Privilege

PostgreSQL **manages database access permissions** using the concept of *roles*.

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.

Roles can **own database objects (for example, tables and functions)** and can assign privileges on those objects to other roles to control who has access to which objects.

Furthermore, it is possible to **grant *membership* in a role to another role**, thus allowing the member role to use privileges assigned to another role.

The concept of roles subsumes the concepts of "users" and "groups". In PostgreSQL versions before 8.1, users and groups were distinct kinds of entities, but **now there are only roles**. **Any role can act as a user, a group, or both**.

Create Database Roles

> *CREATE ROLE name;*
>
> *DROP ROLE name;*

A role's attributes can be modified after creation with `ALTER ROLE`.

See the reference pages for the CREATE ROLE , ALTER ROLE and DROP ROLE commands for details.

Experiment 1: ROLE and USER

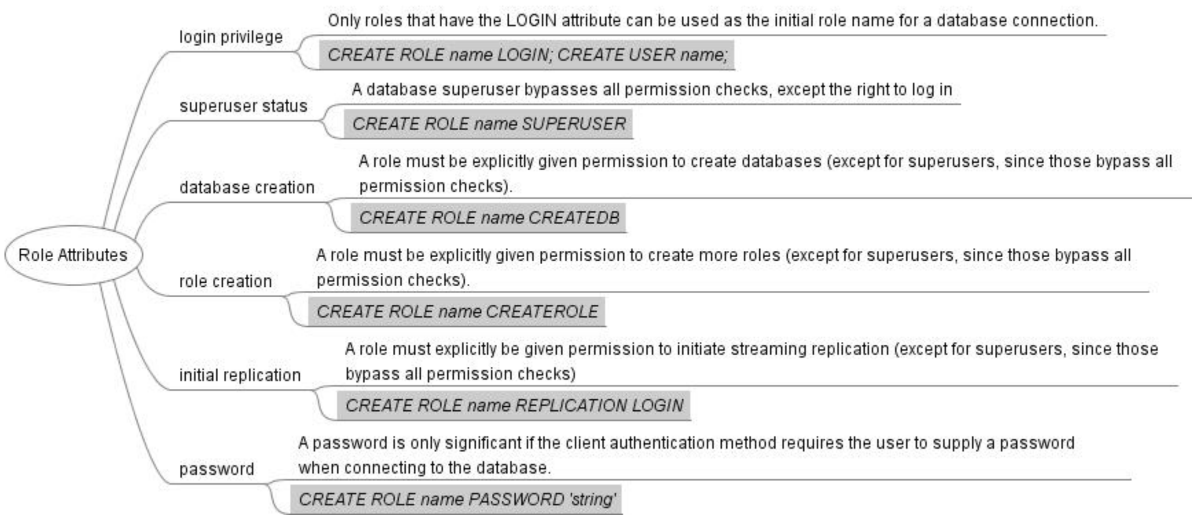
```
CREATE ROLE usrtest;
CREATE USER usr_test2;

select rolname from PG_ROLES;--show roles in pgsql

DROP ROLE usrtest;
DROP USER usr_test2;

CREATE USER "usrtest3";
DROP ROLE "usrtest3";
```

Tips:

- `name` follows the rules for SQL identifiers: either **unadorned without special characters**, or **double-quoted**.
- In order to bootstrap the database system, a freshly initialized system always contains one predefined role. This role is always a "superuser", and by default (unless altered when running `initdb`) it will have the same name as the operating system user that initialized the database cluster. Customarily, this role will be named `postgres`.

Role Attributes

A database role can have a number of attributes that define its privileges and interact with the client authentication system.


Role Attributes
login privilege
Only roles that have the LOGIN attribute can be used as the initial role name for a database connection.
CREATE ROLE name LOGIN; CREATE USER name;
superuser status
A database superuser bypasses all permission checks, except the right to log in
CREATE ROLE name SUPERUSER
database creation
A role must be explicitly given permission to create databases (except for superusers, since those bypass all permission checks).
CREATE ROLE name CREATEDB
role creation
A role must be explicitly given permission to create more roles (except for superusers, since those bypass all permission checks).
CREATE ROLE name CREATEROLE
initial replication
A role must explicitly be given permission to initiate streaming replication (except for superusers, since those bypass all permission checks)
CREATE ROLE name REPLICATION LOGIN
password
A password is only significant if the client authentication method requires the user to supply a password when connecting to the database.
CREATE ROLE name PASSWORD 'string'

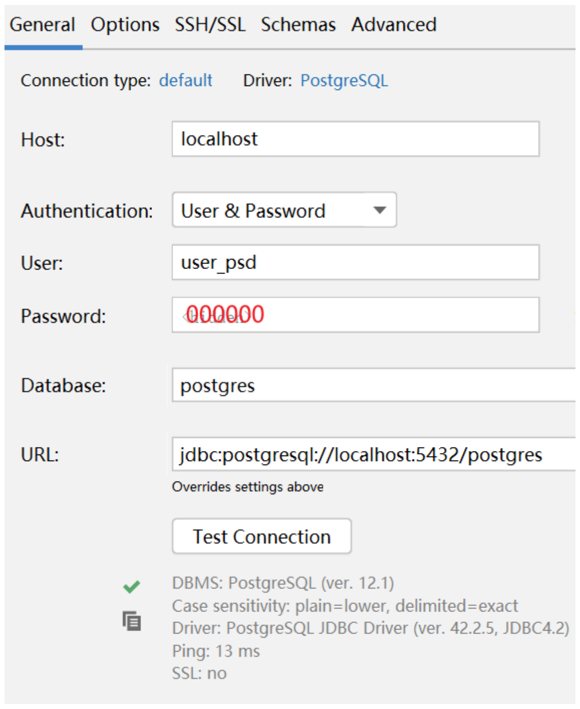
Tips: It is good practice to create a role that has the `CREATEDB` and `CREATEROLE` privileges, but is not a superuser, and then use this role for all routine management of databases and roles. This approach avoids the dangers of operating as a superuser for tasks that do not really require it.

Experiment 2: About login

Create these ROLE/USER with **superuser**

```
create USER user_psd PASSWORD '000000';--success
create ROLE role_psd PASSWORD '000000';--fail
create ROLE role_psd_login LOGIN PASSWORD '000000';--success
```

You can check **USER: user_psd** by logining in Data Source Properties of Datagrip, you'll get successful login in



Connection type: default Driver: PostgreSQL

Host: localhost

Authentication: User & Password

User: user_psd

Password: 000000

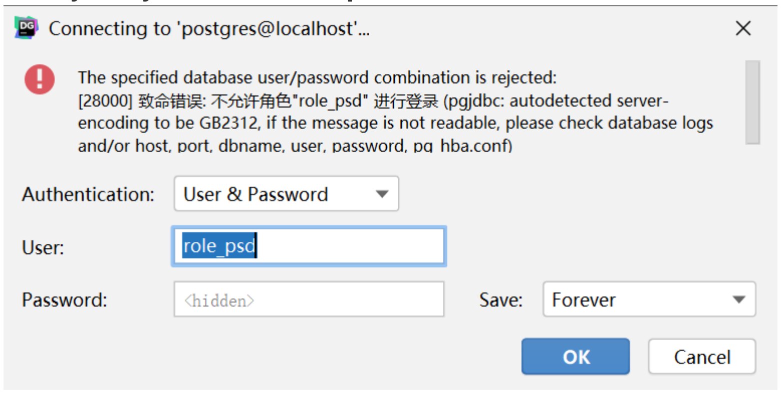
Database: postgres

URL: jdbc:postgresql://localhost:5432/postgres

Overrides settings above

DBMS: PostgreSQL (ver. 12.1)
Case sensitivity: plain=lower, delimited=exact
Driver: PostgreSQL JDBC Driver (ver. 42.2.5, JDBC4.2)
Ping: 13 ms
SSL: no

Then you try the **ROLE:role_psd**, it's a different result


Connecting to 'postgres@localhost'...

The specified database user/password combination is rejected:
[28000] 致命错误: 不允许角色"role_psd" 进行登录 (pgjdbc: autodetected server-encoding to be GB2312, if the message is not readable, please check database logs and/or host, port, dbname, user, password, pg_hba.conf)

Authentication: User & Password

User: role_psd

Password: <hidden>   Save: Forever

But the **ROLE:role_psd_login** can login in successfully. Here the clause `LOGIN` is needed.

Experiment 3: create database/role

Login in with **ROLE:role_psd_login** , and create a new database *db_privilege*

```
create database  db_privilege;
```

[42501] 错误: 创建数据库权限不够

```
create role role_create_role LOGIN ;
```

[42501] 错误: 创建角色的权限不够

Here we create a **ROLE:admin** with **superuser**, and give it some create privileges

```
CREATE ROLE admin WITH CREATEDB CREATEROLE LOGIN PASSWORD '000000';
```

Login in with new **ROLE: admin**, reexecute above sqls

```
create database  db_privilege;
create role role_create_role LOGIN ;
```

Different Privileges

When an object is created, it is assigned an owner. The owner is normally the role that executed the creation statement. For most kinds of objects, the initial state is that only the owner (or a superuser) can do anything with the object. To allow other roles to use it, *privileges* must be granted.

There are different kinds of privileges: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, `REFERENCES`, `TRIGGER`, `CREATE`, `CONNECT`, `TEMPORARY`, `EXECUTE`, `USAGE`, `SET` and `ALTER SYSTEM`. The privileges applicable to a particular object vary depending on the object's type (table, function, etc.).

Experiment 4: Grant

Create a table use **superuser** in public schema

```
create table t_privilege(
    num int,
    prvl varchar(20),
    field varchar(20)
);
insert into t_privilege values (1,'SELECT','table'),
                               (2,'INSERT','table,table column'),
                               (3,'UPDATE','LARGE OBJECT, SEQUENCE, TABLE, table
column'),
                               (4,'DELETE','table'),
                               (5,'TRUNCATE','table'),
                               (6,'REFERENCES','table'),
                               (7,'TRIGGER','table');
```

Switch to **ROLE: admin**, and excute following sqls.

```
select * from t_privilege;
insert into t_privilege values (8,'create','all');
update t_privilege set field='everything' where num=8;
```

[42501] 错误: 对表 emp 权限不够

Switch to **superuser** to give suitable privileges to **ROLE: admin** use clause `GRANT`.

```
GRANT SELECT on public.t_privilege to admin;
GRANT INSERT on public.t_privilege to admin;
GRANT UPDATE on public.t_privilege to admin;
```

Tips: Writing `ALL` in place of a specific privilege grants all privileges that are relevant for the object type.

```
GRANT ALL on public.t_privilege to admin;
```

Then, you can execute SELECT/INSERT/UPDATE on t_privilege table.

Tips: Use `REVOKE` to revoke a previously-granted privilege.

```
REVOKE ALL on schema public from role/public
```

Here the second "public" means all users.

Experiment 5: schema's privilege

Create a new schema with **superuser**, then switch to **ROLE: admin** to login in.

Set the path to new schema, try to create a table.

```
create table t_test(
    id integer,
    text varchar(100)
             );
```

[3F000] 错误: 创建中没有选择模式

Here we can give suitable creation privilege to **ROLE: admin** or change the owner to this role

```
ALTER schema schema_new OWNER TO admin;
```

Experiment 6: Query the ROLES/USERS

```
select usename, usecreatedb, usesuper,useconfig from pg_user;
```

Tips: In this https://www.postgresql.org/docs/16/ddl-priv.html, you can find more details about the privilege.