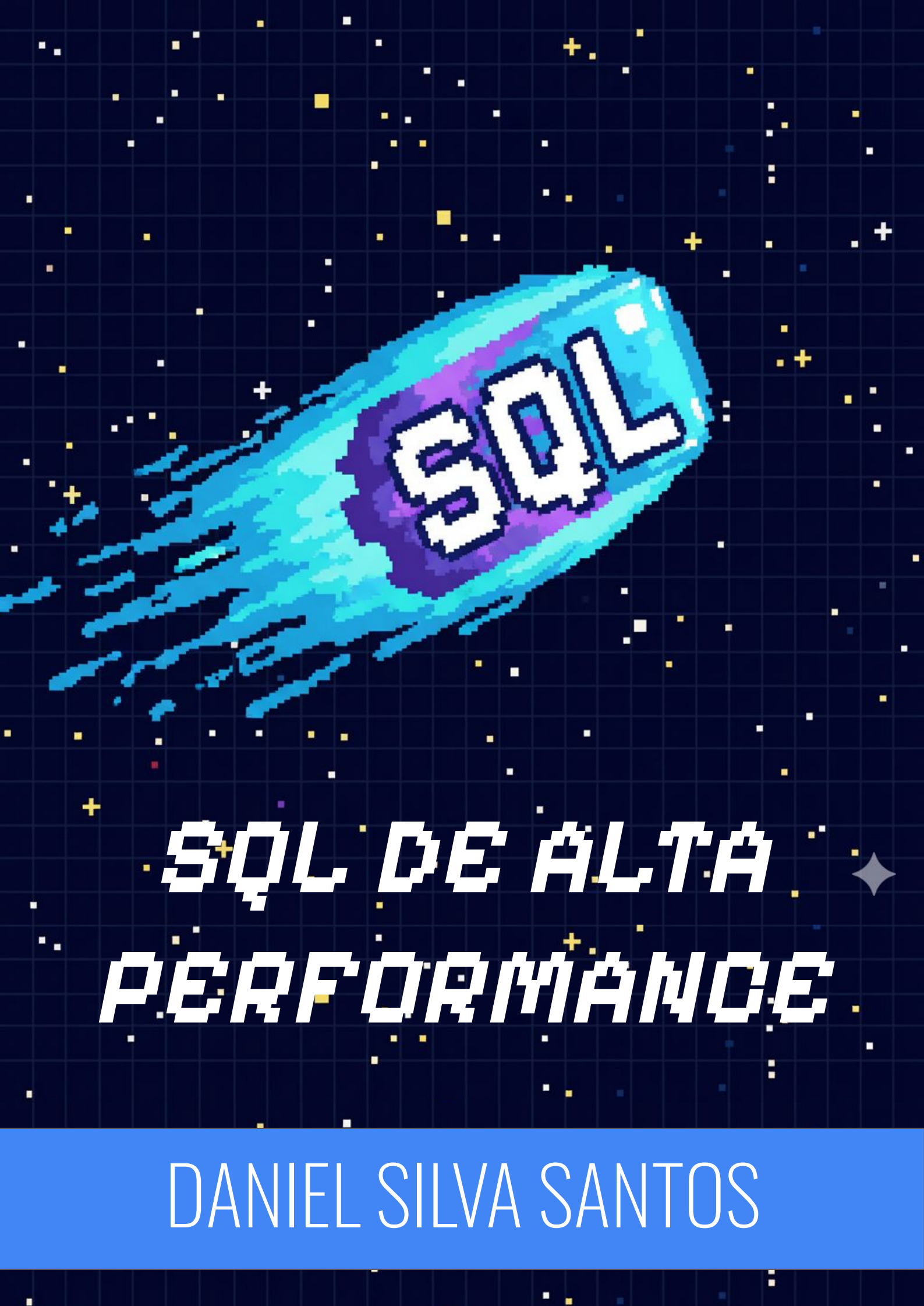


SQL DE ALTA PERFORMANCE

DANIEL SILVA SANTOS

PERFORMANCE EM SQL

A performance em SQL depende menos do hardware e muito mais de como você escreve suas consultas. Em qualquer banco de dados, existem três fatores críticos:

1. Quantidade de dados que o banco precisa ler.

2. Como o plano de execução é montado.

3. Uso correto de índices, filtros e junções.

Quando uma consulta está lenta, geralmente é por um destes motivos:

- Falta de índice adequado

- Filtro mal escrito

- JOIN desnecessário

PERFORMANCE EM SQL

Exemplo de um caso clássico de lentidão:

```sql
SELECT * FROM vendas WHERE data_criacao::DATE = '2024-01-10';
```

O uso de ::DATE ou DATE() na coluna faz o banco ignorar índices.

A otimização seria:

```sql
SELECT * FROM vendas
WHERE data_criacao >= '2024-01-10'
  AND data_criacao < '2024-01-11';
```

Pronto para descobrir como otimizar suas queries?

01

ÍNDICES: O CORAÇÃO DA PERFORMANCE

ÍNDICES

O que são índices?

Índices são estruturas internas que permitem ao banco encontrar dados sem ler a tabela inteira.
Pense como um índice de livro: você não lê o livro todo para achar um capítulo.

Quando criar índices

- Colunas usadas em **WHERE**
- Colunas usadas em **JOIN**
- Colunas usadas em **ORDER BY**
- Colunas usadas em **GROUP BY**

ÍNDICES

Exemplo prático

```sql
CREATE INDEX idx_email_users ON users(email);

SELECT id, nome FROM users WHERE email = 'exemplo@site.com';
```

Quando não usar

- Colunas booleanas
- Colunas com poucos valores distintos
- Tabelas muito pequenas

02

SELEÇÃO EFICIENTE: USE APENAS O NECESSÁRIO

SELEÇÃO EFICIENTE

Muitos problemas de performance vêm do uso de SELECT *.

Por que evitar SELECT *?

- Carrega mais dados do que o necessário.
- Prejudica cache interno do banco.
- Aumenta o tráfego entre servidor e aplicação.

Exemplo ruim

```sql
SELECT * FROM pedidos WHERE status = 'PAGO';
```

SELEÇÃO EFICIENTE

Exemplo otimizado

```sql
SELECT id, valor_total, data
FROM pedidos
WHERE status = 'PAGO';
```

03

FILTROS OTIMIZADOS: WHERE SEM TRAVAR

FILTROS OTIMIZADOS

O erro mais comum:

Usar funções na coluna filtrada.

```sql
-- Lento
SELECT * FROM clientes WHERE LOWER(email) = 'teste@a.com';
```

Versão rápida

```sql
SELECT * FROM clientes WHERE email = 'teste@a.com';
```

04

JOINS E SUBCONSULTAS: UNIR COM VELOCIDADE

SUBCONSULTAS COM JOIN

Subconsultas podem ser reexecutadas centenas de vezes

```sql
SELECT nome,
       (SELECT COUNT(*) FROM pedidos p WHERE p.cliente_id = c.id)
FROM clientes c;
```

Aqui, a subconsulta roda 1 vez por cliente.

Otimização com JOIN

```sql
SELECT c.nome, COUNT(p.id) AS total_pedidos
FROM clientes c
LEFT JOIN pedidos p ON p.cliente_id = c.id
GROUP BY c.nome;
```

05

CONTROLE DE VOLUME

CONTROLE DE VOLUME

LIMIT evita desperdício.

Sem LIMIT, mesmo queries simples podem ser pesadas.

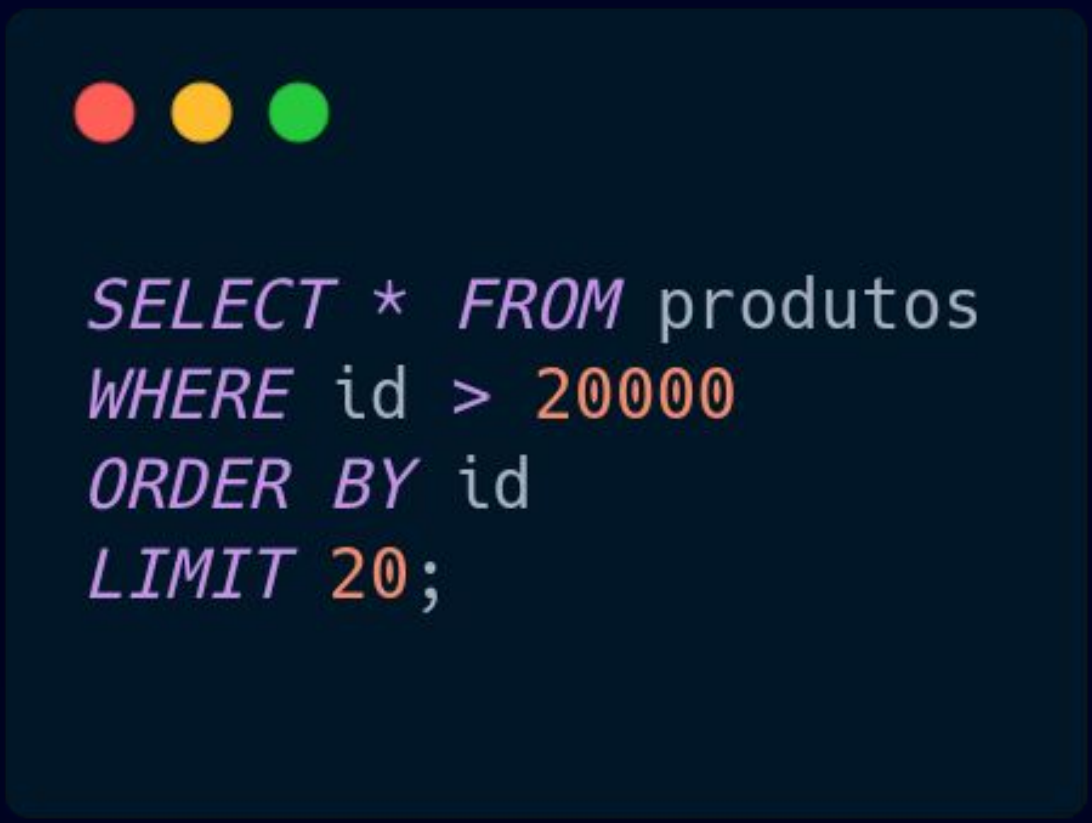
```sql
SELECT * FROM logs ORDER BY criado_em DESC LIMIT 100;
```

OFFSET fica lento em tabelas grandes

```sql
-- Lento
SELECT * FROM produtos ORDER BY id LIMIT 20 OFFSET 20000;
```

CONTROLE DE VOLUME

Keyset Pagination



06

CONDIÇÕES COMPLEXAS

CONDIÇÕES COMPLEXAS

IN gigante = lentidão

```sql
WHERE cliente_id IN (1,2,3,4,5,6,7,8,9,...)
```

Use JOIN

```sql
SELECT v.*
FROM vendas v
JOIN clientes_ativos ca ON ca.cliente_id = v.cliente_id;
```

CONDIÇÕES COMPLEXAS

Quando usar EXISTS:

Boa prática quando o objetivo é verificar existência

```sql
SELECT * FROM clientes c
WHERE EXISTS (
      SELECT 1 FROM pedidos p
      WHERE p.cliente_id = c.id
);
```

07

TRANSAÇÕES OTIMIZADAS

TRANSAÇÕES OTIMIZADAS

Transações longas criam bloqueios e geram filas de espera.

Bom exemplo

```sql
BEGIN;
UPDATE contas SET saldo = saldo - 100 WHERE id = 10;
UPDATE contas SET saldo = saldo + 100 WHERE id = 20;
COMMIT;
```

Regras

- Nunca faça lógica da aplicação dentro da transação.
- Leia somente o necessário.
- Feche rápido.

AGRADECIMENTOS

OBRIGADO POR LER ATÉ AQUI!

Meu GitHub:

https://github.com/dsilvasantos9125

Conteúdo gerado por IA e diagramado e revisado por um humano.

DANIEL SILVA SANTOS