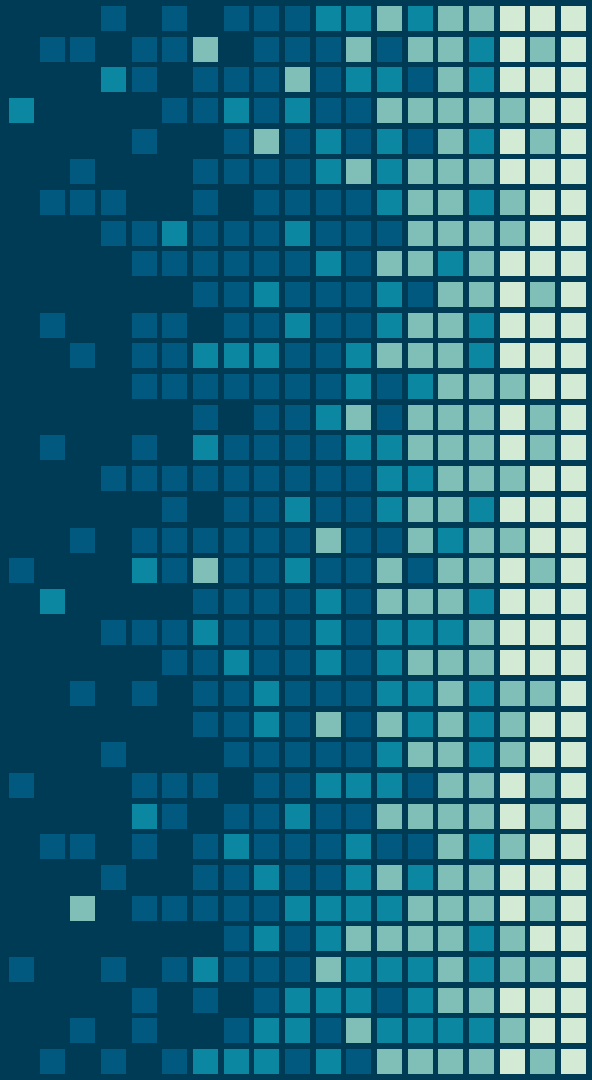
Otimização de Consultas

Postgres X SQL server

Juliana Isabel de Freitas Rosin
Vinícius Marcelo Rifam Laurindo



Sumário

DATASET

Financeiro -

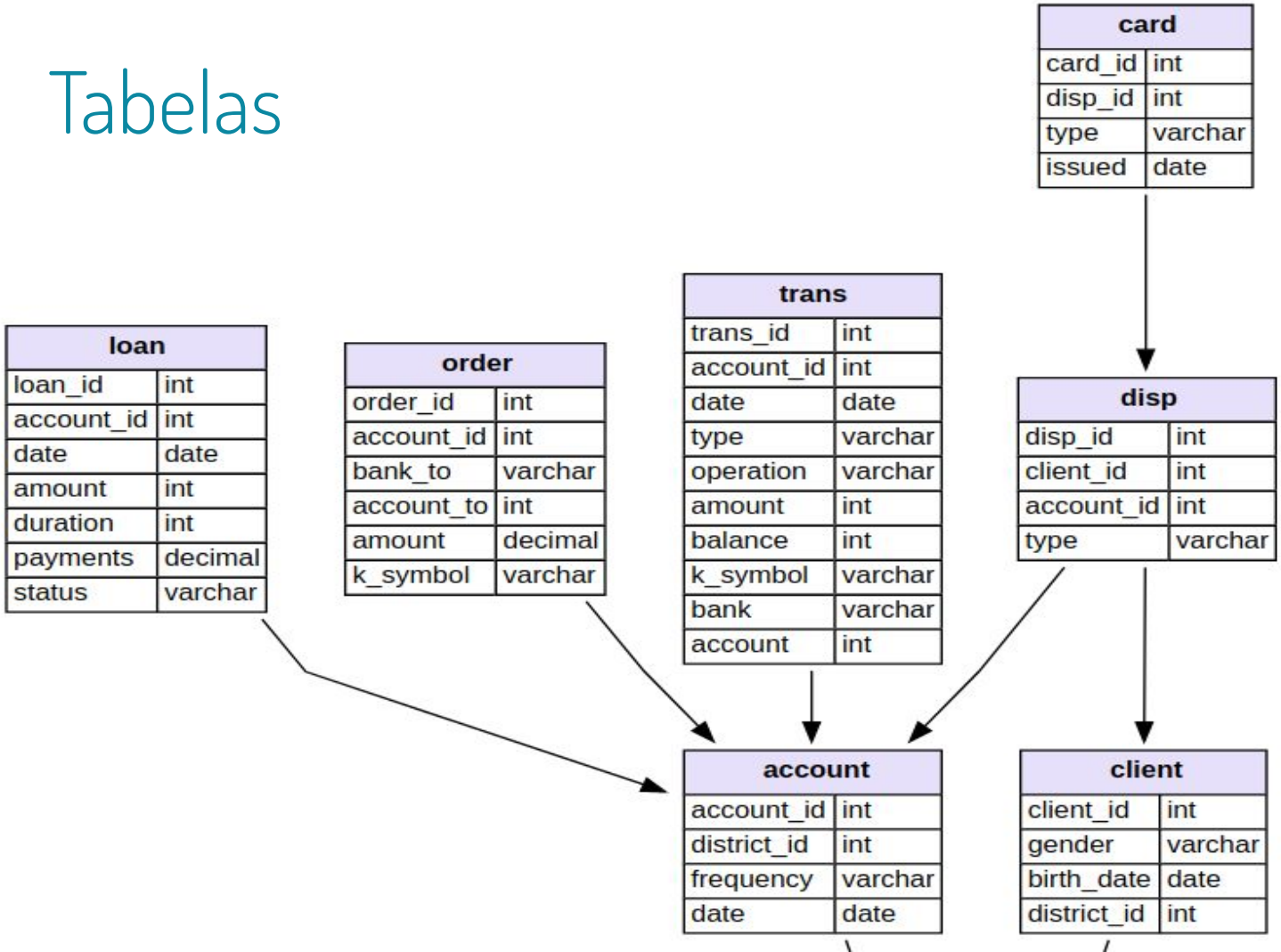
Pedidos de empréstimos

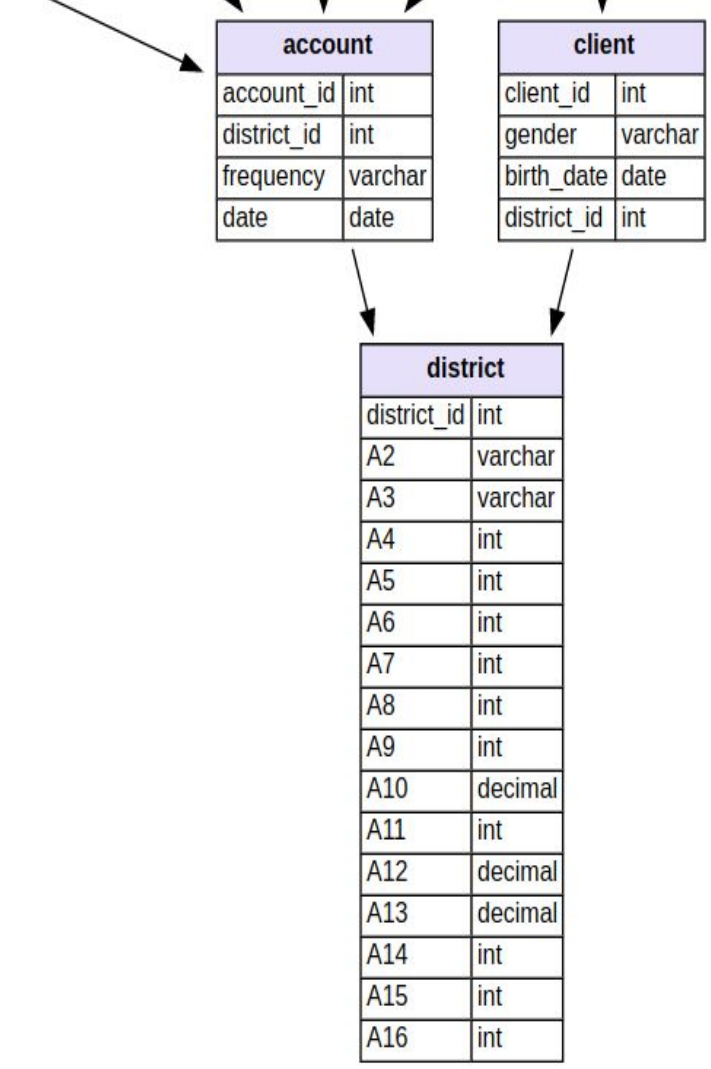
O conjunto de dados financeiros contém 606 empréstimos bem-sucedidos e 76 sem êxito, juntamente com suas informações e transações.

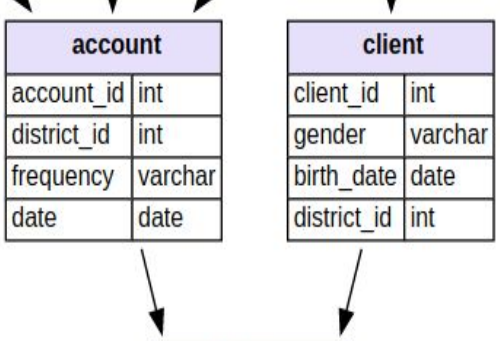
Disponível em:

https://bit.ly/3101Bhp

Tabelas


card
card_id int
disp_id int
type varchar
issued date
trans
trans_id int
account_id int
date date
type varchar
operation varchar
amount int
balance int
k_symbol varchar
bank varchar
account int
loan
loan_id int
account_id int
date date
amount int
duration int
payments decimal
status varchar
order
order_id int
account_id int
bank_to varchar
account_to int
amount decimal
k_symbol varchar
disp
disp_id int
client_id int
account_id int
type varchar
account
account_id int
district_id int
frequency varchar
date date
client
client_id int
gender varchar
birth_date date
district_id int


account
account_id int
district_id int
frequency varchar
date date
client
client_id int
gender varchar
birth_date date
district_id int
district
district_id int
A2 varchar
A3 varchar
A4 int
A5 int
A6 int
A7 int
A8 int
A9 int
A10 decimal
A11 int
A12 decimal
A13 decimal
A14 int
A15 int
A16 int



district	
district_id	int
A2	varchar
A3	varchar
A4	int
A5	int
A6	int
A7	int
A8	int
A9	int
A10	decimal
A11	int
A12	decimal
A13	decimal
A14	int
A15	int
A16	int

- Card - 892 linhas
- Disp - 5.369 linhas
- Order - 6.471 linhas
- Loan - 682 linhas
- Account - 4.500 linhas
- Client - 5.369 linhas
- Trans - **1.056.320** linhas
- District - 77 linhas

Planos de Consultas

Plano de consulta 1

Retornar o id da ordem, o montante, a operação, o id da conta e a data, onde o montante é maior que 2400.

Plano de consulta 1

Retornar o id da ordem, o montante, a operação, o id da conta e a data, onde o montante é maior que 2400.

order_id	amount	operation	account_id	date
29401	2452.0	PREVOD NA UCET	1	1995-09-05
29401	2452.0	PREVOD NA UCET	1	1995-10-05
29401	2452.0	PREVOD NA UCET	1	1995-11-05

Plano de consulta 1

Não otimizada

```
SELECT o.order_id, o.amount,
tb.operation,tb.account_id, tb.date
FROM _order o JOIN (SELECT
t.operation,t.account_id, t.date,
t.amount FROM _trans t WHERE
t.amount >= 2400) tb on
o.amount=tb.amount;
```

Otimizada

```
SELECT o.order_id, o.amount,
t.operation, t.account_id, t.date
FROM _trans t, _order o WHERE
t.amount >= 2400 and o.amount =
t.amount;
```

POSTGRES

Execução das consultas

Plano de consulta 1

NÃO OTIMIZADA

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=192.60..42963.00 rows=733301 width=32) (actual time=5.682..274.346 rows=259977 loops=1)
   Hash Cond: ((t.amount)::numeric = o.amount)
   ->  Seq Scan on _trans t  (cost=0.00..24188.00 rows=499973 width=27) (actual time=0.084..125.472 rows=502801 loops=1)
         Filter: (amount >= 2400)
         Rows Removed by Filter: 553519
   ->  Hash  (cost=111.71..111.71 rows=6471 width=9) (actual time=5.537..5.537 rows=6471 loops=1)
         Buckets: 8192  Batches: 1  Memory Usage: 325kB
         ->  Seq Scan on _order o  (cost=0.00..111.71 rows=6471 width=9) (actual time=0.039..2.371 rows=6471 loops=1)
Planning time: 0.198 ms
Execution time: 282.828 ms
(10 rows)
```

NÃO OTIMIZADA

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=192.60..42963.00 rows=733301 width=32) (actual time=5.682..274.346 rows=259977 loops=1)
   Hash Cond: ((t.amount)::numeric = o.amount)
   ->  Seq Scan on _trans t  (cost=0.00..24188.00 rows=499973 width=27) (actual time=0.084..125.472 rows=502801 loops=1)
         Filter: (amount >= 2400)
         Rows Removed by Filter: 553519
   ->  Hash  (cost=111.71..111.71 rows=6471 width=9) (actual time=5.537..5.537 rows=6471 loops=1)
         Buckets: 8192  Batches: 1  Memory Usage: 325kB
         ->  Seq Scan on _order o  (cost=0.00..111.71 rows=6471 width=9) (actual time=0.039..2.371 rows=6471 loops=1)
 Planning time: 0.198 ms
 Execution time: 282.828 ms
(10 rows)
```

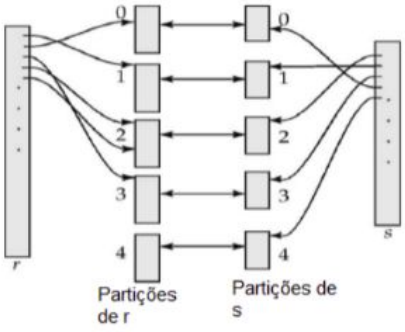
OTIMIZADA

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=192.60..42963.00 rows=733301 width=32) (actual time=3.628..277.347 rows=259977 loops=1)
   Hash Cond: ((t.amount)::numeric = o.amount)
   ->  Seq Scan on _trans t  (cost=0.00..24188.00 rows=499973 width=27) (actual time=0.115..129.904 rows=502801 loops=1)
         Filter: (amount >= 2400)
         Rows Removed by Filter: 553519
   ->  Hash  (cost=111.71..111.71 rows=6471 width=9) (actual time=3.455..3.455 rows=6471 loops=1)
         Buckets: 8192  Batches: 1  Memory Usage: 325kB
         ->  Seq Scan on _order o  (cost=0.00..111.71 rows=6471 width=9) (actual time=0.029..1.341 rows=6471 loops=1)
 Planning time: 0.109 ms
 Execution time: 285.474 ms
(10 rows)
```

Algoritmos

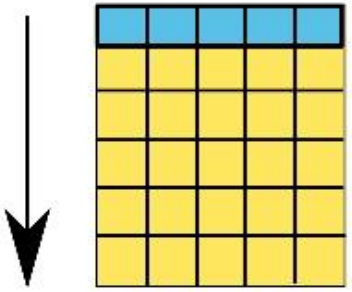
Hash Join

Carrega os registros candidatos de um lado da junção em uma tabela de hash, que é então examinada para cada registro do outro lado da junção.



Seq Scan

Varre a tabela inteira sequencialmente na forma em que ela foi armazenada no disco, descartando linhas que não corresponderem com o filtro. O resultado são linhas desordenadas.



NÃO OTIMIZADA

0,311ms

Tempo médio de Planejamento

(DP: 0,0614 ms)

278,73ms

Tempo médio Execução

(DP: 7,81 ms)

OTIMIZADA

0,224 ms

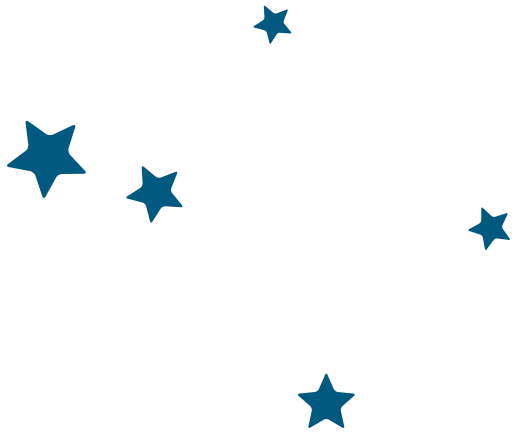
Tempo médio de Planejamento

(DP: 0,0563 ms)

260,32 ms

Tempo médio Execução

(DP: 12,17 ms)

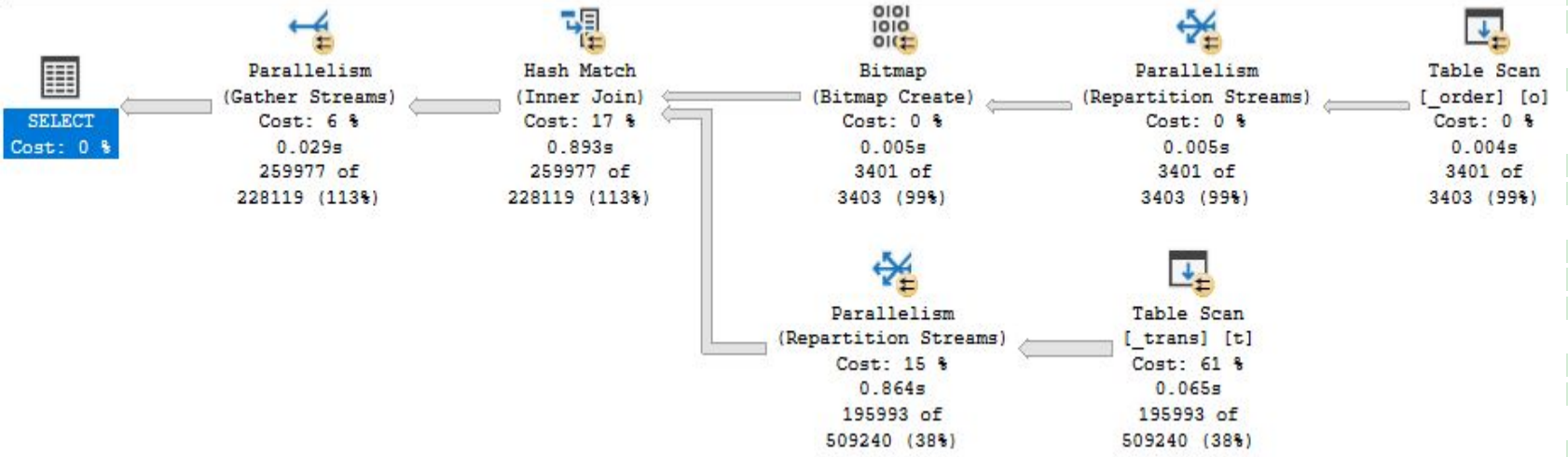


SQL Server

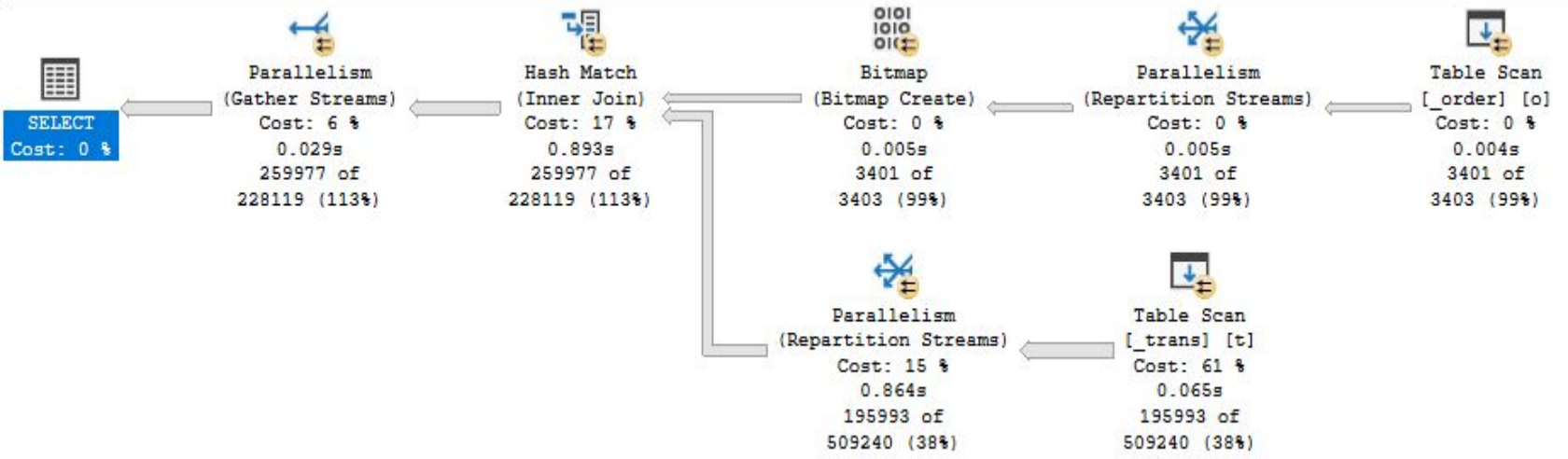
Execução das consultas

Plano de consulta 1

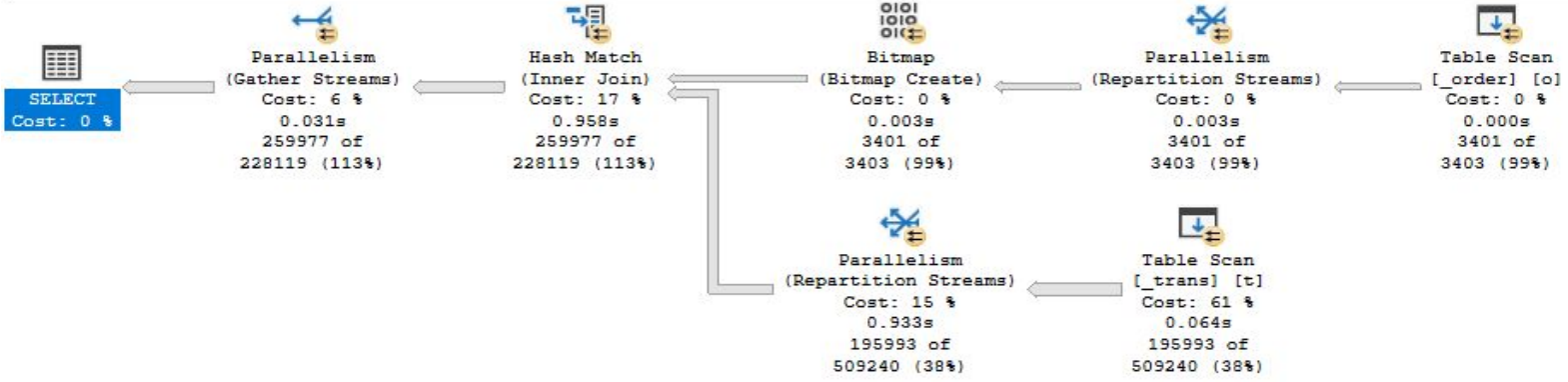
NÃO OTIMIZADA



NÃO OTIMIZADA



OTIMIZADA



Algoritmos

Hash Match

O operador Hash Match implementa várias operações lógicas diferentes que usam uma tabela hash na memória para localizar dados correspondentes.

Dois tipos: Join ou Aggregate

https://bit.ly/2Yp6z

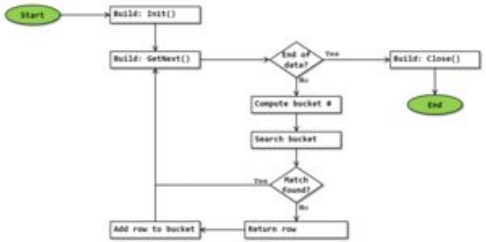
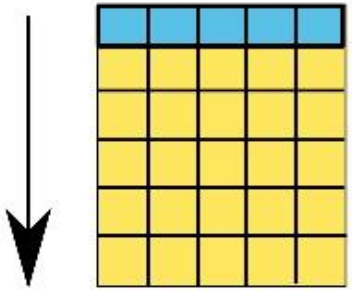


Table Scan

Busca em todos os elementos da tabela, de forma sequencial;

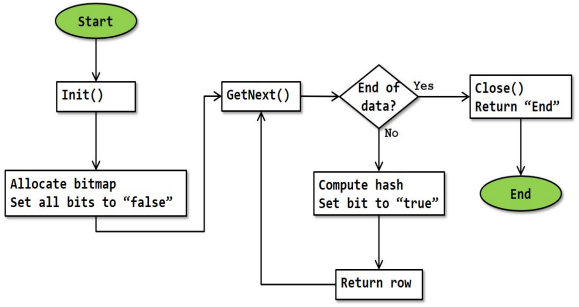


Algoritmos

Bitmap

Um bitmap é uma estrutura que armazena valores booleanos para um intervalo consecutivo de valores em uma pequena quantidade de memória.

https://bit.ly/2NhQH0h



Parallelism

Este operador altera a distribuição das linhas de (semi-) aleatório para determinístico. Processa as linhas em diferentes CPU, retornando o resultado ``correto``.

https://bit.ly/2J6Wxwd

Parallelism

Gather streams.

Physical Operation	Parallelism
Logical Operation	Gather Streams
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	259977
Actual Number of Batches	0
Estimated I/O Cost	0
Estimated Operator Cost	0,6987 (6%)
Estimated Subtree Cost	11,4084
Estimated CPU Cost	0,698656
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	228119
Estimated Row Size	41 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	0

Output List

[teste].[dbo].[_order].order_id; [teste].[dbo].[_order].amount; [teste].[dbo].[_trans].account_id; [teste].[dbo].[_trans].date; [teste].[dbo].[_trans].operation

NÃO OTIMIZADA

725,3ms

Tempo médio de CPU

(DP: 64,9 ms)

3049,4ms

Tempo médio Execução

(DP: 637,08 ms)

OTIMIZADA

650,5 ms

Tempo médio de CPU

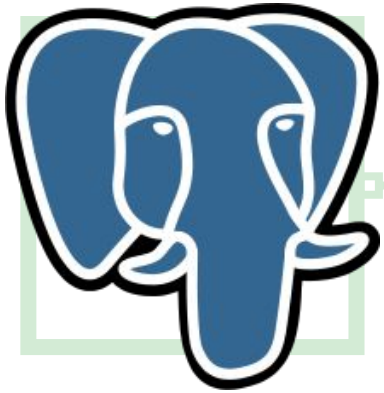
(DP: 126,18 ms)

2795,9 ms

Tempo médio Execução

(DP: 98,12 ms)

Comparação Tempo Execução



	POSTGRES	SQL SERVER
otimizada	260,327	2795,9
Não otimizada	278,733	3049,4

Plano de consulta 2

Retornar a quantidade de cartões, seus tipos e a quantidade de transações realizadas por todas as contas que estão ativas.

Plano de consulta 2

Retornar a quantidade de cartões, seus tipos e a quantidade de transações realizadas por todas as contas que estão ativas.

qtd_card	type	account_id	qtd_trans
1	gold	7	130
1	classic	14	124
1	gold	33	344

Plano de consulta 2

Não otimizada

```
SELECT COUNT(DISTINCT tc.card_id) as qtd_card, tc.type, tc.account_id, COUNT(DISTINCT tr.trans_id) as qtd_trans FROM _trans tr JOIN (SELECT ca.card_id, ca.type, tb.account_id FROM _card ca JOIN (SELECT d.account_id, d.disp_id FROM _disp d WHERE d.type = 'OWNER') as tb on ca.disp_id=tb.disp_id) tc on tc.account_id = tr.account_id GROUP BY tc.account_id, tc.type;
```

Otimizada

```
SELECT COUNT( c.card_id) as qtd_card, c.type, t.account_id, COUNT( t.trans_id) as qtd_trans FROM _disp d JOIN _card c on c.disp_id = d.disp_id and d.type = 'OWNER' JOIN _trans t on t.account_id = d.account_id GROUP BY t.account_id,c.type;
```

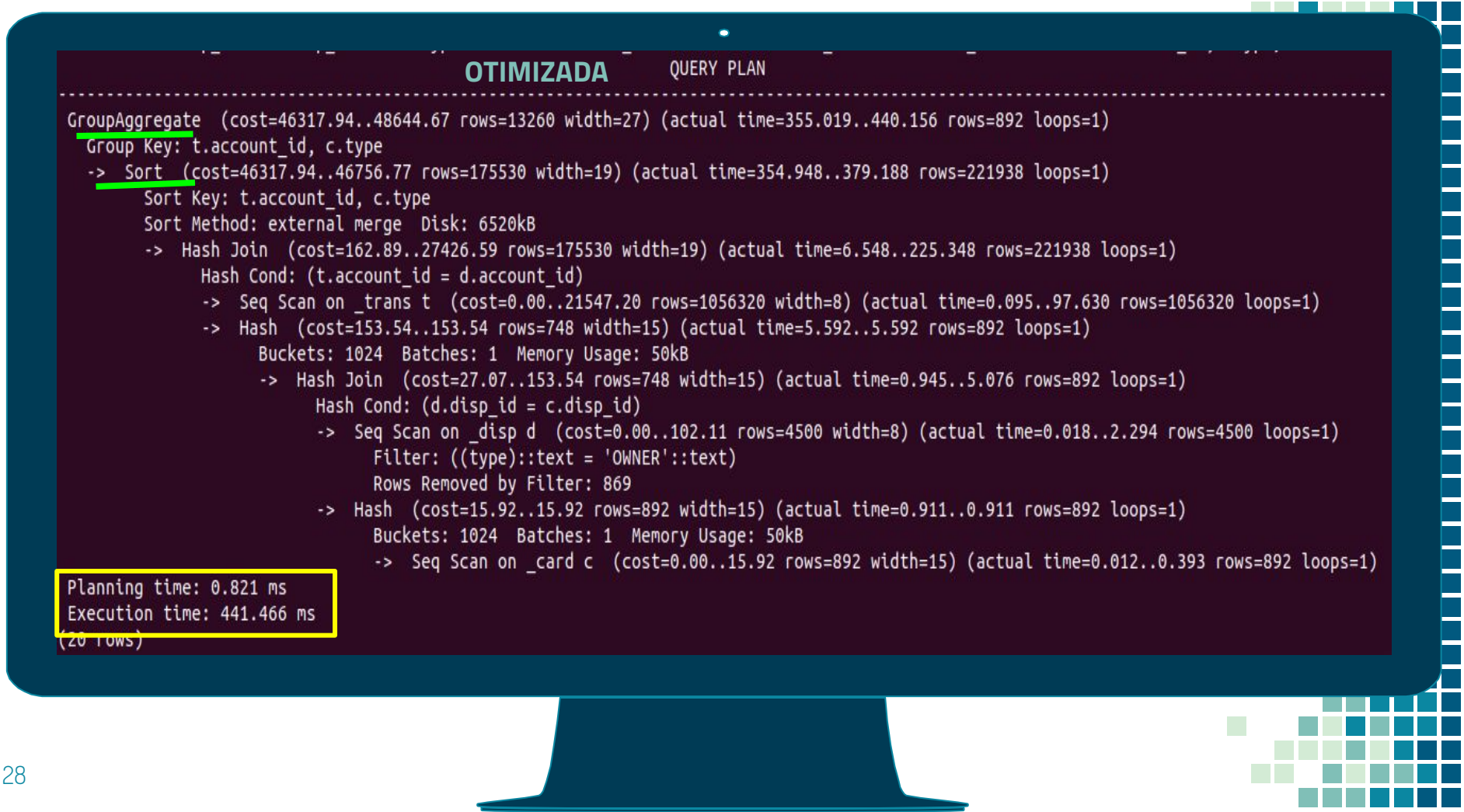
POSTGRES

Execução das consultas

Plano de consulta 2

NÃO OTIMIZADA

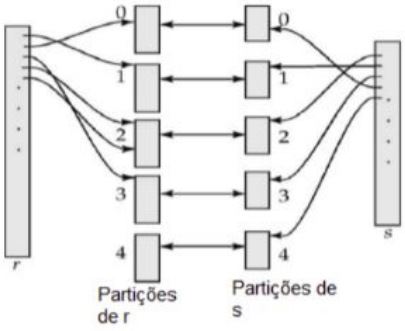
```
                                QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=46306.14..48618.81 rows=11964 width=27) (actual time=352.288..436.956 rows=892 loops=1)
   Group Key: d.account_id, ca.type
   ->  Sort  (cost=46306.14..46744.75 rows=175442 width=19) (actual time=352.218..376.289 rows=221938 loops=1)
         Sort Key: d.account_id, ca.type
         Sort Method: external merge  Disk: 6520kB
         ->  Hash Join  (cost=162.89..27426.59 rows=175442 width=19) (actual time=6.611..223.891 rows=221938 loops=1)
               Hash Cond: (tr.account_id = d.account_id)
               ->  Seq Scan on _trans tr  (cost=0.00..21547.20 rows=1056320 width=8) (actual time=0.097..98.372 rows=1056320 loops=1)
               ->  Hash  (cost=153.54..153.54 rows=748 width=15) (actual time=5.657..5.657 rows=892 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 50kB
                     ->  Hash Join  (cost=27.07..153.54 rows=748 width=15) (actual time=0.990..5.123 rows=892 loops=1)
                           Hash Cond: (d.disp_id = ca.disp_id)
                           ->  Seq Scan on _disp d  (cost=0.00..102.11 rows=4500 width=8) (actual time=0.020..2.266 rows=4500 loops=1)
                                 Filter: ((type)::text = 'OWNER'::text)
                                 Rows Removed by Filter: 869
                           ->  Hash  (cost=15.92..15.92 rows=892 width=15) (actual time=0.944..0.944 rows=892 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 50kB
                                 ->  Seq Scan on _card ca  (cost=0.00..15.92 rows=892 width=15) (actual time=0.015..0.415 rows=892 loops=1)
 Planning time: 1.404 ms
 Execution time: 438.239 ms
(20 rows)
```


OTIMIZADA
QUERY PLAN
GroupAggregate (cost=46317.94..48644.67 rows=13260 width=27) (actual time=355.019..440.156 rows=892 loops=1)
Group Key: t.account_id, c.type
-> Sort (cost=46317.94..46756.77 rows=175530 width=19) (actual time=354.948..379.188 rows=221938 loops=1)
Sort Key: t.account_id, c.type
Sort Method: external merge Disk: 6520kB
-> Hash Join (cost=162.89..27426.59 rows=175530 width=19) (actual time=6.548..225.348 rows=221938 loops=1)
Hash Cond: (t.account_id = d.account_id)
-> Seq Scan on _trans t (cost=0.00..21547.20 rows=1056320 width=8) (actual time=0.095..97.630 rows=1056320 loops=1)
-> Hash (cost=153.54..153.54 rows=748 width=15) (actual time=5.592..5.592 rows=892 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 50kB
-> Hash Join (cost=27.07..153.54 rows=748 width=15) (actual time=0.945..5.076 rows=892 loops=1)
Hash Cond: (d.disp_id = c.disp_id)
-> Seq Scan on _disp d (cost=0.00..102.11 rows=4500 width=8) (actual time=0.018..2.294 rows=4500 loops=1)
Filter: ((type)::text = 'OWNER'::text)
Rows Removed by Filter: 869
-> Hash (cost=15.92..15.92 rows=892 width=15) (actual time=0.911..0.911 rows=892 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 50kB
-> Seq Scan on _card c (cost=0.00..15.92 rows=892 width=15) (actual time=0.012..0.393 rows=892 loops=1)
Planning time: 0.821 ms
Execution time: 441.466 ms
(20 rows)

Algoritmos

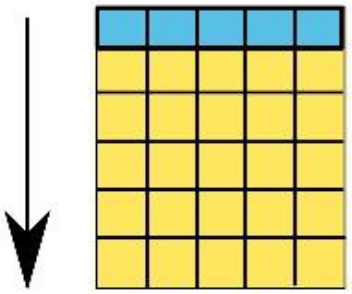
Hash Join

Carrega os registros candidatos de um lado da junção em uma tabela de hash, que é então examinada para cada registro do outro lado da junção.



Seq Scan

Varre a tabela inteira sequencialmente na forma em que ela foi armazenada no disco, descartando linhas que não corresponderem com o filtro. O resultado são linhas desordenadas.



Algoritmos

Sort/Sort Key

Classifica o conjunto nas colunas mencionadas em Sort Key. A operação precisa de grandes quantidades de memória para materializar o resultado intermediário.

Usa merge sort como método.

6 5 3 1 8 7 2 4

GroupAggregate

Agrega um conjunto pré-definido de acordo com a group by cláusula. Esta operação não armazena grandes quantidades de dados..

NÃO OTIMIZADA

1,4701ms

Tempo médio de Planejamento

(DP: 0,7481 ms)

414,04ms

Tempo médio Execução

(DP: 27,96 ms)

OTIMIZADA

0,8178 ms

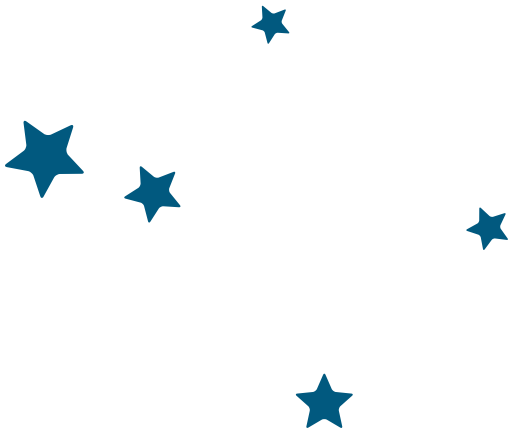
Tempo médio de Planejamento

(DP: 0,20 ms)

408,17 ms

Tempo médio Execução

(DP: 8,71 ms)

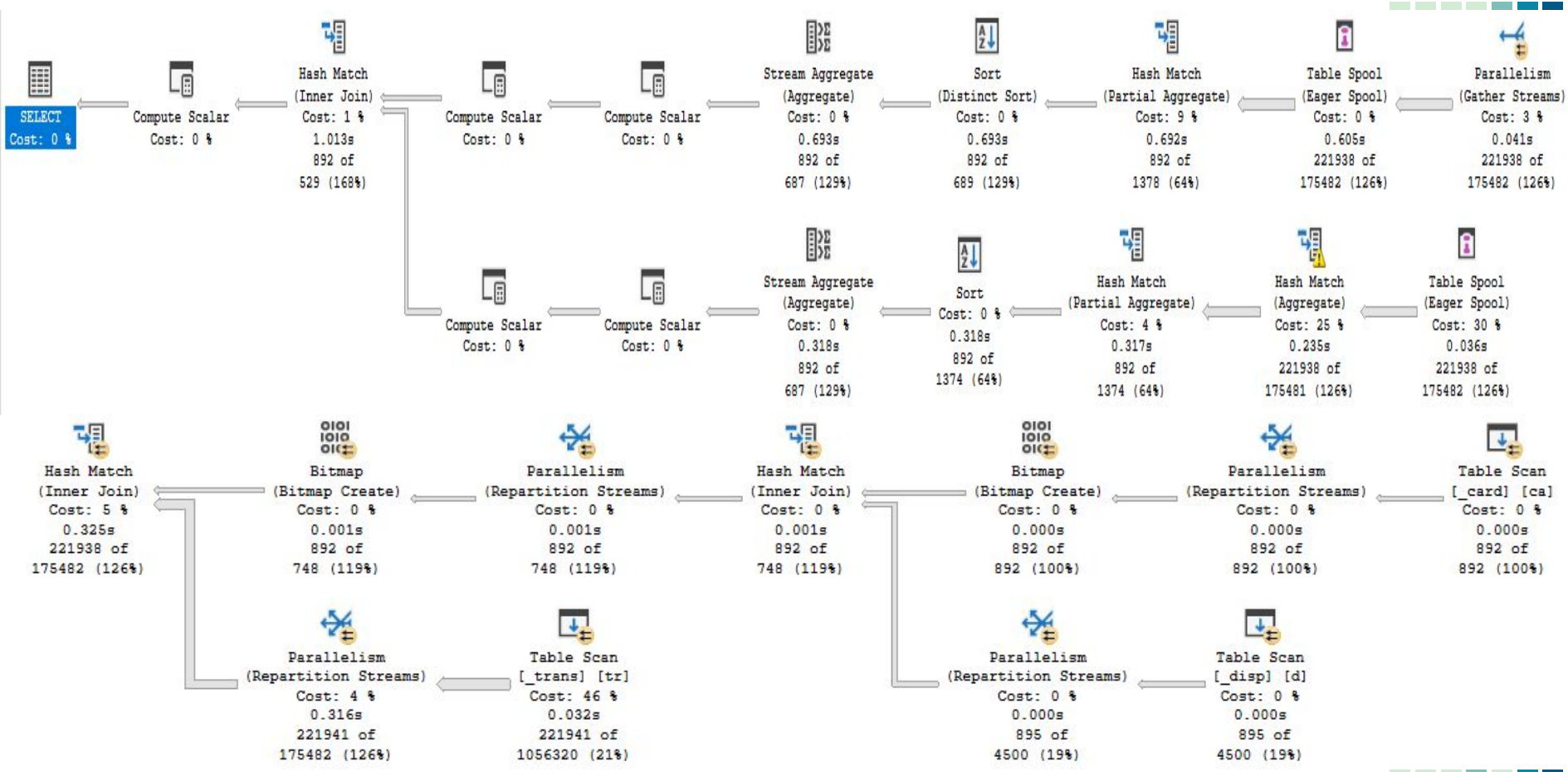


SQL Server

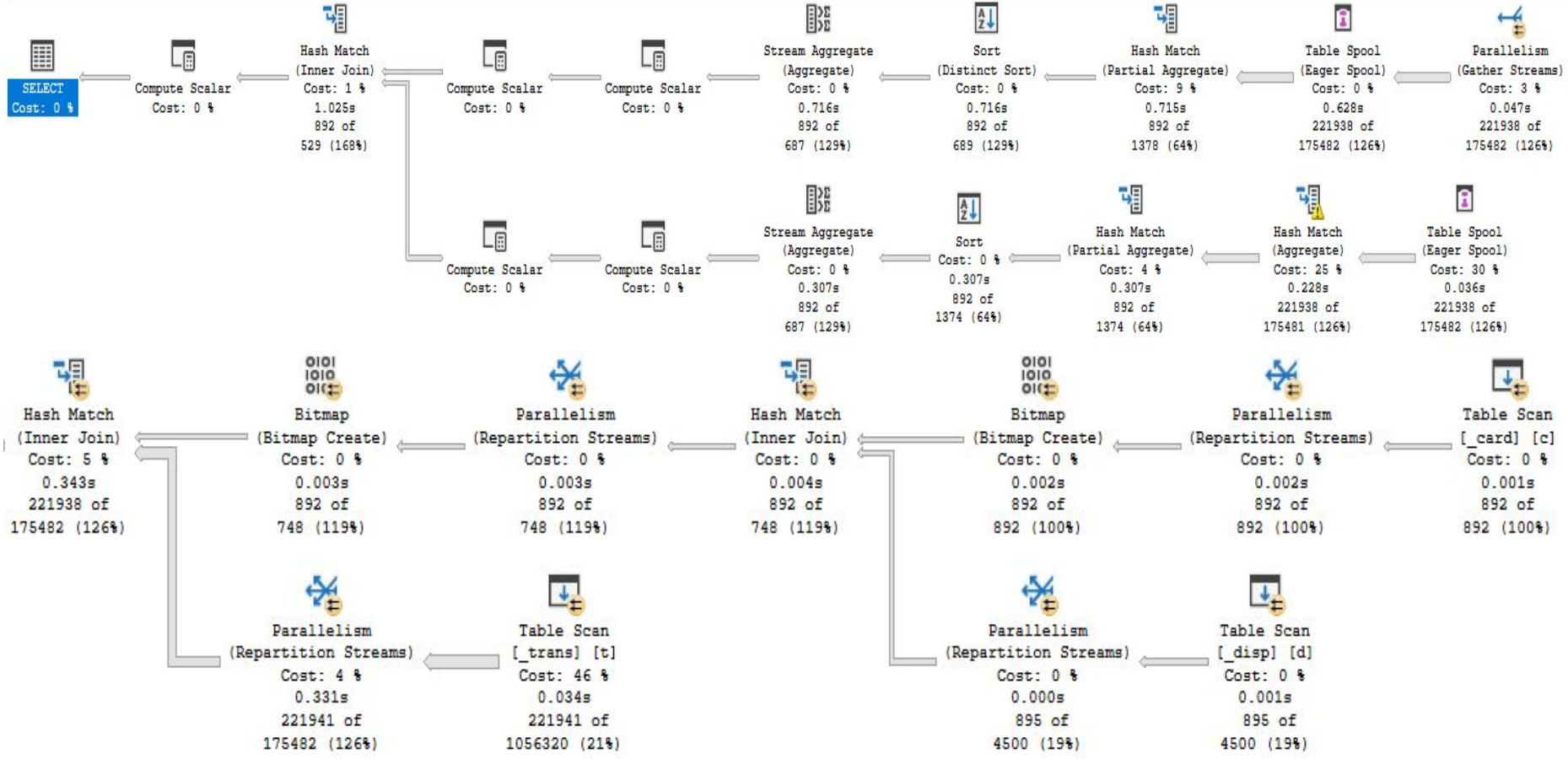
Execução das consultas

Plano de consulta 2

NÃO OTIMIZADA



OTIMIZADA


SELECT
Cost: 0 %
Compute Scalar
Cost: 0 %
Hash Match
(Inner Join)
Cost: 1 %
1.025s
892 of
529 (168%)
Compute Scalar
Cost: 0 %
Compute Scalar
Cost: 0 %
Stream Aggregate
(Aggregate)
Cost: 0 %
0.716s
892 of
687 (129%)
Sort
(Distinct Sort)
Cost: 0 %
0.716s
892 of
689 (129%)
Hash Match
(Partial Aggregate)
Cost: 9 %
0.715s
892 of
1378 (64%)
Table Spool
(Eager Spool)
Cost: 0 %
0.628s
221938 of
175482 (126%)
Parallelism
(Gather Streams)
Cost: 3 %
0.047s
221938 of
175482 (126%)
Compute Scalar
Cost: 0 %
Compute Scalar
Cost: 0 %
Stream Aggregate
(Aggregate)
Cost: 0 %
0.307s
892 of
687 (129%)
Sort
Cost: 0 %
0.307s
892 of
1374 (64%)
Hash Match
(Partial Aggregate)
Cost: 4 %
0.307s
892 of
1374 (64%)
Hash Match
(Aggregate)
Cost: 25 %
0.228s
221938 of
175481 (126%)
Table Spool
(Eager Spool)
Cost: 30 %
0.036s
221938 of
175482 (126%)
Hash Match
(Inner Join)
Cost: 5 %
0.343s
221938 of
175482 (126%)
Bitmap
(Bitmap Create)
Cost: 0 %
0.003s
892 of
748 (119%)
Parallelism
(Repartition Streams)
Cost: 0 %
0.003s
892 of
748 (119%)
Hash Match
(Inner Join)
Cost: 0 %
0.004s
892 of
748 (119%)
Bitmap
(Bitmap Create)
Cost: 0 %
0.002s
892 of
892 (100%)
Parallelism
(Repartition Streams)
Cost: 0 %
0.002s
892 of
892 (100%)
Table Scan
[_card] [c]
Cost: 0 %
0.001s
892 of
892 (100%)
Parallelism
(Repartition Streams)
Cost: 4 %
0.331s
221941 of
175482 (126%)
Table Scan
[_trans] [t]
Cost: 46 %
0.034s
221941 of
1056320 (21%)
Parallelism
(Repartition Streams)
Cost: 0 %
0.000s
895 of
4500 (19%)
Table Scan
[_disp] [d]
Cost: 0 %
0.001s
895 of
4500 (19%)

Algoritmos

Hash Match

O operador Hash Match implementa várias operações lógicas diferentes que usam uma tabela hash na memória para localizar dados correspondentes.

Dois tipos: Join ou Aggregate

https://bit.ly/2Yp6zPu

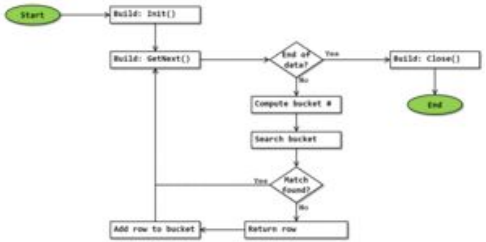
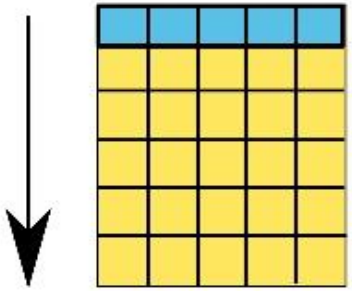


Table Scan

Busca em todos os elementos da tabela, de forma sequencial;

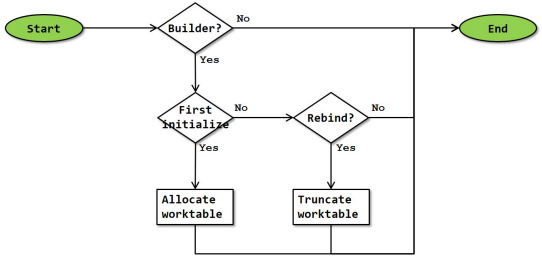


Algoritmos

Table Spool

É um dos quatro operadores de spool que o SQL Server suporta. Ele retém uma cópia de todos os dados que lê em tempdb Essas cópias podem ser disponibilizadas na mesma parte dos planos de execução, ou em outra parte.

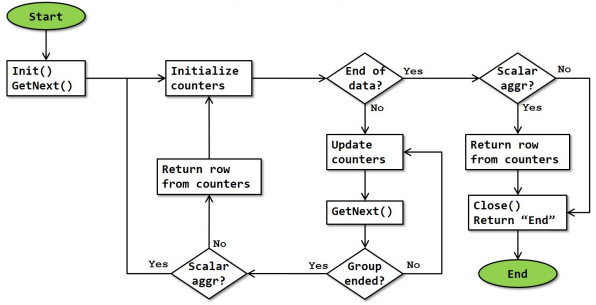
https://bit.ly/31VzFVe



Stream Aggregate

Realiza agregação nas entradas. Ele tem uma única entrada e uma única saída , normalmente, o número de linhas em sua saída é muito menor do que em sua entrada porque retorna apenas uma única linha para cada grupo de linhas na entrada.

https://bit.ly/2KEw10s

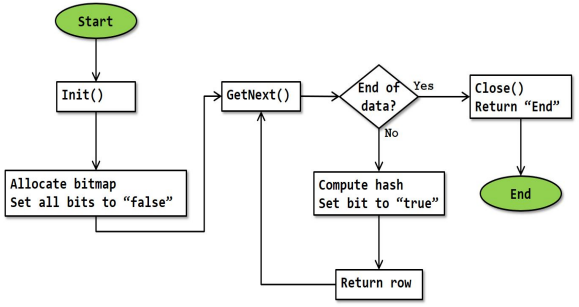


Algoritmos

Bitmap

Um bitmap é uma estrutura que armazena valores booleanos para um intervalo consecutivo de valores em uma pequena quantidade de memória.

https://bit.ly/2NhQH0h



Parallelism

Este operador altera a distribuição das linhas de (semi-) aleatório para determinístico. Processa as linhas em diferentes CPU, retornando o resultado ``correto``.

https://bit.ly/2J6Wxwd

Parallelism

Gather streams.

Physical Operation	Parallelism
Logical Operation	Gather Streams
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	259977
Actual Number of Batches	0
Estimated I/O Cost	0
Estimated Operator Cost	0,6987 (6%)
Estimated Subtree Cost	11,4084
Estimated CPU Cost	0,698656
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	228119
Estimated Row Size	41 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	0

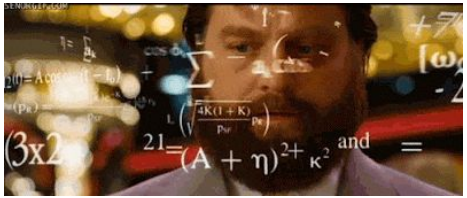
Output List

[teste].[dbo].[_order].order_id; [teste].[dbo].[_order].amount; [teste].[dbo].[_trans].account_id; [teste].[dbo].[_trans].date; [teste].[dbo].[_trans].operation

Algoritmos

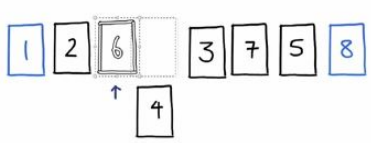
Compute Scalar

O operador Compute Scalar avalia uma expressão para produzir um valor escalar computado. Isso pode ser retornado ao usuário, ter referência em outro lugar na consulta, ou ambos.



Sort

O operador Sort classifica todas as linhas de entrada.



https://bit.ly/2KHmaab

NÃO OTIMIZADA

1167,7ms

Tempo médio de CPU
(DP: 57,09 ms)

1163,5ms

Tempo médio Execução
(DP: 30,91 ms)

OTIMIZADA

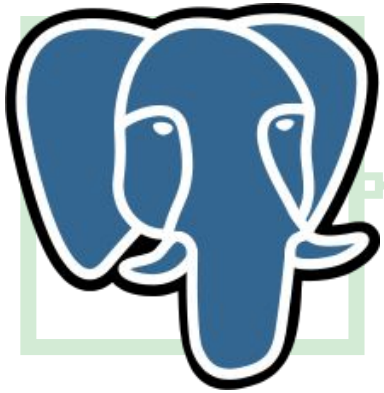
1260,4 ms

Tempo médio de CPU
(DP: 61,34 ms)

1192,8 ms

Tempo médio Execução
(DP: 45,2 ms)

Comparação Tempo Execução



	POSTGRES	SQL SERVER
otimizada	408,17	1192,8
Não otimizada	414,04	1163,5

Plano de consulta 3

Retornar a data e tipo de transação de todas as contas onde a ordem id é igual a 29410 e o símbolo da ordem é igual a SIPO

Plano de consulta 3

Retornar a data e tipo de transação de todas as contas onde a ordem id é igual a 29410 e o símbolo da ordem é igual a SIPO

date	type
1993-06-11	Prijem
1994-10-11	Prijem
1994-11-11	Prijem

Plano de consulta 3

Não otimizada

```
SELECT date, type FROM _trans  WHERE
account_id = (SELECT account_id FROM
(SELECT o.order_id,o.account_id FROM
_order o WHERE o.k_symbol = 'SIPO') as
t  WHERE t.order_id = 29410);
```

Otimizada

```
SELECT date, type FROM _trans
WHERE  account_id = (SELECT
o.account_id FROM _order o
WHERE o.k_symbol = 'SIPO' AND
o.order_id = 29410 );
```

;

POSTGRES

Execução das consultas

Plano de consulta 3

NÃO OTIMIZADA

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on _trans  (cost=144.06..24332.06 rows=239 width=17) (actual time=3.141..88.059 rows=246 loops=1)
   Filter: (account_id = $0)
   Rows Removed by Filter: 1056074
   InitPlan 1 (returns $0)
     ->  Seq Scan on _order o  (cost=0.00..144.06 rows=1 width=4) (actual time=0.021..2.555 rows=1 loops=1)
           Filter: (((k_symbol)::text = 'SIPO'::text) AND (order_id = 29410))
           Rows Removed by Filter: 6470
 Planning time: 0.206 ms
 Execution time: 88.137 ms
(9 rows)

postgres=#
```

NÃO OTIMIZADA

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on _trans  (cost=144.06..24332.06 rows=239 width=17) (actual time=3.141..88.059 rows=246 loops=1)
   Filter: (account_id = $0)
   Rows Removed by Filter: 1056074
   InitPlan 1 (returns $0)
     ->  Seq Scan on _order o  (cost=0.00..144.06 rows=1 width=4) (actual time=0.021..2.555 rows=1 loops=1)
           Filter: (((k_symbol)::text = 'SIPO'::text) AND (order_id = 29410))
           Rows Removed by Filter: 6470
 Planning time: 0.206 ms
 Execution time: 88.137 ms
(9 rows)

postgres=#
```

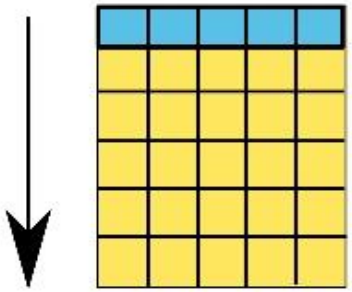
OTIMIZADA

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on _trans  (cost=144.06..24332.06 rows=239 width=17) (actual time=2.894..87.041 rows=246 loops=1)
   Filter: (account_id = $0)
   Rows Removed by Filter: 1056074
   InitPlan 1 (returns $0)
     ->  Seq Scan on _order o  (cost=0.00..144.06 rows=1 width=4) (actual time=0.021..2.388 rows=1 loops=1)
           Filter: (((k_symbol)::text = 'SIPO'::text) AND (order_id = 29410))
           Rows Removed by Filter: 6470
 Planning time: 0.186 ms
 Execution time: 87.110 ms
(9 rows)
```

Algoritmos

Seq Scan

Varre a tabela inteira sequencialmente na forma em que ela foi armazenada no disco, descartando linhas que não corresponderem com o filtro. O resultado são linhas desordenadas.



NÃO OTIMIZADA

0,1427ms

Tempo médio de Planejamento

(DP: 0,0278 ms)

109,69ms

Tempo médio Execução

(DP: 5,25 ms)

OTIMIZADA

0,1848 ms

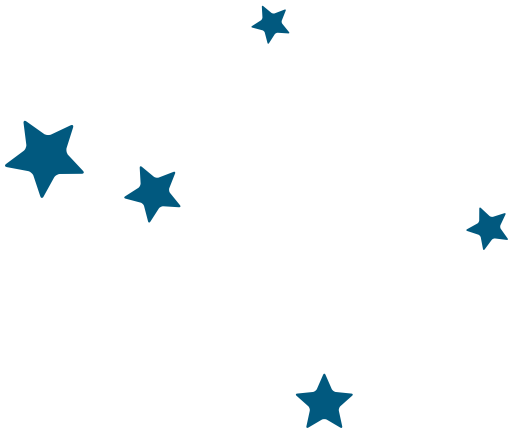
Tempo médio de Planejamento

(DP: 0,04 ms)

100,54 ms

Tempo médio Execução

(DP: 13,95 ms)



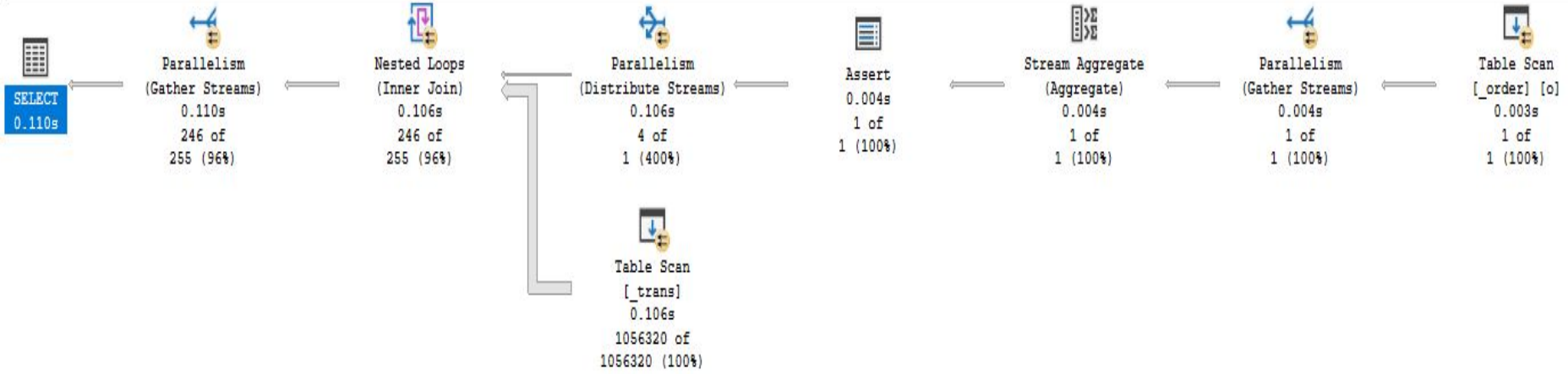
Microsoft® SQL Server®

SQL Server

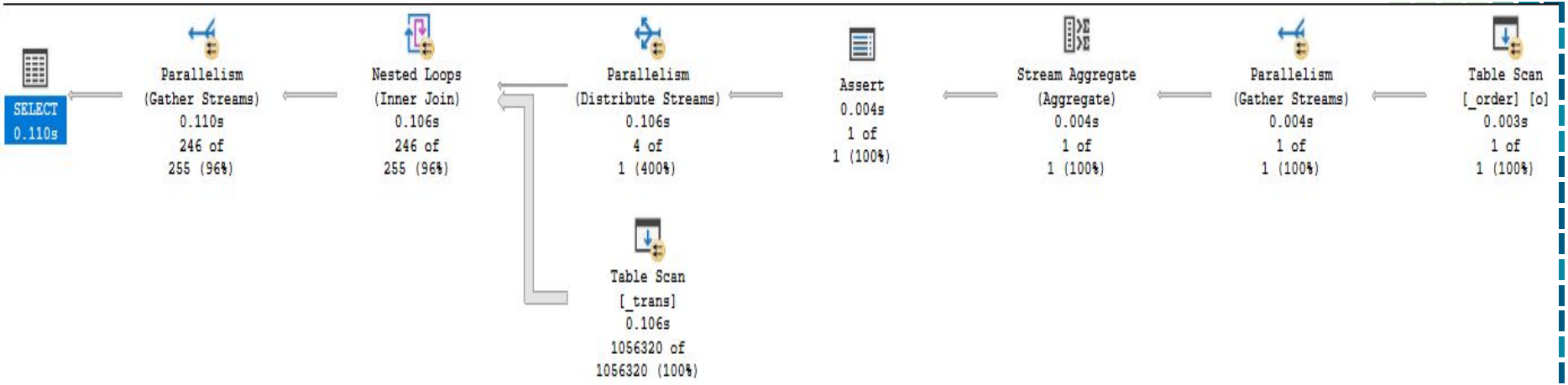
Execução das consultas

Plano de consulta 2

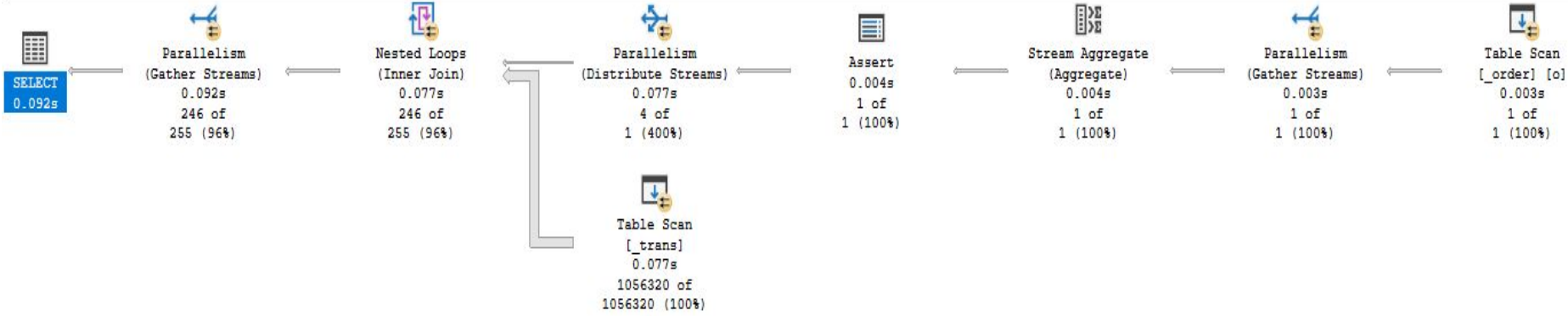
NÃO OTIMIZADA



NÃO OTIMIZADA



OTIMIZADA



Algoritmos

Parallelism

Este operador altera a distribuição das linhas de (semi-) aleatório para determinístico. Processa as linhas em diferentes CPU, retornando o resultado ¨correto¨.

https://bit.ly/2J6Wxw

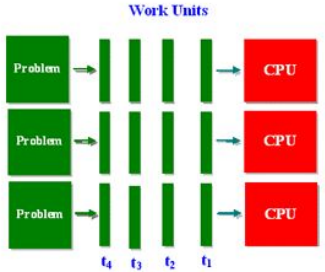
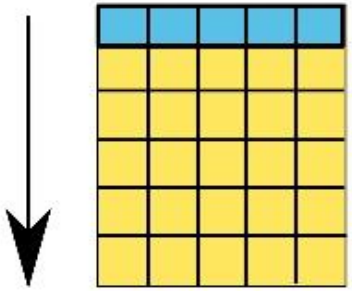


Table Scan

Busca em todos os elementos da tabela, de forma sequencial;

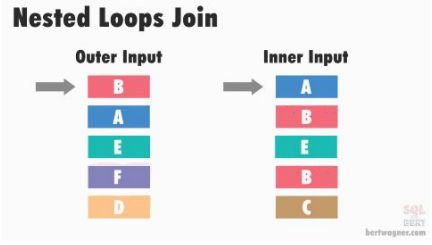


Algoritmos

Nested Loop

O operador Nested Loops executa operações lógicas de junção interna, junção externa esquerda, left semi join e left anti semi join. As junções de loops aninhados executam uma pesquisa na tabela interna para cada linha da tabela externa, normalmente usando um índice.

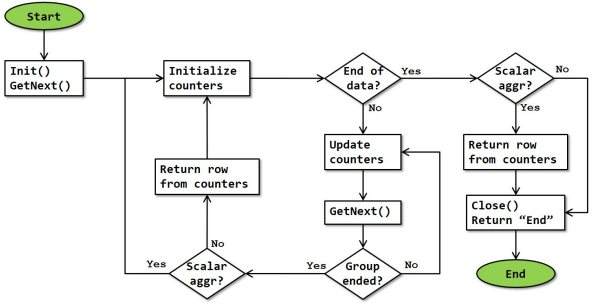
https://bit.ly/2KHmaab



Stream Aggregate

Realiza agregação nas entradas. Ele tem uma única entrada e uma única saída , normalmente, o número de linhas em sua saída é muito menor do que em sua entrada porque retorna apenas uma única linha para cada grupo de linhas na entrada.

https://bit.ly/2KEw10s



OTIMIZADA

192,1ms

Tempo médio de CPU

(DP: 31,12 ms)

184,3ms

Tempo médio Execução

(DP: 15,16 ms)

NÃO OTIMIZADA

198,6 ms

Tempo médio de CPU

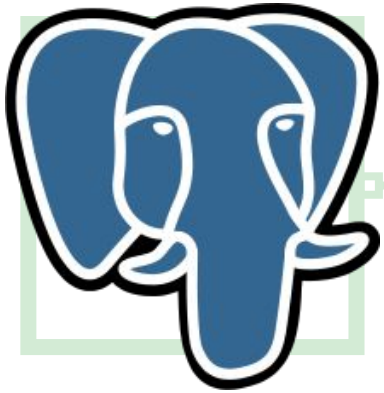
(DP: 31,18 ms)

195,8 ms

Tempo médio Execução

(DP: 10,75 ms)

Comparação Tempo Execução



	POSTGRES	SQL SERVER
otimizada	109,69	184,3
Não otimizada	100,54	195,8

Considerações Finais

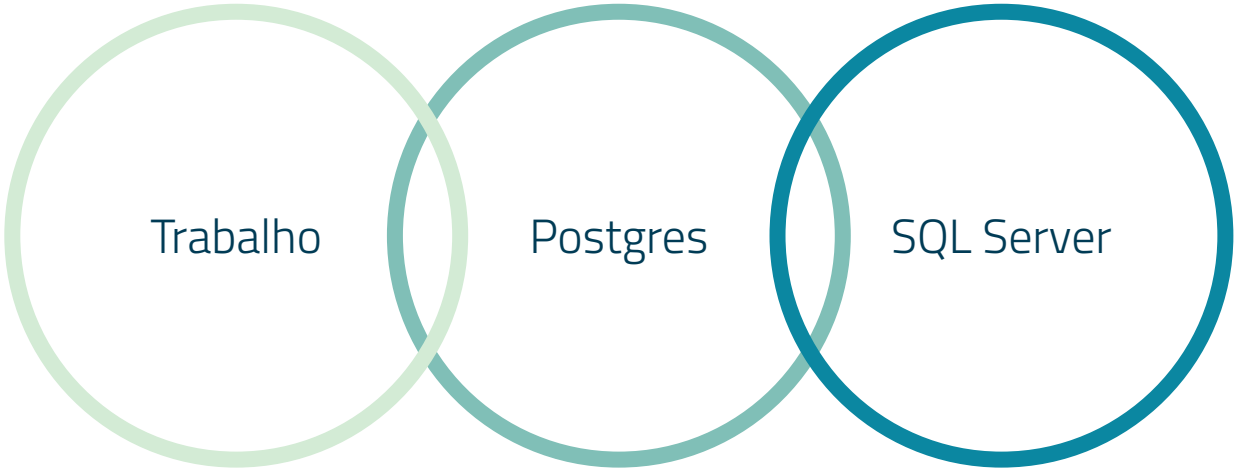


Table Scan

Scan rows from a table.

Physical Operation	Table Scan
Logical Operation	Table Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	6471
Actual Number of Rows	3401
Actual Number of Batches	0
Estimated I/O Cost	0,0261665
Estimated Operator Cost	0,0297648 (0%)
Estimated CPU Cost	0,0035983
Estimated Subtree Cost	0,0297648
Number of Executions	4
Estimated Number of Executions	1
Estimated Number of Rows	3403,32
Estimated Number of Rows to be Read	6471
Estimated Row Size	16 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	4

Predicate

[teste].[dbo].[_order].[amount] as [o].[amount]>=(2400)

Object

[teste].[dbo].[_order] [o]

Output List

[teste].[dbo].[_order].order_id; [teste].[dbo].[_order].amount

SELECT

Cached plan size	56 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	4
Estimated Subtree Cost	11,4084
Memory Grant	7528
Estimated Number of Rows	228119

Statement

```
SELECT o.order_id, o.amount,
tb.operation,tb.account_id, tb.date
FROM _order o JOIN (SELECT
t.operation,t.account_id, t.date, t.amount
FROM _trans t WHERE t.amount >=
2400) tb on o.amount=tb.amount
```

Table Scan

Scan rows from a table.

Physical Operation	Table Scan
Logical Operation	Table Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	1056320
Actual Number of Rows	195993
Actual Number of Batches	0
Estimated I/O Cost	6,4032
Estimated Operator Cost	6,98422 (61%)
Estimated CPU Cost	0,581015
Estimated Subtree Cost	6,98422
Number of Executions	4
Estimated Number of Executions	1
Estimated Number of Rows	509240
Estimated Number of Rows to be Read	1056320
Estimated Row Size	36 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	6

Predicate

[teste].[dbo].[_trans].[amount] as [t].[amount]>=(2400) AND PROBE([Bitmap1004],[teste].[dbo].[_trans].[amount] as [t].[amount])

Object

[teste].[dbo].[_trans] [t]

Output List

[teste].[dbo].[_trans].account_id; [teste].[dbo].[_trans].date; [teste].[dbo].[_trans].operation; [teste].[dbo].[_trans].amount

Hash Match

Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.

Physical Operation	Hash Match
Logical Operation	Inner Join
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	259977
Actual Number of Batches	0
Estimated Operator Cost	1,9597814 (17%)
Estimated I/O Cost	0
Estimated CPU Cost	1,95977
Estimated Subtree Cost	10,7097
Number of Executions	4
Estimated Number of Executions	1
Estimated Number of Rows	228119
Estimated Row Size	41 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	1

Output List

[teste].[dbo].[_order].order_id; [teste].[dbo].[_order].amount; [teste].[dbo].[_trans].account_id; [teste].[dbo].[_trans].date; [teste].[dbo].[_trans].operation

Hash Keys Probe

[teste].[dbo].[_trans].amount

Probe Residual

[teste].[dbo].[_trans].[amount] as [t].[amount]=[teste].[dbo].[_order].[amount] as [o].[amount]

Obrigada!

Otimização de consultas
Postgres X SQL server

Juliana Isabel de Freitas Rosin
Vinícius Marcelo Rifam Laurindo