PostgreSQL Roles and Privileges Simplified

PostgreSQL has a fine grained system for for managing user roles and privileges. This helps admins decide who can access certain data and what they're allowed to do with it. It's about managing permissions, where you can create different roles and roles can be a member of other roles.

In this blog, we will dive into PostgreSQL user roles and permissions, covering databases, schemas, and other objects level privileges, following a FAQ format.

What is the Difference between user, roles and groups?

In PostgreSQL, users, groups, and roles are essentially the same, but users have the added ability to log in by default. The commands CREATE USER and CREATE GROUP are the same as CREATE ROLE.

The image part with relationship ID rId4 was not found in the file.

Source: AWS

Note: When you use the CREATE ROLE command, the created role doesn't have login privileges by default. In our blog, we'll be using the terms role and user interchangeably.

In PostgreSQL, every connection to the database server is linked to a specific role. Initially, the connection passes through the authentication module (**pg_hba.conf**), and then this role decides the initial access privileges for commands executed during that connection.

Roles attributes

Roles have various attributes that define what they can do within the database system. These attributes are important because they determine the extent of actions a role can take. Here are some key attributes and what they mean:

Attribute	Description
LOGIN	This attribute enables the role to authenticate and connect to the database cluster. Roles without this attribute are unable to establish connections, limiting their access to the database.
PASSWORD	Enables the role to authenticate using a password for secure access to the database. All PostgreSQL users must have a password for authentication.
SUPERUSER	Provides the highest level of privileges within the database cluster. Superusers can do anything and everything such as access all databases, perform any operation, and manage other roles. In PostgreSQL this initial superuser account is called `postgres` by default, which typically matches the system username that is running the PostgreSQL cluster process.

CREATEDB	Allow role to create new databases within the PostgreSQL cluster.
CREATEROLE	Allow role the ability to create new roles within the database system and manage them.
INHERIT	This attribute decides if a role can inherit privileges from other roles. When activated, the role gains the privileges of roles it's a part of. If **NOINHERIT** is specified, the role must use the SET command to access the privileges of the parent role.

How to create and assign a role?

```
CREATE ROLE readonly; => This role cannot login inside databaseCopy to Clipboard
```

CREATE ROLE testuser LOGIN ENCRYPTED PASSWORD ‘SECRET’; => This role can login and access databases inside the cluster.

GRANT readonly to testuser; => All privileges assigned to readonly role with be inherit by testuser

What is a PUBLIC role?

The **PUBLIC** role in PostgreSQL is a special role that grants a privilege to every role on the system. It’s like a default group that indicates all roles, even those created later. It ensures that all roles have access to certain privileges. All other roles are automatically granted membership in PUBLIC by default and inherit its privileges.

Before PostgreSQL 15, every user had full privileges (create/usage) on the public schema, which was shared among all users. However, starting with PostgreSQL 15, users no longer have the ability to create any objects within the public schema. This change applies to the public role, which represents all users. It’s now standard practice to revoke the CREATE privilege for the public schema.

What database level privileges are provided in PostgreSQL?

PostgreSQL provides various privileges at the database level, including:

Privilege	Description
CONNECT	Allows users to connect to a specific database.
CREATE	Grants the ability to create new objects within the database.
TEMPORARY	Permits the creation of temporary tables.

By default, the **CONNECT** privilege is granted to the PUBLIC role (all users). However, it’s not possible to revoke this privilege from individual users directly. Instead, you would need to revoke it from the PUBLIC role and then grant it individually to other users who require it.

```
REVOKE CONNECT ON DATABASE postgres FROM PUBLIC;Copy to Clipboard
```

Now connect testuser with postgres database

```
psql -U testuser -d postgres

psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed:
FATAL:  permission denied for database "postgres"DETAIL:  User does not have CONNECT
privilege.Copy to Clipboard
```

How to assign schema level privileges?

PostgreSQL database can consist of one or more schemas, which contains tables and other objects like views, data types, functions, procedures and operators. Different schemas can have objects with the same name without conflict. For instance, both schema1 and schema2 can contain tables named table1. Unlike databases, schemas aren't strictly separated. A user can access objects in any schema within the database they're connected to, provided they have the necessary privileges.

By default, users can't access objects in schemas they don't own. To allow access, the owner of the schema must grant the USAGE privilege on the schema. For instance:

```
GRANT USAGE ON SCHEMA schema1 TO testuser;Copy to Clipboard
```

Users can also be permitted to create objects in someone else's schema. To allow this, the CREATE privilege on the schema must be granted. For example:

```
GRANT CREATE ON SCHEMA schema1 TO abc;Copy to Clipboard
```

In order to provide both privileges in a single command, run the following:

```
GRANT ALL ON SCHEMA schema1 TO testuser;Copy to Clipboard
```

How to assign table level privileges?

PostgreSQL offers various DDL and DML privileges at the table level:

- **SELECT:** Allows fetching data from a table.
- **INSERT:** Enables adding new rows into a table.
- **UPDATE:** Modifying existing rows in a table.
- **DELETE:** Grants the ability to remove rows from a table.
- **REFERENCES:** Allows creating foreign key constraints.
- **TRIGGER:** Enables the creation of triggers on a table.
- **TRUNCATE:** Permits the use of TRUNCATE on a table.

Granting INSERT and UPDATE Privileges:

```
Grant insert,update on table tablename to testuser;Copy to Clipboard
```

Granting All Privileges on a Single Table:

```
Grant all on table tablename to testuser;Copy to Clipboard
```

Granting All Privileges on All Tables in a Schema:

```
Grant all on all tables in schema schemaname to testuser;Copy to Clipboard
```

What is the purpose of the ALTER DEFAULT PRIVILEGE command?

ALTER DEFAULT PRIVILEGES command allow admins to define default access levels for objects created by specific roles within a schema. This is useful for scenarios where **newly created objects** require consistent access controls. By default, newly created objects are only accessible to the owner (the user who created them) unless explicit privileges are granted. For example:

```
ALTER DEFAULT PRIVILEGES FOR ROLE developer IN SCHEMA test GRANT SELECT, INSERT ON TABLES TO
reporting;Copy to Clipboard
```

The above command implies that each time the **developer role** creates any new table inside the **test schema**, the SELECT and INSERT privilege will be automatically granted on that table to the **reporting role**.

Note: It’s important to understand that GRANT permissions are separate for each object. Granting privileges on a database does not automatically extend those rights to the schema within it. Similarly, granting permissions on a schema does not automatically grant rights on the tables within that schema.

When accessing a table, the permission checks occur in a specific order:

1. Do you have USAGE privilege on the schema?

   - If no: Access is denied.
   - If yes: Proceed to the next step.

2. Do you also have the appropriate rights on the table?

   - If no: Access is denied.
   - If yes: Proceed to check column privileges.

This means if you have access on all tables within a schema but lack the USAGE privilege on that schema, PostgreSQL will throw permission denied error.

What are predefined roles in PostgreSQL?

A predefined role refers to a built-in role designed to grant access to commonly needed information. This simplifies the task for administrators, allowing them to easily provide access to others without the need to execute multiple SQL queries for the same grants. These roles are outlined as follows:

Role	Description
pg_read_all_data	Read all data (tables, views, sequences), as if having SELECT rights on those objects, and USAGE rights on all schemas, even without having it explicitly. This role does not have the role attribute BYPASSRLS set. If RLS is being used, an administrator may wish to set BYPASSRLS on roles which this role is GRANTed to.
pg_write_all_data	Write all data (tables, views, sequences), as if having INSERT, UPDATE, and DELETE rights on those objects, and USAGE rights on all schemas, even

	without having it explicitly. This role does not have the role attribute BYPASSRLS set. If RLS is being used, an administrator may wish to set BYPASSRLS on roles which this role is GRANTed to.
pg_read_all_settings	Read all configuration variables, even those normally visible only to superusers.
pg_read_all_stats	Read all pg_stat_* views and use various statistics related extensions, even those normally visible only to superusers.
pg_stat_scan_tables	Execute monitoring functions that may take ACCESS SHARE locks on tables, potentially for a long time.
pg_monitor	Read/execute various monitoring views and functions. This role is a member of pg_read_all_settings, pg_read_all_stats and pg_stat_scan_tables.
pg_database_owner	None. Membership consists, implicitly, of the current database owner.
pg_signal_backend	Signal another backend to cancel a query or terminate its session.
pg_read_server_files	Allow reading files from any location the database can access on the server with COPY and other file-access functions.
pg_write_server_files	Allow writing to files in any location the database can access on the server with COPY and other file-access functions.
pg_execute_server_program	Allow executing programs on the database server as the user the database runs as with COPY and other functions which allow executing a server-side program.
pg_checkpoint	Allow executing the CHECKPOINT command.
pg_use_reserved_connections	Allow use of connection slots reserved via reserved_connections.
pg_create_subscription	Allow users with CREATE permission on the

	database to issue CREATE SUBSCRIPTION.

PostgreSQL 14 has introduced two new roles that offer significant benefits from the perspective of application developers:

pg_read_all_data
pg_write_all_data

For example when providing read-only privileges to an application user, the normal approach involves executing multiple commands for each schema:

```
GRANT SELECT on all tables in schema1 to application_user;

GRANT SELECT on all tables in schema2 to application_user;

ALTER DEFAULT PRIVILEGES in schema1 GRANT select on tables to application_user;

ALTER DEFAULT PRIVILEGES in schema2 GRANT select on tables to application_user;Copy to
Clipboard
```

The issue is that this process becomes repetitive when dealing with numerous schemas and write users.

With a predefined role like **pg_read_all_data**, we only need to run a single command:

```
GRANT pg_read_all_data to application_user;Copy to Clipboard
```

Best practices

The recommended approach for setting up fine-grained access control in PostgreSQL is as follows:

- Create separate users for each application or service. This can help in identifying any connection issues easily and assists in tracking access through audit logs.
- Define multiple roles with specific permissions based on application access requirements. Assign these roles to users accordingly as it helps in streamlining administration and simplifying the process of granting or revoking permissions.
- Grant users and roles only the permissions necessary for their respective tasks.
- Beware of superuser privileges as they possess unrestricted access to the database, presenting a significant security risk if compromised.
- Implement strong password policies for database users to prevent unauthorized access. Rotate passwords periodically.
- Utilize SSL/TLS protocols to encrypt communication between clients and the database server.