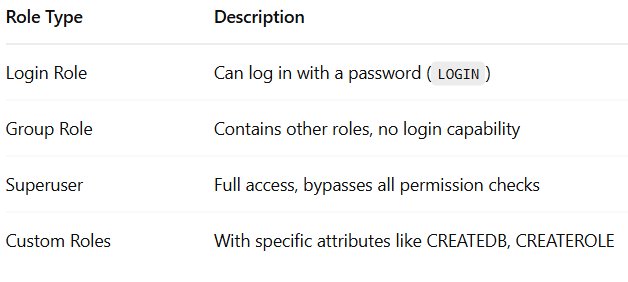
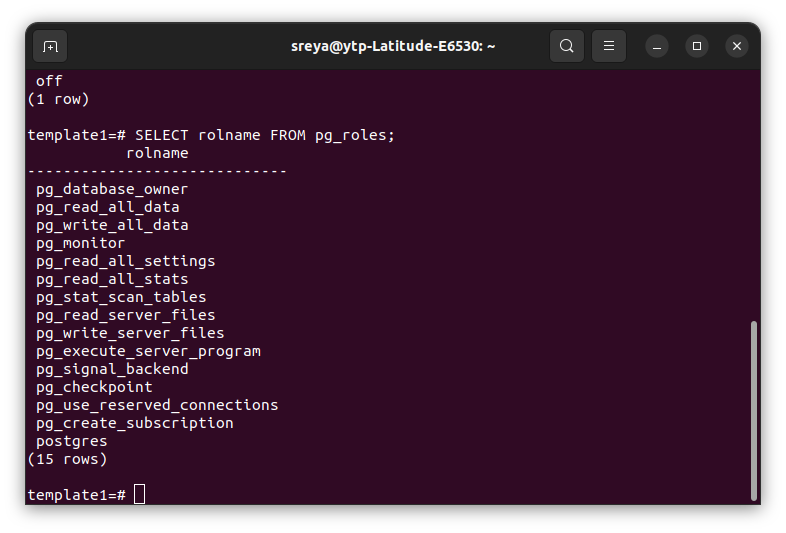
# ROLES

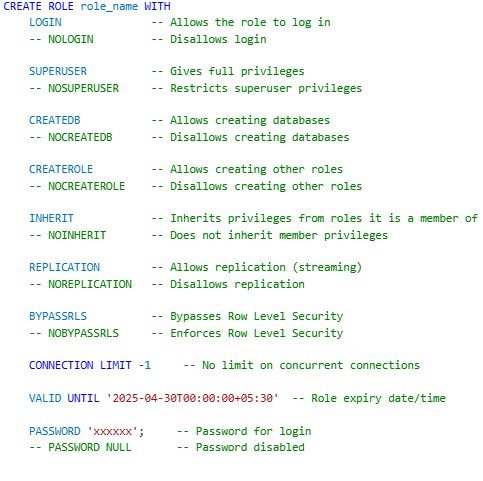
* Manage users and permissions
* Can be a single user or a group
* A **user** in PostgreSQL is just a **role with LOGIN privilege**.
* A role can:
  + Own database objects
  + Be assigned privileges
  + Log in (user)
  + Contain other roles (group).
* **TYPES:**
  + To create a new role in a PostgreSQL server, you use the CREATE ROLE statement.

CREATE ROLE role\_name;

* + To retrieve all roles in the current PostgreSQL server, you can query them from the pg\_roles

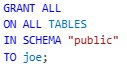
SELECT rolname FROM pg\_roles;



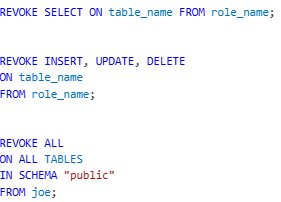
* + In psql, you can use the \du command to show all roles that you create including the postgres role in the current PostgreSQL server
* **ROLE ATTRIBUTES:**
  + define privileges for that role, including login, superuser status, database creation, role creation, password management
  + **GRANT:** 
    - After creating a role with the LOGIN attribute, the role can log in to the PostgreSQL database server.
    - However, it cannot do anything to the database objects like tables, views, functions, etc. For example, the role cannot select data from a table or execute a specific function.
    - To allow a role to interact with database objects, you need to grant privileges on the database objects to the role using the GRANT statement







* + **REVOKE:** The REVOKE statement revokes previously granted privileges on database objects from a role.



* **ROLE MEMBERSHIP:**
  + Group role is a role that serves as a container for other individual roles
  + Group roles allow you to simplify permission management. Instead of granting privileges to individual roles, you can group these roles into a group, grant privileges to a group role, and all the members of that group role will inherit those privileges.

CREATE ROLE group\_role;

GRANT group\_role TO role;

GRANT SELECT ON table\_name TO group\_role;

The role **role** will automatically inherit the SELECT privilege on the table from the **group\_role**.

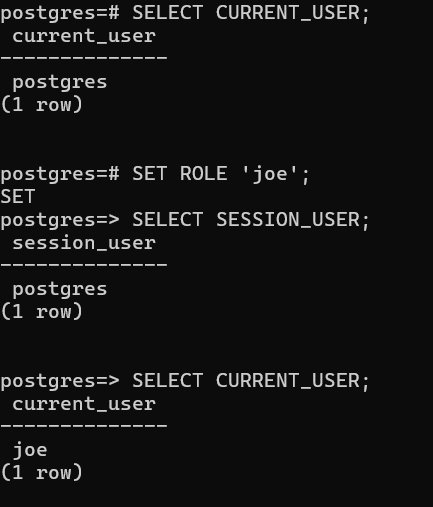
* + **ROLE SWITCHING AND MANAGEMENT:**
    - SET ROLE statement allows you to temporarily change the current role within a database session

SET ROLE role\_name;

* + - RESET ROLE helps to return to the original role.

RESET ROLE;

* + - CURRENT\_USER is a function that returns the name of the currently logged-in database user.
    - SESSION\_USER function used to retrieve the original user who connected to the session



**NOTE:**

Superusers can change any of the attributes for any role.

Roles with CREATEROLE attributes can change any of these attributes for only non-superusers and no-replication roles.

Ordinal roles can only change their passwords.

* + **ALTER ROLE:**
    - Change the attributes of roel
    - ALTER ROLE role\_name option; – Change role attributes (e.g., add login).
    - ALTER ROLE old\_name RENAME TO new\_name;
  + **DROP ROLE:**
    - Allows to delete a role
    - When you remove a role referenced in any database, it raises an error
      * either remove the database objects owned by the role using

DROP OWNED (OR)

reassign the ownership of the database objects to another role

REASSIGN OWNED.

* + - * Secondly,revoke any privileges granted to the role.

REASSIGN OWNED BY target\_role TO another\_role;

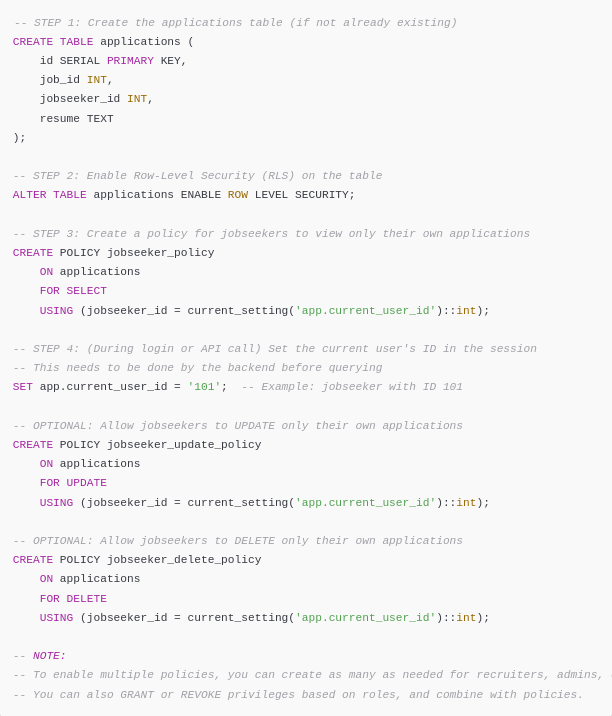
DROP OWNED BY target\_role;

DROP ROLE target\_role;

* + **ROW LEVEL SECURITY:**
    - allows you to restrict rows returned by a query based on the user

executing the query.

* + - The RLS allows you to control access to individual rows in tables based on the current user and specific conditions defined by policies.



* **Why roles?**
  + Let's say you have multiple users in a project (e.g. JobEase)
  + You can define roles like:
    - jobseeker\_role
    - recruiter\_role
    - admin\_role
  + Instead of giving all users full access to the database, you grant them **only what they need**
  + This helps:
    - Prevent accidental data changes
    - Block malicious access

# BACKUP AND RECOVERY

* Need for backup:
  + **Disaster Recovery**: Restore data after system failure, crash, or corruption.
  + **Fault Tolerance**: Recover from partial system failure.
  + **Accidental Errors**: Undo unwanted changes or deletions.
* **Types of Backups:**
  + **Logical Backup**
    - Uses SQL commands to **dump schema and data**.
    - Tool: pg\_dump, pg\_dumpall
    - Output: SQL script or plain/custom/tar format
    - **Advantages**:
      * Portability across versions
      * Can back up individual objects
    - **Disadvantages**:
      * Slower for large databases
      * Doesn’t include WAL or system-level data



To restore a database created by the pg\_dump or pg\_dumpall tools, you can use the pg\_restore tool



* **Physical Backup:**
  + Copies PostgreSQL **data directory** and WAL logs.
  + Tools: pg\_basebackup, file-system-level tools (tar, rsync)
  + **Advantages**:
    - Faster
    - Suitable for full cluster backup
  + **Disadvantages**:
    - Not portable across versions or OS



To restore a physical backup:

**Stop the PostgreSQL server**

**Copy the backup files back** to the PostgreSQL data directory



* **Continuous Archiving + Point-In-Time Recovery (PITR):**
  + PostgreSQL uses a Write-Ahead Log (WAL) to ensure data durability. Every change to the database is first written to WAL files.
  + These logs can be archived and later used to restore the database to any point in time, even if the main data files are lost. This is called Point-in-Time Recovery (PITR).
  + **Steps for PITR :** sudo nano /etc/postgresql/<version>/main/postgresql.conf
    - Enable WAL Archiving:

Modify postgresql.conf:

archive\_mode = on

archive\_command = 'cp %p /path/to/archive/%f'

* Take a Base Backup:

Use pg\_basebackup:

pg\_basebackup -D /path/to/basebackup/ -F tar -z -P

* Archive WAL Files:

WAL files are automatically archived after enabling archive\_mode.

* Restore Base Backup:

Copy base backup to data directory.

* Create recovery.signal:

touch /var/lib/postgresql/16/main/recovery.signal

Define the recovery settings, e.g., target time:

recovery\_target\_time = '2025-04-01 12:00:00'

* Start PostgreSQL Server:

Replay WAL files to the target time.

* Post-Recovery Cleanup:

Remove recovery.signal after recovery.

rm /var/lib/postgresql/16/main/recovery.signal

**SCREENSHOTS:**

