Banco de Dados II

Ronierison Maciel

Agosto 2024

Quem sou eu?

Nome: Ronierison Maciel / Roni
Formação: Mestre em Ciência da Computação
Ocupação: Pesquisador, Professor e Desenvolvedor de Software
Hobbies: Jogar cartas, ficar com a família no final de semana conversando sobre diversos temas
Interesses: Carros, aprimoramento na área educacional, desenvolvimento de software, data science e machine learning
Email: ronierison.maciel@pe.senac.br
GitHub: `https://github.com/ronierisonmaciel`

Conteúdo

Revisão de conceitos fundamentais

O que é um Banco de Dados?

- Um banco de dados é uma coleção organizada de dados que podem ser facilmente acessados, gerenciados e atualizados.
- É utilizado para armazenar informações de maneira estruturada, permitindo que dados relacionados sejam conectados.

Exemplos:

- Um sistema de gerenciamento de alunos de uma escola (informações sobre alunos, cursos, notas).
- Um banco de dados de clientes de uma empresa (informações sobre clientes, pedidos, produtos).

Por que utilizar um banco de dados?

Vantagens:

- **Organização e estrutura:** Facilita a organização dos dados em um formato que permite consultas eficientes.
- **Integridade e consistência:** Garante que os dados sejam precisos e consistentes ao longo do tempo.
- **Segurança:** Protege dados contra acessos não autorizados.
- **Escalabilidade:** Suporta o crescimento dos dados sem perda de performance.
- **Compartilhamento:** Permite que vários usuários acessem os dados simultaneamente.

Tipos de Bancos de Dados

- **Relacional:** Dados organizados em tabelas, que são relacionadas entre si. Exemplo: MySQL, PostgreSQL.
- **NoSQL:** Projetado para lidar com grandes volumes de dados não estruturados. Exemplo: MongoDB.
- **Orientado a Objetos:** Integra conceitos de orientação a objetos com bancos de dados. Exemplo: db4o.
- **Distribuído:** Dados armazenados em múltiplos locais geográficos. Exemplo: Apache Cassandra.

O que é um Sistema de Gerenciamento de Banco de Dados (SGBD)?

Definição:

- Um SGBD é um software que permite a criação, gerenciamento e manipulação de bancos de dados.
- Facilita a organização dos dados, permitindo consultas eficientes e garantindo a integridade e segurança dos dados.

Componentes principais de um SGBD:

- **Processador de consultas:** Interpreta e executa comandos SQL enviados pelos usuários.
- **Gestor de armazenamento:** Gerencia a forma como os dados são armazenados no disco, otimizando o acesso e recuperação dos dados.
- **Gerenciador de transações:** Garante que as operações realizadas no banco de dados sejam consistentes e concluídas corretamente.
- **Segurança e integridade:** Controla o acesso aos dados, garantindo que apenas usuários autorizados possam manipular informações sensíveis.

Tipos de SGBD

- **Relacional:** Baseado em tabelas e relações entre elas. Exemplo: MySQL, PostgreSQL.
- **Orientado a Objetos:** Combina conceitos de orientação a objetos com bancos de dados. Exemplo: db4o.
- **Distribuído:** Gerencia dados distribuídos em múltiplos servidores ou locais. Exemplo: Google Spanner.
- **NoSQL:** Adequado para grandes volumes de dados que não seguem um esquema fixo. Exemplo: MongoDB.

Comandos SQL

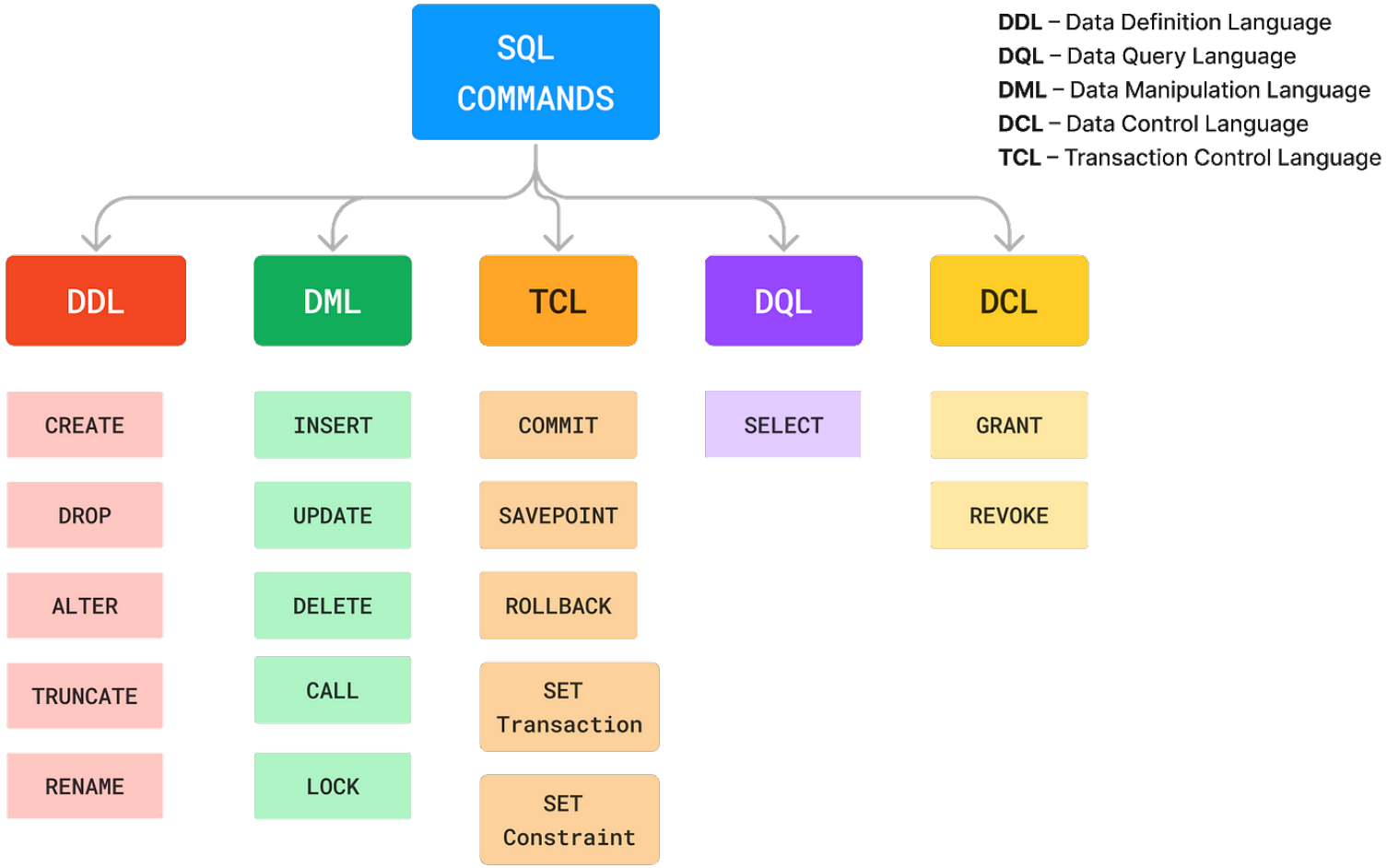


Figure: Tipos de comandos SQL

Configuração do ambiente de trabalho

- **Escolha do SGBD:** Decidimos utilizar MySQL, uma plataforma robusta e amplamente utilizada no mercado.
- **Instalação e Configuração:** Instalação do SGBD: Orientações sobre como instalar o SGBD no sistema operacional utilizado (Windows, macOS, Linux).
- **Configuração inicial:** Passos para configurar a instância do SGBD, como definir usuário, senha e configurar permissões básicas.
- **Ferramentas adicionais:** SQL Workbench Ferramentas gráficas para interação com o SGBD. Facilita a execução de comandos SQL, visualização de dados e administração do banco.

Criação de um Banco de Dados simples

- **Conectar ao SGBD:** Acessar a interface do SGBD e conectar utilizando as credenciais configuradas.
- **Criar um Banco de Dados:** Comando SQL para criar um novo banco de dados chamado escola.

```sql
CREATE DATABASE escola;
```

- **Criar uma tabela alunos:** Definir uma tabela chamada alunos com os seguintes campos:

```
id (INT, Primary Key)
nome (VARCHAR(100))
data_nascimento (DATE)
```

Criação de um Banco de Dados simples

- **Inserir dados na tabela:** Adicionar alguns registros à tabela alunos.

```
1 INSERT INTO alunos (id, nome, data_nascimento) VALUES (1, '
    Paulo Roberto', '2001-05-20');
2 INSERT INTO alunos (id, nome, data_nascimento) VALUES (2, '
    Maria Oliveira', '2000-11-15');
```

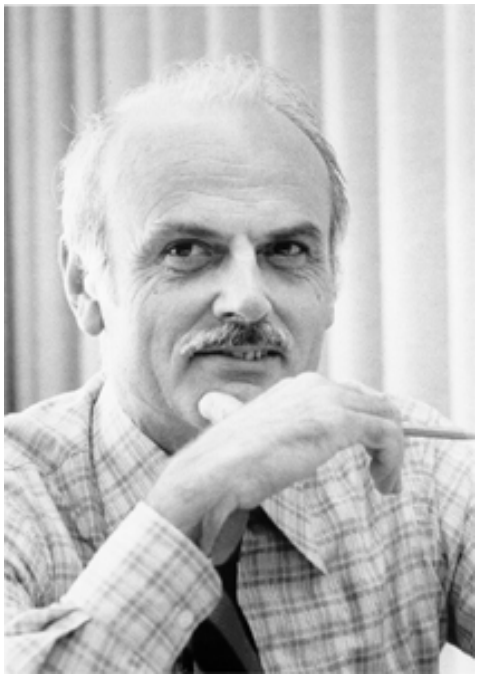
- **Realizar uma consulta básica:** Executar uma consulta para selecionar todos os registros da tabela alunos.

```
1 SELECT * FROM alunos;
```

Reflexão

- **Importância de um SGBD:** Discutir como um SGBD organiza e protege os dados em comparação com outros métodos de armazenamento.
- **Cenários de uso:** Explorar diferentes cenários em que bancos de dados relacionais, NoSQL e outros tipos de SGBD são utilizados.
- **Desafios na administração de Banco de Dados:** Considerar questões como escalabilidade, segurança e manutenção.

Modelo de dados relacional



O Que é o modelo de dados relacional?

O modelo relacional organiza os dados em tabelas chamadas relações. Cada tabela contém linhas, chamadas tuplas, e colunas, chamadas atributos. Esse modelo foi proposto por Edgar F. Codd em 1970 e se tornou a base para a maioria dos SGBDs modernos.

Componentes de uma relação

- **Tabela (relação):** Estrutura principal onde os dados são armazenados em linhas e colunas.
- **Linha (tupla):** Um registro na tabela, representando uma única instância de dados.
- **Coluna (atributo):** Um campo na tabela, representando um tipo de dado específico.

Componentes de uma relação

Chaves em bancos de dados relacionais:

- **Chave primária (Primary Key):** Um ou mais atributos que identificam unicamente cada tupla (linha) em uma tabela.
- **Chave estrangeira (Foreign Key):** Um atributo em uma tabela que referencia a chave primária de outra tabela, estabelecendo uma relação entre as duas

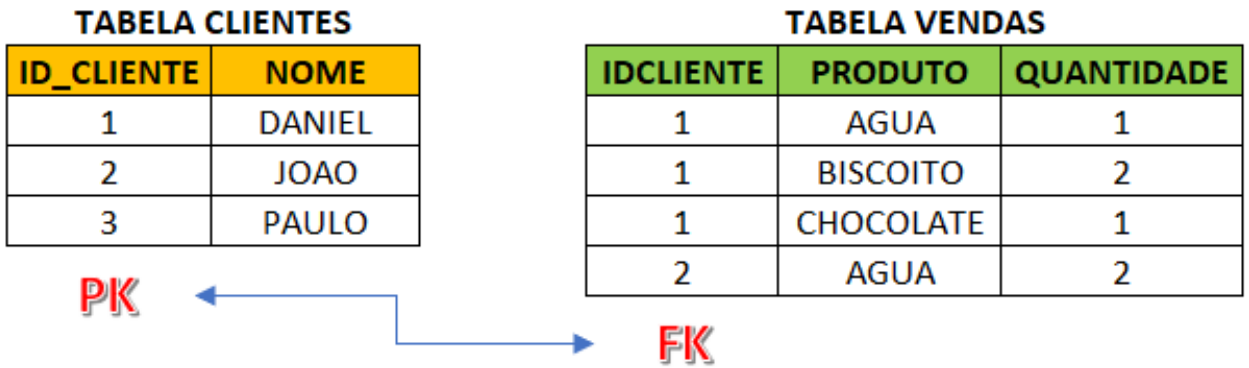


Figure: Foreign Key (FK) and Primary Key (PK)

Introdução ao SQL

- **SQL (Structured Query Language)**

Linguagem padrão utilizada para comunicação com um banco de dados relacional.

Principais comandos:

- **DDL (Data Definition Language):** Usado para definir a estrutura do banco de dados (CREATE, ALTER, DROP).
- **DML (Data Manipulation Language):** Usado para manipular dados dentro da tabela (SELECT, INSERT, UPDATE, DELETE).
- **DCL (Data Control Language):** Usado para controlar o acesso aos dados (GRANT, REVOKE).
- **TCL (Transaction Control Language):** Usado para gerenciar transações (COMMIT, ROLLBACK).

Consultas básicas com SQL

- Comando **SELECT**: Usado para recuperar dados de uma ou mais tabelas.

```sql
SELECT nome, data_nascimento FROM alunos;
```

- Comando **INSERT**: Usado para inserir novos dados em uma tabela.

```sql
INSERT INTO alunos (id, nome, data_nascimento) VALUES (3, '
    Carlos Mendes', '1999-12-10');
```

- Comando **UPDATE**: Usado para atualizar dados existentes em uma tabela.

```sql
UPDATE alunos SET nome = 'João Pedro' WHERE id = 1;
DELETE FROM alunos WHERE id = 2;
```

Consultas complexas com SQL

As consultas com múltiplas tabelas (JOINs) pode ser feita da seguinte forma.

- **INNER JOIN:** Retorna as linhas onde há correspondência em ambas as tabelas.

```sql
SELECT alunos.nome, cursos.nome_curso
FROM alunos
INNER JOIN cursos ON alunos.curso_id = cursos.curso_id;
```

- **LEFT JOIN:** Retorna todas as linhas da tabela à esquerda e as correspondências da tabela à direita. Se não houver correspondência, retorna NULL.

```sql
SELECT alunos.nome, cursos.nome_curso
FROM alunos
LEFT JOIN cursos ON alunos.curso_id = cursos.curso_id;
```

Consultas complexas com SQL

- **RIGHT JOIN:** Retorna todas as linhas da tabela à direita e as correspondências da tabela à esquerda. Se não houver correspondência, retorna NULL.

```sql
SELECT alunos.nome, cursos.nome_curso
FROM alunos
RIGHT JOIN cursos ON alunos.curso_id = cursos.curso_id;
```

Subconsultas

- **Definição:** Uma subconsulta é uma consulta dentro de outra consulta, usada para retornar dados que serão utilizados pela consulta principal.
- **Tipos de subconsultas:** Subconsulta de uma única linha: Retorna apenas um valor.

```
1 SELECT nome FROM alunos WHERE curso_id = (SELECT curso_id
    FROM cursos WHERE nome_curso = 'Engenharia de Software')
    ;
```

- **Subconsulta de múltiplas linhas:** Retorna mais de um valor.

```
1 SELECT nome FROM alunos WHERE curso_id IN (SELECT curso_id
    FROM cursos WHERE nome_curso LIKE 'Ciência%');
```

Subconsultas

- **Subconsulta correlacionada:** A subconsulta depende dos valores da consulta principal.

```
1 SELECT nome FROM alunos a WHERE EXISTS (SELECT 1 FROM cursos
      c WHERE c.curso_id = a.curso_id AND c.nome_curso = '
    Engenharia de Software');
```

Operações de agregação

- **Funções de agregação:** COUNT: Conta o número de linhas.

```sql
SELECT COUNT(*) FROM alunos;
```

- **SUM:** Soma os valores de uma coluna.

```sql
SELECT SUM(curso_id) FROM alunos;
```

- **AVG:** Calcula a média dos valores de uma coluna.

```sql
SELECT AVG(curso_id) FROM alunos;
```

- **MAX e MIN:** Retorna o valor máximo ou mínimo de uma coluna.

```sql
SELECT MAX(curso_id), MIN(curso_id) FROM alunos;
```

Views (Tabelas Virtuais)

Uma view é uma tabela virtual criada a partir de uma consulta SQL. Ela não armazena dados fisicamente, mas sim uma consulta que pode ser usada como se fosse uma tabela.

Vantagens das views:

- **Simplificação**: Facilita consultas complexas com um comando simples.
- **Segurança**: Pode restringir o acesso a determinadas colunas ou linhas.
- **Reutilização**: A mesma view pode ser usada em diferentes partes do sistema.

Criação e manipulação de views:

Exemplo:

```sql
CREATE VIEW alunos_cursos AS
SELECT alunos.nome, cursos.nome_curso
FROM alunos
JOIN cursos ON alunos.curso_id = cursos.curso_id;
```

Views (Tabelas Virtuais)

Consulta de uma view:

Exemplo:

```
1 SELECT * FROM alunos_cursos;
```

Atualização de uma view: Algumas views podem ser atualizadas diretamente, mas isso depende da complexidade da view.

Exemplo:

```
1 UPDATE alunos_cursos SET nome_curso = 'Engenharia de Computa
    ção' WHERE nome = 'João Silva';
```

Exclusão de uma view:

Exemplo:

```
1 DROP VIEW alunos_cursos;
```

Mão na massa

Consultas utilizando diferentes tipos de JOINs

Objetivo:

- Realizar consultas utilizando diferentes tipos de JOINs.

Consultas:

1. **INNER JOIN**: Listar todos os alunos e seus respectivos cursos.

```sql
SELECT alunos.nome, cursos.nome_curso
FROM alunos
INNER JOIN cursos ON alunos.curso_id = cursos.curso_id;
```

1. **LEFT JOIN**: Listar todos os alunos, incluindo aqueles que não estão matriculados em nenhum curso.

```sql
SELECT alunos.nome, cursos.nome_curso
FROM alunos
LEFT JOIN cursos ON alunos.curso_id = cursos.curso_id;
```

Continua...

Continuação

1. **RIGHT JOIN:** Listar todos os cursos, incluindo aqueles que não têm alunos matriculados.

```sql
SELECT alunos.nome, cursos.nome_curso
FROM alunos
RIGHT JOIN cursos ON alunos.curso_id = cursos.curso_id;
```

1. **FULL OUTER JOIN (simulado com UNION):** Listar todos os alunos e cursos, incluindo os que não têm correspondência.

```sql
SELECT alunos.nome, cursos.nome_curso
FROM alunos
LEFT JOIN cursos ON alunos.curso_id = cursos.curso_id
UNION
SELECT alunos.nome, cursos.nome_curso
FROM alunos
RIGHT JOIN cursos ON alunos.curso_id = cursos.curso_id;
```

Trabalhando com subconsultas

Objetivo: Realizar consultas utilizando subconsultas.

Consultas:

1. **Subconsulta de única linha:**
   - Encontrar os alunos que estão matriculados no curso de Engenharia de Software.

```
SELECT nome FROM alunos WHERE curso_id = (SELECT
    curso_id FROM cursos WHERE nome_curso = '
    Engenharia de Software');
```

1. **Subconsulta de múltiplas linhas:**
   - Listar os alunos que estão matriculados em cursos que começam com "Ciência".

```
SELECT nome FROM alunos WHERE curso_id IN (SELECT
    curso_id FROM cursos WHERE nome_curso LIKE 'Ciê
    ncia%');
```

Continua...

Continuação

1. **Subconsulta correlacionada:**
   - Listar os alunos que estão matriculados em cursos específicos com base em critérios dinâmicos.

```sql
SELECT nome FROM alunos a WHERE EXISTS (SELECT 1
    FROM cursos c WHERE c.curso_id = a.curso_id AND c
    .nome_curso = 'Engenharia de Software');
```

Trabalhando com views

Objetivo: Criar e utilizar views para simplificar consultas complexas.

Atividades:

1. **Criação de uma View:**
   - Criar uma view para simplificar a consulta de alunos e seus cursos.

```sql
CREATE VIEW alunos_cursos AS
SELECT alunos.nome, cursos.nome_curso
FROM alunos
JOIN cursos ON alunos.curso_id = cursos.curso_id;
```

1. **Consulta de uma View:**
   - Usar a view alunos_cursos para listar todos os alunos e seus cursos.

```sql
SELECT * FROM alunos_cursos;
```

Continua...

1. **Atualização através de uma View:**
   - Tentar atualizar um curso diretamente pela view (depende da complexidade da view).

```sql
UPDATE alunos_cursos SET nome_curso = 'Engenharia de Computação' WHERE nome = 'João Silva';
```

1. **Exclusão de uma View:**
   - Excluir a view criada.

```sql
DROP VIEW alunos_cursos;
```

Álgebra relacional

- Álgebra relacional é um conjunto de operações matemáticas que são usadas para trabalhar com dados armazenados em bancos de dados relacionais. Ela fornece um framework formal e teórico para descrever as operações de consulta e manipulação de dados. Essas operações são a base para as linguagens de consulta de banco de dados, como **SQL** (Structured Query Language).
- A álgebra relacional é considerada uma linguagem procedural porque descreve como os dados devem ser obtidos e processados. Cada operação na álgebra relacional toma uma ou mais tabelas como entrada e gera uma nova tabela como saída.

Operações Unárias

Operações Unárias são aquelas que trabalham com uma única tabela ou conjunto de dados por vez. Elas realizam filtragens e seleções dentro de uma única relação.

- **Seleção:** A operação de seleção serve para filtrar linhas em uma tabela com base em uma condição específica.
  - **Por exemplo**, selecionar todos os alunos que nasceram depois do ano 2000. Ela ajuda a focar apenas nas tuplas (linhas) que atendem a determinados critérios.
- **Projeção:** A projeção é usada para selecionar apenas certas colunas de uma tabela, ou seja, atributos específicos.
  - **Por exemplo**, extrair apenas os nomes e datas de nascimento dos alunos. Ela é útil para simplificar a visualização dos dados, mostrando apenas as informações relevantes.
- **Renomeação:** A operação de renomeação permite alterar o nome de uma tabela ou de suas colunas.
  - Isso pode ser útil para dar nomes mais descritivos às tabelas ou colunas, tornando os dados mais compreensíveis e organizados.

Operações Binárias

Operações Binárias envolvem a combinação de duas tabelas ou conjuntos de dados. Elas são usadas para comparar ou combinar dados provenientes de diferentes relações.

- **União:** A união combina todas as linhas de duas tabelas que têm o mesmo tipo de dados, eliminando as duplicatas.
  - É como juntar listas de alunos de duas turmas diferentes em uma única lista, mantendo apenas um registro de cada aluno.
- **Diferença:** A diferença mostra as linhas que estão em uma tabela, mas não estão na outra.
  - Por exemplo, se quisermos saber quais alunos estão matriculados em um curso específico, mas não em outro, essa operação nos mostra apenas os que estão na primeira tabela, mas não aparecem na segunda.

Continua...

Continuação

- **Produto Cartesiano:** Essa operação combina cada linha de uma tabela com todas as linhas de outra tabela, criando todas as combinações possíveis.
  - **Por exemplo**, associando cada aluno com todos os cursos oferecidos, independentemente de qual curso cada aluno esteja realmente inscrito. É uma operação base para outras mais específicas, como junção.
- **Junção:** A junção é uma operação que combina linhas de duas tabelas baseando-se em um critério comum, como uma chave estrangeira que conecta alunos a seus cursos.
  - Ela é essencial para relacionar dados distribuídos em diferentes tabelas de forma significativa e útil.

Conclusão

Pra quê serve essas operações?

1. Manipular dados;
2. Consultar dados;
3. Trabalhar em uma única tabela e
4. Comparar dados entre múltiplas tabelas.

Compreender essas operações nos permite realizar consultas complexas e obter ideias valiosos a partir dos dados armazenados.

Mão na massa

Operações unárias (seleção, projeção e renomeação)

Seleção: Alunos nascidos após o ano 2000

```sql
SELECT * FROM alunos WHERE data_nascimento > '2000-01-01';
```

Projeção: Apenas os nomes e as datas de nascimento dos alunos

```sql
SELECT nome, data_nascimento FROM alunos;
```

Renomeação: Renomear a coluna nome para estudante_nome

```sql
SELECT nome AS estudante_nome, data_nascimento FROM alunos;
```

Operações binárias (união, diferença, produto cartesiano)

União: Combinação de registros de duas tabelas semelhantes

```sql
SELECT nome, data_nascimento FROM alunos
UNION
SELECT nome, data_nascimento FROM alunos_b;
```

Diferença: Alunos em uma tabela mas não na outra

```sql
SELECT nome, data_nascimento FROM alunos
EXCEPT
SELECT nome, data_nascimento FROM alunos_b;
```

Produto cartesiano: Todas as combinações possíveis entre alunos e cursos

```sql
SELECT * FROM alunos, cursos;
```

Operação de junção

Junção: Juntar alunos com seus respectivos cursos

```sql
SELECT alunos.nome, cursos.nome_curso
FROM alunos
JOIN cursos ON alunos.curso_id = cursos.curso_id;
```

Cálculo relacional

Cálculo relacional de tupla: Alunos com idade maior que 20 anos

```
1 SELECT nome FROM alunos WHERE (YEAR(CURDATE()) - YEAR(
    data_nascimento)) > 20;
```

Cálculo relacional de domínio: Alunos cujo curso é 'Engenharia de Software'

```
1 SELECT nome FROM alunos WHERE curso_id = (SELECT curso_id
    FROM cursos WHERE nome_curso = 'Engenharia de Software')
    ;
```

Índices (Indexes)

Índices são usados para aumentar a performance das consultas, especialmente em grandes bases de dados.
Exemplo:

```
1 CREATE INDEX idx_cliente_nome ON Cliente(nome);
```

Utilizamos índices para melhorar a performance de consultas que pesquisam por nomes de clientes, permitindo que o banco de dados localize rapidamente as linhas que contêm valores específicos na coluna nome.

Triggers

Triggers são funções automáticas que são executadas em resposta a eventos como INSERT, UPDATE, ou DELETE em uma tabela. Criar um triggers serve para auditar alterações em dados ou aplicar regras de negócios.

Triggers

Exemplo:

```
1 DELIMITER //
2 CREATE TRIGGER before_cliente_update
3 BEFORE UPDATE ON Cliente
4 FOR EACH ROW
5 BEGIN
6     INSERT INTO auditoria_cliente (cliente_id, alteracao,
      data)
7     VALUES (OLD.cliente_id, 'Modificado', NOW());
8 END //
9 DELIMITER ;
```

Usamos triggers para criar auditorias automáticas de alterações. Sempre que um cliente for atualizado, o trigger insere um registro na tabela auditoria_cliente, registrando **quem** foi alterado e **quando**.

Stored Procedures

Stored procedures são conjuntos de comandos SQL que podem ser armazenados e reutilizados. Elas permitem encapsular lógica de negócios no banco de dados. Uma stored procedure garante a realização de uma tarefa complexa, como atualizar múltiplas tabelas.

Stored Procedures

Exemplo:

```
1  DELIMITER //
2  CREATE PROCEDURE AtualizarTelefone(IN cliente_id INT, IN
       novo_telefone VARCHAR(20))
3  BEGIN
4      UPDATE Cliente
5      SET telefone = novo_telefone
6      WHERE cliente_id = cliente_id;
7  END //
8  DELIMITER ;
9
10 CALL AtualizarTelefone(1, '1234-5678');
```

Os stored procedures encapsulam lógica de negócios no banco de dados. Um desenvolvedor pode chamar a procedure para atualizar o número de telefone de um cliente específico, sem precisar reescrever o código SQL sempre.

Funções Definidas pelo Usuário (UDF)

MySQL permite a criação de funções personalizadas que podem ser usadas como parte das consultas. Cria uma função que calcule o tempo desde o cadastro de um cliente.

Exemplo:

```
1 DELIMITER //
2 CREATE FUNCTION CalcularIdade(data_nascimento DATE)
3 RETURNS INT
4 BEGIN
5     RETURN YEAR(CURDATE()) - YEAR(data_nascimento);
6 END //
7 DELIMITER ;
8
9 SELECT nome, CalcularIdade(data_nascimento) AS idade FROM
    Cliente;
```

Criamos funções personalizadas para reutilizar lógica comum, como calcular a idade de clientes a partir da data de nascimento.

Views complexas

As views podem ser usadas para criar consultas complexas e simplificar a apresentação dos dados para os usuários finais. Elas podem encapsular lógica complexa e abstrair o acesso ao banco de dados. Criar uma view que combine várias tabelas e inclua subconsultas.

Exemplo:

```
1 CREATE VIEW ClientesComEndereco AS
2 SELECT c.nome, e.rua, e.cidade, e.estado
3 FROM Cliente_2 c
4 JOIN Endereco e ON c.endereco_id = e.id;
5
6 SELECT * FROM ClientesComEndereco;
```

As views podem/deve ser usadas para simplificar consultas complexas, permitindo ao desenvolvedor abstrair a lógica de junção (JOIN) e acessar informações de clientes e seus endereços de forma fácil e reutilizável.

Full-Text Search

O MySQL oferece suporte à pesquisa de texto completo, que permite encontrar palavras ou frases dentro de textos armazenados em colunas. É possível criar um índice de texto completo em uma coluna e mostrar como realizar buscas de texto avançadas.

Exemplo:

```
1 CREATE FULLTEXT INDEX idx_texto ON artigos(conteudo);
2 SELECT * FROM artigos WHERE MATCH(conteudo) AGAINST('
    palavras chave');
```

A pesquisa de texto completo é usada para localizar rapidamente textos ou palavras-chave em grandes volumes de conteúdo, como artigos, blogs ou redes sociais.

Objetivos

Objetivos:

- Entender e aplicar técnicas de otimização de consultas.
- Compreender como funcionam transações e controle de concorrência.
- Implementar replicação de banco de dados.

Otimização de consultas (Query Optimization)

Importância da Otimização de Consultas

- **Performance de sistemas**: Consultas mal otimizadas podem causar lentidão em aplicações, especialmente com grandes volumes de dados.
- **Uso eficiente de recursos**: O tempo de CPU e memória são limitados, e consultas eficientes reduzem o consumo desses recursos.
- **Escalabilidade**: Em ambientes de produção, otimizar consultas é essencial para permitir que o sistema cresça sem comprometer a performance.

Comando EXPLAIN no MySQL

O comando EXPLAIN permite analisar o plano de execução de uma consulta, mostrando como o MySQL irá processar a mesma.

- **id**: Identifica a ordem de execução.
- **select_type**: Tipo de consulta (simples, subconsulta, etc.).
- **table**: Tabela referenciada.
- **type**: Tipo de acesso (ALL, index, range, etc.).
- **possible_keys**: Índices possíveis de serem usados.
- **key**: Índice realmente utilizado.
- **rows**: Número estimado de linhas que serão lidas.

```
1 EXPLAIN SELECT * FROM clientes WHERE cidade = 'Recife';
```

Técnicas de otimização

Uso de índices:

- Criar índices em colunas frequentemente usadas em cláusulas WHERE e JOIN.
- **Exemplo**: Índice para a coluna cidade na tabela clientes.

```
1 CREATE INDEX idx_cidade ON clientes(cidade);
```

Reestruturação de consultas:

- Evitar o uso de SELECT * em grandes tabelas, especificando as colunas necessárias.
- **Exemplo**:

```
1 SELECT nome, cidade FROM clientes WHERE cidade = 'Recife';
```

Continua...

Eliminação de subconsultas desnecessárias:

- Subconsultas podem ser lentas; use JOIN em vez de subconsultas quando possível.
- Exemplo:

```
1 -- Evitar:
2 SELECT * FROM clientes WHERE id IN (SELECT cliente_id FROM
      pedidos);
3
4 -- Melhor opção com JOIN:
5 SELECT c.* FROM clientes c JOIN pedidos p ON c.id = p.
      cliente_id;
```

Mão na massa

Objetivo: Aplicar a otimização de consultas usando EXPLAIN e índices.

- **Passo 1**: Execute a seguinte consulta sem o uso de índice:

```
SELECT * FROM pedidos WHERE data_pedido = '2024-10-01';
```

- **Passo 2**: Use o comando EXPLAIN para analisar o plano de execução.
- **Passo 3**: Crie um índice para a coluna data_pedido:

```
CREATE INDEX idx_data_pedido ON pedidos(data_pedido);
```

- **Passo 4**: Execute novamente a consulta e use o EXPLAIN para comparar os resultados.
- **Passo 5**: Reestruture a consulta para selecionar apenas as colunas id e data_pedido e observe a diferença no tempo de execução.

Transações e controle de concorrência

O que são transações?

- Transação: Uma unidade de trabalho que consiste em uma ou mais operações no banco de dados, executadas de forma que todas ou nenhuma sejam realizadas.
- Atomicidade: Garante que todas as operações da transação sejam completadas com sucesso ou, em caso de falha, revertidas (rollback).
- Integridade: Garante que o banco de dados permaneça em um estado válido antes e depois da transação.

Níveis de isolamento

Objetivo: Controlar como e quando as alterações feitas por uma transação se tornam visíveis para outras transações.

- **READ UNCOMMITTED**: Permite que transações leiam dados não confirmados (dirty reads), resultando em inconsistências.
- **READ COMMITTED**: Garante que uma transação só pode ler dados confirmados. Elimina o dirty read, mas permite não-repetibilidade.
- **REPEATABLE READ**: Garante que todas as leituras feitas durante a transação retornem o mesmo valor. Evita leituras não repetíveis, mas pode ocorrer phantom reads.
- **SERIALIZABLE**: O nível mais restrito, força a execução de transações como se fossem serializadas. Elimina phantom reads, mas pode impactar a performance devido ao uso de bloqueios.

O que é Deadlocks?

Ocorre quando duas ou mais transações ficam bloqueadas indefinidamente esperando por recursos que uma delas possui.

- **Como evitar Deadlocks?**:
  - Ordenação dos recursos: Garantir que todas as transações adquiram os bloqueios na mesma ordem.
  - Timeout: Definir um tempo limite para transações que esperam por bloqueios.
  - Transações curtas: Manter as transações pequenas e rápidas para evitar que muitos bloqueios sejam mantidos ao mesmo tempo.

Replicação de Banco de Dados (Database Replication)

O que é replicação de Banco de Dados?

- **Replicação**: O processo de copiar e manter dados de um banco de dados em múltiplos servidores.

Objetivo:

- **Alta disponibilidade (HA)**: Garante que o sistema continue funcionando mesmo que um servidor falhe.
- **Balanceamento de carga**: Distribui as operações de leitura entre múltiplos servidores.
- **Recuperação de desastres**: Protege os dados replicando-os para um servidor remoto.

Modelos de replicação

Mestre-escravo (Master-Slave):

- O servidor mestre processa as operações de escrita e envia essas alterações para um ou mais servidores escravos.
- Os escravos são usados para operações de leitura, melhorando a performance.

Mestre-mestre (Master-Master):

- Ambos os servidores podem realizar operações de escrita e leitura, garantindo redundância total.
- Mais complexo devido à possibilidade de conflitos de dados.

Configurações típicas de replicação no MySQL

Servidor mestre:

- Configura variáveis como server_id e ativa o log binário (binlog).
- Exemplo de configuração no my.cnf:

```
[mysqld]
server-id=1
log-bin=mysql-bin
```

Servidor escravo:

- Define um server_id diferente e conecta-se ao mestre com o comando CHANGE MASTER TO.
- Exemplo de configuração no my.cnf:

```
[mysqld]
server-id=2
relay-log=mysql-relay-bin
```

Configuração do escravo:

- Indique o mestre no escravo com o comando: