NAME

CREATE_TABLE_AS − define a new table from the results of a query

SYNOPSIS

CREATE [[GLOBAL | LOCAL] { TEMPORARY | TEMP } | UNLOGGED] TABLE [IF NOT EXISTS] *table_name*
[(*column_name* [, ...])]
[USING *method*]
[WITH (*storage_parameter* [= *value*] [, ...]) | WITHOUT OIDS]
[ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP }]
[TABLESPACE *tablespace_name*]
AS *query*
[WITH [NO] DATA]

DESCRIPTION

CREATE TABLE AS creates a table and fills it with data computed by a **SELECT** command. The table columns have the names and data types associated with the output columns of the **SELECT** (except that you can override the column names by giving an explicit list of new column names).

CREATE TABLE AS bears some resemblance to creating a view, but it is really quite different: it creates a new table and evaluates the query just once to fill the new table initially. The new table will not track subsequent changes to the source tables of the query. In contrast, a view re−evaluates its defining **SELECT** statement whenever it is queried.

CREATE TABLE AS requires CREATE privilege on the schema used for the table.

PARAMETERS

GLOBAL or LOCAL
: Ignored for compatibility. Use of these keywords is deprecated; refer to CREATE TABLE (**CREATE_TABLE**(7)) for details.

TEMPORARY or TEMP
: If specified, the table is created as a temporary table. Refer to CREATE TABLE (**CREATE_TABLE**(7)) for details.

UNLOGGED
: If specified, the table is created as an unlogged table. Refer to CREATE TABLE (**CREATE_TABLE**(7)) for details.

IF NOT EXISTS
: Do not throw an error if a relation with the same name already exists; simply issue a notice and leave the table unmodified.

table_name
: The name (optionally schema−qualified) of the table to be created.

column_name
: The name of a column in the new table. If column names are not provided, they are taken from the output column names of the query.

USING *method*
: This optional clause specifies the table access method to use to store the contents for the new table; the method needs be an access method of type TABLE. See Chapter 62 for more information. If this option is not specified, the default table access method is chosen for the new table. See default_table_access_method for more information.

WITH (*storage_parameter* [= *value*] [, ...])
: This clause specifies optional storage parameters for the new table; see Storage Parameters in the CREATE TABLE (**CREATE_TABLE**(7)) documentation for more information. For backward−compatibility the WITH clause for a table can also include OIDS=FALSE to specify that rows of the new table should contain no OIDs (object identifiers), OIDS=TRUE is not supported anymore.

WITHOUT OIDS
: This is backward−compatible syntax for declaring a table WITHOUT OIDS, creating a table WITH OIDS is not supported anymore.

ON COMMIT
: The behavior of temporary tables at the end of a transaction block can be controlled using ON COMMIT. The three options are:

  PRESERVE ROWS
  : No special action is taken at the ends of transactions. This is the default behavior.

  DELETE ROWS
  : All rows in the temporary table will be deleted at the end of each transaction block. Essentially, an automatic **TRUNCATE** is done at each commit.

  DROP
  : The temporary table will be dropped at the end of the current transaction block.

TABLESPACE *tablespace_name*
: The *tablespace_name* is the name of the tablespace in which the new table is to be created. If not specified, default_tablespace is consulted, or temp_tablespaces if the table is temporary.

query
: A **SELECT**, **TABLE**, or **VALUES** command, or an **EXECUTE** command that runs a prepared **SELECT**, **TABLE**, or **VALUES** query.

WITH [NO] DATA
: This clause specifies whether or not the data produced by the query should be copied into the new table. If not, only the table structure is copied. The default is to copy the data.

NOTES

This command is functionally similar to SELECT INTO (**SELECT_INTO**(7)), but it is preferred since it is less likely to be confused with other uses of the **SELECT INTO** syntax. Furthermore, **CREATE TABLE AS** offers a superset of the functionality offered by **SELECT INTO**.

EXAMPLES

Create a new table films_recent consisting of only recent entries from the table films:

```
CREATE TABLE films_recent AS
  SELECT * FROM films WHERE date_prod >= '2002−01−01';
```

To copy a table completely, the short form using the TABLE command can also be used:

```
CREATE TABLE films2 AS
  TABLE films;
```

Create a new temporary table films_recent, consisting of only recent entries from the table films, using a prepared statement. The new table will be dropped at commit:

```
PREPARE recentfilms(date) AS
  SELECT * FROM films WHERE date_prod > $1;
CREATE TEMP TABLE films_recent ON COMMIT DROP AS
  EXECUTE recentfilms('2002−01−01');
```

COMPATIBILITY

CREATE TABLE AS conforms to the SQL standard. The following are nonstandard extensions:

- The standard requires parentheses around the subquery clause; in PostgreSQL, these parentheses are optional.
- In the standard, the WITH [NO] DATA clause is required; in PostgreSQL it is optional.

- PostgreSQL handles temporary tables in a way rather different from the standard; see CREATE TABLE (**CREATE_TABLE**(7)) for details.
- The WITH clause is a PostgreSQL extension; storage parameters are not in the standard.
- The PostgreSQL concept of tablespaces is not part of the standard. Hence, the clause TABLESPACE is an extension.

SEE ALSO

CREATE MATERIALIZED VIEW (**CREATE_MATERIALIZED_VIEW**(7)), CREATE TABLE (**CREATE_TABLE**(7)), **EXECUTE**(7), **SELECT**(7), SELECT INTO (**SELECT_INTO**(7)), **VALUES**(7)