

View, CTE ou SubQuery?

Acelere as suas consultas, e as dos seus usuários e stakeholders, em até 90%

**9 perguntas-chave
que vão te direcionar para a
melhor escolha de estrutura!**



A 8 e a 9 vão te fazer
ganhar um aumento!
(eu to falando sério)

Decisões eficientes

Você tem dúvida sobre qual a melhor estrutura utilizar? Se deve criar uma **View, CTE, SubQuery** ou até mesmo algo mais avançado?

Vou te ajudar nesse guia **passo a passo** para que você tenha confiança de tomar **a melhor decisão!**



Contexto

Imagine que você é o responsável pela **operação de um ecommerce** com **milhões de acessos**. Escolher a estrutura correta é **fundamental para aumentar as vendas**.

Se você não gosta de esperar suas query, seus clientes vão gostar de **esperar para carregar os produtos** da página principal?

Vamos lá! **9 perguntas** para você escolher a melhor estrutura.



1 Pergunta:

Sua análise é simples e direta?

Se sim, execute uma query simples

```
SELECT nome_produto, preco
FROM produtos
WHERE preco > 100
```

Dica de especialista: Para necessidades básicas, uma query simples é geralmente a mais rápida e eficiente.

Se não, avance para a pergunta 2.



2 Pergunta:

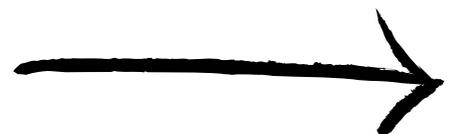
A análise necessita de interações com múltiplas tabelas ou dados relacionados?

Se sim, considere usar Subquery

```
SELECT nome, total_compra
FROM (
    SELECT cliente_id, SUM(valor) AS total_compra
    FROM vendas
    GROUP BY cliente_id
) AS subconsulta;
```

Dica de especialista: Subqueries são ideais para operações com várias tabelas, permitindo consultas aninhadas e extração de dados relacionados.

Se não, avance para a pergunta 3.



3 Pergunta:

A consulta é para uma análise única e complexa, mas você quer manter a legibilidade?

Se sim, utilize uma CTE

```
WITH VendasMensais AS (  
    SELECT MONTH(data_venda) as mes,  
           SUM(preco) as total_vendas  
    FROM vendas  
    GROUP BY MONTH(data_venda)  
)  
SELECT mes, total_vendas FROM VendasMensais;
```

Dica de especialista: CTEs fornecem uma forma estruturada de criar consultas complexas, mantendo-as claras e legíveis. Se não, avance para a pergunta 4.



4 Pergunta:

Você está lidando com uma análise ao longo do dia e quer manter a estrutura organizada sem afetar o esquema principal?

Se sim, considere usar TempView.

```
CREATE TEMPORARY VIEW temp_vendas AS
SELECT nome_produto, SUM(quantidade_vendida) as total_vendido
FROM dados_vendas
GROUP BY nome_produto;

SELECT * FROM temp_vendas
```

Dica de especialista: TempViews são perfeitas para análises temporárias dos dados, sem persistência além da sessão atual, garantindo flexibilidade.

Se não, avance para a pergunta 5.



5 Pergunta:

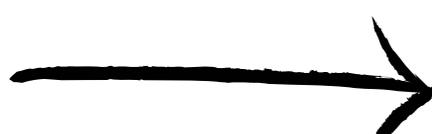
Você está criando relatórios ou dashboards que serão utilizados com frequência e precisa de acessos e permissões diferenciadas?

Se sim, utilize uma View.

```
CREATE VIEW VendasFilialSaoPaulo AS
SELECT nome_produto,
       quantidade_vendida
FROM vendas
WHERE regioao = 'São Paulo';

-- Dar acesso de leitura apenas para o usuário "TeamSP"
GRANT SELECT ON VendasFilialSaoPaulo TO TeamSP;

SELECT nome_produto,
       quantidade_vendida
FROM VendasFilialSaoPaulo;
```

Dica de especialista: Com Views, é possível estabelecer regras de acesso diferenciadas, garantindo que usuários não vejam dados sensíveis em diferentes relatórios.  Se não, avance para a pergunta 6.

6 Pergunta:

Você está testando novas consultas, como estratégia de Index, ou precisa armazenar dados por um curto período?

Se sim, utilize uma TempTable.

```
CREATE TEMPORARY TABLE TempComprasPorPais AS
SELECT pais,
       id_cliente,
       nome_produto,
       SUM(preco) as total_gasto
FROM vendas
GROUP BY pais, id_cliente, nome_produto;

CREATE INDEX idx_temp_pais ON TempComprasPorPais(pais);

# Usando o índice para tornar a consulta mais performática
SELECT * FROM TempComprasPorPais WHERE pais = 'Brasil';
```

Dica de especialista: TempTables são ideais para operações temporárias, existindo apenas durante a sessão atual. Você pode usar o desempenho de índices específicos nelas. Se não, avance para a pergunta 7. 

7 Pergunta:

Você precisa armazenar informações a longo prazo? Elas serão usadas pela aplicação principal para realizar o famoso CRUD?

Se sim, utilize uma Tabela.

```
CREATE TABLE DadosClientes (  
    id_cliente INT PRIMARY KEY,  
    nome_cliente VARCHAR(100),  
    email VARCHAR(100) UNIQUE,  
    data_registro DATE  
);  
  
SELECT id_cliente,  
    nome_cliente,  
    email,  
    data_registro  
FROM DadosClientes
```

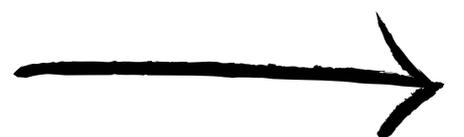
Dica de especialista: Tabelas são fundamentais para armazenar e relacionar informações de forma durável.

Se não, avance para a pergunta 8.



**Até aqui meu amigo,
você já vai tirar onda nas
suas análises SQL.**

**Mas a 8 e a 9 é para ser
promovido!**



8 Pergunta:

Os dados serão acessados por milhões de usuários? Exemplo, Black Friday!

Se sim, considere utilizar uma estrutura de cache no Redis. (Vamos usar um pouco de Python, ok?)

```
import redis, schedule, psycpg2

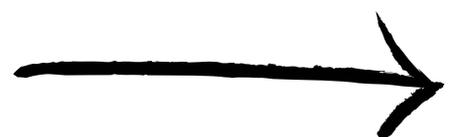
# Função que busca os produtos em promoção na Black Friday da view
# 'VendasBlackFriday', sim você pode misturar os conceitos!
def fetch_black_friday_promotions():
    cursor.execute("SELECT nome_produto FROM VendasBlackFriday;")
    promotions = cursor.fetchall()
    return [item[0] for item in promotions]

# Atualizando o cache a cada 5 minutos para pegar sempre as novidades
# da View e não sobrecarregar o banco
def update_cache():
    promotions = fetch_black_friday_promotions()
    client.set('black_friday_promotions', str(promotions))

# Agendamento da atualização do cache
schedule.every(5).minutes.do(update_cache)

# Mantenha o script rodando para executar a tarefa agendada
while True:
    schedule.run_pending()
```

Dica de especialista: Cache é vital quando se espera um alto tráfego, já que acelera o acesso aos dados, garantindo uma experiência de usuário fluida. Na casa de 90% mais rápido!
Não, avance para a pergunta 9



9 Pergunta:

Essa análise será feita frequentemente e envolve uma quantidade massiva de dados?

Se sim, considere migrar para um banco de dados OLAP.

```
import psycopg2, snowflake
# Conexão ao banco de dados PostgreSQL e Snowflake
conn_pg = psycopg2.connect("...seu_password...")
conn_sf = connect("...seu_password...")

# Suponha que você queira copiar uma tabela chamada "produtos"
cursor_pg.execute("SELECT * FROM produtos;")
data = cursor_pg.fetchall()
column_names = [desc[0] for desc in cursor_pg.description]

# Criar tabela no Snowflake, se ela ainda não existir (opcional)
create_table_query = "CREATE TABLE IF NOT EXISTS minha_tabela (...);"
cursor_sf.execute(create_table_query)

# Inserir dados no Snowflake
for row in data:
    insert_query = "INSERT INTO minha_tabela ({columns}) VALUES
({values});".format(
        columns=", ".join(column_names),
        values=", ".join(["%s" * len(row)])
    )
    cursor_sf.execute(insert_query, row)
```

Dica de especialista: Bancos OLAP são incríveis. Snowflake, Databricks, BigQuery e Fabric são alguns exemplos. Eles vão proporcionar velocidade e eficiência em análises intensivas para toda a organização!



Concluindo a jornada

Neste guia, exploramos diferentes **estruturas de dados**, todas valiosas dependendo do cenário. No mundo do varejo, **otimizar suas escolhas de banco de dados é crucial.**

A melhor estrutura não é uma solução única. Mas sim a **combinação delas.**

A melhor estrutura é determinada pelas suas necessidades, dos seus usuários, e principalmente, **dos seus clientes!**



Obrigado!

Se essa postagem te ajudou, curte e comente! **Sua interação pode ajudar outras pessoas** a se beneficiarem desse conhecimento.

Siga-me para receber dicas, insights e tutoriais de problemas que já passei muita noite virado para resolver!

(e espero que você não passe!)

