Todos os testes foram realizados no PG Admin 3 v 1.22.2

Especificações do computador:
Nome do Sistema Operacional: Microsoft Windows 10 Home Single Language
Versão: 10.0.17134 Compilação 17134
Modelo do sistema: Aspire E1-572
Processador: Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz, 2301 Mhz, 2 Núcleo(s), 4 Processador(es) Lógico(s)
Memória Física (RAM) Instalada8,00 GB

Informações do SSD
Modelo:KINGSTON SA400S37240G
Tamanho 223,57 GB (240.054.796.800 bytes)
Interface: SATA Rev. 3.0 (6Gb/s) compatível com a versão anterior SATA Rev. 2.0 (3Gb/s)
Leituras: ATÉ - 550MBs
Gravações: ATÉ - 490MBs

Consulta de despesas de um cartão por um determinado mês:

```
SELECT d.* as qtd_d FROM despesa d
JOIN parcelamento p ON (p.fk_despesa = d.id)
JOIN cartao c ON (p.fk_cartao = c.id)
WHERE c.id =  85107 AND d.data_compra > '2018-11-01' AND d.data_compra < '2018-11-30'
```

Testes sem indíce:

Planning time(ms)	0.543	0.679	0.815	0.353	0.369	0.351	0.345
Execution time (ms)	191.964	265.687	190.497	190.347	192.308	187.168	191.624

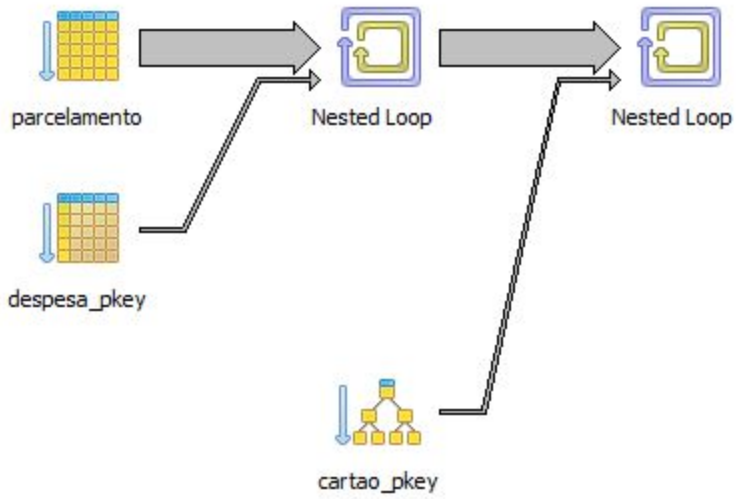
Tirando o maior tempo e o menor tempo, temos como média:
Planning time: 0.459 ms
Execution time: 191.348 ms

Data output:

```
Nested Loop  (cost=0.85..21571.59 rows=1 width=47) (actual time=95.962..337.982 rows=5 loops=1)
  Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag
  ->  Nested Loop  (cost=0.43..21563.14 rows=1 width=51) (actual time=94.725..336.699 rows=5 loops=1)
        Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag, p.fk_cartao
        ->  Seq Scan on public.parcelamento p  (cost=0.00..21487.00 rows=9 width=8) (actual time=80.458..315.858 rows=14 loops=1)
              Output: p.num_parcelas, p.fk_despesa, p.fk_cartao
```

Filter: (p.fk_cartao = 85107)
Rows Removed by Filter: 1199986
-> Index Scan using despesa_pkey on public.despesa d (cost=0.43..8.45 rows=1 width=47) (actual time=1.484..1.484 rows=0 loops=14)
Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag
Index Cond: (d.id = p.fk_despesa)
Filter: ((d.data_compra > '2018-11-01'::date) AND (d.data_compra < '2018-11-30'::date))
Rows Removed by Filter: 1
-> Index Only Scan using cartao_pkey on public.cartao c (cost=0.42..8.44 rows=1 width=4) (actual time=0.253..0.254 rows=1 loops=5)
Output: c.id
Index Cond: (c.id = 85107)
Heap Fetches: 5
Planning time: 42.614 ms
Execution time: 338.262 ms

Explain:



Criando índice B-tree:
Foi criado um índice na chave estrangeira do cartão
CREATE INDEX idx_parcelamento ON parcelamento(fk_cartao);

Query returned successfully with no result in 1.5 secs.

Data_output:
Nested Loop (cost=5.34..124.19 rows=1 width=47) (actual time=0.210..0.659 rows=5 loops=1)
-> Nested Loop (cost=4.92..115.74 rows=1 width=51) (actual time=0.200..0.639 rows=5 loops=1)
-> Bitmap Heap Scan on parcelamento p (cost=4.50..39.60 rows=9 width=8) (actual time=0.154..0.541 rows=14 loops=1)
Recheck Cond: (fk_cartao = 85107)
Heap Blocks: exact=14

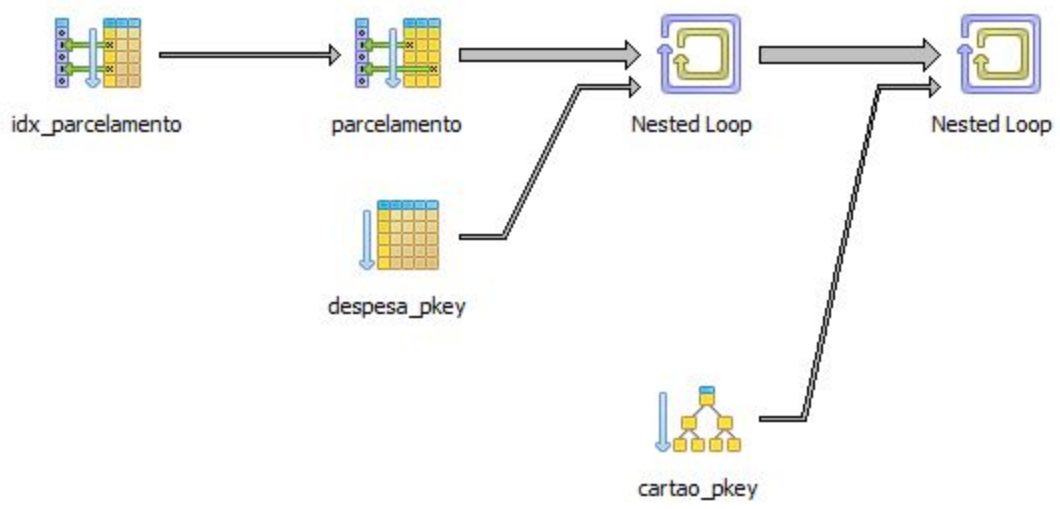
```
          ->  Bitmap Index Scan on idx_parcelamento  (cost=0.00..4.50 rows=9 width=0) (actual
time=0.090..0.090 rows=14 loops=1)
                Index Cond: (fk_cartao = 85107)
      ->  Index Scan using despesa_pkey on despesa d  (cost=0.43..8.45 rows=1 width=47) (actual
time=0.006..0.006 rows=0 loops=14)
          Index Cond: (id = p.fk_despesa)
          Filter: ((data_compra > '2018-11-01'::date) AND (data_compra < '2018-11-30'::date))
          Rows Removed by Filter: 1
 ->  Index Only Scan using cartao_pkey on cartao c  (cost=0.42..8.44 rows=1 width=4) (actual
time=0.003..0.003 rows=1 loops=5)
      Index Cond: (id = 85107)
      Heap Fetches: 5
Planning time: 1.092 ms
Execution time: 0.731 ms
```

Testes com índice:

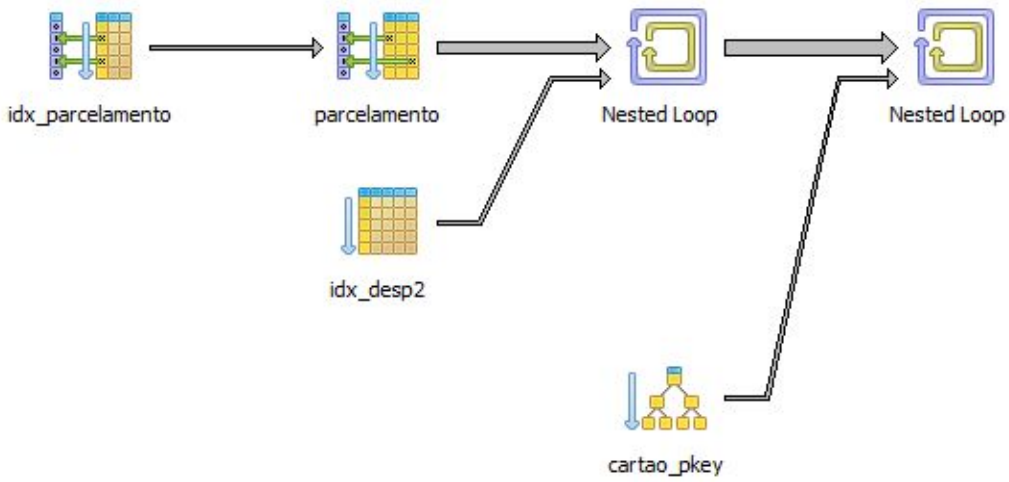
Planning time(ms)	1.092	0.457	0.632	0.368	0.371	0.656	0.365
Execution time (ms)	0.731	0.191	0.141	0.153	0.152	0.141	0.152

Tirando o maior tempo e o menor tempo, temos como média:
Planning time: 0.497 ms
Execution time: 0.158 ms

Explain:



Criei um indice para a chave estrangeira de despesa em parcelamento, mas não houve mudança no explain, ou seja, continuou a mesma figura acima.
Então decidi criar um índice na chave primária de despesa(com o nome idx_desp2, então explain ficou desta forma:

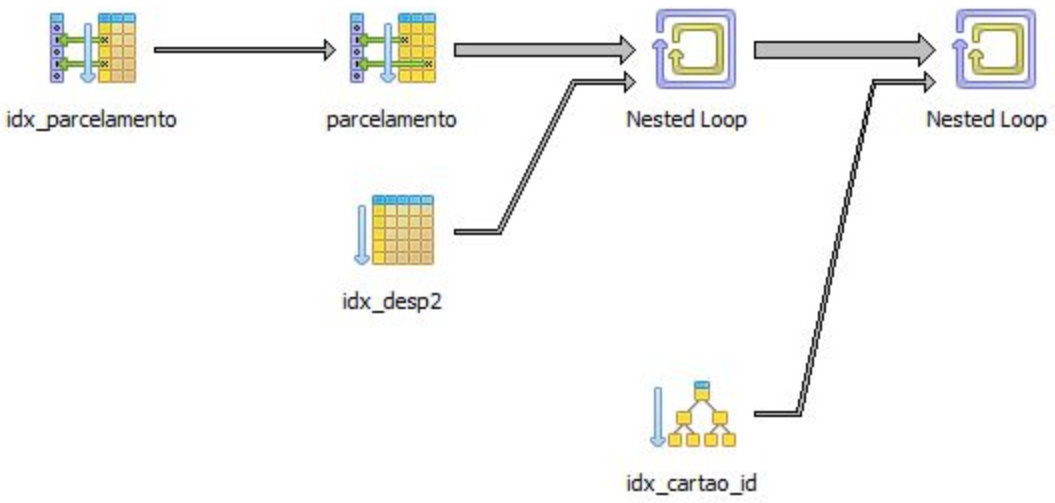


Testes de desempenho:

Planning time(ms)	0.405	0.444	0.547	0.398	0.448	0.392	0.394
Execution time (ms)	1.029	0.148	0.165	0.138	0.143	0.138	0.137

Como podemos ver não houveram mudanças significativas, mesmo o explain demonstrado que o idx_desp2 foi utilizado. Obviamente, já chegamos em tempos expressivamente baixos com o primeiro índice.

Então criamos um indice no cartão.



Podemos ver que apesar de utilizar o índice, a forma continua a mesma.

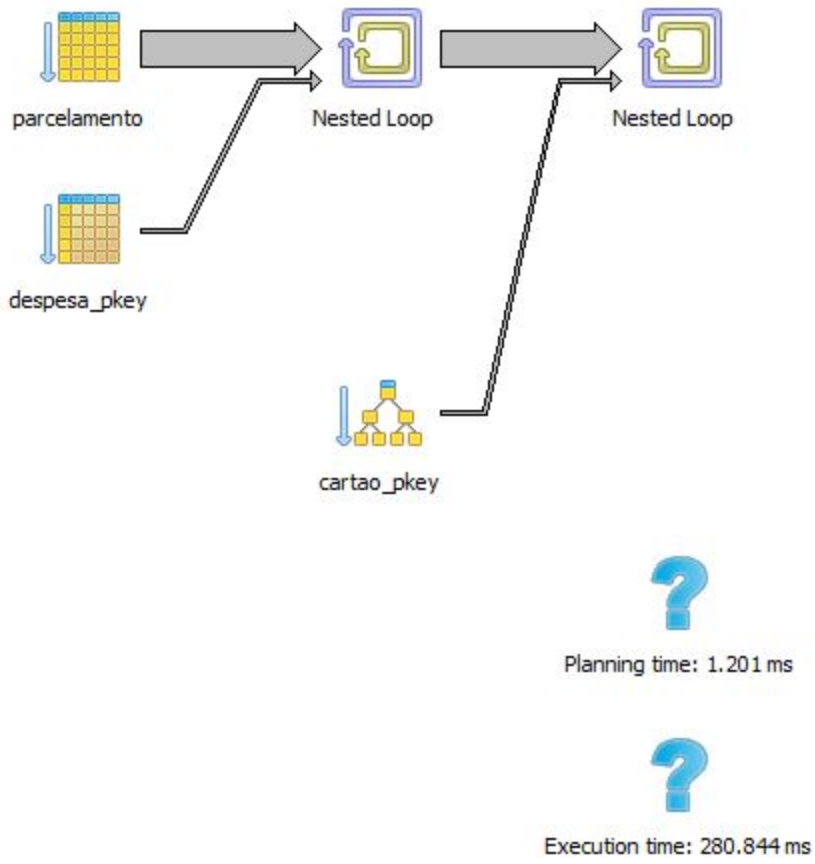
Removemos todos os índices B-tree, para testar outros tipos de índices.

USANDO GIN

CREATE INDEX gin_idx_despesa ON despesa USING gin (nome gin_trgm_ops);
Query returned successfully with no result in 15.8 secs.

Data output:
"Nested Loop (cost=0.85..21571.59 rows=1 width=47) (actual time=112.718..239.564 rows=5 loops=1)"
" -> Nested Loop (cost=0.43..21563.14 rows=1 width=51) (actual time=112.393..239.205 rows=5 loops=1)"
"   -> Seq Scan on parcelamento p (cost=0.00..21487.00 rows=9 width=8) (actual time=104.033..238.996 rows=14 loops=1)"
"   Filter: (fk_cartao = 85107)"
"   Rows Removed by Filter: 1199986"
"   -> Index Scan using despesa_pkey on despesa d (cost=0.43..8.45 rows=1 width=47) (actual time=0.011..0.011 rows=0 loops=14)"
"   Index Cond: (id = p.fk_despesa)"
"   Filter: ((data_compra > '2018-11-01'::date) AND (data_compra < '2018-11-30'::date))"
"   Rows Removed by Filter: 1"
" -> Index Only Scan using cartao_pkey on cartao c (cost=0.42..8.44 rows=1 width=4) (actual time=0.070..0.070 rows=1 loops=5)"
"   Index Cond: (id = 85107)"
"   Heap Fetches: 5"
"Planning time: 16.473 ms"
"Execution time: 239.717 ms"

Testes de desempenho:



Como podemos ver para a consulta de despesas por cartão, este índice não surtiu efeito. Podemos dizer que ele até piorou um pouco.

O GIN, com alguns testes a parte, para uma consulta que pesquise por nome, ele melhora sutilmente o desempenho, e alguns poucos casos até 50% do tempo.

Usando GIST:

CREATE INDEX gist_idx_parc ON parcelamento USING gist(fk_cartao);

Query returned successfully with no result in 15.5 secs.

Data output:
"Nested Loop (cost=5.20..124.05 rows=1 width=47) (actual time=0.146..0.558 rows=5 loops=1)"
" -> Nested Loop (cost=4.78..115.60 rows=1 width=51) (actual time=0.138..0.534 rows=5 loops=1)"
"   -> Bitmap Heap Scan on parcelamento p (cost=4.35..39.46 rows=9 width=8) (actual time=0.093..0.435 rows=14 loops=1)"
"   Recheck Cond: (fk_cartao = 85107)"
"   Heap Blocks: exact=14"
"   -> Bitmap Index Scan on gist_idx_parc (cost=0.00..4.35 rows=9 width=0) (actual time=0.038..0.038 rows=14 loops=1)"
"   Index Cond: (fk_cartao = 85107)"

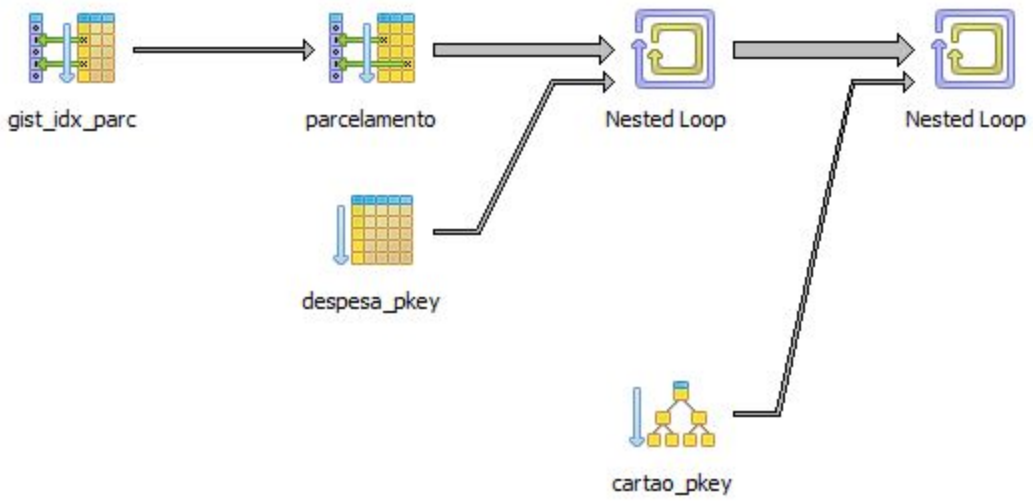
" -> Index Scan using despesa_pkey on despesa d (cost=0.43..8.45 rows=1 width=47) (actual time=0.006..0.006 rows=0 loops=14)"
" Index Cond: (id = p.fk_despesa)"
" Filter: ((data_compra > '2018-11-01'::date) AND (data_compra < '2018-11-30'::date))"
" Rows Removed by Filter: 1"
" -> Index Only Scan using cartao_pkey on cartao c (cost=0.42..8.44 rows=1 width=4) (actual time=0.004..0.004 rows=1 loops=5)"
" Index Cond: (id = 85107)"
" Heap Fetches: 5"
"Planning time: 5.651 ms"
"Execution time: 0.624 ms"

Testes de desempenho:

Planning time(ms)	5.651	0.448	0.358	0.488	0.355	0.668	0.355
Execution time (ms)	0.624	0.241	0.184	0.242	0.196	0.259	0.185

Tirando o maior tempo e o menor tempo, temos como média:
Planning time: 0.463 ms
Execution time: 0.225 ms

Explain:



Agora será feito testes com outra consulta:

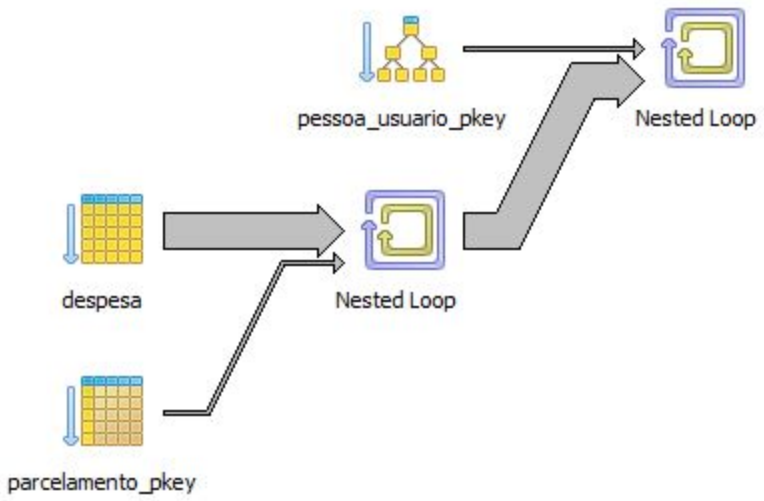
SELECT parcelamento.*,despesa.* FROM despesa
INNER JOIN pessoa_usuario on (pessoa_usuario.id = despesa.fk_pessoa_usuario)

INNER JOIN parcelamento on (parcelamento.fk_despesa = despesa.id)
WHERE pessoa_usuario.id = 2291 AND despesa.data_compra > '2018-11-01' AND despesa.data_compra < '2018-11-30';

Testes sem índice:

Data output:
"Nested Loop (cost=0.71..41471.08 rows=5 width=59) (actual time=2.926..247.296 rows=11 loops=1)"
" -> Index Only Scan using pessoa_usuario_pkey on pessoa_usuario (cost=0.29..8.30 rows=1 width=4) (actual time=0.089..0.091 rows=1 loops=1)"
"   Index Cond: (id = 2291)"
"   Heap Fetches: 1"
" -> Nested Loop (cost=0.43..41462.73 rows=5 width=59) (actual time=2.835..247.197 rows=11 loops=1)"
"   -> Seq Scan on despesa (cost=0.00..41412.00 rows=6 width=47) (actual time=2.088..238.969 rows=13 loops=1)"
"   Filter: ((data_compra > '2018-11-01'::date) AND (data_compra < '2018-11-30'::date) AND (fk_pessoa_usuario = 2291))"
"   Rows Removed by Filter: 1499987"
"   -> Index Scan using parcelamento_pkey on parcelamento (cost=0.43..8.45 rows=1 width=12) (actual time=0.626..0.627 rows=1 loops=13)"
"   Index Cond: (fk_despesa = despesa.id)"
"Planning time: 0.782 ms"
"Execution time: 247.345 ms"

Explain:



Testes de desempenho:

Planning time(ms)	0.782	0.288	0.304	0.327	0.288	0.291	0.288

Execution time (ms)	247.345	239.091	241.255	279.761	237.932	231.078	233.426

Tirando o maior tempo e o menor tempo, temos como média:
Planning time: 0.299 ms
Execution time: 239.809 ms

Usando B-tree:
O índice utilizaddo para esta consulta será:

CREATE INDEX idx_despesa ON despesa(fk_pessoa_usuario);

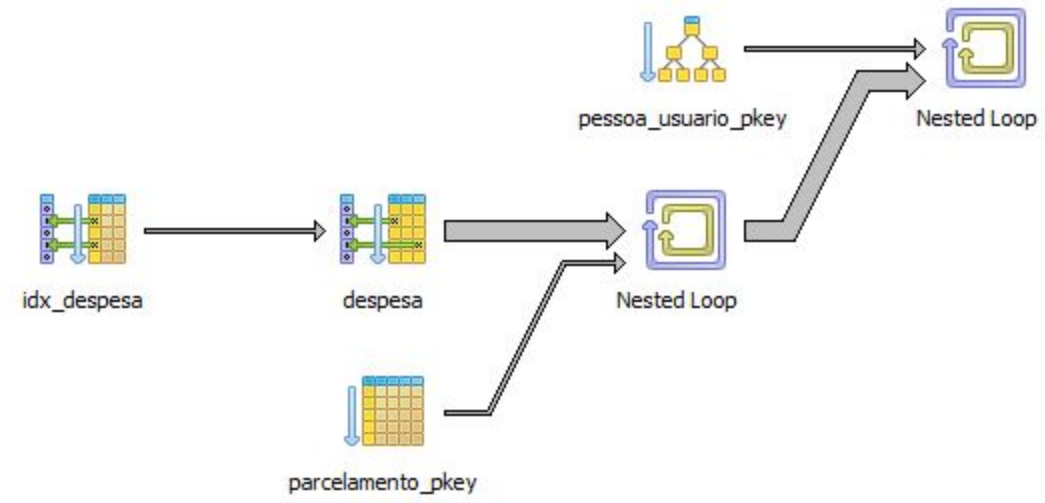
Tempo de criação do índice:
Query returned successfully with no result in 1.9 secs.

Testes com o índice:

Data output:

"Nested Loop (cost=6.24..611.76 rows=5 width=59) (actual time=0.188..1.959 rows=11 loops=1)"
" -> Index Only Scan using pessoa_usuario_pkey on pessoa_usuario (cost=0.29..8.30 rows=1 width=4) (actual time=0.007..0.007 rows=1 loops=1)"
"   Index Cond: (id = 2291)"
"   Heap Fetches: 1"
" -> Nested Loop (cost=5.96..603.41 rows=5 width=59) (actual time=0.172..1.941 rows=11 loops=1)"
"   -> Bitmap Heap Scan on despesa (cost=5.53..552.68 rows=6 width=47) (actual time=0.165..1.855 rows=13 loops=1)"
"   Recheck Cond: (fk_pessoa_usuario = 2291)"
"   Filter: ((data_compra > '2018-11-01'::date) AND (data_compra < '2018-11-30'::date))"
"   Rows Removed by Filter: 156"
"   Heap Blocks: exact=169"
"   -> Bitmap Index Scan on idx_despesa (cost=0.00..5.53 rows=147 width=0) (actual time=0.078..0.078 rows=169 loops=1)"
"   Index Cond: (fk_pessoa_usuario = 2291)"
"   -> Index Scan using parcelamento_pkey on parcelamento (cost=0.43..8.45 rows=1 width=12) (actual time=0.005..0.006 rows=1 loops=13)"
"   Index Cond: (fk_despesa = despesa.id)"
"Planning time: 0.987 ms"
"Execution time: 2.011 ms"

Explain:



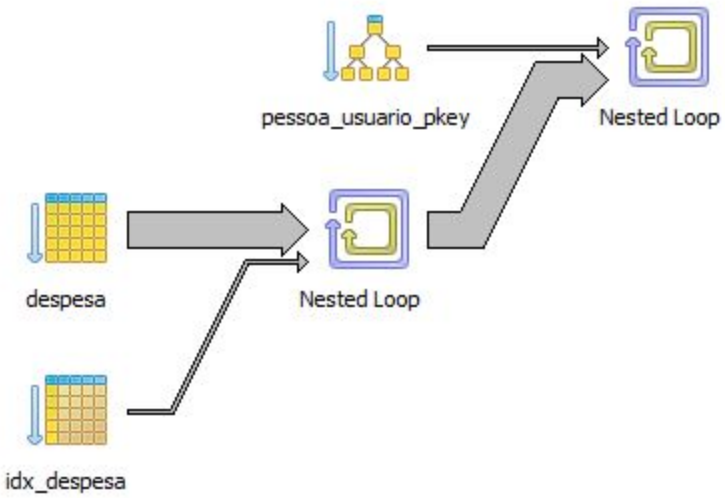
Testes de desempenho:

Planning time(ms)	0.987	0.386	0.326	0.620	0.306	0.320	0.323
Execution time (ms)	2.011	0.596	0.334	0.320	0.311	0.344	0.344

Tirando o maior tempo e o menor tempo, temos como média:
Planning time: 0.395 ms
Execution time: 0.388 ms

Ainda para esta mesma consulta, decidimos criar um índice para chave estrangeira de despesa, e claro removemos o índice anterior:
CREATE INDEX idx_despesa_parc ON parcelamento(fk_despesa);

Explain:



Planning time: 0.666 ms

Execution time: 252.844 ms

Acima podemos ver que não houve melhora no tempo, apesar de remover um processo(parcelamento p_key), o processo da despesa ficou mais custosa.

Como vimos anteriormente, o tipo de índice GIN não é muito usual para as consultas que estamos utilizando em nosso sistema, por isso, não usarei o GIN nesta ou em outras consultas.

Apesar de não ser um índice muito recomendado, será utilizado o índice hash, apenas por fins de testes de desempenho:

Usando índice HASH

CREATE INDEX hash_idx_despesa ON despesa USING hash (fk_pessoa_usuario);

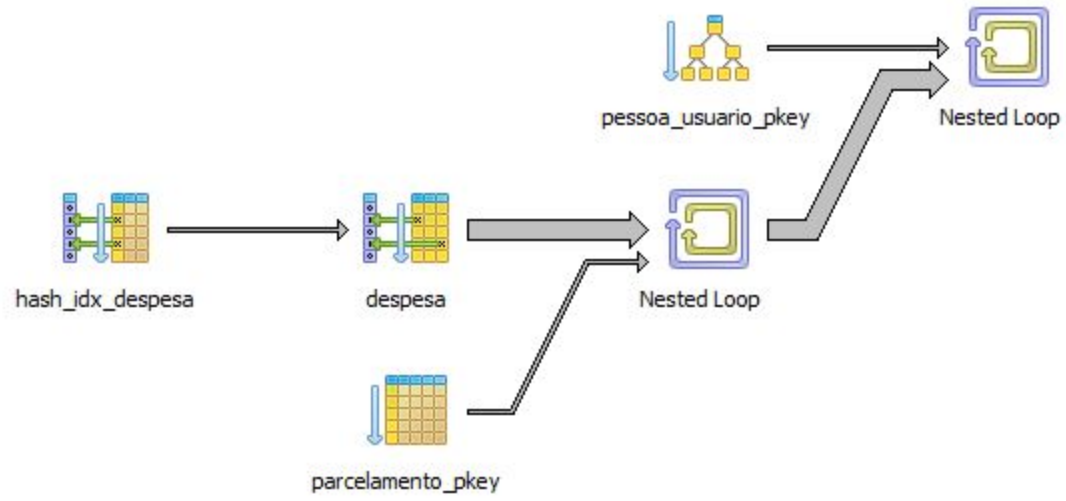
WARNING: hash indexes are not WAL-logged and their use is discouraged
Query returned successfully with no result in 2.8 secs.

Utilizando índice na consulta:

Data output:
"Nested Loop (cost=5.82..611.34 rows=5 width=59) (actual time=0.097..0.335 rows=11 loops=1)"
" -> Index Only Scan using pessoa_usuario_pkey on pessoa_usuario (cost=0.29..8.30 rows=1 width=4) (actual time=0.007..0.007 rows=1 loops=1)"

"   Index Cond: (id = 2291)"
"   Heap Fetches: 1"
" -> Nested Loop (cost=5.53..602.98 rows=5 width=59) (actual time=0.082..0.316 rows=11 loops=1)"
"   -> Bitmap Heap Scan on despesa (cost=5.10..552.25 rows=6 width=47) (actual time=0.075..0.240 rows=13 loops=1)"
"   Recheck Cond: (fk_pessoa_usuario = 2291)"
"   Filter: ((data_compra > '2018-11-01'::date) AND (data_compra < '2018-11-30'::date))"
"   Rows Removed by Filter: 156"
"   Heap Blocks: exact=169"
"   -> Bitmap Index Scan on hash_idx_despesa (cost=0.00..5.10 rows=147 width=0) (actual time=0.030..0.031 rows=169 loops=1)"
"   Index Cond: (fk_pessoa_usuario = 2291)"
"   -> Index Scan using parcelamento_pkey on parcelamento (cost=0.43..8.45 rows=1 width=12) (actual time=0.005..0.005 rows=1 loops=13)"
"   Index Cond: (fk_despesa = despesa.id)"
"Planning time: 6.036 ms"
"Execution time: 0.391 ms"

Explain:



Testes de desempenho:

Planning time(ms)	6.036	0.307	0.322	0.321	0.307	0.311	0.338
Execution time (ms)	0.391	0.364	0.311	0.337	0.324	0.339	0.324

Tirando o maior tempo e o menor tempo, temos como média:
Planning time: 0.320 ms
Execution time: 0.338 ms

Apesar de tecnicamente a complexidade do algoritmo de hash ser O(1) e o B-tree O(log n), percebemos que não houve redução significativa de tempo comparado ao B-tree, isto porque, apesar da complexidade do algoritmo ser O(1), é utilizado a função para cálculo de hash.

Utilizando índice com duas colunas

Se por acaso, quisermos pegar as despesas parceladas de uma pessoa, mas por um caminho diferente. Ou seja, ao invés de pegar as despesas parceladas por pessoa_usuario -> despesa -> parcelamento ;
Pegamos então, por pessoa_usuario -> pessoa_cartao -> cartao -> parcelamento -> despesa (ou vice-versa).
Temos então a seguinte consulta:

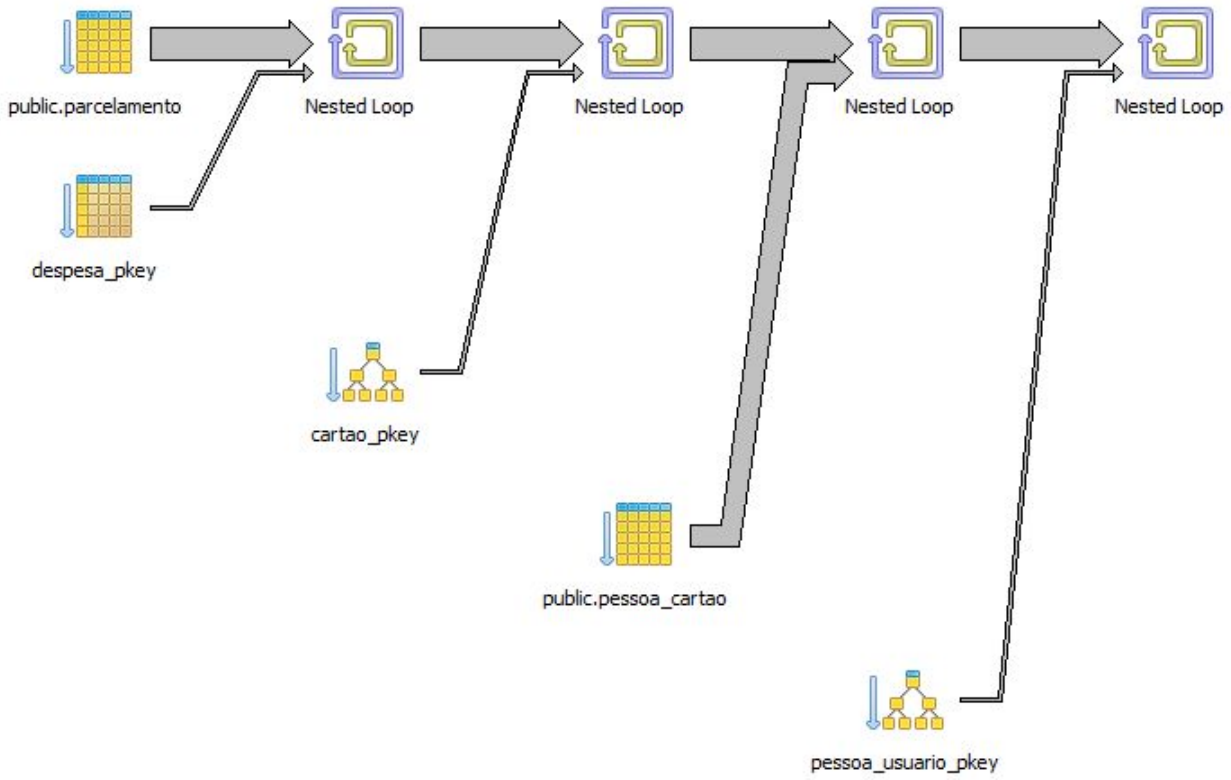
```
SELECT d.* FROM despesa d
JOIN parcelamento p ON (p.fk_despesa = d.id)
JOIN cartao c ON (c.id = p.fk_cartao)
JOIN pessoa_cartao pc ON (c.id = pc.fk_cartao)
JOIN pessoa_usuario pu ON (pu.id = pc.fk_pessoa_usuario)
WHERE pc.fk_pessoa_usuario = 2291 AND pc.fk_cartao = 19247  AND d.data_compra > '2018-11-01' AND
d.data_compra < '2018-11-30';
```

Testes sem índice:
Data output:

```
Nested Loop  (cost=1.13..22551.91 rows=1 width=47) (actual time=91.409..187.305 rows=1 loops=1)
  Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa,
d.fk_forma_pag
  ->  Nested Loop  (cost=0.85..22543.60 rows=1 width=51) (actual time=91.399..187.294 rows=1 loops=1)
        Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa,
d.fk_forma_pag, pc.fk_pessoa_usuario
        ->  Nested Loop  (cost=0.85..21571.59 rows=1 width=51) (actual time=87.018..182.752 rows=1 loops=1)
              Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa,
d.fk_forma_pag, c.id
              ->  Nested Loop  (cost=0.43..21563.14 rows=1 width=51) (actual time=87.009..182.742 rows=1
loops=1)
                    Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa,
d.fk_forma_pag, p.fk_cartao
                    ->  Seq Scan on public.parcelamento p  (cost=0.00..21487.00 rows=9 width=8) (actual
time=6.646..182.611 rows=8 loops=1)
                          Output: p.num_parcelas, p.fk_despesa, p.fk_cartao
                          Filter: (p.fk_cartao = 19247)
                          Rows Removed by Filter: 1199992
                    ->  Index Scan using despesa_pkey on public.despesa d  (cost=0.43..8.45 rows=1 width=47) (actual
time=0.012..0.012 rows=0 loops=8)
                          Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario,
d.fk_categoria_despesa, d.fk_forma_pag
                          Index Cond: (d.id = p.fk_despesa)
                          Filter: ((d.data_compra > '2018-11-01'::date) AND (d.data_compra < '2018-11-30'::date))
                          Rows Removed by Filter: 1
              ->  Index Only Scan using cartao_pkey on public.cartao c  (cost=0.42..8.44 rows=1 width=4) (actual
time=0.008..0.008 rows=1 loops=1)
                    Output: c.id
                    Index Cond: (c.id = 19247)
```

Heap Fetches: 1
-> Seq Scan on public.pessoa_cartao pc (cost=0.00..972.00 rows=1 width=8) (actual time=4.380..4.540 rows=1 loops=1)
Output: pc.fk_pessoa_usuario, pc.fk_cartao
Filter: ((pc.fk_cartao = 19247) AND (pc.fk_pessoa_usuario = 2291))
Rows Removed by Filter: 49999
-> Index Only Scan using pessoa_usuario_pkey on public.pessoa_usuario pu (cost=0.29..8.30 rows=1 width=4) (actual time=0.008..0.008 rows=1 loops=1)
Output: pu.id
Index Cond: (pu.id = 2291)
Heap Fetches: 1
Planning time: 0.840 ms
Execution time: 187.365 ms

Explain:



Testes de desempenho:

Planning time(ms)	0.840	0.470	0.469	0.731	0.460	0.499	0.464
Execution time (ms)	187.365	186.432	187.379	190.532	179.485	180.346	180.609

Tirando o maior tempo e o menor tempo, temos como média:
Planning time: 0.527 ms
Execution time: 184.426 ms

Criando o índice:

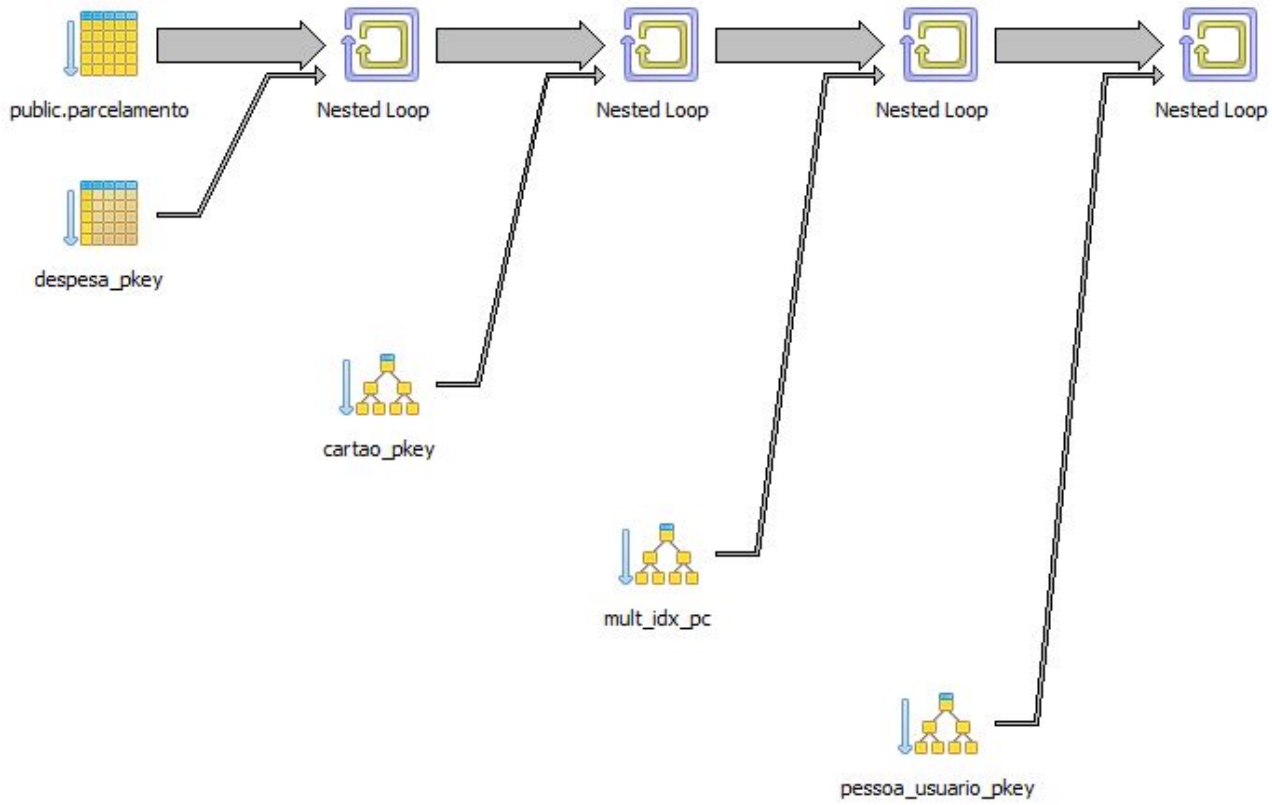
CREATE INDEX mult_idx_pc ON pessoa_cartao (fk_cartao, fk_pessoa_usuario);

Executando:
Query returned successfully with no result in 54 msec.

Executando a consulta com o índice:
Data output:
"Nested Loop (cost=1.42..21588.22 rows=1 width=47) (actual time=277.121..373.010 rows=1 loops=1)"
" Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag"
" -> Nested Loop (cost=1.14..21579.91 rows=1 width=51) (actual time=277.110..372.997 rows=1 loops=1)"
"   Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag, pc.fk_pessoa_usuario"
"   -> Nested Loop (cost=0.85..21571.59 rows=1 width=51) (actual time=276.608..372.494 rows=1 loops=1)"
"   Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag, c.id"
"   -> Nested Loop (cost=0.43..21563.14 rows=1 width=51) (actual time=276.594..372.478 rows=1 loops=1)"
"   Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag, p.fk_cartao"
"   -> Seq Scan on public.parcelamento p (cost=0.00..21487.00 rows=9 width=8) (actual time=23.843..372.286 rows=8 loops=1)"
"   Output: p.num_parcelas, p.fk_despesa, p.fk_cartao"
"   Filter: (p.fk_cartao = 19247)"
"   Rows Removed by Filter: 1199992"
"   -> Index Scan using despesa_pkey on public.despesa d (cost=0.43..8.45 rows=1 width=47) (actual time=0.018..0.018 rows=0 loops=8)"
"   Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag"
"   Index Cond: (d.id = p.fk_despesa)"
"   Filter: ((d.data_compra > '2018-11-01'::date) AND (d.data_compra < '2018-11-30'::date))"
"   Rows Removed by Filter: 1"
"   -> Index Only Scan using cartao_pkey on public.cartao c (cost=0.42..8.44 rows=1 width=4) (actual time=0.012..0.013 rows=1 loops=1)"
"   Output: c.id"
"   Index Cond: (c.id = 19247)"
"   Heap Fetches: 1"
"   -> Index Only Scan using mult_idx_pc on public.pessoa_cartao pc (cost=0.29..8.31 rows=1 width=8) (actual time=0.500..0.500 rows=1 loops=1)"
"   Output: pc.fk_cartao, pc.fk_pessoa_usuario"
"   Index Cond: ((pc.fk_cartao = 19247) AND (pc.fk_pessoa_usuario = 2291))"
"   Heap Fetches: 1"
" -> Index Only Scan using pessoa_usuario_pkey on public.pessoa_usuario pu (cost=0.29..8.30 rows=1 width=4) (actual time=0.008..0.010 rows=1 loops=1)"

" Output: pu.id"
" Index Cond: (pu.id = 2291)"
" Heap Fetches: 1"
"Planning time: 3.549 ms"
"Execution time: 373.130 ms"

Explain:



Testes de desempenho:

Planning time(ms)	3.549	1.596	1.686	1.318	1.322	1.324	1.332
Execution time (ms)	373.130	360.373	383.483	377.022	406.688	236.815	266.099

Tirando o maior tempo e o menor tempo, temos como média:
Planning time: 1.452 ms
Execution time: 353.021 ms

Apesar da utilização do índice, como visto no explain, o resultado final demonstra uma horrível piora na execução do tempo, sendo quase o dobro tempo. Mas isso não quer dizer que ele não possa ser útil, no final

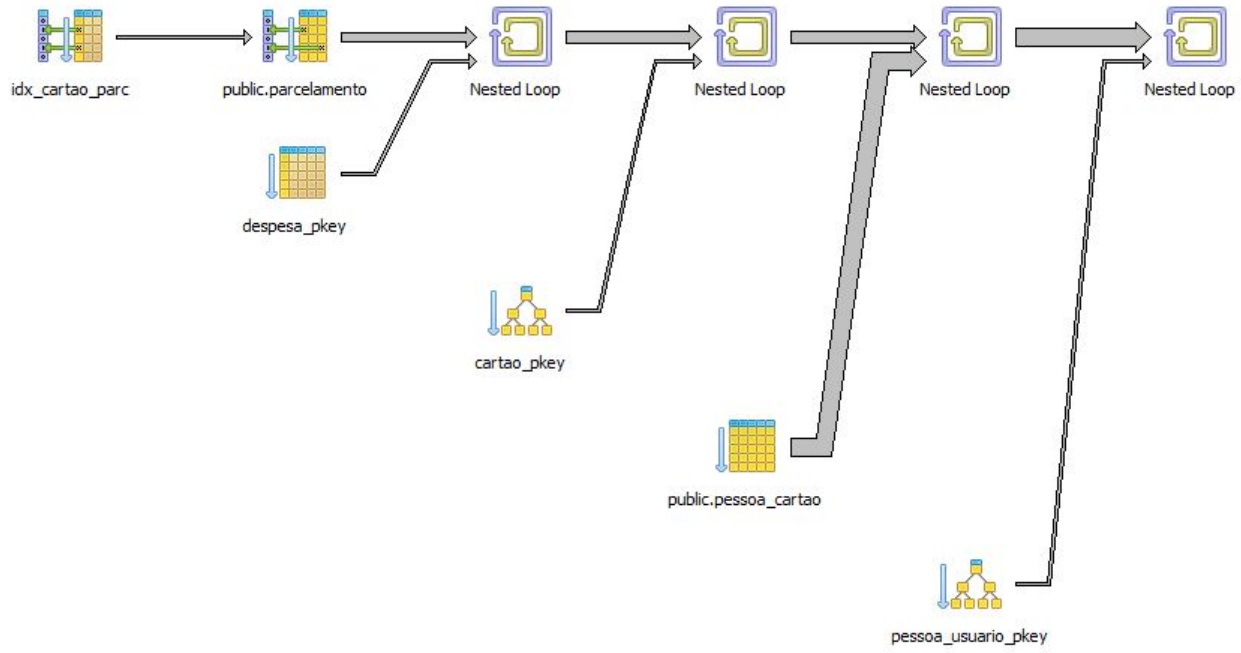
das contas o banco terá vários índices, e em uma determinada consulta, pode acontecer de haver mais de um índice.
Por exemplo, logo no ínicio deste documento apresentamos um índice na chave estrangeira do cartão na tabela parcelamento.
Para iniciar vamos colocar novamente este índice, só que fazendo esta nova consulta (com o id_cartao e id_pessoa), mas sem o índice com múltiplas colunas por enquanto:

Testes sem índice multicolumn:
CREATE INDEX idx_cartao_parc ON parcelamento (fk_cartao)
Query returned successfully with no result in 1.5 secs.

Data output:
"Nested Loop (cost=5.63..1104.51 rows=1 width=47) (actual time=4.536..4.743 rows=1 loops=1)"
" Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag"
" -> Nested Loop (cost=5.34..1096.20 rows=1 width=51) (actual time=4.528..4.734 rows=1 loops=1)"
"   Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag, pc.fk_pessoa_usuario"
"   -> Nested Loop (cost=5.34..124.19 rows=1 width=51) (actual time=0.131..0.171 rows=1 loops=1)"
"   Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag, c.id"
"   -> Nested Loop (cost=4.92..115.74 rows=1 width=51) (actual time=0.123..0.163 rows=1 loops=1)"
"   Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag, p.fk_cartao"
"   -> Bitmap Heap Scan on public.parcelamento p (cost=4.50..39.60 rows=9 width=8) (actual time=0.098..0.105 rows=8 loops=1)"
"   Output: p.num_parcelas, p.fk_despesa, p.fk_cartao"
"   Recheck Cond: (p.fk_cartao = 19247)"
"   Heap Blocks: exact=8"
"   -> Bitmap Index Scan on idx_cartao_parc (cost=0.00..4.50 rows=9 width=0) (actual time=0.093..0.093 rows=8 loops=1)"
"   Index Cond: (p.fk_cartao = 19247)"
"   -> Index Scan using despesa_pkey on public.despesa d (cost=0.43..8.45 rows=1 width=47) (actual time=0.007..0.007 rows=0 loops=8)"
"   Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa, d.fk_forma_pag"
"   Index Cond: (d.id = p.fk_despesa)"
"   Filter: ((d.data_compra > '2018-11-01'::date) AND (d.data_compra < '2018-11-30'::date))"
"   Rows Removed by Filter: 1"
"   -> Index Only Scan using cartao_pkey on public.cartao c (cost=0.42..8.44 rows=1 width=4) (actual time=0.008..0.008 rows=1 loops=1)"
"   Output: c.id"
"   Index Cond: (c.id = 19247)"
"   Heap Fetches: 1"
"   -> Seq Scan on public.pessoa_cartao pc (cost=0.00..972.00 rows=1 width=8) (actual time=4.396..4.561 rows=1 loops=1)"
"   Output: pc.fk_pessoa_usuario, pc.fk_cartao"
"   Filter: ((pc.fk_cartao = 19247) AND (pc.fk_pessoa_usuario = 2291))"
"   Rows Removed by Filter: 49999"

" -> Index Only Scan using pessoa_usuario_pkey on public.pessoa_usuario pu (cost=0.29..8.30 rows=1 width=4) (actual time=0.007..0.008 rows=1 loops=1)"
"   Output: pu.id"
"   Index Cond: (pu.id = 2291)"
"   Heap Fetches: 1"
"Planning time: 1.434 ms"
"Execution time: 4.832 ms"

Explain:



Testes de desempenho:

Planning time(ms)	1.434	0.519	0.505	0.511	0.494	0.498	0.494
Execution time (ms)	4.832	4.760	5.830	4.773	4.699	4.777	5.095

Tirando o maior tempo e o menor tempo, temos como média:
Planning time: 0.505 ms
Execution time: 4.847 ms

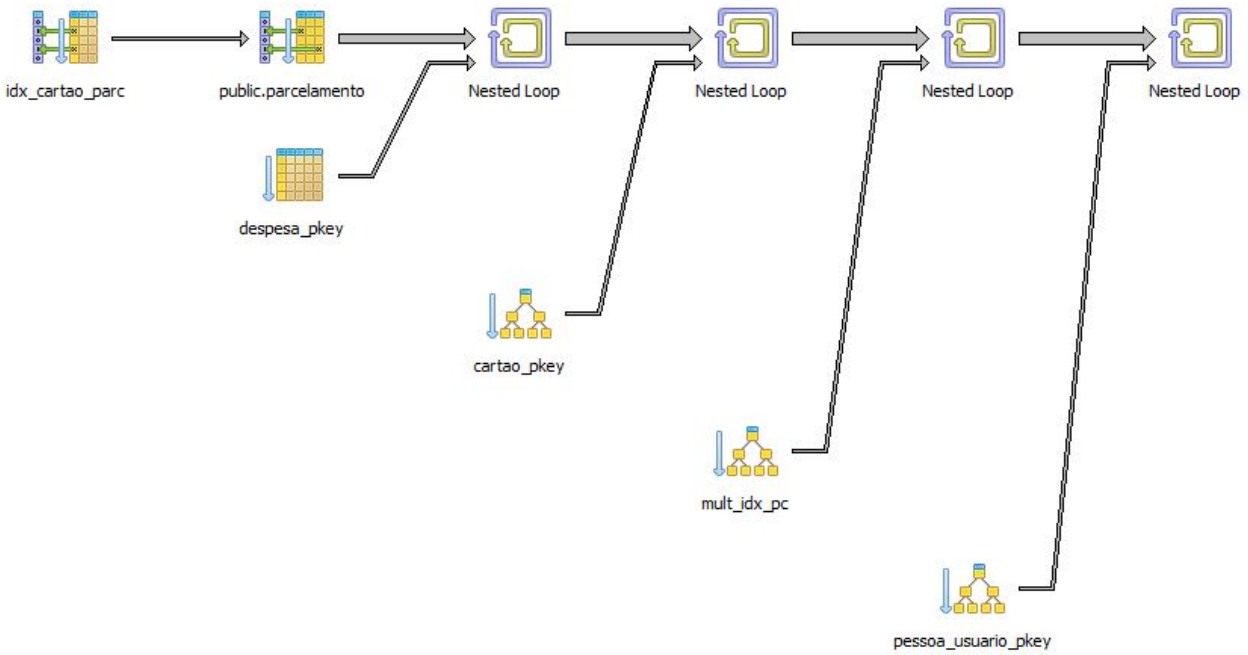
Agora faremos o teste com o índice de duas coluna, mantendo o índice da tabela parcelamento:
CREATE INDEX mult_idx_pc ON pessoa_cartao (fk_cartao, fk_pessoa_usuario);

Query returned successfully with no result in 53 msec.

Data output:

```
"Nested Loop  (cost=5.92..140.82 rows=1 width=47) (actual time=0.324..0.355 rows=1 loops=1)"
"  Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa,
d.fk_forma_pag"
"  ->  Nested Loop  (cost=5.63..132.51 rows=1 width=51) (actual time=0.317..0.348 rows=1 loops=1)"
"        Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa,
d.fk_forma_pag, pc.fk_pessoa_usuario"
"        ->  Nested Loop  (cost=5.34..124.19 rows=1 width=51) (actual time=0.042..0.072 rows=1 loops=1)"
"              Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario, d.fk_categoria_despesa,
d.fk_forma_pag, c.id"
"              ->  Nested Loop  (cost=4.92..115.74 rows=1 width=51) (actual time=0.034..0.064 rows=1 loops=1)"
"                    Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario,
d.fk_categoria_despesa, d.fk_forma_pag, p.fk_cartao"
"                    ->  Bitmap Heap Scan on public.parcelamento p  (cost=4.50..39.60 rows=9 width=8) (actual
time=0.014..0.021 rows=8 loops=1)"
"                          Output: p.num_parcelas, p.fk_despesa, p.fk_cartao"
"                          Recheck Cond: (p.fk_cartao = 19247)"
"                          Heap Blocks: exact=8"
"                          ->  Bitmap Index Scan on idx_cartao_parc  (cost=0.00..4.50 rows=9 width=0) (actual
time=0.009..0.009 rows=8 loops=1)"
"                                Index Cond: (p.fk_cartao = 19247)"
"                    ->  Index Scan using despesa_pkey on public.despesa d  (cost=0.43..8.45 rows=1 width=47)
(actual time=0.005..0.005 rows=0 loops=8)"
"                          Output: d.valor, d.data_compra, d.fixo, d.id, d.nome, d.fk_pessoa_usuario,
d.fk_categoria_despesa, d.fk_forma_pag"
"                          Index Cond: (d.id = p.fk_despesa)"
"                          Filter: ((d.data_compra > '2018-11-01'::date) AND (d.data_compra < '2018-11-30'::date))"
"                          Rows Removed by Filter: 1"
"              ->  Index Only Scan using cartao_pkey on public.cartao c  (cost=0.42..8.44 rows=1 width=4) (actual
time=0.007..0.007 rows=1 loops=1)"
"                    Output: c.id"
"                    Index Cond: (c.id = 19247)"
"                    Heap Fetches: 1"
"        ->  Index Only Scan using mult_idx_pc on public.pessoa_cartao pc  (cost=0.29..8.31 rows=1 width=8)
(actual time=0.274..0.274 rows=1 loops=1)"
"              Output: pc.fk_cartao, pc.fk_pessoa_usuario"
"              Index Cond: ((pc.fk_cartao = 19247) AND (pc.fk_pessoa_usuario = 2291))"
"              Heap Fetches: 1"
"  ->  Index Only Scan using pessoa_usuario_pkey on public.pessoa_usuario pu  (cost=0.29..8.30 rows=1
width=4) (actual time=0.007..0.007 rows=1 loops=1)"
"        Output: pu.id"
"        Index Cond: (pu.id = 2291)"
"        Heap Fetches: 1"
"Planning time: 1.215 ms"
"Execution time: 0.461 ms"
```

Explain:



Testes de desempenho:

Planning time(ms)	1.215	0.551	0.529	0.504	1.103	0.770	0.829
Execution time (ms)	0.461	0.265	0.245	0.217	0.215	0.212	0.248

Tirando o maior tempo e o menor tempo, temos como média:
Planning time: 0.756 ms
Execution time: 0.238 ms

Neste teste houve leve aumento na tempo de planejamento, mas percebemos um queda de 95% no tempo execução, o que podemos concluir que vale a pena usar o índice mult_idx_pc, desde que use o índice idx_cartao_parc. Apesar deste índice multicolumn reduzir bastante combinado ao outro índice em parcelamento, uma simples criação de índice para a chave estrangeira do cartao na tabela pessoa_carto, resolveria o problema, mas somente no caso desta consulta. O interessante de usar neste caso, é que tanto para pessoa quanto para cartao, este índice servirá.

Finalmente chega a parte para testes de índices sobre receitas, e não mais despesas:
Para termos pelo menos dois joins, criaremos uma consulta que tenha categoria de receita:

Consulta de receitas de uma pessoa, e um determinado mês e de uma determinada categoria:

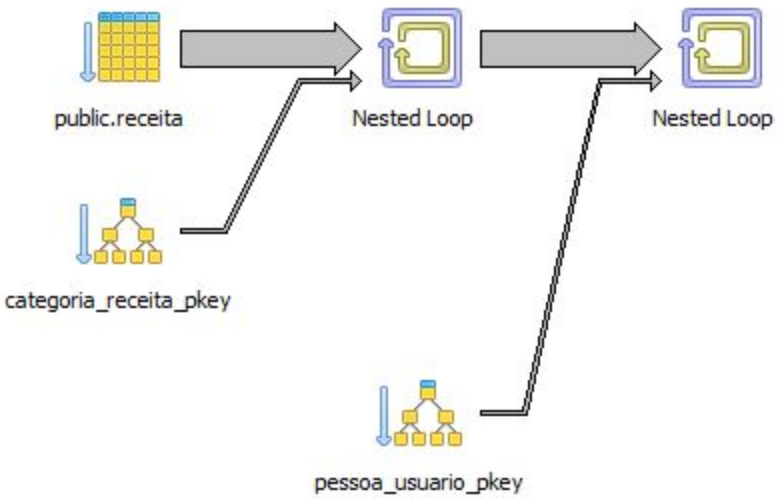
```
SELECT r.* FROM receita r
JOIN categoria_receita cr ON (cr.id = r.fk_categoria_receita)
JOIN pessoa_usuario pu ON (pu.id = r.fk_pessoa_usuario)
WHERE pu.id = 9242 AND cr.id = 2 AND r.data_recebimento > '2018-11-01' AND r.data_recebimento <
'2018-11-30';
```

Testes sem índice:

Data output:

```
"Nested Loop  (cost=0.44..43852.49 rows=1 width=43) (actual time=29.594..330.079 rows=3 loops=1)"
"  Output: r.id, r.valor, r.data_recebimento, r.fixo, r.nome, r.fk_pessoa_usuario, r.fk_categoria_receita"
"  ->  Nested Loop  (cost=0.15..43844.18 rows=1 width=43) (actual time=29.588..330.057 rows=3 loops=1)"
"        Output: r.id, r.valor, r.data_recebimento, r.fixo, r.nome, r.fk_pessoa_usuario, r.fk_categoria_receita"
"        ->  Seq Scan on public.receita r  (cost=0.00..43836.00 rows=1 width=43) (actual time=29.577..330.025
rows=3 loops=1)"
"              Output: r.id, r.valor, r.data_recebimento, r.fixo, r.nome, r.fk_pessoa_usuario, r.fk_categoria_receita"
"              Filter: ((r.data_recebimento > '2018-11-01'::date) AND (r.data_recebimento < '2018-11-30'::date) AND
(r.fk_categoria_receita = 2) AND (r.fk_pessoa_usuario = 9242))"
"              Rows Removed by Filter: 1499997"
"        ->  Index Only Scan using categoria_receita_pkey on public.categoria_receita cr  (cost=0.15..8.17 rows=1
width=4) (actual time=0.007..0.007 rows=1 loops=3)"
"              Output: cr.id"
"              Index Cond: (cr.id = 2)"
"              Heap Fetches: 3"
"  ->  Index Only Scan using pessoa_usuario_pkey on public.pessoa_usuario pu  (cost=0.29..8.30 rows=1
width=4) (actual time=0.005..0.006 rows=1 loops=3)"
"        Output: pu.id"
"        Index Cond: (pu.id = 9242)"
"        Heap Fetches: 3"
"Planning time: 0.444 ms"
"Execution time: 330.170 ms"
```

Explain:



Testes de desempenho:

Planning time(ms)	0.444	0.263	0.438	0.257	0.203	0.458	0.438

Execution time (ms)	330.170	330.071	336.832	336.432	326.706	327.613	333.585

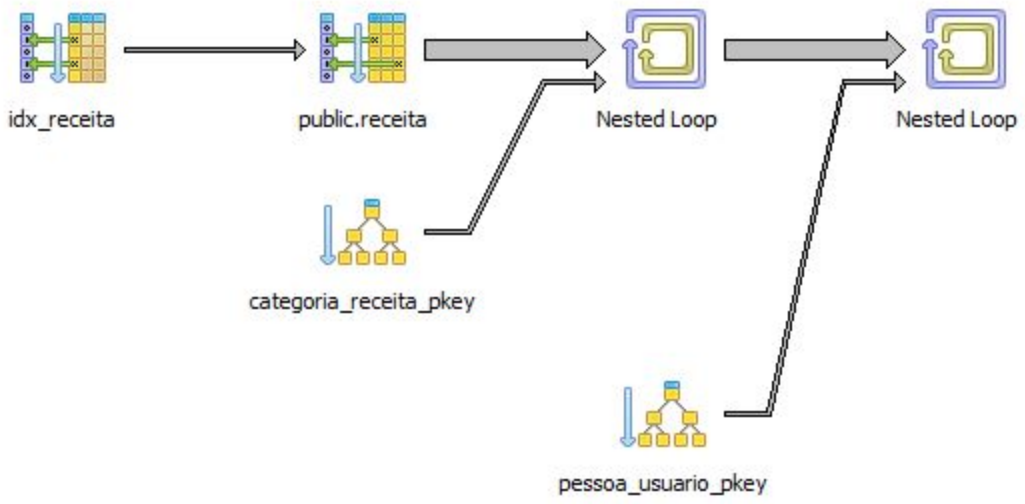
Tirando o maior tempo e o menor tempo, temos como média:
Planning time: 0.368 ms
Execution time: 331.574 ms

Testes com índice:
Criando índice:
CREATE INDEX idx_receita ON receita(fk_pessoa_usuario);

Executando:
Query returned successfully with no result in 2.0 secs.

Data output:
"Nested Loop (cost=5.97..567.50 rows=1 width=43) (actual time=0.674..6.039 rows=3 loops=1)"
" Output: r.id, r.valor, r.data_recebimento, r.fixo, r.nome, r.fk_pessoa_usuario, r.fk_categoria_receita"
" -> Nested Loop (cost=5.68..559.19 rows=1 width=43) (actual time=0.668..6.027 rows=3 loops=1)"
"   Output: r.id, r.valor, r.data_recebimento, r.fixo, r.nome, r.fk_pessoa_usuario, r.fk_categoria_receita"
"   -> Bitmap Heap Scan on public.receita r (cost=5.53..551.01 rows=1 width=43) (actual time=0.660..6.014 rows=3 loops=1)"
"   Output: r.id, r.valor, r.data_recebimento, r.fixo, r.nome, r.fk_pessoa_usuario, r.fk_categoria_receita"
"   Recheck Cond: (r.fk_pessoa_usuario = 9242)"
"   Filter: ((r.data_recebimento > '2018-11-01'::date) AND (r.data_recebimento < '2018-11-30'::date) AND (r.fk_categoria_receita = 2))"
"   Rows Removed by Filter: 188"
"   Heap Blocks: exact=191"
"   -> Bitmap Index Scan on idx_receita (cost=0.00..5.53 rows=147 width=0) (actual time=0.116..0.116 rows=191 loops=1)"
"   Index Cond: (r.fk_pessoa_usuario = 9242)"
"   -> Index Only Scan using categoria_receita_pkey on public.categoria_receita cr (cost=0.15..8.17 rows=1 width=4) (actual time=0.003..0.003 rows=1 loops=3)"
"   Output: cr.id"
"   Index Cond: (cr.id = 2)"
"   Heap Fetches: 3"
" -> Index Only Scan using pessoa_usuario_pkey on public.pessoa_usuario pu (cost=0.29..8.30 rows=1 width=4) (actual time=0.003..0.004 rows=1 loops=3)"
"   Output: pu.id"
"   Index Cond: (pu.id = 9242)"
"   Heap Fetches: 3"
"Planning time: 0.921 ms"
"Execution time: 6.097 ms"

Explain



Testes de desempenho:

Planning time(ms)	0.921	0.498	0.222	0.454	0.486	0.493	0.227
Execution time (ms)	6.097	0.398	0.348	0.359	0.384	0.388	0.346

Tirando o maior tempo e o menor tempo, temos como média:
Planning time: 0.432 ms
Execution time: 0.375 ms