c) (artefato do tipo texto) Para cada consulta definida na Parte I do trabalho, o grupo deverá intervir no banco de dados de forma a criar diferentes tipos de índices, excluir índices, aumentar ou diminuir quantidade de dados que devem ser retornados pela consulta, modificar levemente as condições da cláusula where ou do join, de forma a executar novamente a consulta e estudar as modificações que ocorrem no plano de execução. As modificações deverão ser analisadas pelo grupo e a análise deverá fazer parte do trabalho (os mesmos itens de relatórios e screenshots do item (b) devem ser expostos aqui para fins de comparação). O grupo deve destacar as mudanças claramente. Pelo menos duas alterações em cada uma das quatro consultas são requeridas. Procure atender diferentes solicitações deste item do trabalho no conjunto de 8 novas consultas que serão criadas aqui. O texto deve ser entregue em formato PDF. O código deverá ser entregue em formato ASCII (.txt) na plataforma adotada pelo professor da sua turma (e-disciplina) (ver observação em vermelho abaixo).

1) Consulta que busca encontrar o nome e id de cada cliente que comprou algum produto que entrou em estoque entre 1/1/2017 e 31/12/2020 junto ao id do produto.

ORIGINAL:

```
SELECT cliente.nome, cliente.idcliente, produto.idproduto FROM produto, historico, cliente
WHERE cliente.idcliente = historico.idcliente AND (historico.idproduto = (SELECT
produto.idproduto where(dataadquirida BETWEEN '2017-01-01' AND '2020-12-31') AND
(vendido = true)))
ORDER BY dataadquirida;
```

MODIFICADA 1:

```
SELECT cliente.nome, cliente.idcliente, produto.idproduto FROM cliente,
produto join historico on
historico.idproduto = produto.idproduto WHERE cliente.idcliente = historico.idcliente AND
(dataadquirida BETWEEN '2017-01-01' AND '2020-12-31') AND (vendido = true)
ORDER BY dataadquirida;
```

Modificação do WHERE substituindo subquery por mais condições com AND

```
"QUERY PLAN"
"Sort  (cost=41.08..41.09 rows=2 width=44)"
"  Sort Key: produto.dataadquirida"
"  ->  Nested Loop  (cost=21.28..41.07 rows=2 width=44)"
"        ->  Hash Join  (cost=21.13..40.60 rows=2 width=12)"
"              Hash Cond: (historico.idproduto = produto.idproduto)"
"              ->  Seq Scan on historico  (cost=0.00..17.50 rows=750 width=8)"
"              ->  Hash  (cost=21.10..21.10 rows=2 width=8)"
"                    ->  Seq Scan on produto  (cost=0.00..21.10 rows=2 width=8)"
"                          Filter: (vendido AND (dataadquirida >= '2017-01-01'::date) AND
(dataadquirida <= '2020-12-31'::date))"
"        ->  Index Scan using id_cliente_id on cliente  (cost=0.15..0.23 rows=1 width=36)"
"              Index Cond: (idcliente = historico.idcliente)"
```

Index Cond: (idcliente = historico.idcliente)

Index da condição idcliente = historico.idcliente

Index Scan using id_cliente_id on cliente (cost=0.15..0.23 rows=1 width=36)

Escaneamento do Index com custo estimado 0.15, e custo total 0.23, há baixa diferença do estimado para o total.

Filter: (vendido AND (dataadquirida >= '2017-01-01'::date) AND (dataadquirida <= '2020-12-31'::date))

Filtro vendido = true ^ 2017-01-01 <= dataadquirida >= 2020-12-31

Seq Scan on produto (cost=0.00..21.10 rows=2 width=8)

Lê do disco como entrada a tabela de serviço cada tupla relevante para a consulta.

Custo total estimado 0.00, custo total 21.10, custo estimado teve grande discrepância com o custo total.

Hash (cost=21.10..21.10 rows=2 width=8)

Constrói e preenche uma estrutura de armazenamento para os dados em hash.

Custo total estimado 21.10, custo total 21.10, hash consegue estimar com precisão o custo total

Seq Scan on historico (cost=0.00..17.50 rows=750 width=8)

Leitura da tabela histórico com custo total estimado 0.00 e custo total 17.50

Scan sequencial não conseguiu estimar corretamente o custo total.

Hash Cond: (historico.idproduto = produto.idproduto)

Leitura do Hash em condição historico.idproduto = produto.idproduto

Hash Join (cost=21.13..40.60 rows=2 width=12)

Leitura do Hash em Join com custo total estimado 21.13 e custo total 40.60

Nested Loop (cost=21.28..41.07 rows=2 width=44)

Loop de custo total estimado 21.28 e custo total 41.07

Sort Key: produto.dataadquirida

Sort da tabela usando da chave produto.dataadquirida

Sort (cost=41.08..41.09 rows=2 width=44)

Organização da tabela com sort tem custo total estimado 41.08 e custo total 41.09

MODIFICADA 2:

```
SELECT cliente.nome, cliente.idcliente, produto.idproduto FROM produto, historico, cliente
WHERE cliente.idcliente = historico.idcliente OR (historico.idproduto = (SELECT
produto.idproduto WHERE (dataadquirida BETWEEN '2017-01-01' AND '2020-12-31') OR
(vendido = true)))
ORDER BY dataadquirida;
```

Modificação das cláusulas WHERE onde todos os ANDs foram alterados por ORs.

```
"QUERY PLAN"
"Sort  (cost=23613002.47..23623193.66 rows=4076475 width=44)"
"  Sort Key: produto.dataadquirida"
"  ->  Nested Loop  (cost=0.00..22914625.13 rows=4076475 width=44)"
"        Join Filter: ((cliente.idcliente = historico.idcliente) OR (historico.idproduto = (SubPlan 1)))"
"        ->  Nested Loop  (cost=0.00..6974.25 rows=555000 width=17)"
"              ->  Seq Scan on historico  (cost=0.00..17.50 rows=750 width=8)"
"              ->  Materialize  (cost=0.00..21.10 rows=740 width=9)"
"                    ->  Seq Scan on produto  (cost=0.00..17.40 rows=740 width=9)"
```

"   -> Materialize (cost=0.00..29.05 rows=1270 width=36)"
"   -> Seq Scan on cliente (cost=0.00..22.70 rows=1270 width=36)"
"   SubPlan 1"
"   -> Result (cost=0.01..0.01 rows=1 width=4)"
"   One-Time Filter: (((produto.dataadquirida >= '2017-01-01'::date) AND (produto.dataadquirida <= '2020-12-31'::date)) OR produto.vendido)"

One-Time Filter: (((produto.dataadquirida >= '2017-01-01'::date) AND (produto.dataadquirida <= '2020-12-31'::date)) OR produto.vendido)

Result (cost=0.01..0.01 rows=1 width=4)
Resultado do filtro, custo total estimado e custo total tem o mesmo valor de 0.01 fazendo parte do subplano.
SubPlan 1

Seq Scan on cliente (cost=0.00..22.70 rows=1270 width=36)
Leitura da tabela cliente com custo total estimado 0.00 e custo total 22.70
Scan sequencial não conseguiu estimar corretamente o custo total e teve um alto número de filas acessadas.
Materialize (cost=0.00..29.05 rows=1270 width=36)
No materialize o output das ações seguintes são materializadas na memória com custo total estimado de 0.00 e custo total de 29.05
Seq Scan on produto (cost=0.00..17.40 rows=740 width=9)
Leitura da tabela produto com custo total estimado 0.00 e custo total 17.40
Scan sequencial não conseguiu estimar corretamente o custo total e teve um alto número de filas acessadas.
Nested Loop (cost=0.00..6974.25 rows=555000 width=17)
Cada fileira de dado da primeira tabela é checado com todas as fileiras de dados da segunda tabela para obter a condição, logo há um alto número de filas percorridas e custo total, sendo custo total esperado 0.00 e custo total 6974.25
Join Filter: ((cliente.idcliente = historico.idcliente) OR (historico.idproduto = (SubPlan 1)))
Filtro de join
Nested Loop (cost=0.00..22914625.13 rows=4076475 width=44)
Cada fileira de dado da primeira tabela é checado com todas as fileiras de dados da segunda tabela para obter a condição, logo há um alto número de filas percorridas e custo total, sendo custo total esperado 0.00 e custo total 22914625.13
Sort Key: produto.dataadquirida
Sort da tabela usando da chave produto.dataadquirida
Sort (cost=23613002.47..23623193.66 rows=4076475 width=44)
Organização da tabela com sort tem custo total estimado 23613002.47 e custo total 23623193.66 tendo assim um custo extremamente alto e estimativa similar ao valor total.

2) Consulta que procura o fornecedor com maior número de produtos devolvidos, retorna o id do fornecedor, o nome da fornecedora, e o número de produtos devolvidos fornecidos pela empresa.

ORIGINAL:
SELECT fornecedores.idfornecedor,fornecedores.nome, count(fornecedores.idfornecedor) AS ProdutosDevolvidos FROM fornecedores join produto

```
on produto.idfornecedor = fornecedores.idfornecedor
where produto.idproduto = (SELECT idproduto where vendido = 't' EXCEPT (SELECT
idproduto FROM produto WHERE devolvido = 'f'))
group by fornecedores.idfornecedor
order by ProdutosDevolvidos DESC
Limit 1;
```

MODIFICADA 1:

```
SELECT fornecedores.idfornecedor,fornecedores.nome, count(fornecedores.idfornecedor)
AS ProdutosDevolvidos FROM fornecedores join produto
on produto.idfornecedor = fornecedores.idfornecedor
where produto.idproduto = (SELECT idproduto where vendido = 't')
group by fornecedores.idfornecedor
order by ProdutosDevolvidos DESC
Limit 4;
```

Aumenta a quantidade de dados a ser retornado ao aumentar Limite para 4 e remove a condição EXCEPT (SELECT idproduto FROM produto WHERE devolvido = 'f')

```
"QUERY PLAN"
"Limit  (cost=48.03..48.04 rows=4 width=44)"
"  ->  Sort  (cost=48.03..48.04 rows=4 width=44)"
"        Sort Key: (count(fornecedores.idfornecedor)) DESC"
"        ->  GroupAggregate  (cost=47.92..47.99 rows=4 width=44)"
"              Group Key: fornecedores.idfornecedor"
"              ->  Sort  (cost=47.92..47.93 rows=4 width=36)"
"                    Sort Key: fornecedores.idfornecedor"
"                    ->  Hash Join  (cost=26.70..47.88 rows=4 width=36)"
"                          Hash Cond: (fornecedores.idfornecedor = produto.idfornecedor)"
"                          ->  Seq Scan on fornecedores  (cost=0.00..18.10 rows=810 width=36)"
"                          ->  Hash  (cost=26.65..26.65 rows=4 width=4)"
"                                ->  Seq Scan on produto  (cost=0.00..26.65 rows=4 width=4)"
"                                      Filter: (idproduto = (SubPlan 1))"
"                                      SubPlan 1"
"                                        ->  Result  (cost=0.00..0.01 rows=1 width=4)"
"                                              One-Time Filter: produto.vendido"
```

One-Time Filter: produto.vendido

Filtro produto.vendido = true

Result (cost=0.00..0.01 rows=1 width=4)

Resultado filtro de custo total estimado 0.00 e custo total 0.01

SubPlan 1

Filter: (idproduto = (SubPlan 1))

Filtro idproduto é igual ao output de SubPlan 1

Seq Scan on produto (cost=0.00..26.65 rows=4 width=4)

Leitura da tabela produto com custo total estimado 0.00 e custo total 26.65

Scan sequencial não conseguiu estimar corretamente o custo total

Hash (cost=26.65..26.65 rows=4 width=4)

Constrói e preenche uma estrutura de armazenamento para os dados em hash com os dados lidos da tabela cliente anteriormente com custo total previsto correto ao total de 26.65.

Seq Scan on fornecedores (cost=0.00..18.10 rows=810 width=36)

Leitura da tabela fornecedores com custo total estimado 0.00 e custo total 18.10 Scan sequencial não conseguiu estimar corretamente o custo total

Hash Cond: (fornecedores.idfornecedor = produto.idfornecedor)

Leitura do Hash em condição fornecedores.idfornecedor = produto.idfornecedor

Hash Join (cost=26.70..47.88 rows=4 width=36)

Leitura da tabela histórico de um registro cada vez para serem colocados na memória em uma estrutura hash para que depois, ao ler a segunda tabela de cliente, é comparado a correspondência de cada registro com relação às da primeira tabela respeitando a condição Hash Cond: (fornecedores.idfornecedor = produto.idfornecedor).

Custo total estimado de 26.70 e custo total 47.88.

Sort Key: fornecedores.idfornecedor"

Sort usando chave fornecedores.idfornecedor

Sort (cost=47.92..47.93 rows=4 width=36)

Operação Sort com custo total estimado de 47.92 e custo total 47.93, estimado chegando muito perto do total.

Group Key: fornecedores.idfornecedor"

Chave de agrupação fornecedores.idfornecedor

GroupAggregate (cost=47.92..47.99 rows=4 width=44)

custo de agregação de grupo total estimado de 47.92 e custo total de 47.99

Sort Key: (count(fornecedores.idfornecedor)) DESC

A chave utilizada para organizar a tabela é (count(fornecedores.idfornecedor)) em ordem decrescente.

Sort (cost=48.03..48.04 rows=4 width=44)

Organização da tabela com sort tem custo total estimado 48.03 e custo total 48.04

Limit (cost=48.03..48.04 rows=4 width=44)

Limite de custo estimado 48.03 e custo total 48.04

MODIFICADA 2:

```
SELECT fornecedores.idfornecedor,fornecedores.nome, count(fornecedores.idfornecedor)
AS ProdutosDevolvidos
FROM fornecedores
INNER JOIN produto ON produto.idfornecedor = fornecedores.idfornecedor AND
produto.vendido = ‘t’
WHERE vendido = 't' EXCEPT (SELECT produto.idproduto FROM produto WHERE
devolvido = 'f')
group by fornecedores.idfornecedor
order by ProdutosDevolvidos DESC
Limit 1;
```

3) Informações dos serviços (id, preço e tipo) que teve o maior preço dentro de cada tipo de serviço e que tenham já sido terminados e feito pelo técnico com id específico (idtecnico: 7):

ORIGINAL:

```
SELECT s.idservico, s.preco, s.tipo
FROM servico s
WHERE status='T' AND idtecnico = 7
AND (s.tipo, s.preco) IN
(SELECT tipo, MAX(preco)
FROM servico
GROUP BY tipo);
```

MODIFICADA 1:

```
SELECT s.idservico, s.preco, s.tipo
FROM servico s
WHERE status='T' AND datapedido > '2020/06/01'
AND (s.tipo, s.preco) IN
(SELECT tipo, MAX(preco)
FROM servico
GROUP BY tipo);
```

data do pedido deve ser maior do que 2020/06/01 e o id do técnico não é pedido

```
"Hash Join  (cost=41.61..47.12 rows=1 width=68)"
"  Hash Cond: (((servico.tipo)::text = (s.tipo)::text) AND ((max(servico.preco)) =
s.preco))"
"  ->  HashAggregate  (cost=20.80..22.80 rows=200 width=64)"
"        Group Key: servico.tipo"
"        ->  Seq Scan on servico  (cost=0.00..17.20 rows=720 width=64)"
"  ->  Hash  (cost=20.80..20.80 rows=1 width=68)"
"        ->  Seq Scan on servico s  (cost=0.00..20.80 rows=1 width=68)"
"              Filter: ((datapedido > '2020-06-01'::date) AND (status = 'T'::bpchar))"
```

Filter: ((datapedido > '2020-06-01'::date) AND (status = 'T'::bpchar))

Filtro datapedido > '2020-06-01 ^ status = 'T'

Seq Scan on servico s (cost=0.00..20.80 rows=1 width=68)

Leitura da tabela serviço com custo total estimado 0.00 e custo total 20.80 Scan sequencial não conseguiu estimar corretamente o custo total

Hash (cost=20.80..20.80 rows=1 width=68)

Constrói e preenche uma estrutura de armazenamento para os dados em hash com custo total previsto correto ao total de 20.80.

Seq Scan on servico (cost=0.00..17.20 rows=720 width=64)

Leitura da tabela serviço com custo total estimado 0.00 e custo total 17.20 Scan sequencial não conseguiu estimar corretamente o custo total

Group Key: servico.tipo

Chave de agrupação servico.tipo

HashAggregate (cost=20.80..22.80 rows=200 width=64)

Agrupamento entre todos registros. (Group Key: servico.tipo).
Hash Cond: (((servico.tipo)::text = (s.tipo)::text) AND ((max(servico.preco)) = s.preco))"
Leitura do Hash em condição servico.tipo = s.tipo ^ max(servico.preco = s.preco
Hash Join (cost=41.61..47.12 rows=1 width=68)
Leitura da tabela histórico de um registro cada vez para serem colocados na memória em uma estrutura hash para que depois, ao ler a segunda tabela de cliente, é comparado a correspondência de cada registro com relação às da primeira tabela respeitando a condição Hash Cond:(((servico.tipo)::text = (s.tipo)::text) AND ((max(servico.preco)) = s.preco))

MODIFICADA 2:

```
SELECT s.idservico, s.preco, s.tipo
FROM servico s
WHERE status='A'
AND (s.tipo, s.preco) IN
(SELECT tipo, MAX(preco)
FROM servico
GROUP BY tipo);
```

Busca por todos os serviços em andamento

```
"QUERY PLAN"
"Hash Join  (cost=39.86..45.37 rows=1 width=68)"
"  Hash Cond: (((servico.tipo)::text = (s.tipo)::text) AND ((max(servico.preco)) = s.preco))"
"  ->  HashAggregate  (cost=20.80..22.80 rows=200 width=64)"
"        Group Key: servico.tipo"
"        ->  Seq Scan on servico  (cost=0.00..17.20 rows=720 width=64)"
"  ->  Hash  (cost=19.00..19.00 rows=4 width=68)"
"        ->  Seq Scan on servico s  (cost=0.00..19.00 rows=4 width=68)"
"              Filter: (status = 'A'::bpchar)"
```

Filter: (status = 'A'::bpchar)
Filtro status = ‘A’
Seq Scan on servico s (cost=0.00..19.00 rows=4 width=68)
Leitura da tabela serviço com custo total estimado 0.00 e custo total 19.00
Scan sequencial não conseguiu estimar corretamente o custo total
Hash (cost=19.00..19.00 rows=4 width=68)
Constrói e preenche uma estrutura de armazenamento para os dados em hash com custo total previsto correto ao total de 19.00.
Seq Scan on servico (cost=0.00..17.20 rows=720 width=64)
Leitura da tabela serviço com custo total estimado 0.00 e custo total 17.20
Scan sequencial não conseguiu estimar corretamente o custo total
Group Key: servico.tipo
Chave de agrupação servico.tipo
HashAggregate (cost=20.80..22.80 rows=200 width=64)
Agrupamento entre todos registros. (Group Key: servico.tipo)
Hash Cond: (((servico.tipo)::text = (s.tipo)::text) AND ((max(servico.preco)) = s.preco))

Leitura do Hash em condição (((servico.tipo)::text = (s.tipo)::text) AND ((max(servico.preco)) = s.preco)

Hash Join (cost=39.86..45.37 rows=1 width=68)

Leitura da tabela histórico de um registro cada vez para serem colocados na memória em uma estrutura hash para que depois, ao ler a segunda tabela de cliente, é comparado a correspondência de cada registro com relação às da primeira tabela respeitando a condição Hash Cond:(((servico.tipo)::text = (s.tipo)::text) AND ((max(servico.preco)) = s.preco))

4) O nome do fornecedor preferido da pessoa que gastou mais dinheiro na loja, junto com o nome dessa pessoa e o quanto ela gastou com os produtos deste fornecedor

ORIGINAL

```
select f.nome, nome_idProd_dinGasto.nome, nome_idProd_dinGasto.dinheiro_gasto
from fornecedores f natural join produto p,
(       select x.idproduto, x.nome, dinheiro_gasto
        from (historico natural join cliente) x,(select c.nome, sum(h.valorpago) as dinheiro_gasto
                from cliente c natural join historico h
                group by c.nome) as tp
        where x.nome = tp.nome) as nome_idProd_dinGasto
where nome_idProd_dinGasto.idproduto = p.idproduto and
                nome_idProd_dinGasto.dinheiro_gasto = (select max(dinheiro_gasto)
                from(select c.nome, sum(h.valorpago) as dinheiro_gasto
                        from cliente c natural join historico h
                        group by c.nome) as temp)
group by nome_idProd_dinGasto.nome, f.nome, nome_idProd_dinGasto.dinheiro_gasto
```

MODIFICADA 1

```
select f.nome, nome_idProd_dinGasto.nome, nome_idProd_dinGasto.dinheiro_gasto
from fornecedores f natural join produto p,
(       select x.idproduto, x.nome, dinheiro_gasto
        from (historico natural join cliente) x,(select c.nome, sum(h.valorpago) as dinheiro_gasto
                from cliente c natural join historico h
                group by c.nome) as tp
        where x.nome = tp.nome) as nome_idProd_dinGasto
where nome_idProd_dinGasto.idproduto = p.idproduto and
                nome_idProd_dinGasto.dinheiro_gasto = (select dinheiro_gasto
                from(select c.nome, sum(h.valorpago) as dinheiro_gasto
                        from cliente c natural join historico h
                        group by c.nome) as temp
                        ORDER BY dinheiro_gasto DESC LIMIT 1)
group by nome_idProd_dinGasto.nome, f.nome, nome_idProd_dinGasto.dinheiro_gasto
```

Troca de select MAX(dinheiro_gasto) por select dinheiro_gasto order by DESC LIMIT 1

“QUERY PLAN”

```
" Group  (cost=93.99..94.00 rows=1 width=96)"
"  Group Key: cliente.nome, f.nome, tp.dinheiro_gasto"
"   InitPlan 1 (returns $0)"
"     ->  Limit  (cost=33.34..33.34 rows=1 width=32)"
"           ->  Sort  (cost=33.34..33.53 rows=75 width=32)"
"                 Sort Key: temp.dinheiro_gasto DESC"
"                 ->  Subquery Scan on temp  (cost=31.28..32.96 rows=75 width=32)"
"                       ->  HashAggregate  (cost=31.28..32.21 rows=75 width=64)"
"                             Group Key: c_1.nome"
"                             ->  Hash Join  (cost=2.69..30.90 rows=75 width=64)"
"                                   Hash Cond: (c_1.idcliente = h_1.idcliente)"
"                                   ->  Seq Scan on cliente c_1  (cost=0.00..22.70 rows=1270 width=36)"
"                                   ->  Hash  (cost=1.75..1.75 rows=75 width=36)"
"                                         ->  Seq Scan on historico h_1  (cost=0.00..1.75 rows=75
width=36)"
"   ->  Sort  (cost=60.65..60.66 rows=1 width=96)"
"         Sort Key: cliente.nome, f.nome"
"         ->  Nested Loop  (cost=32.87..60.64 rows=1 width=96)"
"               ->  Nested Loop  (cost=32.72..60.41 rows=1 width=68)"
"                     ->  Nested Loop  (cost=32.57..59.60 rows=1 width=68)"
"                           ->  Hash Join  (cost=32.42..58.52 rows=6 width=68)"
"                                 Hash Cond: ((cliente.nome)::text = (tp.nome)::text)"
"                                 ->  Seq Scan on cliente  (cost=0.00..22.70 rows=1270 width=36)"
"                                 ->  Hash  (cost=32.41..32.41 rows=1 width=64)"
"                                       ->  Subquery Scan on tp  (cost=31.28..32.41 rows=1 width=64)"
"                                             ->  HashAggregate  (cost=31.28..32.40 rows=1 width=64)"
"                                                   Group Key: c.nome"
"                                                   Filter: (sum(h.valorpago) = $0)"
"                                                   ->  Hash Join  (cost=2.69..30.90 rows=75 width=64)"
"                                                         Hash Cond: (c.idcliente = h.idcliente)"
"                                                         ->  Seq Scan on cliente c  (cost=0.00..22.70 rows=1270
width=36)"
"                                                         ->  Hash  (cost=1.75..1.75 rows=75 width=36)"
"                                                               ->  Seq Scan on historico h  (cost=0.00..1.75
rows=75 width=36)"
"                           ->  Index Scan using id_historico_id on historico  (cost=0.14..0.17 rows=1
width=8)"
"                                 Index Cond: (idcliente = cliente.idcliente)"
"                     ->  Index Scan using id_produto_id on produto p  (cost=0.15..0.81 rows=1
width=8)"
"                           Index Cond: (idproduto = historico.idproduto)"
"               ->  Index Scan using id_fornecedor_id on fornecedores f  (cost=0.15..0.23 rows=1
width=36)"
"                     Index Cond: (idfornecedor = p.idfornecedor)"
```

Index Scan using id_fornecedor_id on fornecedores f (cost=0.15..0.23 rows=1 width=36

Leitura da tabela de fornecedores performada em uma estrutura de árvore B, onde ela percorre até as folhas para conseguir alcançar os registros relevantes para a relação que respeite a condição (Index Cond: (idfornecedor = p.idfornecedor)).

Index Scan using id_produto_id on produto p (cost=0.15..0.81 rows=1 width=8)

Leitura da tabela de produto performada em uma estrutura de árvore B, onde ela percorre até as folhas para conseguir alcançar os registros relevantes para a relação que respeite a condição (Index Cond: (idproduto = historico.idproduto)).

Index Scan using id_historico_id on historico (cost=0.14..0.17 rows=1 width=8)

Leitura da tabela de historico performada em uma estrutura de árvore B, onde ela percorre até as folhas para conseguir alcançar os registros relevantes para a relação que respeite a condição (Index Cond: (idcliente = cliente.idcliente)).

Seq Scan on historico h (cost=0.00..1.75 rows=75 width=36)

Leitura sequencial de historico em disco.

Hash (cost=1.75..1.75 rows=75 width=36)

Construção e armazenamento dos registros de historico em uma tabela hash.

Seq Scan on cliente c (cost=0.00..22.70 rows=1270 width=36)

Leitura sequencial de cliente em disco.

Hash Join (cost=2.69..30.90 rows=75 width=64)

Leitura da tabela historico de um registro cada vez para serem colocados na memória em uma estrutura hash para que depois, ao ler a segunda tabela de cliente, é comparado a correspondência de cada registro com relação às da primeira tabela respeitando a condição (Hash Cond: (c.idcliente = h.idcliente)).

HashAggregate (cost=31.28..32.40 rows=1 width=64)

Agrupamento entre os registros se utilizando como chave o atributo nome da tabela de cliente (Group Key: c.nome).

Subquery Scan on tp (cost=31.28..32.41 rows=1 width=64)

Leitura de subquery de tp.

Hash (cost=32.41..32.41 rows=1 width=64)

Construção e armazenamento dos registros de tp em uma tabela hash.

Seq Scan on cliente (cost=0.00..22.70 rows=1270 width=36)

Leitura sequencial de cliente em disco.

Hash Join (cost=32.42..58.52 rows=6 width=68)

Leitura da tabela cliente de um registro cada vez para serem colocados na memória em uma estrutura hash para que depois, ao ler a segunda tabela de tp, é comparado a correspondência de cada registro com relação às da primeira tabela respeitando a condição (Hash Cond: ((cliente.nome)::text = (tp.nome)::text)).

Nested Loop (cost=32.57..59.60 rows=1 width=68)

Leitura de um outer table, e para cada registro do outer table que respeite a condição, há leitura de um registro de uma inner table.

Nested Loop (cost=32.72..60.41 rows=1 width=68)

Leitura de um outer table, e para cada registro do outer table que respeite a condição, há leitura de um registro de uma inner table.

Nested Loop (cost=32.87..60.64 rows=1 width=96)

Leitura de um outer table, e para cada registro do outer table que respeite a condição, há leitura de um registro de uma inner table.

Sort (cost=60.65..60.66 rows=1 width=96)

Organização física da tabela usando a chave como critério (Sort Key: cliente.nome, f.nome)

Seq Scan on historico h_1 (cost=0.00..1.75 rows=75 width=36)

Leitura sequencial de historico em disco.

Hash (cost=1.75..1.75 rows=75 width=36)

Construção e armazenamento dos registros de historico em uma tabela hash.

Seq Scan on cliente c_1 (cost=0.00..22.70 rows=1270 width=36)

Leitura sequencial de cliente em disco.

Hash Join (cost=2.69..30.90 rows=75 width=64)

Leitura da tabela cliente de um registro cada vez para serem colocados na memória em uma estrutura hash para que depois, ao ler a segunda tabela de historico, é comparado a correspondência de cada registro com relação às da primeira tabela respeitando a condição (Hash Cond: (c_1.idcliente = h_1.idcliente)).

HashAggregate (cost=31.28..32.21 rows=75 width=64)

Agrupamento entre todos registros (Group Key: c_1.nome).

Subquery Scan on temp (cost=31.28..32.96 rows=75 width=32)

Leitura de subquery de temp.

Sort (cost=33.34..33.53 rows=75 width=32)

Organização física da tabela usando a chave como critério (Sort Key: temp.dinheiro_gasto DESC)

Limit (cost=33.34..33.34 rows=1 width=32)

Limite de custo estimado 33.34.