Descrição Textual

Uma classe de usuários com a permissão do retail, dos vendedores e da **fabricante intel**, foi designada como administradora de vendas daquela fabricante. Por motivos de segurança, essa classe não pode ter acesso a vendas de produtos de outras fabricantes e só pode ver **vendas com datas de até 90 dias anteriores**, que é a data máxima para devolução dos produtos. Foi proposta uma visão que particione a tabela **Operacao_venda**, trazendo todos os campos das vendas de fabricante intel no período mencionado a partir da data de acesso a visão.

Query em SQL Padrão

```
CREATE VIEW vendas_recentes_intel AS

SELECT
ov.*,
DATE_DIFF(CURRENT_DATE, ov.data_venda, DAY)

FROM
Produto p

INNER JOIN Fabricante fab
ON p.fk_fabricante_id_fab = fab.id_fab

INNER JOIN Operacao_venda ov
ON p.id_produto = ov.fk_produto_id_produto

WHERE
nome_fab = 'intel'
AND
DATE_DIFF(CURRENT_DATE, ov.data_venda, DAY) < 90

ORDER BY
data_venda DESC;
```

Crie uma view vendas_recentes_intel que seleciona as vendas e a diferença de datas de venda da tabela de produto que cruza com a tabela fabricante e a operação_venda, na qual o nome da fabricante seja igual a Intel e que diferença de datas sejam de três meses(dia de hoje – 90 dias) e orderne pela data de venda de forma decrescente.

Materialização

Essa visão é "quente" na escrita, já que muitas vendas são feitas por dia e consequentemente muitas atualizações são necessárias para mantê-la atualizada, cada venda é feita necessariamente no dia mais atual, vulgo o dia atual. Também é fato que a leitura dela não é muito comum, já que a maioria das vendas é inserida com dados corretos no banco e devoluções ou alterações são raras. Dado essas duas características, não é benéfica a materialização dessa visão, uma vez o benefício de lê-la mais rápido não compensa o custo de mantê-la sempre atualizada.

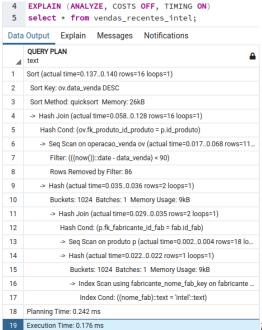


Figura 2: Visão

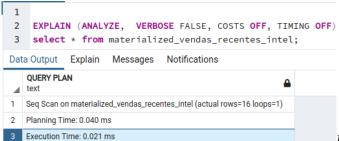


Figura 1: Visão Materializada

Alterações via visão

O **postgressql** não permite atualizações por essa visão devida ao dela ser feita com mais de uma relação, porém há a opção de usar triggers **instead of** para capturar os eventos de atualização e realizá-los de forma adequada, assim evitando a restrição. Embora a visão use mais de uma relação, ela é um mapeamento de 1:1 das colunas da tabela **Operacao_venda**, então basta replicar a atualização da visão sobre aquela tabela.



```
Data Output Explain Messages Notifications

ERROR: cannot delete from view "vendas_recentes_intel"

DETAIL: Views that do not select from a single table or view are not automatically updatable.

HINT: To enable deleting from the view, provide an INSTEAD OF DELETE trigger or an unconditional ON DELETE DO INSTEAD rule.

SQL state: 55000
```

Teste

Criando o trigger de delete somente.

```
1 CREATE OR REPLACE FUNCTION delete_vendas_recentes_intel()
 2 RETURNS TRIGGER language plpgsql
 3 as
 4 $$
 5 ▼ BEGIN
 6
 7
        delete from Operacao_venda
 8
        where
 9
                (OLD.id_venda = id_venda OR (OLD.id_venda is null))
            AND (OLD.valor_venda = valor_venda OR (OLD.valor_venda is null))
10
11
            AND (OLD.quantidade = quantidade OR (OLD.quantidade is null))
            AND (OLD.data_venda = data_venda OR (OLD.data_venda is null))
12
13
            AND (OLD.fk_produto_id_produto = fk_produto_id_produto OR (OLD.fk_produto_id_produto is null))
           AND (OLD.fk_vendedor_id_vendedor = fk_vendedor_id_vendedor OR (OLD.fk_vendedor_id_vendedor is null))
14
15
            AND (OLD.fk_cliente_id_cliente = fk_cliente_id_cliente OR (OLD.fk_cliente_id_cliente is null))
16
            AND (OLD.dias_distancia = dias_distancia OR (OLD.dias_distancia is null));
17
18
        return NULL:
19 END;
20 $$;
21 create trigger trigger_delete_vendas_recentes_intel
22 instead of delete on vendas_recentes_intel
23 for each row execute procedure delete_vendas_recentes_intel();
```

Tentando novamente.

