Unknown Title

⋮ 13/11/2025

CREATE ROLE

CREATE ROLE — define a new database role

Synopsis

```
CREATE ROLE
```
name
```
[ [ WITH ]
```
option
```
[ ... ] ]
```

where
option
can be:

```
      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | BYPASSRLS | NOBYPASSRLS
    | CONNECTION LIMIT
```
connlimit
```

    | [ ENCRYPTED ] PASSWORD '
```
password
```
' | PASSWORD NULL
    | VALID UNTIL '
```
timestamp
```
'
    | IN ROLE
```
role_name

```
[, ...]
    | ROLE
```

role_name

```
[, ...]
    | ADMIN
```

role_name

```
[, ...]
    | SYSID
```

uid

Description

`CREATE ROLE` adds a new role to a PostgreSQL database cluster. A role is an entity that can own database objects and have database privileges; a role can be considered a “user”, a “group”, or both depending on how it is used. Refer to Chapter 21 and Chapter 20 for information about managing users and authentication. You must have `CREATEROLE` privilege or be a database superuser to use this command.

Note that roles are defined at the database cluster level, and so are valid in all databases in the cluster.

During role creation it is possible to immediately assign the newly created role to be a member of an existing role, and also assign existing roles to be members of the newly created role. The rules for which initial role membership options are enabled are described below in the `IN ROLE`, `ROLE`, and `ADMIN` clauses. The GRANT command has fine-grained option control during membership creation, and the ability to modify these options after the new role is created.

Parameters

name

The name of the new role.

`SUPERUSER`
`NOSUPERUSER`

These clauses determine whether the new role is a “superuser”, who can override all access restrictions within the database. Superuser status is dangerous and should be used only when really needed. You must yourself be a superuser to create a new superuser. If not specified, `NOSUPERUSER` is the default.

`CREATEDB`
`NOCREATEDB`

These clauses define a role's ability to create databases. If `CREATEDB` is specified, the role being defined will be allowed to create new databases. Specifying `NOCREATEDB` will deny a role the

ability to create databases. If not specified, NOCREATEDB is the default. Only superuser roles or roles with CREATEDB can specify CREATEDB.

CREATEROLE
NOCREATEROLE

These clauses determine whether a role will be permitted to create, alter, drop, comment on, and change the security label for other roles. See role creation for more details about what capabilities are conferred by this privilege. If not specified, NOCREATEROLE is the default.

INHERIT
NOINHERIT

This affects the membership inheritance status when this role is added as a member of another role, both in this and future commands. Specifically, it controls the inheritance status of memberships added with this command using the IN ROLE clause, and in later commands using the ROLE clause. It is also used as the default inheritance status when adding this role as a member using the GRANT command. If not specified, INHERIT is the default.

In PostgreSQL versions before 16, inheritance was a role-level attribute that controlled all runtime membership checks for that role.

LOGIN
NOLOGIN

These clauses determine whether a role is allowed to log in; that is, whether the role can be given as the initial session authorization name during client connection. A role having the LOGIN attribute can be thought of as a user. Roles without this attribute are useful for managing database privileges, but are not users in the usual sense of the word. If not specified, NOLOGIN is the default, except when CREATE ROLE is invoked through its alternative spelling CREATE USER.

REPLICATION
NOREPLICATION

These clauses determine whether a role is a replication role. A role must have this attribute (or be a superuser) in order to be able to connect to the server in replication mode (physical or logical replication) and in order to be able to create or drop replication slots. A role having the REPLICATION attribute is a very highly privileged role, and should only be used on roles actually used for replication. If not specified, NOREPLICATION is the default. Only superuser roles or roles with REPLICATION can specify REPLICATION.

BYPASSRLS
NOBYPASSRLS

These clauses determine whether a role bypasses every row-level security (RLS) policy. NOBYPASSRLS is the default. Only superuser roles or roles with BYPASSRLS can specify BYPASSRLS.

Note that pg_dump will set `row_security` to `OFF` by default, to ensure all contents of a table are dumped out. If the user running pg_dump does not have appropriate permissions, an error will be returned. However, superusers and the owner of the table being dumped always bypass RLS.

`CONNECTION LIMIT` *connlimit*

If role can log in, this specifies how many concurrent connections the role can make. -1 (the default) means no limit. Note that only normal connections are counted towards this limit. Neither prepared transactions nor background worker connections are counted towards this limit.

`[ ENCRYPTED ] PASSWORD` '*password*'
`PASSWORD NULL`

Sets the role's password. (A password is only of use for roles having the `LOGIN` attribute, but you can nonetheless define one for roles without it.) If you do not plan to use password authentication you can omit this option. If no password is specified, the password will be set to null and password authentication will always fail for that user. A null password can optionally be written explicitly as `PASSWORD NULL`.

The password is always stored encrypted in the system catalogs. The `ENCRYPTED` keyword has no effect, but is accepted for backwards compatibility. The method of encryption is determined by the configuration parameter password_encryption. If the presented password string is already in MD5-encrypted or SCRAM-encrypted format, then it is stored as-is regardless of `password_encryption` (since the system cannot decrypt the specified encrypted password string, to encrypt it in a different format). This allows reloading of encrypted passwords during dump/restore.

Note

Specifying an empty string will also set the password to null, but that was not the case before PostgreSQL version 10. In earlier versions, an empty string could be used, or not, depending on the authentication method and the exact version, and libpq would refuse to use it in any case. To avoid the ambiguity, specifying an empty string should be avoided.

Warning

Support for MD5-encrypted passwords is deprecated and will be removed in a future release of PostgreSQL. Refer to Section 20.5 for details about migrating to another password type.

`VALID UNTIL` '*timestamp*'

The `VALID UNTIL` clause sets a date and time after which the role's password is no longer valid. If this clause is omitted the password will be valid for all time.

`IN ROLE` *role_name*

The `IN ROLE` clause causes the new role to be automatically added as a member of the specified existing roles. The new membership will have the `SET` option enabled and the `ADMIN` option disabled. The `INHERIT` option will be enabled unless the `NOINHERIT` option is specified.

`ROLE` *role_name*

The `ROLE` clause causes one or more specified existing roles to be automatically added as members, with the `SET` option enabled. This in effect makes the new role a “group”. Roles named in this clause with the role-level `INHERIT` attribute will have the `INHERIT` option enabled in the new membership. New memberships will have the `ADMIN` option disabled.

`ADMIN` *`role_name`*

The `ADMIN` clause has the same effect as `ROLE`, but the named roles are added as members of the new role with `ADMIN` enabled, giving them the right to grant membership in the new role to others.

`SYSID` *`uid`*

The `SYSID` clause is ignored, but is accepted for backwards compatibility.

Notes

Use `ALTER ROLE` to change the attributes of a role, and `DROP ROLE` to remove a role. All the attributes specified by `CREATE ROLE` can be modified by later `ALTER ROLE` commands.

The preferred way to add and remove members of roles that are being used as groups is to use `GRANT` and `REVOKE`.

The `VALID UNTIL` clause defines an expiration time for a password only, not for the role per se. In particular, the expiration time is not enforced when logging in using a non-password-based authentication method.

The role attributes defined here are non-inheritable, i.e., being a member of a role with, e.g., `CREATEDB` will not allow the member to create new databases even if the membership grant has the `INHERIT` option. Of course, if the membership grant has the `SET` option the member role would be able to `SET ROLE` to the createdb role and then create a new database.

The membership grants created by the `IN ROLE`, `ROLE`, and `ADMIN` clauses have the role executing this command as the grantor.

The `INHERIT` attribute is the default for reasons of backwards compatibility: in prior releases of PostgreSQL, users always had access to all privileges of groups they were members of. However, `NOINHERIT` provides a closer match to the semantics specified in the SQL standard.

PostgreSQL includes a program createuser that has the same functionality as `CREATE ROLE` (in fact, it calls this command) but can be run from the command shell.

The `CONNECTION LIMIT` option is only enforced approximately; if two new sessions start at about the same time when just one connection “slot” remains for the role, it is possible that both will fail. Also, the limit is never enforced for superusers.

Caution must be exercised when specifying an unencrypted password with this command. The password will be transmitted to the server in cleartext, and it might also be logged in the client's command history or the server log. The command createuser, however, transmits the password encrypted. Also, psql contains a command `\password` that can be used to safely change the password later.

Examples

Create a role that can log in, but don't give it a password:

```
CREATE ROLE jonathan LOGIN;
```

Create a role with a password:

```
CREATE USER davide WITH PASSWORD 'jw8s0F4';
```

(`CREATE USER` is the same as `CREATE ROLE` except that it implies `LOGIN`.)

Create a role with a password that is valid until the end of 2004. After one second has ticked in 2005, the password is no longer valid.

```
CREATE ROLE miriam WITH LOGIN PASSWORD 'jw8s0F4' VALID UNTIL '2005-01-01';
```

Create a role that can create databases and manage roles:

```
CREATE ROLE admin WITH CREATEDB CREATEROLE;
```

Compatibility

The `CREATE ROLE` statement is in the SQL standard, but the standard only requires the syntax

```
CREATE ROLE
 name
 [ WITH ADMIN
 role_name
 ]
```

Multiple initial administrators, and all the other options of `CREATE ROLE`, are PostgreSQL extensions.

The SQL standard defines the concepts of users and roles, but it regards them as distinct concepts and leaves all commands defining users to be specified by each database implementation. In PostgreSQL we have chosen to unify users and roles into a single kind of entity. Roles therefore have many more optional attributes than they do in the standard.

The behavior specified by the SQL standard is most closely approximated creating SQL-standard users as PostgreSQL roles with the `NOINHERIT` option, and SQL-standard roles as PostgreSQL roles with the `INHERIT` option.

The `USER` clause has the same behavior as `ROLE` but has been deprecated:

```
USER
 role_name
 [, ...]
```

The `IN GROUP` clause has the same behavior as `IN ROLE` but has been deprecated:

```
IN GROUP
 role_name
```

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