Relacionamentos

Em bancos de dados, um relacionamento é uma conexão lógica estabelecida entre duas ou mais tabelas com base em chaves de dados relacionadas. Esses relacionamentos são fundamentais para organizar e estruturar os dados de forma eficiente, garantindo integridade, consistência e facilitando consultas e análises.

Existem diferentes tipos de relacionamentos em bancos de dados relacionais, sendo os mais comuns:

1. **Relacionamento Um para Um (1:1)**: Nesse tipo de relacionamento, cada registro em uma tabela está relacionado a apenas um registro em outra tabela e vice-versa.

2. **Relacionamento Um para Muitos (1:N)**: Aqui, um registro em uma tabela pode estar associado a vários registros em outra tabela, mas um registro nessa segunda tabela só pode estar relacionado a um único registro na primeira tabela.

3. **Relacionamento Muitos para Muitos (N:M)**: Nesse caso, múltiplos registros em uma tabela podem estar relacionados a múltiplos registros em outra tabela. Isso é normalmente implementado por meio de uma tabela intermediária, também conhecida como tabela de junção ou tabela de associação.

Para estabelecer e manter esses relacionamentos, são utilizadas chaves estrangeiras (foreign keys), que são campos ou conjuntos de campos em uma tabela que referenciam a chave primária (primary key) em outra tabela. As chaves primárias garantem a unicidade dos registros em uma tabela, enquanto as chaves estrangeiras mantêm a integridade referencial, garantindo que os registros relacionados estejam sempre consistentes entre as tabelas.

> Uma primary key geralmente é a chave principal de uma tabela. O padrão utilizado no mercado é utilizar a diretiva AUTO_INCREMENT (no caso do MySQL, MSSQL, Oracle SQL) ou SERIAL (no caso Postgres). Essa diretiva entrega na mão do Sistema Gerenciador de Banco de Dados (SGBD) o total controle sobre a criação de valores para o campo chave (primary key) bem como encontrar algum registro pelo índice ou remover/editar também (UPDATE ou DELETE).

> Uma foreign key é uma chave que recebe um valor que apontará para a chave primária de uma outra tabela. Por exemplo, existem duas tabelas em um database: *clientes* e *compras*. Um cliente pode fazer uma compra. Portanto, uma compra deve (para fins de **normalização**) receber o id do cliente que comprou. Ou seja, a tabela *compras* terá a coluna **id_cliente** (Foreign Key). A tabela *compras* terá a coluna **id_cliente** (Primary Key). Dessa forma eu estabeleço um relacionamento entre uma compra e um cliente.

Tomando ainda como caso de estudo o exemplo dado, podemos estabelecer a seguinte relação:
1 compra pertence a 1 cliente, enquanto 1 cliente pode ter diversas compras.

Portanto, o relacionamento entre *clientes x compras* é um 1:N (um pra muitos).

Operação de Junção em Álgebra Relacional

A operação de junção é uma das operações fundamentais na álgebra relacional, que é uma linguagem formal para manipulação de dados em bancos de dados relacionais. A junção é usada para combinar registros de duas ou mais relações (tabelas) com base em uma condição de correspondência especificada.

Existem diferentes tipos de junção, sendo os mais comuns:

1. **Junção Interna (INNER JOIN)**: Retorna apenas os registros que possuem correspondência nas duas relações, com base na condição de junção especificada. Registros que não têm correspondência são excluídos do resultado.

2. **Junção Externa (OUTER JOIN)**: Retorna todos os registros de uma ou ambas as relações, dependendo do tipo de junção externa (esquerda, direita ou completa) e mantém as correspondências com base na condição de junção especificada.

3. **Junção Cruzada (CROSS JOIN)**: Retorna o produto cartesiano das duas relações, ou seja, combina cada registro de uma relação com cada registro da outra relação, sem considerar nenhuma condição de junção. É útil quando se deseja combinar todas as linhas de uma tabela com todas as linhas de outra tabela.

4. **Junção Natural**: É uma junção que ocorre automaticamente com base em colunas com o mesmo nome em ambas as tabelas. Não é necessário especificar uma

condição de junção, pois o banco de dados realiza a correspondência automaticamente com base nas colunas com nomes iguais.

A sintaxe para realizar uma junção em álgebra relacional pode variar dependendo do sistema de gerenciamento de banco de dados que está sendo utilizado, mas geralmente segue a estrutura básica de:

```
R ⋈_condição S
```

Onde:

- R e S são as relações (tabelas) que serão combinadas.
- ⋈ é o operador de junção.
- _condição é a condição de junção que especifica como os registros devem ser combinados.

Relacionamentos com JOINS

Uma cláusula *join* correpsonde a uma operação de junção em álgebra relacional. Ela combina colunas de duas ou mais tabelas, criando um conjunto que pode ser salvo como uma tabela ou usado da forma que está. Existem 5 tipos de JOINS no SQL:

1. **Inner Join**:
   Seleciona as colunas dos registros que possuem valores correspondentes em ambas tabelas.

> SELECT **<lista de colunas ou um * para selecionar todas as colunas>** FROM **<tabela_a>** INNER JOIN **<tabela_b>** ON **<condição de junção, ex. tabela_a.primaryKey = tabela_b.foreignKey>**

2. **Left Join**:
   Retorna todos os registros da tabela à esquerda (tabela_a) e os registros correspondentes da tabela à direita (tabela_b). O resultado é 0 registros do lado direito, se não houver correspondência.

> SELECT **<lista de colunas ou um * para selecionar todas as colunas>** FROM **<tabela_a>** LEFT JOIN **<tabela_b>** ON **<condição de junção, ex. tabela_a.primaryKey = tabela_b.foreignKey>**

3. **Right Join**:
   Retorna todos os registros da tabela à direita (tabela_b) e os registros correspondentes da tabela à esquerda (tabela_a). O resultado é 0 registros do lado esquerdo, se não houver correspondência.

> SELECT **<lista de colunas ou um * para selecionar todas as colunas>** FROM **<tabela_a>** RIGHT JOIN **<tabela_b>** ON **<condição de junção, ex. tabela_a.primaryKey = tabela_b.foreignKey>**

4. **Cross Join**:
   Retorna todos os registros de ambas tabelas combinados.

> SELECT **<lista de colunas ou um * para selecionar todas as colunas>** FROM **<tabela_a>** CROSS JOIN **<tabela_b>**

5. **Full Join**:
   Retorna todos os registros quando há uma correspondência à esquerda (tabela_a) ou à direita (tabela_b).

> SELECT **<lista de colunas ou um * para selecionar todas as colunas>** FROM **<tabela_a>** FULL OUTER JOIN **<tabela_b>** ON **<condição de junção, ex. tabela_a.primaryKey = tabela_b.foreignKey>**

Referências

> **SQL INNER JOIN**
> **SQL LEFT JOIN**
> **SQL RIGHT JOIN**
> **SQL CROSS JOIN**
> **SQL FULL JOIN**
> **SQL CROSS JOIN with examples**
> **What Is FULL JOIN in SQL? An Explanation with 4 Examples**