Resumo do Capítulo: Relacionamentos entre Tabelas

Tipos de Relacionamentos entre Tabelas

Quando uma coluna contém os valores do campo de outra tabela, a isso se chama **chave estrangeira**. Ela é responsável pelo relacionamento dessas tabelas.

Há três tipos de relacionamento:

- um-para-um
- um-para-muitos
- muitos-para-muitos

Em um relacionamento **um-para-um**, cada linha em uma tabela está conectada com apenas uma linha na outra tabela. É como se uma tabela tivesse sido dividida ao meio. Este é um tipo raro de relacionamento e é usado predominantemente por motivos de segurança.

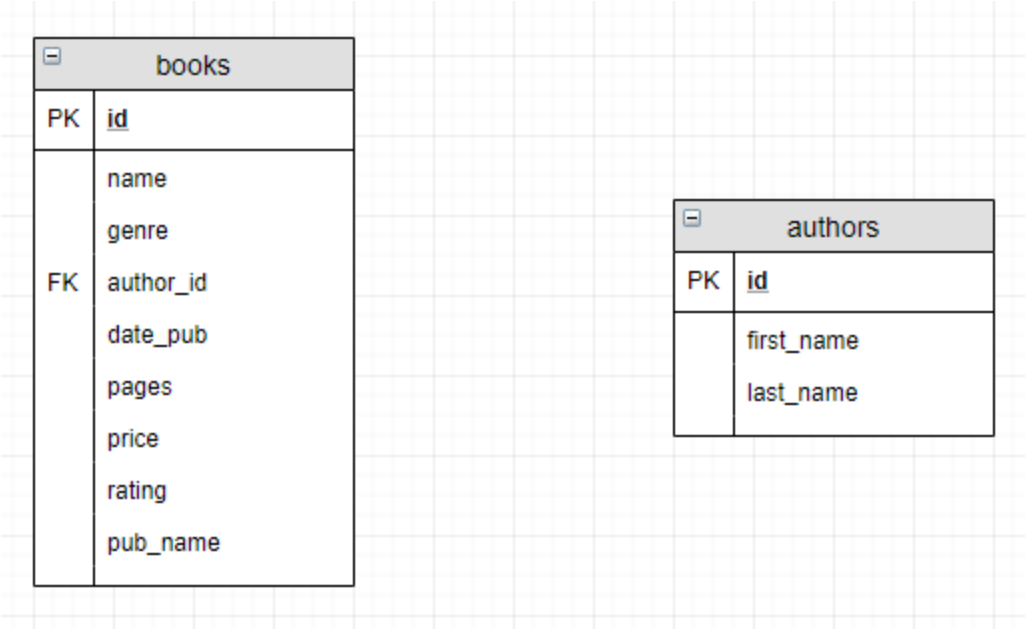
Em um relacionamento **um-para-muitos**, cada linha de uma tabela corresponde a múltiplas linhas na outra tabela.

Em um relacionamento **muitos-para-muitos**, várias linhas de uma tabela correspondem a várias linhas da outra tabela. Esse tipo de relacionamento produz uma **tabela de associação**, que combina as chaves primárias de ambas tabelas.

Diagramas ER

A estrutura de bancos de dados pode ser visualizada com **diagramas ER (entidade-relacionamento)**. Eles mostram as tabelas e seus relacionamentos.

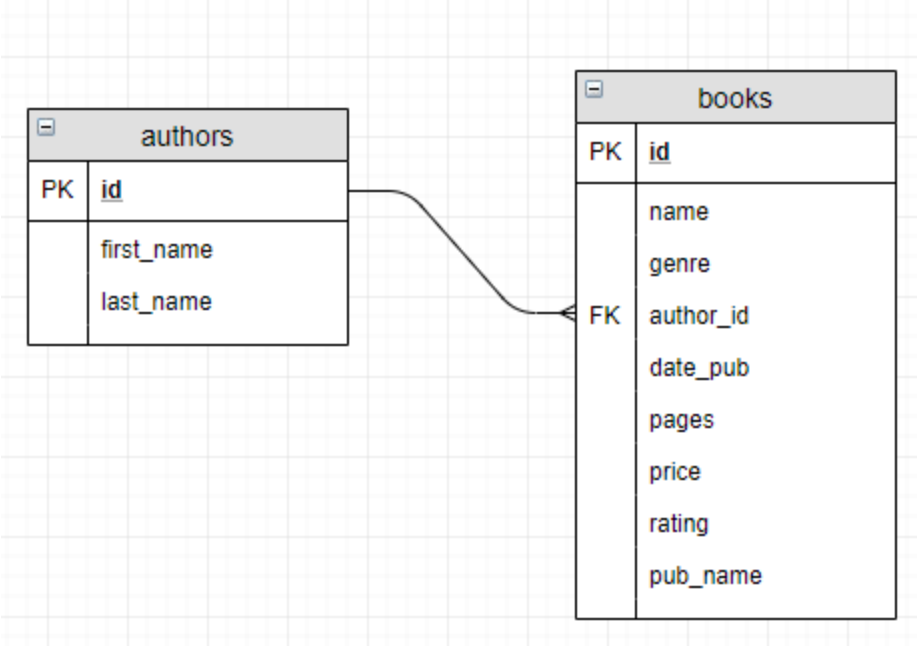
As tabelas são mostradas como retângulos (caixas) com duas partes. O nome da tabela vai na parte superior.



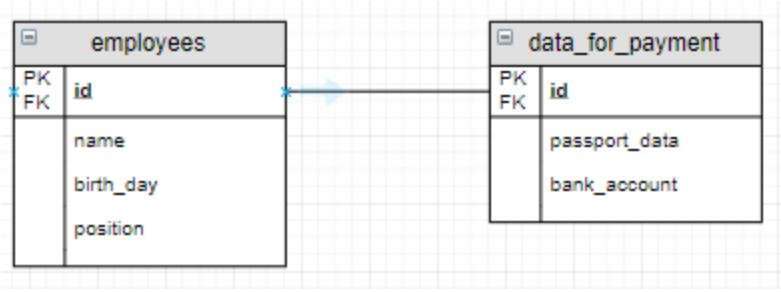
Na parte inferior vemos listas de campos de tabelas com uma indicação dos seus tipos de chaves: primária ou estrangeira. As chaves são geralmente marcadas com **PK** (primária) or **FK** (estrangeira), mas elas também podem ser marcadas com um ícone de chave, um #, ou outro símbolo.

Diagramas ER também mostram relacionamentos. O fim da linha conectando duas tabelas indica se um ou diversos valores de uma tabela correspondem a valores da outra.

Aqui está uma relação de um-para-muitos:



E aqui está uma relação de um-para-um:



Tipos de Usuários de Bancos de Dados

Muitos funcionários usam o banco de dados principal de uma empresa ao mesmo tempo. ECada um deles precisa apenas de certos dados para fazer o seu serviço. Então tudo deve estar organizado de modo que os usuários não interfiram no trabalho dos outros. Por isso é necessário ter a**dministradores do banco de dados**. Eles gerenciam o acesso dos usuários, monitoram a carga de trabalho do sistema, cuidam da segurança e fazem backups.

Bancos de dados são como criaturas vivas, crescendo constantemente. **Arquitetos** e **desenvolvedores** de banco de dados garantem que eles crescem de forma saudável. As decisões desses especialistas determinam a estrutura, integridade e completude do banco de dados, assim como suas possibilidades de escalabilidade (adição de novas tabelas, relações e funções). Os arquitetos e desenvolvedores são responsáveis pela performance do banco de dados.

Os **engenheiros de dados** são responsáveis por adicionar os dados ao banco de dados. Eles também são chamados de **especialistas em ETL**, pois extraem, transformam e carregam dados em bancos de dados.

Analistas, nesse modelo, são usuários típicos. Eles escrevem consultas para bancos de dados e obtém os dados necessários, que eles então analisam e usam para testar hipóteses. Os analistas trabalham mais de perto com os dados do que os outros, e é ele o primeiro a encontrar os campos ou tabelas ausentes. Crie o hábito de comunicar imediatamente essas "descobertas" aos desenvolvedores se quiser que esses erros sejam resolvidos prontamente.

Procurando por Valores Vazios

Em SQL, dizemos que células vazias são **NULL**. O operador **IS NULL** efetua sua busca:

```
SELECT
  *
FROM
  table_name
WHERE
  column_name IS NULL;
```

Não esqueça que o IS é importante! Isso aqui não vai funcionar:

```
SELECT
 *
FROM
  table_name
```

```
WHERE
  column_name = NULL; -- este código não vai compilar!
```

Para excluir linhas com valores NULL da seleção, usamos o operador NOT:

```
SELECT
  *
FROM
  table_name
WHERE
  column_name IS NOT NULL;
```

A construção **CASE** é usada para realizar ações quando certas condições são atendidas. É muito parecido com `if-elif-else` em Python:

```
CASE
    WHEN condition_1 THEN result_1
    WHEN condition_2 THEN result_2
    WHEN condition_3 THEN result_3
    ELSE result_4
END;
```

Uma condição segue o operador **WHEN**. Se a linha de uma tabela satisfaz essa condição, o código retorna o resultado indicado em **THEN**. Caso contrário, a mesma linha é testada com a próxima condição. Se a linha não corresponde a nenhuma das condições determinadas em WHEN, o código retorna o valor indicado após **ELSE**. A construção CASE é então fechada com o operador **END**.

Procurando por Dados na Tabela

O operador **LIKE** procura na tabela por valores que seguem um determinado padrão. Você pode procurar não apenas por uma palavra, mas também por um fragmento dela.

Aqui está a sintaxe da instrução LIKE:

```
column_name LIKE 'expressão regular'
```

Indique a coluna necessária antes de LIKE e depois dela escreva uma expressão regular.

Expressões regulares em SQL são um pouco diferentes daquelas em Python. Por exemplo, o símbolo `_` indica um valor substituto (1 caractere) em uma expressão regular. O símbolo `%` substitui qualquer quantidade de caracteres. Um intervalo ou sequência de caracteres que uma string deve conter são escritos entre colchetes `[]`. Se os caracteres devem ser excluídos, usa-se a construção `[^]`.

Intervalo ou sequência de caracteres

Se precisarmos encontrar um símbolo da expressão regular como uma substring, usamos o operador **ESCAPE**. É passado um símbolo como, por exemplo, um ponto de exclamação. Na expressão regular, o ponto de exclamação significa que o símbolo que o segue não faz parte da expressão, mas sim da substring que há de ser pesquisada. Aqui está o pedaço de código que encontrará todos os finais de substrings com o símbolo `%` (digamos, "100%") em uma tabela:

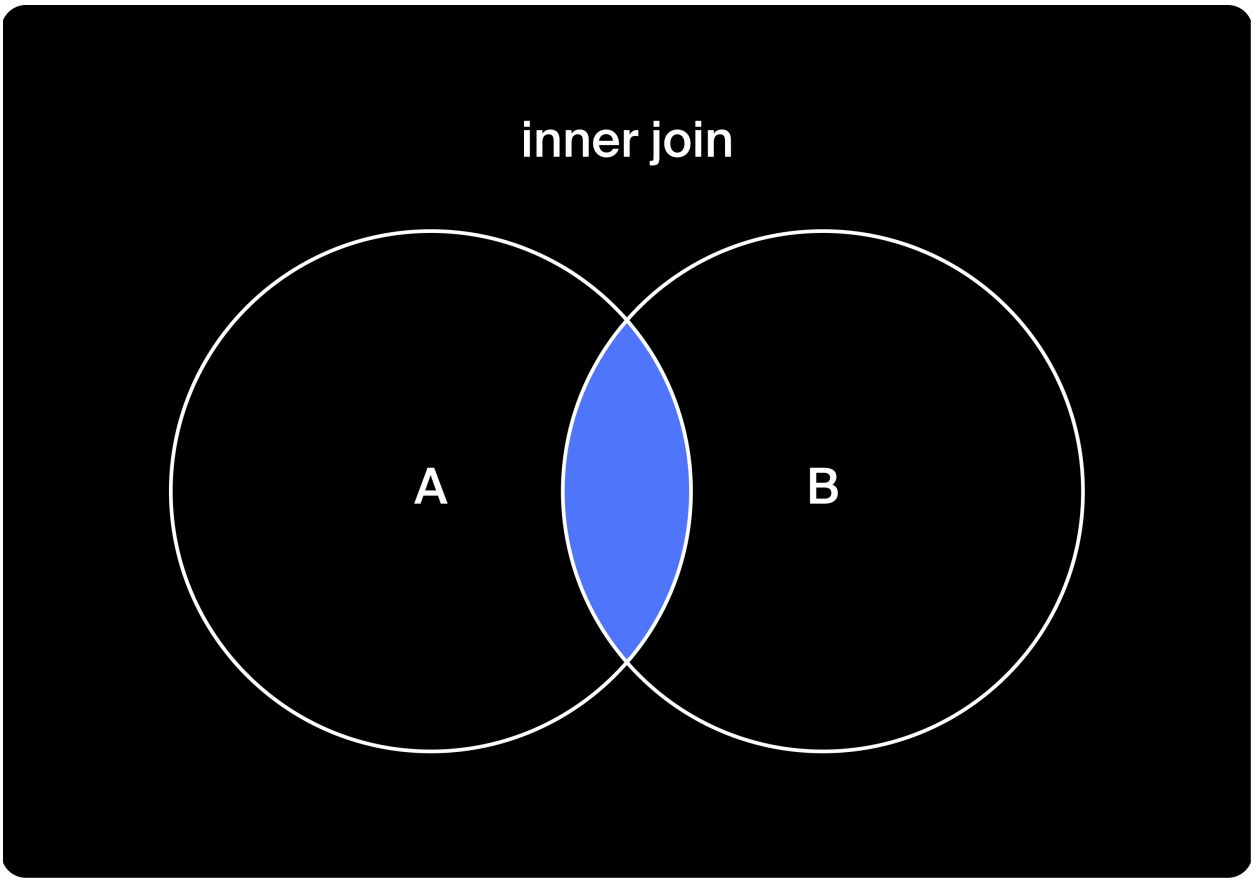
```
column_name LIKE '%!%' ESCAPE '!'
--encontre todas as substrings terminadas com %
```

JOIN. INNER JOIN.

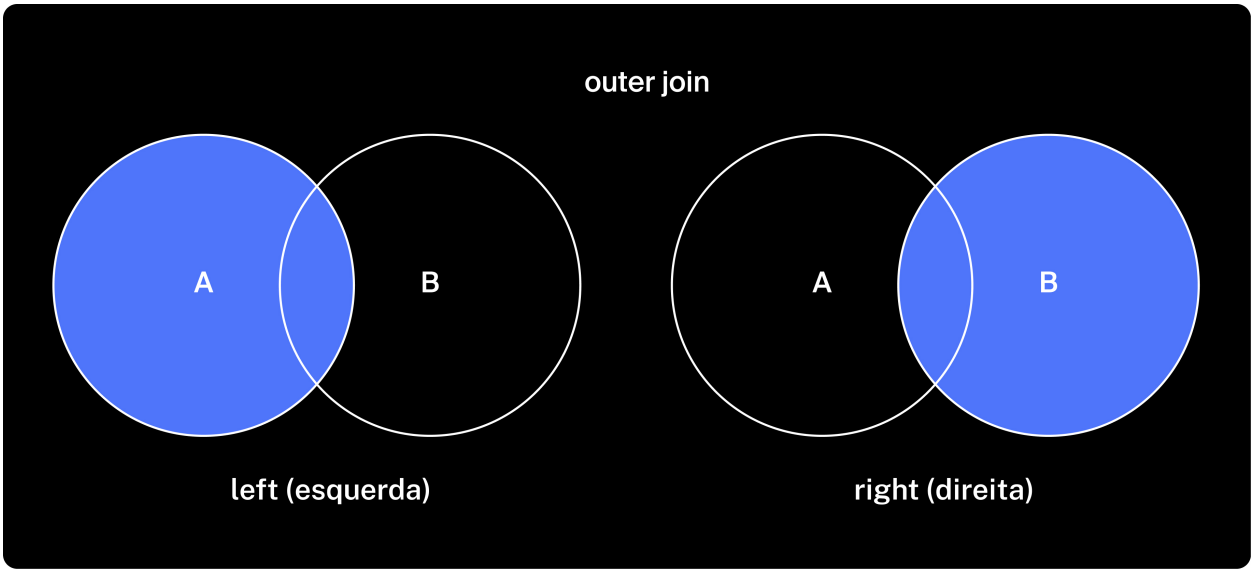
É raro que todos os dados estejam armazenados em uma tabela. Analistas geralmente precisam juntar tabelas; para isso existe o operador **JOIN**.

Há duas maneiras de unir tabelas: a **INNER** e **OUTER** JOIN.

A INNER JOIN retorna apenas aquelas linhas que possuem valores correspondentes nas tabela que estão sendo juntadas (a intersecção das tabelas).



A OUTER JOIN recupera **todos** os dados de uma tabela e adiciona dados da outra quando há linhas correspondentes.



INNER JOIN

INNER JOIN seleciona somente os dados para os quais as condições de junção são atendidas. A ordem na qual as tabelas são juntadas não afeta o resultado final.

Aqui está um exemplo de consulta com INNER JOIN:

```
SELECT --listando somente os campos necessários
  TABLE_1.field_1 AS field_1,
  TABLE_1.field_2 AS field_2,
  ...
  TABLE_2.field_n AS field_n
FROM
  TABLE_1
INNER JOIN TABLE_2 ON TABLE_2.field_1 = TABLE_1.field_2;
```

Vamos observar a sintaxe mais de perto:

- INNER JOIN é o nome do método de junção. Depois vem o nome da tabela a ser juntada à tabela do bloco FROM.
- ON precede a condição de junção: `TABLE_2.field_1 = TABLE_1.field_2`. Isso quer dizer que apenas as linhas da tabela que satisfazem essa condição serão juntadas. Em nosso caso, a condição é que `field_1` da segunda tabela é igual a `field_2` da primeira.

Como campos em tabelas diferentes podem ter os mesmos nomes, nos referimos a eles com o nome da tabela e com o nome do campo. Primeiro o nome da tabela, então o do campo: `TABLE_1.field_1`.

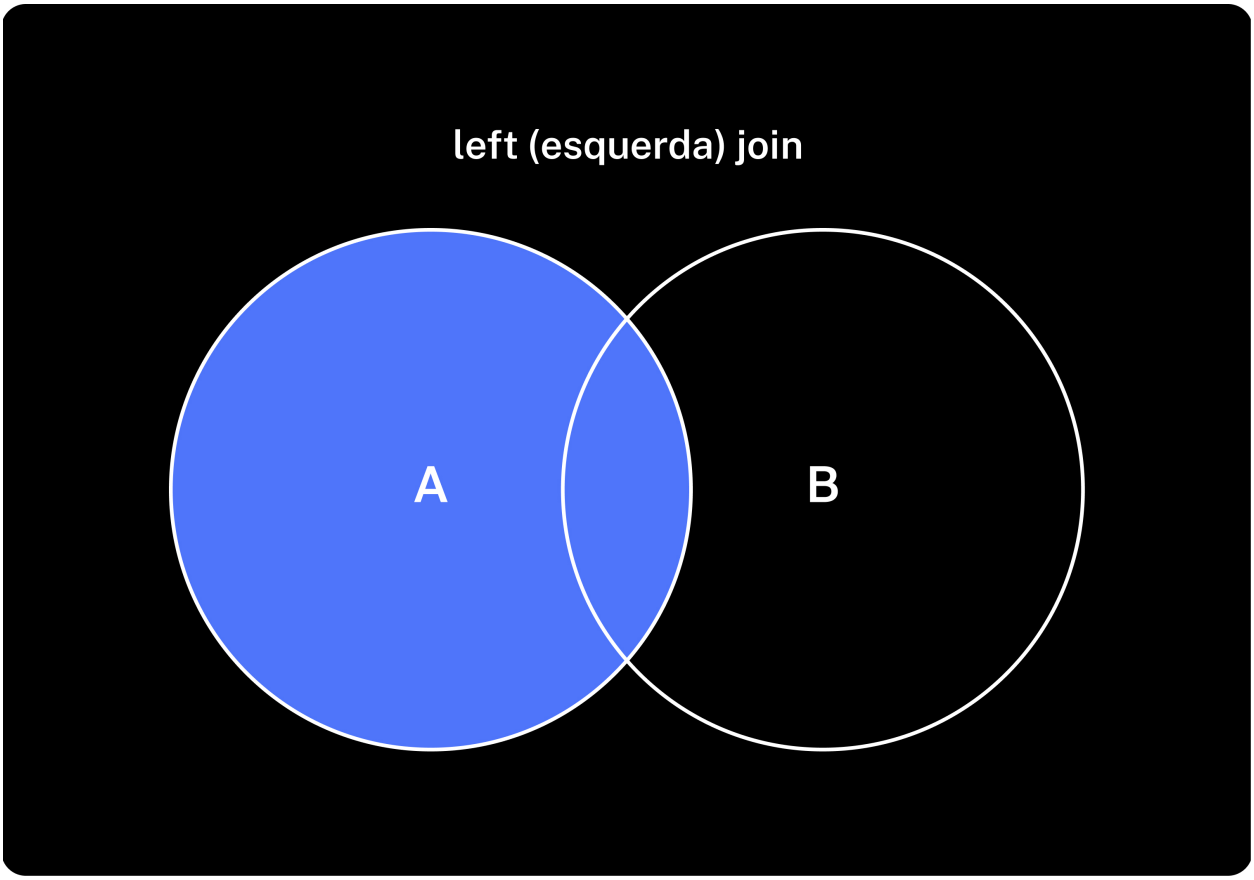
Outer Join. LEFT OUTER JOIN

Existem dois tipos de OUTER JOIN:

- **LEFT OUTER JOIN**
- **RIGHT OUTER JOIN**

Vamos dar nomes mais curtos para esses métodos: **LEFT JOIN** e **RIGHT JOIN**.

LEFT JOIN irá selecionar todos os dados da tabela à esquerda junto com as linhas da tabela direita que satisfazerem a condição de junção. RIGHT JOIN fará o mesmo, mas para a tabela à direita.



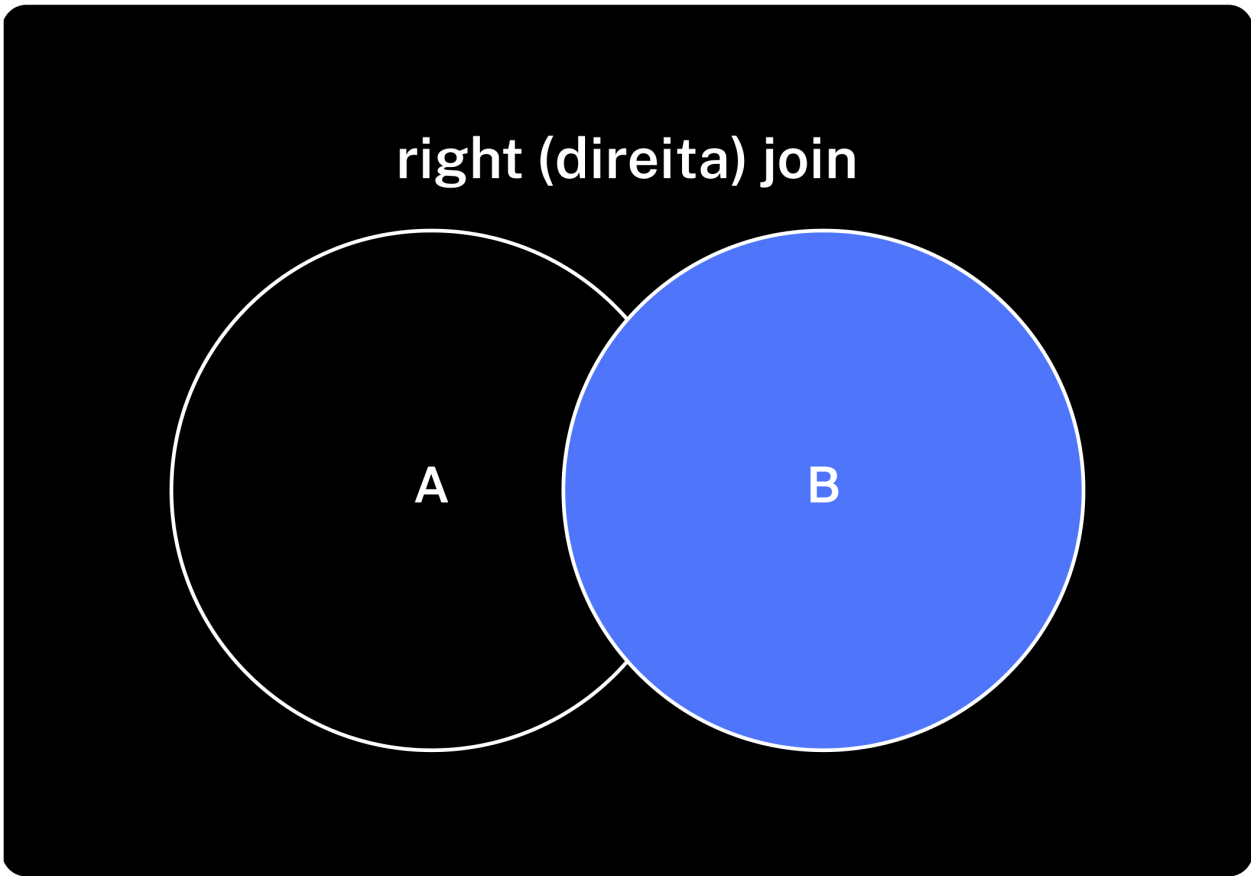
Aqui está a sintaxe de uma sentença com LEFT JOIN:

```
SELECT
  TABLE_1.field_1 AS field_1,
  TABLE_1.field_2 AS field_2,
  ...
  TABLE_2.field_n AS field_n
FROM
  TABLE_1
LEFT JOIN TABLE_2 ON TABLE_2.field = TABLE_1.field;
```

Assim como em consultas INNER JOIN, o nome da tabela é indicado para cada campo. Note que com INNER JOIN a ordem na qual as tabelas são listadas é significante.

Outer Join. RIGHT JOIN

A RIGHT JOIN é a irmã gêmea da LEFT JOIN. Porém, ao contrário da irmã, ela pega todos os dados da tabela à direita, e as linhas correspondentes da tabela à esquerda.



Essa é a aparência de uma consulta de RIGHT JOIN:

```
SELECT
  TABLE_1.field_1 AS field_1,
  TABLE_1.field_2 AS field_2,
  ...
  TABLE_2.field_n AS field_n
FROM
  TABLE_1
RIGHT JOIN TABLE_2 ON TABLE_1.field = TABLE_2.field;
```

JUNTANDO MÚLTIPLAS TABELAS

Aqui está a sintaxe de uma consulta que usa INNER JOIN várias vezes:

```
SELECT --listando somente os campos necessários
  TABLE_1.field_1 AS field_1,
  TABLE_1.field_2 AS field_2,
  ...
  TABLE_3.field_n AS field_n
FROM
  TABLE_1
INNER JOIN TABLE_2 ON TABLE_2.field = TABLE_1.field
INNER JOIN TABLE_3 ON TABLE_3.field = TABLE_1.field;
```

Vamos juntar a segunda tabela, e depois a terceira, à primeira.

Juntando instruções

Os operadores **UNION** e **UNION ALL** fazem a união dos dados das tabelas. Esta é a sintaxe:

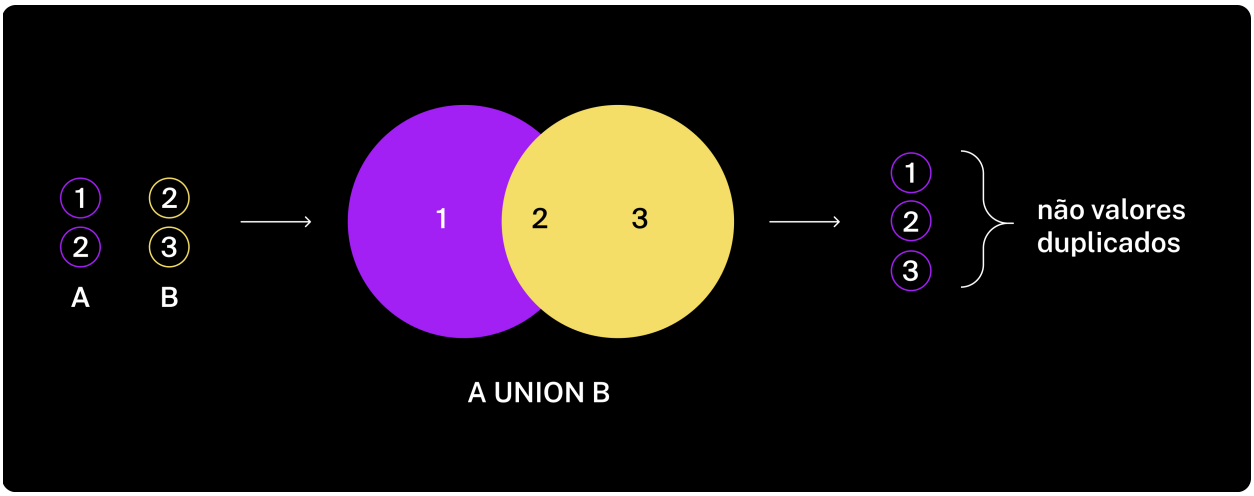
```
SELECT
  column_name_1
FROM
  table_1
UNION --( ou UNION ALL)
SELECT
  column_name_1
FROM
  table_2;
```

Aqui duas sentenças SELECT - FROM estão separadas pelo comando UNION.

Estas são as condições que devem ser cumpridas para que um UNION funcione:

- As duas primeiras tabelas devem possuir correspondências em relação ao número de colunas selecionadas e seus tipos de dados.
- Os campos devem estar organizados do mesmo modo na primeira e na segunda tabela.

UNION evita linhas duplicadas ao gerar uma tabela.


1
2
A
2
3
B
1
2
3
A UNION B
1
2
3
não valores
duplicados