NAME

CREATE_DOMAIN − define a new domain

SYNOPSIS

CREATE DOMAIN *name* [AS] *data_type*
[COLLATE *collation*]
[DEFAULT *expression*]
[*domain_constraint* [...]]

where *domain_constraint* is:

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

DESCRIPTION

CREATE DOMAIN creates a new domain. A domain is essentially a data type with optional constraints (restrictions on the allowed set of values). The user who defines a domain becomes its owner.

If a schema name is given (for example, CREATE DOMAIN myschema.mydomain ...) then the domain is created in the specified schema. Otherwise it is created in the current schema. The domain name must be unique among the types and domains existing in its schema.

Domains are useful for abstracting common constraints on fields into a single location for maintenance. For example, several tables might contain email address columns, all requiring the same CHECK constraint to verify the address syntax. Define a domain rather than setting up each table's constraint individually.

To be able to create a domain, you must have USAGE privilege on the underlying type.

PARAMETERS

name
The name (optionally schema−qualified) of a domain to be created.

data_type
The underlying data type of the domain. This can include array specifiers.

collation
An optional collation for the domain. If no collation is specified, the domain has the same collation behavior as its underlying data type. The underlying type must be collatable if COLLATE is specified.

DEFAULT *expression*
The DEFAULT clause specifies a default value for columns of the domain data type. The value is any variable−free expression (but subqueries are not allowed). The data type of the default expression must match the data type of the domain. If no default value is specified, then the default value is the null value.

The default expression will be used in any insert operation that does not specify a value for the column. If a default value is defined for a particular column, it overrides any default associated with the domain. In turn, the domain default overrides any default value associated with the underlying data type.

CONSTRAINT *constraint_name*
An optional name for a constraint. If not specified, the system generates a name.

NOT NULL
Values of this domain are prevented from being null (but see notes below).

NULL
Values of this domain are allowed to be null. This is the default.

This clause is only intended for compatibility with nonstandard SQL databases. Its use is discouraged in new applications.

CHECK (*expression*)

CHECK clauses specify integrity constraints or tests which values of the domain must satisfy. Each constraint must be an expression producing a Boolean result. It should use the key word VALUE to refer to the value being tested. Expressions evaluating to TRUE or UNKNOWN succeed. If the expression produces a FALSE result, an error is reported and the value is not allowed to be converted to the domain type.

Currently, CHECK expressions cannot contain subqueries nor refer to variables other than VALUE.

When a domain has multiple CHECK constraints, they will be tested in alphabetical order by name. (PostgreSQL versions before 9.5 did not honor any particular firing order for CHECK constraints.)

NOTES

Domain constraints, particularly NOT NULL, are checked when converting a value to the domain type. It is possible for a column that is nominally of the domain type to read as null despite there being such a constraint. For example, this can happen in an outer−join query, if the domain column is on the nullable side of the outer join. A more subtle example is

```
INSERT INTO tab (domcol) VALUES ((SELECT domcol FROM tab WHERE false));
```

The empty scalar sub−SELECT will produce a null value that is considered to be of the domain type, so no further constraint checking is applied to it, and the insertion will succeed.

It is very difficult to avoid such problems, because of SQL's general assumption that a null value is a valid value of every data type. Best practice therefore is to design a domain's constraints so that a null value is allowed, and then to apply column NOT NULL constraints to columns of the domain type as needed, rather than directly to the domain type.

PostgreSQL assumes that CHECK constraints' conditions are immutable, that is, they will always give the same result for the same input value. This assumption is what justifies examining CHECK constraints only when a value is first converted to be of a domain type, and not at other times. (This is essentially the same as the treatment of table CHECK constraints, as described in Section 5.5.1.)

An example of a common way to break this assumption is to reference a user−defined function in a CHECK expression, and then change the behavior of that function. PostgreSQL does not disallow that, but it will not notice if there are stored values of the domain type that now violate the CHECK constraint. That would cause a subsequent database dump and restore to fail. The recommended way to handle such a change is to drop the constraint (using **ALTER DOMAIN**), adjust the function definition, and re−add the constraint, thereby rechecking it against stored data.

It's also good practice to ensure that domain CHECK expressions will not throw errors.

EXAMPLES

This example creates the us_postal_code data type and then uses the type in a table definition. A regular expression test is used to verify that the value looks like a valid US postal code:

```
CREATE DOMAIN us_postal_code AS TEXT
CHECK(
   VALUE ~ '^\d{5}$'
OR VALUE ~ '^\d{5}−\d{4}$'
);

CREATE TABLE us_snail_addy (
  address_id SERIAL PRIMARY KEY,
  street1 TEXT NOT NULL,
  street2 TEXT,
  street3 TEXT,
  city TEXT NOT NULL,
```

```
  postal us_postal_code NOT NULL
);
```

COMPATIBILITY

The command **CREATE DOMAIN** conforms to the SQL standard.

The syntax NOT NULL in this command is a PostgreSQL extension. (A standard−conforming way to write the same for non−composite data types would be CHECK (VALUE IS NOT NULL). However, per the section called “NOTES”, such constraints are best avoided in practice anyway.) The NULL “constraint” is a PostgreSQL extension (see also Compatibility).

SEE ALSO

ALTER DOMAIN (**ALTER_DOMAIN**(7)), DROP DOMAIN (**DROP_DOMAIN**(7))