Unknown Title

⋮ 13/11/2025

GRANT

GRANT — define access privileges

Synopsis

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER
| MAINTAIN }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ]
```

```
table_name
```

```
[, ...]
         | ALL TABLES IN SCHEMA
```

```
schema_name
```

```
[, ...] }
    TO
```

```
role_specification
```

```
[, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
```

```
role_specification
```

```
]
```

```
GRANT { { SELECT | INSERT | UPDATE | REFERENCES } (
```

```
column_name
```

```
[, ...] )
    [, ...] | ALL [ PRIVILEGES ] (
```

```
column_name
```

```
[, ...] ) }
    ON [ TABLE ]
```

```
table_name
```

```
[, ...]
```

```
    TO
role_specification
[, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
role_specification
]

GRANT { { USAGE | SELECT | UPDATE }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { SEQUENCE
sequence_name
[, ...]
         | ALL SEQUENCES IN SCHEMA
schema_name
[, ...] }
    TO
role_specification
[, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
role_specification
]

GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...] | ALL [ PRIVILEGES ]
}
    ON DATABASE
database_name
[, ...]
    TO
role_specification
[, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
role_specification
]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN
domain_name
[, ...]
    TO
role_specification
```

```
 [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
 role_specification
 ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER
 fdw_name
 [, ...]
    TO
 role_specification
 [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
 role_specification
 ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER
 server_name
 [, ...]
    TO
 role_specification
 [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
 role_specification
 ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { { FUNCTION | PROCEDURE | ROUTINE }
 routine_name
 [ ( [ [
 argmode
 ] [
 arg_name
 ]
 arg_type
 [, ...] ] ) ] [, ...]
        | ALL { FUNCTIONS | PROCEDURES | ROUTINES } IN SCHEMA
 schema_name
 [, ...] }
    TO
 role_specification
 [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
 role_specification
 ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE
 lang_name
 [, ...]
    TO
 role_specification
 [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
 role_specification
 ]

GRANT { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT
 loid
 [, ...]
    TO
 role_specification
 [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
 role_specification
 ]

GRANT { { SET | ALTER SYSTEM } [, ... ] | ALL [ PRIVILEGES ] }
    ON PARAMETER
 configuration_parameter
 [, ...]
    TO
 role_specification
 [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
 role_specification
 ]
```

```
GRANT { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA
 schema_name
 [, ...]
    TO
 role_specification
 [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
 role_specification
 ]

GRANT { CREATE | ALL [ PRIVILEGES ] }
    ON TABLESPACE
 tablespace_name
 [, ...]
    TO
 role_specification
 [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
 role_specification
 ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON TYPE
 type_name
 [, ...]
    TO
 role_specification
 [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY
 role_specification
 ]

GRANT
 role_name
 [, ...] TO
 role_specification
 [, ...]
    [ WITH { ADMIN | INHERIT | SET } { OPTION | TRUE | FALSE } ]
    [ GRANTED BY
role_specification
 ]

where
role_specification
 can be:

    [ GROUP ]
role_name

  | PUBLIC
  | CURRENT_ROLE
  | CURRENT_USER
  | SESSION_USER
```

Description

The GRANT command has two basic variants: one that grants privileges on a database object (table, column, view, foreign table, sequence, database, foreign-data wrapper, foreign server, function, procedure, procedural language, large object, configuration parameter, schema, tablespace, or type), and one that grants membership in a role. These variants are similar in many ways, but they are different enough to be described separately.

GRANT on Database Objects

This variant of the GRANT command gives specific privileges on a database object to one or more roles. These privileges are added to those already granted, if any.

The key word PUBLIC indicates that the privileges are to be granted to all roles, including those that might be created later. PUBLIC can be thought of as an implicitly defined group that always includes all roles. Any particular role will have the sum of privileges granted directly to it, privileges granted to any role it is presently a member of, and privileges granted to PUBLIC.

If WITH GRANT OPTION is specified, the recipient of the privilege can in turn grant it to others. Without a grant option, the recipient cannot do that. Grant options cannot be granted to PUBLIC.

If GRANTED BY is specified, the specified grantor must be the current user. This clause is currently present in this form only for SQL compatibility.

There is no need to grant privileges to the owner of an object (usually the user that created it), as the owner has all privileges by default. (The owner could, however, choose to revoke some of their own privileges for safety.)

The right to drop an object, or to alter its definition in any way, is not treated as a grantable privilege; it is inherent in the owner, and cannot be granted or revoked. (However, a similar effect can be obtained by granting or revoking membership in the role that owns the object; see below.) The owner implicitly has all grant options for the object, too.

The possible privileges are:

`SELECT`
`INSERT`
`UPDATE`
`DELETE`
`TRUNCATE`
`REFERENCES`
`TRIGGER`
`CREATE`
`CONNECT`
`TEMPORARY`
`EXECUTE`
`USAGE`
`SET`
`ALTER SYSTEM`
`MAINTAIN`

> Specific types of privileges, as defined in Section 5.8.

`TEMP`

> Alternative spelling for `TEMPORARY`.

`ALL PRIVILEGES`

> Grant all of the privileges available for the object's type. The `PRIVILEGES` key word is optional in PostgreSQL, though it is required by strict SQL.

The `FUNCTION` syntax works for plain functions, aggregate functions, and window functions, but not for procedures; use `PROCEDURE` for those. Alternatively, use `ROUTINE` to refer to a function, aggregate function, window function, or procedure regardless of its precise type.

There is also an option to grant privileges on all objects of the same type within one or more schemas. This functionality is currently supported only for tables, sequences, functions, and procedures. `ALL TABLES` also affects views and foreign tables, just like the specific-object `GRANT` command. `ALL FUNCTIONS` also affects aggregate and window functions, but not procedures, again just like the specific-object `GRANT` command. Use `ALL ROUTINES` to include procedures.

GRANT on Roles

This variant of the `GRANT` command grants membership in a role to one or more other roles, and the modification of membership options `SET`, `INHERIT`, and `ADMIN`; see Section 21.3 for details.

Membership in a role is significant because it potentially allows access to the privileges granted to a role to each of its members, and potentially also the ability to make changes to the role itself. However, the actual permissions conferred depend on the options associated with the grant. To modify that options of an existing membership, simply specify the membership with updated option values.

Each of the options described below can be set to either `TRUE` or `FALSE`. The keyword `OPTION` is accepted as a synonym for `TRUE`, so that `WITH ADMIN OPTION` is a synonym for `WITH ADMIN TRUE`. When altering an existing membership the omission of an option results in the current value being retained.

The `ADMIN` option allows the member to in turn grant membership in the role to others, and revoke membership in the role as well. Without the admin option, ordinary users cannot do that. A role is not considered to hold `WITH ADMIN OPTION` on itself. Database superusers can grant or revoke membership in any role to anyone. This option defaults to `FALSE`.

The `INHERIT` option controls the inheritance status of the new membership; see Section 21.3 for details on inheritance. If it is set to `TRUE`, it causes the new member to inherit from the granted role. If set to `FALSE`, the new member does not inherit. If unspecified when creating a new role membership, this defaults to the inheritance attribute of the new member.

The `SET` option, if it is set to `TRUE`, allows the member to change to the granted role using the `SET ROLE` command. If a role is an indirect member of another role, it can use `SET ROLE` to change to that role only if there is a chain of grants each of which has `SET TRUE`. This option defaults to `TRUE`.

To create an object owned by another role or give ownership of an existing object to another role, you must have the ability to `SET ROLE` to that role; otherwise, commands such as `ALTER ... OWNER TO` or `CREATE DATABASE ... OWNER` will fail. However, a user who inherits the privileges of a role but does not have the ability to `SET ROLE` to that role may be able to obtain full access to the role by manipulating existing objects owned by that role (e.g. they could redefine an existing function to act as a Trojan horse). Therefore, if a role's privileges are to be inherited but should not be accessible via `SET ROLE`, it should not own any SQL objects.

If `GRANTED BY` is specified, the grant is recorded as having been done by the specified role. A user can only attribute a grant to another role if they possess the privileges of that role. The role recorded as the grantor must have `ADMIN OPTION` on the target role, unless it is the bootstrap superuser. When a grant is recorded as having a grantor other than the bootstrap superuser, it depends on the grantor continuing to possess `ADMIN OPTION` on the role; so, if `ADMIN OPTION` is revoked, dependent grants must be revoked as well.

Unlike the case with privileges, membership in a role cannot be granted to `PUBLIC`. Note also that this form of the command does not allow the noise word `GROUP` in *`role_specification`*.

Notes

The `REVOKE` command is used to revoke access privileges.

Since PostgreSQL 8.1, the concepts of users and groups have been unified into a single kind of entity called a role. It is therefore no longer necessary to use the keyword GROUP to identify whether a grantee is a user or a group. GROUP is still allowed in the command, but it is a noise word.

A user may perform SELECT, INSERT, etc. on a column if they hold that privilege for either the specific column or its whole table. Granting the privilege at the table level and then revoking it for one column will not do what one might wish: the table-level grant is unaffected by a column-level operation.

When a non-owner of an object attempts to GRANT privileges on the object, the command will fail outright if the user has no privileges whatsoever on the object. As long as some privilege is available, the command will proceed, but it will grant only those privileges for which the user has grant options. The GRANT ALL PRIVILEGES forms will issue a warning message if no grant options are held, while the other forms will issue a warning if grant options for any of the privileges specifically named in the command are not held. (In principle these statements apply to the object owner as well, but since the owner is always treated as holding all grant options, the cases can never occur.)

It should be noted that database superusers can access all objects regardless of object privilege settings. This is comparable to the rights of root in a Unix system. As with root, it's unwise to operate as a superuser except when absolutely necessary.

If a superuser chooses to issue a GRANT or REVOKE command, the command is performed as though it were issued by the owner of the affected object. In particular, privileges granted via such a command will appear to have been granted by the object owner. (For role membership, the membership appears to have been granted by the bootstrap superuser.)

GRANT and REVOKE can also be done by a role that is not the owner of the affected object, but is a member of the role that owns the object, or is a member of a role that holds privileges WITH GRANT OPTION on the object. In this case the privileges will be recorded as having been granted by the role that actually owns the object or holds the privileges WITH GRANT OPTION. For example, if table t1 is owned by role g1, of which role u1 is a member, then u1 can grant privileges on t1 to u2, but those privileges will appear to have been granted directly by g1. Any other member of role g1 could revoke them later.

If the role executing GRANT holds the required privileges indirectly via more than one role membership path, it is unspecified which containing role will be recorded as having done the grant. In such cases it is best practice to use SET ROLE to become the specific role you want to do the GRANT as.

Granting permission on a table does not automatically extend permissions to any sequences used by the table, including sequences tied to SERIAL columns. Permissions on sequences must be set separately.

See Section 5.8 for more information about specific privilege types, as well as how to inspect objects' privileges.

Examples

Grant insert privilege to all users on table films:

```
GRANT INSERT ON films TO PUBLIC;
```

Grant all available privileges to user `manuel` on view `kinds`:

```
GRANT ALL PRIVILEGES ON kinds TO manuel;
```

Note that while the above will indeed grant all privileges if executed by a superuser or the owner of `kinds`, when executed by someone else it will only grant those permissions for which the someone else has grant options.

Grant membership in role `admins` to user `joe`:

```
GRANT admins TO joe;
```

Compatibility

According to the SQL standard, the `PRIVILEGES` key word in `ALL PRIVILEGES` is required. The SQL standard does not support setting the privileges on more than one object per command.

PostgreSQL allows an object owner to revoke their own ordinary privileges: for example, a table owner can make the table read-only to themselves by revoking their own `INSERT`, `UPDATE`, `DELETE`, and `TRUNCATE` privileges. This is not possible according to the SQL standard. The reason is that PostgreSQL treats the owner's privileges as having been granted by the owner to themselves; therefore they can revoke them too. In the SQL standard, the owner's privileges are granted by an assumed entity "_SYSTEM". Not being "_SYSTEM", the owner cannot revoke these rights.

According to the SQL standard, grant options can be granted to `PUBLIC`; PostgreSQL only supports granting grant options to roles.

The SQL standard allows the `GRANTED BY` option to specify only `CURRENT_USER` or `CURRENT_ROLE`. The other variants are PostgreSQL extensions.

The SQL standard provides for a `USAGE` privilege on other kinds of objects: character sets, collations, translations.

In the SQL standard, sequences only have a `USAGE` privilege, which controls the use of the `NEXT VALUE FOR` expression, which is equivalent to the function `nextval` in PostgreSQL. The sequence privileges `SELECT` and `UPDATE` are PostgreSQL extensions. The application of the sequence `USAGE` privilege to the `currval` function is also a PostgreSQL extension (as is the function itself).

Privileges on databases, tablespaces, schemas, languages, and configuration parameters are PostgreSQL extensions.