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: ronierisonsouza@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.

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:

```sql
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');
```

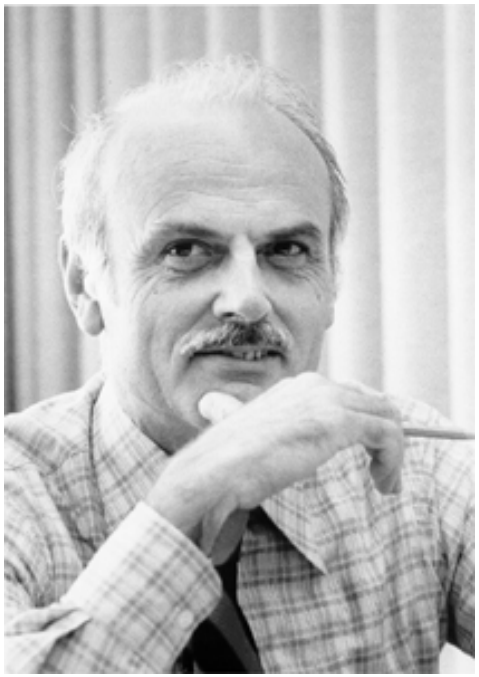
- **Realizar uma consulta básica:** Executar uma consulta para selecionar todos os registros da tabela alunos.

```
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 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

Exemplo: *id* na tabela *alunos*.

Exemplo: *curso_id* na tabela *alunos*, referenciando a tabela *cursos*.

Normalização

- **Objetivo:** Processo de organizar os dados para minimizar a redundância e evitar anomalias de inserção, atualização e exclusão.
- **Formas normais:** Primeira Forma Normal (1NF): Elimina grupos repetitivos, garantindo que cada valor atômico.
- **Segunda Forma Normal (2NF):** Elimina dependências parciais em tabelas compostas.
- **Terceira forma normal (3NF):** Elimina dependências transitivas, garantindo que atributos não-chave dependam apenas da chave primária.

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;
```

- **FULL OUTER JOIN:** Retorna todas as linhas quando há correspondência em uma das tabelas. Se não houver correspondência, retorna NULL.

```sql
SELECT alunos.nome, cursos.nome_curso
FROM alunos
FULL OUTER 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

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:** Listar todos os alunos e cursos, incluindo os que não têm correspondência.

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