**b) (artefato do tipo texto) Para cada consulta definida na Parte I do trabalho, o grupo deverá executar o comando EXPLAIN a fim de solicitar ao SGBD que mostre o plano de consulta que foi criado para execução da consulta, os índices usados na execução da consulta e o custo das operações realizadas durante a execução da consulta. O documento entregue pelo grupo neste quesito deve contar com informações fornecidas pelo SGBD (relatórios ou screenshots) e uma análise circunstanciada feita pelo grupo sobre a execução da consulta. Nesta análise, o grupo deve interpretar as informações fornecidas pelo SGBD e relacioná-las com a teoria de banco de dados discutida em aula e presente nos livros textos da disciplina e em bibliografia correlata encontrada pelo próprio grupo em pesquisa em bases bibliográficas . O texto deve ser entregue em formato PDF. O código deverá ser entregue em formato**

**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."**

```
"QUERY PLAN"
"Sort  (cost=403.29..410.23 rows=2775 width=44)"
"  Sort Key: produto.dataadquirida"
"  ->  Merge Join  (cost=146.54..244.59 rows=2775 width=44)"
"        Merge Cond: (((SubPlan 1)) = historico.idproduto)"
"        ->  Sort  (cost=52.67..54.52 rows=740 width=9)"
"              Sort Key: ((SubPlan 1))"
"              ->  Seq Scan on produto  (cost=0.00..17.40 rows=740 width=9)"
"                    SubPlan 1"
"                      ->  Result  (cost=0.01..0.01 rows=1 width=4)"
"                                  One-Time Filter: (produto.vendido AND (produto.dataadquirida >=
'2017-01-01'::date) AND (produto.dataadquirida <= '2020-12-31'::date))"
"        ->  Sort  (cost=93.87..95.75 rows=750 width=40)"
"              Sort Key: historico.idproduto"
"              ->  Hash Join  (cost=38.58..58.05 rows=750 width=40)"
"                    Hash Cond: (historico.idcliente = cliente.idcliente)"
"                    ->  Seq Scan on historico  (cost=0.00..17.50 rows=750 width=8)"
"                    ->  Hash  (cost=22.70..22.70 rows=1270 width=36)"
"                          ->  Seq Scan on cliente  (cost=0.00..22.70 rows=1270 width=36)"
```

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

Lê do disco como entrada a tabela de clientes cada tupla relevante para a consulta de uma forma sequencial. O tempo deste é o maior dentre todos os scans feitos já que ele é relativo ao quanto de registros foram envolvidos. No caso nesta operação houveram 1270, explicando o maior tempo em scan da consulta.

Hash  (cost=22.70..22.70 rows=1270 width=36)

Escreve na forma de hash que foi lido anteriormente, tendo um custo maior de processamento.

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

Lê como entrada a tabela de historico cada tupla relevante para a consulta, mas consome pouco do tempo de processamento total.

Hash Join  (cost=38.58..58.05 rows=750 width=40)

Leitura do hash feito e incluindo os registros e utilizando das suas chaves para se encontrar as tuplas que respeitem a condição na tabela (“Hash Cond: (historico.idcliente = cliente.idcliente)”).

Sort  (cost=93.87..95.75 rows=750 width=40)

Reorganização da ordem física dos dados da tabela de cliente em disco. Portanto este acaba exigindo de muitas operações de leitura e escrita em disco, o que explicaria o maior tempo de processamento até então.

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

Selecionando em produtos aqueles que respeitem as condições (produto.vendido AND (produto.dataadquirida >= '2017-01-01'::date) AND (produto.dataadquirida <= '2020-12-31'::date).

Seq Scan on produto  (cost=0.00..17.40 rows=740 width=9)

Lê do disco como entrada a tabela de produto cada tupla relevante para a consulta de uma forma sequencial.

Sort  (cost=52.67..54.52 rows=740 width=9)

Reorganização da ordem física dos dados da tabela de produto em disco. Portanto este acaba exigindo de muitas operações de leitura e escrita em disco.

Merge Join  (cost=146.54..244.59 rows=2775 width=44)

Junção de duas tabelas carregadas por uma chave comum (Merge Cond: (((SubPlan 1)) = historico.idproduto)).

Sort  (cost=403.29..410.23 rows=2775 width=44)

Reorganização da ordem física dos dados da tabela de produto em disco. Portanto este acaba exigindo de muitas operações de leitura e escrita em disco, se utilizando de uma chave como critério de organização (Sort Key: produto.dataadquirida).

**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.”**

Expect:

```
"QUERY PLAN"
"Limit  (cost=16443.39..16443.39 rows=1 width=44)"
"  ->  Sort  (cost=16443.39..16443.40 rows=4 width=44)"
"        Sort Key: (count(fornecedores.idfornecedor)) DESC"
"        ->  GroupAggregate  (cost=0.15..16443.37 rows=4 width=44)"
"              Group Key: fornecedores.idfornecedor"
"              ->  Nested Loop  (cost=0.15..16443.31 rows=4 width=36)"
"                    Join Filter: (fornecedores.idfornecedor = produto.idfornecedor)"
"                          ->  Index Scan using id_fornecedor_id on fornecedores  (cost=0.15..60.30 rows=810 width=36)"
"                    ->  Materialize  (cost=0.00..16334.42 rows=4 width=4)"
"                          ->  Seq Scan on produto  (cost=0.00..16334.40 rows=4 width=4)"
"                                Filter: (idproduto = (SubPlan 1))"
"                                SubPlan 1"
"                                  ->  HashSetOp Except  (cost=0.00..22.05 rows=1 width=8)"
"                                        ->  Append  (cost=0.00..21.12 rows=371 width=8)"
```

```
"                                    ->  Subquery Scan on ""*SELECT* 1""  (cost=0.00..0.02
rows=1 width=8)"
"                                  ->  Result  (cost=0.00..0.01 rows=1 width=4)"
"                                        One-Time Filter: produto.vendido"
"                                    ->  Subquery Scan on ""*SELECT* 2""  (cost=0.00..21.10
rows=370 width=8)"
"                                          ->  Seq Scan on produto produto_1  (cost=0.00..17.40
rows=370 width=4)"
"                                          Filter: (NOT devolvido)"
```

Filter: (NOT devolvido)
filtro para False atributo devolvido
Seq Scan on produto produto_1  (cost=0.00..17.40 rows=370 width=4)
Em Seq scan, scan do banco de dados onde cada linha da tabela é lida em ordem sequencial (serial) e as colunas encontradas tem validade de condição checada, seu output é o idproduto ,o custo estimado para a primeira linha é de 0.00, e custo total 17.40
Subquery Scan on ""*SELECT* 2""  (cost=0.00..21.10 rows=370 width=8)
Subquery scan é uma consulta dentro de outra consulta no caso é o SELECT idproduto FROM produto WHERE devolvido = 'f'
custo total estimado 0.00 e custo total 21.10
One-Time Filter: produto.vendido
Filtra se o produto foi vendido é verdadeiro
Result  (cost=0.00..0.01 rows=1 width=4)
Computa o resultado de Subquery scan passado, tem output idproduto de produto
Subquery Scan on ""*SELECT* 1""  (cost=0.00..0.02 rows=1 width=8)
Subquery scan de custo comSELECT idproduto
Append  (cost=0.00..21.12 rows=371 width=8)
Controla o buffer e tem custo total estimado de 0.00 e custo total de 21.12
HashSetOp Except  (cost=0.00..22.05 rows=1 width=8)
HashSet para comando Except tem custo total estimado de 0.00 e custo total de 21.05
SubPlan 1
Filter: (idproduto = (SubPlan 1))
Filtro que procura o subplano
Seq Scan on produto  (cost=0.00..16334.40 rows=4 width=4)
Scan sequencial de produto com custo total estimado de 0.00 e custo total de 16334.40
Materialize  (cost=0.00..16334.42 rows=4 width=4)
No materialize o output das ações seguintes são materializadas na memória com custo total estimado de 0.00 e custo total de 16334.42
Index Scan using id_fornecedor_id on fornecedores  (cost=0.15..60.30 rows=810 width=36)
Scan de index de fornecedores com id de fornecedor, custo total estimado de 0.15 e custo total de 60.30
Join Filter: (fornecedores.idfornecedor = produto.idfornecedor)
Filtro join
Nested Loop  (cost=0.15..16443.31 rows=4 width=36)
Loop de custo total estimado de 0.15 e custo total de 1644.31
Group Key: fornecedores.idfornecedor
chave escolhida para agrupar fornecedores.idfornecedor

GroupAggregate  (cost=0.15..16443.37 rows=4 width=44)
custo de agregação de grupo total estimado de 0.15 e custo total de 1644.37

Sort Key: (count(fornecedores.idfornecedor)) DESC
A chave utilizada para organizar a tabela é (count(fornecedores.idfornecedor)) em ordem decrescente.
Sort  (cost=16443.39..16443.40 rows=4 width=44)
Custo total estimado de 16443.39 e total estimado da operação Sort é de 16443.40
"Limit  (cost=16443.39..16443.39 rows=1 width=44)"
Como há Limit nem todas as colunas devem ser adquiridas, o custo mostrado 16443.39 é o custo total estimado junto com o número total de linhas que é igual a 1 como se tivesse adquirido todo o scan

O custo total estimado é cumulativo.

No materialize os dados são armazenados no index definido pela chave primária,

**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)."**

```
"QUERY PLAN"
"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: ((status = 'T'::bpchar) AND (idtecnico = 7))"
```

Seq Scan on servico s  (cost=0.00..20.80 rows=1 width=68)
Lê do disco como entrada a tabela de serviço cada tupla relevante para a consulta de uma forma sequencial. Para a filtragem, os registros precisavam entrar em duas condições ((status = 'T'::bpchar) AND (idtecnico = 7)).
Hash  (cost=20.80..20.80 rows=1 width=68)
Constrói a estrutura de armazenamento dos dados da tabela de serviço lidas anteriormente respeitando as condições anteriores.
Seq Scan on servico  (cost=0.00..17.20 rows=720 width=64)
Lê do disco como entrada a tabela de serviço cada tupla relevante para a consulta. O interessante desse scan com relação ao último é que mesmo o resultado da seleção daqui ter mais tuplas (720) se comparados com o scan anterior (1), o tempo que levou aqui também foi menor, levando a crer que a diferença foi ao fato das comparações feitas no scan anterior. Então a cada tupla analisada na tabela, mesmo sendo a mesma, no scan anterior houve a etapa de verificação das condições.
HashAggregate  (cost=20.80..22.80 rows=200 width=64)

Agrupamento entre os registros se utilizando como chave um atributo da tabela de serviço (Group Key: servico.tipo). O que explicaria o custo seria a reorganização de dados em hash e da quantidade de dados.

Hash Join  (cost=41.61..47.12 rows=1 width=68)

Leitura do hash feito e incluindo os registros e utilizando das suas chaves para se encontrar as tuplas que respeitem a condição na tabela (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."**

```
"QUERY PLAN"
"HashAggregate  (cost=262.52..270.02 rows=750 width=96)"
"  Group Key: cliente.nome, f.nome, tp.dinheiro_gasto"
"  InitPlan 1 (returns $0)"
"    ->  Aggregate  (cost=66.80..66.81 rows=1 width=32)"
"          ->  HashAggregate  (cost=61.80..64.30 rows=200 width=64)"
"                Group Key: c_1.nome"
"                ->  Hash Join  (cost=38.58..58.05 rows=750 width=64)"
"                      Hash Cond: (h_1.idcliente = c_1.idcliente)"
"                      ->  Seq Scan on historico h_1  (cost=0.00..17.50 rows=750 width=36)"
"                      ->  Hash  (cost=22.70..22.70 rows=1270 width=36)"
"                            ->  Seq Scan on cliente c_1  (cost=0.00..22.70 rows=1270 width=36)"
"  ->  Hash Join  (cost=164.63..190.08 rows=750 width=96)"
"        Hash Cond: ((cliente.nome)::text = (tp.nome)::text)"
"        ->  Hash Join  (cost=93.45..116.89 rows=750 width=64)"
"              Hash Cond: (historico.idcliente = cliente.idcliente)"
"              ->  Hash Join  (cost=54.88..76.33 rows=750 width=36)"
"                    Hash Cond: (p.idfornecedor = f.idfornecedor)"
"                    ->  Hash Join  (cost=26.65..46.13 rows=750 width=8)"
"                          Hash Cond: (historico.idproduto = p.idproduto)"
"                          ->  Seq Scan on historico  (cost=0.00..17.50 rows=750 width=8)"
"                          ->  Hash  (cost=17.40..17.40 rows=740 width=8)"
"                                ->  Seq Scan on produto p  (cost=0.00..17.40 rows=740 width=8)"
"                    ->  Hash  (cost=18.10..18.10 rows=810 width=36)"
"                          ->  Seq Scan on fornecedores f  (cost=0.00..18.10 rows=810 width=36)"
"              ->  Hash  (cost=22.70..22.70 rows=1270 width=36)"
"                    ->  Seq Scan on cliente  (cost=0.00..22.70 rows=1270 width=36)"
"        ->  Hash  (cost=68.68..68.68 rows=200 width=64)"
"              ->  Subquery Scan on tp  (cost=63.68..68.68 rows=200 width=64)"
"                    ->  HashAggregate  (cost=63.68..66.68 rows=200 width=64)"
"                          Group Key: c.nome"
"                          Filter: (sum(h.valorpago) = $0)"
"                          ->  Hash Join  (cost=38.58..58.05 rows=750 width=64)"
"                                Hash Cond: (h.idcliente = c.idcliente)"
"                                ->  Seq Scan on historico h  (cost=0.00..17.50 rows=750 width=36)"
"                                ->  Hash  (cost=22.70..22.70 rows=1270 width=36)"
"                                      ->  Seq Scan on cliente c  (cost=0.00..22.70 rows=1270 width=36)"
```

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

Leitura em disco da tabela de cliente.

Hash  (cost=22.70..22.70 rows=1270 width=36)

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

Seq Scan on historico h  (cost=0.00..17.50 rows=750 width=36)

Leitura em disco da tabela de histórico.

Hash Join  (cost=38.58..58.05 rows=750 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: (h.idcliente = c.idcliente).

HashAggregate  (cost=63.68..66.68 rows=200 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=63.68..68.68 rows=200 width=64)

Leitura de uma saída da subquery chamada de tp. Ou seja, tp é o resultado das operações de Hash Join e HashAggregate feitas anteriormente envolvendo as tabelas de cliente e histórico.

Hash  (cost=68.68..68.68 rows=200 width=64)

Construção da estrutura em hash para os dados lidos anteriormente (tp).

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

Leitura da tabela cliente do disco.

Hash  (cost=22.70..22.70 rows=1270 width=36)

Constrói e preenche uma estrutura de armazenamento para os dados em hash com os dados lidos da tabela cliente anteriormente.

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

Leitura da tabela de fornecedores do disco. O custo em tempo de processamento é menor, relativo ao tamanho da tabela.

Hash  (cost=18.10..18.10 rows=810 width=36)

Constrói e preenche uma estrutura de armazenamento para os dados em hash com os dados lidos da tabela fornecedores anteriormente.

Seq Scan on produto p  (cost=0.00..17.40 rows=740 width=8)

Leitura da tabela de produto do disco.

Hash  (cost=17.40..17.40 rows=740 width=8)

Constrói e preenche uma estrutura de armazenamento para os dados em hash com os dados lidos da tabela produto anteriormente.

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

Leitura da tabela de histórico do disco.

Hash Join  (cost=26.65..46.13 rows=750 width=8)

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 produto, é comparado a correspondência de cada registro com relação às da primeira tabela respeitando a condição (Hash Cond: (historico.idproduto = p.idproduto)).

Hash Join  (cost=54.88..76.33 rows=750 width=36)

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

Hash Join  (cost=93.45..116.89 rows=750 width=64)

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: (historico.idcliente = cliente.idcliente)).

Hash Join  (cost=164.63..190.08 rows=750 width=96)

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)).

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

Leitura da tabela cliente do disco.

Hash  (cost=22.70..22.70 rows=1270 width=36)

Constrói e preenche uma estrutura de armazenamento para os dados em hash com os dados lidos da tabela cliente anteriormente.

Seq Scan on historico h_1  (cost=0.00..17.50 rows=750 width=36)

Leitura da tabela histórica do disco.

Hash Join  (cost=38.58..58.05 rows=750 width=64)

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: (h_1.idcliente = c_1.idcliente)).

HashAggregate  (cost=61.80..64.30 rows=200 width=64)

Agrupamento entre os registros se utilizando como chave o atributo nome da tabela de cliente (Group Key: c_1.nome).

Aggregate  (cost=66.80..66.81 rows=1 width=32)

Agregamento desses vários resultados das várias entradas anteriores.

HashAggregate  (cost=262.52..270.02 rows=750 width=96)

Agrupamento entre todos registros se utilizando como chave o atributo nome da tabela de cliente, o nome do fornecedor e o dinheiro gasto da tabela tp criada. (Group Key: cliente.nome, f.nome, tp.dinheiro_gasto).