PostgreSQL Users vs. Roles: What’s the Difference?

When working with PostgreSQL, understanding the difference between Users and roles is crucial for managing database security, access control, and permissionseffectively. Let’s dive deep into how PostgreSQL handles these concepts — and whyroles are at the heart of PostgreSQL’s flexible permission system.

- **The Core Concept:-**

In PostgreSQL, users and roles are not completely separate entities. Technically,PostgreSQL handles everything as a role . The difference lies in whether a role has login privileges or not.

User:
A role with login privileges. This is an account that can connect to the databaseusing a username and password.

Role:
A broader concept that can represent users, groups, or collections of privileges.Roles can be granted to other roles, allowing flexible management ofpermissions.

1. How to Create user and role:-

```
CREATE ROLE  john  With  PASSWORD 'secure_password';
```

```
Create user  john   With  PASSWORD 'secure_password';
```

Step 1: Creating an Admin Role and Assigning Users

```
Creating the admin Role
CREATE ROLE admin WITH
  LOGIN
  NOSUPERUSER
  NOINHERIT
  CREATEDB
  CREATEROLE
  NOREPLICATION
  CONNECTION LIMIT 10
  PASSWORD 'securePass123'
  VALID UNTIL '2025-01-01 00:00:00+03';
```

In this example:

- **NOSUPERUSER** prevents the role from having superuser access.
- **CREATEDB** allows the role to create new databases.
- **CREATEROLE** lets the role create other roles.
- **CONNECTION LIMIT** 10 restricts the number of simultaneous connections.

This role is useful for trusted users who need to create other roles and databases but should not have superuser privileges.

Adding Users to the admin Role

Let’s create users and assign them to the admin role:

```
CREATE ROLE kemal WITH LOGIN PASSWORD '**************';
CREATE ROLE ali WITH LOGIN PASSWORD '**************';

GRANT admin TO kemal;
GRANT admin TO ali;
```

Here, we created two users, kemal and ali, and granted them the admin role. Now, these users can inherit permissions from admin, but they still need to SET ROLE admin explicitly to access those privileges due to NOINHERIT being set.

Assigning Permission to user (GRANT) :-

- PostgreSQL, the **GRANT** command is used to grant privileges or permissions to database objects such as tables, views, functions, and schemas

Commnad : grant < privileges> on <objects> To <user or roles> ;

NOTE:- permissions are typically granted at the schema or object level within a database.

1. **Connect To Database:-**

GRANT CONNECT ON DATABASE <database_name > TO <username>;

--the CONNECT privilege allows a user to connect to a specific database within the PostgreSQL cluster.--

2. Schame access:-

```
GRANT ALL ON SCHEMA public TO <username>;
```

This command is used when you want to provide full access and control over all objects within the public schema to a specific user or role.

```
GRANT USAGE ON  SCHEMA public  TO readonly;
```

the USAGE privilege on the public schema allows a user or role to access objects (such as tables, views, and functions) within the public schema of a database.
By default, all users have USAGE privileges on the public schema.

3. OBJECT LEVEL PERMISSION:- 1.select 2.Delete 3. insert 4.Update

Condition 1:- If want to grant read only permission for specific user for specific schema then we use :-

```
Grant select on all tables in schema <schema_name> to user_name;
```

Condition 2:- if want grant all permission:-

```
Grant all privileges on all tables in schema <schema_name> to username;
```

Condition 3.:- if we want to give multiple permission for multiple user:-

```
Grant select,update,insert on all tables in schema <schema_name> to
username1,username2;
```

Conditon4:- if we want to grant read only permission on specific table ,specific database:-

```
Postgres# \c dvd--------------(dvd=database name)
Dvd#  grant select on actor to gaurav;
```

(table_name=actor ; username:- gaurav)

Cluster level:-

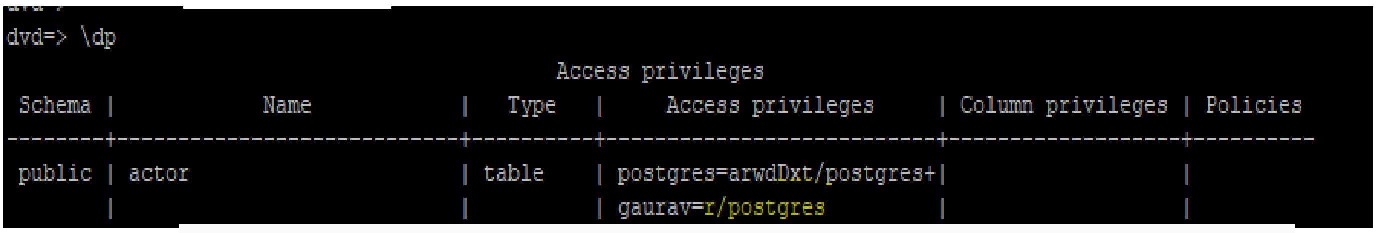
1:- Grant permission to user to cretate database:-

```
Command:- alter user <username> createdb;
```

2:- Make user superuser

```
Alter user <username> with superuser
```

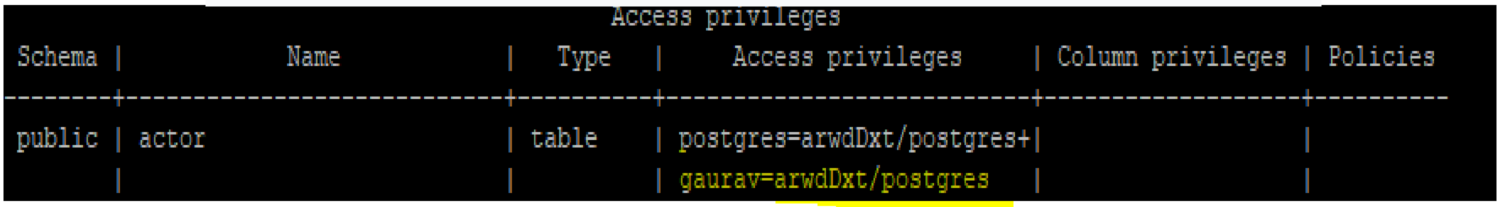
\dp command to obtain information about existing privileges for tables and columns.



```
dvd=> \dp
                                         Access privileges
 Schema |            Name            |   Type   |     Access privileges     | Column privileges | Policies
--------+----------------------------+----------+---------------------------+-------------------+----------
 public | actor                      | table    | postgres=arwdDxt/postgres+|                   |
        |                            |          | gaurav=r/postgres         |                   |
```

r -- SELECT ("read")
w -- UPDATE ("write")
a -- INSERT ("append")
d -- DELETE
D -- TRUNCATE
x -- REFERENCES
t -- TRIGGER
X -- EXECUTE
U -- USAGE
C -- CREATE
c -- CONNECT
T -- TEMPORARY
arwdDxt -- ALL PRIVILEGES (for tables, varies for other objects)

* -- grant option for preceding privilege



```
                                        Access privileges
 Schema |              Name              |  Type    |      Access privileges       | Column privileges | Policies
--------+--------------------------------+----------+------------------------------+-------------------+----------
 public | actor                          | table    | postgres=arwdDxt/postgres+|                   |
        |                                |          | gaurav=arwdDxt/postgres      |                   |
```

--ACCESS TABLES

REVOKE ALL ON ALL TABLES **IN SCHEMA** public **FROM** PUBLIC ;
GRANT SELECT ON ALL TABLES **IN SCHEMA** public **TO** read_only ;
GRANT SELECT, **INSERT**, **UPDATE**, **DELETE ON ALL** TABLES **IN SCHEMA** public **TO** read_write ;
GRANT ALL ON ALL TABLES **IN SCHEMA** public **TO ADMIN** ;

Create Read Only User :-

CREATE USER readonly **WITH ENCRYPTED PASSWORD** 'yourpassword' ;
GRANT CONNECT **ON DATABASE** <database_name > **TO** readonly;
GRANT USAGE **ON SCHEMA** public **TO** readonly;
GRANT SELECT ON ALL SEQUENCES **IN SCHEMA** public **TO** readonly;
GRANT SELECT ON ALL TABLES **IN SCHEMA** public **TO** readonly;

Grant access privileges on objects created in the future :-

ALTER DEFAULT PRIVILEGES IN SCHEMA myschema **GRANT SELECT ON** TABLES **TO** read_only;

ALTER DEFAULT PRIVILEGES IN SCHEMA myschema **GRANT SELECT**,**INSERT**,**DELETE**,**UPDATE ON** TABLES **TO** read_write;

ALTER DEFAULT PRIVILEGES IN SCHEMA myschema **GRANT ALL ON** TABLES **TO ADMIN**;

Or, you can set access privileges on objects created in the future by specified user

ALTER DEFAULT PRIVILEGES FOR ROLE ADMIN GRANT SELECT ON TABLES **TO** read_only;

Alter command

1. ALTER DATABASE:-

Used to change database properties.

Examples:

- **Rename a database**

```
ALTER DATABASE old_db_name RENAME TO new_db_name;
```

- **Change owner**

```
ALTER DATABASE db_name OWNER TO new_owner;
```

2. ALTER SCHEMA

Used to modify schemas.

Examples:

- **Rename schema**

```
ALTER SCHEMA old_schema_name RENAME TO new_schema_name;
```

- **Change owner**

```
ALTER SCHEMA schema_name OWNER TO new_owner;
```

3. ALTER ROLE (User)

Used to modify roles or users.

Examples:

- **Rename role**

```
ALTER ROLE old_role_name RENAME TO new_role_name;
```

- **Change password**

```
ALTER ROLE role_name WITH PASSWORD 'new_password';
```

- **Set role privileges**
-

```
ALTER ROLE role_name WITH LOGIN;
ALTER ROLE role_name NOLOGIN;
ALTER ROLE role_name SUPERUSER;
```

6. ALTER SEQUENCE

Used to modify sequence objects.

Examples:

Restart sequence

```
ALTER SEQUENCE sequence_name RESTART WITH 1;
```

Change increment

```
ALTER SEQUENCE sequence_name INCREMENT BY 5;
```

Change min/max values

```
ALTER SEQUENCE sequence_name MINVALUE 10 MAXVALUE 1000;
```

5. ALTER VIEW

Used to rename or modify views.

- **Rename a view**

```
ALTER VIEW old_view_name RENAME TO new_view_name;
```

- **Change owner**

```
ALTER VIEW view_name OWNER TO new_owner;
```

6. ALTER INDEX

Used to rename or change ownership of an index.

Rename index

```
ALTER INDEX old_index_name RENAME TO new_index_name;
```

Change owner

```
ALTER INDEX index_name OWNER TO new_owner;
```

Revoke

1. Revoke Database Privileges:-

- If a user has privileges on a database (like CONNECT or CREATE):

-- Revoke CONNECT privilege on a database

```
REVOKE CONNECT ON DATABASE database_name FROM role_name;
```

-- Revoke CREATE privilege on a database

```
REVOKE CREATE ON DATABASE database_name FROM role_name;
```

-- Revoke ALL privileges on a database

```
REVOKE ALL PRIVILEGES ON DATABASE database_name FROM role_name;
```

2. Revoke Schema Privileges:-

For schemas (USAGE, CREATE):

```
-- Revoke USAGE on a schema

REVOKE USAGE ON SCHEMA schema_name FROM role_name;

-- Revoke CREATE on a schema

REVOKE CREATE ON SCHEMA schema_name FROM role_name;

-- Revoke ALL privileges on a schema

REVOKE ALL PRIVILEGES ON SCHEMA schema_name FROM role_name;
```

3. Revoke Table Privileges

For tables (SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, ALL):

```
-- Revoke specific privileges on a table

REVOKE SELECT, INSERT, UPDATE, DELETE ON table_name FROM role_name;

-- Revoke ALL privileges on a table

REVOKE ALL PRIVILEGES ON table_name FROM role_name;
```

4. Revoke Sequence Privileges

For sequences (USAGE, SELECT, UPDATE, ALL):

```
-- Revoke specific privileges on a sequence

REVOKE USAGE, SELECT, UPDATE ON SEQUENCE sequence_name FROM role_name;

-- Revoke ALL privileges on a sequence

REVOKE ALL PRIVILEGES ON SEQUENCE sequence_name FROM role_name;
```

5. Revoke Function/Procedure Privileges

For functions or procedures:

```
-- Revoke EXECUTE on a function

REVOKE EXECUTE ON FUNCTION function_name(arg_types) FROM role_name;

-- Revoke ALL privileges on a function

REVOKE ALL PRIVILEGES ON FUNCTION function_name(arg_types) FROM role_name;
```

6. Revoke Role Memberships

If a role was granted membership to another role:

```
-- Revoke role membership

REVOKE role_name FROM member_role_name;
```

7. Revoke default privileges for future tables and sequences:-

```
ALTER DEFAULT PRIVILEGES IN SCHEMA plf REVOKE SELECT, INSERT, UPDATE, DELETE ON TABLES
FROM pennantvendor;
ALTER DEFAULT PRIVILEGES IN SCHEMA plf REVOKE USAGE, SELECT ON SEQUENCES FROM
pennantvendor;

ALTER DEFAULT PRIVILEGES IN SCHEMA plfaudit REVOKE SELECT, INSERT, UPDATE, DELETE ON TABLES
FROM pennantvendor;
ALTER DEFAULT PRIVILEGES IN SCHEMA plfaudit REVOKE USAGE, SELECT ON SEQUENCES FROM
pennantvendor;

ALTER DEFAULT PRIVILEGES IN SCHEMA bkp REVOKE SELECT, INSERT, UPDATE, DELETE ON TABLES
FROM pennantvendor;
ALTER DEFAULT PRIVILEGES IN SCHEMA bkp REVOKE USAGE, SELECT ON SEQUENCES FROM
pennantvendor;
```

✅Tip: If you had previously granted privileges like:

```
GRANT USAGE ON SCHEMA plf TO pennantvendor;
GRANT CONNECT ON DATABASE PRODPLFM TO gan;
GRANT SELECT, INSERT ON ALL TABLES IN SCHEMA plf TO pennantvendor;


The corresponding REVOKE commands would be:

REVOKE USAGE ON SCHEMA plf FROM pennantvendor;
REVOKE CONNECT ON DATABASE PRODPLFM FROM gan;
REVOKE SELECT, INSERT ON ALL TABLES IN SCHEMA plf FROM pennantvendor;
```

Live Production exmaple:-

First Example:-

- Step by to give permission

-- Switch to the target database

```
\c your_database_name
```

-- Grant usage on schemas

```
GRANT USAGE ON SCHEMA plf TO pennantvendor;
```

GRANT USAGE ON SCHEMA plfaudit TO pennantvendor;

GRANT USAGE ON SCHEMA bkp TO pennantvendor;

-- Grant read (SELECT) and write (INSERT, UPDATE, DELETE) permissions on all tables in the schemas

GRANT CONNECT ON DATABASE "PRODPLFM" TO "ganeshkale.ext.pcpl";

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO appadmin;

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA plfaudit TO pennantvendor;

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA bkp TO pennantvendor;

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA "DL_EOM","LeadSquared","PLF_DR","bkp","cron","magic","otherdata","plf","plfaudit","plfext","plfsnap","public" TO "finappuser";

==
==

-- Grant permissions for sequences in all schemas (if sequences are used)

GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA plf TO pennantvendor;

GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA plfaudit TO pennantvendor;

GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA bkp TO pennantvendor;

==
==

-- Ensure future tables and sequences in the schemas inherit the same permissions

ALTER DEFAULT PRIVILEGES IN SCHEMA plf GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO pennantvendor;

ALTER DEFAULT PRIVILEGES IN SCHEMA plf GRANT USAGE, SELECT ON SEQUENCES TO pennantvendor;

ALTER DEFAULT PRIVILEGES IN SCHEMA plfaudit GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO pennantvendor;

ALTER DEFAULT PRIVILEGES IN SCHEMA plfaudit GRANT USAGE, SELECT ON SEQUENCES TO pennantvendor;

ALTER DEFAULT PRIVILEGES IN SCHEMA bkp GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO pennantvendor;

ALTER DEFAULT PRIVILEGES IN SCHEMA bkp GRANT USAGE, SELECT ON SEQUENCES TO pennantvendor;

===
===

View Permission:-

GRANT SELECT, INSERT, UPDATE, DELETE ON facilitydetails_view TO finappuser;

GRANT SELECT, INSERT, UPDATE, DELETE ON facilitydetailsenquiry_view TO finappuser;

- To see View:-

\dv+ facilitydetails_view

- To see permission correct given nor not :-

SELECT has_table_privilege('magicxpauser', 'vw_pcpl_banking_dtl_scf', 'SELECT','INSERT', 'UPDATE', 'DELETE');

```
SELECT has_table_privilege('postgres', 'public.my_table', 'SELECT');
SELECT has_table_privilege('magicxpauser', 'vw_pcpl_banking_dtl_scf', 'SELECT');

SELECT
   privilege_type,
   has_table_privilege('magicxpauser', 'vw_pcpl_banking_dtl', privilege_type) AS has_privilege
FROM (
   VALUES ('SELECT'), ('INSERT'), ('UPDATE'), ('DELETE')
) AS privileges(privilege_type);
```

5.Function:-

- **To change owner of that funaction:-**

```
SELECT 'ALTER FUNCTION ' || n.nspname || '.' || p.proname || ' OWNER TO Tarik;'
FROM pg_proc p
JOIN pg_namespace n ON p.pronamespace = n.oid
WHERE n.nspname IN ('_bsmart_times')
AND pg_catalog.pg_get_userbyid(p.proowner) = 'postgres'
ORDER BY n.nspname, p.proname;
```

```
SELECT 'ALTER FUNCTION ' || n.nspname || '.' || p.proname || ' OWNER TO "emsuat";'
FROM pg_proc p
JOIN pg_namespace n ON p.pronamespace = n.oid
WHERE n.nspname IN ('public')
ORDER BY n.nspname, p.proname;
```

- **To see all function :-**

```
SELECT n.nspname AS schema_name,
       p.proname AS function_name,
       r.rolname AS owner
FROM pg_proc p
JOIN pg_namespace n ON p.pronamespace = n.oid
JOIN pg_roles r ON p.proowner = r.oid
WHERE n.nspname NOT IN ('pg_catalog', 'information_schema')
ORDER BY schema_name, function_name;
```

Stored Procedures

- **Check Owners of All Stored Procedures**

```
SELECT proname, nspname AS schema_name, pg_roles.rolname AS owner
FROM pg_proc
JOIN pg_namespace ON pg_proc.pronamespace = pg_namespace.oid
JOIN pg_roles ON pg_proc.proowner = pg_roles.oid
WHERE prokind = 'p';
```

```
SELECT n.nspname AS schema_name, p.proname AS procedure_name, r.rolname AS owner
FROM pg_proc p
JOIN pg_namespace n ON p.pronamespace = n.oid
JOIN pg_roles r ON p.proowner = r.oid
WHERE p.prokind = 'p'   -- Replace with procedure name
AND r.rolname = 'postgres';  -- Check for a specific owner
```

- To see argument :-

```
SELECT proname, pg_get_function_arguments(oid) AS arguments
FROM pg_proc
WHERE proname = 'sp_tms_usp_ps_delete_tour_bill_expences';
```

- **To change owner :-**

```
ALTER PROCEDURE public.sp_tms_usp_ps_cancel_journey_details_for_full_reschedule(IN rjdrhkey
bigint) OWNER TO 'Medical_Retirement';

ALTER PROCEDURE public.delete_tour_request_journey_details(srgument palce here) OWNER TO
retmeddb;
```

- **To check arg:-**

```
SELECT p.proname, r.rolname AS owner
FROM pg_proc p
JOIN pg_roles r ON p.proowner = r.oid
WHERE p.proname = 'sp_tms_usp_ps_cancel_journey_details_for_full_reschedule';
```

Second Example

1.

```
 create role hes;
```

2.

```
 create role read_role;

grant usage on schema hes,bsmartframework to read_role;
grant SELECT on ALL tables in schema hes,bsmartframework to read_role ;
grant SELECT on ALL sequences in schema hes,bsmartframework to read_role ;
```

3.

```
create role write_role;
 grant usage on schema hes,bsmartframework to write_role;
grant INSERT,update,delete,select on ALL tables in schema hes,bsmartframework to write_role ;
grant ALL on ALL sequences in schema hes,bsmartframework to write_role ;
```

4.

```
create role create_role;
```

```
grant create on SCHEMA hes,bsmartframework to create_role;
```

5.

```
alter schema bsmartframework owner to hes;
alter schema _bsmart_times owner to hes;
```

6.

```
-- change ownership of all tables/sequences
```

Table:-

```
DO $$
DECLARE
   tbl_schema TEXT;
   tbl_name TEXT;
BEGIN
   FOR tbl_schema, tbl_name IN
      SELECT schemaname, tablename
      FROM pg_tables
      WHERE schemaname IN ('hes','bsmartframework')
      AND tableowner NOT IN ('hes')
   LOOP
      EXECUTE format(
         'ALTER TABLE %I.%I OWNER TO hes',
         tbl_schema, tbl_name
      );
   END LOOP;
END $$;
```

Schema:-

```
DO $$
DECLARE
   seq_schema TEXT;
   seq_name TEXT;
BEGIN
   FOR seq_schema, seq_name IN
      SELECT sequence_schema, sequence_name
      FROM information_schema.sequences
      WHERE sequence_schema IN ('hes','bsmartframework')
   LOOP
      EXECUTE format(
         'ALTER SEQUENCE %I.%I OWNER TO hes',
         seq_schema, seq_name
      );
   END LOOP;
END $$;
```

Third Example

--- Set Default Privileges for a Role:

1. You can set default privileges for a role so
that any objects created by this role will have the specified permissions.

command:-

```
ALTER DEFAULT PRIVILEGES IN SCHEMA your_schema
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO your_role;
```

2. This command ensures that any new tables created in the specified schema will automatically
have the SELECT, INSERT, UPDATE, and DELETE permissions granted to your_role.

---------- Set Default Privileges for a Specific User:-

1. You can also specify default privileges for objects created by a specific user.

command:-

```
ALTER DEFAULT PRIVILEGES FOR USER your_user IN SCHEMA your_schema
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO your_role;
```

This command ensures that any new tables created in the specified schema by your_user
will automatically have the specified permissions granted to your_role.

-------------- Set Default Privileges for All Users:

1. If you want to set default privileges for all users who create tables in a schema, you can do so without specifying a particular user.

command:-

```
ALTER DEFAULT PRIVILEGES IN SCHEMA your_schema
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO PUBLIC;
```

This command ensures that any new tables created in the specified schema will automatically have the specified permissions granted to all users.

Revoke:-

```
ALTER DEFAULT PRIVILEGES IN SCHEMA bkp
REVOKE SELECT, INSERT, UPDATE ,DELETE ON TABLES FROM pennantvendor;
```

```
psql -h PROD-DB-PENNANT-ENCRYPTED-UPGRADED.CN0WXNKVPVCB.AP-SOUTH-1.RDS.AMAZONAWS.COM  -U proadmin -d postgres
```

```
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA "DL_EOM","LeadSquared","PLF_DR","bkp","cron","magic","otherdata","plf","plfaudit","plfext","plfsnap","public" TO "pennantvendor";
```

before-eod-prod-db-pennant-encrypted-upgraded-03-07-2024.cn0wxnkvpvcb.ap-south-1.rds.amazonaws.com

```
GRANT SELECT ON view_name TO finappuser;
```

C:\Program Files\PostgreSQL\15\bin

===

```
CREATE USER "Aparna.bhilare.pcpl" WITH PASSWORD 'Aparna@123' VALID UNTIL '2024-08-15';

ALTER USER myuser VALID UNTIL '2024-10-17';

ALTER USER pennantuser VALID UNTIL '2024-10-17' PASSWORD 'Support@123';
ALTER USER "nehajadhav.ext" VALID UNTIL '2024-07-17' PASSWORD 'Neha@1234';
```

==
```
ALTER DEFAULT PRIVILEGES IN SCHEMA your_schema
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO your_role;
```
===

check permission:-

- **schema level:-**

```
SELECT nspname AS schema_name,
    pg_roles.rolname AS user_name,
    has_schema_privilege(pg_roles.rolname, nspname, 'USAGE') AS has_usage,
    has_schema_privilege(pg_roles.rolname, nspname, 'CREATE') AS has_create
FROM pg_namespace, pg_roles
WHERE rolname = 'username';
```

- **Check Table-Level Permissions:-**

```
SELECT grantee AS user_name,
       table_schema,
       table_name,
       MAX(CASE WHEN privilege_type = 'SELECT' THEN 'YES' ELSE 'NO' END) AS can_select,
       MAX(CASE WHEN privilege_type = 'INSERT' THEN 'YES' ELSE 'NO' END) AS can_insert,
       MAX(CASE WHEN privilege_type = 'UPDATE' THEN 'YES' ELSE 'NO' END) AS can_update,
       MAX(CASE WHEN privilege_type = 'DELETE' THEN 'YES' ELSE 'NO' END) AS can_delete
FROM information_schema.role_table_grants
WHERE table_schema = 'schema_name'
  AND grantee = 'username'
GROUP BY grantee, table_schema, table_name;
```

```
SELECT grantee AS user_name,
       table_schema,
       table_name,
       ARRAY_AGG(privilege_type) AS privileges
FROM information_schema.role_table_grants
WHERE table_schema = 'DL_EOM'
  AND grantee = 'pennantvendor'
GROUP BY grantee, table_schema, table_name;
```

```
SELECT r.rolname AS user_name,
       c.relname AS table_name,
       has_table_privilege(r.rolname, c.oid, 'SELECT') AS can_select,
       has_table_privilege(r.rolname, c.oid, 'INSERT') AS can_insert,
       has_table_privilege(r.rolname, c.oid, 'UPDATE') AS can_update,
       has_table_privilege(r.rolname, c.oid, 'DELETE') AS can_delete
FROM pg_catalog.pg_roles r
JOIN pg_catalog.pg_class c ON c.relowner = r.oid
JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE n.nspname = 'public'
  AND r.rolname = 'john_doe'
  AND c.relkind = 'r';
```