UniSENAI

Módulo 4: Relacionamentos Entre Tabelas

Módulo 4: Relacionamentos Entre Tabelas

Uma das principais forças dos bancos de dados relacionais é a capacidade de estabelecer relacionamentos entre tabelas. Isso permite organizar e conectar dados de forma eficiente, evitando redundâncias e assegurando a integridade das informações.

---

1. Conceitos de Relacionamentos

Relacionamentos entre tabelas são baseados no uso de chaves primárias (PRIMARY KEY) e chaves estrangeiras (FOREIGN KEY). Esses elementos garantem que os dados sejam estruturados de maneira lógica e consistente.

Chave Primária (PRIMARY KEY)

A chave primária é um identificador único para cada registro de uma tabela. É usada para assegurar que cada linha seja distinta, facilitando a identificação e recuperação de informações.

Características da PRIMARY KEY

- Deve ser única para cada registro.
- Não pode conter valores nulos.
- Pode ser composta por mais de uma coluna (chave composta).

Exemplo Prático

Imagine a tabela `clientes`:

id_cliente (PRIMARY KEY)	Nome	E-mail
1	João Silva	joao.silva@email.com
2	Maria Silva	maria@email.com

No exemplo, a coluna `id_cliente` é definida como chave primária:

```
CREATE TABLE clientes (
    id_cliente INT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100)
);
```

Aplicabilidade:
A chave primária é usada para identificar unicamente cada cliente no sistema, evitando duplicatas.

Chave Estrangeira (FOREIGN KEY)

A chave estrangeira é uma coluna (ou conjunto de colunas) em uma tabela que referencia a chave primária de outra tabela. Ela estabelece um vínculo entre as tabelas, criando relacionamentos.

Características da FOREIGN KEY

- Garante que os valores inseridos correspondem a uma chave primária existente.
- Restringe ações que possam comprometer a integridade dos dados (como exclusão de registros referenciados).

Exemplo Prático

Agora, considere uma tabela `pedidos` que armazena pedidos realizados por clientes:

id_pedido	id_cliente	valor_total
101	1	150.00
102	2	200.00

A `id_cliente` na tabela `pedidos` referencia a chave primária da tabela `clientes`. Sua definição seria:

```
CREATE TABLE pedidos (
    id_pedido INT PRIMARY KEY,
    id_cliente INT,
    valor_total DECIMAL(10, 2),
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
);
```

Aplicabilidade:
Esse relacionamento permite associar cada pedido a um cliente específico.

Integridade Referencial

A **integridade referencial** assegura que os relacionamentos entre tabelas permaneçam válidos. Isso significa que toda chave estrangeira deve sempre referenciar um registro existente na tabela relacionada.

Regras de Integridade Referencial

1. **Inserção**: Não é permitido inserir um valor em uma chave estrangeira que não exista na tabela referenciada.
2. **Atualização**: A alteração de uma chave primária deve ser refletida em todas as tabelas relacionadas, ou a operação será rejeitada.
3. **Exclusão**: Não é permitido excluir um registro que esteja sendo referenciado, a menos que ações específicas sejam definidas (`CASCADE`, `SET NULL`).

Exemplo Prático

- **Tentativa de inserção inválida**: Inserir um pedido com um `id_cliente` que não existe.

```
INSERT INTO pedidos (id_pedido, id_cliente, valor_total)
VALUES (103, 3, 100.00); -- Erro, pois id_cliente = 3 não existe na tabela clientes.
```

- Excluir um cliente referenciado:

```
DELETE FROM clientes
WHERE id_cliente = 1;
```

Este comando falharia, pois o cliente com `id_cliente = 1` está sendo referenciado na tabela `pedidos`. Para evitar o erro, pode-se configurar a chave estrangeira com a ação `ON DELETE CASCADE`, permitindo a exclusão automática de todos os pedidos relacionados:

```
CREATE TABLE pedidos (
    id_pedido INT PRIMARY KEY,
    id_cliente INT,
    valor_total DECIMAL(10, 2),
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente) ON DELETE CASCADE
);
```

Aplicabilidade:

A integridade referencial é essencial para garantir que os dados relacionados permaneçam consistentes, evitando problemas como pedidos associados a clientes inexistentes.

Tipos de Relacionamentos Entre Tabelas

Os relacionamentos entre tabelas em bancos de dados relacionais são fundamentais para organizar e conectar informações. Existem três tipos principais de relacionamentos: **Um para Um (1:1)**, **Um para Muitos (1:N)** e **Muitos para Muitos (N:M)**. Cada tipo é aplicável a diferentes situações, e sua implementação depende de como os dados precisam ser estruturados.

---

1. Um para Um (1:1)

Um relacionamento **Um para Um** ocorre quando um registro em uma tabela está associado, no máximo, a um único registro em outra tabela. Esse tipo de relacionamento é usado quando você deseja dividir informações detalhadas em tabelas separadas para organizar melhor os dados ou melhorar a segurança.

Exemplo Prático

Imagine um sistema de clientes que armazena informações sensíveis separadas da tabela principal de clientes.

- **Tabela `clientes`**:

id_cliente (PRIMARY KEY)	Nome	E-mail
1	João Silva	joao.silva@email.com
2	Maria Silva	maria@email.com

- Tabela `detalhes_cliente`:

id_cliente	cpf	data_nascimento
1	12345678901	1980-05-15
2	98765432109	1990-08-22

Definição do Relacionamento:
Cada cliente na tabela `clientes` tem **exatamente um registro correspondente** na tabela `detalhes_cliente`.

```sql
CREATE TABLE clientes (
    id_cliente INT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100)
);

CREATE TABLE detalhes_cliente (
    id_cliente INT PRIMARY KEY,
    cpf CHAR(11),
    data_nascimento DATE,
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
);
```

Aplicabilidade:

Esse relacionamento pode ser usado em situações como separar informações confidenciais (como CPF) de informações gerais para melhorar a segurança.

2. Um para Muitos (1:N)

Um relacionamento **Um para Muitos** ocorre quando um registro em uma tabela está relacionado a **vários registros** em outra tabela. Esse é o tipo mais comum de relacionamento em bancos de dados.

Exemplo Prático

Um cliente pode realizar vários pedidos, mas cada pedido pertence a um único cliente.

- **Tabela `clientes`**:

id_cliente (PRIMARY KEY)	Nome	E-mail
1	João Silva	joao.silva@email.com
2	Maria Silva	maria@email.com

Tabela `pedidos`:

id_pedido	id_cliente	valor_total
101	1	150.00
102	1	200.00
103	2	300.00

Definição do Relacionamento:

Um cliente na tabela `clientes` pode estar associado a **vários pedidos** na tabela `pedidos`.

```
CREATE TABLE pedidos (
    id_pedido INT PRIMARY KEY,
    id_cliente INT,
    valor_total DECIMAL(10, 2),
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
);
```

Aplicabilidade:

Esse relacionamento é útil em sistemas de vendas, onde é necessário relacionar clientes com múltiplos pedidos ou transações.

3. Muitos para Muitos (N:M)

Um relacionamento **Muitos para Muitos** ocorre quando vários registros em uma tabela estão relacionados a **vários registros** em outra tabela. Esse tipo de relacionamento é implementado usando uma **tabela intermediária** (ou tabela de junção) para associar os registros.

Exemplo Prático

Em um sistema de vendas, um pedido pode conter vários produtos, e cada produto pode estar em vários pedidos.

- **Tabela `produtos`**:

id_produto	nome	preco
1	Notebook	3000.00
2	Smartphone	2000.00
3	Teclado	150,00

- **Tabela `pedidos`**:

id_pedido	data
1	2024-12-30
2	2024-12-31

- **Tabela `pedido_produto`** (tabela intermediária):

id_pedido	id_produto	quantidade
101	1	1
101	3	2
102	3	1
102	3	1

Definição do Relacionamento:

A tabela intermediária `pedido_produto` associa cada pedido a vários produtos e cada produto a vários pedidos.

```sql
CREATE TABLE produtos (
    id_produto INT PRIMARY KEY,
    nome VARCHAR(100),
    preco DECIMAL(10, 2)
);

CREATE TABLE pedidos (
    id_pedido INT PRIMARY KEY,
    data DATE
);

CREATE TABLE pedido_produto (
    id_pedido INT,
    id_produto INT,
    quantidade INT,
    PRIMARY KEY (id_pedido, id_produto),
    FOREIGN KEY (id_pedido) REFERENCES pedidos(id_pedido),
    FOREIGN KEY (id_produto) REFERENCES produtos(id_produto)
);
```

Aplicabilidade:
Esse relacionamento é útil para sistemas de gerenciamento de inventário ou catálogos, onde produtos podem ser relacionados a múltiplos pedidos.

Resumo dos Relacionamentos

Tipo	Definição	Exemplo
Um para Um (1:1)	Um registro em uma tabela relacionado a um único registro em outra.	**`clientes` e `detalhes_cliente`.**
Um para Muitos (1:N)	Um registro em uma tabela relacionado a vários registros em outra.	`clientes` e seus `pedidos`.
Muitos para Muitos Um (N:M)	Vários re gistros em uma tabela relacionados a vários registros em outra.	`pedidos` e `produtos` usando tabela intermediária.

3. Criação de Relacionamentos no MySQL

A criação de relacionamentos em bancos de dados relacionais no MySQL envolve a definição de **chaves primárias** e **chaves estrangeiras** para estabelecer vínculos entre tabelas. Esses relacionamentos são complementados por ações automáticas, como `ON DELETE CASCADE` e `ON UPDATE CASCADE`, que ajudam a manter a consistência e integridade referencial.

Definição de Chaves Primárias (`PRIMARY KEY`)

A **chave primária** é usada para identificar unicamente cada registro em uma tabela. Ao criar tabelas no MySQL, você define a chave primária usando a cláusula `PRIMARY KEY`.

Exemplo Prático

Criando uma tabela de clientes onde `id_cliente` é a chave primária:

```
CREATE TABLE clientes (
    id_cliente INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100)
);
```

- **`AUTO_INCREMENT`**: Gera automaticamente valores únicos para `id_cliente`.
- **`PRIMARY KEY`**: Define que a coluna `id_cliente` será a chave primária da tabela.

Definição de Chaves Estrangeiras (`FOREIGN KEY`)

A **chave estrangeira** cria um vínculo entre uma coluna em uma tabela e a chave primária de outra tabela. Ela assegura que os valores inseridos correspondam a registros existentes.

Exemplo Prático

Criando uma tabela de pedidos onde `id_cliente` referencia a tabela `clientes`:

```
CREATE TABLE pedidos (
    id_pedido INT AUTO_INCREMENT PRIMARY KEY,
    id_cliente INT,
    valor_total DECIMAL(10, 2),
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
);
```

- **`FOREIGN KEY`**: Define `id_cliente` na tabela `pedidos` como chave estrangeira.
- **`REFERENCES`**: Especifica a tabela (`clientes`) e a coluna (`id_cliente`) referenciada.

Uso de `ON DELETE CASCADE` e `ON UPDATE CASCADE`

Esses modificadores definem como o MySQL deve reagir quando os registros vinculados na tabela de referência (chave primária) são excluídos ou atualizados.

- **`ON DELETE CASCADE`**: Exclui automaticamente os registros relacionados quando o registro referenciado for excluído.

- **`ON UPDATE CASCADE`**: Atualiza automaticamente os registros relacionados quando o valor da chave primária referenciada for alterado.

Exemplo Prático

Definindo ações para exclusão e atualização em cascata:

```
CREATE TABLE pedidos (
    id_pedido INT AUTO_INCREMENT PRIMARY KEY,
    id_cliente INT,
    valor_total DECIMAL(10, 2),
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

- **Cenário**: Se um cliente for excluído da tabela `clientes`, todos os pedidos associados na tabela `pedidos` também serão removidos.
- **Cenário**: Se o `id_cliente` for alterado, a mudança será refletida automaticamente na tabela `pedidos`.

4. Tabelas de Junção para Relacionamentos N:N

Em um relacionamento **Muitos para Muitos (N:N)**, é necessário criar uma **tabela intermediária** para associar registros de ambas as tabelas. Essa tabela de junção contém chaves estrangeiras que referenciam as tabelas principais.

Criação de Tabelas Intermediárias

Imagine um sistema de vendas onde um pedido pode conter vários produtos e um produto pode estar em vários pedidos.

Tabelas Principais

- **Tabela `produtos`**:
-

id_produto	nome	preco
1	Notebook	3000.00

2	Smartphone	2000.00

```sql
CREATE TABLE produtos (
    id_produto INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100),
    preco DECIMAL(10, 2)
);
```

Tabela `pedidos`:

id_pedido	data
101	2024-12-30
102	2024-12-31

```sql
CREATE TABLE pedidos (
    id_pedido INT AUTO_INCREMENT PRIMARY KEY,
    data DATE
);
```

Tabela Intermediária

- **Tabela `pedido_produto`:**

id_pedido	id_produto	quantidade
101	1	1
101	2	2
102	1	3

```sql
CREATE TABLE pedido_produto (
  id_pedido INT,
  id_produto INT,
  quantidade INT,
  PRIMARY KEY (id_pedido, id_produto),
  FOREIGN KEY (id_pedido) REFERENCES pedidos(id_pedido) ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (id_produto) REFERENCES produtos(id_produto) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Criação da Tabela de Junção:

- `PRIMARY KEY (id_pedido, id_produto)`: Define uma chave composta para evitar duplicatas no relacionamento.
- `FOREIGN KEY`: Estabelece o relacionamento com as tabelas `pedidos` e `produtos`.

Utilização de Tabelas Intermediárias

Inserindo Dados

Adicionar registros à tabela de junção para relacionar pedidos e produtos:

```
INSERT INTO pedido_produto (id_pedido, id_produto, quantidade)
VALUES
    (101, 1, 1),
    (101, 2, 2),
    (102, 1, 3);
```

Consultando Dados

Listar todos os produtos de um pedido específico:

```
SELECT pedidos.id_pedido, produtos.nome, pedido_produto.quantidade
FROM pedido_produto
JOIN pedidos ON pedido_produto.id_pedido = pedidos.id_pedido
JOIN produtos ON pedido_produto.id_produto = produtos.id_produto
WHERE pedidos.id_pedido = 101;
```

Resultado:

id_pedido	nome	quantidade
101	Notebook	1
101	Smartphone	2

Aplicabilidade Prática

1. **Sistemas de E-commerce**: Gerenciar pedidos e produtos, relacionando itens em carrinhos de compras.
2. **Sistemas Acadêmicos**: Relacionar alunos e cursos em matrículas.
3. **Gerenciamento de Projetos**: Associar projetos a múltiplos membros de equipe.

Com a definição de **chaves primárias**, **chaves estrangeiras**, e o uso de **tabelas de junção**, é possível criar sistemas robustos e escaláveis que aproveitam a flexibilidade dos bancos de dados relacionais.

Atividades de Fixação

Questões Práticas

1. **Chave Primária e Estrangeira**:
   Crie duas tabelas chamadas `clientes` e `pedidos`. Configure:
   - `id_cliente` como chave primária na tabela `clientes`.
   - `id_cliente` como chave estrangeira na tabela `pedidos`, referenciando `clientes`.
     Qual a importância de definir essas chaves corretamente?
2. **Integridade Referencial**:
   Configure a tabela `pedidos` para que a exclusão de um cliente também exclua automaticamente os pedidos associados (ON DELETE CASCADE). Explique como isso ajuda a manter a integridade dos dados.
3. **Tipos de Relacionamentos**:
   Desenvolva um relacionamento 1:N entre `clientes` e `pedidos`. Insira 10 registros em cada tabela e demonstre como listar todos os pedidos de um cliente específico usando uma consulta SQL.
4. **Relacionamento N:N**:
   Crie as tabelas `produtos`, `pedidos` e a tabela intermediária `pedido_produto`. Insira registros nas tabelas e demonstre como listar todos os produtos de um pedido.
5. **Consulta com JOIN**:
   Realize uma consulta SQL para retornar os nomes dos clientes, os produtos comprados e a quantidade de cada produto em um pedido.
6. **Exclusão em Cascata**:
   Demonstre o comportamento de uma exclusão em cascata na tabela `clientes`, excluindo um cliente e verificando se os pedidos relacionados foram removidos.
7. **Atualização em Cascata**:
   Configure a tabela `pedidos` para que a alteração do `id_cliente` na tabela `clientes` atualize automaticamente os registros relacionados. Teste com um exemplo prático.
8. **Cardinalidades**:
   Explique e configure exemplos práticos para os tipos de cardinalidades 1:1, 1:N e N:M, criando tabelas e inserindo registros.
9. **Tabelas Intermediárias**:
   Crie uma consulta para listar todos os pedidos contendo um produto específico usando a tabela intermediária `pedido_produto`.
10. **Boas Práticas em Relacionamentos**:
    Analise um banco de dados existente (pode ser fictício) e identifique como os

relacionamentos foram estruturados. Proponha melhorias para garantir integridade e eficiência.

Questões Teóricas

1. **Definição de Relacionamentos**:
   Explique o conceito de relacionamento entre tabelas e como as chaves primárias e estrangeiras são usadas para implementá-los.
2. **Chave Primária**:
   Quais são as principais características de uma chave primária? Por que ela é essencial em um banco de dados relacional?
3. **Chave Estrangeira**:
   O que é uma chave estrangeira? Explique sua importância na criação de relacionamentos entre tabelas.
4. **Integridade Referencial**:
   O que é integridade referencial? Cite exemplos práticos de como ela é aplicada em um banco de dados.
5. **Tipos de Relacionamentos**:
   Diferencie os relacionamentos 1:1, 1:N e N:M, dando exemplos práticos para cada tipo.
6. **Exclusão e Atualização em Cascata**:
   Explique as diferenças entre ON DELETE CASCADE e ON UPDATE CASCADE. Quando é apropriado usar cada um?
7. **Tabelas Intermediárias**:
   Qual é a função de uma tabela intermediária em um relacionamento N:M? Explique sua estrutura e aplicabilidade.
8. **Vantagens dos Relacionamentos**:
   Quais são os benefícios de organizar dados em tabelas relacionadas em vez de armazenar tudo em uma única tabela?
9. **Problemas sem Relacionamentos**:
   O que pode acontecer em um banco de dados que não utiliza relacionamentos corretamente? Dê exemplos de inconsistências que podem surgir.
10. **Aplicabilidade Prática**:
    Em quais situações práticas você utilizaria relacionamentos 1:N e N:M? Explique com base em sistemas reais, como e-commerce ou gestão acadêmica.