7 SQL JOIN Examples With Detailed Explanations

Do you need to join several tables to get the necessary result set? The SQL JOIN is a basic yet important tool used by data analysts working with relational databases. And I understand it can be difficult to choose from the zillions of introductory guides to joins. In this article, I will focus on real-world examples with detailed explanations.

Introduction to 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**. You can get an overview of the SQL JOIN tool in this introductory article.

In this guide, I want to cover the basic types of SQL JOINs by going through several examples. I will discuss in detail the syntax of each query, how it works, how to build a condition, and how to interpret the results.

Want to know how to use JOINs? Check out our interactive SQL JOINs course.

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
	1Time to Grow Up!	original	11	21	
2	Your Trip	translated	15	22	32
	3Lovely Love	original	14	24	
	4Dream Your Life	original	11	24	
	5Oranges	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

If you want to practice joining tables in SQL with many examples, I recommend taking the SQL JOINs course. It includes 93 coding challenges!

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;
```

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.

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

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. 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

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. You can learn about other SQL JOIN types in this detailed guide.

Okay. Now we know how to join tables when we need only the matched records to be displayed. But, what if we want to keep all of the books in the resulting set without cutting the table to the translated books only? It's time to learn about outer joins!

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;
```

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.

Here's the resulting table:

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

Great! We kept all of the books!

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.

Hopefully, you have grasped the intuition behind LEFT JOINs. You can learn more about this type of SQL JOIN in this introductory guide.

Okay, let's go through another `LEFT JOIN` example to consolidate knowledge on the topic.

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

Pretty simple, right? We again kept all of the books in the result set, including the last one, which doesn't have a corresponding editor in our database (note the `NULL` value in the last row).

We can imagine that the editor is not present in the table of our current editors simply because they left the publishing house after editing the book.

Wait! What if we have some editors on the team who haven't yet published any books? Let's check with the next outer join type.

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

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. Looks like new hires!

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.

For example, in our case, we could take our query from example #4 and just swap **books** and **editors** by putting **editors** in the `FROM` clause, making it the *left table*, and putting **books** in the `LEFT JOIN` clause, making it the *right table*. The result would have been the same as the above table.

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

Looks great! As expected, we kept all of the books, even the one without a matching editor. Also, we kept all of the editors, even the ones that don't have any corresponding books yet.

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

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
	1Time to Grow Up!	Writer	Brown	NULL
2	Your Trip	Dou	Johnson	Weng
	3Lovely Love	Brain	Roberts	NULL
	4Dream Your Life	Writer	Roberts	NULL
	5Oranges	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

As requested, the table displays all of the books, authors, editors, and translators. The records that were not matched have `NULL` values. That's a great overview of the data stored in our database.

Do you want to practice SQL JOINs? Check out our SQL JOINs course!

Time to Practice SQL JOINs!

Proficiency with SQL JOINs is one of the key requirements for anybody working with relational databases. To help you navigate the different types of SQL JOINs, LearnSQL.com has developed a two-page SQL JOIN cheat sheet. It provides the syntax of the different JOINs as well as examples.

However, to master JOINs, you need lots of practice. I recommend starting with the interactive SQL JOINs course that covers the important types of JOINs by going through dozens of examples and exercises. Learn more about this course in this overview article.

BONUS: Here are the top 10 SQL JOIN interview questions and how to answer them.

Happy learning!