NAME

ALTER_DOMAIN − change the definition of a domain

SYNOPSIS

```
ALTER DOMAIN name
  { SET DEFAULT expression | DROP DEFAULT }
ALTER DOMAIN name
  { SET | DROP } NOT NULL
ALTER DOMAIN name
  ADD domain_constraint [ NOT VALID ]
ALTER DOMAIN name
  DROP CONSTRAINT [ IF EXISTS ] constraint_name [ RESTRICT | CASCADE ]
ALTER DOMAIN name
   RENAME CONSTRAINT constraint_name TO new_constraint_name
ALTER DOMAIN name
  VALIDATE CONSTRAINT constraint_name
ALTER DOMAIN name
  OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER | SESSION_USER }
ALTER DOMAIN name
  RENAME TO new_name
ALTER DOMAIN name
  SET SCHEMA new_schema
```

where *domain_constraint* is:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL | CHECK (expression) }
```

DESCRIPTION

ALTER DOMAIN changes the definition of an existing domain. There are several sub−forms:

SET/DROP DEFAULT
: These forms set or remove the default value for a domain. Note that defaults only apply to subsequent **INSERT** commands; they do not affect rows already in a table using the domain.

SET/DROP NOT NULL
: These forms change whether a domain is marked to allow NULL values or to reject NULL values. You can only SET NOT NULL when the columns using the domain contain no null values.

ADD *domain_constraint* [NOT VALID]
: This form adds a new constraint to a domain. When a new constraint is added to a domain, all columns using that domain will be checked against the newly added constraint. These checks can be suppressed by adding the new constraint using the NOT VALID option; the constraint can later be made valid using **ALTER DOMAIN ... VALIDATE CONSTRAINT**. Newly inserted or updated rows are always checked against all constraints, even those marked NOT VALID. NOT VALID is only accepted for CHECK constraints.

DROP CONSTRAINT [IF EXISTS]
: This form drops constraints on a domain. If IF EXISTS is specified and the constraint does not exist, no error is thrown. In this case a notice is issued instead.

RENAME CONSTRAINT
: This form changes the name of a constraint on a domain.

VALIDATE CONSTRAINT
: This form validates a constraint previously added as NOT VALID, that is, it verifies that all values in table columns of the domain type satisfy the specified constraint.

OWNER
: This form changes the owner of the domain to the specified user.

RENAME
: This form changes the name of the domain.

SET SCHEMA
: This form changes the schema of the domain. Any constraints associated with the domain are moved into the new schema as well.

You must own the domain to use **ALTER DOMAIN**. To change the schema of a domain, you must also have CREATE privilege on the new schema. To alter the owner, you must be able to SET ROLE to the new owning role, and that role must have CREATE privilege on the domain's schema. (These restrictions enforce that altering the owner doesn't do anything you couldn't do by dropping and recreating the domain. However, a superuser can alter ownership of any domain anyway.)

PARAMETERS

name
: The name (possibly schema−qualified) of an existing domain to alter.

domain_constraint
: New domain constraint for the domain.

constraint_name
: Name of an existing constraint to drop or rename.

NOT VALID
: Do not verify existing stored data for constraint validity.

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

RESTRICT
: Refuse to drop the constraint if there are any dependent objects. This is the default behavior.

new_name
: The new name for the domain.

new_constraint_name
: The new name for the constraint.

new_owner
: The user name of the new owner of the domain.

new_schema
: The new schema for the domain.

NOTES

Although **ALTER DOMAIN ADD CONSTRAINT** attempts to verify that existing stored data satisfies the new constraint, this check is not bulletproof, because the command cannot “see” table rows that are newly inserted or updated and not yet committed. If there is a hazard that concurrent operations might insert bad data, the way to proceed is to add the constraint using the NOT VALID option, commit that command, wait until all transactions started before that commit have finished, and then issue **ALTER DOMAIN VALIDATE CONSTRAINT** to search for data violating the constraint. This method is reliable because once the constraint is committed, all new transactions are guaranteed to enforce it against new values of the domain type.

Currently, **ALTER DOMAIN ADD CONSTRAINT**, **ALTER DOMAIN VALIDATE CONSTRAINT**, and **ALTER DOMAIN SET NOT NULL** will fail if the named domain or any derived domain is used within a container−type column (a composite, array, or range column) in any table in the database. They should eventually be improved to be able to verify the new constraint for such nested values.

EXAMPLES

To add a NOT NULL constraint to a domain:

ALTER DOMAIN zipcode SET NOT NULL;

To remove a NOT NULL constraint from a domain:

ALTER DOMAIN zipcode DROP NOT NULL;

To add a check constraint to a domain:

ALTER DOMAIN zipcode ADD CONSTRAINT zipchk CHECK (char_length(VALUE) = 5);

To remove a check constraint from a domain:

ALTER DOMAIN zipcode DROP CONSTRAINT zipchk;

To rename a check constraint on a domain:

ALTER DOMAIN zipcode RENAME CONSTRAINT zipchk TO zip_check;

To move the domain into a different schema:

ALTER DOMAIN zipcode SET SCHEMA customers;

COMPATIBILITY

ALTER DOMAIN conforms to the SQL standard, except for the OWNER, RENAME, SET SCHEMA, and VALIDATE CONSTRAINT variants, which are PostgreSQL extensions. The NOT VALID clause of the ADD CONSTRAINT variant is also a PostgreSQL extension.

SEE ALSO

CREATE DOMAIN (**CREATE_DOMAIN**(7)), DROP DOMAIN (**DROP_DOMAIN**(7))