Learning Objectives

Learners will be able to...

- Describe the PostgreSQL role system
- Enumerate the list of available role attributes
- Create and remove roles in a database
- List the available actions on objects associated with their privileges
- Add and remove a privilege to an object

info

Make Sure You Know

Understand how user authentication and access control works in a database management system.

Roles

Security in PostgreSQL also involves creating roles that allow access to server resources and then granting appropriate privileges on database objects.

In PostgreSQL, users and groups are **roles**. One role can be a member of another role. Roles in PostgreSQL have no relationship to users in the operating system. Roles are global objects for the entire database cluster.

> info
>
> ## Role
>
> A role can be thought of as either a database user, or a group of database users, depending on how the role is set up.

Superuser

Just like in operating systems, the database has the concept of a superuser.

When a user is a superuser, no permissions are checked when executing SQL code or administering the database. Only permission to enter and connect to it is checked.

To function properly, each PostgreSQL instance must have at least one superuser to perform administrative tasks.

In PostgreSQL, this initial superuser account is named `postgres` by default.

It is not recommended to create many superuser accounts, each created role should have the minimum required set to perform the functions possible on them, but no more.

Role attributes

Each role has a set of attributes that indicate what actions it can perform.

Attribute	Description(items in bold are default)
SUPERUSER/**NOSUPERUSER**	Shows if the user is a superuser
CREATEDB/**NOCREATEDB**	Reflects whether the user can create databases

LOGIN/**NOLOGIN**	These items determine whether the role is allowed to log on
INHERIT/NOINHERIT	These clauses determine whether a role “inherits” the privileges of roles it is a member of
CREATEROLE/**NOCREATEROLE**	Indicates whether the given role can create new roles. A superuser can only be created by a superuser

Every PostgreSQL database has another implicit role called **PUBLIC** which cannot be deleted. All other roles are always granted membership in PUBLIC by default and inherit whatever privileges are currently assigned to it.

By default, this role grants the following privileges(depends on version):
- CONNECT
- TEMPORARY
- CREATE
- EXECUTE (functions and procedures)
- USAGE (domains, languages, and types)

Roles manipulation

List roles

You can view the list of available roles on the PostgreSQL command line using the `\du` command.

```
codio=# \du
                                   List of roles
 Role name |                         Attributes
| Member of
-----------+------------------------------------------------------
--------+-----------
 codio     | Superuser, Create role, Create DB
| {}
 postgres  | Superuser, Create role, Create DB, Replication,
Bypass RLS | {}
```

Create new role

To create a PostgreSQL user, use the following SQL statement:

```
CREATE USER my_user WITH PASSWORD 'secret_passwd';
```

You can also create a user with the following SQL statement:

```
CREATE ROLE my_user WITH LOGIN PASSWORD 'secret_passwd';
```

Both of these statements create the exact same user. This new user does not have any permissions other than the default permissions available to the PUBLIC role.

See the documentation for a complete list of options.

Update role

To change the attributes of an already created role, use the ALTER ROLE command. The syntax for this command is:

```
ALTER ROLE role_name WITH attribute_options;
```

For example, we can add the ability to create databases with the following command.

```
ALTER ROLE my_user WITH CREATEDB;
```

Delete role

To remove the role, you can use the following command.

```
DROP ROLE role_name;
```

Delete the previously created role with the following command.

```
DROP ROLE my_user;
```

Roles In Action

Create new two users in the console.

```
CREATE ROLE simple_user WITH LOGIN PASSWORD 'secret_passwd';
CREATE ROLE create_db_user WITH CREATEDB LOGIN PASSWORD
        'secret_passwd';
```

Since we are initially logged in as the superuser, we can change roles to other users at run time using the role change command - `SET ROLE`.

Log in initially as `simple_user` and try to display the contents of the table `actor` - an permission denied error will occur. The same error will occur when trying to create objects, such as a database.

```
codio=> SET ROLE simple_user;
SET
codio=> SELECT * FROM actor LIMIT 10;
ERROR:  permission denied for table actor
codio=> CREATE DATABASE test;
ERROR:  permission denied to create database
```

The database creation failed because the user does not have enough permissions, and he cannot retrieve data from existing tables because he is not assigned the rights to do so.

The situation with the second user(`create_db_user`) will be slightly different - he will already be able to create a database, but will not be able to retrieve data from the existing one.

```
codio=# SET ROLE create_db_user;
SET
codio=> SELECT * FROM actor LIMIT 10;
ERROR:  permission denied for table actor
codio=> CREATE DATABASE test;
CREATE DATABASE
```

info

Objects owner

The second user, after creating the database, can switch to it and create other objects in it - tables, indexes, and so on. And he will already be able to perform all operations on these objects - because he will be their owner and he does not need additional permissions.

In addition to using the role switch command, you can use the psql client to connect as a specific existing user. (Don’t forget to close the existing session with the `\q` command.)

```
psql -h localhost -U simple_user codio
```

Granting privileges

To allow the user role to interact with database objects, you need to grant privileges on the database objects to the user role by using the GRANT statement.

Simplified syntax will looks like:

```
GRANT privilege_list | ALL
ON  table_name
TO  role_name;
```

To give permission to select data for `simple_user`, you can use the following command

```
GRANT SELECT ON TABLE actor TO simple_user;
```

And test it

```
SET ROLE simple_user; -- switch user
SELECT * FROM actor LIMIT 10; -- run select query
SET ROLE NONE; -- switch back
```

But you still won't be able to do the rest of the actions:

```
SET ROLE simple_user; -- switch user
INSERT INTO actor (first_name, last_name) VALUES('Jackie',
        'Chan');
SET ROLE NONE; -- switch back
```

Will result error.

```
SET
ERROR:  permission denied for table actor
SET
```

Access to all tables could be given by command:

```
GRANT SELECT ON ALL TABLES IN SCHEMA public TO simple_user;
```

Privileges

info

Principle of Least Privilege

When assigning rights, the principle of least privilege (PoLP), an information security methodology, should be applied, according to which users should be granted access to only the smallest amount of information necessary to perform their job or task. Any access outside of the files or data they own must be granted to them specifically.

The available privileges are:

SELECT - Allows SELECT from any column, or specific column(s), of a table, view, materialized view, or other table-like object.

INSERT - Allows INSERT of a new row into a table, view, etc.

UPDATE - Allows UPDATE of any column, or specific column(s), of a table, view, etc.

DELETE - Allows DELETE of a row from a table, view, etc.

TRUNCATE - Allows TRUNCATE on a table.

REFERENCES - Allows creation of a foreign key constraint referencing a table, or specific column(s) of a table.

TRIGGER - Allows creation of a trigger on a table, view, etc.

CREATE - For databases, allows new schemas and publications to be created within the database, and allows trusted extensions to be installed within the database.

Full list and description could be reviewed in documentation.

Revoking privileges

The **REVOKE** command revokes previously granted privileges from one or more roles. The key word PUBLIC refers to the implicitly defined group of all roles.

Simplified syntax will looks like:

```
REVOKE privilege | ALL
ON TABLE table_name | ALL TABLES IN SCHEMA schema_name
FROM role_name;
```

Example:

```
REVOKE SELECT ON TABLE actor FROM simple_user;
```

Ore remove all privileges from all objects in current schema.

```
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM simple_user;
```