2022-01-27T18:00:00+02:00 | Kateryna Koidan

How Do You Get Rid of Duplicates in an SQL JOIN?

Do you have unwanted duplicates from your SQL JOIN query? In this article, I'll discuss the possible reasons for getting duplicates after joining tables in SQL and show how to fix a query depending on the reason behind the duplicates.

Data analysts with little experience in SQL JOINs often encounter unwanted duplicates in the result set. It's challenging for beginners to identify the reason behind these duplicates in JOINs.

In this article, I'll discuss the most common issues leading to duplicates in SQL JOIN outputs. I'll also show possible solutions to these common issues.

If you want to get a comprehensive understanding of SQL JOINs, check out this interactive course that provides coding challenges for all JOIN types.

Level up your SQL with our interactive SQL JOINs course. Its 93 hands-on exercises let you learn by doing.

Let's start with a very brief overview of SQL JOINs.

SQL JOINs Overview

`JOIN` is an SQL construct for requesting information from two or more tables within the same query.

For example, let's say you have a list of the top 100 movies from the 20th century, and you want to subset it to the movies made by currently living directors. In your **movies** table, you don't have detailed information on the movie directors, just their IDs. But you do have a separate **directors** table, with the ID, the full name, the birth year, and the death year (if applicable) of each director.

In your query, you can join two tables by the director's ID to get a list of movies made by currently living directors:

```
SELECT movies.title, directors.full_name
FROM movies
JOIN directors
ON movies.director_id = directors.id
WHERE directors.death_year IS NULL;
```

As you can see, we specify the tables we want to join in the `FROM` and `JOIN` clauses. Then in the ON clause, we specify the columns from each table to be used for joining these tables. If you are new to SQL JOINs, check out this introductory guide. Here's also an SQL JOIN cheat sheet with syntax and examples of different JOINs.

The SQL JOIN is a great tool that provides a variety of options beyond the simple join of two tables. If you are not familiar with SQL JOIN types, read this article that explains them with illustrations and examples. Depending on your use case, you can choose `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, and `FULL JOIN`. You may even need to join tables without a common column or join more than two tables.

Now, let's see how these different JOINs may result in unwanted duplicates.

What Are the Reasons for Duplicates in SQL JOINs?

There are many possible reasons for getting duplicates in the result of your SQL JOIN query. I'll go through the top 5 reasons; for each one, I'll show a sample query with the problem and a corrected query to get a result without duplicates.

Let's start by briefly reviewing the data to be used for our examples. Imagine we run a real estate agency that sells houses somewhere in the United States. We have tables with **agents**, **customers**, and **sales**. See below for what data is stored in each table.

agents

id	first_name	last_name	experience_years
1	Kate	White	5
2	Melissa	Brown	2
3	Alexandr	McGregor	3
4	Sophia	Scott	3
5	Steven	Black	1
6	Maria	Scott	1

customers

id	first_name	last_name	email
11	Xaviera	Lopez	xaviera111111@gmail.com
12	Gabriel	Cumberly	gabriel111111@gmail.com
13	Elisabeth	Stevens	elisabeth111111@gmail.com
14	Oprah	Winfrey	oprah111111@gmail.com
15	Ivan	Lee	ivan111111@gmail.com

sales

id	house_id	date	agent_first_name	agent_last_name	custome
101	1012	2021-11-03	Kate	White	14
102	2134	2021-12-06	Sophia	Scott	12
103	1015	2021-12-10	Maria	Scott	13
104	2013	2021-12-12	Alexandr	McGregor	15
105	2112	2021-12-12	Alexandr	McGregor	15
106	1010	2022-01-10	Steven	Black	11

Without any further delay, let's move to our examples.

1. Missing ON Condition

Beginners unfamiliar with SQL JOINs often simply list the tables in `FROM` without specifying the `JOIN` condition at all when trying to combine without specifying the `JOIN` condition at all when trying to combine information from two or more tables. This is valid syntax, so you do not get any error messages. But the result is a cross join with all rows from one table combined with all rows from another table.

For example, suppose we want to get information on the customer who bought a particular house (ID #2134). If we use the following query:

```
SELECT house_id, first_name, last_name, email
FROM sales, customers
WHERE house_id = 2134;
```

This is the result we get:

house_id	first_name	last_name	email
2134	Xaviera	Lopez	xaviera111111@gmail.com
2134	Gabriel	Cumberly	gabriel111111@gmail.com
2134	Elisabeth	Stevens	elisabeth111111@gmail.com
2134	Oprah	Winfrey	oprah111111@gmail.com
2134	Ivan	Lee	ivan111111@gmail.com

Instead of one record with the customer we want, we have all our customers listed in the result set.

To fix the query, you need an explicit `JOIN` syntax. The tables to be combined are specified in `FROM` and `JOIN`, and the join condition is specified in the `ON` clause:

```
SELECT s.house_id, c.first_name, c.last_name, c.email
FROM sales s
JOIN customers c
ON s.customer_id = c.id
WHERE s.house_id = 2134;
```

Here, we specify the customer ID from the **`sales`** table to match the customer ID from the **`customers`** table. This gives us the desired result:

house_id	first_name	last_name	email
2134	Gabriel	Cumberly	gabriel111111@gmail.com

You could specify the join condition in the `WHERE` clause to get the same result. But that is against the intended use of the WHERE clause. Also, there are additional benefits from using the `JOIN` syntax rather than listing the tables in `FROM`. Check out this article to understand why the listing the tables in FROM. Check out this article to understand why the JOIN syntax is preferred.

2. Using an Incomplete ON Condition

Unwanted rows in the result set may come from incomplete ON conditions. In some cases, you need to join tables by multiple columns. In these situations, if you use only one pair of columns, it results in duplicate rows.

Let's say we want to see the experience level of the real estate agent for every house sold. If we start by joining the **sales** and **agents** tables by the agent's last name:

```
SELECT s.house_id, a.first_name, a.last_name, a.experience_years
FROM sales s
JOIN agents a
ON s.agent_last_name = a.last_name
ORDER BY s.house_id;
```

this is what you get:

house_id	first_name	last_name	experience_years
1010	Steven	Black	1
1012	Kate	White	5
1015	Maria	Scott	1
1015	Sophia	Scott	3
2013	Alexandr	McGregor	3
2112	Alexandr	McGregor	3
2134	Maria	Scott	1
2134	Sophia	Scott	3

That didn't work well. We have two different agents with the last name Scott: Maria and Sophia. As a result, houses #1015 and #2134 are each included twice with different agents.

To fix this query, we need to join the **sales** and **agents** tables using two pairs of columns, corresponding to the last name and the first name of the agent:

```
SELECT s.house_id, a.first_name, a.last_name, a.experience_years
FROM sales s
FROM sales s
JOIN agents a
ON s.agent_last_name = a.last_name
   AND s.agent_first_name = a.first_name
ORDER BY s.house_id;
```

And here is the result we were looking for.

house_id	first_name	last_name	experience_years
1010	Steven	Black	1
1012	Kate	White	5
1015	Maria	Scott	1
2013	Alexandr	McGregor	3
2112	Alexandr	McGregor	3
2134	Sophia	Scott	3

While JOIN is one of the basic tools in SQL, you need to be aware of the many different nuances to join tables effectively. I recommend practicing SQL JOINs with this interactive course that covers a variety of joining scenarios with 93 coding challenges.

3. Selecting a Subset of Columns

In some cases, the records in the result set are not duplicates but appear as if they are because the selected subset of columns doesn't show all differences between records.

For example, imagine we want to see the dates each real estate agent sold a house. If we use the following query:

```
SELECT a.first_name, a.last_name, s.date
FROM agents a
JOIN sales s

ON s.agent_last_name = a.last_name
   AND s.agent_first_name = a.first_name;
```

it produces the following result:

first_name	last_name	date
Kate	White	2021-11-03
Sophia	Scott	2021-12-06
Alexandr	McGregor	2021-12-12
Alexandr	McGregor	2021-12-12

first_name	last_name	date
Maria	Scott	2021-12-10
Steven	Black	2022-01-10

The result set includes two records with Alexandr McGregor that appear identical. However, if you add house ID to the `SELECT` statement, you see these two records correspond to the sale of two different houses on the same day.

If you are not interested in this additional information and want to have only one row displayed here, use `DISTINCT`:

```
SELECT DISTINCT a.first_name, a.last_name, s.date
FROM agents a
JOIN sales s
ON s.agent_last_name = a.last_name
   AND s.agent_first_name = a.first_name;
```

Now, the result is:

first_name	last_name	date
Kate	White	2021-11-03
Sophia	Scott	2021-12-06
Alexandr	McGregor	2021-12-12
Maria	Scott	2021-12-10
Steven	Black	2022-01-10

LearnSQL.com lets you learn SQL by writing SQL code on your own. You build your SQL skills gradually. Each new concept is reinforced by an interactive exercise. By actually writing SQL code, you build your confidence.

4. Listing Matching Rows Only

A similar problem may occur if you want to list only the rows from one table but there are several matching records in the other table. You end up with unwanted duplicates in your result set.

For instance, say we want to list all customers who bought houses via our agency. If we use the following query:

```
SELECT c.first_name, c.last_name, c.email
FROM customers c
JOIN sales s
ON c.id = s.customer_id;
```

here is the result:

first_name	last_name	email
Gabriel	Cumberly	gabriel111111@gmail.com
Elisabeth	Stevens	elisabeth111111@gmail.com
Xaviera	Lopez	xaviera111111@gmail.com
Oprah	Winfrey	oprah111111@gmail.com
Ivan	Lee	ivan111111@gmail.com
Ivan	Lee	ivan111111@gmail.com

As you see, the resulting table includes Ivan Lee twice. This is because he bought two houses and there are two corresponding records in the **`sales`** table. One possible solution is to use `DISTINCT` as in the previous example. An even better solution is to avoid using SQL JOIN at all by filtering the result set using the `EXISTS` keyword:

```
SELECT c.first_name, c.last_name, c.email
FROM customers c
WHERE EXISTS (SELECT customer_id FROM sales);
```

Now, the result is:

first_name	last_name	email
Gabriel	Cumberly	gabriel111111@gmail.com
Elisabeth	Stevens	elisabeth111111@gmail.com
Xaviera	Lopez	xaviera111111@gmail.com
Oprah	Winfrey	oprah111111@gmail.com
Ivan	Lee	ivan111111@gmail.com

This gives you the desired output and also makes the intention of your query clearer.

5. Using Self Joins

Finally, unwanted duplicates in JOINs often result from an incorrect specification of joining conditions in self joins – that is, when a table is joined with itself.

Let’s say we want our agents to form pairs for our next training. Obviously, we don’t want any agent to be paired with himself/herself. So, we might specify the `ON` condition `a1.id <> a2.id`:

```
SELECT
    a1.first_name as agent1_first_name,
    a1.last_name as agent1_last_name,
    a1.experience_years as agent1_experience,
    a2.first_name as agent2_first_name,
    a2.last_name as agent2_last_name,
    a2.experience_years as agent2_experience
FROM agents a1
JOIN agents a2
ON a1.id <> a2.id
ORDER BY a1.id;
```

However, this query outputs each pair twice. For example, in the first row of the table below, Kate White is considered Agent 1, and Maria Scott is considered Agent 2. But closer to the end of the table, you get the same pair of agents but with Maria Scott as Agent 1 and Kate White as Agent 2.

agent1_first_name	agent1_last_name	agent1_experience	agent2_first_na
Kate	White	5	Maria
Kate	White	5	Steven
Kate	White	5	Melissa
Kate	White	5	Sophia
Kate	White	5	Alexandr
Melissa	Brown	2	Steven
Melissa	Brown	2	Sophia
Melissa	Brown	2	Maria
Melissa	Brown	2	Alexandr
Melissa	Brown	2	Kate
Alexandr	McGregor	3	Maria
Alexandr	McGregor	3	Melissa
Alexandr	McGregor	3	Sophia
Alexandr	McGregor	3	Kate
Alexandr	McGregor	3	Steven

agent1_first_name	agent1_last_name	agent1_experience	agent2_first_na
Alexandr	McGregor	3	Steven
Sophia	Scott	3	Maria
Sophia	Scott	3	Steven
Sophia	Scott	3	Alexandr
Sophia	Scott	3	Melissa
Sophia	Scott	3	Kate
Steven	Black	1	Sophia
Steven	Black	1	Alexandr
Steven	Black	1	Maria
Steven	Black	1	Melissa
Steven	Black	1	Kate
Maria	Scott	1	Kate
Maria	Scott	1	Alexandr
Maria	Scott	1	Sophia
Maria	Scott	1	Steven
Maria	Scott	1	Melissa

To solve this issue, you need to add an explicit condition to include each pair only once. One common solution is to specify the joining condition `a1.id < a2.id`. With this, you get the pair Kate White and Maria Scott but not vice versa. This is because Kate's ID (1) is a lower number than Maria's ID (6).

In practice, you may have some other conditions for pairing the agents. For instance, you may want to pair more experienced agents (3+ years) with less experienced ones (< 3 years). The corresponding filtering condition in `WHERE` solves the problem:

```
SELECT
    a1.first_name as agent1_first_name,

    a1.last_name as agent1_last_name,
    a1.experience_years as agent1_experience,
    a2.first_name as agent2_first_name,
    a2.last_name as agent2_last_name,
    a2.experience_years as agent2_experience
FROM agents a1
JOIN agents a2
ON a1.id <> a2.id
WHERE a1.experience_years>=3 AND a2.experience_years < 3
ORDER BY a1.id;
```

Here's the result:

Here's the result:

agent1_first_name	agent1_last_name	agent1_experience	agent2_first_na
Kate	White	5	Steven
Kate	White	5	Melissa
Kate	White	5	Maria
Alexandr	McGregor	3	Maria
Alexandr	McGregor	3	Steven
Alexandr	McGregor	3	Melissa
Sophia	Scott	3	Maria
Sophia	Scott	3	Steven
Sophia	Scott	3	Melissa

This result set looks much better and makes it easier to select three pairs, each consisting of an agent with more experience and another with less experience.

Let's Practice SQL JOINs!

Joining tables in SQL is not that difficult. But it does require a lot of practice. If you want to avoid pitfalls like unwanted duplicates in JOINs and missing records, follow this guide on practicing SQL JOINs.

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

If you have only basic experience with SQL and want to combine data more confidently from multiple tables, I recommend this SQL JOINs interactive course. It covers all major types of JOINs as well as joining a

table with itself, joining multiple tables in one query, and joining tables on non-key columns. Get more details about this course in this overview article.

Bonus. Here are the top 10 SQL JOIN interview questions with answers.

Thanks for reading, and happy learning!

Viewed using Just Read