delete

- Description
- Parameters
- Outputs
- Notes
- Examples
- Compatibility
- See Also

DELETE

DELETE

7

SQL - Language Statements

DELETE

delete rows of a table

[WITH [RECURSIVE]

with_query

[, ...]] DELETE FROM [ONLY]

table_name

[*] [[AS]

alias

] [USING

from_item

[, ...]] [WHERE

condition

| WHERE CURRENT OF

cursor_name

] [RETURNING [WITH ({ OLD | NEW } AS

output_alias

[, ...])] { * |

output_expression

[[AS]

output_name

] } [, ...]]

Description

`DELETE` deletes rows that satisfy the `WHERE` clause from the specified table. If the `WHERE` clause is absent, the effect is to delete all rows in the table. The result is a valid, but empty table.

> [!TIP] `TRUNCATE` provides a faster mechanism to remove all rows from a table.

There are two ways to delete rows in a table using information contained in other tables in the database: using sub-selects, or specifying additional tables in the `USING` clause. Which technique is more appropriate depends on the specific circumstances.

The optional `RETURNING` clause causes `DELETE` to compute and return value(s) based on each row actually deleted. Any expression using the table's columns, and/or columns of other tables mentioned in `USING`, can be computed. The syntax of the `RETURNING` list is identical to that of the output list of `SELECT`.

You must have the `DELETE` privilege on the table to delete from it, as well as the `SELECT` privilege for any table in the `USING` clause or whose values are read in the <condition>.

Parameters

<with_query>
The `WITH` clause allows you to specify one or more subqueries that can be referenced by name in the `DELETE` query. See ??? and ??? for details.

<table_name>
The name (optionally schema-qualified) of the table to delete rows from. If `ONLY` is specified before the table name, matching rows are deleted from the named table only. If `ONLY` is not specified, matching rows are also deleted from any tables inheriting from the named table. Optionally, `*` can be specified after the table name to explicitly indicate that descendant tables are included.

<alias>
A substitute name for the target table. When an alias is provided, it completely hides the actual name of the table. For example, given `DELETE FROM foo AS f`, the remainder of the `DELETE` statement must refer to this table as `f` not `foo`.

<from_item>
A table expression allowing columns from other tables to appear in the `WHERE` condition. This uses the same syntax as the `FROM` clause of a `SELECT` statement; for example, an alias for the table name can be specified. Do not repeat the target table as a <from_item> unless you wish to set up a self-join (in which case it must appear with an alias in the <from_item>).

<condition>
An expression that returns a value of type `boolean`. Only rows for which this expression returns `true` will be deleted.

<cursor_name>
The name of the cursor to use in a `WHERE CURRENT OF` condition. The row to be deleted is the one most recently fetched from this cursor. The cursor must be a non-grouping query on the `DELETE`'s target table. Note that `WHERE CURRENT OF` cannot be specified together with a Boolean condition. See ??? for more information about using cursors with `WHERE CURRENT OF`.

<output_alias>
An optional substitute name for `OLD` or `NEW` rows in the `RETURNING` list.

By default, old values from the target table can be returned by writing `OLD.column_name` or `OLD.*`, and new values can be returned by writing `NEW.column_name` or `NEW.*`. When an alias is provided, these names are hidden and the old or new rows must be referred to using the alias. For example `RETURNING WITH (OLD AS o, NEW AS n) o.*, n.*`.

<output_expression>
An expression to be computed and returned by the `DELETE` command after each row is deleted.

The expression can use any column names of the table named by <table_name> or table(s) listed in `USING`. Write `*` to return all columns.

A column name or `*` may be qualified using `OLD` or `NEW`, or the corresponding <output_alias> for `OLD` or `NEW`, to cause old or new values to be returned. An unqualified column name, or `*`, or a column name or `*` qualified using the target table name or alias will return old values.

For a simple `DELETE`, all new values will be `NULL`. However, if an `ON DELETE` rule causes an `INSERT` or `UPDATE` to be executed instead, the new values may be non-`NULL`.

<output_name>
A name to use for a returned column.

Outputs

On successful completion, a `DELETE` command returns a command tag of the form

```
DELETE count
```

The <count> is the number of rows deleted. Note that the number may be less than the number of rows that matched the <condition> when deletes were suppressed by a `BEFORE DELETE` trigger. If <count> is 0, no rows were deleted by the query (this is not considered an error).

If the `DELETE` command contains a `RETURNING` clause, the result will be similar to that of a `SELECT` statement containing the columns and values defined in the `RETURNING` list, computed over the row(s) deleted by the command.

Notes

PostgreSQL lets you reference columns of other tables in the `WHERE` condition by specifying the other tables in the `USING` clause. For example, to delete all films produced by a given producer, one can do:

```
DELETE FROM films USING producers
  WHERE producer_id = producers.id AND producers.name = 'foo';
```

What is essentially happening here is a join between films and producers, with all successfully joined films rows being marked for deletion. This syntax is not standard. A more standard way to do it is:

```
DELETE FROM films
  WHERE producer_id IN (SELECT id FROM producers WHERE name = 'foo');
```

In some cases the join style is easier to write or faster to execute than the sub-select style.

Examples

Delete all films but musicals:

```
DELETE FROM films WHERE kind <> 'Musical';
```

Clear the table films:

```
DELETE FROM films;
```

Delete completed tasks, returning full details of the deleted rows:

```
DELETE FROM tasks WHERE status = 'DONE' RETURNING *;
```

Delete the row of tasks on which the cursor `c_tasks` is currently positioned:

```
DELETE FROM tasks WHERE CURRENT OF c_tasks;
```

While there is no `LIMIT` clause for `DELETE`, it is possible to get a similar effect using the same method described in the documentation of `UPDATE`:

```
WITH delete_batch AS (
  SELECT l.ctid FROM user_logs AS l
    WHERE l.status = 'archived'
    ORDER BY l.creation_date
    FOR UPDATE
    LIMIT 10000
)
DELETE FROM user_logs AS dl
  USING delete_batch AS del
  WHERE dl.ctid = del.ctid;
```

This use of ctid is only safe because the query is repeatedly run, avoiding the problem of changed ctids.

Compatibility

This command conforms to the SQL standard, except that the `USING` and `RETURNING` clauses are PostgreSQL extensions, as is the ability to use `WITH` with `DELETE`.

See Also

□□□ #2
□ □□□ □□□ 2026-10-31 14:10:23 UTC
□ □□□ □□□ 2026-10-31 14:10:24 UTC