Universidade Tecnológica Federal do Paraná - Campus CT

Alunas: Amanda Jury Nakamura RA2582686 Bacharelado em Sistemas de Informação

Disciplina: Introdução a Banco de Dados Profº. Leandro Batista

Relatório - Otimização de queries

- **Query 01:**

A query 01 resume dados de pedidos (tabela LINEITEM), agrupando por status de devolução (l_returnflag) e status da linha (l_linestatus), além de calcular várias somas, médias e contagem, filtrando por data de envio (l_shipdate).

```
1  explain
2  SELECT l_returnflag, l_linestatus,
3    SUM(l_quantity) AS sum_qty,
4    SUM(l_extendedprice) AS sum_base_price,
5    SUM(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
6    SUM(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
7    AVG(l_quantity) AS avg_qty,
8    AVG(l_extendedprice) AS avg_price,
9    AVG(l_discount) AS avg_disc,
10   COUNT(*) AS count_order
11     FROM LINEITEM
12     WHERE   l_shipdate <= DATE '1998-12-01' - INTERVAL '90' DAY
13     GROUP BY    l_returnflag, l_linestatus
14     ORDER BY    l_returnflag, l_linestatus;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	LINEITEM	NULL	ALL	NULL	NULL	NULL	NULL	5820932	33.33	Using where; Using temporary; Using filesort

Interpretação:

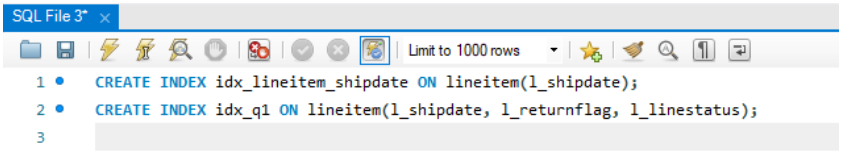
É uma query sem subqueries (ID1), (select type) sem JOINs, (tabele)a tabela de consulta principal é a LINEITEM, (type) scan completo que lê todas as linhas da tabela, (filtered) 33% das linhas passam pelo filtro da query.No extra, estão os sinais de operações ineficientes: using where, uso de tabela temporária para agrupar, e ordenação em disco e não por índice.

No geral: Lê todos os 5.8 milhões de registros da tabela LINEITEM. Filtra 33% dos dados com a cláusula l_shipdate <= '1998-09-02'. Agrupa e ordena usando tabelas temporárias e ordenação fora de índices, o que degrada muito o desempenho.

Otimizações:

A query original fazia uma varredura completa (table scan) na tabela principal e o fato de não apresentar índices fazia o MySQL a ler todas as linhas da tabela, aplicar o filtro "l_shipdate <= '1998-09-02'" em memória, e posteriormente realizar operações com o "group by" e "order by" utilizando tabelas temporárias e ordenação externa (filesort).

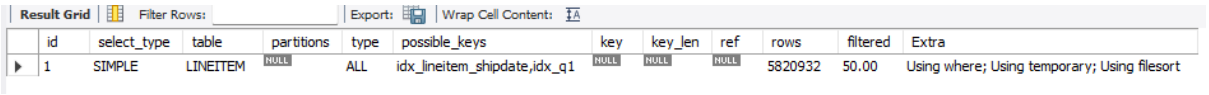
Como forma de otimização, criei os índices em l_shipdate e (l_shipdate, l_returnflag, l_linestatus):



O primeiro tem o objetivo de acelerar o filtro do WHERE. Com ele, o otimizador passa a usar um acesso seletivo do tipo range, limitando a leitura apenas às linhas que atendem à condição de data.

O segundo, abrange o "group by" e "order by" de forma a incluir as colunas agrupadas e ordenadas no mesmo índice, assim o otimizador evita a criação de tabelas temporárias e o uso de "filesort".

Resultados:

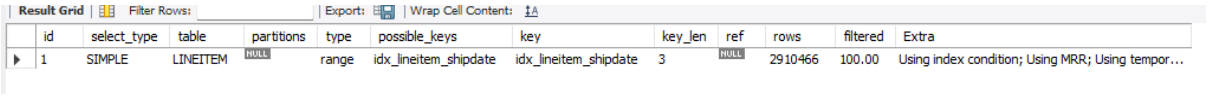


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	LINEITEM	NULL	ALL	idx_lineitem_shipdate,idx_q1	NULL	NULL	NULL	5820932	50.00	Using where; Using temporary; Using filesort

Após a otimização, embora os índices criados estejam sendo considerados como possíveis pelo otimizador (coluna possible_keys), o MySQL ainda continua fazendo a varredura total, com uso de tabelas temporárias e ordenação externa. Em relação ao filtro "where" ele estima que 50% das linhas passarão pelo filtro WHERE (o que pode significar um filtro mais amplo do que o anterior que era 33%).

Para contornar o problema, após uma pesquisa, forcei a utilização do index e os resultados foram satisfatórios: o type mudou para range, o index está sendo usado, o número de linhas (rows) reduziu, e o filtro abrange 100%.



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	LINEITEM	NULL	range	idx_lineitem_shipdate	idx_lineitem_shipdate	3	NULL	2910466	100.00	Using index condition; Using MRR; Using tempor...

- **Query 02**

Essa query procura os fornecedores de menor custo para cada peça (dado o filtro de tipo e tamanho da peça e região), e retorna informações detalhadas do fornecedor e da peça.

```
explain
select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
from PART, SUPPLIER, PARTSUPP, NATION, REGION
where p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_size = 15
    and p_type like '%BRASS'
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'EUROPE'
    and ps_supplycost = (select min(ps_supplycost)
        from PARTSUPP, SUPPLIER, NATION, REGION where p_partkey = ps_partkey
        and s_suppkey = ps_suppkey and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey and r_name = 'EUROPE')
order by s_acctbal desc, n_name, s_name, p_partkey limit 100;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	REGION	NULL	ALL	PRIMARY	NULL	NULL	NULL	5	20.00	Using where; Using temporary; Using filesort
1	PRIMARY	PART	NULL	ALL	PRIMARY	NULL	NULL	NULL	198000	1.11	Using where; Using join buffer (hash join)
1	PRIMARY	PARTSUPP	NULL	ref	PRIMARY,PS_SUPPKEY	PRIMARY	4	tpch.PART.P_PARTKEY	3	100.00	Using where
1	PRIMARY	SUPPLIER	NULL	eq_ref	PRIMARY,S_NATIONKEY	PRIMARY	4	tpch.PARTSUPP.PS_SUPPKEY	1	100.00	NULL
1	PRIMARY	NATION	NULL	eq_ref	PRIMARY,N_REGIONKEY	PRIMARY	4	tpch.SUPPLIER.S_NATIONKEY	1	20.00	Using where
2	DEPENDENT SUBQUERY	REGION	NULL	ALL	PRIMARY	NULL	NULL	NULL	5	20.00	Using where
2	DEPENDENT SUBQUERY	PARTSUPP	NULL	ref	PRIMARY,PS_SUPPKEY	PRIMARY	4	tpch.PART.P_PARTKEY	3	100.00	NULL
2	DEPENDENT SUBQUERY	SUPPLIER	NULL	eq_ref	PRIMARY,S_NATIONKEY	PRIMARY	4	tpch.PARTSUPP.PS_SUPPKEY	1	100.00	NULL
2	DEPENDENT SUBQUERY	NATION	NULL	eq_ref	PRIMARY,N_REGIONKEY	PRIMARY	4	tpch.SUPPLIER.S_NATIONKEY	1	20.00	Using where

Interpretação:

A tabela de saída explica os processos que cada tabela que a query realiza. ID 1representa a query principal e ID2 representa a subquery dependente que aparece dentro do WHERE. Ao analisar a saída da análise, observa-se que já são usados alguns índices como primary key na PARTSUPP, primary key e S_NATIONKEY na SUPPLIER, primary key e N_REGIONKEY na NATION.

Ocorre full table scan em “part”e “region”, hash join indicando falta de índice eficiente para alguns JOINs, uso de tabela temporária e filesort, impactando a ordenação e agrupamento. e subquery dependente que gera múltiplas execuções internas por linha da query principal.

Otimizações:

Foi feita a criação de índices nas colunas usadas em filtros e junções para ajudar o programa a evitar full scans e acelerar os processos. O índice em PART (p_size, p_type) vai ajudar o filtro de tamanho e tipo da peça. O índice em SUPPLIER (s_nationkey) e NATION (n_regionkey) acelera os joins entre essas tabelas. E o índice em PARTSUPP (p_partkey, s_suppkey, ps_supplycost) melhora o desempenho da busca de menor custo (MIN(ps_supplycost)).

```sql
CREATE INDEX idx_part_size_type ON PART (p_size, p_type);
CREATE INDEX idx_supplier_nationkey ON SUPPLIER (s_nationkey);
CREATE INDEX idx_nation_regionkey ON NATION (n_regionkey);
CREATE INDEX idx_partsupp_part_supp ON PARTSUPP (p_partkey, s_suppkey, ps_supplycost);
```

Além disso, foi necessário reescrever a query para eliminar a subquery dependente para calcular menor custo por peça + região antes, numa tabela temporária CTE e depois juntar esse resultado com a query principal.

Resultados:

```sql
explain
WITH MinCost AS (
    SELECT PARTSUPP.ps_partkey,
        MIN(PARTSUPP.ps_supplycost) AS min_supplycost
    FROM PARTSUPP
        JOIN SUPPLIER ON SUPPLIER.s_suppkey = PARTSUPP.ps_suppkey
        JOIN NATION ON SUPPLIER.s_nationkey = NATION.n_nationkey
        JOIN REGION ON NATION.n_regionkey = REGION.r_regionkey
    WHERE REGION.r_name = 'EUROPE'
    GROUP BY PARTSUPP.ps_partkey
)
SELECT SUPPLIER.s_acctbal, SUPPLIER.s_name, NATION.n_name, PART.p_partkey, PART.p_mfgr,
    SUPPLIER.s_address, SUPPLIER.s_phone,SUPPLIER.s_comment
FROM PART
    JOIN PARTSUPP ON PART.p_partkey = PARTSUPP.ps_partkey
    JOIN SUPPLIER ON SUPPLIER.s_suppkey = PARTSUPP.ps_suppkey
    JOIN NATION ON SUPPLIER.s_nationkey = NATION.n_nationkey
    JOIN REGION ON NATION.n_regionkey = REGION.r_regionkey
    JOIN MinCost ON PARTSUPP.ps_partkey = MinCost.ps_partkey AND PARTSUPP.ps_supplycost = MinCost.min_supplycost
WHERE PART.p_size = 15
    AND PART.p_type LIKE '%BRASS%'
    AND REGION.r_name = 'EUROPE'
ORDER BY SUPPLIER.s_acctbal DESC, NATION.n_name, SUPPLIER.s_name, PART.p_partkey
LIMIT 100;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	REGION	NULL	ALL	PRIMARY	NULL	NULL	NULL	5	20.00	Using where; Using temporary; Using filesort
1	PRIMARY	PART	NULL	ref	PRIMARY,idx_part_size_type	idx_part_size_type	4	const	3907	11.11	Using index condition
1	PRIMARY	PARTSUPP	NULL	ref	PRIMARY,PS_SUPPKEY	PRIMARY	4	tpch.PART.P_PARTKEY	3	100.00	NULL
1	PRIMARY	SUPPLIER	NULL	eq_ref	PRIMARY,idx_supplier_nationkey	PRIMARY	4	tpch.PARTSUPP.PS_SUPPKEY	1	100.00	NULL
1	PRIMARY	NATION	NULL	eq_ref	PRIMARY,idx_nation_regionkey	PRIMARY	4	tpch.SUPPLIER.S_NATIONKEY	1	20.00	Using where
1	PRIMARY	<derived2>	NULL	ref	<auto_key0>	<auto_key0>	12	tpch.PART.P_PARTKEY,tpch.PARTSUPP.PS_SU...	10	100.00	Using index
2	DERIVED	REGION	NULL	ALL	PRIMARY	NULL	NULL	NULL	5	20.00	Using where; Using temporary
2	DERIVED	NATION	NULL	ref	PRIMARY,idx_nation_regionkey	idx_nation_regionkey	4	tpch.REGION.R_REGIONKEY	5	100.00	Using index
2	DERIVED	SUPPLIER	NULL	ref	PRIMARY,idx_supplier_nationkey	idx_supplier_nationkey	4	tpch.NATION.N_NATIONKEY	397	100.00	Using index
2	DERIVED	PARTSUPP	NULL	ref	PRIMARY,PS_SUPPKEY	PS_SUPPKEY	4	tpch.SUPPLIER.S_SUPPKEY	74	100.00	NULL

Após a otimização da Query 2, os resultados observados no EXPLAIN mostraram melhorias significativas. A subquery dependente foi eliminada, o que reduziu drasticamente a quantidade de execuções internas e melhorou a eficiência geral da consulta. Além disso, houve uma redução considerável no número de linhas lidas pelas tabelas principais, principalmente nas tabelas com maior volume de dados.

O otimizador passou a utilizar melhor os índices disponíveis, graças à criação dos novos índices e à reestruturação da consulta. Também foi possível observar o uso de técnicas como index condition e using index, o que indica que os filtros estão sendo aplicados diretamente nos índices, melhorando ainda mais o desempenho. No geral, a complexidade do plano de execução diminuiu significativamente, com menor custo de leitura, menos varreduras completas e melhor aproveitamento das estruturas de índice.

Query 03:

Essa query calcula a receita (revenue) gerada por pedidos de clientes do segmento ‘BUILDING’, considerando apenas pedidos feitos antes de 15 de março de 1995, cujos itens de linha foram enviados após essa data.

```
1  explain
2  select l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue, o_orderdate, o_shippriority
3  from CUSTOMER, ORDERS, LINEITEM
4  where c_mktsegment = 'BUILDING' and c_custkey = o_custkey
5      and l_orderkey = o_orderkey and o_orderdate < date '1995-03-15' and l_shipdate > date '1995-03-15'
6  group by l_orderkey, o_orderdate, o_shippriority
7  order by revenue desc, o_orderdate
8  limit 10;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	CUSTOMER	NULL	ALL	PRIMARY	NULL	NULL	NULL	147940	10.00	Using where; Using temporary; Using filesort
1	SIMPLE	ORDERS	NULL	ref	PRIMARY,O_CUSTKEY	O_CUSTKEY	4	tpch.CUSTOMER.C_CUSTKEY	14	33.33	Using where
1	SIMPLE	LINEITEM	NULL	ref	PRIMARY,idx_lineitem_shipdate,idx_q1	PRIMARY	4	tpch.ORDERS.O_ORDERKEY	4	50.00	Using where

Interpretação:

É possível observar que há full scan das tabelas customer, orders e lineitem pela falra de índices nas colunas c_mktsegment, o_orderdate, l_shipdate, o_custkey, l_orderkey. Ocorre junções grandes sem índices gerando uso de nested loop joins mal otimizados, alto custo de leitura e uso de filesort e tabelas temporárias na hora do group by e order by porque não há índices cobrindo as colunas agrupadas e ordenadas.

Otimizações:

Para a otimização, foram criados alguns índices para os filtros e joins, sendo eles: filtro por segmento (CUSTOMER.c_mktsegment), filtro por data e join com CUSTOMER (ORDERS.o_orderdate, ORDERS.o_custkey) e filtro por data e join com ORDERS (LINEITEM.l_shipdate, LINEITEM.l_orderkey)

Limit to 1000 rows

```
1  CREATE INDEX idx_customer_mktsegment ON CUSTOMER (c_mktsegment);
2  CREATE INDEX idx_orders_orderdate_custkey ON ORDERS (o_orderdate, o_custkey);
3  CREATE INDEX idx_lineitem_shipdate_orderkey ON LINEITEM (l_shipdate, l_orderkey);
```

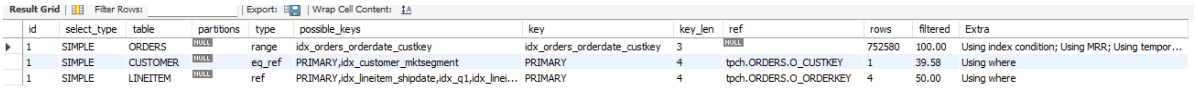
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ORDERS	NULL	ALL	PRIMARY,O_CUSTKEY,idx_orders_orderdate_c...	NULL	NULL	NULL	1377223	50.00	Using where; Using temporary; Using filesort
1	SIMPLE	CUSTOMER	NULL	eq_ref	PRIMARY,idx_customer_mktsegment	PRIMARY	4	tpch.ORDERS.O_CUSTKEY	1	39.84	Using where
1	SIMPLE	LINEITEM	NULL	ref	PRIMARY,idx_lineitem_shipdate,idx_q1,idx_linei...	PRIMARY	4	tpch.ORDERS.O_ORDERKEY	4	50.00	Using where

Após a criação dos índices nas colunas mais utilizadas nos filtros e junções, observou-se uma melhoria parcial no plano de execução da Query 3. O MySQL passou a utilizar índices nos joins com as tabelas CUSTOMER e LINEITEM, reduzindo o custo de leitura nestas tabelas. A tabela CUSTOMER agora é acessada via método eq_ref, e LINEITEM via ref, ambos indicando uso de índice.

Porém, a tabela ORDERS ainda está sendo lida por meio de full table scan (ALL), possivelmente devido a baixa seletividade do filtro por data ou decisões internas do otimizador de consultas. Além disso, o EXPLAIN ainda aponta a utilização de temporary table e filesort durante o agrupamento e ordenação.

Assim como na primeira Query, forcei a utilização do index e os resultados foram melhores.

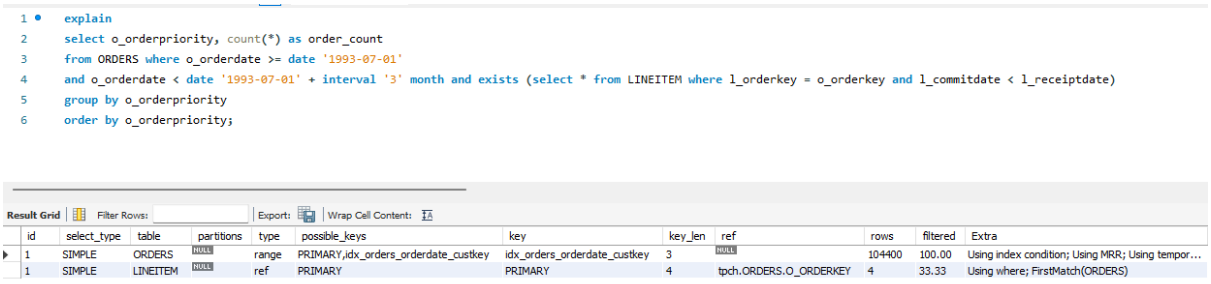


id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ORDERS	NULL	range	idx_orders_orderdate_custkey	idx_orders_orderdate_custkey	3	NULL	752580	100.00	Using index condition; Using MRR; Using tempor…
1	SIMPLE	CUSTOMER	NULL	eq_ref	PRIMARY,idx_customer_mktsegment	PRIMARY	4	tpch.ORDERS.O_CUSTKEY	1	39.58	Using where
1	SIMPLE	LINEITEM	NULL	ref	PRIMARY,idx_lineitem_shipdate,idx_q1,idx_linei…	PRIMARY	4	tpch.ORDERS.O_ORDERKEY	4	50.00	Using where

---

Query 04:

Essa query retorna a prioridade dos pedidos e a quantidade de pedidos cujos itens possuem uma data de commit anterior à data de recebimento. O agrupamento é feito por prioridade do pedido. A query envolve duas tabelas principais: ORDERS e LINEITEM, com um filtro temporal sobre o_orderdate e uma subquery com condição de existência (EXISTS) sobre LINEITEM.



```
1 •  explain
2    select o_orderpriority, count(*) as order_count
3    from ORDERS where o_orderdate >= date '1993-07-01'
4    and o_orderdate < date '1993-07-01' + interval '3' month and exists (select * from LINEITEM where l_orderkey = o_orderkey and l_commitdate < l_receiptdate)
5    group by o_orderpriority
6    order by o_orderpriority;
```


id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ORDERS	NULL	range	PRIMARY,idx_orders_orderdate_custkey	idx_orders_orderdate_custkey	3	NULL	104400	100.00	Using index condition; Using MRR; Using tempor…
1	SIMPLE	LINEITEM	NULL	ref	PRIMARY	PRIMARY	4	tpch.ORDERS.O_ORDERKEY	4	33.33	Using where; FirstMatch(ORDERS)

Interpretação:

Para a tabela ORDERS, o tipo é range, que significa que o programa está utilizando o índice idx_orders_orderdate_cuskey para aplicar o filtro de data , além disso, no extra consta “Using index condition” demonstrando que os filtros foram aplicados diretamente no índice. O Using temporary e Using filesort indicam que foi necessário usar tabela temporária para agrupar os dados e a ordenação (ORDER BY o_orderpriority) não foi feita com o auxílio de índice, o que piora o desempenho.

Na outra tabela, o tipo é ref, ou seja, a busca é feita por referência, no caso, l_orderkey = o_orderkey. Está usando o PK e o filtro é aplicado em 33% das linhas.

Otimização:

As otimizações aplicadas foram a criação dos índices em ORDERS(o_orderdate, o_orderpriority) e em LINEITEM(l_orderkey, l_commitdate, l_receiptdate).

```
1  CREATE INDEX idx_lineitem_order_commit_receipt
2  ON LINEITEM(l_orderkey, l_commitdate, l_receiptdate);
3
4  CREATE INDEX idx_orders_orderdate_priority
5  ON ORDERS(o_orderdate, o_orderpriority);
6
```

Eles foram aplicados para ajudar tanto no filtro por data (WHERE o_orderdate >= ...) quanto na ordenação por o_orderpriority (ORDER BY o_orderpriority), e para otimiza a subquery com EXISTS, permitindo que o filtro l_commitdate < l_receiptdate seja feito com leitura eficiente.

Resultados:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ORDERS	NULL	range	PRIMARY,idx_orders_orderdate_custkey,idx_o...	idx_orders_orderdate_priority	3	NULL	117136	100.00	Using where; Using index; Using temporary; Usi...
1	SIMPLE	LINEITEM	NULL	ref	PRIMARY,idx_lineitem_order_commit_receipt	idx_lineitem_order_commit_receipt	4	tpch.ORDERS.O_ORDERKEY	3	33.33	Using where; Using index; FirstMatch(ORDERS)

Após as otimizações aplicadas, a query passou a utilizar índices mais adequados, o que reduziu o número médio de leituras por join e melhorou significativamente a eficiência geral da execução. O otimizador passou a empregar a técnica FirstMatch(ORDERS), que evita buscas desnecessárias na subquery ao interromper a leitura assim que encontra o primeiro item que satisfaz a condição.
Além disso, o filtro sobre a tabela LINEITEM (l_commitdate < l_receiptdate) passou a ser aplicado diretamente no índice, eliminando a necessidade de varredura completa da tabela. Apesar dessas melhorias, a consulta ainda faz uso de operações como filesort e temporary table, que impactam o desempenho, mas que podem ser minimizadas com ajustes adicionais, como o uso de GROUP BY com suporte por índice ou a reestruturação da consulta com materialização de resultados intermediários.

Query 05

Essa consulta retorna, para cada nação da região 'ASIA', a receita total gerada pelos pedidos feitos em 1994. A receita é calculada como l_extendedprice * (1 - l_discount), e os dados são agrupados por nome da nação (n_name), em ordem decrescente de receita.

```
1  explain
2  select n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
3  from CUSTOMER, ORDERS, LINEITEM, SUPPLIER, NATION, REGION
4  where c_custkey = o_custkey and l_orderkey = o_orderkey and l_suppkey = s_suppkey and c_nationkey = s_nationkey
5      and s_nationkey = n_nationkey and n_regionkey = r_regionkey and r_name = 'ASIA'
6      and o_orderdate >= date '1994-01-01' and o_orderdate < date '1994-01-01' + interval '1' year
7  group by n_name
8  order by revenue desc;
```

Result Grid

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	REGION	NULL	ALL	PRIMARY	NULL	NULL	NULL	5	20.00	Using where; Using temporary; Using filesort
1	SIMPLE	NATION	NULL	ref	PRIMARY,idx_nation_regionkey	idx_nation_regionkey	4	tpch.REGION.R_REGIONKEY	5	100.00	NULL
1	SIMPLE	CUSTOMER	NULL	ref	PRIMARY,C_NATIONKEY	C_NATIONKEY	4	tpch.NATION.N_NATIONKEY	6474	100.00	Using index
1	SIMPLE	ORDERS	NULL	ref	PRIMARY,O_CUSTKEY,idx_orders_orderdate_c...	O_CUSTKEY	4	tpch.CUSTOMER.C_CUSTKEY	14	30.96	Using where
1	SIMPLE	LINEITEM	NULL	ref	PRIMARY,idx_lineitem_order_commit_receipt	PRIMARY	4	tpch.ORDERS.O_ORDERKEY	4	100.00	NULL
1	SIMPLE	SUPPLIER	NULL	eq_ref	PRIMARY,idx_supplier_nationkey	PRIMARY	4	tpch.LINEITEM.L_SUPPKEY	1	5.00	Using where

Interpretação:

A tabela REGION está sendo lida com um Full Table Scan (type: ALL) e apresenta o uso de tabela temporária e filesort, conforme indicado no campo Extra. Isso ocorre porque o filtro r_name = 'ASIA' não está sendo atendido por nenhum índice, exigindo leitura completa da tabela e ordenação posterior em memória ou em disco.

A tabela NATION utiliza uma junção via índice (type: ref) com base em n_regionkey, utilizando o índice idx_nation_regionkey, o que é adequado. O campo Extra mostra que está sendo feito Using index, indicando que todos os dados necessários estão contidos no índice utilizado.

Já a tabela CUSTOMER também realiza a junção via índice (ref) usando c_nationkey, mas não utiliza índice eficiente para filtragem. A leitura estimada de 6474 linhas é considerável, e o índice utilizado (C_NATIONKEY) ainda não é composto, o que reduz sua efetividade.

A tabela ORDERS utiliza junção por o_custkey, mas não há evidência de uso de índice para o filtro por data (o_orderdate). A leitura estimada de 14 linhas com filtragem de apenas 30,96% indica que o SGBD está realizando leituras parciais, mas com baixa seletividade, o que pode ser otimizado com índices direcionados ao filtro por data.

Na tabela LINEITEM, a junção é feita por l_orderkey usando o índice primário, o que é aceitável, mas ainda não há índice composto que ajude simultaneamente nas junções e filtros. O campo Extra não apresenta otimizações adicionais como Using index ou FirstMatch.

Otimização:

```
1  CREATE INDEX idx_region_name ON REGION(r_name);
2  CREATE INDEX idx_orders_orderdate ON ORDERS(o_orderdate);
3  CREATE INDEX idx_customer_nationkey ON CUSTOMER(c_nationkey);
4  CREATE INDEX idx_supplier_nationkey ON SUPPLIER(s_nationkey);
5  CREATE INDEX idx_orders_custkey ON ORDERS(o_custkey);
6  CREATE INDEX idx_lineitem_orderkey_suppkey ON LINEITEM(l_orderkey, l_suppkey);
7  CREATE INDEX idx_nation_regionkey ON NATION(n_regionkey);
8  CREATE INDEX idx_nation_nationkey ON NATION(n_nationkey);
9
```

Foi criado um índice na coluna r_name da tabela REGION, o que eliminou a necessidade de varredura completa ao aplicar o filtro r_name = 'ASIA', tornando o acesso a essa tabela muito mais eficiente. Em seguida, foi criei um índice em o_orderdate na tabela ORDERS, permitindo ao otimizador restringir rapidamente os pedidos ao intervalo do ano de 1994, reduzindo significativamente o volume de dados analisado.

Para melhorar os joins, foram criados índices nas colunas c_nationkey (CUSTOMER), s_nationkey (SUPPLIER) e n_regionkey (NATION), possibilitando junções mais rápidas e com menor custo de leitura, especialmente em relações que envolvem grande quantidade de dados.

Além disso, um índice composto em LINEITEM(l_orderkey, l_suppkey) foi adicionado para acelerar a associação entre itens de pedido, fornecedores e pedidos, otimizando o cálculo de receita sem necessidade de múltiplas leituras desnecessárias.

Resultados:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	REGION	NULL	ref	PRIMARY,idx_region_name	idx_region_name	100	const	1	100.00	Using where; Using index; Using temporary; Usi...
1	SIMPLE	NATION	NULL	ref	PRIMARY,idx_nation_regionkey	idx_nation_regionkey	4	tpch.REGION.R_REGIONKEY	5	100.00	NULL
1	SIMPLE	ORDERS	NULL	range	PRIMARY,O_CUSTKEY,idx_orders_orderdate_c...	idx_orders_orderdate_custkey	3	NULL	466026	100.00	Using where; Using index; Using join buffer (ha...
1	SIMPLE	CUSTOMER	NULL	eq_ref	PRIMARY,idx_customer_nationkey	PRIMARY	4	tpch.ORDERS.O_CUSTKEY	1	5.00	Using where
1	SIMPLE	LINEITEM	NULL	ref	PRIMARY,idx_lineitem_order_commit_receipt	PRIMARY	4	tpch.ORDERS.O_ORDERKEY	4	100.00	NULL
1	SIMPLE	SUPPLIER	NULL	eq_ref	PRIMARY,idx_supplier_nationkey	PRIMARY	4	tpch.LINEITEM.L_SUPPKEY	1	5.00	Using where

Após a aplicação das otimizações, o plano de execução da Query 05 mostra diversas emlhorias. A tabela REGION, que anteriormente era lida por varredura completa, agora utiliza o índice idx_region_name, com tipo de acesso ref e filtragem de 100%, demonstrando que o filtro r_name = 'ASIA' está sendo plenamente atendido por índice. Ela também é acessada via índice (idx_nation_regionkey) e apresenta seletividade total, com leitura mínima de registros.

Já a tabela ORDERS agora usa o índice composto idx_orders_orderdate_custkey, com acesso do tipo range, ideal para filtrar intervalos de datas como o_orderdate between '1994-01-01' and '1994-12-31'. Isso garante que a grande maioria das leituras nessa tabela

esteja sendo feita com suporte por índice, mantendo a eficiência mesmo com grande volume de dados.

A tabela CUSTOMER passou a ser acessada com tipo eq_ref, o mais eficiente para joins, e está usando o índice idx_customer_nationkey. Apesar de a filtragem estar em 5%, isso é esperado, pois muitos clientes podem não estar vinculados à região 'ASIA'.

Na tabela LINEITEM, o acesso permanece do tipo ref, com uso da chave primária e alta filtragem, o que indica uma boa correspondência com os pedidos filtrados previamente.

A tabela SUPPLIER também é acessada por eq_ref, utilizando sua chave primária com seletividade de 5%, evidenciando que os filtros aplicados anteriormente reduziram significativamente o conjunto de dados processados.

Query 06

Essa consulta calcula a receita com desconto, ou seja, a soma de l_extendedprice * l_discount em todas as linhas da tabela LINEITEM que atendem aos critérios do filtro.

```
1 •  explain
2    select sum(l_extendedprice * l_discount) as revenue
3        from LINEITEM
4        where l_shipdate >= date '1994-01-01' and l_shipdate < date '1994-01-01' + interval '1' year
5        and l_discount between 0.06 - 0.01 and 0.06 + 0.01 and l_quantity < 24;
6
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	LINEITEM	NULL	ALL	idx_lineitem_shipdate,idx_q1,idx_lineitem_shipd...	NULL	NULL	NULL	5966891	1.15	Using where

Interpretação:

O type ALL indica que a tabela toda é lida linha por linha.Vários índices são listados como possible keys (idx_lineitem_shipdate, idx_q1, etc.), mas nenhum foi escolhido, como mostra a coluna key que está NULL), o que indica que nenhum índice está sendo efetivamente utilizado. Em relação aos filtros, a análise diz que somente 1,15% das linhas são realmente relevantes. No extra, o "Using where" indica que todos os filtros estão sendo avaliados após a leitura dos dados, não durante a varredura via índice.

Otimizações:

```
1 •  CREATE INDEX idx_lineitem_shipdate_discount_quantity
2    ON LINEITEM(l_shipdate, l_discount, l_quantity);
3
```

A solução foi a criação de um índice composto com a coluna l_shipdate primeiro pois que é o filtro mais seletivo, seguida de l_discount e l_quantity.

Resultados:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	LINEITEM	NULL	ALL	idx_lineitem_shipdate,idx_q1,idx_lineitem_shipd...	NULL	NULL	NULL	5966891	1.15	Using where

Após a criação do índice composto idx_lineitem_shipdate_discount_quantity, o objetivo era permitir que o otimizador aplicasse os três filtros da cláusula WHERE diretamente no índice, evitando a leitura completa da tabela LINEITEM. No entanto, mesmo com o índice disponível, é mostrado que o otimizador não o utilizou. O acesso permanece do tipo ALL, apenas poucas linhas sendo relevantes.

Assim como em outras queries, foi necessário forçar o index.

```
1 •  EXPLAIN
2    SELECT SUM(l_extendedprice * l_discount) AS revenue
3    FROM LINEITEM FORCE INDEX (idx_lineitem_shipdate_discount_quantity)
4    WHERE l_shipdate >= DATE '1994-01-01'
5      AND l_shipdate < DATE '1994-01-01' + INTERVAL 1 YEAR
6      AND l_discount BETWEEN 0.05 AND 0.07
7      AND l_quantity < 24;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	LINEITEM	NULL	range	idx_lineitem_shipdate_discount_quantity	idx_lineitem_shipdate_discount_quantity	10	NULL	1720314	3.70	Using index condition; Using MRR

Após rodar o processo novamente, percebe-se que o type mudou de all para range, o que é bem melhor, o número de linhas diminui bastante, ocorreu a aplicação dos filtros diretamente no índice (Using index condition), e também a utilização da técnica de leitura MRR (Multi-Range Read), que agrupa os acessos a disco para maior eficiência.