Lecture 9.2 Roles

What Is a Role?

- PostgreSQL manages permissions using **roles**.
- A **role** can represent:
  - a **single database user** (login identity),
  - a **group of users** (like a role “developers”),
  - or both at the same time.
- Roles can:
  - **own objects** (tables, functions, views...),
  - be granted **privileges** on objects,
  - **grant** privileges to other roles.

Key idea:

> PostgreSQL merged “users” and “groups” into **roles**. A “user” is just a **role with LOGIN**.

Roles vs OS Users, and Scope

- Database roles are **separate** from operating system users (Linux/Windows accounts). There may be similar names, but there is no requirement for a 1:1 match.
- Roles are **global for the entire PostgreSQL cluster**, not per database.
  - If you create role `alice`, it exists for all databases on that server.

Creating Roles

Basic command:

```
CREATE ROLE role_name;
```

To see existing roles:

```
SELECT rolname FROM pg_roles;
```

Role Attributes

LOGIN

- Only roles with `LOGIN` can be used to connect to the database (“real users”).
- A role with LOGIN is essentially what we call a **database user**.

Create a login role:

```
CREATE ROLE alice LOGIN PASSWORD 'secret';
```

or equivalent:

```
CREATE USER alice WITH PASSWORD 'secret';
```

(`CREATE USER` is shorthand for `CREATE ROLE ... LOGIN`)

SUPERUSER

- Bypasses **almost all permission checks** (except the right to log in).
- Should be used **very carefully** – best practice: work as a normal role and switch only when needed.

Create a superuser:

```
CREATE ROLE admin SUPERUSER LOGIN PASSWORD 'super_secret';
```

You must already be a superuser to create another superuser.

CREATEDB

Allows the role to **create databases**.

```
CREATE ROLE dev CREATEDB LOGIN PASSWORD 'dev_pass';
```

Now `dev` can do:

```
CREATE DATABASE dev_db;
```

CREATEROLE

- Allows the role to **create, alter and drop other roles**, and **grant/revoke membership** in them.
- But: only a superuser can create/alter/drop **superuser roles**.

```
CREATE ROLE hr_manager CREATEROLE LOGIN PASSWORD 'hr_pass';
```

REPLICATION

- Allows the role to initiate **streaming replication**.
- Role must also have `LOGIN`.

```
CREATE ROLE replicator REPLICATION LOGIN PASSWORD 'rep_pass';
```

PASSWORD

- Used if the authentication method requires a password (e.g. `md5`, `password`).
- Database passwords are **separate** from OS passwords.

```
CREATE ROLE report_user LOGIN PASSWORD 'r3port!';
```

Modifying Roles – `ALTER ROLE`

You can change role attributes after creation:

```
ALTER ROLE alice WITH CREATEDB;
ALTER ROLE alice WITH NOCREATEDB;
ALTER ROLE alice PASSWORD 'new_secret';
```

Slides: “A role's attributes can be modified after creation with `ALTER ROLE` ... where option can be: [list of attributes like LOGIN, SUPERUSER, CREATEDB, CREATEROLE, REPLICATION, PASSWORD, etc.]”

You can also rename a role:

```
ALTER ROLE old_name RENAME TO new_name;
```

Role Membership (Group roles)

Using roles as **groups** makes permission management much easier.

Idea

- Create a **group role** that represents a function/department:
    - e.g. `sales_read`, `admin`, `analytics_team`.
- Grant privileges to that group role.
- Add individual user roles **as members** of the group role.

Creating a Group Role

Group role (no login):

```
CREATE ROLE sales_read;
```

Give it priveleges, for example:

```
GRANT SELECT ON TABLE orders TO sales_read;
```

Adding / Removing Members

```
GRANT sales_read TO alice, bob;
REVOKE sales_read FROM bob;
```

Notes:

- Members of `sales_read` now **inherit** its privileges (depending on INHERIT setting; default in many setups is "inherit").
- You can also grant **group roles to other group roles** (role hierarchy).
- PostgreSQL prevents **circular membership** loops (role A member of B and B member of A).

Dropping Roles - `DROP ROLE`

To destroy a role:

```
DROP ROLE name;
```

Example:

```
DROP ROLE temp_user;
```

Slide: "To destroy a group role, use `DROP ROLE name;`"

⚠️ Important:

You cannot drop a role that **owns objects** or is still referenced in privileges, unless you fix ownership/privileges first.

Ownership and REASSIGN OWNED

Roles can **own objects** (tables, views, functions, etc.). Sometimes you want to drop a role but **keep** its objects.

Change Ownership One-By-One

```
ALTER TABLE bobs_table OWNER TO alice;
```

Reassign All at Once

```
REASSIGN OWNED BY doomed_role TO successor_role;
```

Optional cleanup of remaining objects owned by the role:

```
DROP OWNED BY doomed_role;
DROP ROLE doomed_role;
```

Pattern from slide:

```
REASSIGN OWNED BY doomed_role TO successor_role;
DROP OWNED BY doomed_role;
DROP ROLE doomed_role;
```

This:

- moves all objects from `doomed_role` to `successor_role`,
- removes remaining owned objects and grants,
- then safely drops the old role.