PostgreSQL

Module 8

Security

Module Overview

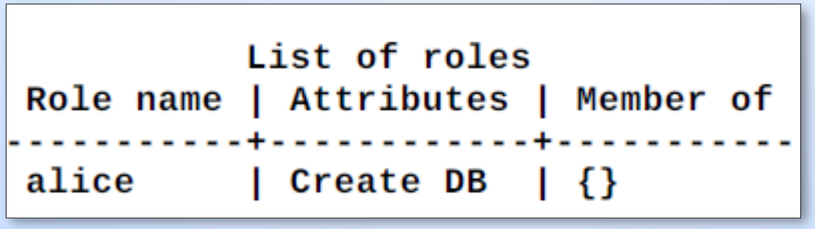
- The Postgresql Superuser
- Creating A New User
- Giving Limited Superuser Powers To Specific Users
- Granting User Access To A Table
- Revoking User Access To A Table
- Temporarily Preventing A User From Connecting
- Removing A User Without Dropping Their Data
- Checking Whether All Users Have A Secure Password
- Always Knowing Which User Is Logged In
- Inspecting permissions
- Connecting using SSL
- Using SSL certificates to authenticate

The Postgresql Superuser

- A PostgreSQL **superuser** is a user that can do anything in the database regardless of what privileges it has been granted.
- A user becomes a superuser when it is created with the SUPERUSER attribute set:
  - **CREATE USER username SUPERUSER;**
- The PostgreSQL system comes set up with at least one superuser. Most commonly, this superuser is named **postgres**.
- In addition to SUPERUSER, there are two lesser attributes—CREATEDB and CREATEUSER—that give the user only some of the power reserved to superusers, namely creating new databases and users.

Creating A New User

- To create new users, you must either be a superuser or have the CREATEROLE or CREATEUSER privilege.
- Create the users by following commands:
  - **CREATE USER bob;**
  - **CREATE USER alice CREATEDB;**
  - You can check the attributes of a given user in psql, as follows:
    - **pguser=# \du alice**



```
           List of roles
 Role name | Attributes | Member of
-----------+------------+-----------
 alice     | Create DB  | {}
```

There's more…

- The **CREATE USER** and **CREATE GROUP** commands are actually variations of **CREATE ROLE**.
- The **CREATE USER username;** statement is equivalent to **CREATE ROLE username LOGIN;**
- The **CREATE GROUP groupname;** statement is equivalent to **CREATE ROLE groupname NOLOGIN;** .

PostgreSQL security levels

- PostgreSQL has different security levels defined on PostgreSQL object
- postgres=# \h GRANT
  - Database security level
    - Disallow users from connecting to the database
    - postgres=# **REVOKE ALL ON DATABASE warehouse FROM public;**
    - To allow the user to connect to the database
    - postgres=# **GRANT CONNECT ON DATABASE warehouse TO test_user;**
  - Schema security level
    - To allow a user access to a certain schema, the usage permissions should be granted:
    - postgres=# **GRANT USAGE ON SCHEMA finance TO test_user, public_user;**

PostgreSQL security levels

- Table-level security
  - The table permissions are INSERT, UPDATE, DELETE, TRIGGER, REFERENCE, and TRUNCATE
  - GRANT ALL ON <table_name> TO <role>;
- Column-level security
  - PostgreSQL allows permissions to be defined on the column level
  - CREATE TABLE test_column_acl(f1 integer, f2 integer);
  - Insert into test_column_acl values (1,2), (3,4);
  - CREATE ROLE test_column_acl login password ‘root’;
  - GRANT SELECT (f1) ON test_column_acl TO test_column_acl;
  - GRANT USAGE ON SCHEMA public TO test_column_acl;
  - \c warehouse test_column_acl
  - SELECT * FROM public.test_column_acl;
  - SELECT f1 FROM public.test_column_acl;

PostgreSQL security levels

- A table has always been shown as a whole
- Row Level security
  - To configure permissions is to come up with policies
  - The CREATE POLICY command is there
  - Example:
    - test=# \c test postgres
    - test=# CREATE TABLE t_person (gender text, name text);
    - test=# INSERT INTO t_person VALUES ('male', 'joe'),('male', 'paul'),('female', 'sarah'),(NULL, 'R2- D2');
    - Then access is granted to the joe role:
      - Test=#Create user joe password ‘root’;
      - test=# GRANT ALL ON t_person TO joe;
      - test=# \c test joe
      - test=> SELECT * FROM t_person;
      - test=# \c test postgres
      - test=# ALTER TABLE t_person ENABLE ROW LEVEL SECURITY;
      - test=# \c test joe
      - test=> SELECT * FROM t_person;

PostgreSQL security levels

```
test=# \c test postgres
test=# CREATE POLICY joe_pol_1
   ON t_person FOR SELECT TO joe
   USING  (gender = 'male');
test=# \c test joe
test=> SELECT * FROM  t_person;
test=# \c test postgres
test=# CREATE POLICY joe_pol_2
 ON t_person FOR SELECT TO joe
 USING  (gender IS NULL);
test=# \c test joe
test=> SELECT * FROM  t_person;
```

Granting User Access To A Table

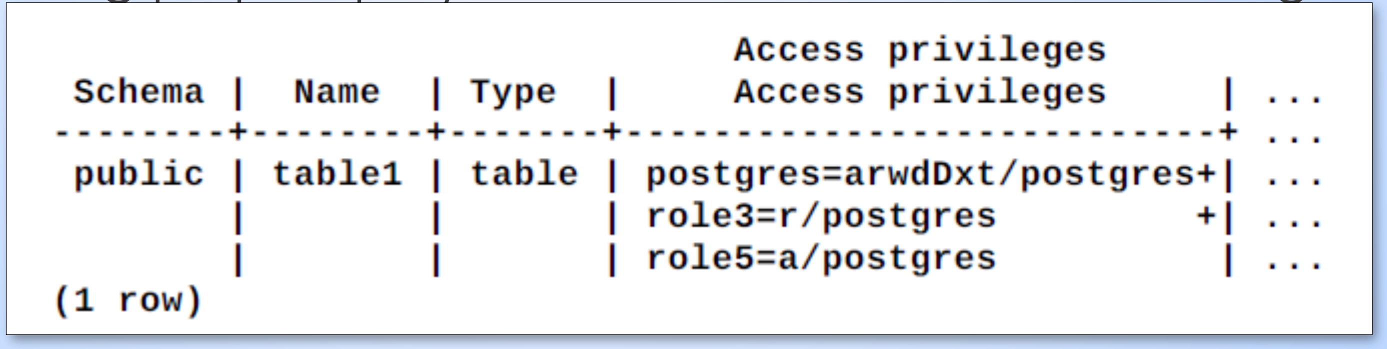
- Granting access to a table through a group role
  - CREATE GROUP webreaders;
  - GRANT SELECT ON sometable TO webreaders;
  - GRANT INSERT ON sometable TO webreaders;
  - GRANT webreaders TO tim, bob;

Granting User Access To A Table

- A user needs to have access to a table in order to perform any action on it.
- Grant access to the schema containing the table, as follows:
  - **GRANT SELECT, INSERT, UPDATE, DELETE ON someschema.sometable TO somerole;**
  - **GRANT somerole TO someuser, otheruser;s**
- There is no requirement in PostgreSQL to have some privileges in order to have others. This means that you may well have "write-only" tables, where you are allowed to insert but you can't select
- Grant access to all objects is schema:
  - **GRANT SELECT ON ALL TABLES IN SCHEMA staging TO bob;**

Revoking User Access To A Table

- The current user must either be a superuser, the owner of the table, or a user with a GRANT option for the table.
  - To revoke all rights on the table1 table from the user2 user, you must run the following SQL command:
    - **REVOKE ALL ON table1 FROM user2;**
    - **REVOKE ALL ON table1 FROM PUBLIC**;
  - Using psql, display the list of roles that have been granted at least one



```
                                      Access privileges
 Schema |  Name   | Type  |       Access privileges      | ...
--------+---------+-------+------------------------------+ ...
 public | table1  | table | postgres=arwdDxt/postgres+| ...
        |         |       | role3=r/postgres          +| ...
        |         |       | role5=a/postgres           | ...
(1 row)
```

Revoking User Access To A Table

- Sample extract from database creation script

```
CREATE TABLE table1(
...
);
REVOKE ALL ON table1 FROM GROUP PUBLIC;
GRANT SELECT ON table1 TO GROUP webreaders;
GRANT SELECT, INSERT, UPDATE, DELETE ON table1 TO editors;
GRANT ALL ON table1 TO admins;
```

Setting Parameters For Particular Groups Of Users

- You can set parameters for each of the following:
  - Database
  - User (which is named **role** by PostgreSQL)
  - Database/user combination
- Define parameter settings for various user groups:
  - For all users in the demo database, use the following commands:
    - **ALTER DATABASE demo SET configuration_parameter = value1;**
  - For a user named simon connected to any database, use this:
    - **ALTER ROLE Simon SET configuration_parameter = value2;**
  - For a user only when connected to a specific database, as follows:
    - **ALTER ROLE Simon IN DATABASE demo SET configuration_parameter = value3;**

Giving Users Their Own Private Database

- Separating data and users is a key part of administration. There will always be a need to give users a private, secure, or simply risk-free area.
- Create a database for a specific user command:
  - **create database fred owner = fred;**
  - As the database owners, users have login privileges, so they can connect to any database by default.
  - We need to revoke the privilege to connect to our new database from everybody except the designated user.
    - **BEGIN;**
    - **REVOKE connect ON DATABASE fred FROM public;**
    - **GRANT connect ON DATABASE fred TO fred;**
    - **COMMIT;**
- Superusers can still connect to the new database, and there is no way to prevent them from doing so.

Preventing New Connections

- In certain emergencies, you may need to lock down the server completely, or just prevent specific users from accessing the database.
- Connections can be prevented in a number of ways, as follows:
  - Stop the server
  - Restrict the connections for a specific database to zero.
    - **ALTER DATABASE foo_db CONNECTION LIMIT 0;**
  - Restrict the connections for a specific user to zero by setting the connection limit to zero.
    - **ALTER USER foo CONNECTION LIMIT 0;**

Restricting Users To Only One Session Each

- We can restrict users to only one connection using the following command:
  - **postgres=# ALTER ROLE fred CONNECTION LIMIT 1;**
- Even if you set the connection limit to zero for superusers, they will still be able to connect.
- If you lower the limit, you should immediately check to see whether there are more sessions connected than the new limit you just set.
  - **postgres=> SELECT rolconnlimit FROM pg_roles WHERE rolname = 'fred';**

Temporarily Preventing A User From Connecting

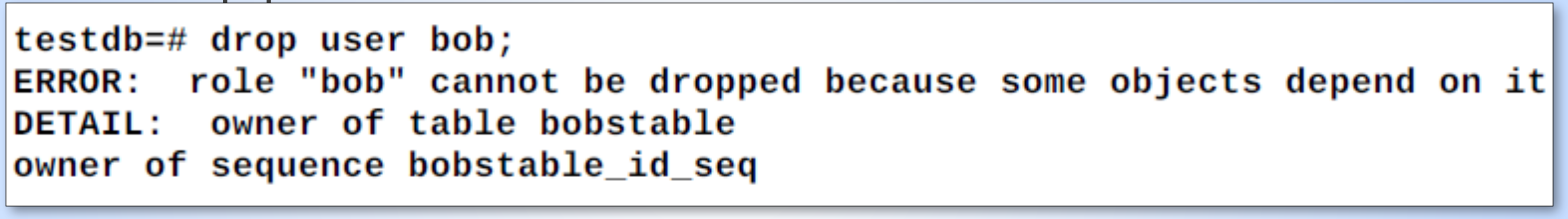
- Sometimes, you need to temporarily revoke a user's connection rights without actually deleting the user or changing the user's password.
- To temporarily prevent the user from logging in, run this command:
  - **pguser=# alter user bob nologin;**
- The same result can be achieved by setting a connection limit for that user to 0:
  - **pguser=# alter user bob connection limit 0;**

Pushing Users Off The System

- You can terminate a user's session with the **pg_terminate_backend()** function. That function takes the **PID**, or the process ID, of the user's session on the server.
- A safer and more useful query that gives a useful response in all cases, which is as follows:
  - **postgres=# SELECT count(pg_terminate_backend(pid)) FROM pg_stat_activity WHERE usename NOT IN (SELECT usename FROM pg_user WHERE usesuper);**

Removing a User Without Dropping Their Data

- When trying to drop a user who owns some tables or other database objects, you get the following error, and the user is not dropped:

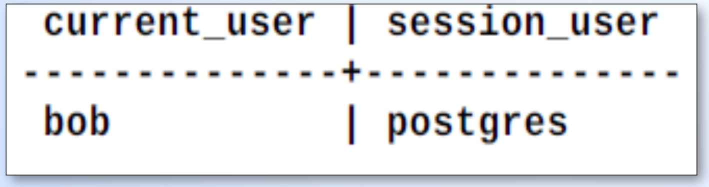


```
testdb=# drop user bob;
ERROR:  role "bob" cannot be dropped because some objects depend on it
DETAIL:  owner of table bobstable
owner of sequence bobstable_id_seq
```

  - Prevent the user from connecting:
    - **pguser=# alter user bob nologin;**
  - Assign the rights of the user to a new user, using the following code:
    - **pguser=# grant bob bobs_replacement;**
  - Assigns ownership of all database objects currently owned by the bob role to the bobs_replacement role and it works only on current database:
    - **REASSIGN OWNED BY bob TO bobs_replacement;**

Always Knowing Which User Is Logged In

- we just logged the value of the user variable in the current PostgreSQL session to log the current user role.
- It is possible to check the logged-in role using the current_user variables:
  - **postgres=> select current_user, session_user;**



```
 current_user | session_user
--------------+--------------
 bob          | postgres
```

- Prepare the required group roles for different tasks and access levels by granting the necessary privileges and options.

Authentication best practices

- Depends on the whole infrastructure setup, the application's nature, the user's characteristics, data sensitivity etc
- Often, database servers are isolated from the world using firewalls
- If the application server and database server are not on the same machine, one can use a strong authentication method, such as LDAP, SSL
- To authenticate an application, recommended to use only one user and reduce the maximum number of allowed connections using a connection pooling software
- If the database server is accessed from the outer world, it is useful to encrypt sessions using SSL certificates