semana 15
SGBD relacional
Aula 1: Introdução aos SGBDs relacionais
Entendendo as vantagens dos SGBDs relacionais com MySQL
Normalização de banco de dados

A construção de um banco de dados robusto e eficiente depende de uma fundação sólida, e dois conceitos são cruciais nesse processo: os **Diagramas de Entidade-Relacionamento (ER)** e a **Normalização de Dados**. Enquanto o primeiro oferece uma representação visual da estrutura do banco de dados, o segundo garante a integridade e a consistência dos dados armazenados.

Diagramas ER: A Planta Baixa do seu Banco de Dados

O Diagrama de Entidade-Relacionamento é uma ferramenta de modelagem conceitual que descreve a estrutura de um banco de dados de forma gráfica e intuitiva. Ele é composto por três elementos principais:

- **Entidades:** Representam objetos do mundo real sobre os quais desejamos armazenar informações. Podem ser pessoas (Clientes, Alunos), objetos (Produtos, Carros) ou conceitos abstratos (Vendas, Matrículas). No diagrama, as entidades são geralmente representadas por retângulos.
- **Atributos:** São as propriedades ou características que descrevem uma entidade. Por exemplo, a entidade "Cliente" pode ter os atributos "Nome", "CPF" e "Endereço". Os atributos são comumente representados por elipses conectadas à sua entidade.
- **Relacionamentos:** Indicam como as entidades se associam entre si. Um "Cliente" pode "realizar" uma "Venda", e uma "Venda" "contém" "Produtos". Os relacionamentos são tipicamente representados por losangos, conectando as entidades envolvidas.

Um aspecto fundamental dos relacionamentos é a **cardinalidade**, que define a quantidade de instâncias de uma entidade que podem se relacionar com instâncias de outra entidade. A cardinalidade pode ser de três tipos principais:

- **Um-para-Um (1:1):** Cada instância de uma entidade se relaciona com no máximo uma instância de outra entidade. Por exemplo, um "Marido" tem uma única "Esposa" e vice-versa.
- **Um-para-Muitos (1:N):** Uma instância de uma entidade pode se relacionar com várias instâncias de outra, mas o inverso não é verdadeiro. Por exemplo, um "Cliente" pode ter vários "Pedidos", mas cada "Pedido" pertence a um único "Cliente".

- **Muitos-para-Muitos (N:M):** Instâncias de ambas as entidades podem se relacionar com múltiplas instâncias da outra. Por exemplo, um "Aluno" pode se matricular em vários "Cursos", e um "Curso" pode ter vários "Alunos".

Através da combinação desses elementos, os Diagramas ER fornecem um mapa claro e conciso da estrutura lógica do banco de dados, facilitando o entendimento e a comunicação entre desenvolvedores, analistas e clientes.

Normalização: Organizando para a Consistência

A normalização é um processo passo a passo para organizar as tabelas de um banco de dados relacional com o objetivo de minimizar a redundância de dados e, consequentemente, evitar anomalias de inserção, atualização e exclusão. As três primeiras formas normais (1FN, 2FN e 3FN) são as mais utilizadas e, em geral, suficientes para a maioria das aplicações.

Primeira Forma Normal (1FN)

Objetivo: Eliminar grupos de repetição e garantir que os atributos sejam atômicos.

Uma tabela está na 1FN se:

- Cada célula da tabela contiver um único valor (atomicidade).
- Não houver colunas ou grupos de colunas que se repitam.

Exemplo: Uma tabela de pedidos que armazena múltiplos produtos em uma única coluna viola a 1FN. Para adequá-la, cria-se uma tabela separada para os itens do pedido, onde cada linha representa um único produto daquele pedido.

Segunda Forma Normal (2FN)

Objetivo: Eliminar dependências parciais de chaves primárias compostas.

Uma tabela está na 2FN se:

- Já estiver na 1FN.
- Todos os seus atributos não-chave dependerem funcionalmente da chave primária em sua totalidade, e não apenas de parte dela (no caso de chaves primárias compostas).

Exemplo: Considere uma tabela com a chave primária composta (ID_Pedido, ID_Produto) e um campo "Descricao_Produto". A descrição do produto depende apenas do "ID_Produto" e não da combinação com o "ID_Pedido". Para estar na 2FN, a "Descricao_Produto" deve ser movida para uma tabela de "Produtos", onde a chave primária é "ID_Produto".

Terceira Forma Normal (3FN)

Objetivo: Eliminar dependências transitivas.

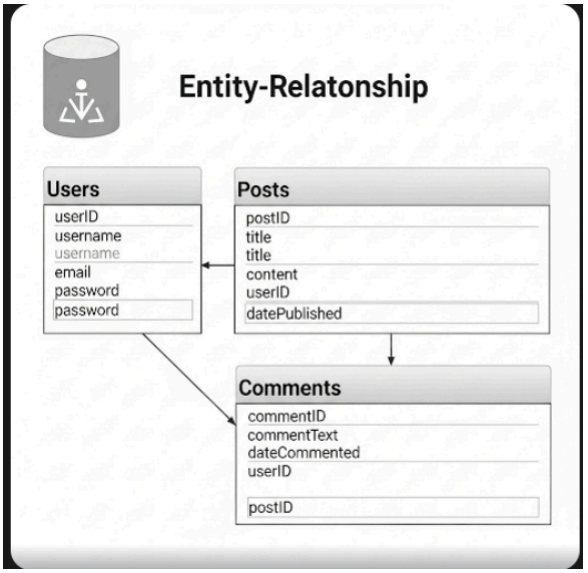
Uma tabela está na 3FN se:

- Já estiver na 2FN.
- Nenhum atributo não-chave depender de outro atributo não-chave.

Exemplo: Em uma tabela de "Funcionários", se houver os campos "ID_Departamento" e "Nome_Departamento", e "Nome_Departamento" depender de "ID_Departamento" (que não é a chave primária da tabela de funcionários), existe uma dependência transitiva. Para resolver isso, cria-se uma tabela separada de "Departamentos" com "ID_Departamento" como chave primária e o campo "Nome_Departamento".

A aplicação das formas normais resulta em um banco de dados mais organizado, com menor redundância, maior integridade e mais fácil de manter e evoluir. Em conjunto, os Diagramas ER e a normalização são ferramentas indispensáveis para o projeto de sistemas de banco de dados eficientes, consistentes e escaláveis.

semana 15
SGBD relacional
Aula 2: Modelagem de dados relacional



Explicando o Diagrama:

Neste exemplo, podemos identificar os seguintes componentes:

- **Entidades:**
    - **Users (Utilizadores):** Representa as pessoas que podem escrever posts e comentários.
    - **Posts (Publicações):** Representa os artigos ou publicações no blog.

    - **Comments (Comentários):** Representa os comentários feitos nas publicações.
- **Atributos:**
    - A entidade **Users** tem atributos como `userID` (a chave primária que identifica unicamente cada utilizador), `username`, `email`, e `password`.
    - A entidade **Posts** tem atributos como `postID` (chave primária), `title`, `content`, `datePublished`, e `userID` (uma chave estrangeira que a liga à entidade Users, indicando quem escreveu o post).
    - A entidade **Comments** tem atributos como `commentID` (chave primária), `commentText`, `dateCommented`, `userID` (chave estrangeira para saber quem fez o comentário), e `postID` (chave estrangeira para saber a que publicação o comentário pertence).
- **Relacionamentos:**
    - **Um Utilizador "escreve" um ou muitos Posts:** Este é um relacionamento de um-para-muitos (1:N), pois um utilizador pode escrever vários posts, mas cada post é escrito por apenas um utilizador.
    - **Um Utilizador "faz" um ou muitos Comentários:** Outro relacionamento de um-para-muitos (1:N).
    - **Um Post "tem" um ou muitos Comentários:** Também um relacionamento de um-para-muitos (1:N), pois uma publicação pode ter vários comentários, mas cada comentário pertence a uma única publicação.

semana 15
SGBD relacional
Aula 4: SQL básico para SGBD relacional

SQL Básico para Operações CRUD

SQL (Structured Query Language) é a linguagem padrão para interagir com bancos de dados relacionais. As operações CRUD formam a base da manipulação de dados, permitindo criar, ler, atualizar e apagar registros.

1. CREATE (Criar) - O Comando `INSERT`

A operação de **Criação** é usada para adicionar novos registros (linhas) a uma tabela. O comando SQL para isso é o `INSERT INTO`.

Sintaxe:

```SQL
INSERT INTO nome_da_tabela (coluna1, coluna2, coluna3)
```

```
VALUES (valor1, valor2, valor3);
```

Exemplo Prático:

Vamos supor que temos uma tabela chamada Clientes com as colunas ID, Nome e Email. Para adicionar um novo cliente:

```
SQL

INSERT INTO Clientes (ID, Nome, Email)
VALUES (1, 'João Silva', 'joao.silva@email.com');
```

---

2. READ (Ler) - O Comando `SELECT`

A operação de **Leitura** é a mais comum e serve para consultar e recuperar dados de uma ou mais tabelas. O comando principal para isso é o `SELECT`. Ele é extremamente versátil.

Sintaxe Básica:

- Para selecionar colunas específicas:
- SQL

```
SELECT coluna1, coluna2 FROM nome_da_tabela;
```

-
-
- Para selecionar todas as colunas:
- SQL

```
SELECT * FROM nome_da_tabela;
```

-
-

Filtrando os Resultados com WHERE:

A cláusula WHERE é usada para extrair apenas os registros que satisfazem uma condição específica.

Exemplo Prático:

- Para ler o nome e o email de todos os clientes:
- SQL

```
SELECT Nome, Email FROM Clientes;
```

-
-
- Para ler todas as informações do cliente com `ID = 1`:
- SQL

```
SELECT * FROM Clientes WHERE ID = 1;
```

-
-

---

3. UPDATE (Atualizar) - O Comando `UPDATE`

A operação de **Atualização** é usada para modificar registros existentes em uma tabela. O comando para isso é o `UPDATE`.

Sintaxe:

```
SQL

UPDATE nome_da_tabela
SET coluna1 = valor1, coluna2 = valor2
WHERE condicao;
```

Atenção: A cláusula `WHERE` é crucial no comando `UPDATE`. Se você não a especificar, **todos os registros da tabela serão atualizados**, o que raramente é o desejado e pode levar à perda de dados.

Exemplo Prático:

- Para atualizar o email do cliente 'João Silva':
- SQL

```
UPDATE Clientes
SET Email = 'joao.novo@email.com'
WHERE Nome = 'João Silva';
```

-
-

---

4. DELETE (Apagar) - O Comando `DELETE`

A operação de **Exclusão** é usada para remover registros existentes de uma tabela. O comando para isso é o `DELETE FROM`.

Sintaxe:

```
SQL

DELETE FROM nome_da_tabela WHERE condicao;
```

Atenção: Assim como no UPDATE, a cláusula WHERE é fundamental. Se omitida, **todos os registros da tabela serão apagados**.

Exemplo Prático:

- Para apagar o cliente com ID = 1 da tabela Clientes:
- SQL

```
DELETE FROM Clientes WHERE ID = 1;
```

-
-

Esses quatro comandos (INSERT, SELECT, UPDATE, DELETE) são os pilares do SQL e permitem realizar a grande maioria das tarefas de manipulação de dados necessárias no dia a dia.