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;
```

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

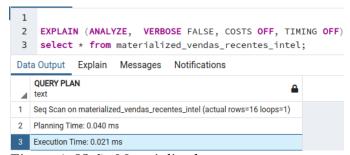
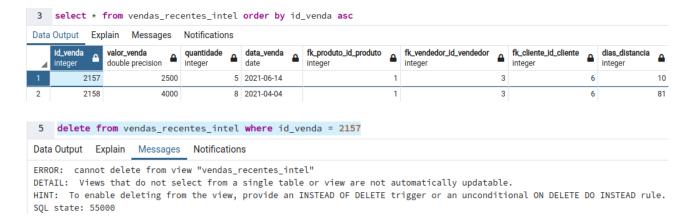


Figura 1: Visão Materializada

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.



Teste

Criando o trigger de delete somente.

```
CREATE OR REPLACE FUNCTION delete_vendas_recentes_intel()
    RETURNS TRIGGER language plpgsql
 2
 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))
10
             AND (OLD.valor_venda = valor_venda OR (OLD.valor_venda is null))
11
             AND (OLD.quantidade = quantidade OR (OLD.quantidade is null))
12
             AND (OLD.data_venda = data_venda OR (OLD.data_venda is null))
             AND (OLD.fk_produto_id_produto = fk_produto_id_produto OR (OLD.fk_produto_id_produto is null))
13
14
              \textbf{AND (OLD.} fk\_vendedor\_id\_vendedor = fk\_vendedor\_id\_vendedor \ \textbf{OR (OLD.} fk\_vendedor\_id\_vendedor \ \textbf{is} \ null)) 
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.

2	<pre>delete from vendas_recentes_intel where id_venda = 2157; select * from vendas_recentes_intel order by id_venda asc;</pre>								
Data Output Explain Messages Notifications									
4	Id_venda integer	valor_venda double precision	quantidade integer	data_venda date	fk_produto_id_produto integer	fk_vendedor_id_vendedor integer	fk_cliente_id_cliente integer	dias_distancia integer	
1	2158	4000	8	2021-04-04	1	3	6	81	
2	2159	4500	9	2021-05-21	1	4	6	34	