Resumen del capítulo: Relaciones entre tablas

Tipos de relaciones entre tablas

Cuando una columna contiene los valores del campo de otra tabla, se le llama **clave foránea**. Es responsable de la relación entre las tablas.

Hay tres tipos de relaciones:

- una a una
- una a muchas
- muchas a muchas

En una relación **una a una**, cada fila de una tabla está conectada con una y solo una fila de la otra tabla. Es como si una tabla estuviera dividida en dos. Este es un tipo raro de relación y se usa predominantemente por razones de seguridad.

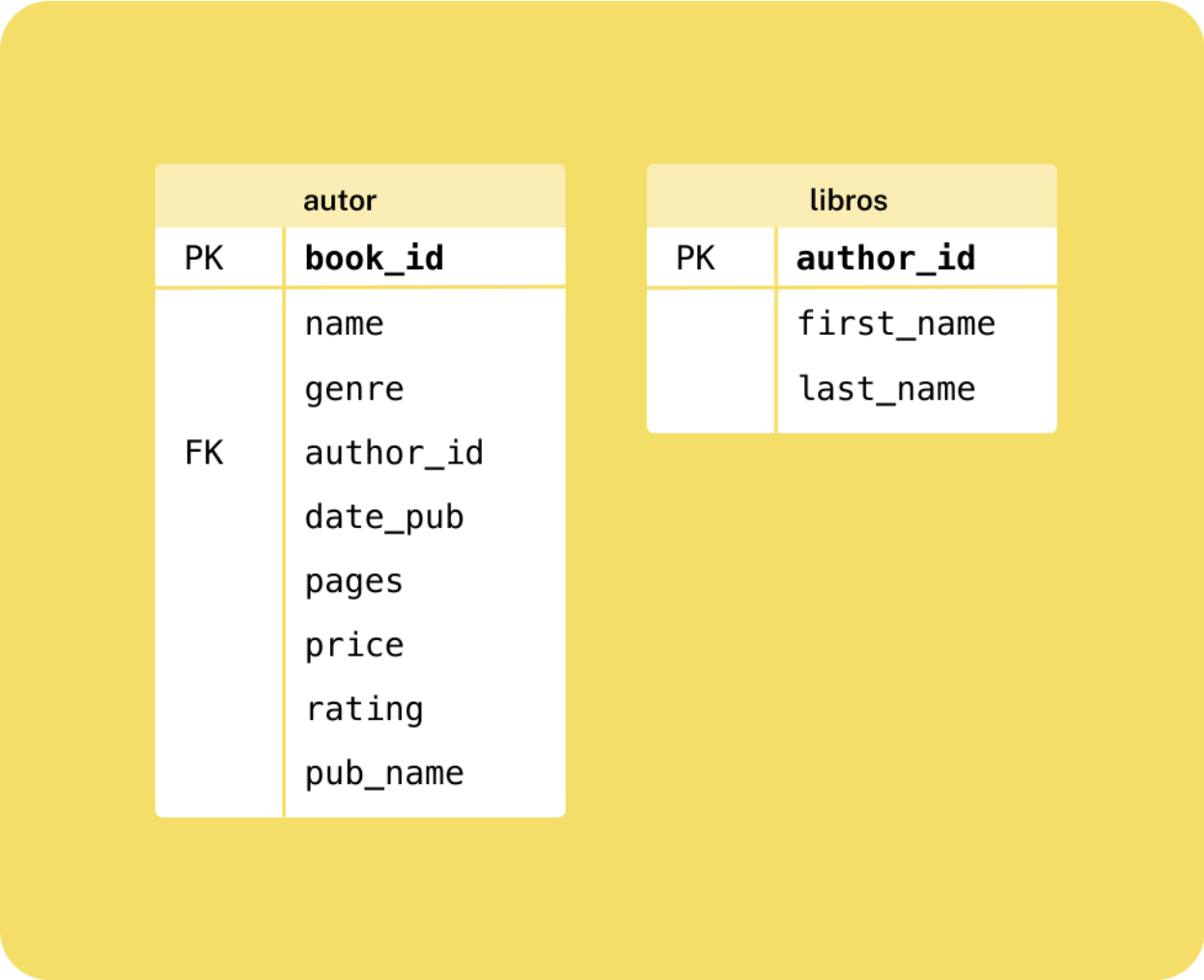
En una relación de **una a muchas**, cada fila de una tabla coincide con varias filas de otra tabla.

En una relación de **muchas a muchas**, varias filas de una tabla coinciden con varias filas de otra tabla. Este tipo de relación produce una tabla de asociación, que combina las claves primarias de ambas tablas.

Diagramas ER

La estructura de las bases de datos se puede mostrar con diagramas **ER (entidad-relación)**. Los diagramas muestran tablas y las relaciones entre ellas.

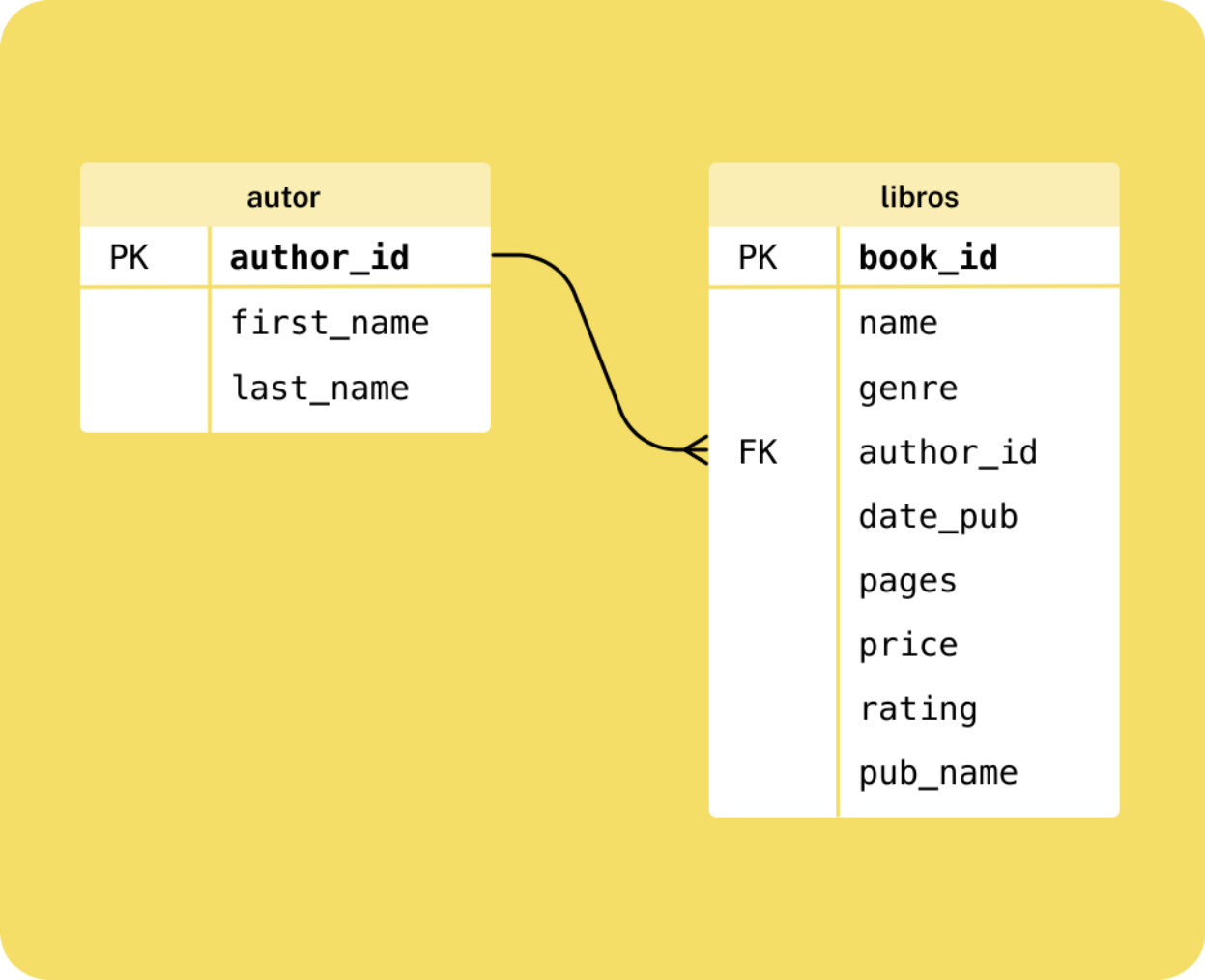
Las tablas se muestran como rectángulos (cajas) con dos partes. El nombre de la tabla va en la parte superior.



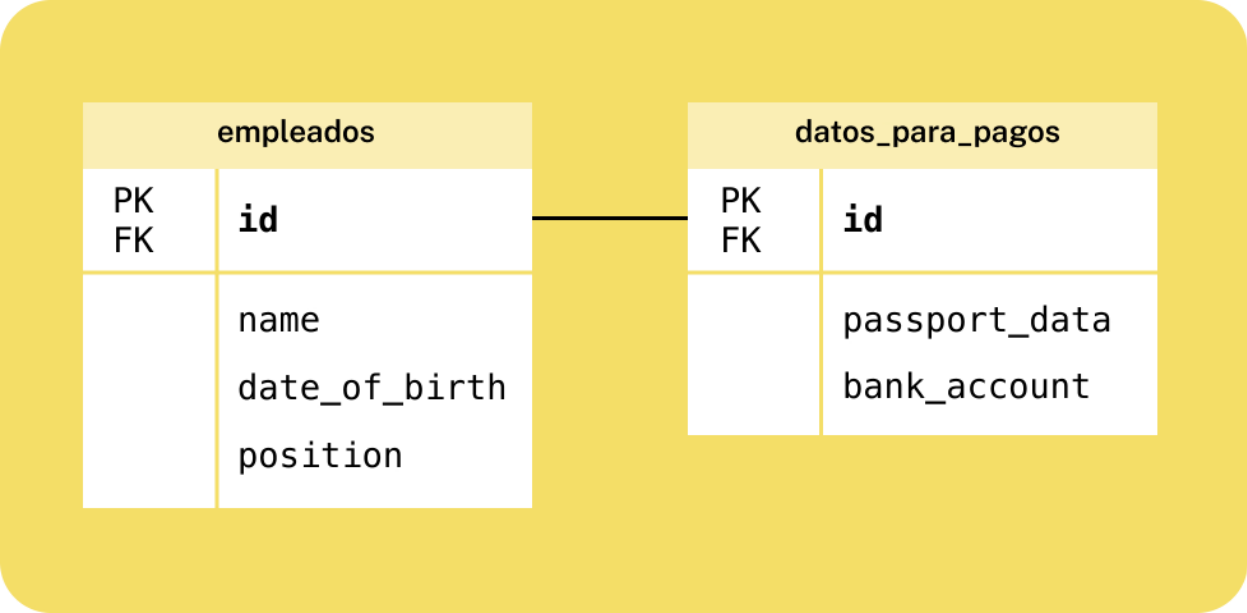
En la parte inferior vemos listas de campos de la tabla con una indicación de claves con las que están relacionados: primarias o foráneas. Las claves suelen estar marcadas como **PK** (primaria) o **FK** (foránea), pero también pueden estar marcadas con un icono de llave, un # u otro símbolo.

Los diagramas ER también muestran relaciones. El final de la línea que conecta dos tablas indica si uno o varios valores de una tabla coinciden con los valores de la otra.

Aquí hay una relación de una a muchas:



Y aquí hay una relación de una a una:



Tipos de usuarios de bases de datos

Muchos empleados están utilizando la base de datos maestra de una empresa al mismo tiempo. Cada uno de ellos necesita solo ciertos datos para hacer su trabajo. Por lo tanto, las cosas deben organizarse de tal manera que los usuarios no interfieran en el trabajo de los demás. De ahí la necesidad de **administradores de bases de datos**. Ellos gestionan el acceso de los usuarios, controlan la carga en el sistema, se encargan de la seguridad y hacen copias de seguridad.

Las bases de datos son como criaturas vivas que crecen constantemente. **Los arquitectos** y **desarrolladores de bases** de datos se aseguran de que estas crezcan de manera saludable. Las decisiones de estos especialistas determinan la estructura, integridad y plenitud de la base de datos, así como sus posibilidades de escalado (añadiendo nuevas tablas, relaciones y funciones). Los arquitectos y desarrolladores son responsables del rendimiento de la base de datos.

Los ingenieros de datos son responsables de agregar datos a la base de datos. También se les llama especialistas en **ETL**, ya que extraen, transforman y cargan datos en bases de datos.

Los analistas, en este esquema, son usuarios típicos. Escriben consultas a bases de datos y recuperan los datos necesarios, los que luego analizan y utilizan para probar

hipótesis. Los analistas trabajan más de cerca que los demás con los datos, y son los primeros en encontrar campos o tablas faltantes. Acostúmbrate a comunicar de inmediato tales "descubrimientos" a los desarrolladores, si deseas que dichos errores se resuelvan rápidamente.

Buscar valores vacíos

En SQL, se dice que las celdas vacías son **NULL**. El operador **IS NULL** los busca:

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

¡Ten en cuenta que IS importa! Lo siguiente no funcionará:

```
SELECT
 *
FROM
  table_name
WHERE
  column_name = NULL; -- ¡este código no cumplirá!
```

Para excluir filas con valores NULL de la selección, usamos el operador NOT:

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

La construcción CASE se utiliza para realizar acciones cuando se cumplen ciertas condiciones. Es muy parecido a `if-elif-else` en Python:

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

Una condición sigue al operador **WHEN**. Si una fila de la tabla cumple esta condición, el código devuelve el resultado indicado en **THEN**. Si no, la misma fila se prueba con la siguiente condición. Si la fila no cumple con ninguna de las condiciones establecidas en WHEN, el código devuelve el valor indicado después de **ELSE**. A continuación, la construcción CASE se cierra con el operador **END**.

Buscar datos en la tabla

El operador **LIKE** busca en una tabla valores que sigan un patrón dado. Puedes buscar no solo una palabra, sino también un fragmento de ella.

Aquí tienes la sintaxis de las declaraciones LIKE:

```
column_name LIKE 'regular expression'
```

Indica la columna necesaria antes de LIKE y escribe una expresión regular después.

Las expresiones regulares en SQL son un poco diferentes a las de Python. Por ejemplo, el símbolo `_` reemplaza un valor de sustitución (1 carácter) en una expresión regular. El símbolo `%` reemplaza cualquier cantidad de caracteres. Un rango o secuencia de caracteres que debe contener una cadena se escribe entre corchetes `[]`. Si se van a excluir los caracteres, se utiliza la construcción `[^]`.

Rango o secuencia de caracteres

Si necesitamos encontrar un símbolo de una expresión regular como una subcadena, usamos el operador **ESCAPE**. Ha pasado un símbolo como un signo de exclamación. En la expresión regular, el signo de exclamación significa que el símbolo que le sigue no es parte de la expresión, sino la subcadena que se busca. Aquí hay un fragmento de código que encontrará todas las subcadenas que terminan con el símbolo `%` (digamos, "100%") en una tabla:

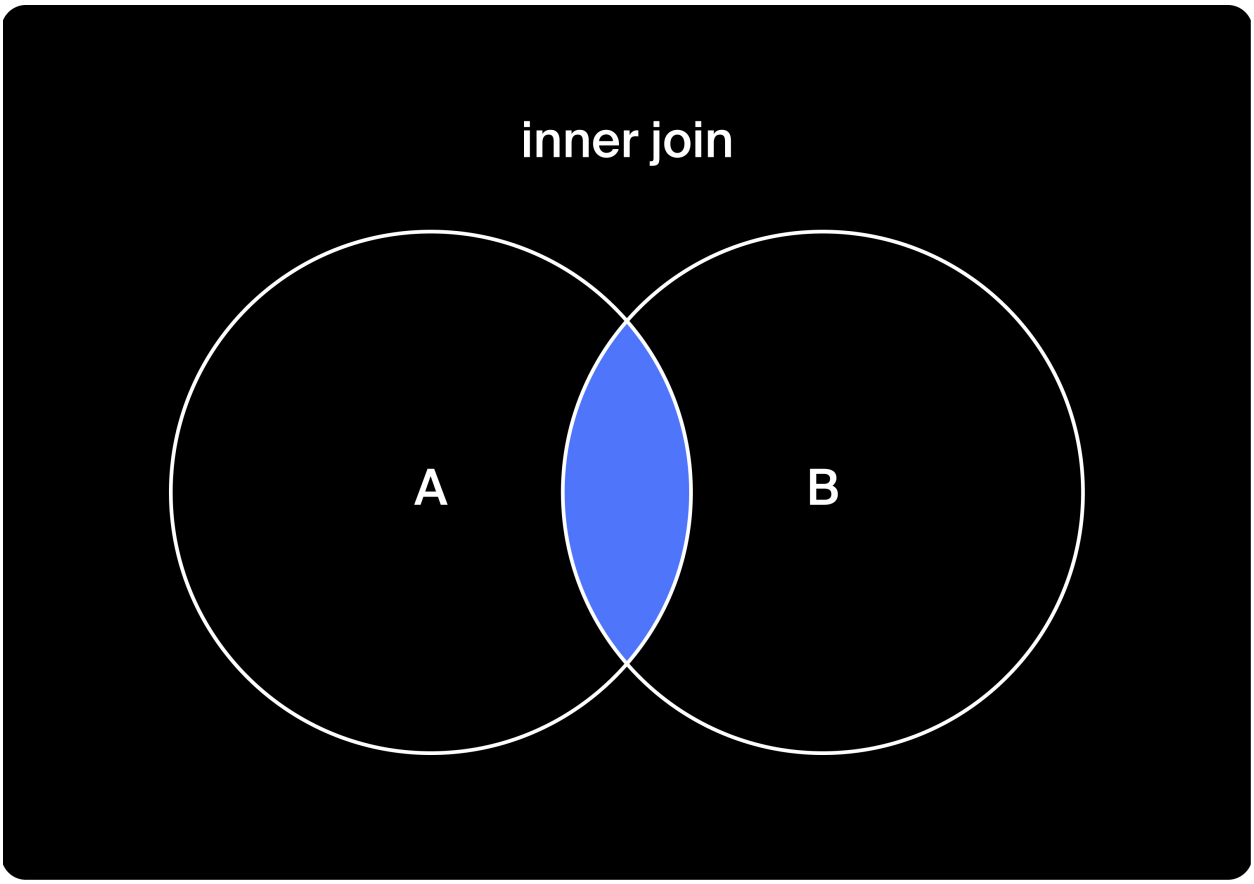
```
column_name LIKE '%!%' ESCAPE '!'
--busca todas las substrings que terminan con %
```

JOIN. INNER JOIN.

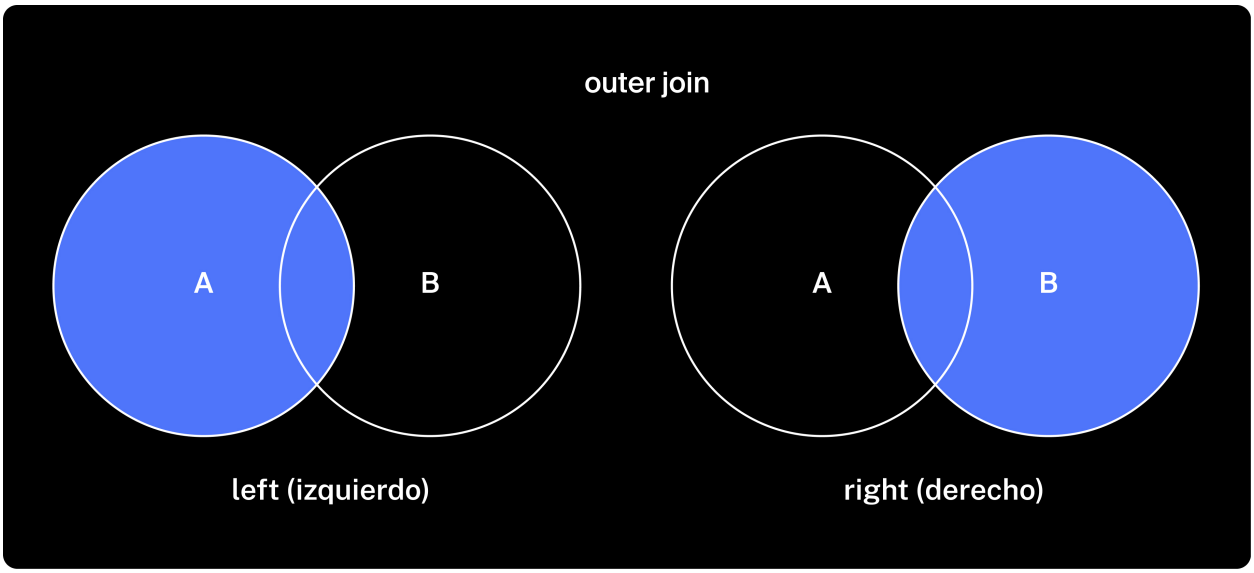
Es raro que todos los datos se almacenen en una tabla. Los analistas generalmente necesitan fusionar tablas; de ahí el operador **JOIN**.

Hay dos formas de unir tablas: la unión **INNER** y **OUTER.**

INNER JOIN devuelve solo aquellas filas que tienen valores coincidentes de una tabla a otra (la intersección de las tablas).



La unión OUTER recupera todos los datos de una tabla y agrega datos de la otra cuando hay filas coincidentes.



INNER JOIN

INNER JOIN selecciona solo los datos para los que se cumplen las condiciones de unión. El orden en que se unen las tablas no afecta el resultado final.

Aquí tienes un ejemplo de consulta con INNER JOIN:

```
SELECT --enlistar solo los campos necesarios
  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 a echar un vistazo más de cerca a la sintaxis.

- INNER JOIN es el nombre del método de unión. Luego viene el nombre de la tabla que se unirá a la tabla del bloque FROM.
- ON precede a la condición de unión: `TABLE_2.field_1 = TABLE_1.field_2`. Esto significa que solo se unirán las filas de la tabla que cumplan esta condición. En nuestro

caso, la condición es que el `field_1` de la segunda tabla coincida con el `field_2` de la primera.

Dado que los campos en diferentes tablas pueden tener los mismos nombres, se hace referencia a ellos tanto por el nombre de la tabla como por el nombre del campo. Primero viene el nombre de la tabla, luego el campo: `TABLE_1.field_1`.

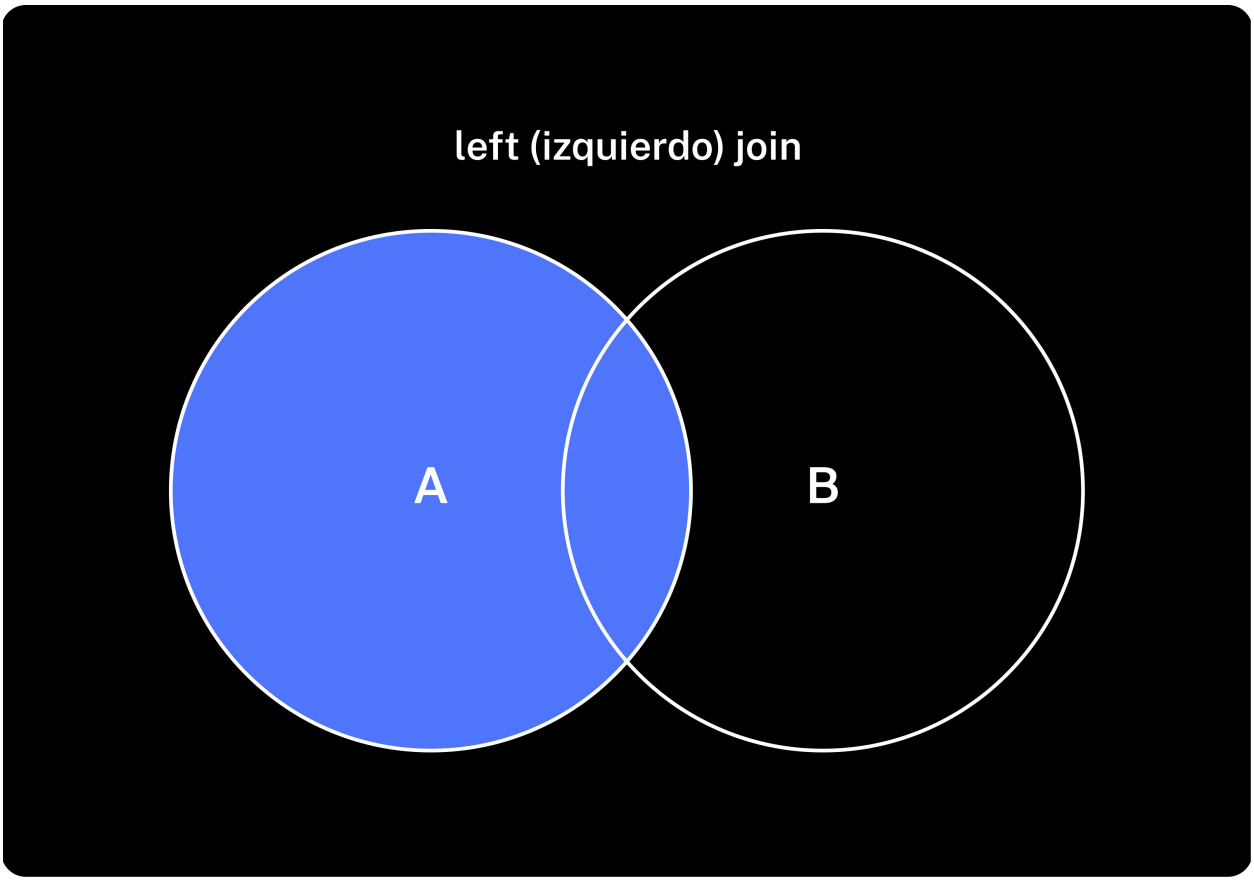
Outer Join. LEFT JOIN

Hay dos tipos de OUTER JOIN:

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

Daremos a estos métodos nombres cortos: LEFT JOIN y RIGHT JOIN.

LEFT JOIN seleccionará todos los datos de la tabla de la izquierda junto con las filas de la tabla de la derecha que cumplen con la condición de unión. RIGHT JOIN hará lo mismo, pero para la tabla de la derecha.



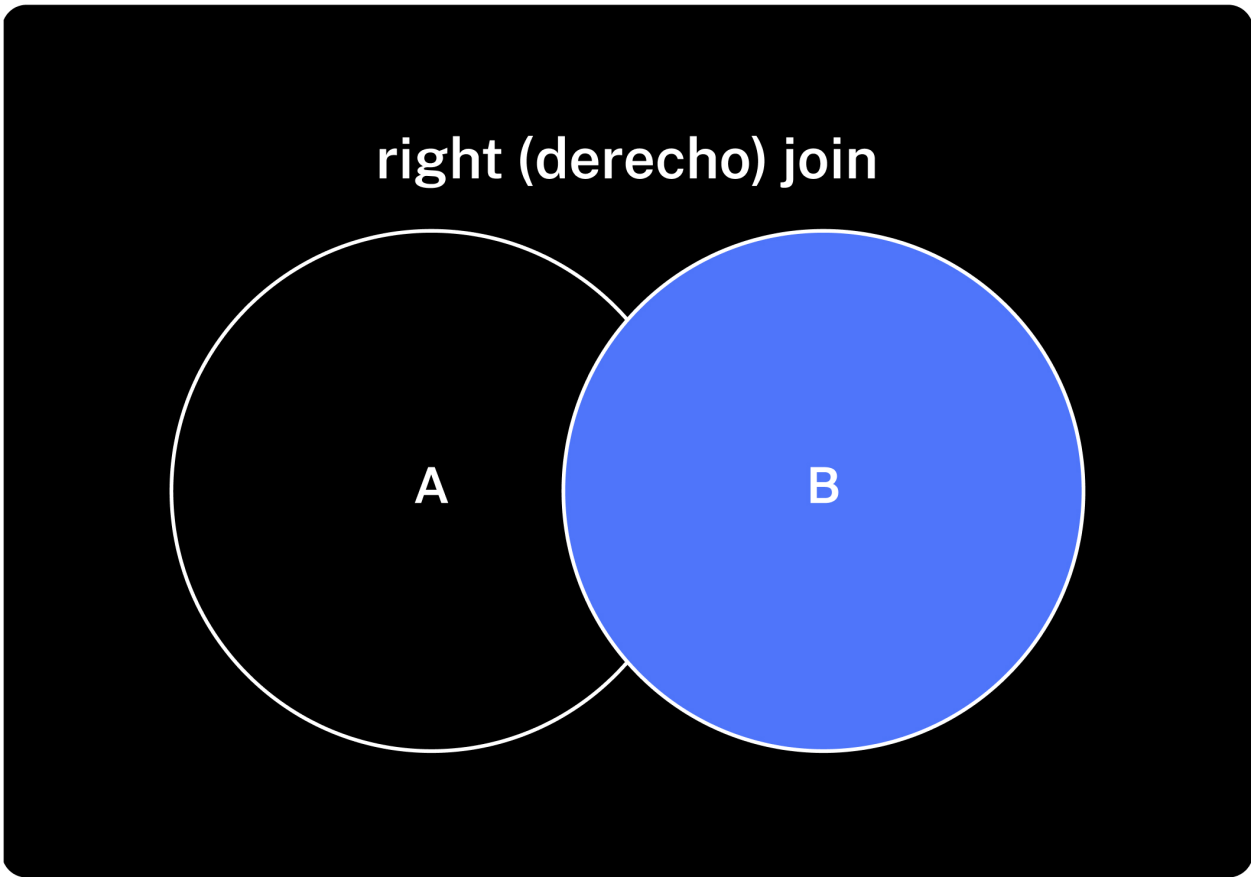
Aquí está la sintaxis de una declaración con LEFT JOIN:

```
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;
```

Al igual que con las consultas INNER JOIN, el nombre de la tabla se indica para cada campo. Ten en cuenta que con OUTER JOIN, el orden en que se mencionan las tablas tiene importancia.

Outer Join. RIGHT JOIN

RIGHT JOIN es el gemelo de LEFT JOIN. Pero a diferencia de su hermano, toma todos los datos de la tabla de la derecha y las filas correspondientes de la tabla de la izquierda.



Así es como se ve la consulta con 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;
```

Unir varias tablas

Esta es la sintaxis de una consulta que usa INNER JOIN varias veces:

```
SELECT --enlistar solo los campos necesarios
  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 a unir la segunda tabla, después la tercera, a la primera.

Unir declaraciones

Los operadores **UNION** y **UNION ALL** reúnen datos de tablas. La sintaxis es la siguiente:

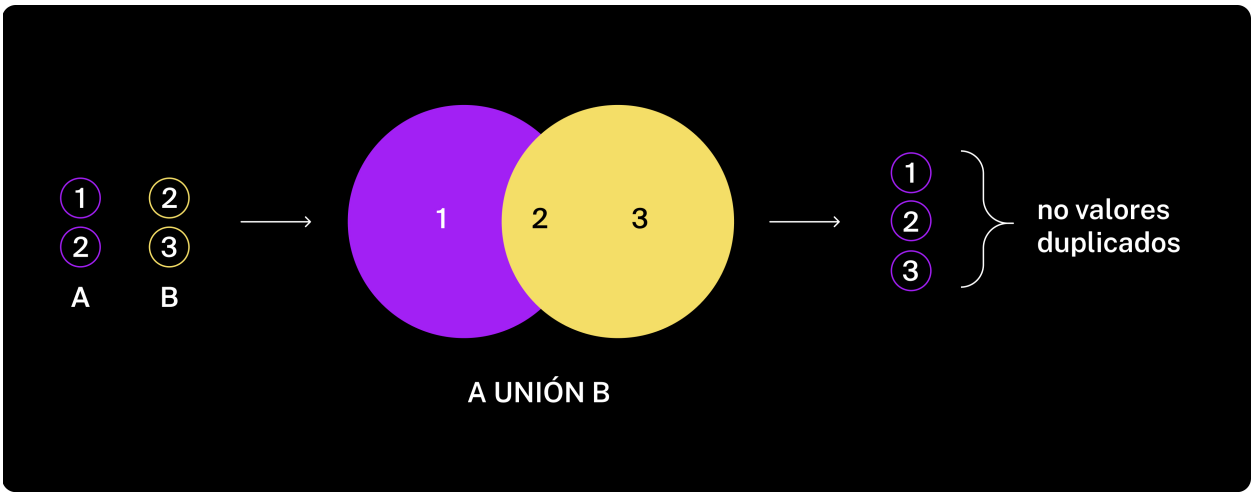
```
SELECT
  column_name_1
FROM
  table_1
UNION --( o UNION ALL)
SELECT
  column_name_1
FROM
  table_2;
```

Aquí dos declaraciones SELECT - FROM están separadas por el comando UNION.

Estas son las condiciones que se deben cumplir para que UNION funcione:

- La primera y la segunda tabla deben coincidir con respecto al número de columnas seleccionadas y sus tipos de datos
- Los campos deben estar en el mismo orden en la primera y la segunda tabla.

UNION evita duplicar filas cuando genera una tabla.


1
2
A
2
3
B
1
2
3
A UNIÓN B
1
2
3
no valores
duplicados