..35.15 rows=4 width=16)
Hash Cond: (pr.codprod = i.codprod)
-> Seq Scan on produto pr (cost=0.00..20.02 rows=502 width=18)
-> Hash (cost=13.75..13.75 rows=4 width=18)
-> Index Scan using itemped_pkey on itemped i (cost=0.43..13.75 rows=4 width=18)
Index Cond: (numped = p1.numped)

Consulta 2:

- Selecione os clientes que compraram em novembro/2021, apresente o valor total dos pedidos por cliente e a quantidade de pedidos realizados por cliente. Ordene o resultado em ordem descendente de valor total para permitir encontrar os maiores compradores.

nome	count	soma
character varying (50)	bigint	double precision
Anderson Souza	14	158186.40000000002
Bruno Santos	15	141031.3
Rita Gomes	15	134497.49999999997
Lucas Freitas	13	107236.90000000001
Guilherme Lopes	12	106686.59999999998
Paulo Santos	12	102208.70000000001
Pedro Miranda	9	100502.49999999999
Elaine Souza	10	98654.09999999999
Tais Santos	8	97602.49999999999
Gabriel Silva	11	97469.6

- Analise o plano de consulta do comando SELECT. Verifique se os índices de chave primária são empregados.

Sort (cost=91054.24..91169.94 rows=46279 width=38)	Sort (cost=91054.24..91169.94 rows=46279 width=38) (actual time=1790.530..1797.921 rows=21975 loops=1)
Sort Key: (sum(((i.quant)::double precision * pr.valunit))) DESC	Sort Key: (sum(((i.quant)::double precision * pr.valunit))) DESC
-> GroupAggregate (cost=70049.92..87468.06 rows=46279 width=38)	Sort Method: quicksort Memory: 2554kB
Group Key: c.nome	-> GroupAggregate (cost=70049.92..87468.06 rows=46279 width=38) (actual time=1478.252..1766.070 rows=21975 loops=1)
-> Gather Merge (cost=70049.92..85361.89 rows=131471 width=38)	Group Key: c.nome
Workers Planned: 2	-> Gather Merge (cost=70049.92..85361.89 rows=131471 width=38) (actual time=1478.190..1658.799 rows=128443 loops=1)
-> Sort (cost=69049.90..69186.85 rows=54780 width=38)	Workers Planned: 2
Sort Key: c.nome	Workers Launched: 2
-> Hash Join (cost=18110.54..64738.34 rows=54780 width=38)	-> Sort (cost=69049.90..69186.85 rows=54780 width=38) (actual time=1394.504..1426.242 rows=42814 loops=3)
Hash Cond: (i.codprod = pr.codprod)	Sort Key: c.nome
-> Parallel Hash Join (cost=18084.24..64567.03 rows=54780 width=40)	Sort Method: external merge Disk: 2200kB
Hash Cond: (p.codcli = c.codcli)	Worker 0: Sort Method: external merge Disk: 2248kB
-> Parallel Hash Join (cost=15542.71..61881.69 rows=54780 width=22)	Worker 1: Sort Method: external merge Disk: 2024kB
Hash Cond: (i.numped = p.numped)	-> Hash Join (cost=18110.54..64738.34 rows=54780 width=38) (actual time=39.823..835.479 rows=42814 loops=3)
-> Parallel Seq Scan on itemped i (cost=0.00..41980.95 rows=1660195 width=18)	Hash Cond: (i.codprod = pr.codprod)
-> Parallel Hash (cost=15370.86..15370.86 rows=13748 width=8)	-> Parallel Hash Join (cost=18084.24..64567.03 rows=54780 width=40) (actual time=39.307..809.783 rows=42814 loops=3)
-> Parallel Bitmap Heap Scan on pedido p (cost=458.63..15370.86 rows=13748 width=8)	Hash Cond: (p.codcli = c.codcli)
Recheck Cond: ((data >= '2021-11-01'::date) AND (data <= '2021-11-30'::date))	-> Parallel Hash Join (cost=15542.71..61881.69 rows=54780 width=22) (actual time=12.047..758.776 rows=42814 loops=3)
-> Bitmap Index Scan on pedido_data (cost=0.00..450.38 rows=32996 width=0)	Hash Cond: (i.numped = p.numped)
Index Cond: ((data >= '2021-11-01'::date) AND (data <= '2021-11-30'::date))	-> Parallel Seq Scan on itemped i (cost=0.00..41980.95 rows=1660195 width=18) (actual time=0.010..380.688 rows=1328156 loops=3)
-> Parallel Hash (cost=1806.24..1806.24 rows=58824 width=26)	-> Parallel Hash (cost=15370.86..15370.86 rows=13748 width=8) (actual time=7.859..7.861 rows=10764 loops=3)
-> Parallel Seq Scan on cliente c (cost=0.00..1806.24 rows=58824 width=26)	Buckets: 65536 Batches: 1 Memory Usage: 1824kB
-> Hash (cost=20.02..20.02 rows=502 width=18)	-> Parallel Bitmap Heap Scan on pedido p (cost=458.63..15370.86 rows=13748 width=8) (actual time=0.476..2.967 rows=10764 loops=3)
-> Seq Scan on produto pr (cost=0.00..20.02 rows=502 width=18)	Recheck Cond: ((data >= '2021-11-01'::date) AND (data <= '2021-11-30'::date))
	Heap Blocks: exact=164
	-> Bitmap Index Scan on pedido_data (cost=0.00..450.38 rows=32996 width=0) (actual time=1.300..1.300 rows=32292 loops=1)
	Index Cond: ((data >= '2021-11-01'::date) AND (data <= '2021-11-30'::date))
	-> Parallel Hash (cost=1806.24..1806.24 rows=58824 width=26) (actual time=26.864..26.865 rows=33333 loops=3)
	Buckets: 131072 Batches: 1 Memory Usage: 7104kB
	-> Parallel Seq Scan on cliente c (cost=0.00..1806.24 rows=58824 width=26) (actual time=0.009..35.002 rows=100000 loops=1)
	-> Hash (cost=20.02..20.02 rows=502 width=18) (actual time=0.398..0.399 rows=502 loops=3)
	Buckets: 1024 Batches: 1 Memory Usage: 33kB
	-> Seq Scan on produto pr (cost=0.00..20.02 rows=502 width=18) (actual time=0.075..0.185 rows=502 loops=3)
	Planning Time: 1.253 ms
	Execution Time: 1907.544 ms

Analisando o plano de consulta do comando SELECT, podemos ver que os índices de chave primária são empregados.

- Verifique se a criação do índice melhora (ou piora) o desempenho dessa consulta. Qual o tempo (em segundos) para executar a consulta sem o índice e com o índice?

Antes (sem):

Custo total: 96755.08

Tempo de execução: 1 sec 580 msec

Depois (com):

Custo total: 91169.94

Tempo de execução: 1 secs 862 msec

Logo a criação dos índices melhorou o desempenho dessa atualização. Mesmo que o tempo de execução da consulta com os índices seja maior, o custo total é 5,77% menor que a consulta sem o índice. Se essa consulta fosse maior, como não limitado a novembro/2021, provavelmente o tempo de execução seria menor com o índice.

PS: Essas análises abaixo são sem o índice, as de cima são com os índices.

Sort (cost=96639.38..96755.08 rows=46279 width=38)	Sort (cost=96639.38..96755.08 rows=46279 width=38) (actual time=2014.293..2023.639 rows=21975 loops=1)
Sort Key: (sum(((i.quant)::double precision * pr.valunit))) DESC	Sort Key: (sum(((i.quant)::double precision * pr.valunit))) DESC
	Sort Method: quicksort Memory: 2554kB
-> GroupAggregate (cost=75635.06..93053.20 rows=46279 width=38)	-> GroupAggregate (cost=75635.06..93053.20 rows=46279 width=38) (actual time=1718.104..1993.522 rows=21975 loops=1)
Group Key: c.nome	Group Key: c.nome
-> Gather Merge (cost=75635.06..90947.03 rows=131471 width=38)	-> Gather Merge (cost=75635.06..90947.03 rows=131471 width=38) (actual time=1718.044..1892.697 rows=128443 loops=1)
Workers Planned: 2	Workers Planned: 2
	Workers Launched: 2
-> Sort (cost=74635.04..74771.99 rows=54780 width=38)	-> Sort (cost=74635.04..74771.99 rows=54780 width=38) (actual time=1631.865..1659.746 rows=42814 loops=3)
Sort Key: c.nome	Sort Key: c.nome
	Sort Method: external merge Disk: 2024kB
	Worker 0: Sort Method: external merge Disk: 2024kB
	Worker 1: Sort Method: external merge Disk: 2424kB
-> Hash Join (cost=23695.68..70323.48 rows=54780 width=38)	-> Hash Join (cost=23695.68..70323.48 rows=54780 width=38) (actual time=213.197..977.244 rows=42814 loops=3)
Hash Cond: (i.codprod = pr.codprod)	Hash Cond: (i.codprod = pr.codprod)
-> Parallel Hash Join (cost=23669.39..70152.17 rows=54780 width=40)	-> Parallel Hash Join (cost=23669.39..70152.17 rows=54780 width=40) (actual time=212.549..939.171 rows=42814 loops=3)
Hash Cond: (p.codcli = c.codcli)	Hash Cond: (p.codcli = c.codcli)
-> Parallel Hash Join (cost=21127.85..67466.83 rows=54780 width=22)	-> Parallel Hash Join (cost=21127.85..67466.83 rows=54780 width=22) (actual time=186.930..878.277 rows=42814 loops=3)
Hash Cond: (i.numped = p.numped)	Hash Cond: (i.numped = p.numped)
-> Parallel Seq Scan on itemped i (cost=0.00..41980.95 rows=1660195 width=18)	-> Parallel Seq Scan on itemped i (cost=0.00..41980.95 rows=1660195 width=18) (actual time=0.015..341.040 rows=1328156 loops=1)
-> Parallel Hash (cost=20956.00..20956.00 rows=13748 width=8)	-> Parallel Hash (cost=20956.00..20956.00 rows=13748 width=8) (actual time=178.887..178.888 rows=10764 loops=3)
	Buckets: 65536 Batches: 1 Memory Usage: 1824kB
	-> Parallel Seq Scan on pedido p (cost=0.00..20956.00 rows=13748 width=8) (actual time=163.040..175.074 rows=10764 loops=3)
	Filter: ((data >= '2021-11-01'::date) AND (data <= '2021-11-30'::date))
	Rows Removed by Filter: 322569
-> Parallel Seq Scan on pedido p (cost=0.00..20956.00 rows=13748 width=8)	-> Parallel Hash (cost=1806.24..1806.24 rows=58824 width=26) (actual time=25.231..25.232 rows=33333 loops=3)
Filter: ((data >= '2021-11-01'::date) AND (data <= '2021-11-30'::date))	Buckets: 131072 Batches: 1 Memory Usage: 7104kB
-> Parallel Hash (cost=1806.24..1806.24 rows=58824 width=26)	-> Parallel Seq Scan on cliente c (cost=0.00..1806.24 rows=58824 width=26) (actual time=0.012..28.508 rows=100000 loops=1)
-> Parallel Seq Scan on cliente c (cost=0.00..1806.24 rows=58824 width=26)	-> Hash (cost=20.02..20.02 rows=502 width=18) (actual time=0.468..0.468 rows=502 loops=3)
	Buckets: 1024 Batches: 1 Memory Usage: 33kB
-> Hash (cost=20.02..20.02 rows=502 width=18)	-> Seq Scan on produto pr (cost=0.00..20.02 rows=502 width=18) (actual time=0.106..0.283 rows=502 loops=3)
-> Seq Scan on produto pr (cost=0.00..20.02 rows=502 width=18)	Planning Time: 1.439 ms
	Execution Time: 2033.316 ms

Consulta 3:

- Crie um trigger para manter o atributo PEDIDO.VALORTOTAL atualizado.

```
SET search_path TO novovendas;
CREATE TRIGGER t_att_ValorTotal
BEFORE INSERT OR UPDATE OR DELETE ON itemped
FOR EACH ROW
EXECUTE PROCEDURE att_ValorTotal();

CREATE OR REPLACE FUNCTION att_ValorTotal() RETURNS TRIGGER
AS
$$
DECLARE
    vf integer; -- quantidade total de cada item pedido
    np integer; -- novo valor do numped
BEGIN
    np = NEW numped;
    SELECT SUM(p.valunit * i.quant) FROM produto as p, itemped as i
    WHERE p.codprod = i.codprod AND i.numped = np GROUP BY i.numped INTO vf;
    update pedido set valortotal = vf WHERE pedido.numped = np;

END
$$ LANGUAGE plpgsql;
```

Consulta 4:

- Verifique se o Postgres utiliza o índice composto PEDIDO_PRAZO_DATA para execução da consulta abaixo considerando a existência ou não do índice PEDIDO_DATA.

Os dois índices

```
Bitmap Heap Scan on pedido (cost=321.58..11895.30 rows=6233 width=27) (actual time=0.410..1.817 rows=6334 loops=1)
  Recheck Cond: ((data >= '2022-10-15'::date) AND (data <= '2022-10-30'::date) AND (prazoentr >= 15) AND (prazoentr <= 25))
  Heap Blocks: exact=128
-> Bitmap Index Scan on pedido_prazo_data (cost=0.00..320.03 rows=6233 width=0) (actual time=0.369..0.369 rows=6334 loops=1)
   Index Cond: ((data >= '2022-10-15'::date) AND (data <= '2022-10-30'::date) AND (prazoentr >= 15) AND (prazoentr <= 25))
Planning Time: 0.284 ms
Execution Time: 2.119 ms
```

Somente pedido_prazo_data

Bitmap Heap Scan on pedido (cost=321.58..11895.30 rows=6233 width=27) (actual time=0.388..1.676 rows=6334 loops=1)
Recheck Cond: (((data >= '2022-10-15'::date) AND (data <= '2022-10-30'::date) AND (prazoentr >= 15) AND (prazoentr <= 25)))
Heap Blocks: exact=128
-> Bitmap Index Scan on pedido_prazo_data (cost=0.00..320.03 rows=6233 width=0) (actual time=0.358..0.358 rows=6334 loops=1)
Index Cond: (((data >= '2022-10-15'::date) AND (data <= '2022-10-30'::date) AND (prazoentr >= 15) AND (prazoentr <= 25)))
Planning Time: 0.433 ms
Execution Time: 1.978 ms

Somente pedido_data

Bitmap Heap Scan on pedido (cost=236.38..16009.98 rows=6233 width=27) (actual time=1.122..5.321 rows=6334 loops=1)
Recheck Cond: (((data >= '2022-10-15'::date) AND (data <= '2022-10-30'::date)))
Filter: ((prazoentr >= 15) AND (prazoentr <= 25))
Rows Removed by Filter: 10956
Heap Blocks: exact=128
-> Bitmap Index Scan on pedido_data (cost=0.00..234.83 rows=17040 width=0) (actual time=1.091..1.092 rows=17290 loops=1)
Index Cond: (((data >= '2022-10-15'::date) AND (data <= '2022-10-30'::date)))
Planning Time: 0.486 ms
Execution Time: 5.610 ms

Sem nenhum

Gather (cost=1000.00..24662.63 rows=6233 width=27) (actual time=107.993..334.099 rows=6334 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Parallel Seq Scan on pedido (cost=0.00..23039.33 rows=2597 width=27) (actual time=37.858..255.544 rows=2111 loops=3)
Filter: (((prazoentr >= 15) AND (prazoentr <= 25) AND (data >= '2022-10-15'::date) AND (data <= '2022-10-30'::date)))
Rows Removed by Filter: 331222
Planning Time: 0.209 ms
Execution Time: 334.734 ms

Podemos ver que, mesmo com o os dois índices, não altera em nada a execução da consulta. O tempo de execução é mais lento contendo os dois índices, do que com apenas o pedido_prazo_data.

Mesmo testando varias vezes, os tempos variam de acordo com:

Os dois índices: variam de 0.94 pra 2.01.

Com somente o índice pedido_prazo_data: 0.75 pra 1.60.

Com isso, concluímos que o Postgres não utiliza o índice composto PEDIDO_PRAZO_DATA para execução da consulta considerando a existência do índice PEDIDO_DATA.

OBS: Como quase todos os exercícios eram escritos, decidi fazê-lo completo em pdf. O exercício do trigger está anexado trigger.sql.