What Is the Difference Between WHERE and ON in SQL JOINs?

Kateryna Koidan
Kateryna is a data science writer from Kyiv, Ukraine. She worked for BNP Paribas, the leading European banking group, as an internal auditor for more than 6 years. More recently, she decided to pursue only the favorite part of her job—data analysis. Now she is continuing her self-education with deep-learning courses, enjoys coding for data analysis and visualization projects, and writes on the topics of data science and artificial intelligence. Kateryna is also a proud mother of two lovely toddlers, who make her life full of fun.

Tags: SQL LEARN SQL SQL JOINS WHERE

When you join tables in SQL, you may have conditions in an ON clause and in a WHERE clause. Many get confused by the difference between them. In this article, we will discuss this topic by first reminding you the purpose of the ON and WHERE clauses then by demonstrating with examples which types of conditions should be in each of these clauses.

Both the `ON` clause and the `WHERE` clause can specify conditions. But are there any differences between them? If so, where should you specify what conditions in your SQL query? Let’s find out together!

Improve your SQL JOIN skills with our special interactive course, SQL JOINs!

ON vs. WHERE Conditions

The purpose of the **ON clause is to specify the join conditions**, that is, to define how the tables should be joined. Specifically, you define how the records should be matched.

In contrast, the purpose of the `WHERE` **clause is to specify the filtering conditions**, that is, to define which rows should be kept in the result set.

Let's look at an example to understand the difference. We have the following two tables that (1) list the users (the table `users`) of our rental website and (2) list the houses (the table `houses`) available for rent.

users

id	name	registration_date
11	Jane Stewart	2020-11-30
12	Mary Cooper	2015-06-12
13	John Watson	2015-01-31
14	Christian Wood	2018-03-03
15	William Grey	2021-05-12
16	Brandon Evans	2018-05-08
17	Isabella Gonsalez	2020-12-12
18	Diana Taylor	2020-06-30
19	Luke Wilson	2019-11-17
20	Michael Lee	2020-02-15

houses

id	address	city	owner_id	bedrooms
101	Brook Street, 5	Cardiff	12	4
102	Richmond Street, 1	Cardiff	12	1

houses

103	Cromwell Road, 23	Liverpool	13	2
104	Hastings Road, 109	York	15	2
105	Bedford Road, 2	Bristol	16	1
106	Queen Street, 45	Bristol	16	3
107	Mayfield Road, 34	Cardiff	12	3

```
SELECT u.id, u.name, u.registration_date, h.address, h.city
FROM users u
JOIN houses h
ON u.id = h.owner_id
WHERE u.registration_date < '2020-01-01';
```

Note that we have conditions in both the ON clause and the `WHERE` clause:

- With the `ON` condition, we specify that the tables be joined by matching the id column in the users table and the `owner_id` column in the houses
- With the `WHERE` condition, we filter the result set by keeping only the users who registered before January 1, 2020.

Thus, we have used the `ON` and `WHERE` conditions according to their purpose, resulting in a **clear and readable SQL query**.

Here is the result set:

id	name	registration_date	address	city
12	Mary Cooper	2015-06-12	Brook Street, 5	Cardiff
12	Mary Cooper	2015-06-12	Richmond Street, 1	Cardiff
13	John Watson	2015-01-31	Cromwell Road, 23	Liverpool
16	Brandon Evans	2018-05-08	Bedford Road, 2	Bristol

id	name	registration_date	address	city
16	Brandon Evans	2018-05-08	Queen Street, 45	Bristol
12	Mary Cooper	2015-06-12	Mayfield Road, 34	Cardiff

Not sure how the `JOIN` works in our SQL query? Practice joining tables with this interactive SQL JOINS course.

ON and WHERE Conditions in INNER JOINs

In the above example, we can see how the `ON` and `WHERE` conditions are used according to their respective purpose and common practice.

However, it is useful to know that, for `(INNER) JOINs`, you can specify both the `JOIN`condition and the filtering condition with an ON clause. For example, we can get the same result as the above with the following SQL query:

```
SELECT u.id, u.name, u.registration_date, h.address, h.city
FROM users u
JOIN houses h
ON u.id = h.owner_id AND u.registration_date < '2020-01-01';
```

This query is executed in the same way. That said, I do not recommend mixing the join condition and the filtering condition in the same clause. If you compare the two queries, you see the first one is more readable:

- It's easier to follow the first query: first, you join the tables by a certain condition, then you filter the result by a different condition.
- The intent of the entire query is clearer to the outside reader when the conditions are separated by following the rules.

ON and WHERE Conditions in OUTER JOINs

When it comes to `OUTER JOINs` (i.e., `LEFT JOIN`, `RIGHT JOIN`, and `FULL JOIN`), it is crucial to use the `ON` and `WHERE` conditions the way they are intended. Otherwise, you'll get wrong results. Let's see with an example.

Again, we want to get the list of users who registered before Jan 1st, 2020, along with their respective houses. This time, however, we want to keep all users, including those that do not have registered houses on our rental website. Thus, we are going to do a `LEFT JOIN` instead of a `JOIN` (i.e., an `INNER JOIN)`.

We will see whether there are any differences between specifying the filtering condition in the ON clause and specifying it in the `WHERE` clause. If we follow the rules and use the conditions as intended, we have the following query:

```
SELECT u.id, u.name, u.registration_date, h.address, h.city
FROM users u
LEFT JOIN houses h
ON u.id = h.owner_id
WHERE u.registration_date < '2020-01-01';
```

id	name	registration_date	address	city

id	name	registration_date	address	city
12	Mary Cooper	2015-06-12	Brook Street, 5	Cardiff
12	Mary Cooper	2015-06-12	Richmond Street, 1	Cardiff
13	John Watson	2015-01-31	Cromwell Road, 23	Liverpool
16	Brandon Evans	2018-05-08	Bedford Road, 2	Bristol
16	Brandon Evans	2018-05-08	Queen Street, 45	Bristol
12	Mary Cooper	2015-06-12	Mayfield Road, 34	Cardiff
19	Luke Wilson	2019-11-17	NULL	NULL
14	Christian Wood	2018-03-03	NULL	NULL

The result looks good. We got all of the users we got in our initial example. In addition, we have two more users who do not have corresponding houses on our website but were included in the result set because of the `LEFT JOIN`. Note that both registered before January 1, 2020 as specified in our filtering condition.

Do we get the same result if we mix the join condition and the filtering condition in the `ON` clause? Let's find out:

```
SELECT u.id, u.name, u.registration_date, h.address, h.city
FROM users u
LEFT JOIN houses h
ON u.id = h.owner_id AND u.registration_date < '2020-01-01';
```

id	name	registration_date	address	city
11	Jane Stewart	2020-11-30	NULL	NULL
12	Mary Cooper	2015-06-12	Mayfield Road, 34	Cardiff
12	Mary Cooper	2015-06-12	Richmond Street, 1	Cardiff
12	Mary Cooper	2015-06-12	Brook Street, 5	Cardiff

id	name	registration_date	address	city
13	John Watson	2015-01-31	Cromwell Road, 23	Liverpool
14	Christian Wood	2018-03-03	NULL	NULL
15	William Grey	2021-05-12	NULL	NULL
16	Brandon Evans	2018-05-08	Queen Street, 45	Bristol
16	Brandon Evans	2018-05-08	Bedford Road, 2	Bristol
17	Isabella Gonsalez	2020-12-12	NULL	NULL
18	Diana Taylor	2020-06-30	NULL	NULL
19	Luke Wilson	2019-11-17	NULL	NULL
20	Michael Lee	2020-02-15	NULL	NULL

As you can see, the results are different. We have all the users included, even the ones who registered in 2020 or 2021. This is because the `LEFT JOIN` keeps all the records from the left table even when the `ON` logic fails. So, in this example, specifying the filtering condition in the `ON` clause doesn't work for us. To get the correct result, we need to specify the conditions as intended.

Interestingly, there are situations in which the `WHERE` condition can "cancel" the intent of an `OUTER JOIN`. As an example, let's say we want to list all users with their corresponding houses but only if the houses have 3 or more bedrooms.

Since we want to keep all users, we will use an `OUTER JOIN`, specifically a `LEFT JOIN`. Our requirement for the number of bedrooms is clearly a filtering condition. So, we'll include it in the `WHERE` clause. Here's our SQL query with conditions specified as intended:

```
SELECT u.id, u.name, h.address, h.city, h.bedrooms
FROM users u
LEFT JOIN houses h
ON u.id = h.owner_id
```

```
WHERE h.bedrooms > 2;
```

Doesn't seem right, does it? The result looks as if we used an `INNER JOIN` rather than a `LEFT JOIN`. Users without houses are not included in the resulting table, because they have `NULL` in the bedrooms column when the tables are joined. Since the `NULL` values are considered less than 0, the corresponding rows are removed when we apply the filtering condition – the number of bedrooms greater than 2.

There are two possible solutions to this problem:

- Add another filtering condition to the `WHERE` clause, like bedrooms is `NULL`:

```
SELECT u.id, u.name, h.address, h.city, h.bedrooms
FROM users u
LEFT JOIN houses h
ON u.id = h.owner_id
WHERE h.bedrooms > 2 OR h.bedrooms is NULL;
```

- Moving the filtering condition to the `ON` clause:

```
SELECT u.id, u.name, h.address, h.city, h.bedrooms
FROM users u
LEFT JOIN houses h
ON u.id = h.owner_id AND h.bedrooms > 2;
```

Either of these queries gives us the following result:

id	name	address	city	bedrooms
11	Jane Stewart	NULL	NULL	NULL
12	Mary Cooper	Mayfield Road, 34	Cardiff	3
12	Mary Cooper	Brook Street, 5	Cardiff	4

id	name	address	city	bedrooms
13	John Watson	Cromwell Road, 23	Liverpool	NULL
14	Christian Wood	NULL	NULL	NULL
15	William Grey	NULL	NULL	NULL
16	Brandon Evans	Queen Street, 45	Bristol	3
17	Isabella Gonsalez	NULL	NULL	NULL
18	Diana Taylor	NULL	NULL	NULL
19	Luke Wilson	NULL	NULL	NULL
20	Michael Lee	NULL	NULL	NULL

Now you know! In `OUTER JOINs`, it does make a difference how we specify the conditions.

Let's Practice JOINs in SQL!

SQL `JOINs` are not too difficult to understand. However, as you could see from the examples in this article, there are nuances that should be considered when joining tables and writing join conditions in SQL.

> Learn about different types of JOINs. Check out our interactive SQL JOINs course.

If you really want to MASTER SQL JOINS, practicing with real-world data sets is a key success factor. I recommend starting with the interactive SQL JOINS COURSE – it includes 93 coding challenges covering the most common types of joins like `JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `FULL JOIN`, and even self-joins and non-equi joins. After taking this course, you'll know how to join multiple tables, how to JOIN TABLES WITHOUT A COMMON COLUMN, and how to correctly filter data with different kinds of `JOINs`.

For those wanting experience with even more `SQL JOIN` use cases, I recommend taking the SQL PRACTICE track. It includes five interactive courses with 600+ coding challenges, covering not only the basics of `SQL JOINs` but also how to filter the result set with a `WHERE` clause, how to aggregate data with `GROUP BY` and `HAVING`, and how to use subqueries including correlated subqueries. You're going to have lots of fun!