Understanding the Use of NULL in SQL Three-Valued Logic

Maria Alcaraz
Former Freelance Database Developer, Mother of 4 children

Tags: NULL MASKING NULLS MATCHING NULLS NULL VALUES

NULLs are necessary in relational databases, and learning to use them is fundamental to SQL success. However, NULLs should also be handled with care, as we explain in this post.

In relational databases, we don't always have a value to put in a column. For example, suppose we have a table called "persons" that has "first_name", "last_name", "birth_date" and "marriage_date" columns. What value will we store in the "marriage_date" column for single persons? In that case, the column couldn't have a value because single persons are not married. We need a way indicate that we don't have a value for this column. Fortunately, we have such a thing: the NULL value, which is widely used in relational databases.

NULLs can be applied to any data type: integers, dates, VARCHARs, or any other column type. But we need to be careful when we create calculations or expressions containing one or more operators with a NULL value. Let's see why.

NULLs and Three-Valued Logic

The reason that NULLs can sometimes trip people up has to do with something called three-valued logic. While binary or Boolean logic has two values ("true" and "false"), three-valued logic (abbreviated as 3VL and also known as ternary logic) has an additional value — "unknown".

Let's illustrate 3VL with a simple scenario. Suppose we want to obtain the names of employees who make more than € 1,200 a month. The `employee` table looks like this:

First Name	Last Name	Salary	Bonus
John	Smith	1000	500
Mary	Smith	1000	1500
Peter	White	1800	NULL
Nick	Perry	1000	NULL

We try the following query:

```
SELECT * FROM employee WHERE bonus + salary > 1200;
```

The result is:

First Name	Last Name	Salary	Bonus
John	Smith	1000	500
Mary	Smith	1000	1500

We should see three records in the result set, but there are only two. Why is Peter White not included? The reason is related to the NULL value in the `bonus` column. In SQL, **every arithmetic operation that includes an operand with a NULL value returns a NULL result**.

So, with this in mind, look at how Peter White's record was evaluated:

```
1800 + NULL > 1200
```

In other words, Peter's salary (1,800) and his bonus (NULL) added up to NULL. We can reduce this condition to :

```
NULL > 1200
```

So is NULL greater than 1,200? Remember that NULL represents a nonexistent value, which means we don't have anything to compare against 1,200: we can't know if this statement is true or false. This is how three-valued logic works. When we have a NULL value in a condition, the result of this condition will be "unknown".

Coming to Grips with 3VL in SQL Queries

To understand how three-valued logic works, let's go through the process step-by-step. First, think about how records are filtered in the WHERE clause. Only records that evaluate to "true" in the WHERE clause are part of the query result set. Records evaluating to "false" or "unknown" are not part of the result. This is why Peter White's record was left out of the previous query results. His total salary evaluates to "Unknown" in the WHERE clause; *"1800 + NULL > 1200"* is "Unknown" because we cannot know what NULL is.

AND, OR and NOT tables are also important in three-valued logic, so we'll look at them individually and see how they compute. Let's start with the NOT table:

Value	NOT Result
True	False
False	True

Value	NOT Result
Unknown	Unknown

Next, let's see the AND table:

AND	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

Let's analyze why *"false" AND "unknown"* equates to "false". One "false" is enough to make the entire result "false" in an AND operation. This is true regardless of whether the second value is "true", "false", or "unknown".

Finally, we have the OR table:

OR	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

Let's try a query that uses the OR operator:

```
SELECT * FROM employee WHERE salary < 1500 OR bonus > 200
```

The result is the following:

First Name	Last Name	Salary	Bonus
John	Smith	1000	500

First Name	Last Name	Salary	Bonus
Mary	Smith	1000	1500
Nick	Perry	1000	NULL

Notice that one record containing a NULL value in the “bonus” column is shown in the result, but the other NULL record is not. The reason is the OR operator. Since the condition *salary < 1500* is true, it is not necessary to evaluate the condition *bonus > 200*.

Coping with the Unknown Variable

You can sometimes avoid using “unknown” values in WHERE conditions by converting NULL values to other values (like 0) using the COALESCE() function. Consider the previous examples. If you were to convert every NULL to “0” before the value was compared in the WHERE, you’d get different results than the ones shown above. However, this conversion may or may not be possible, depending on the semantics of the query. Still, let’s give it a try and see what happens.

```
SELECT * FROM employee WHERE coalesce(bonus,0) > 200 OR salary < 1
```

The query will only generate “true” or “false” results. In this case, two-valued logic will be applied.

Let’s suppose we receive an order to give a 5% salary raise to employees making less than €1,600 a month. How should we write this query? Let’s have a try:

```
UPDATE employee
SET salary = salary *1.05
WHERE salary + bonus <= 1600
```

Here are the results:

First Name	Last Name	Salary	Bonus
John	Smith	1050	500
Mary	Smith	1000	1500
Peter	White	1800	NULL
Nick	Perry	1000	NULL

Wait a minute – Nick's salary hasn't changed! That's not fair! Again, the "unknown" result of the *"salary + bonus <= 1600"* condition for Nick's record is causing a problem.

Our second try is better:

```
UPDATE employee
SET salary = salary *1.05
WHERE salary + coalesce(bonus,0) <= 1600
```

As we can see in the following query result, the salary of both records (John and Nick) has been modified.

First Name	Last Name	Salary	Bonus
John	Smith	1050	500
Mary	Smith	1000	1500
Peter	White	1800	NULL
Nick	Perry	1050	NULL

Try It Yourself!

There can be many different WHERE clauses that evaluate to “unknown” or “false”. Why not try some SQL queries out for yourself and see which return “true”, “false” or “unknown” values? LEARNSQL.COM can teach you all about the WHERE clause and its conditions. Try it out for free!