NAME

ALTER_FOREIGN_TABLE − change the definition of a foreign table

SYNOPSIS

ALTER FOREIGN TABLE [IF EXISTS] [ONLY] *name* [*]
action [, ...]
ALTER FOREIGN TABLE [IF EXISTS] [ONLY] *name* [*]
RENAME [COLUMN] *column_name* TO *new_column_name*
ALTER FOREIGN TABLE [IF EXISTS] *name*
RENAME TO *new_name*
ALTER FOREIGN TABLE [IF EXISTS] *name*
SET SCHEMA *new_schema*

where *action* is one of:

ADD [COLUMN] [IF NOT EXISTS] *column_name data_type* [COLLATE *collation*] [*column_constraint* [...]]
DROP [COLUMN] [IF EXISTS] *column_name* [RESTRICT | CASCADE]
ALTER [COLUMN] *column_name* [SET DATA] TYPE *data_type* [COLLATE *collation*]
ALTER [COLUMN] *column_name* SET DEFAULT *expression*
ALTER [COLUMN] *column_name* DROP DEFAULT
ALTER [COLUMN] *column_name* { SET | DROP } NOT NULL
ALTER [COLUMN] *column_name* SET STATISTICS *integer*
ALTER [COLUMN] *column_name* SET (*attribute_option* = *value* [, ...])
ALTER [COLUMN] *column_name* RESET (*attribute_option* [, ...])
ALTER [COLUMN] *column_name* SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN | DEFAULT }
ALTER [COLUMN] *column_name* OPTIONS ([ADD | SET | DROP] *option* ['*value*'] [, ...])
ADD *table_constraint* [NOT VALID]
VALIDATE CONSTRAINT *constraint_name*
DROP CONSTRAINT [IF EXISTS] *constraint_name* [RESTRICT | CASCADE]
DISABLE TRIGGER [*trigger_name* | ALL | USER]
ENABLE TRIGGER [*trigger_name* | ALL | USER]
ENABLE REPLICA TRIGGER *trigger_name*
ENABLE ALWAYS TRIGGER *trigger_name*
SET WITHOUT OIDS
INHERIT *parent_table*
NO INHERIT *parent_table*
OWNER TO { *new_owner* | CURRENT_ROLE | CURRENT_USER | SESSION_USER }
OPTIONS ([ADD | SET | DROP] *option* ['*value*'] [, ...])

DESCRIPTION

ALTER FOREIGN TABLE changes the definition of an existing foreign table. There are several subforms:

ADD [COLUMN] [IF NOT EXISTS]
: This form adds a new column to the foreign table, using the same syntax as **CREATE FOREIGN TABLE**. If IF NOT EXISTS is specified and a column already exists with this name, no error is thrown. Unlike the case when adding a column to a regular table, nothing happens to the underlying storage: this action simply declares that some new column is now accessible through the foreign table.

DROP [COLUMN] [IF EXISTS]
: This form drops a column from a foreign table. You will need to say CASCADE if anything outside the table depends on the column; for example, views. If IF EXISTS is specified and the column does not exist, no error is thrown. In this case a notice is issued instead.

SET DATA TYPE
: This form changes the type of a column of a foreign table. Again, this has no effect on any underlying storage: this action simply changes the type that PostgreSQL believes the column to have.

SET/DROP DEFAULT
: These forms set or remove the default value for a column. Default values only apply in subsequent **INSERT** or **UPDATE** commands; they do not cause rows already in the table to change.

SET/DROP NOT NULL
: Mark a column as allowing, or not allowing, null values.

SET STATISTICS
: This form sets the per−column statistics−gathering target for subsequent **ANALYZE** operations. See the similar form of **ALTER TABLE** for more details.

SET (*attribute_option* = *value* [, ...])
RESET (*attribute_option* [, ...])
: This form sets or resets per−attribute options. See the similar form of **ALTER TABLE** for more details.

SET STORAGE
: This form sets the storage mode for a column. See the similar form of **ALTER TABLE** for more details. Note that the storage mode has no effect unless the table's foreign−data wrapper chooses to pay attention to it.

ADD *table_constraint* [NOT VALID]
: This form adds a new constraint to a foreign table, using the same syntax as **CREATE FOREIGN TABLE**. Currently only CHECK and NOT NULL constraints are supported.

  Unlike the case when adding a constraint to a regular table, nothing is done to verify the constraint is correct; rather, this action simply declares that some new condition should be assumed to hold for all rows in the foreign table. (See the discussion in **CREATE FOREIGN TABLE**.) If the constraint is marked NOT VALID (allowed only for the CHECK case), then it isn't assumed to hold, but is only recorded for possible future use.

VALIDATE CONSTRAINT
: This form marks as valid a constraint that was previously marked as NOT VALID. No action is taken to verify the constraint, but future queries will assume that it holds.

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

DISABLE/ENABLE [REPLICA | ALWAYS] TRIGGER
: These forms configure the firing of trigger(s) belonging to the foreign table. See the similar form of **ALTER TABLE** for more details.

SET WITHOUT OIDS
: Backward compatibility syntax for removing the oid system column. As oid system columns cannot be added anymore, this never has an effect.

INHERIT *parent_table*
: This form adds the target foreign table as a new child of the specified parent table. See the similar form of **ALTER TABLE** for more details.

NO INHERIT *parent_table*
: This form removes the target foreign table from the list of children of the specified parent table.

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

OPTIONS ([ADD | SET | DROP] *option* ['*value*'] [, ...])
: Change options for the foreign table or one of its columns. ADD, SET, and DROP specify the action to be performed. ADD is assumed if no operation is explicitly specified. Duplicate option names are not allowed (although it's OK for a table option and a column option to have the same name). Option names and values are also validated using the foreign data wrapper library.

RENAME
: The RENAME forms change the name of a foreign table or the name of an individual column in a foreign table.

SET SCHEMA
: This form moves the foreign table into another schema.

All the actions except RENAME and SET SCHEMA can be combined into a list of multiple alterations to apply in parallel. For example, it is possible to add several columns and/or alter the type of several columns in a single command.

If the command is written as ALTER FOREIGN TABLE IF EXISTS ... and the foreign table does not exist, no error is thrown. A notice is issued in this case.

You must own the table to use **ALTER FOREIGN TABLE**. To change the schema of a foreign table, 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 table's schema. (These restrictions enforce that altering the owner doesn't do anything you couldn't do by dropping and recreating the table. However, a superuser can alter ownership of any table anyway.) To add a column or alter a column type, you must also have USAGE privilege on the data type.

PARAMETERS

name
: The name (possibly schema−qualified) of an existing foreign table to alter. If ONLY is specified before the table name, only that table is altered. If ONLY is not specified, the table and all its descendant tables (if any) are altered. Optionally, * can be specified after the table name to explicitly indicate that descendant tables are included.

column_name
: Name of a new or existing column.

new_column_name
: New name for an existing column.

new_name
: New name for the table.

data_type
: Data type of the new column, or new data type for an existing column.

table_constraint
: New table constraint for the foreign table.

constraint_name
: Name of an existing constraint to drop.

CASCADE
: Automatically drop objects that depend on the dropped column or constraint (for example, views referencing the column), and in turn all objects that depend on those objects (see Section 5.15).

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

trigger_name
: Name of a single trigger to disable or enable.

ALL
: Disable or enable all triggers belonging to the foreign table. (This requires superuser privilege if any of the triggers are internally generated triggers. The core system does not add such triggers to foreign tables, but add−on code could do so.)

USER
: Disable or enable all triggers belonging to the foreign table except for internally generated triggers.

parent_table
A parent table to associate or de−associate with this foreign table.

new_owner
The user name of the new owner of the table.

new_schema
The name of the schema to which the table will be moved.

NOTES

The key word COLUMN is noise and can be omitted.

Consistency with the foreign server is not checked when a column is added or removed with ADD COLUMN or DROP COLUMN, a NOT NULL or CHECK constraint is added, or a column type is changed with SET DATA TYPE. It is the user's responsibility to ensure that the table definition matches the remote side.

Refer to **CREATE FOREIGN TABLE** for a further description of valid parameters.

EXAMPLES

To mark a column as not−null:

```
ALTER FOREIGN TABLE distributors ALTER COLUMN street SET NOT NULL;
```

To change options of a foreign table:

```
ALTER FOREIGN TABLE myschema.distributors OPTIONS (ADD opt1 'value', SET opt2 'value2', DROP opt3);
```

COMPATIBILITY

The forms ADD, DROP, and SET DATA TYPE conform with the SQL standard. The other forms are PostgreSQL extensions of the SQL standard. Also, the ability to specify more than one manipulation in a single **ALTER FOREIGN TABLE** command is an extension.

ALTER FOREIGN TABLE DROP COLUMN can be used to drop the only column of a foreign table, leaving a zero−column table. This is an extension of SQL, which disallows zero−column foreign tables.

SEE ALSO

CREATE FOREIGN TABLE (**CREATE_FOREIGN_TABLE**(7)), DROP FOREIGN TABLE (**DROP_FOREIGN_TABLE**(7))