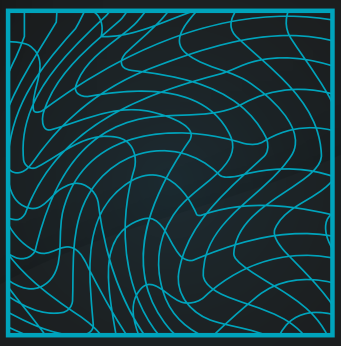
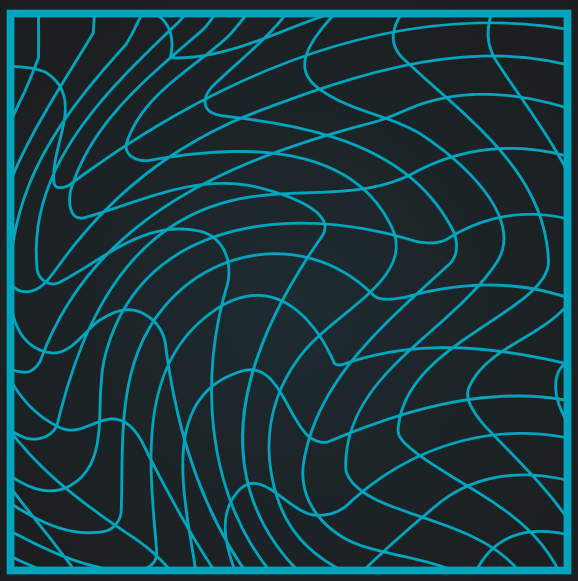


The Data Pattern

DATA ANALYSIS

Playbook para Análise de Dados: SQL

Playbook para Análise de Dados: SQL



The Data Pattern

DATA ANALYSIS

Playbook para Análise de Dados: SQL

Introdução

Se você pesquisar vagas para Analista de Dados Júnior, Analista de BI ou Analytics, encontrará um requisito repetido inúmeras vezes: SQL.

Isso acontece porque praticamente todas as empresas armazenam seus dados em bancos de dados relacionais. Antes de criar dashboards, desenvolver relatórios ou construir modelos analíticos, é necessário acessar e consultar os dados. É exatamente para isso que o SQL existe.

SQL (Structured Query Language) é a linguagem utilizada para consultar, filtrar, organizar e transformar dados armazenados em bancos de dados. Para um profissional de dados, SQL é tão importante quanto o Excel para um analista financeiro.

Neste playbook você aprenderá os principais conceitos que todo Analista de Dados precisa dominar para ingressar no mercado.

Como os Dados São Armazenados?

Imagine um e-commerce que vende produtos pela internet.

Os dados desse negócio podem estar organizados em diferentes tabelas.

Tabela Clientes

id_clientenomecidade		
1	João	São Paulo
2	Maria	Rio de Janeiro
3	Carlos	Belo Horizonte
4	Ana	São Paulo

Tabela Produtos

id_produtoprodutocategoria		
101	Notebook	Informática
102	Mouse	Informática
103	Cadeira Gamer	Móveis
104	Monitor	Informática

Tabela Pedidos

id_pedidoid_clienteid_produtoquantidadevalor				
1	1	101	1	5000
2	2	102	2	200
3	3	103	1	800
4	1	104	1	1200
5	4	102	3	300

Ao longo deste playbook utilizaremos essas tabelas para todos os exemplos.

SELECT

O comando SELECT é utilizado para consultar dados.

Sempre que você quiser visualizar informações armazenadas em uma tabela, utilizará SELECT.

Consultando todas as colunas

```
SELECT *
FROM clientes;
```

Resultado:

id_clientenomecidade		
1	João	São Paulo
2	Maria	Rio de Janeiro
3	Carlos	Belo Horizonte
4	Ana	São Paulo

O caractere * significa "todas as colunas".

Embora seja muito utilizado para exploração inicial dos dados, não é recomendado em ambientes produtivos porque pode retornar informações desnecessárias.

Consultando colunas específicas

```
SELECT
    nome,
    cidade
FROM clientes;
```

Resultado:

nomecidade	
João	São Paulo
Maria	Rio de Janeiro
Carlos	Belo Horizonte
Ana	São Paulo

Essa abordagem é mais eficiente e facilita a leitura da consulta.

WHERE

O comando WHERE é utilizado para filtrar registros.

Imagine que seu gestor faça a seguinte pergunta:

"Quais clientes são da cidade de São Paulo?"

Você pode responder utilizando SQL.

```
SELECT *
FROM clientes
WHERE cidade = 'São Paulo';
```

Resultado:

id_clientenomecidade		
1	João	São Paulo
4	Ana	São Paulo

Utilizando AND

Imagine que você queira encontrar pedidos acima de R$ 500.

```
SELECT *
FROM pedidos
WHERE valor > 500
AND quantidade >= 1;
```

O operador AND exige que todas as condições sejam verdadeiras.

Utilizando OR

```
SELECT *
FROM clientes
WHERE cidade = 'São Paulo'
OR cidade = 'Rio de Janeiro';
```

O operador OR exige que apenas uma das condições seja verdadeira.

Utilizando IN

```
SELECT *
FROM clientes
WHERE cidade IN ('São Paulo', 'Rio de Janeiro');
```

O resultado será igual ao exemplo anterior, porém com uma sintaxe mais organizada.

Utilizando BETWEEN

```
SELECT *
FROM pedidos
WHERE valor BETWEEN 500 AND 2000;
```

Resultado:

id_pedidovalor	
3	800
4	1200

BETWEEN é muito utilizado para filtros de datas e intervalos numéricos.

Utilizando LIKE

Suponha que você queira encontrar clientes cujo nome começa com a letra A.

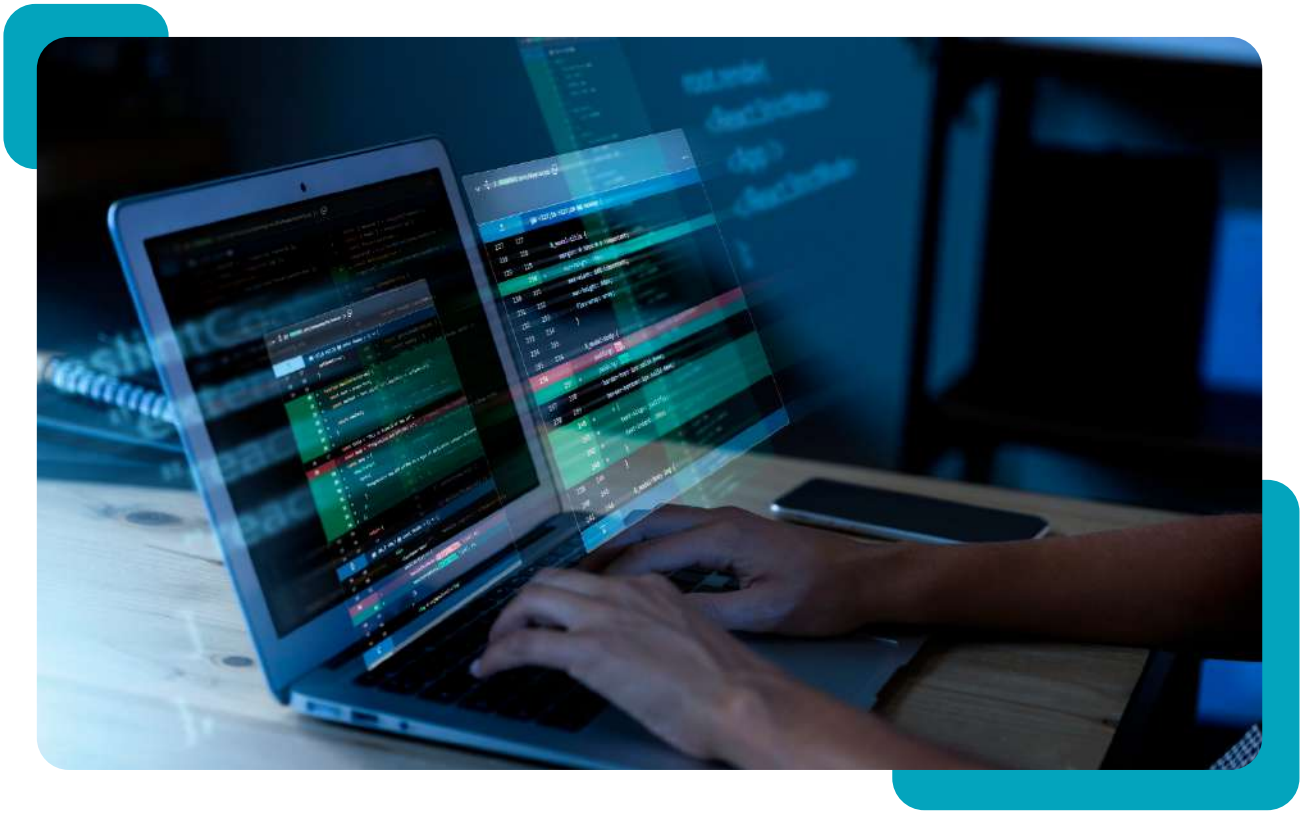
```
SELECT *
FROM clientes
WHERE nome LIKE 'A%';
```

Resultado:

nome
Ana

Símbolos importantes:

SímboloSignificado	
%	Qualquer quantidade de caracteres
_	Apenas um caractere



ORDER BY

O ORDER BY é utilizado para ordenar resultados.

Imagine que você deseja visualizar os pedidos do maior para o menor valor.

```
SELECT *
FROM pedidos
ORDER BY valor DESC;
```

Resultado:

pedidovalor	
1	5000
4	1200
3	800
5	300
2	200

DESC significa ordem decrescente.

Ordem Crescente

```
SELECT *
FROM pedidos
ORDER BY valor ASC;
```

ASC significa ordem crescente.

Embora seja opcional, é considerado uma boa prática informá-lo explicitamente.

Funções de Agregação

As funções de agregação resumem grandes volumes de dados em indicadores.

São extremamente utilizadas em relatórios e dashboards.

SUM

Calcula uma soma.

Pergunta de negócio:

"Quanto a empresa faturou?"

```
SELECT
    SUM(valor) AS faturamento_total
FROM pedidos;
```

Resultado:

faturamento_total
7500

AVG

Calcula uma média.

Pergunta:

"Qual o valor médio dos pedidos?"

```
SELECT
    AVG(valor) AS ticket_medio
FROM pedidos;
```

Resultado:

ticket_medio
1500

COUNT

Conta registros.

Pergunta:

"Quantos pedidos foram realizados?"

```
SELECT
    COUNT(*) AS total_pedidos
FROM pedidos;
```

Resultado:

total_pedidos
5

MAX

Retorna o maior valor.

```
SELECT
    MAX(valor) AS maior_pedido
FROM pedidos;
```

Resultado:

maior_pedido
5000

MIN

Retorna o menor valor.

```
SELECT
    MIN(valor) AS menor_pedido
FROM pedidos;
```

Resultado:

menor_pedido
200

GROUP BY

O GROUP BY é um dos comandos mais importantes da análise de dados.

Ele permite transformar milhares de registros em informação gerencial.

Imagine a seguinte tabela simplificada:

categoria
Informática
Informática
Informática
Móveis
Móveis

Visualmente percebemos que existem categorias repetidas.

O GROUP BY agrupa esses registros.

Quantidade de vendas por categoria

```
SELECT
    categoria,
    COUNT(*) AS quantidade_vendas
FROM vendas
GROUP BY categoria;
```

Resultado:

categoriaquantidade_vendas	
Informática	3
Móveis	2

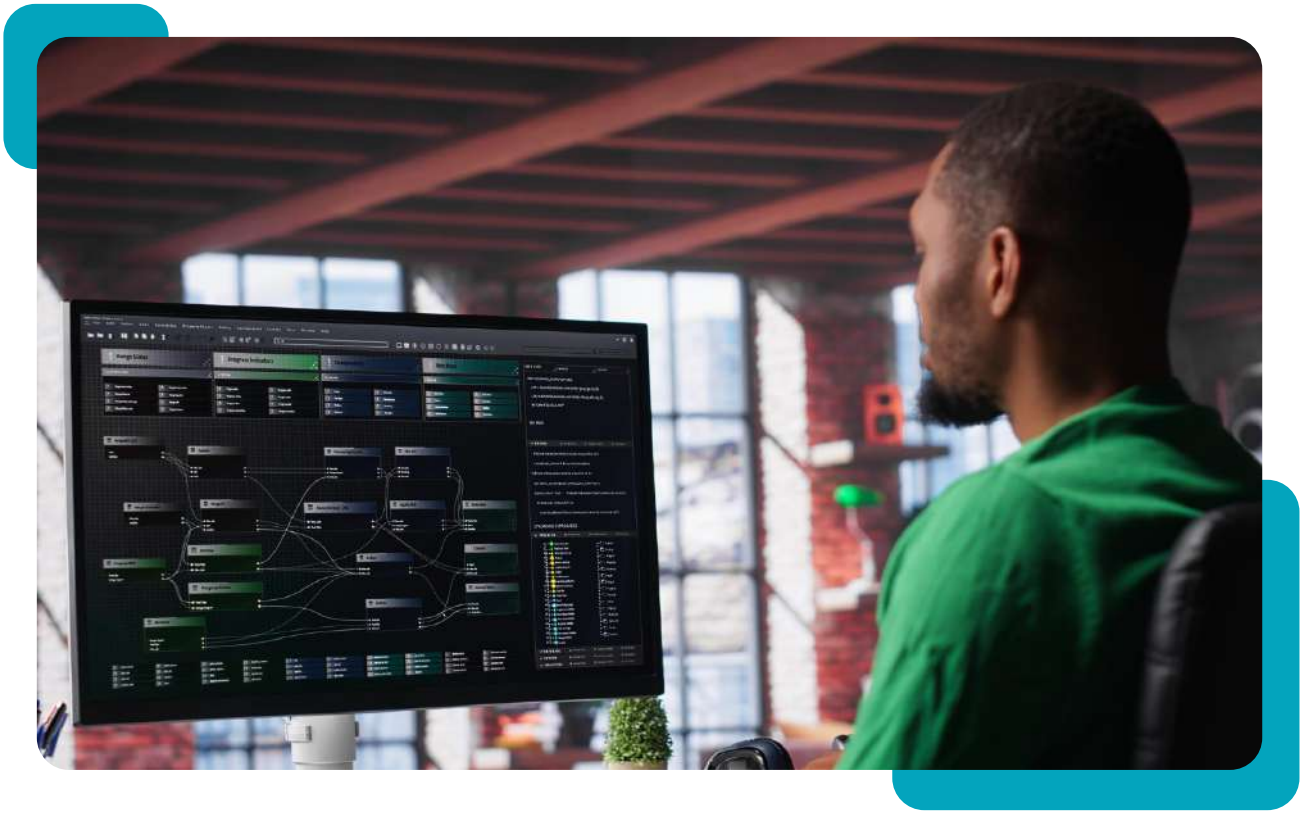
Faturamento por categoria

```
SELECT
    categoria,
    SUM(valor) AS faturamento
FROM vendas
GROUP BY categoria;
```

Resultado:

categoriafaturamento	
Informática	6400
Móveis	800

Este é exatamente o tipo de consulta utilizada em dashboards corporativos.



HAVING

O WHERE filtra linhas.

O HAVING filtra grupos.

Essa diferença é extremamente importante.

Imagine que você queira visualizar apenas categorias com faturamento superior a R$ 1.000.

```
SELECT
    categoria,
    SUM(valor) AS faturamento
FROM vendas
GROUP BY categoria
HAVING SUM(valor) > 1000;
```

Resultado:

categoriafaturamento	
Informática	6400

JOINs

Na prática, os dados raramente ficam em uma única tabela.

Por isso, JOIN é uma das habilidades mais importantes para um Analista de Dados.

Entendendo o Problema

Tabela Clientes:

id_clientenome	
1	João
2	Maria
3	Carlos

Tabela Pedidos:

id_pedidoid_clientevalor		
100	1	500
101	2	800
102	1	1200

Observe que a tabela de pedidos não possui o nome do cliente.

Ela possui apenas o id_cliente.

Precisamos relacionar as duas tabelas.

INNER JOIN

Retorna apenas registros que existem em ambas as tabelas.

```
SELECT
    c.nome,
    p.valor
FROM clientes c
INNER JOIN pedidos p
    ON c.id_cliente = p.id_cliente;
```

Resultado:

nomevalor	
João	500
Maria	800
João	1200

LEFT JOIN

Retorna todos os registros da tabela da esquerda.

```
SELECT
    c.nome,
    p.valor
FROM clientes c
LEFT JOIN pedidos p
    ON c.id_cliente = p.id_cliente;
```

Mesmo que um cliente não tenha realizado pedidos, ele aparecerá no resultado.

RIGHT JOIN

Retorna todos os registros da tabela da direita.

```
SELECT
    c.nome,
    p.valor
FROM clientes c
RIGHT JOIN pedidos p
    ON c.id_cliente = p.id_cliente;
```

Menos utilizado na prática, mas importante para entrevistas e processos seletivos.

Subqueries

Uma Subquery é uma consulta dentro de outra consulta.

Imagine a seguinte pergunta:

"Quais pedidos possuem valor acima da média?"

Primeiro precisamos descobrir qual é a média.

Depois comparar cada pedido com essa média.

```
SELECT
    id_pedido,
    valor
FROM pedidos
WHERE valor >
(
    SELECT AVG(valor)
    FROM pedidos
);
```

O SQL executa primeiro a consulta interna.

Depois utiliza o resultado para filtrar a consulta principal.

CTEs

CTE significa Common Table Expression.

Ela permite criar tabelas temporárias durante a execução de uma consulta.

É muito utilizada para organizar consultas complexas.

Exemplo Básico

```
WITH faturamento_clientes AS
(
    SELECT
        id_cliente,
        SUM(valor) AS faturamento
    FROM pedidos
    GROUP BY id_cliente
)

SELECT *
FROM faturamento_clientes;
```

Resultado:

id_clientefaturamento	
1	6200
2	200
3	800
4	300

Exemplo Mais Próximo da Realidade

Quais clientes faturaram acima da média?

```
WITH faturamento_clientes AS
(
    SELECT
        id_cliente,
        SUM(valor) AS faturamento
    FROM pedidos
    GROUP BY id_cliente
)

SELECT *
FROM faturamento_clientes
WHERE faturamento >
(
    SELECT AVG(faturamento)
    FROM faturamento_clientes
);
```

Observe como a consulta fica muito mais organizada do que utilizando várias subqueries aninhadas.

Caso Prático de Negócio

Imagine que você acabou de entrar em uma empresa de e-commerce.

Seu gestor solicita algumas análises.

Qual o faturamento total da empresa?

```
SELECT
    SUM(valor) AS faturamento_total
FROM pedidos;
```

Quem são os clientes que mais compram?

```
SELECT
    id_cliente,
    SUM(valor) AS faturamento
FROM pedidos
GROUP BY id_cliente
ORDER BY faturamento DESC;
```

Qual categoria vende mais?

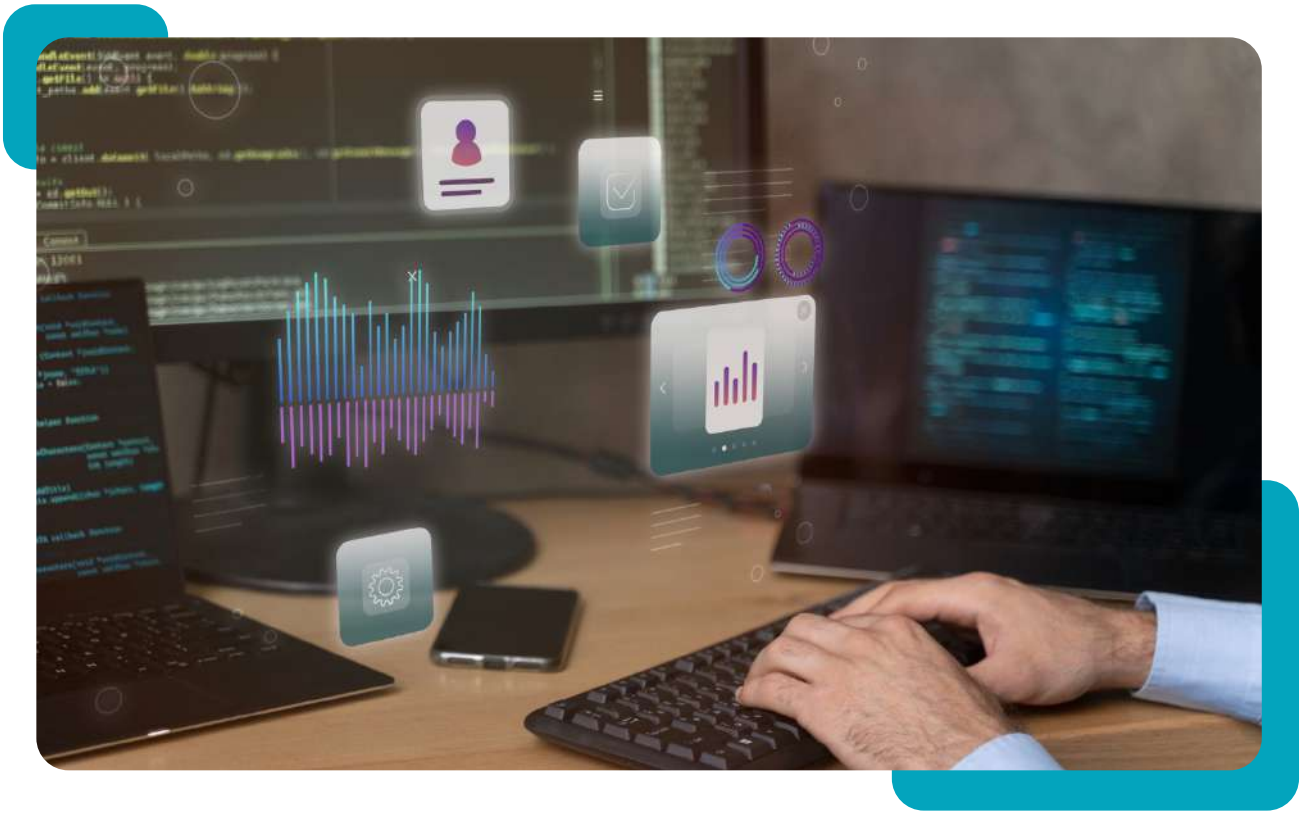
```
SELECT
    categoria,
    SUM(valor) AS faturamento
FROM vendas
GROUP BY categoria
ORDER BY faturamento DESC;
```

Qual cidade gera mais receita?

```
SELECT
    c.cidade,
    SUM(p.valor) AS faturamento
FROM pedidos p
INNER JOIN clientes c
    ON p.id_cliente = c.id_cliente
GROUP BY c.cidade
ORDER BY faturamento DESC;
```

Quais são os produtos mais vendidos?

```
SELECT
    produto,
    SUM(quantidade) AS total_vendido
FROM vendas
GROUP BY produto
ORDER BY total_vendido DESC;
```



Erros Mais Comuns de Iniciantes

Esquecer o GROUP BY

Erro:

```
SELECT
    categoria,
    SUM(valor)
FROM vendas;
```

Correto:

```
SELECT
    categoria,
    SUM(valor)
FROM vendas
GROUP BY categoria;
```

Utilizar HAVING quando deveria utilizar WHERE

Errado:

```
SELECT *
FROM vendas
HAVING valor > 100;
```

Correto:

```
SELECT *
FROM vendas
WHERE valor > 100;
```

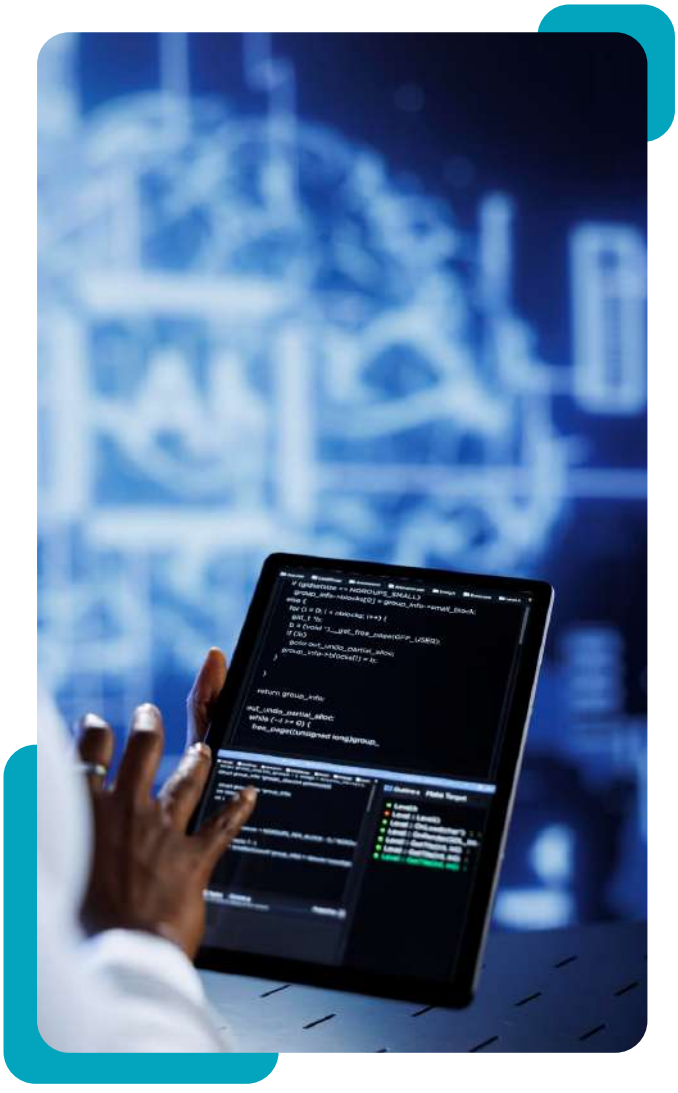
Utilizar JOIN sem condição

Errado:

```
SELECT *
FROM clientes
JOIN pedidos;
```

Correto:

```
SELECT *
FROM clientes c
JOIN pedidos p
    ON c.id_cliente = p.id_cliente;
```



O Que Você Precisa Saber Para Conseguir uma Vaga Júnior?

Se você domina os tópicos abaixo, já possui uma base sólida para concorrer a vagas de Analista de Dados Júnior:

- SELECT
- WHERE
- ORDER BY
- GROUP BY
- HAVING
- SUM
- AVG
- COUNT
- MAX
- MIN
- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- Subqueries básicas
- CTEs básicas
- Interpretação de regras de negócio
- Transformação de perguntas de negócio em consultas SQL

O mercado não espera que um profissional júnior domine otimização avançada, modelagem complexa ou arquitetura de bancos de dados. O que realmente importa é a capacidade de utilizar SQL para responder perguntas de negócio de forma clara, confiável e eficiente.

Conclusão

SQL é a principal linguagem da área de dados e continua sendo a habilidade técnica mais importante para quem deseja ingressar no mercado.

Praticamente todo dashboard, relatório ou análise começa com uma consulta SQL. Quanto mais confortável você estiver lendo, escrevendo e interpretando consultas, maior será sua capacidade de gerar valor para o negócio.

Dominar SELECT, WHERE, GROUP BY, HAVING, JOINs, Subqueries e CTEs é o primeiro passo para construir uma carreira sólida em Dados, Business Intelligence, Analytics e Ciência de Dados.

Rodrigo C. Furlan

thedatapattern.com

linkedin.com/in/rodrigocfurlan

+55 (16) 997838533