Instituto Federal de Educação, Ciência e Tecnologia da Bahia - IFBA
Departamento de Informática
Análise e Desenvolvimento de Sistemas / Licenciatura em Computação

Otimização de Desempenho em DBMSs

André L. R. Madureira <andre.madureira@ifba.edu.br>
Doutorando em Ciência da Computação (UFBA)
Mestre em Ciência da Computação (UFBA)
Engenheiro da Computação (UFBA)

Desempenho em DBMSs

- Por mais avançado que seja o DBMS, ele ainda pode sofrer gargalos de desempenho, devido à:
  - **Consultas SQL mal otimizadas**
  - **Tabelas mal projetadas**
  - **Limitações de hardware**
- Nós enquanto DBAs devemos nos atentar à esses problemas que podem prejudicar o desempenho do DB
  - Nosso alvo é **MITIGAR** esses problemas (próximos slides)

Fatores que afetam o desempenho do DBMS

- **Uso de SELECT ***
- **Uso de subconsultas aninhadas**
- **Ausência ou excesso de índices**
- **Filtros WHERE ineficientes**
- **JOINs mal otimizados**
- **Ausência ou excesso de normalização**
- **Tipos de dados ou chaves estrangeiras inapropriados(as)**
- **TRIGGERs em excesso**

Uso de SELECT *

- Traz dados desnecessários e consome recursos (DBMS e rede)
- **Ex:**
  - SELECT * FROM clientes;
- **Solução:**
  - Selecionar apenas as colunas necessárias para a consulta
  - **Ex:** SELECT nome, email FROM clientes;

Uso de subconsultas aninhadas

- Pode ser lento quando temos grandes volumes de dados no DB
- **Ex**: **SELECT** nome **FROM** clientes **WHERE** id **IN** (
  SELECT cliente_id **FROM** pedidos **WHERE** total > 1000
  );
- **Solução**: Usar JOINs ou WITH
  - **SELECT** c.nome
    FROM clientes c
    JOIN pedidos p **ON** c.id = p.cliente_id
    WHERE p.total > 1000;

Ausência de índices

- Índices são arquivos que melhoram o desempenho do DB
- Sem índices, o banco busca dados linha por linha (**full scan**)
  - **SELECT** *
    FROM usuarios
    WHERE email = 'teste@exemplo.com';
- **Solução:** Criar índices conforme necessário
  - **CREATE INDEX** idx_usuarios_email **ON** usuarios(email);

Nome do índice Tabela Atributos

Excesso de índices

- Muitos índices tornam **INSERT / UPDATE / DELETE** lentos
  - A cada operação que altera dados na tabela, **todos** os índices devem ser atualizados
- **Ex:** Tabela de notas fiscais, com 10 índices, sofre INSERT de vários caixas de um supermercado
- **Solução:** Avaliar a real necessidade de índices para a tabela
  - “As colunas mais frequentemente utilizadas em cláusulas **WHERE**, **ORDER BY, JOIN** e **HAVING** de consultas **SELECT** são as que mais precisam de índices”

Excesso de índices (exemplo)

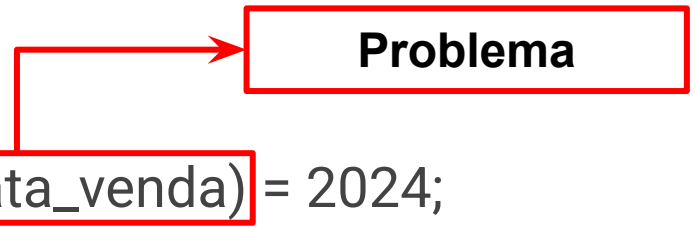
- Índices redundantes ou raramente úteis:
  - **CREATE INDEX** idx_usuarios_nome **ON** usuarios(nome);
    CREATE INDEX idx_usuarios_email **ON** usuarios(email);
    CREATE INDEX idx_usuarios_telefone **ON** usuarios(telefone);
    CREATE INDEX idx_usuarios_data **ON** usuarios(data_cadastro);
    CREATE INDEX idx_usuarios_ativo **ON** usuarios(ativo);
    CREATE INDEX idx_usuarios_email_nome **ON** usuarios(email, nome);
    CREATE INDEX idx_usuarios_nome_email **ON** usuarios(nome, email);

Excesso de índices (solução)

- Manter apenas os índices mais úteis:
  - **CREATE INDEX** idx_usuarios_email **ON** usuarios(email);
  - **CREATE INDEX** idx_usuarios_nome **ON** usuarios(nome);
- Sabemos que eles são úteis pois temos consultas SQL frequentes:
  - **SELECT** nome, email **FROM** usuarios **WHERE** nome **LIKE** ‘Ana%’
  - **SELECT** nome **FROM** usuarios **WHERE** email **LIKE** ‘ana%@gmail.com’

Filtros ineficientes

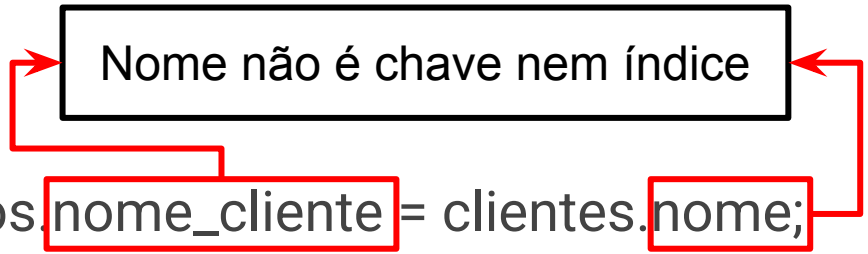
- Usar funções na cláusula **WHERE** impede uso de índices
  - **SELECT** *
    FROM vendas
    WHERE **YEAR**(data_venda) = 2024;



- **Solução**: reescrever a consulta para preservar o índice
  - **SELECT** *
    FROM vendas
    WHERE data_venda >= '2024-01-01'
    AND data_venda < '2025-01-01';

JOINs mal otimizados

- JOINs em tabelas grandes, sem índices ou com chaves mal escolhidas, geram operações custosas
  - **SELECT** *
    FROM pedidos
    JOIN clientes **ON** pedidos.nome_cliente = clientes.nome;



- **Solução:** Usar índices ou chaves nos JOINs
  - **SELECT** *
    FROM pedidos
    JOIN clientes **ON** pedidos.cliente_id = clientes.id;

Ausência ou excesso de normalização

- **No caso de excesso de normalização**
  - Precisamos de muitos JOINs para obter dados (nome do cliente, produto, categoria, etc)
- **No caso da falta de normalização**
  - Armazenamento repetido de dados em cada tabela (baixo desempenho e alto consumo de espaço em disco)
- **Solução:** balancear normalização e performance de acordo com a necessidade da aplicação

Tipos de dados inapropriados

- Por exemplo, usar **TEXT** onde um **VARCHAR**(255) bastaria
- **Ex:** Coluna **produtos.descricao** definida como **TEXT** sendo que descrição armazena no máximo 100 caracteres
- **Solução:** Ajustar o tipo de dados para o tipo mais adequado
  - **ALTER TABLE** produtos **MODIFY** descricao **VARCHAR**(100);

Chaves estrangeiras mal dimensionadas

- Campos muito longos como chave dificultam atualizações e inserções de registros nas tabelas
  - **Ex:** email **VARCHAR**(255) como chave estrangeira
- **Solução:** usar chaves estrangeiras menores
  - cpf **BIGINT** como chave estrangeira
  - **OU**
  - id **INT**

TRIGGERs em excesso

- Atrasam o tempo de resposta das operações **INSERT** / **UPDATE** / **DELETE**
  - **Ex:** Trigger para verificar cada **UPDATE** de **funcionario.salário**
- **Solução:** Avaliar a necessidade real e o custo do TRIGGER
  - *“Realmente precisamos desse trigger no DB?”*
  - *“Quantas vezes o trigger executará?”*
  - *“Qual o consumo adicional de CPU e disco do trigger?”*

Como avaliar o desempenho de um DBMS?

- Precisamos de técnicas para análise do desempenho do DBMS
  - **Comandos SQL para análise**
    - **EXPLAIN / EXPLAIN ANALYZE**
    - **SHOW PROFILE**
    - **SHOW ENGINE**
    - **SET GLOBAL**

Comando **EXPLAIN** / **EXPLAIN ANALYZE**

- **Mostra o plano de execução da consulta SQL**
  - Como o banco acessa os dados, se usa índice, se faz varredura completa dos registros da tabela, etc
- **EXPLAIN** faz uma análise de como o plano SERIA executado
  - **EXPLAIN SELECT * FROM** produtos;
- **EXPLAIN ANALYZE** executa a consulta e mostra o plano que **FOI** efetivamente executado
  - **EXPLAIN ANALYZE SELECT * FROM** produtos;

Exemplo de **EXPLAIN**

Suportado pelo MySQL e PostgreSQL

- Seja a seguinte tabela ***'pedidos':***

id	cliente	data_pedido	valor
1	Cliente 6707	2021-08-17	351.84
2	Cliente 8489	2023-01-27	360.58

- **EXPLAIN SELECT** cliente **FROM** pedidos **WHERE** valor > 0;

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	pedidos	ALL	*NULL*	*NULL*	*NULL*	*NULL*	498609	Using where

ALL = varredura completa da tabela (lenta)

Busca em todos os 498609 registros

Exemplo de **EXPLAIN**

- **EXPLAIN SELECT** cliente **FROM** pedidos **WHERE id = 8;**

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	pedidos	const	PRIMARY	PRIMARY	8	const	1	

const = varredura em tempo constante (rápida)

PRIMARY = usando chave primária

Busca verifica apenas 1 registro da tabela

Exemplo de **EXPLAIN**

- **CREATE INDEX** idx_cliente **ON** pedidos(cliente);
  EXPLAIN SELECT cliente **FROM** pedidos **WHERE cliente = 'Mark';**

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	pedidos	ref	idx_cliente	idx_cliente	403	const	1	Using where; Using index

ref = varredura por referência (rápida)

idx_cliente = usando índice com nome *idx_cliente*

Busca verifica apenas 1 registro da tabela

Exemplo de **EXPLAIN ANALYZE**

Suportado apenas pelo PostgreSQL

- **EXPLAIN ANALYZE SELECT** * **FROM** produtos **WHERE** preco > 100;
  - Seq Scan on produtos (cost=0.00..1.05 rows=10 width=12)
  - **Seq scan** = varredura completa da tabela (lenta)
- **CREATE INDEX** idx_preco **ON** produtos(preco);
- **EXPLAIN ANALYZE SELECT** * **FROM** produtos **WHERE** preco > 100;
  - Index Scan using idx_preco on produtos [...]
  - **Index scan** = varredura usando índice (rápida)

Exemplo de **SHOW PROFILE**

- **Mostra o tempo de execução real de uma consulta SQL.**
  - **MySQL:**
    - **SET** profiling = 1;
      SELECT * FROM pedidos **WHERE** cliente_id = 42;
      SHOW PROFILES;
      SHOW PROFILE FOR QUERY 1;
  - **PostgreSQL:**
    - **\timing**
      SELECT * FROM pedidos **WHERE** cliente_id = 42;

Exemplo de **SHOW ENGINE**

- **Mostra bloqueios, deadlocks e acessos concorrentes no DB**
  - **MySQL:**
    - **SHOW ENGINE INNODB STATUS;**
  - **PostgreSQL:**
    - **SELECT ***
      FROM pg_locks Pl
      JOIN pg_stat_activity Psa
      ON Pl.pid = Psa.pid;

Exemplo de **SET GLOBAL**

- **Mostra logs do DB (e consultas SQL lentas)**
  - **MySQL:**
    - **SET GLOBAL** slow_query_log = 'ON';
    - **SET GLOBAL** long_query_time = 1; **-- >= 1 segundo**
  - **PostgreSQL:**
    - No arquivo ***postgresql.conf*** adicione as seguintes linhas:
      - log_min_duration_statement = 1000 -- >= 1000 ms

Referencial Bibliográfico

- KORTH, H.; SILBERSCHATZ, A.; SUDARSHAN, S. **Sistemas de bancos de dados**. 5. ed. Rio de Janeiro: Ed. Campus, 2006.

- DATE, C. J. **Introdução a sistemas de bancos de dados**. Rio de Janeiro: Ed. Campus, 2004. Tradução da 8ª edição americana.