IN vs ANY operator in PostgreSQL

Asked 7 years, 9 months ago Modified 9 months ago Viewed 331k times

275

What is the difference between `IN` and `ANY` operator in PostgreSQL?
The working mechanism of both seems to be the same. Can anyone explain this with an example?

sql postgresql sql-in

Share Improve this question Follow

edited Feb 10, 2022 at 18:16
user330315

asked Jan 6, 2016 at 6:33
mohangraj
9,922 19 60 94

4 Possible duplicate of postgreSQL - in vs any – Vivek S. Jan 6, 2016 at 6:47

Does this answer your question? Difference between in and any operators in sql – philipxy Feb 16, 2020 at 11:06

3 Answers

Sorted by: Highest score (default)

391

(Strictly speaking, `IN` and `ANY` are Postgres "constructs" or "syntax elements", rather than "operators".)

Logically, quoting the manual:

> `IN` is equivalent to `= ANY` .

But there are two ***syntax variants*** of `IN` and two variants of `ANY` . Details:

- How to use ANY instead of IN in a WHERE clause?

`IN` **taking a *set*** is equivalent to `= ANY` taking a *set*, as demonstrated here:

- PostgreSQL - IN vs ANY

But the second variant of each is subtly different. The second variant of the `ANY` construct takes an **array** (must be an actual array type), while the second variant of `IN` takes a comma-separated **list of values**. This leads to different restrictions in passing values and *can* also lead to different query plans in special cases:

- Index not used with `=any()` but used with `in`
- Pass multiple sets or arrays of values to a function

- How to match elements in an array of composite type?

`ANY` is more versatile

The `ANY` construct is far more versatile, as it can be combined with various operators, not just `=`. Example:

```
SELECT 'foo' LIKE ANY('{FOO,bar,%oo%}');
```

For a big number of values, providing a *set* scales better for each:

- Optimizing a Postgres query with a large IN

Related:

- Can PostgreSQL index array columns?

Inversion / opposite / exclusion

"Find rows where `id` is in the given array":

```
SELECT * FROM tbl WHERE id = ANY (ARRAY[1, 2]);
```

Inversion: *"Find rows where `id` is **not** in the array"*:

```
SELECT * FROM tbl WHERE id <> ALL (ARRAY[1, 2]);
SELECT * FROM tbl WHERE id <> ALL ('{1, 2}');  -- equivalent array literal
SELECT * FROM tbl WHERE NOT (id = ANY ('{1, 2}'));
```

All three equivalent. The first with ARRAY constructor, the other two with array literal. The type of the **untyped array literal** is derived from (known) element type to the left.
In other constellations (typed array value / you want a different type / ARRAY constructor for a non-default type) you may need to cast explicitly.

Rows with **`id IS NULL`** do not pass either of these expressions. To include `NULL` values additionally:

```
SELECT * FROM tbl WHERE (id = ANY ('{1, 2}')) IS NOT TRUE;
```

Share Improve this answer Follow

edited May 18, 2022 at 21:07

answered Jan 6, 2016 at 7:21

Erwin Brandstetter
610k 145 1085 1235

7 It'd be nice to explicitly clarify that the results of the second variants will always be the same. I'm 99% sure that is in fact the case but the answer doesn't seem to state it. Meaning that `SELECT * from mytable where id in (1, 2, 3)` will always result in the same rows as `SELECT * from`

`mytable where id = ANY('{1, 2, 3}')` , even if they potentially might have different query plans. – KPD Apr 8, 2018 at 23:44

2 `ANY` **cannot** be combined with the `!=` operator. I don't think it's documented, but `select * from foo where id != ANY (ARRAY[1, 2])` is not the same as `select * from foo where id NOT IN (1, 2)` . On the other hand, `select * from foo where NOT (id = ANY (ARRAY[1, 2]))` works as expected. – qris Dec 7, 2018 at 12:36

1 @qris: `ANY` can be combined with the `!=` operator. But there is more to it. I added a chapter above. (Note that `<>` is the operator in standard SQL - though `!=` is accepted as well in Postgres.) – Erwin Brandstetter Feb 7, 2019 at 12:24

2 @dvtan: `(id = ...) IS NOT TRUE` works because `id = ...` only evaluates to `TRUE` if there is an actual match. Outcomes `FALSE` or `NULL` pass our test. See: stackoverflow.com/a/23767625/939860. Your added expression tests for something else. This would be equivalent `WHERE id <> ALL (ARRAY[1, 2]) OR id IS NULL;` – Erwin Brandstetter Feb 3, 2020 at 23:17

1 @ErwinBrandstetter I found out this commit patched. not sure how faster can it be to speed IN clause. git.postgresql.org/gitweb/... – jian Jul 6 at 11:26

16

There are two obvious points, as well as the points in the other answer:

- They are exactly equivalent when using sub queries:

```
SELECT * FROM table
WHERE column IN(subquery);

SELECT * FROM table
WHERE column = ANY(subquery);
```

On the other hand:

- Only the `IN` operator allows a simple list:

```
SELECT * FROM table
WHERE column IN(… , … , …);
```

Presuming they are exactly the same has caught me out several times when forgetting that `ANY` doesn’t work with lists.

Share Improve this answer Follow

answered Feb 16, 2020 at 7:51
Manngo
14.3k 10 90 111

For the "simple list" case, `WHERE id = ANY(array[1,2])` works. – Nathan Long Sep 29 at 14:13

or, using the example in this answer, `WHERE id = ANY(array(<subquery>))` – BrDaHa Oct 4 at 23:34

2

`'in'` is syntaxis sugar, you can take a look to plan analyse and will see that 'in ' will be transform to `=ANY('...,...')::yourType[]`

Share Improve this answer Follow

answered Jan 11 at 15:50
java developer 111
101 9

I don't think this is true. I have a query that was taking 26 seconds to execute using an `IN` constraint (values from a subquery), but when I switched to an `= ANY(array(<subquery>))` the same query took a couple hundred milliseconds. The query plan definitely changed. PG 12.3
– BrDaHa Oct 4 at 23:32