**EXAMPLE 1: INNER JOIN**

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:

In the SELECT statement, we list the columns to be displayed: book id, book title, author’s first name, and author’s last name. In the FROM clause, we specify the first table to join (also referred to as the left table). In the INNER JOIN clause, we specify the second table to join (also referred to as the right table).

Then, we use the ON keyword to tell the database which columns should be used for matching the records (i.e., the author\_id column from the books table and the id column from the authors table).

Note also that we are using aliases for table names (i.e., b for books and a for authors). We assign the aliases in the FROM and INNER JOIN clauses and use them throughout the query. The table aliases reduce typing and make the query more readable.

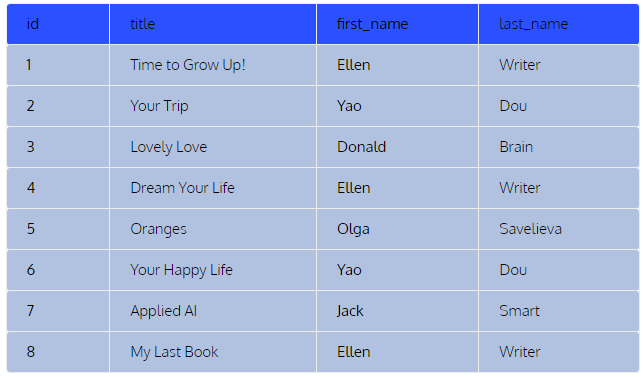
SELECT b.id, b.title, a.first\_name, a.last\_name

FROM books b

INNER JOIN authors a

ON b.author\_id = a.id

ORDER BY b.id;



For each record in the left table (i.e., books), the query checks the author\_id, then looks for the same id in the first column of the authors table. It then pulls the corresponding first name and last name.

Note that the **order of the tables doesn’t matter with INNER JOIN**, or simple JOIN. The result set would be exactly the same if we put the authors table in the FROM clause and the books table in the INNER JOIN clause.

INNER JOIN only displays records that are available in both tables. In our example, all books have a corresponding author and all authors have at least one corresponding book. So, let’s see what happens if some of the records are not matched.

**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.

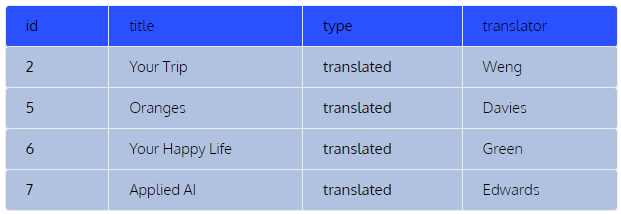
SELECT b.id, b.title, b.type, t.last\_name AS translator

FROM books b

JOIN translators t

ON b.translator\_id = t.id

ORDER BY b.id;



The query outputs only those books that have been translated. I’ve added the type column to make it clear. The rest of the books couldn’t be matched with the translators table and thus are not displayed. That’s how INNER JOIN works.

Also, note that in the second example, we were using JOIN rather than the INNER JOIN keyword. It has no impact on the result because INNER JOIN is the default join type in SQL.

**EXAMPLE 3: LEFT JOIN**

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).

SELECT b.id, b.title, b.type, a.last\_name AS author,

 t.last\_name AS translator

FROM books b

LEFT JOIN authors a

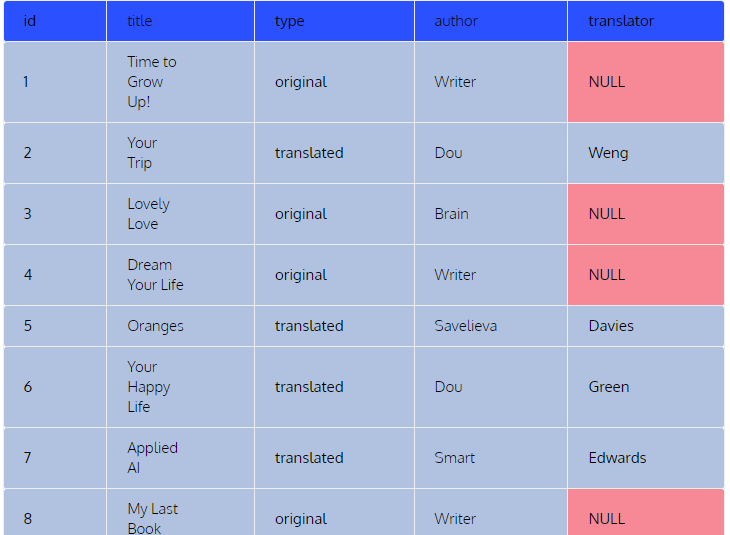
ON b.author\_id = a.id

LEFT JOIN translators t

ON b.translator\_id = t.id

ORDER BY b.id;

See that we start with the books table in the FROM clause, making it the **left table**. That’s because we want to keep all of the records from this table. The order of the other tables doesn’t matter.

In our query, we first LEFT JOIN the authors table based on the author\_id column from the books table and the id column from the authors table. Then, we join the translators table based on the translator\_id column from the books table and the id column from the translators table.

Note the NULL values in the translator column. These NULL values correspond to the records that were not matched in the translators table. These records are for original books without any translators involved.

**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.

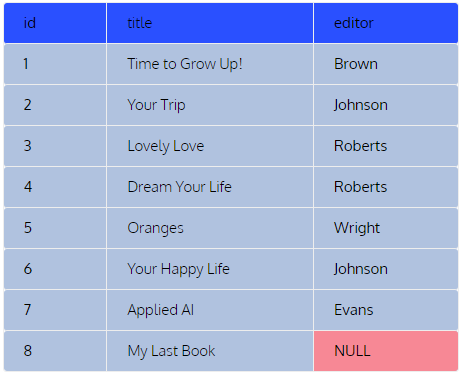
SELECT b.id, b.title, e.last\_name AS editor

FROM books b

LEFT JOIN editors e

ON b.editor\_id = e.id

ORDER BY b.id;



**EXAMPLE 5: RIGHT JOIN**

Let’s repeat our previous example, but this time, our task will be to keep all of the records from the editors table.

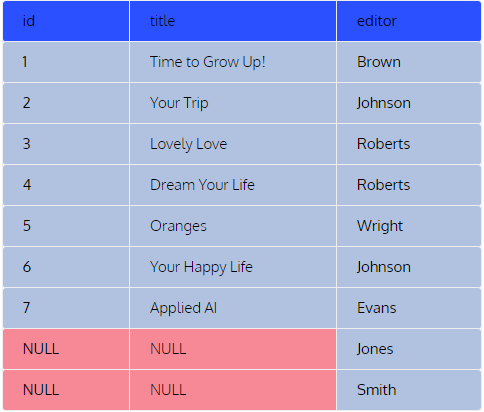
SELECT b.id, b.title, e.last\_name AS editor

FROM books b

RIGHT JOIN editors e

ON b.editor\_id = e.id

ORDER BY b.id;



With only one word changed in the query, the result is very different. We can see that indeed we have two editors (Jones and Smith) that don’t have corresponding books in our database.

And that’s not the only change. We also don’t have My Last Book in the result set. This record of the left table (i.e., books) was not matched in the right table (i.e., editors) and didn’t make it to the final result.

RIGHT JOINs are rarely used in practice because they usually can be replaced with LEFT JOINs that are much more common.

**EXAMPLE 6: FULL JOIN**

Let’s again join the books and editors tables, but this time, we’ll be keeping all records from both tables.

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

NOTE: this code will not work in MySQL because it does not support full outer join.

SELECT b.id, b.title, e.last\_name AS editor

FROM books b

LEFT JOIN editors e

ON b.editor\_id = e.id

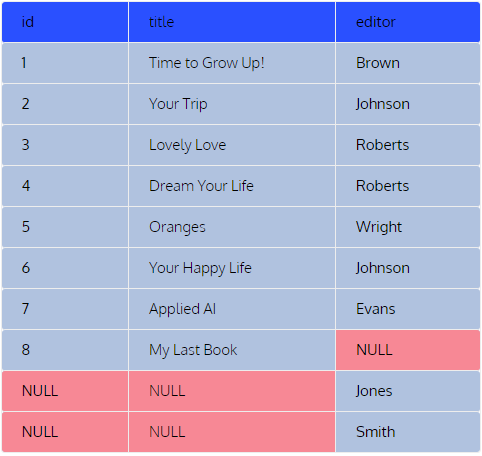
UNION ALL

SELECT b.id, b.title, e.last\_name AS editor

FROM books b

RIGHT JOIN editors e

ON b.editor\_id = e.id;



Note that **the order of the tables doesn’t matter with**FULL JOIN. The result would be the same if we swapped the tables by putting the editors table in the FROM clause and the books table in the FULL JOIN clause.

**EXAMPLE 7**

We want to join all four tables to get information about all of the books, authors, editors, and translators in one table.

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;

NOTE: this code will not work in MySQL because it does not support full outer join.

SELECT b.id, b.title, a.last\_name AS author, e.last\_name AS editor,

t.last\_name AS translator

FROM books b

LEFT JOIN authors a

ON b.author\_id = a.id

LEFT JOIN editors e

ON b.editor\_id = e.id

LEFT JOIN translators t

ON b.translator\_id = t.id

UNION ALL

SELECT b.id, b.title, a.last\_name AS author, e.last\_name AS editor,

t.last\_name AS translator

FROM books b

RIGHT JOIN authors a

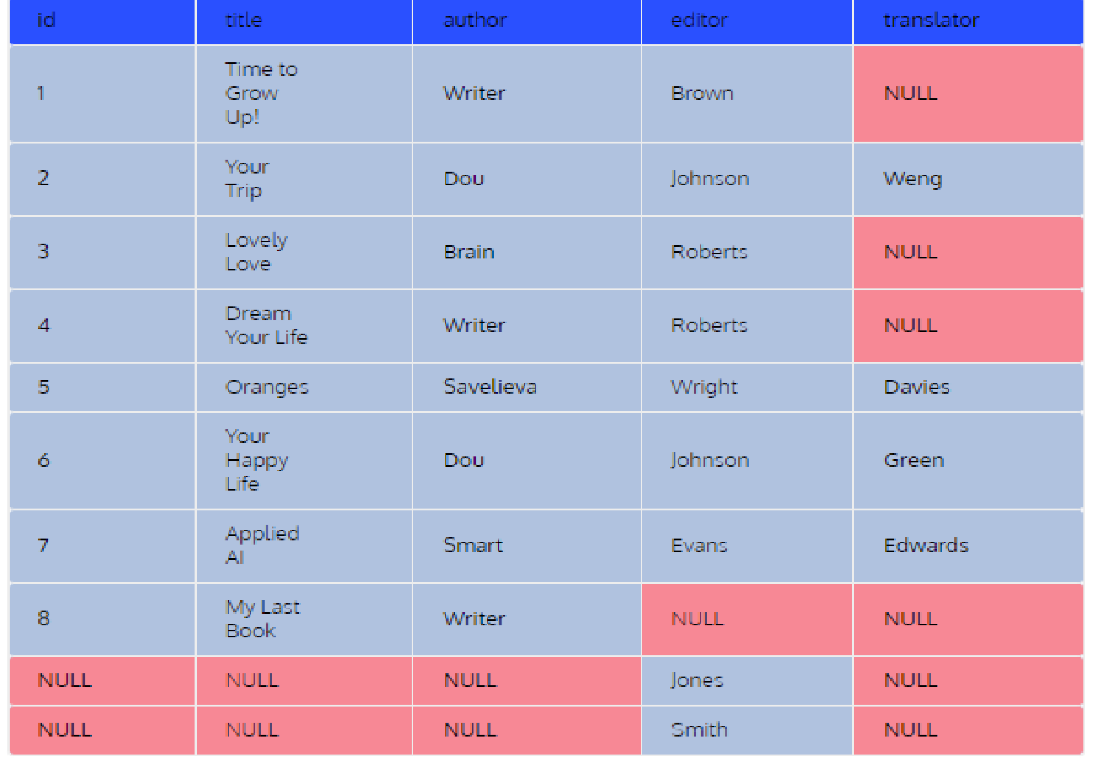
ON b.author\_id = a.id

RIGHT JOIN editors e

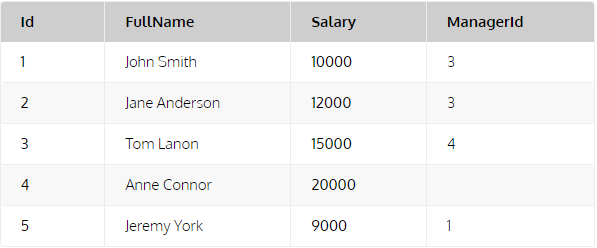
ON b.editor\_id = e.id

RIGHT JOIN translators t

ON b.translator\_id = t.id;

****

**EXAMPLE: SELF JOIN**

Let’s take a look at an example. Consider the table **Employees**:

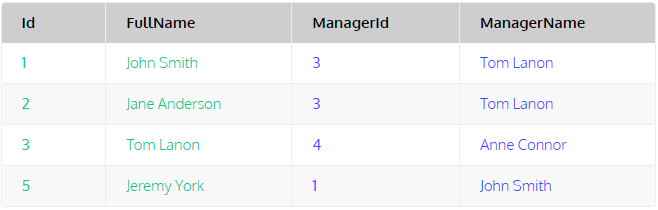
SELECT employee.id, employee.full\_name, employee.manager\_id,

manager.full\_name as manager\_name

FROM employees employee

JOIN employees manager

ON employee.manager\_id = manager.id;



The query selects the columns Id, FullName, and ManagerId from the table aliased **employee**. It also selects the FullName column of the table aliased **manager** and designates this column as ManagerName. As a result, every employee who has a manager is output along with his/her manager’s ID and name.

In this query, the **Employees** table is joined with itself and has two different roles:

* Role 1: It stores the employee data (alias **employee**).
* Role 2: It stores the manager data (alias **manager**).

By doing so, we are essentially considering the two copies of the **Employees** table as if they are two distinct tables, one for the employees and another for the managers.