

**PostgreSQL Roles & Users Creation**

**Purpose:**

Implement role-based access control (RBAC) in PostgreSQL using industry-standard best practices, emphasizing least privilege, verification logic, and dry-run capability for safe execution.

**🔐 Standard 1: Create Group Roles (No Login)**

DO  
$$  
DECLARE  
 role\_name TEXT := 'read\_only';  
 execute\_flag BOOLEAN := FALSE;  
BEGIN  
 RAISE NOTICE 'Execution mode is %', CASE WHEN execute\_flag THEN 'ON (changes applied)' ELSE 'OFF (dry-run only)' END;  
  
 IF NOT EXISTS (  
 SELECT 1 FROM pg\_roles WHERE rolname = role\_name  
 ) THEN  
 RAISE NOTICE '[DRY-RUN] Would create group role: %', role\_name;  
 IF execute\_flag THEN  
 EXECUTE format('CREATE ROLE %I NOLOGIN', role\_name);  
 END IF;  
 ELSE  
 RAISE NOTICE 'Group role % already exists', role\_name;  
 END IF;  
END  
$$ LANGUAGE plpgsql;

**👤 Standard 2: Create Application Users and Assign Roles**

DO  
$$  
DECLARE  
 user\_list TEXT[] := ARRAY['alice', 'bob', 'carol'];  
 user\_name TEXT;  
 base\_role TEXT := 'read\_only';  
 execute\_flag BOOLEAN := FALSE;  
BEGIN  
 RAISE NOTICE 'Execution mode is %', CASE WHEN execute\_flag THEN 'ON' ELSE 'OFF (dry-run only)' END;  
  
 FOREACH user\_name IN ARRAY user\_list  
 LOOP  
 IF NOT EXISTS (  
 SELECT 1 FROM pg\_roles WHERE rolname = user\_name  
 ) THEN  
 RAISE NOTICE '[DRY-RUN] Would create user: %', user\_name;  
 IF execute\_flag THEN  
 EXECUTE format('CREATE ROLE %I WITH LOGIN PASSWORD %L', user\_name, user\_name || '\_TempPass!');  
 END IF;  
 ELSE  
 RAISE NOTICE 'User % already exists', user\_name;  
 END IF;  
  
 IF NOT EXISTS (  
 SELECT 1  
 FROM pg\_auth\_members m  
 JOIN pg\_roles r1 ON m.roleid = r1.oid  
 JOIN pg\_roles r2 ON m.member = r2.oid  
 WHERE r1.rolname = base\_role AND r2.rolname = user\_name  
 ) THEN  
 RAISE NOTICE '[DRY-RUN] Would grant % to %', base\_role, user\_name;  
 IF execute\_flag THEN  
 EXECUTE format('GRANT %I TO %I', base\_role, user\_name);  
 END IF;  
 ELSE  
 RAISE NOTICE 'User % is already a member of %', user\_name, base\_role;  
 END IF;  
 END LOOP;  
END  
$$ LANGUAGE plpgsql;

**🧱 Standard 3: Use Schema-Level and Table-Level Privileges**

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;  
  
DO

$$

DECLARE

execute\_flag BOOLEAN := false; -- TRUE to apply changes

target\_role TEXT := 'read\_only';

schema\_name TEXT := 'public';

already\_granted BOOLEAN := false;

BEGIN

RAISE NOTICE 'Execution mode is %', CASE WHEN execute\_flag THEN 'ON (changes will be applied)' ELSE 'OFF (dry-run only)' END;

-- Check if SELECT has already been granted on future tables in public schema to read\_only

SELECT EXISTS (

SELECT 1

FROM pg\_default\_acl da

JOIN pg\_roles gr ON da.defaclrole = gr.oid

WHERE gr.rolname = CURRENT\_USER

AND defaclnamespace = (SELECT oid FROM pg\_namespace WHERE nspname = schema\_name)

AND array\_to\_string(defaclacl, ',') LIKE format('%%%s=ar%%', target\_role)

) INTO already\_granted;

IF already\_granted THEN

RAISE NOTICE 'Default privileges already grant SELECT on future tables in schema "%" to role "%"', schema\_name, target\_role;

ELSE

RAISE NOTICE '[DRY-RUN] Would grant default SELECT privileges on future tables in schema "%" to role "%"', schema\_name, target\_role;

IF execute\_flag THEN

EXECUTE format(

'ALTER DEFAULT PRIVILEGES IN SCHEMA %I GRANT SELECT ON TABLES TO %I;',

schema\_name, target\_role

);

END IF;

END IF;

END

$$ LANGUAGE plpgsql;  
  
\z public.\*

**🚫 Standard 4: Remove Dangerous Defaults**

DO

$$

DECLARE

execute\_flag BOOLEAN := false; -- TRUE to apply changes

target\_schema TEXT := 'public';

has\_privs BOOLEAN := false;

BEGIN

RAISE NOTICE 'Execution mode is %', CASE WHEN execute\_flag THEN 'ON (changes will be applied)' ELSE 'OFF (dry-run only)' END;

-- Check if PUBLIC has any privileges on the schema

SELECT EXISTS (

SELECT 1

FROM information\_schema.role\_schema\_grants

WHERE grantee = 'PUBLIC'

AND schema\_name = target\_schema

) INTO has\_privs;

IF has\_privs THEN

RAISE NOTICE '[DRY-RUN] Would revoke ALL privileges on schema "%" from PUBLIC', target\_schema;

IF execute\_flag THEN

EXECUTE format('REVOKE ALL ON SCHEMA %I FROM PUBLIC;', target\_schema);

END IF;

ELSE

RAISE NOTICE 'PUBLIC role does not currently have any privileges on schema "%"', target\_schema;

END IF;

END

$$ LANGUAGE plpgsql;  
  
SELECT grantee, privilege\_type  
FROM information\_schema.schema\_privileges  
WHERE schema\_name = 'public';

**🚨 Standard 5: Avoid SUPERUSER Unless Absolutely Necessary**

SELECT rolname, rolsuper FROM pg\_roles WHERE rolsuper = TRUE;  
  
SELECT rolname, rolsuper FROM pg\_roles WHERE rolname = 'app\_user';

**📚 Supporting Web Resources**

• PostgreSQL Role Management – Official Docs

<https://www.postgresql.org/docs/current/user-manag.html>

• Privileges and GRANT — PostgreSQL Official

<https://www.postgresql.org/docs/current/sql-grant.html>

• Best Practices for PostgreSQL Permissions – Percona

<https://www.percona.com/blog/postgresql-roles-and-permissions-best-practices/>

• PostgreSQL: Managing Users and Permissions – EDB

<https://www.enterprisedb.com/postgres-tutorials/postgresql-roles-and-permissions>

**🧠 TL;DR Recap**

|  |  |  |
| --- | --- | --- |
| Standard |  | Description |
| Use Group Roles |  | Group permissions together for manageability |
| Dry-Run Before Execution |  | Prevent "oops" moments in production |
| Avoid Superuser for Apps |  | It’s not a cape—it’s a liability |
| Verify Everything |  | Every grant, every user, every time |
| Revoke PUBLIC |  | The default is not safe by default |