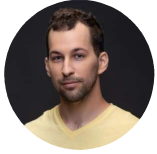


# SQL for Data Analysis - Tutorial for Beginners - ep5

2017-07-10



Tomi Mester

Combining tables is a key component in data science and analytics. And SQL is really good at it! Of course each case is different, but I run into data science tasks all the time in which joining two multi-million-row data tables took *~20-30 minutes* in Python (<https://data36.com/learn-python-for-data-science-from-scratch/>) or bash (<https://data36.com/learn-data-analytics-bash-scratch/>)... **and not more than ~10-20 seconds in SQL.** I'm not saying I couldn't have done those tasks in Python or bash at all... **But for sure SQL JOIN was the easiest and fastest solution!**

So let's learn how to use SQL JOIN to step up your analytics projects!

*Note: to get the most out of this article, you should not just read it, but actually do the coding part with me! So if you are on the phone, I suggest saving this article and continuing on your computer!*


# But be data36!

## start... (<https://data36.com>)

... I highly recommend going through these articles first – if you haven't done so yet:

1. Set up your own data server to practice: [How to install Python, SQL, R and Bash \(for non-devs\)](https://data36.com/data-coding-101-install-python-sql-r-bash/).  
(<https://data36.com/data-coding-101-install-python-sql-r-bash/>).
2. Install SQL Workbench to manage your SQL stuff better: [How to install SQL Workbench for postgresSQL](https://data36.com/install-sql-workbench-postgresql/).  
(<https://data36.com/install-sql-workbench-postgresql/>).
3. [SQL for Data Analysis ep1 \(SQL basics\)](https://data36.com/sql-for-data-analysis-tutorial-beginners/).  
(<https://data36.com/sql-for-data-analysis-tutorial-beginners/>).
4. [SQL for Data Analysis ep2 \(SQL WHERE clause\)](https://data36.com/sql-where-clause-tutorial-beginners-ep2/).  
(<https://data36.com/sql-where-clause-tutorial-beginners-ep2/>).
5. [SQL for Data Analysis ep3 \(SQL functions and GROUP BY\)](https://data36.com/sql-functions-beginners-tutorial-ep3/).  
(<https://data36.com/sql-functions-beginners-tutorial-ep3/>).
6. [SQL for Data Analysis ep4 \(SQL best practices\)](https://data36.com/sql-best-practices-data-analysts/).  
(<https://data36.com/sql-best-practices-data-analysts/>).

**How to Become a  
Data Scientist**



data36

video

course by Tomi Mester

Just subscribe to the Data36 Newsletter here (it's free)!

Email

☐ I accept Data36's [Privacy Policy](https://data36.com/privacy-policy/). (No spam. Only useful data science related content. When you subscribe, I'll keep you updated with a couple emails per week. You'll get articles, courses, cheatsheets, tutorials and many cool stuff.)

Get Access Now!

# What is SQL JOIN?

What does it mean to JOIN two tables? Here's a simple example.

Let's say we have these two datasets:

**TABLE #1**

INVENTION	INVENTOR
Rubik's Cube	Erno Rubik
Carburetor	Janos Csonka
Carburetor	Donat Banki
Ballpen	Laszlo Biro
Dynamo	Anyos Jedlik

(<https://data36.com>).

INVENTOR	PROFESSION
Erno Rubik	architect
Janos Csonka	engineer
Donat Banki	engineer
Laszlo Biro	journalist
Anyos Jedlik	engineer

In TABLE #1 we have inventions and in TABLE #2 we have the same inventors and their original professions. Let’s say we want to see which inventions was invented by an inventor with what original profession. To query that, we have to merge the two tables, based on the column that shows up in both: ***inventor***.

And this is what SQL JOIN is good for. After joining the two tables, this is what we will get:

<u>INVENTION</u>	<u>INVENTOR</u>	<u>PROFESSION</u>
Rubik’s Cube	Erno Rubik	architect
Carburetor	Janos Csonka	engineer
Carburetor	Donat Banki	engineer
Ballpen	Laszlo Biro	journalist

<u>INVENTION</u>	 <u>data36</u>	<u>PROFESSION</u>
Dynamo	<a href="https://data36.com">Anyos Jedlik</a>	engineer

Now we can finally see that the Rubik's Cube was invented by an architect!

*(Extra (not SQL-related) task: find out what these 5 inventions have in common!)*

Perfect, but what does this look like in practice?

## Get some data!

To put SQL JOIN into practice, we will get some data first.

Open up SQL Workbench! We are gonna create [\(https://data36.com/create-table-sql/\)](https://data36.com/create-table-sql/) two new temporary data tables: `playlist` and `toplist`. Run these two queries in SQL Workbench one by one.

```
1 CREATE TABLE playlist (  
2   artist VARCHAR,  
3   song VARCHAR);
```

```
1 CREATE TABLE toplist (  
2   tophit VARCHAR,  
3   play INT);
```

The new SQL tables have been created. Now, load some data into them!

Run these  data 360 by one in  
your SQL Workshop  
(<https://data36.com>).

```
1 INSERT INTO playlist (artist,song)
  VALUES
2   ('ABBA','Dancing Queen'),
3   ('ABBA','Gimme!'),
4   ('ABBA','The Winner Takes It
    All'),
5   ('ABBA','Mamma Mia'),
6   ('ABBA','Take a Chance On Me'),
7   ('Tove Lo','Cool Girl'),
8   ('Tove Lo','Stay High'),
9   ('Tove Lo','Talking Body'),
10  ('Tove Lo','Habits'),
11  ('Tove Lo','True Disaster'),
12  ('Avicii','Wake Me Up'),
13  ('Avicii','Waiting For Love'),
14  ('Avicii','The Nights'),
15  ('Avicii','Hey Brother'),
16  ('Avicii','Levels'),
17  ('Zara Larsson','Lush Life');
```

```
1 INSERT INTO toplist (tophit,play)
  VALUES
2   ('Dancing Queen',95145796),
3   ('Gimme!',32785696),
4   ('The Winner Takes It
    All',34458597),
5   ('Mamma Mia',47901900),
6   ('Take a Chance On Me',30654536),
7   ('Cool Girl',227055115),
8   ('Stay High',263901766),
9   ('Talking Body',272334711),
10  ('Habits',214685822),
11  ('True Disaster',27028538),
12  ('Wake Me Up',520259542),
13  ('Waiting For Love',399906192),
14  ('The Nights',278063930),
15  ('Hey Brother',321270703),
16  ('Levels',206004691),
17  ('Despacito',519689490);
```

*Note: This is actually real data that I pulled from Spotify when I wrote this article.*

Let's see whether everything works fine... Query the tables with:



```
1 SELECT
2     artist,
3     song    (https://data36.com),
4 FROM
5     playlist;
```

```
1 SELECT
2     tophit,
3     play
4 FROM
5     toplist;
```

*Note: Why did I use the column names and why didn't I use `SELECT *` instead? Find out from my [SQL best practices \(https://data36.com/sql-best-practices-data-analysts/\)](https://data36.com/sql-best-practices-data-analysts/) article!*

Ah, cool, some nice songs from some nice artists, that people have played a lot already...

***An important note: when you close SQL Workbench, these temporary data tables will disappear. If you do so, you have to `CREATE` them again and then `INSERT` data into them! If***

you want to  [data36](https://data36.com) list type and  
run this command in SQL  
(<https://data36.com>)  
**Workbench:**

```
1 | COMMIT;
```

## SQL JOIN – the basics

Let's perform our first SQL JOIN!

```
1 | SELECT *  
2 | FROM toplist  
3 | JOIN playlist  
4 | ON tophit = song;
```

First of all, take a look at the results!


We have four columns, two plus two from the two tables. Makes sense, right?

The values in the song and tophit columns are the same. In fact, that was the column that we have joined on, so this is not a big surprise. But hey, we have just merged two tables!

Let's see what has happened code-wise:

- `SELECT *` –» We want to select every column...



- FROM `toplist`  `data360` from the `toplist` table.  
(<https://data36.com>).
- **JOIN playlist** –» ...but we also want to merge the `playlist` table to the `toplist` table...
- **ON tophit = song;** –» ...and we want to connect those lines, where the value of the `tophit` column is matching with the value of the `song` column.

Not that complicated... Yet. 😊

Now there is one important question here...

If you haven't realized it yet: there is one song (*Zara Larsson: Lush Life*) that exists in the `playlist` table, but not in the `toplist` table. And there is another one (*Despacito, 519689490*) that exists in the `toplist` table, but not in the `playlist` table. I did this on purpose, because I wanted to demonstrate what happens when you have to deal with partially missing data, which actually happens quite often in real-life data science projects.

To understand that, take a look at this Venn-diagram. It shows what happens with the data after JOIN-ing the two SQL tables.

When you use the *default* JOIN, your query will manage your data like this:

### *INNER SQL JOIN*

As you can see: **the default SQL JOIN keeps only the data that occurs in both tables.** So Despacito and Zara Larsson were removed.

This method is called INNER JOIN but since this is what we use most often, in SQL it has been implemented as the default behaviour of the JOIN keyword.

## **SQL JOIN – same query, better syntax**

Remember that in the [SQL best practices article](https://data36.com/sql-best-practices) (<https://data36.com/sql-best-practices-data-analysts/>). I strongly recommended not to use \* in a SELECT statement... It's better to use the actual names of the columns instead. And it's even more important when you use JOINS in your queries!

Instead of this query...



```
1 SELECT *
2 FROM topl
3 JOIN playlist(https://data36.com)
4 ON tophit = song;
```

...I recommend using this one:

```
1 SELECT
2     tophit,
3     play,
4     artist,
5     song
6 FROM toplist
7 JOIN playlist
8 ON tophit = song;
```

And to be honest, this is still not the most bulletproof syntax.

The problem is that when you look at this code, you can't instantly see what columns belong to which table. Thus I like to add the table names also to the column names. It works with dot notation and it's really simple.

For instance:

If the `tophit` column is in the `toplist` table, then instead of `tophit`, I'll write: `toplist.tophit`.

If you apply this to all column names in the above query:

```
1 SELECT
2     toplist.tophit,
3     toplist.play,
4     playlist.artist,
5     playlist.song
6 FROM toplist
7 JOIN playlist
8 ON toplist.tophit = playlist.song;
9
```

Much better. But one last small tweak!  
The *playlist.song* and the  
*toplist.tophit* columns are actually  
the same. We don't need both of  
them... so I'll remove one:

```
1 SELECT
2     toplist.tophit,
3     toplist.play,
4     playlist.artist
5 FROM toplist
6 JOIN playlist
7 ON toplist.tophit = playlist.song;
```

Now this is finally a pretty decent SQL  
JOIN.

## The Junior Data Scientist's First Month

A 100% practical online course.  
A 6-week simulation of being a  
junior data scientist at a true-  
to-life startup.

 *“Solving data 36, getting real experience – just like in a real data science job.”*  
(<https://data36.com>)

Learn more...  
(<https://data36.com/the-junior-data-scientists-first-month-online-course/>).

## FULL JOIN

But you might ask: *“What should we do to keep Despacito and Zara Larsson in the data set, even though they don’t show up in both data tables?”* And I’d answer: great question!

Get back to our Venn-diagram... This time, this is what we want to achieve:

*SQL FULL JOIN*

This version of the SQL JOINS is called **FULL JOIN**, and to execute it, you should change only one tiny thing in our previous query: add the FULL keyword before the JOIN keyword.



```
1 SELECT data36
2   toplist
3   toplist(https://data36.com),
4   playlist.artist,
5   playlist.song
6 FROM toplist
7 FULL JOIN playlist
8 ON toplist.tophit = playlist.song;
```

Boom! The magic has happened!

As you can see, Zara Larsson and Despacito are there, but those fields that don't have data, stay empty. These empty fields are called NULL values in SQL. I have already mentioned briefly NULL and its importance ([in the SQL functions article \(https://data36.com/sql-functions-beginners-tutorial-ep3/\)](https://data36.com/sql-functions-beginners-tutorial-ep3/)) but I'll get back to that in more detail later!

However, now you know: the fact that a given value doesn't exist in both data tables, doesn't mean that you can't JOIN them. You can with a FULL JOIN, but if you do so, the cells of the missing values will stay empty.

## LEFT JOIN AND RIGHT JOIN

And this brings us to the next question.  [data36](https://data36.com) (https://data36.com).

What if we want to apply the concept of FULL JOIN — but keeping the missing values only from one of the SQL tables?

Back to the Venn-diagram! Either this...

*LEFT JOIN*

... or this:

*RIGHT JOIN*

Before I reveal the the “big secret,” I’d like to emphasize that **this problem occurs quite often in real data projects too.**

E.g. let’s say, you are running an A/B test (<https://data36.com/ab-test-like-a-data-scientist/>) in which you have

TABLE #1


<u>USER</u>	<u>BUCKET</u>
user1	A
user2	B
user3	A
user4	B

TABLE #2

<u>USER</u>	<u>TIMESTAMP</u>
user1	2018-02-01 19:00:00.000
user2	2018-02-01 19:32:11.000
user4	2018-02-01 19:44:54.000
user5	2018-02-01 19:48:23.000
user1	2018-02-01 19:59:01.000
user5	2018-02-01 20:01:10.000
user2	2018-02-01 20:04:32.000
user4	2018-02-01 20:09:54.000
user1	2018-02-01 20:12:32.000

You want to include only 80% of your audience (e.g. user1, user2, user3, user4 — but *not* user5) in your A/B test, and you put these selected users into a table with the info about which bucket (A or B) they belong to.



You have another  data36 at works as a usage log (see more here: [data collection \(https://data36.com/data-collection/\)](https://data36.com/data-collection)) and collects the feature usage for *all* users (user1, user2, user3, user4 and **also** user5).

To evaluate your A/B test, you have to combine the two tables. You want to keep all users from TABLE #1 (even if they didn't use the feature, ergo didn't show up in the other table at all — like user3), but you don't want to keep the users who showed up only in the second table (used the feature, but not part of the A/B test — like user5).

What do you do?

This:

*another LEFT JOIN example*

This is called LEFT JOIN. And if you want to perform a LEFT JOIN, you simply have to add a LEFT keyword to your JOIN keyword.

Getting back to our playlist + toplist data sets, try something like this:



```
1 SELECT data36
2     toplist
3     toplist(https://data36.com),
4     playlist.artist
5 FROM toplist
6 LEFT JOIN playlist
7 ON toplist.tophit = playlist.song;
```

It keeps every line from the `toplist` table (that's the *LEFT* table) even if it doesn't exist in the `playlist` table (which is the *RIGHT* table). But it keeps only those lines from the `playlist` table that exist in the `toplist` table too. Good!

If you want to execute the opposite, you should do a `RIGHT JOIN` instead:

```
1 SELECT
2     toplist.tophit,
3     toplist.play,
4     playlist.artist
5 FROM toplist
6 RIGHT JOIN playlist
7 ON toplist.tophit = playlist.song;
```

Awesome!

## Test yourself #1

You now know everything you have to know at this point about SQL JOIN!  
But the best way of learning is

practicing!  data36ysis, that  
you should perform on our playlist  
(<https://data36.com>)  
and toplist tables:

**Given the information in the  
*playlist* and *toplist* data tables:**

**How many plays does each artist  
have in total?**

.  
. .  
.

Here's my solution:

```
1 SELECT
2     playlist.artist,
3     SUM(toplist.play)
4 FROM playlist
5 FULL JOIN toplist
6     ON playlist.song = toplist.tophit
7 GROUP BY artist;
```

And a short explanation:

- SELECT –» We select...
- playlist.artist, –» the artists from the playlist table...
- SUM(toplist.play) –» and we sum the number of plays from the toplist table...
- FROM playlist –» we actually specify that we will use the playlist table...
- FULL JOIN toplist –» and we also want to merge the toplist table (using the empty fields too)...

- ON playlist.tophit => use JOIN matches those lines where the song and tophit columns have the same data...
- GROUP BY artist; -> and GROUP BY refers to the SUM function, above - we want to see the sums by artist.

## Test yourself #2

One more test, before I let you go!

**Print the top 5 ABBA songs ordered by number of plays!**


.  
. .

And the solution is:

```
1 SELECT
2     playlist.artist,
3     playlist.song,
4     toplist.play
5 FROM playlist
6 FULL JOIN toplist
7     ON playlist.song = toplist.tophit
8 WHERE playlist.artist = 'ABBA'
9 ORDER BY toplist.play DESC;
10
```

Well, nothing new here. 😊 (But if you need an explanation, just let me know and I'll give you one!

## Conclusion

SQL JOIN is  data36 ant and you will use it quite often as a data analyst or scientist. This article has given you a solid base of knowledge. Further down the road you will meet even more advanced applications, but using what you have learned from this article – combined with what you have learned from the previous ones (<https://data36.com/learn-sql-for-data-analysis-from-scratch/>). – will cover most cases for now.

And there is only one more article left from my SQL for Data Analysis – Tutorial for Beginners (<https://data36.com/learn-sql-data-analysis-scratch-redirect/>) series. In that one, I'll introduce some intermediate SQL concepts:

- how to write a query-in-a-query — aka. *subquery*
- SQL HAVING and
- SQL CASE!

**Continue here: Advanced SQL concepts** (<https://data36.com/sql-data-analysis-advanced-tutorial-ep6/>).

- If you want to learn more about how to become a data scientist, take my 50-minute video course: How to Become a Data Scientist. (<https://data36.com/how-to-become-a-data-scientist/>). (It's free!)

- Also check  data36 week online course: The Junior Data Scientist's First Month video course.  
(<https://data36.com>),  
(<https://data36.com/jds/>).

*Cheers,*  
***Tomi Mester***

---

[Privacy Policy](https://Data36.Com/Privacy-Policy/) ([Https://Data36.Com/Privacy-Policy/](https://Data36.Com/Privacy-Policy/))

[Terms Of Use](https://Courses.Data36.Com/P/Terms) ([Https://Courses.Data36.Com/P/Terms](https://Courses.Data36.Com/P/Terms))

Data36.Com By Tomi Mester | © All Rights Reserved

[Data36.Com](https://Data36.Com) ([Https://Data36.Com](https://Data36.Com))