So far, we have learned how to create multiple tables and set up the relationships with each other, but suppose we now have a report requirement, which needs to display all orders and the details of users who have the order. What do we do?

The `JOIN` keyword in SQL is responsible for this kind of work, combining different tables together. So let's start learning how to use this tool!

LEFT JOIN

When using `JOIN`, the first thing we are familiar with is `LEFT JOIN`, which can be imagined as treating the table on the left side of the statement as the main table, and the other table as the attached table. When using the JOIN related syntax, all columns from both tables are displayed. However, if any the specific record of main table does not include any attached table records, the values of the columns in the attached table will be set to NULL.

This idea will be easier to understand with an example. Let's assume we have two tables:

orders

id	user_id	note
1	1	some information
2	2	some comments
3	2	no comments
4	NULL	weird

users

id	name	age
1	John	40
2	May	30

3	Tim	25

To help solidify the new concepts, you can follow along by editing the provided table below:

Suppose we want to use `orders` as the main table and combine `users` as the attached table, then we can say orders left join users and write the following query:

```
SELECT * FROM `new_schema`.`users`
LEFT JOIN `new_schema`.`orders` ON `users`.`id` = `orders`.`user_id`;
```

Here we can focus on the `LEFT JOIN` line and split it into 2 parts: 1. `LEFT JOIN`: In the above case, the main table is `users`. This is followed by the table used as an attached table, that is, `orders`. 2. `ON`: This keyword is to let SQL know how to connect the two tables. So you can notice that we set the `id` column of `users` to be connected to the `user_id` column of `orders`. This is the most important usage of how to apply relationships since we learned about relationships in the previous chapter.

Result:

id	name	age	id	user_id	note
1	John	40	1	1	some information
2	May	30	2	2	some comments
2	May	30	3	2	no comments
3	Tim	25	NULL	NULL	NULL

We can see that because `users` is the main table, even if there is a user record without corresponding order data, the record will still be fetched. But the columns of `orders` in that record will contain a NULL value. For example, here there are no records in the `orders` table that have a `user_id` equal to 3, so for Tim (whose `id` is 3), we fill in the remaining columns with `NULL`.

RIGHT JOIN

Since there is `LEFT JOIN`, there must also be `RIGHT JOIN`, and `RIGHT JOIN` can actually be thought of as the opposite usage of `LEFT JOIN`. It makes the table on the right the main table, and the table on the left the attached table.

And suppose our SQL statement is as follows:

```
SELECT * FROM `new_schema`.`users`
RIGHT JOIN `new_schema`.`orders` ON `users`.`id` = `orders`.`user_id`;
```

Result:

id	name	age	id	user_id	note
1	John	40	1	1	some information
2	May	30	2	2	some comments
2	May	30	3	2	no comments
NULL	NULL	NULL	4	NULL	weird

We can see that when it fetches all the `orders` data, the parts without `users` data in front will be filled with NULL.

Some of you may have noticed a trick:

If we reverse the order of the two tables when using `LEFT JOIN`, won't this be the same as using a `RIGHT JOIN`?

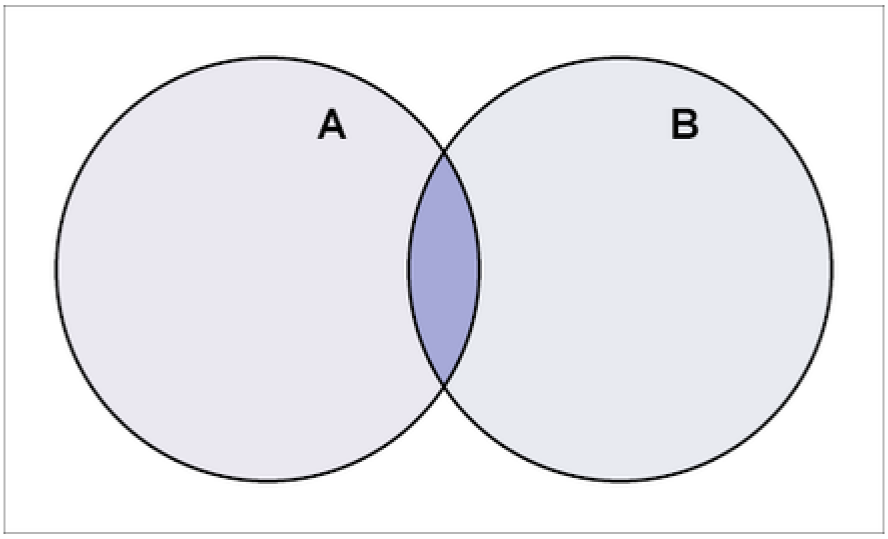
That's right! In fact, the above `RIGHT JOIN` statement is equivalent to:

```
SELECT * FROM `new_schema`.`orders`
LEFT JOIN `new_schema`.`users` ON `users`.`id` = `orders`.`user_id`;
```

The only difference will be the order in which the columns are displayed. When applying this trick with other programming languages, most of them can ignore this difference and use the data we provide normally!

INNER JOIN

Although we have learned to combine data, do you occasionally find it strange to see data with NULL? At this time, we can learn how to use `INNER JOIN` to fetch records where both tables can be linked together. That is a common scenario for filtering data in practice, and the concept is like taking an intersection).



SQL syntax:

```
SELECT * FROM `new_schema`.`users`
INNER JOIN `new_schema`.`orders` ON `users`.`id` = `orders`.`user_id`;
```

Result:

id	name	age	id	user_id	note
1	John	40	1	1	some information
2	May	30	2	2	some comments
2	May	30	3	2	no comments

How to Select the Needed Column With the Same Name?

Notice that both tables have an `id` column in the above example. Suppose we only want to display the `id` of the `order` table. What should we do? Can we write it directly as `SELECT id`?

If this is the case, you cannot directly write the name of the column. When we select a column, and there is a duplicate column name present, we must specify which column we want by including the table name. However, columns with non-overlapping names can be written directly. Here is an example:

```
SELECT `orders`.`id` AS order_id , `name` FROM `new_schema`.`users`
INNER JOIN `new_schema`.`orders` ON `users`.`id` = `orders`.`user_id`;
```

Result:

order_id	name
1	John
2	May
3	May

Note: Except MySQL, most database software systems usually support `FULL OUTER JOIN`, please refer to Oracle - FULL OUTER JOIN for more details