Nulls are a very useful but also very error-prone relational database feature. This talk is designed to help applications developers better manage their use of nulls.

https://momjian.us/presentations  Creative Commons Attribution License

Last updated: July 2022
Nulls in English

Null means “nothing”.

Null means “nothing”.
C-based languages use a NULL pointer to indicate a pointer that does not point to a value. Languages that don’t use pointers often use an “undefined” value for a similar purpose.
What do you place in a field that has no value?
For strings, a zero-length string is reasonable.
What about numerics? -1, -99, 0?
What about dates? 1900-01-01?
Why Use NULLs

The three meanings of NULL:

- Unknown values
- Inapplicable values
- Empty placeholders
The NULL Spouse Example

If `employee.spouse` is NULL, does it mean?

- The spouse’s name is unknown.
- The employee is not married and therefore has no spouse.
- The `employee.spouse` column was an unjoined column from an outer join.
NULLs Can Cause Their Own Problems

Don’t use NULLs in inappropriate situations.

https://www.flickr.com/photos/randar/
In their book A Guide to Sybase and SQL Server, David McGoveran and C. J. Date said:

> It is this writer’s opinion than NULLs, at least as currently defined and implemented in SQL, are far more trouble than they are worth and should be avoided; they display very strange and inconsistent behavior and can be a rich source of error and confusion. (Please note that these comments and criticisms apply to any system that supports SQL-style NULLs, not just to SQL Server specifically.)

...

In the rest of this book, I will be urging you not to use them, which may seem contradictory, but it is not. Think of a NULL as a drug; use it properly and it works for you, but abuse it and it can ruin everything. Your best policy is to avoid NULLs when you can and use them properly when you have to.

Joe Celko, SQL for Smarties: Advanced SQL Programming
Keep Your Eye on the Red (Text)

https://www.flickr.com/photos/alltheaces/
Explicit NULLs

test=> SELECT NULL;
?column?
--------

```
1 row
```

test=> \pset null (null)

```
1 row
```

test=> SELECT NULL;
?column?
--------

```
1 row
```

All queries in this presentation can be downloaded from https://momjian.us/main/writings/pgsql/nulls.sql.
CREATE TABLE nulltest (x INTEGER, y INTEGER);

INSERT INTO nulltest VALUES (1, NULL);

SELECT * FROM nulltest;

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(null)</td>
</tr>
</tbody>
</table>
INSERT INTO nulltest (x) VALUES (2);

SELECT * FROM nulltest;

| x | y  |
|---+----|
| 1 | (null) |
| 2 | (null) |
CREATE TABLE nulltest2 (x INTEGER NOT NULL, y INTEGER NOT NULL);

INSERT INTO nulltest2 VALUES (3, NULL);
ERROR: null value in column "y" violates not-null constraint
DETAIL: Failing row contains (3, null).

INSERT INTO nulltest2 (x) VALUES (4);
ERROR: null value in column "y" violates not-null constraint
DETAIL: Failing row contains (4, null).
The Non-Value of NULLs

SELECT NULL + 1;
?column?
----------
(null)

SELECT NULL || 'a';
?column?
----------
(null)

SELECT 'b' || NULL;
?column?
----------
(null)
CREATE TABLE inctest (x INTEGER);

INSERT INTO inctest VALUES (30), (40), (NULL);

SELECT x + 1 FROM inctest;

<table>
<thead>
<tr>
<th>?column?</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
</tr>
<tr>
<td>41</td>
</tr>
<tr>
<td>(null)</td>
</tr>
</tbody>
</table>
The Three-Valued Logic of NULLs

SELECT NULL = 1;
?column?
-----------
(null)

SELECT NULL = '';
?column?
-----------
(null)

SELECT NULL = NULL;
?column?
-----------
(null)

SELECT NULL < NULL + 1;
?column?
-----------
(null)

NULL represents unknown, not applicable, or unassigned. It has no data type, so comparing it to fixed values always returns NULL.
NULL Query Comparisons

SELECT 1
WHERE true;
  ?column?
----------
  1

SELECT 1
WHERE false;
  ?column?
----------

SELECT 1
WHERE NULL;
  ?column?
----------

WHERE only returns rows whose result is true, not false or NULL.
NULL Is Not False

SELECT true AND NULL;
?column?
----------
(null)

SELECT NOT NULL;
?column?
----------
(null)
SELECT * FROM inctest;
   x
-------
   30
   40
   (null)

SELECT * FROM inctest WHERE x >= 0;
   x
----
   30
   40

SELECT * FROM inctest WHERE x < 0;
   x
---

SELECT * FROM inctest WHERE x < 0 OR x >= 0;
   x
----
   30
   40
SELECT * FROM inctest WHERE x <> 10;

<table>
<thead>
<tr>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
</tr>
<tr>
<td>40</td>
</tr>
</tbody>
</table>

SELECT * FROM inctest WHERE x <> 10 OR x = 10;

<table>
<thead>
<tr>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
</tr>
<tr>
<td>40</td>
</tr>
</tbody>
</table>
SELECT 1 <> 2 AND 1 <> 3;

?column?

---------

t

SELECT 1 <> 2 AND 1 <> 3 AND 1 <> NULL;

?column?

---------

(null)
Subqueries With NULL

SELECT 'a' IN (SELECT NULL::text);
  ?column?
   -------
     (null)

SELECT 'a' NOT IN (SELECT NULL::text);
  ?column?
   -------
     (null)
Multi-Row Subqueries

```sql
SELECT 'a' IN (VALUES ('a'), (NULL));
?column?
---------
t

SELECT 'a' NOT IN (VALUES ('a'), (NULL));
?column?
---------
f

SELECT 'a' IN (VALUES ('b'), (NULL));
?column?
---------
(null)

SELECT 'a' NOT IN (VALUES ('b'), (NULL));
?column?
---------
(null)
```
SELECT 'a' = 'b' OR 'a' = NULL;
?column?
----------
(null)

SELECT 'a' <> 'b' AND 'a' <> NULL;
?column?
----------
(null)

NOT IN subqueries returning NULLs are often problematic.
Explicit NULL Comparison

```sql
SELECT NULL = NULL;
?column?
---------
(null)

SELECT NULL IS NULL;
?column?
---------
t

SELECT NULL IS NOT NULL;
?column?
---------
f
```
Explicit NULL Comparison

```
SELECT * FROM inctest;
   x
--------
   30
   40
   (null)

SELECT * FROM inctest WHERE x IS NULL;
   x
--------
   (null)

SELECT * FROM inctest WHERE x IS NOT NULL;
   x
----
   30
   40
```
Comparing NULLs With True/False Logic

SELECT 2 IS DISTINCT FROM 1;
  ?column?
  --------
  t

SELECT NULL IS DISTINCT FROM 1;
  ?column?
  --------
  t

SELECT NULL IS DISTINCT FROM NULL;
  ?column?
  --------
  f

SELECT NULL <> 1;
  ?column?
  --------
  (null)
Explicit Equality Comparisons With NULL

```
SELECT * FROM inctest WHERE x IS DISTINCT FROM 30;
  x
--------
  40
  (null)

SELECT * FROM inctest WHERE x IS NOT DISTINCT FROM 30;
  x
----
  30
```
Comparing NULLs to Other NULLs

CREATE TABLE disttest (x INTEGER, y INTEGER);

INSERT INTO disttest VALUES (1, 1), (2, 3), (NULL, NULL);

SELECT * FROM disttest where x IS NOT DISTINCT FROM y;

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(null)</td>
<td>(null)</td>
</tr>
</tbody>
</table>

This is particularly useful for joins.
Ordering of NULLs

```sql
SELECT * FROM (VALUES (NULL), (2), (1), (NULL)) ORDER BY 1;
```

<table>
<thead>
<tr>
<th>?column?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>(null)</td>
</tr>
<tr>
<td>(null)</td>
</tr>
</tbody>
</table>

NULLs are treated as equal for ordering purposes.
SELECT * FROM (VALUES (NULL), (2), (1), (NULL)) ORDER BY 1 NULLS FIRST;

<table>
<thead>
<tr>
<th>?column?</th>
</tr>
</thead>
<tbody>
<tr>
<td>(null)</td>
</tr>
<tr>
<td>(null)</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
CREATE TABLE uniqtest (x INTEGER);

CREATE UNIQUE INDEX i_uniqtest ON uniqtest (x);

INSERT INTO uniqtest VALUES (1), (NULL), (NULL);

SELECT * FROM uniqtest;

```
x
---
1
(null)
(null)
```
NULLs and Aggregates

CREATE TABLE aggtest (x INTEGER);

INSERT INTO aggtest VALUES (7), (8), (NULL);

SELECT COUNT(*), COUNT(x), SUM(x), MIN(x), MAX(x), AVG(x) FROM aggtest;
  count | count | sum | min | max | avg
  ------+-------+-----+-----+-----+--------------------
        3 | 2     | 15  | 7   | 8   | 7.5000000000000000

DELETE FROM aggtest;

SELECT COUNT(*), COUNT(x), SUM(x), MIN(x), MAX(x), AVG(x) FROM aggtest;
  count | count | sum   | min   | max   | avg
  ------+-------+-------+-------+-------+-----------------------
         0 | 0     | (null)| (null)| (null)| (null)                 

The sum of zero rows is NULL.
DELETE FROM aggtest;

INSERT INTO aggtest VALUES (7), (8), (NULL), (NULL);

SELECT x, COUNT(*), COUNT(x), SUM(x), MIN(x), MAX(x), AVG(x) FROM aggtest GROUP BY x ORDER BY x;

<table>
<thead>
<tr>
<th>x</th>
<th>count</th>
<th>count</th>
<th>sum</th>
<th>min</th>
<th>max</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7.00000000000000000</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8.00000000000000000</td>
</tr>
<tr>
<td>(null)</td>
<td>2</td>
<td>0</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
</tr>
</tbody>
</table>
Mapping NULLs to Non-NULLs

SELECT COALESCE(NULL, 0);
    coalesce
----------
    0

SELECT COALESCE(NULL, 'I am null.');
    coalesce
----------
    I am null.
CREATE TABLE nullmaptest (x TEXT);

INSERT INTO nullmaptest VALUES ('f'), ('g'), (NULL);

SELECT x, COALESCE(x, 'n/a') FROM nullmaptest;

<table>
<thead>
<tr>
<th></th>
<th>coalesce</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>g</td>
<td>g</td>
</tr>
<tr>
<td>(null)</td>
<td>n/a</td>
</tr>
</tbody>
</table>

SELECT 'a' || COALESCE(NULL, '') || 'b';

?column?

       
ab

SELECT SUM(x), COALESCE(SUM(x), 0) FROM aggtest;

<table>
<thead>
<tr>
<th></th>
<th>coalesce</th>
</tr>
</thead>
<tbody>
<tr>
<td>(null)</td>
<td>0</td>
</tr>
</tbody>
</table>
DELETE FROM nullmaptest;

INSERT INTO nullmaptest VALUES ('f'), ('g'), ('n/a');

SELECT x, NULLIF(x, 'n/a') FROM nullmaptest;

<table>
<thead>
<tr>
<th>x</th>
<th>nullif</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>g</td>
<td>g</td>
</tr>
<tr>
<td>n/a</td>
<td>(null)</td>
</tr>
</tbody>
</table>

SELECT NULLIF('n/a', COALESCE(NULL, 'n/a'));

nullif

--------

(null)
NULLs In Arrays

SELECT NULL::INTEGER[] IS NULL;
?column?
----------
t

SELECT '{}'::INTEGER[] IS NULL;
?column?
----------
f

SELECT '{NULL}'::INTEGER[] IS NULL;
?column?
----------
f
Row Expressions With NULLs

SELECT ROW() IS NULL;
  ?column?
-------------
     t

SELECT ROW(NULL) IS NULL;
  ?column?
-------------
     t

SELECT ROW(NULL,NULL) IS NULL;
  ?column?
-------------
     t
Row Expressions With NULLs

SELECT ROW(NULL,1) IS NULL;
  ?column?
  ----------
  f

SELECT ROW(NULL,1) IS NOT NULL;
  ?column?
  ----------
  f

SELECT ROW(1,2) IS NOT NULL;
  ?column?
  ----------
  t
Queries Returning NULLs in the Target List

CREATE TABLE emptytest (x INTEGER);

SELECT * from emptytest;
  x -----

SELECT (SELECT * from emptytest);
  x -------
  (null)

SELECT (SELECT * from emptytest) IS NULL;
  ?column? -------
  t

A SELECT with no FROM clause is assumed to return one row.
"Oh, that makes sense" — When you see individual behaviors of null, they look systematic, and your brain quickly sees a pattern and extrapolates what might happen in other situations. Often, that extrapolation is wrong, because null semantics are a mix of behaviors. I think the best way to think about null is as a Frankenstein monster of several philosophies and systems stitched together by a series of special cases.

Jeff Davis
Conclusion


https://momjian.us/presentations

https://www.flickr.com/photos/micspecial/inglefttosay/