NAME

DROP_PROCEDURE − remove a procedure

SYNOPSIS

DROP PROCEDURE [IF EXISTS] *name* [([[*argmode*] [*argname*] *argtype* [, ...]])] [, ...]
[CASCADE | RESTRICT]

DESCRIPTION

DROP PROCEDURE removes the definition of one or more existing procedures. To execute this command the user must be the owner of the procedure(s). The argument types to the procedure(s) usually must be specified, since several different procedures can exist with the same name and different argument lists.

PARAMETERS

IF EXISTS

Do not throw an error if the procedure does not exist. A notice is issued in this case.

name

The name (optionally schema−qualified) of an existing procedure.

argmode

The mode of an argument: IN, OUT, INOUT, or VARIADIC. If omitted, the default is IN (but see below).

argname

The name of an argument. Note that **DROP PROCEDURE** does not actually pay any attention to argument names, since only the argument data types are used to determine the procedure's identity.

argtype

The data type(s) of the procedure's arguments (optionally schema−qualified), if any. See below for details.

CASCADE

Automatically drop objects that depend on the procedure, and in turn all objects that depend on those objects (see Section 5.15).

RESTRICT

Refuse to drop the procedure if any objects depend on it. This is the default.

NOTES

If there is only one procedure of the given name, the argument list can be omitted. Omit the parentheses too in this case.

In PostgreSQL, it's sufficient to list the input (including INOUT) arguments, because no two routines of the same name are allowed to share the same input−argument list. Moreover, the **DROP** command will not actually check that you wrote the types of OUT arguments correctly; so any arguments that are explicitly marked OUT are just noise. But writing them is recommendable for consistency with the corresponding **CREATE** command.

For compatibility with the SQL standard, it is also allowed to write all the argument data types (including those of OUT arguments) without any *argmode* markers. When this is done, the types of the procedure's OUT argument(s) *will* be verified against the command. This provision creates an ambiguity, in that when the argument list contains no *argmode* markers, it's unclear which rule is intended. The **DROP** command will attempt the lookup both ways, and will throw an error if two different procedures are found. To avoid the risk of such ambiguity, it's recommendable to write IN markers explicitly rather than letting them be defaulted, thus forcing the traditional PostgreSQL interpretation to be used.

The lookup rules just explained are also used by other commands that act on existing procedures, such as **ALTER PROCEDURE** and **COMMENT ON PROCEDURE**.

EXAMPLES

If there is only one procedure do_db_maintenance, this command is sufficient to drop it:

```
DROP PROCEDURE do_db_maintenance;
```

Given this procedure definition:

```
CREATE PROCEDURE do_db_maintenance(IN target_schema text, OUT results text) ...
```

any one of these commands would work to drop it:

```
DROP PROCEDURE do_db_maintenance(IN target_schema text, OUT results text);
DROP PROCEDURE do_db_maintenance(IN text, OUT text);
DROP PROCEDURE do_db_maintenance(IN text);
DROP PROCEDURE do_db_maintenance(text);
DROP PROCEDURE do_db_maintenance(text, text);  -- potentially ambiguous
```

However, the last example would be ambiguous if there is also, say,

```
CREATE PROCEDURE do_db_maintenance(IN target_schema text, IN options text) ...
```

COMPATIBILITY

This command conforms to the SQL standard, with these PostgreSQL extensions:

- The standard only allows one procedure to be dropped per command.
- The IF EXISTS option is an extension.
- The ability to specify argument modes and names is an extension, and the lookup rules differ when modes are given.

SEE ALSO

CREATE PROCEDURE (**CREATE_PROCEDURE**(7)), ALTER PROCEDURE (**ALTER_PROCEDURE**(7)), DROP FUNCTION (**DROP_FUNCTION**(7)), DROP ROUTINE (**DROP_ROUTINE**(7))