create_foreign_table

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

CREATE FOREIGN TABLE

CREATE FOREIGN TABLE

7

SQL - Language Statements

CREATE FOREIGN TABLE

define a new foreign table

CREATE FOREIGN TABLE [IF NOT EXISTS]

table_name

([{

column_name

data_type

[OPTIONS (

option

'

value

' [, ...])] [COLLATE

collation

] [

column_constraint

[...]] |

table_constraint

| LIKE

source_table

[

like_option

...] } [, ...]]) [INHERITS (

parent_table

[, ...])] SERVER

server_name

[OPTIONS (

option

'

value

' [, ...])] CREATE FOREIGN TABLE [IF NOT EXISTS]

table_name

PARTITION OF

parent_table

[({

column_name

[WITH OPTIONS] [

column_constraint

[...]] |

table_constraint

} [, ...])] { FOR VALUES

partition_bound_spec

| DEFAULT } SERVER

server_name

[OPTIONS (

option

'

value

' [, ...])]

where

column_constraint

is:

[CONSTRAINT

constraint_name

] { NOT NULL [NO INHERIT] | NULL | CHECK (

expression

) [NO INHERIT] | DEFAULT

default_expr

| GENERATED ALWAYS AS (

generation_expr

) [STORED | VIRTUAL] } [ENFORCED | NOT ENFORCED]

and

table_constraint

is:

[CONSTRAINT

constraint_name

] { NOT NULL

column_name

[NO INHERIT] | CHECK (

expression

) [NO INHERIT] } [ENFORCED | NOT ENFORCED]

and

like_option

is:

{ INCLUDING | EXCLUDING } { COMMENTS | CONSTRAINTS | DEFAULTS | GENERATED | STATISTICS | ALL }

and

partition_bound_spec

is:

IN (

partition_bound_expr

[, ...]) | FROM ({

partition_bound_expr

| MINVALUE | MAXVALUE } [, ...]) TO ({

partition_bound_expr

| MINVALUE | MAXVALUE } [, ...]) | WITH (MODULUS

numeric_literal

, REMAINDER

numeric_literal

)

Description

`CREATE FOREIGN TABLE` creates a new foreign table in the current database. The table will be owned by the user issuing the command.

If a schema name is given (for example, `CREATE FOREIGN TABLE myschema.mytable ...`) then the table is created in the specified schema. Otherwise it is created in the current schema. The name of the foreign table must be distinct from the name of any other relation (table, sequence, index, view, materialized view, or foreign table) in the same schema.

`CREATE FOREIGN TABLE` also automatically creates a data type that represents the composite type corresponding to one row of the foreign table. Therefore, foreign tables cannot have the same name as any existing data type in the same schema.

If `PARTITION OF` clause is specified then the table is created as a partition of `parent_table` with specified bounds.

To be able to create a foreign table, you must have `USAGE` privilege on the foreign server, as well as `USAGE` privilege on all column types used in the table.

Parameters

`IF NOT EXISTS`
Do not throw an error if a relation with the same name already exists. A notice is issued in this case. Note that there is no guarantee that the existing relation is anything like the one that would have been created.

<table_name>
The name (optionally schema-qualified) of the table to be created.

<column_name>
The name of a column to be created in the new table.

<data_type>
The data type of the column. This can include array specifiers. For more information on the data types supported by PostgreSQL, refer to ???.

`COLLATE collation`
The `COLLATE` clause assigns a collation to the column (which must be of a collatable data type). If not specified, the column data type's default collation is used.

`INHERITS ( parent_table [, ... ] )`
The optional `INHERITS` clause specifies a list of tables from which the new foreign table automatically inherits all columns. Parent tables can be plain tables or foreign tables. See the similar form of `CREATE TABLE` for more details.

`PARTITION OF parent_table { FOR VALUES partition_bound_spec | DEFAULT }`
This form can be used to create the foreign table as partition of the given parent table with specified partition bound values. See the similar form of `CREATE TABLE` for more details. Note that it is currently not allowed to create the foreign table as a partition of the parent table if there are `UNIQUE` indexes on the parent table. (See also `ALTER TABLE ATTACH PARTITION`.)

`LIKE source_table [ like_option ... ]`
The `LIKE` clause specifies a table from which the new table automatically copies all column names, their data types, and their not-null constraints.

Unlike `INHERITS`, the new table and original table are completely decoupled after creation is complete. Changes to the original table will not be applied to the new table, and it is not possible to include data of the new table in scans of the original table.

Also unlike `INHERITS`, columns and constraints copied by `LIKE` are not merged with similarly named columns and constraints. If the same name is specified explicitly or in another `LIKE` clause, an error is signaled.

The optional <like_option> clauses specify which additional properties of the original table to copy. Specifying `INCLUDING` copies the property, specifying `EXCLUDING` omits the property. `EXCLUDING` is the default. If multiple specifications are made for the same kind of object, the last one is used. The available options are:

`INCLUDING COMMENTS`
Comments for the copied columns, constraints, and extended statistics will be copied. The default behavior is to exclude comments, resulting in the corresponding objects in the new table having no comments.

`INCLUDING CONSTRAINTS`
`CHECK` constraints will be copied. No distinction is made between column constraints and table

constraints. Not-null constraints are always copied to the new table.

`INCLUDING DEFAULTS`
Default expressions for the copied column definitions will be copied. Otherwise, default expressions are not copied, resulting in the copied columns in the new table having null defaults. Note that copying defaults that call database-modification functions, such as `nextval`, may create a functional linkage between the original and new tables.

`INCLUDING GENERATED`
Any generation expressions of copied column definitions will be copied. By default, new columns will be regular base columns.

`INCLUDING STATISTICS`
Extended statistics are copied to the new table.

`INCLUDING ALL`
`INCLUDING ALL` is an abbreviated form selecting all the available individual options. (It could be useful to write individual `EXCLUDING` clauses after `INCLUDING ALL` to select all but some specific options.)

`CONSTRAINT constraint_name`
An optional name for a column or table constraint. If the constraint is violated, the constraint name is present in error messages, so constraint names like `col must be positive` can be used to communicate helpful constraint information to client applications. (Double-quotes are needed to specify constraint names that contain spaces.) If a constraint name is not specified, the system generates a name.

`NOT NULL` [NO INHERIT]
The column is not allowed to contain null values.

A constraint marked with `NO INHERIT` will not propagate to child tables.

`NULL`
The column is allowed to contain null values. This is the default.

This clause is only provided for compatibility with non-standard SQL databases. Its use is discouraged in new applications.

`CHECK ( expression ) [ NO INHERIT ]`
The `CHECK` clause specifies an expression producing a Boolean result which each row in the foreign table is expected to satisfy; that is, the expression should produce TRUE or UNKNOWN, never FALSE, for all rows in the foreign table. A check constraint specified as a column constraint should reference that column's value only, while an expression appearing in a table constraint can reference multiple columns.

Currently, `CHECK` expressions cannot contain subqueries nor refer to variables other than columns of the current row. The system column tableoid may be referenced, but not any other system

column.

A constraint marked with `NO INHERIT` will not propagate to child tables.

`DEFAULT default_expr`
The `DEFAULT` clause assigns a default data value for the column whose column definition it appears within. The value is any variable-free expression (subqueries and cross-references to other columns in the current table are not allowed). The data type of the default expression must match the data type of the column.

The default expression will be used in any insert operation that does not specify a value for the column. If there is no default for a column, then the default is null.

`GENERATED ALWAYS AS ( generation_expr ) [ STORED | VIRTUAL ]`
This clause creates the column as a generated column. The column cannot be written to, and when read the result of the specified expression will be returned.

When `VIRTUAL` is specified, the column will be computed when it is read. (The foreign-data wrapper will see it as a null value in new rows and may choose to store it as a null value or ignore it altogether.) When `STORED` is specified, the column will be computed on write. (The computed value will be presented to the foreign-data wrapper for storage and must be returned on reading.) `VIRTUAL` is the default.

The generation expression can refer to other columns in the table, but not other generated columns. Any functions and operators used must be immutable. References to other tables are not allowed.

<server_name>
The name of an existing foreign server to use for the foreign table. For details on defining a server, see ???.

`OPTIONS ( option 'value' [, ...] )`
Options to be associated with the new foreign table or one of its columns. The allowed option names and values are specific to each foreign data wrapper and are validated using the foreign-data wrapper's validator function. Duplicate option names are not allowed (although it's OK for a table option and a column option to have the same name).

Notes

Constraints on foreign tables (such as `CHECK` or `NOT NULL` clauses) are not enforced by the core PostgreSQL system, and most foreign data wrappers do not attempt to enforce them either; that is, the constraint is simply assumed to hold true. There would be little point in such enforcement since it would only apply to rows inserted or updated via the foreign table, and not to rows modified by

other means, such as directly on the remote server. Instead, a constraint attached to a foreign table should represent a constraint that is being enforced by the remote server.

Some special-purpose foreign data wrappers might be the only access mechanism for the data they access, and in that case it might be appropriate for the foreign data wrapper itself to perform constraint enforcement. But you should not assume that a wrapper does that unless its documentation says so.

Although PostgreSQL does not attempt to enforce constraints on foreign tables, it does assume that they are correct for purposes of query optimization. If there are rows visible in the foreign table that do not satisfy a declared constraint, queries on the table might produce errors or incorrect answers. It is the user's responsibility to ensure that the constraint definition matches reality.

> [!CAUTION] When a foreign table is used as a partition of a partitioned table, there is an implicit constraint that its contents must satisfy the partitioning rule. Again, it is the user's responsibility to ensure that that is true, which is best done by installing a matching constraint on the remote server.

Within a partitioned table containing foreign-table partitions, an `UPDATE` that changes the partition key value can cause a row to be moved from a local partition to a foreign-table partition, provided the foreign data wrapper supports tuple routing. However, it is not currently possible to move a row from a foreign-table partition to another partition. An `UPDATE` that would require doing that will fail due to the partitioning constraint, assuming that that is properly enforced by the remote server.

Similar considerations apply to generated columns. Stored generated columns are computed on insert or update on the local PostgreSQL server and handed to the foreign-data wrapper for writing out to the foreign data store, but it is not enforced that a query of the foreign table returns values for stored generated columns that are consistent with the generation expression. Again, this might result in incorrect query results.

Examples

Create foreign table films, which will be accessed through the server film_server:

```
CREATE FOREIGN TABLE films (
    code        char(5) NOT NULL,
    title       varchar(40) NOT NULL,
    did         integer NOT NULL,
    date_prod   date,
```

```
    kind        varchar(10),
    len         interval hour to minute
)
SERVER film_server;
```

Create foreign table measurement_y2016m07, which will be accessed through the server server_07, as a partition of the range partitioned table measurement:

```
CREATE FOREIGN TABLE measurement_y2016m07
    PARTITION OF measurement FOR VALUES FROM ('2016-07-01') TO ('2016-08-01')
    SERVER server_07;
```

Compatibility

The `CREATE FOREIGN TABLE` command largely conforms to the SQL standard; however, much as with `CREATE TABLE`, `NULL` constraints and zero-column foreign tables are permitted. The ability to specify column default values is also a PostgreSQL extension. Table inheritance, in the form defined by PostgreSQL, is nonstandard. The `LIKE` clause, as supported in this command, is nonstandard.

See Also

□□□ #2
□ □□□ □□□ 2026-10-31 14:09:01 UTC
□ □□□ □□□ 2026-10-31 14:09:02 UTC