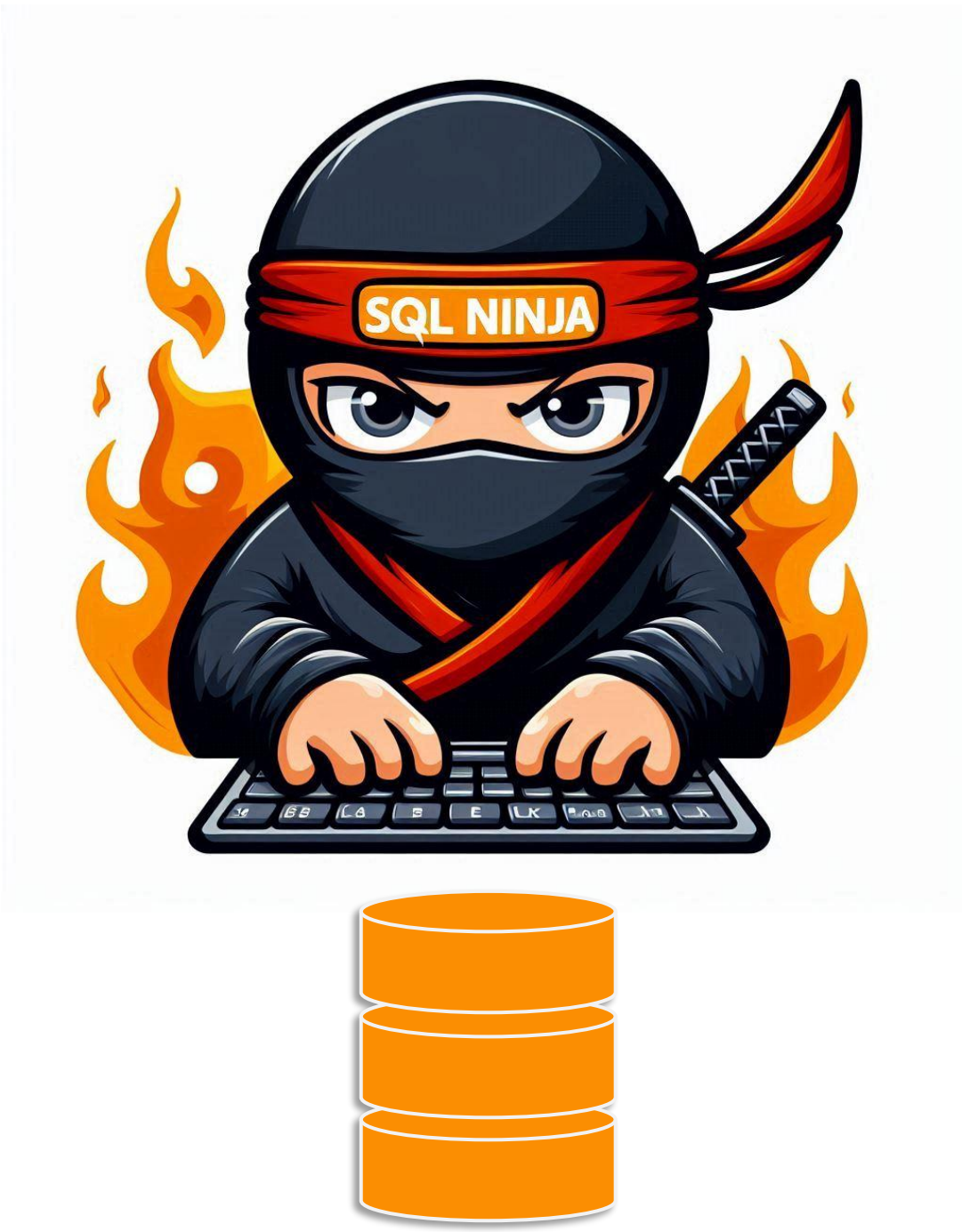

SQL do Ninja

Domine o poder das querys

Diego Castro

Introdução

Melhorando suas consultas

Otimizar consultas SQL é uma habilidade essencial para garantir que seu banco de dados funcione de maneira eficiente e rápida. Neste ebook, vamos nos aprofundar em técnicas avançadas para escrever queries performáticas, com exemplos detalhados para PostgreSQL e SQL Server.

Capítulo 01

Escolha de Índices Adequados

Índices Compostos e Funções de Indexação

Índices compostos são úteis quando você precisa filtrar resultados com base em múltiplas colunas. Funções de indexação podem ser usadas para criar índices em colunas derivadas de expressões.

Exemplos em PostgreSQL / SQL Server: :

```
CREATE INDEX idx_cliente_pedido ON pedidos(cliente_id,
data_pedido);

-- Uso do índice na query
SELECT cliente_id, data_pedido FROM pedidos WHERE cliente_id = 1
AND data_pedido > '2025-01-01';
```

Capítulo 02

Utilize SELECT Apenas nas Colunas Necessárias

Utilize SELECT Apenas nas Colunas Necessárias

Selecionar apenas as colunas necessárias reduz o volume de dados transferidos e processados pelo banco de dados, melhorando a performance da query.

Exemplos em PostgreSQL / SQL Server:

```
SELECT nome, email FROM clientes WHERE cidade = 'Brasília';
```

Capítulo 03

Subqueries Aninhadas e Agregações

Subqueries Aninhadas e Agregações

Subqueries aninhadas podem ser usadas para criar consultas complexas. As funções agregadas permitem calcular valores a partir de várias linhas.

Exemplos em PostgreSQL / SQL Server:

```sql
-- Subquery aninhada para calcular a soma total de pedidos por
cliente
SELECT cliente_id, (SELECT SUM(valor) FROM pedidos WHERE cliente_id
= clientes.id) AS total_pedidos
FROM clientes;

-- Agregação com função de janela
SELECT cliente_id, valor, SUM(valor) OVER (PARTITION BY cliente_id)
AS total_cliente
FROM pedidos;
```

Capítulo 04

Filtros Eficientes com WHERE

Filtros Eficientes com WHERE

Filtrar dados utilizando a cláusula WHERE reduz o número de linhas processadas, melhorando a eficiência da consulta.

Exemplos em PostgreSQL / SQL Server:

```
SELECT nome, email FROM clientes WHERE cidade = 'Brasília' AND
idade > 30;
```

Capítulo 05

Utilização de Common Table Expressions (CTEs)

Utilização de Common Table Expressions (CTEs)

CTEs são úteis para dividir uma query complexa em partes menores e mais legíveis. Eles podem ser usados para realizar cálculos intermediários.

Exemplos em PostgreSQL / SQL Server:

```sql
WITH total_pedidos AS (
    SELECT cliente_id, SUM(valor) AS total
    FROM pedidos
    GROUP BY cliente_id
)
SELECT clientes.nome, total_pedidos.total
FROM clientes
JOIN total_pedidos ON clientes.id = total_pedidos.cliente_id;
```

Capítulo 06

Consultas com Janelas e Funções Analíticas

Consultas com Janelas e Funções Analíticas

Funções de janela permitem calcular valores com base em um conjunto de linhas relacionadas a linha atual sem agregar os resultados.

Exemplos em PostgreSQL / SQL Server:

```
SELECT cliente_id, valor,
       RANK() OVER (PARTITION BY cliente_id ORDER BY valor DESC) AS
rank_valor
FROM pedidos;
```

Capítulo 07

Indexação de Colunas Derivadas

Indexação de Colunas Derivadas

Criação de índices em colunas derivadas de expressões pode melhorar a performance de queries que utilizam essas colunas.

Exemplos em PostgreSQL:

```
CREATE INDEX idx_data_ano ON pedidos ((date_part('year',
data_pedido)));

-- Uso do índice na query
SELECT cliente_id, data_pedido FROM pedidos WHERE date_part('year',
data_pedido) = 2025;
```

Exemplos em SQL Server:

```
-- No SQL Server, funções como `YEAR` são usadas diretamente
SELECT cliente_id, data_pedido FROM pedidos WHERE YEAR(data_pedido)
= 2025;
```

Capítulo 08

Limite o Número de Linhas Retornadas

Limite o Número de Linhas Retornadas

Utilizar a cláusula LIMIT em PostgreSQL ou a cláusula TOP em SQL Server ajuda a limitar o número de linhas retornadas, o que é útil em paginação.

Exemplos em PostgreSQL:

```
SELECT nome FROM clientes ORDER BY nome LIMIT 10 OFFSET 20;
```

Exemplos em SQL Server:

```
SELECT nome FROM clientes ORDER BY nome OFFSET 20 ROWS FETCH NEXT
10 ROWS ONLY;
```

Capítulo 09

Subqueries e Joins

Limite o Número de Linhas Retornadas

Joins tendem a ser mais eficientes e fáceis de entender do que subqueries complexas, pois evitam a execução de várias consultas aninhadas.

Exemplos em PostgreSQL / SQL Server:

```sql
-- Subquery
SELECT nome FROM clientes WHERE id IN (SELECT cliente_id FROM
pedidos WHERE valor > 1000);

-- Join
SELECT clientes.nome FROM clientes INNER JOIN pedidos ON
clientes.id = pedidos.cliente_id WHERE pedidos.valor > 1000;
```

Capítulo 10

Evite o Uso Excessivo de DISTINCT

Evite o Uso Excessivo de DISTINCT

O uso excessivo de DISTINCT pode ser um sinal de que sua query ou estrutura de dados precisa ser revisada para evitar duplicatas desnecessárias.

Exemplos em PostgreSQL / SQL Server:

```
SELECT DISTINCT nome FROM clientes;
```

Capítulo 11

Manutenção e Atualização de Estatísticas

Manutenção e Atualização de Estatísticas

Manter as estatísticas do banco de dados atualizadas garante que o otimizador de consultas tome as melhores decisões ao executar queries.

Exemplos em PostgreSQL:

```
ANALYZE;
```

Exemplos em SQL Server:

```
UPDATE STATISTICS;
```

Capítulo 12

Funções Pesadas

Funções Pesadas

Algumas funções podem impactar significativamente a performance das suas queries, pois exigem mais processamento. Conhecer essas funções e entender por que evitá-las pode ajudar a otimizar suas consultas.

a. **Função LIKE com Wildcards no Início**

Usar LIKE '%texto%' impede que os índices sejam utilizados de forma eficaz, pois o banco de dados precisa percorrer todas as linhas para encontrar as correspondências.

Exemplos em PostgreSQL / SQL Server:

```sql
SELECT nome FROM clientes WHERE nome LIKE '%João%';
```

Alternativa: Prefira usar wildcards no final ou utilize Full-Text Search.

```sql
-- Full-Text Search em PostgreSQL
SELECT nome FROM clientes WHERE to_tsvector(nome) @@
to_tsquery('João');

-- Full-Text Search em SQL Server
SELECT nome FROM clientes WHERE CONTAINS(nome, 'João');
```

Funções Pesadas

b. Função UPPER ou LOWER

Aplicar UPPER ou LOWER nas colunas na cláusula WHERE força o banco de dados a executar a função para cada linha, tornando a query mais lenta.

Exemplos em PostgreSQL / SQL Server:

```sql
SELECT nome FROM clientes WHERE UPPER(nome) = 'JOÃO';
```

Alternativa: Armazene os valores já em maiúsculas/minúsculas ou use colunas derivadas.

```sql
-- Criar coluna derivada em PostgreSQL
ALTER TABLE clientes ADD COLUMN nome_upper TEXT GENERATED ALWAYS AS
(UPPER(nome)) STORED;
CREATE INDEX idx_nome_upper ON clientes(nome_upper);
SELECT nome FROM clientes WHERE nome_upper = 'JOÃO';

-- Criar coluna derivada em SQL Server
ALTER TABLE clientes ADD nome_upper AS UPPER(nome);
CREATE INDEX idx_nome_upper ON clientes(nome_upper);
SELECT nome FROM clientes WHERE nome_upper = 'JOÃO';
```

Funções Pesadas

c. Função SUBSTRING

A função SUBSTRING aplicada na cláusula WHERE também impede o uso de índices e requer a leitura de todas as linhas para extrair e comparar a substring.

Exemplos em PostgreSQL / SQL Server:

```sql
SELECT nome FROM clientes WHERE SUBSTRING(nome, 1, 3) = 'Joã';
```

Alternativa: Crie uma coluna com a substring já calculada.

```sql
-- Criar coluna derivada em PostgreSQL
ALTER TABLE clientes ADD COLUMN nome_prefixo TEXT GENERATED ALWAYS
AS (LEFT(nome, 3)) STORED;
CREATE INDEX idx_nome_prefixo ON clientes(nome_prefixo);
SELECT nome FROM clientes WHERE nome_prefixo = 'Joã';

-- Criar coluna derivada em SQL Server
ALTER TABLE clientes ADD nome_prefixo AS LEFT(nome, 3);
CREATE INDEX idx_nome_prefixo ON clientes(nome_prefixo);
SELECT nome FROM clientes WHERE nome_prefixo = 'Joã';
```

Funções Pesadas

d. Funções Matemáticas Complexas

Funções matemáticas complexas, especialmente quando usadas em filtros, podem ser lentas e evitar o uso eficiente de índices.

Exemplos em PostgreSQL / SQL Server:

```sql
SELECT * FROM vendas WHERE LOG(preco) > 2;
```

Alternativa: Armazene os valores pré-calculados em colunas adicionais.

```sql
-- Criar coluna derivada em PostgreSQL
ALTER TABLE vendas ADD COLUMN log_preco NUMERIC GENERATED ALWAYS AS
(LOG(preco)) STORED;
CREATE INDEX idx_log_preco ON vendas(log_preco);
SELECT * FROM vendas WHERE log_preco > 2;

-- Criar coluna derivada em SQL Server
ALTER TABLE vendas ADD log_preco AS LOG(preco);
CREATE INDEX idx_log_preco ON vendas(log_preco);
SELECT * FROM vendas WHERE log_preco > 2;
```

Conclusão

Aplicar essas técnicas ajudará a garantir que suas consultas SQL sejam rápidas e eficientes. Praticar boas práticas de otimização, monitorar o desempenho e revisar regularmente suas queries são passos essenciais para manter um banco de dados performático. Continue explorando e experimentando com diferentes abordagens para encontrar o que funciona melhor para seu cenário específico. Boas práticas e revisões constantes são a chave para manter a excelência em consultas SQL.

Agradecimentos

Gostaria de expressar minha sincera gratidão a todos que se dedicaram a explorar e aplicar as técnicas apresentadas neste ebook. Esperamos que estas informações sejam valiosas para otimizar suas consultas SQL e melhorar a performance dos seus sistemas.

Agradeço especialmente por seu interesse e por investir seu tempo neste material. Seu compromisso com a melhoria contínua e busca por conhecimento é admirável e essencial para o sucesso em qualquer área.

Para acessar o passo a passo detalhado e consultar os exemplos de código, visite o meu GitHub:

https://github.com/Di-Castro/SQL_do_Ninja/

Desejo sucesso em todas as suas futuras empreitadas no mundo do SQL e da otimização de bancos de dados. Muito obrigado!