SQL Join

With relational databases, the information you want is often stored in several tables. In such scenarios, you'll need to join these tables. This is where the SQL JOIN comes into play.

The JOIN clause in SQL is used to **combine rows from several tables based on a related column between these tables**.

For the examples, we will use information about a publishing house that publishes original and translated books. Our database contains four tables: `books`, `authors`, `editors`, and `translators`.

books

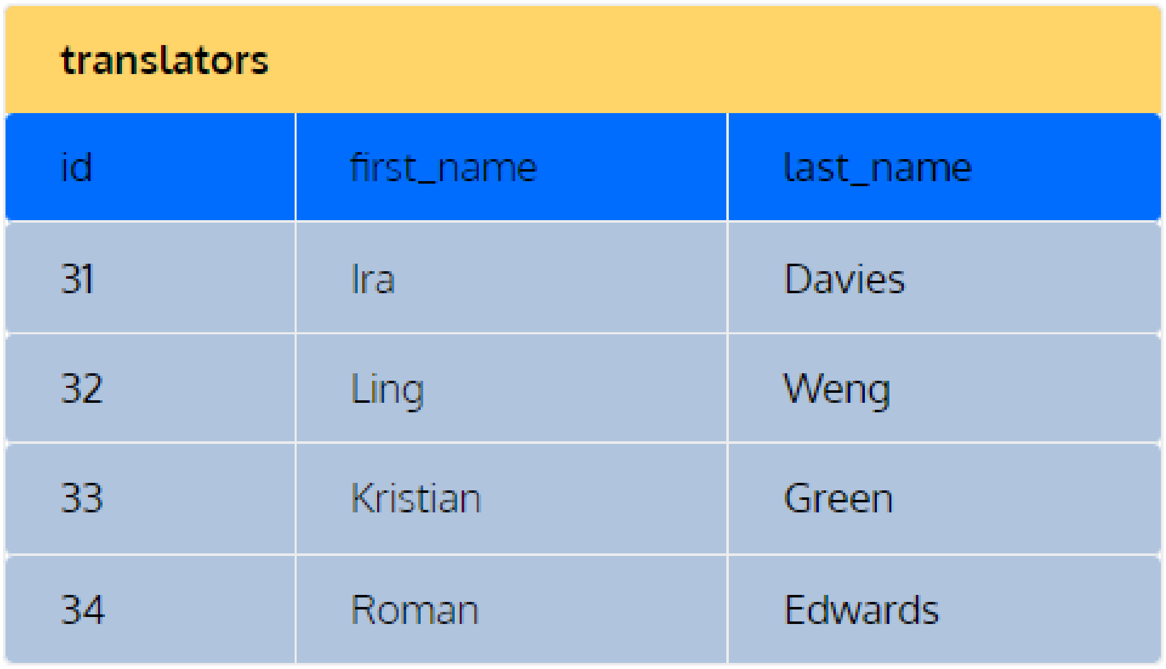
id	title	type	author_id	editor_id	translator_id
1	Time to Grow Up!	original	11	21	
2	Your Trip	translated	15	22	32
3	Lovely Love	original	14	24	
4	Dream Your Life	original	11	24	
5	Oranges	translated	12	25	31
6	Your Happy Life	translated	15	22	33
7	Applied AI	translated	13	23	34
8	My Last Book	original	11	28	

authors

id	first_name	last_name
11	Ellen	Writer
12	Olga	Savelieva
13	Jack	Smart
14	Donald	Brain
15	Yao	Dou

editors

id	first_name	last_name
21	Daniel	Brown
22	Mark	Johnson
23	Maria	Evans
24	Cathrine	Roberts
25	Sebastian	Wright
26	Barbara	Jones
27	Matthew	Smith



translators

id	first_name	last_name
31	Ira	Davies
32	Ling	Weng
33	Kristian	Green
34	Roman	Edwards

INNER JOIN

We'll start with a basic `INNER JOIN`, or simply, `JOIN`. This join type is used when we want to display **matching records from two tables**.

Example #1

Let's say we want to show book titles along with their authors (i.e., the author's first name and last name). The book titles are stored in the `books` table, and the author names are stored in the `authors` table.

In our SQL query, we'll join these two tables by matching the `author_id` column from the `books` table and the `id` column from the `authors` table

```
SELECT b.id, b.title, a.first_name, a.last_name
FROM books b
INNER JOIN authors a
ON b.author_id = a.id
ORDER BY b.id;
```

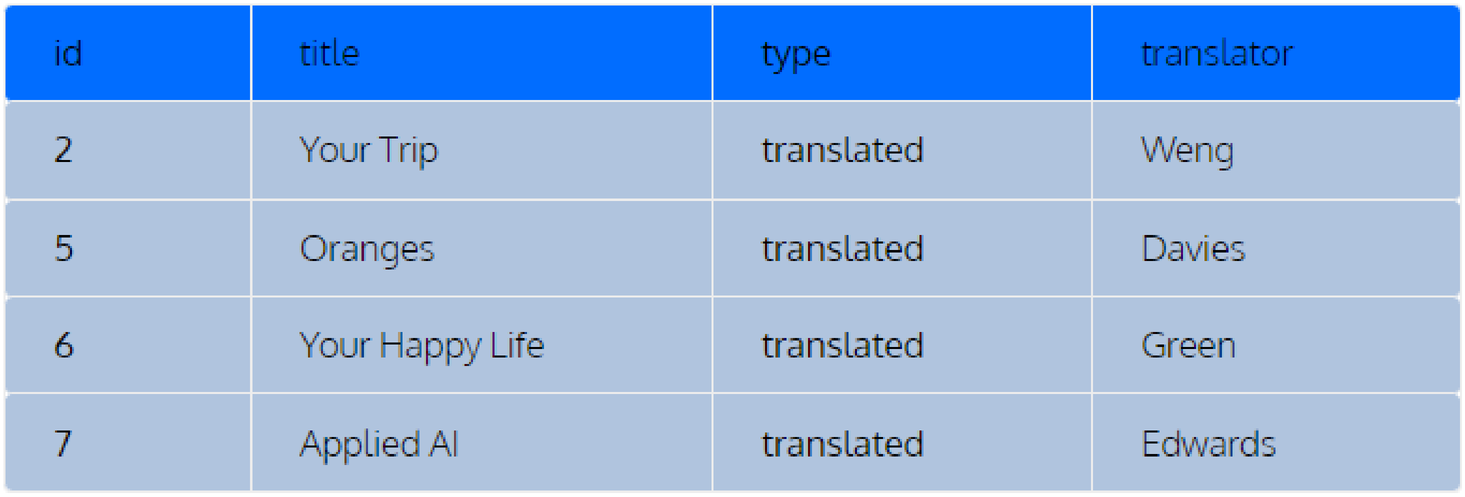
Here's the resulting set:

id	title	first_name	last_name
1	Time to Grow Up!	Ellen	Writer
2	Your Trip	Yao	Dou
3	Lovely Love	Donald	Brain
4	Dream Your Life	Ellen	Writer
5	Oranges	Olga	Savelieva
6	Your Happy Life	Yao	Dou
7	Applied AI	Jack	Smart
8	My Last Book	Ellen	Writer

Example #2

In our second example, we'll be displaying books along with their translators (i.e., the translator's last name). Only half of our books have been translated and thus have a corresponding translator. So, what would be the result of joining the `books` and `translators` tables using `INNER JOIN`?

```
SELECT b.id, b.title, b.type, t.last_name AS translator

FROM books b

JOIN translators t

ON b.translator_id = t.id

ORDER BY b.id;
```



id	title	type	translator
2	Your Trip	translated	Weng
5	Oranges	translated	Davies
6	Your Happy Life	translated	Green
7	Applied AI	translated	Edwards

LEFT JOIN

We'll start our overview of OUTER joins with the `LEFT JOIN`. You should apply this SQL JOIN type when you want to **keep all records from the left table and only the matched records from the right table**.

Example #3

For instance, let's say that we want to display information about each book's author and translator (i.e., their last names). We also want to keep the basic information about each book (i.e., `id`, `title`, and `type`).

To get all of this data, we'll need to join three tables: `books` for basic info on the books, `authors` for the authors' last names, and `translators` for the translators' last names.

As we have seen in the previous example, using the `INNER JOIN` (or simple `JOIN`) to join the `translators` table results in losing all of the records for original (not translated) books. That's not what we want now. So, to keep all of the books in the result set, we'll join the `books`, `authors`, and `translators` tables using the `LEFT JOIN`.

```
SELECT b.id, b.title, b.type, a.last_name AS author,

 t.last_name AS translator
```

```
FROM books b

LEFT JOIN authors a

ON b.author_id = a.id

LEFT JOIN translators t

ON b.translator_id = t.id

ORDER BY b.id;
```

id	title	type	author	translator
1	Time to Grow Up!	original	Writer	NULL
2	Your Trip	translated	Dou	Weng
3	Lovely Love	original	Brain	NULL
4	Dream Your Life	original	Writer	NULL
5	Oranges	translated	Savelieva	Davies
6	Your Happy Life	translated	Dou	Green
7	Applied AI	translated	Smart	Edwards
8	My Last Book	original	Writer	NULL

Example #4

This time, we want to show the basic book information (i.e., ID and title) along with the last names of the corresponding editors. Again, we want to keep all of the books in the result set. So, what would be the query?

```
SELECT b.id, b.title, e.last_name AS editor

FROM books b

LEFT JOIN editors e

ON b.editor_id = e.id

ORDER BY b.id;
```

id	title	editor
1	Time to Grow Up!	Brown
2	Your Trip	Johnson
3	Lovely Love	Roberts
4	Dream Your Life	Roberts
5	Oranges	Wright
6	Your Happy Life	Johnson
7	Applied AI	Evans
8	My Last Book	NULL

RIGHT JOIN

`RIGHT JOIN` is very similar to `LEFT JOIN`. I bet you guessed that the only difference is that `RIGHT JOIN` **keeps all of the records from the right table, even if they cannot be matched to the left table**. If you did, you're correct!

Example #5

Let's repeat our previous example, but this time, our task will be to keep all of the records from the `editors` table. Thus, we will have the same query as in *example #4* except that we replace `LEFT JOIN` with `RIGHT JOIN`:

```
SELECT b.id, b.title, e.last_name AS editor

FROM books b

RIGHT JOIN editors e

ON b.editor_id = e.id

ORDER BY b.id;
```

id	title	editor
1	Time to Grow Up!	Brown
2	Your Trip	Johnson
3	Lovely Love	Roberts
4	Dream Your Life	Roberts
5	Oranges	Wright
6	Your Happy Life	Johnson
7	Applied AI	Evans
NULL	NULL	Jones
NULL	NULL	Smith

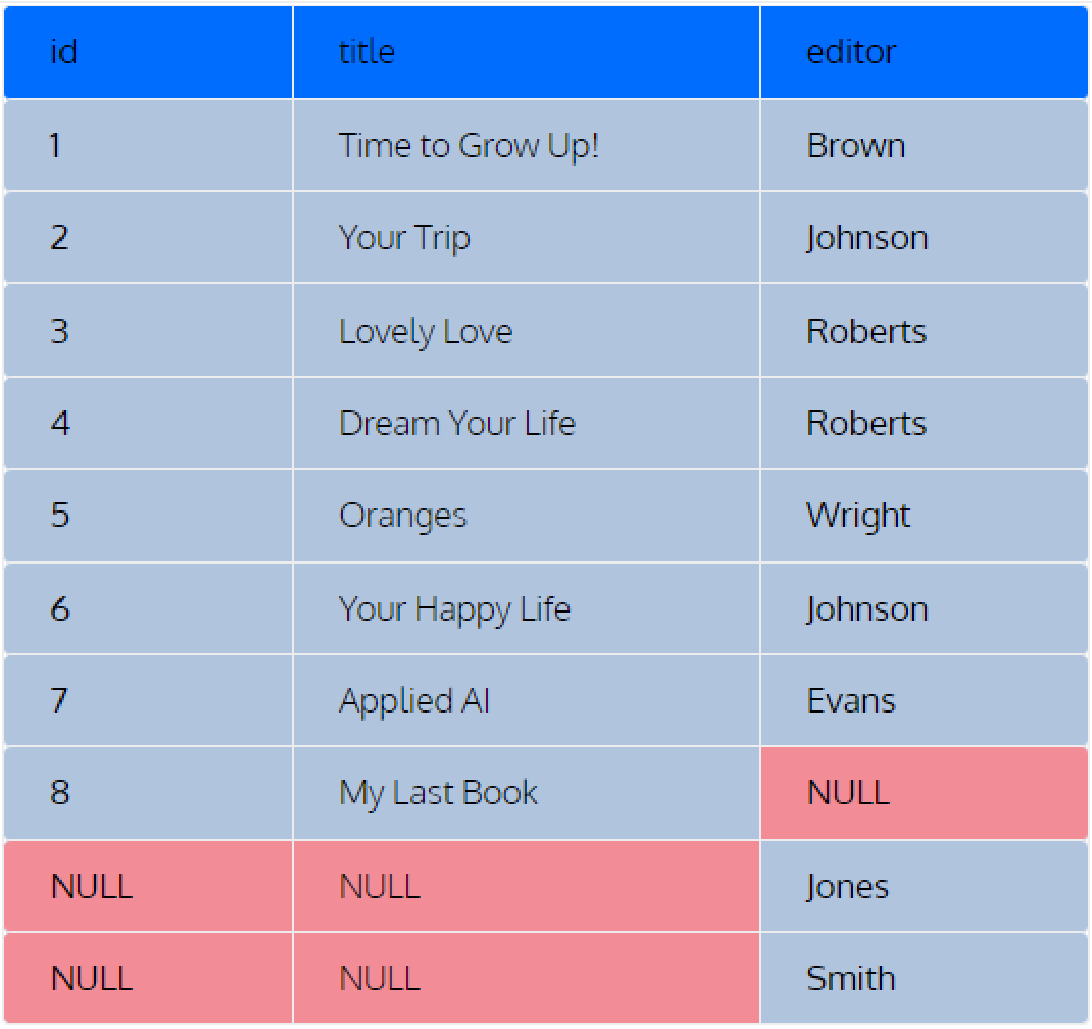
FULL JOIN

Here we arrived at the last outer join type, which is `FULL JOIN`. We use `FULL JOIN` when we want to **keep all records from all tables**, even unmatched ones. So, it's like `LEFT JOIN` and `RIGHT JOIN` combined. Let's go straight to the examples to see how this works in practice.

Example #6

To start with, let's again join the `books` and `editors` tables, but this time, we'll be keeping all records from both tables. We simply use `FULL JOIN` as the join keyword, leaving the rest of the query without any changes:

```
SELECT b.id, b.title, e.last_name AS editor

FROM books b

FULL JOIN editors e

ON b.editor_id = e.id

ORDER BY b.id;
```



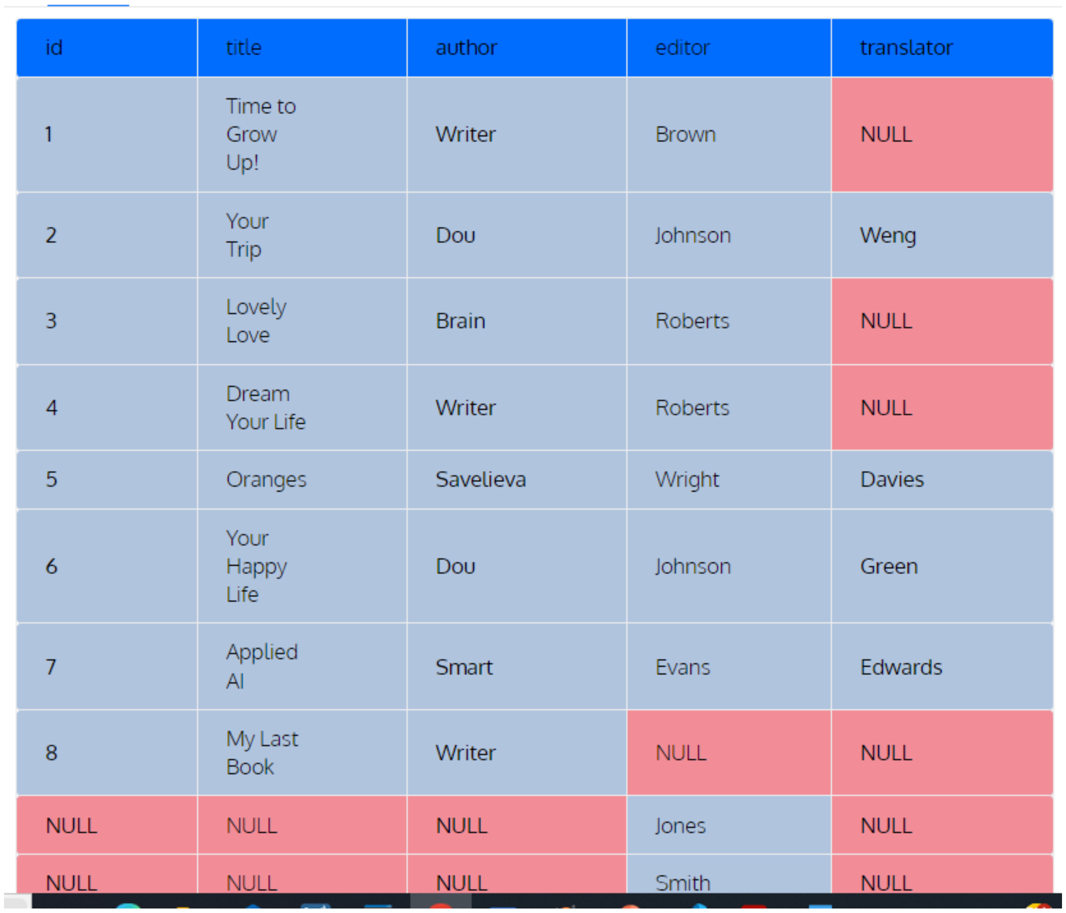
id	title	editor
1	Time to Grow Up!	Brown
2	Your Trip	Johnson
3	Lovely Love	Roberts
4	Dream Your Life	Roberts
5	Oranges	Wright
6	Your Happy Life	Johnson
7	Applied AI	Evans
8	My Last Book	NULL
NULL	NULL	Jones
NULL	NULL	Smith

Example #7

In our final example, we want to join all four tables to get information about all of the books, authors, editors, and translators in one table. So, we'll be using `FULL JOIN` throughout our SQL query:

```
SELECT b.id, b.title, a.last_name AS author, e.last_name AS editor,

    t.last_name AS translator

FROM books b

FULL JOIN authors a
```

```
ON b.author_id = a.id

FULL JOIN editors e

ON b.editor_id = e.id

FULL JOIN translators t

ON b.translator_id = t.id

ORDER BY b.id;
```



id	title	author	editor	translator
1	Time to Grow Up!	Writer	Brown	NULL
2	Your Trip	Dou	Johnson	Weng
3	Lovely Love	Brain	Roberts	NULL
4	Dream Your Life	Writer	Roberts	NULL
5	Oranges	Savelieva	Wright	Davies
6	Your Happy Life	Dou	Johnson	Green
7	Applied AI	Smart	Evans	Edwards
8	My Last Book	Writer	NULL	NULL
NULL	NULL	NULL	Jones	NULL
NULL	NULL	NULL	Smith	NULL

SELF JOIN

This can be useful when you have a table that contains hierarchical data or when you need to compare rows within the same table.

CategoryID	CategoryName	SubcategoryID	SubcategoryName
1	Electronics	2	Laptops
1	Electronics	3	Smartphones
4	Clothing	5	Men's Apparel
4	Clothing	6	Women's Apparel

Now, if you want to retrieve a list of categories along with their subcategories, you can use a self-join:

```
SELECT
    c1.CategoryID AS CategoryID,
    c1.CategoryName AS CategoryName,
    c2.SubcategoryID AS SubcategoryID,
    c2.SubcategoryName AS SubcategoryName
FROM
    your_table_name c1
JOIN
    your_table_name c2 ON c1.CategoryID = c2.CategoryID AND c1.SubcategoryID IS NULL;
```