2021-03-17T17:00:00+02:00 | Gustavo du Mortier

What Is a NULL in SQL?

SQL uses NULLs as a special flag that signals the lack of a value for a field or a variable. NULLs should be used wisely so the database gives a faithful picture of the reality it represents.

Some people associate NULLs with zeros; others think that NULL means “none”. In SQL, neither of these are true. Simply put, an SQL NULL means that we don’t have a value for that particular field. Maybe we don’t know what the value will be and we need a placeholder, like when an order has been shipped to a customer but not delivered yet; in that case, the delivery date will be NULL until the order has arrived. Or maybe there’s an attribute that is true for some people and not others, e.g. not everyone has a middle name. NULL would work in that case, too.

Any field in a table can be defined to allow (or not allow) NULL values, regardless of the data type. Generally, you must define this condition when creating a table, as it can be problematic to change it once the database is in use and the table is filled with data.

How to Determine if a Field Should Allow Null Values

Let's look at an example table called `Springfield_Folks` that will store information about characters from *The Simpsons* TV series. This table will have three columns that store residents’ names, social security numbers, and the names of their spouses (if any).

Name	Social_Sec_No	Spouse_Name
Apu Nahasapeemapetilon	000-00-1000	Manjula Nahasapeemapetilon
Homer Simpson	000-00-5000	Marjorie Bouvier
Lenford Leonard	000-00-3000	
Nedward Flanders	000-00-4000	Maude Flanders

Name	Social_Sec_No	Spouse_Name
Waylon Smithers	000-00-8000	

The Name and `Social_Sec_No` fields do not allow NULL values because every US citizen has both pieces of information. (We'll pretend, for this example, that *The Simpsons* characters are real people.) However, since not all of the characters have a spouse, the `Spouse_Name` field should allow for NULL values.

Do you want to practice your SQL skills? Check out our SQL Practice Set!

Representation of NULL Values

When query results display table data, SQL NULL values can be seen in different ways, depending on the visualization tool used. For example, when data from a table is displayed through an Excel spreadsheet, NULL values commonly appear as blank spaces, as in the previous example.

The downside to Excel's approach is that you can't visually distinguish a NULL value from an empty value – which SQL considers entirely different things.

If the contents of a table are displayed through a database query tool – such as SQL Server Management Studio – the fields containing SQL NULL values will show the text "NULL". However, this is just a visual indication, as those fields do not actually contain the text that is displayed.

Inserting and Updating Data Containing NULL Values

When a NULL is inserted into a table using the SQL `INSERT` command with the `VALUES` clause, the word `NULL` can be used as a literal value. Here's an example:

```
INSERT INTO Springfield_Folks (
Name,
Social_Sec_No,
Spouse_Name
)
VALUES (
'Charles Montgomery Burns',
'000-00-9000',
NULL
)
```

It is important to note that if a column is omitted in an `INSERT` statement, SQL automatically assigns a NULL value to that column. Taking this into account, the example above could be rewritten as follows and achieve exactly the same result:

```
INSERT INTO Springfield_Folks (
Name,
Social_Sec_No
)
VALUES (
'Charles Montgomery Burns',
'000-00-9000'
)
```

In the same way as we've seen for the `INSERT` statement, the word `NULL` can be used as a literal when updating a field in a table:

```
UPDATE Springfield_Folks SET
Spouse_Name = NULL
WHERE Name = 'Nedward Flanders'
```

If an attempt is made to assign a NULL value using either `INSERT` or `UPDATE` to a field that does not allow NULLs, the database engine raises an error and cancels the operation. For example, in our **`Springfield_Folks`** table, we have specified that the Name field does not allow NULL values. Therefore, this `INSERT` ...

```
INSERT INTO Springfield_Folks (
Name,
Social_Sec_No,
Spouse_Name
)
VALUES (
NULL,
'000-00-1100',

'John Doe'
)
```

... will cause SQL to cancel the operation and display an error similar to this one:

```
Cannot insert the value NULL into column 'Name', table 'Springfield_Folks'; column
does not allow nulls. INSERT fails.
The statement has been terminated.
```

If, instead of inserting or updating the field with a literal value, we take the content from another field or table, we must consider the possibility that the field where we take the data from could have NULL values. For example:

```
INSERT INTO Springfield_Folks (
Name,
Social_Sec_No,
Spouse_Name
)
SELECT
Name,
Social_Sec_No,
Spouse_Name
FROM Springfield_Newcomers
```

In this example, we are inserting new rows in the **`Springfield_Folks`** table, taking them from another table named **`Springfield_Newcomers`**. If this table allows NULLs in the `Social_Sec_No` field, we could have trouble when inserting the data in our table. To prevent errors, we need to verify if a field has a NULL value. But first, let's see how NULL values behave in comparisons.

Comparisons with NULLs

As we've seen in the previous examples, the word `NULL` can be used as a literal value. But when doing so we must be cautious, since the use of `NULL` as a literal can be somewhat tricky.

To begin with, the `NULL` value has no type: it can be assigned to a field of the `VARCHAR`, `NUMERIC`, `DATE`, or whatever type. On the other hand, it is an absorbing value: a NULL value used in any operation also results in a NULL value.

And most confusing of all: any comparison with a NULL value yields false, since `NULL` is not equal to anything, not even to itself. Mindblowing, right? Let's clarify this idea by seeing how these comparisons behave when querying our **`Springfield_Folks`** table:

```
SELECT *
FROM Springfield_Folks
WHERE Spouse_Name = NULL
```

As you may have guessed, this query returns zero rows. You may think that, if there are no rows that meet the condition of `Spouse_Name = NULL`, then all rows must meet the opposite condition: `Spouse_Name <> NULL`. Let’s try it:

```
SELECT *
FROM Springfield_Folks
WHERE Spouse_Name <> NULL
```

Also zero rows! No, your database is not trying to trick you. It’s just the way NULL works. Whatever you compare to a NULL value will return false with **any** comparison, be it `<`, `<=`, `>`, `>=`, or `LIKE`.

So if comparing a NULL-valued field with anything always returns false, how can we tell if a field or variable is NULL? Simple: by asking if it `IS NULL`. Let’s make a few small tweaks to the `SELECT` from the above example to make it work:

```
SELECT *
FROM Springfield_Folks
WHERE Spouse_Name IS NULL
```

Name	Social_Sec_No	Spouse_Name
Lenford Leonard	000-00-3000	Manjula Nahasapeemapetilon
Waylon Smithers	000-00-8000	Marjorie Bouvier

This gives us the correct result. In the same way, we could ask for `IS NOT NULL` and it would also give us the correct result for our second query:

```
SELECT *
FROM Springfield_Folks
WHERE Spouse_Name IS NOT NULL
```

Name	Social_Sec_No	Spouse_Name
Apu Nahasapeemapetilon	000-00-1000	Manjula Nahasapeemapetilon
Homer Simpson	000-00-5000	Marjorie Bouvier
Nedward Flanders	000-00-4000	Maude Flanders

How to Filter Columns with NULL Values

When filtering data from a table using a nullable column in the `WHERE` condition, all rows with a NULL value in that column will be excluded, no matter what the `WHERE` condition is. The exception is when you ask specifically for IS NULL values, as we have already seen. Examine this `SELECT` to better understand how this works:

```
SELECT *
FROM Springfield_Folks
WHERE Spouse_Name LIKE 'M%'
```

You can try different literals to see that, no matter what value you compare to the column values, rows with a NULL value will be excluded.

Sorting data in nullable columns also has its secrets. If you need to sort a query by a column that allows NULL values, see this article on how SQL orders NULL values.

Giving Meaning to NULL Values

We can add a little bit more information by defining a column called `Marital_Status` in the `SELECT` list that will show the text "Married" when `Spouse_Name` is not NULL and "Single" when `Spouse_Name` is NULL:

```
SELECT *, 'Married' AS Marital_Status
FROM Springfield_Folks
WHERE Spouse_Name IS NOT NULL

SELECT *, 'Single' AS Marital_Status
FROM Springfield_Folks
WHERE Spouse_Name IS NULL
```

This way, we are helping the user better interpret the meaning of the NULL values in the `Spouse_Name` field.

Going a little further, we can combine the two previous examples into one query with the `UNION` clause:

```
SELECT *, 'Married' AS Marital_Status
FROM Springfield_Folks
WHERE Spouse_Name IS NOT NULL
UNION
SELECT *, 'Single' AS Marital_Status
FROM Springfield_Folks
```

```
WHERE Spouse_Name IS NULL
```

By doing this, we can get all the Springfield folks, either married or single, in one result set:

Name	Social_Sec_No	Spouse_Name	Marital_Statu
Apu Nahasapeemapetilon	000-00-1000	Manjula Nahasapeemapetilon	Married
Homer Simpson	000-00-5000	Marjorie Bouvier	Married
Nedward Flanders	000-00-4000	Maude Flanders	Married
Lenford Leonard	000-00-3000		Single
Waylon Smithers	000-00-8000		Single

Deleting Rows that Have Null Values

If we need to eliminate all rows that have a `NULL` value in a certain column, we can use the `IS NULL` criterion in the `WHERE` clause of the `DELETE` command. For example, if we want to delete all rows from our **`Springfield_Folks`** table that have a NULL value in `Spouse_Name`, we can execute this command:

```
DELETE Springfield_Folks
WHERE Spouse_Name IS NULL
```

Looking for online SQL practice in an interactive environment? Try our SQL Practice track!

The Usefulness of NULL

At this point, it should be clear what SQL NULL means in the design and use of information stored in databases. Null elements are very useful for representing real-life situations where a piece of information is absent. But they should be used with caution to avoid adversely affecting the integrity of the information contained in the database.

You can practice what you've learned so far about SQL NULL values (and learn a little more) in our interactive SQL Practice course and our