Securing Your PostgreSQL Data: One Size Does Not Fit All

Hettie Dombrovskaya
Database Architect

PGDay Chicago 2025

Who Am I

Hettie Dombrovskaya
- Database Architect at DRW
- Prairie Postgres NFP President
- Midwest PostgreSQL User Group
- Illinois Prairie PostgreSQL User Group
- ACM Chicago Chapter Communications Chair

What will be covered

- Identifying common security challenges
- What can we do about them?
- Security framework overview
- Models description
- How we built it
- Ongoing issues and future prospects

Why this talk?

WHY?!

Challenge #1: PostgreSQL does not force you to create roles and schemas in order to start.

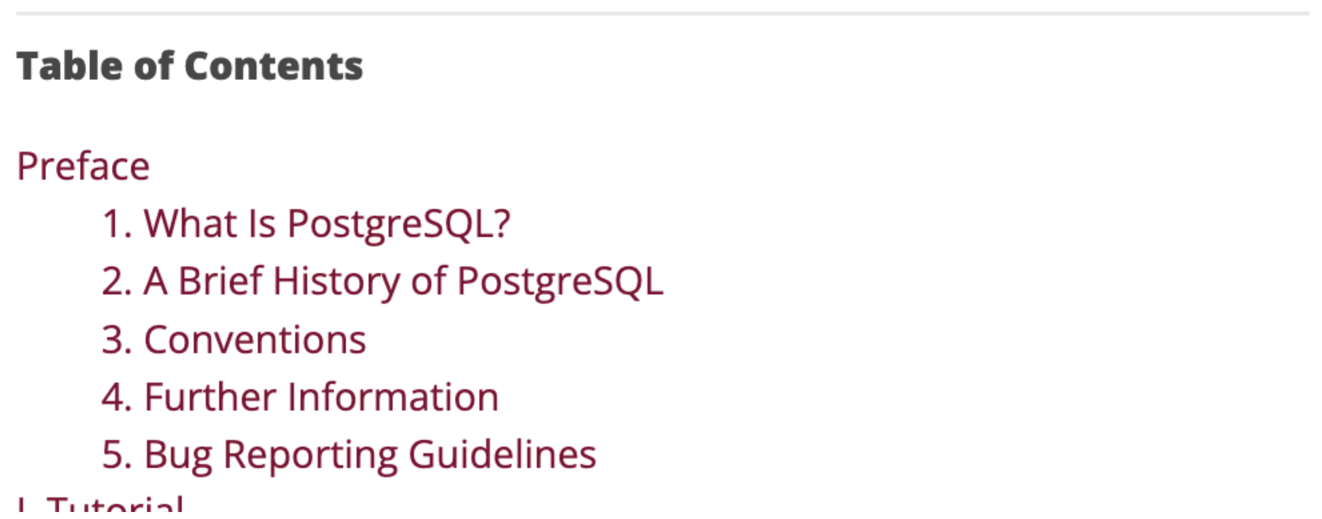
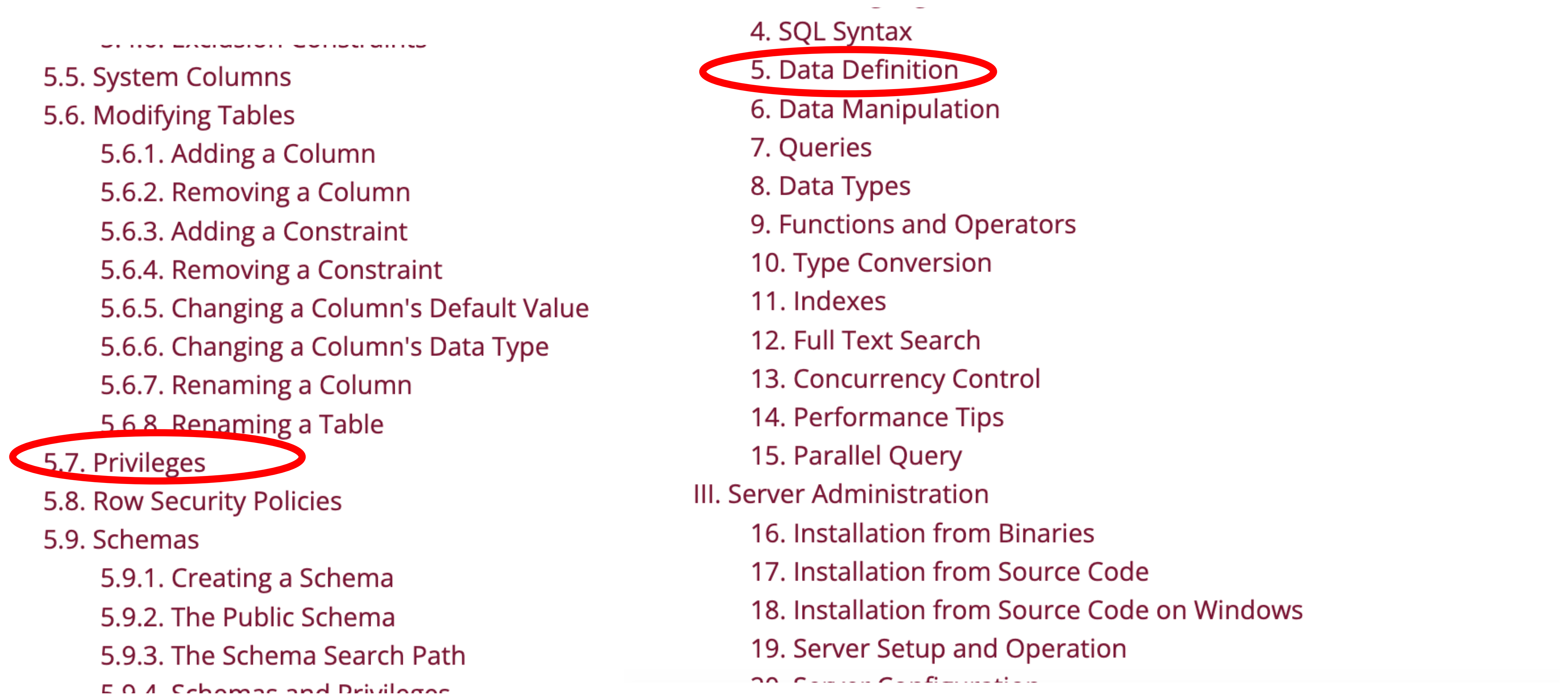


Table of Contents

And all examples in documentation create objects in PUBLIC schema!



As a result…

- Applications are developed using `postgres` user
- When they move to production, developer either forget to change the user or run into permissions problems they ~~do not have time~~ do not know how to fix
- When an application uses connection pools different application users connect as the same database user

Challenge #2: The wonders of inheritance

- **Starting with PG 7.3, there is no distinction between users and roles**

(user=role+login)

```
create role role1;
create role role2 login password ‘pwd’;
create user user1 password ‘pwd’;
```

- All grants below will work:

```
grant role1 to role2;
grant role2 TO user1;
grant user1 to role2;
```

… and if later you will execute

```
create role role3;
grant role3 to role1 ---will be inherited
```

Challenge #3: You think you created a role for a database? Think again!

- Roles are created on the instance level, not the database level
- If there are several databases on one instance, all users will have access to **all** databases, because…

  By default, all user have `CONNECT` privilege to all databases on the instance
- Until PG 15, all users could create objects in `PUBLIC` schema. That includes public schema in all databases on the same instance.
- If a customer requested a superuser privilege, this superuser will be able to do **everything** on **all databases** on that instance.

Trying to do things the right way!

Grouping (objects and users):

- Using schemas for access control: all objects in each schema have the same set of privileges
- Granting privileges to groups (nologin roles) only. And granting roles to users

```
create schema orders owner orders_owner;
grant orders_owner to orders_admin;
create role orders_read_write;
create role orders_read_only;
grant select on all tables in schema orders to orders_read_only;
grant select, insert, update, delete on all tables in schema
orders to orders_read_write;
```

What is not going to work?

Challenge #4: Default privileges

- Yes, you also need to `grant usage`!

```
grant usage on schema orders to orders read_write, orders_read_only
```

- What else?

```
alter default privileges in schema orders grant select on tables to
orders_read_only;
alter default privileges in schema orders grant select, insert, update,
delete on tables to orders_read_write;
```

Now:

```
create table orders.customer (
customer_id int primary key,
customer_name text);
```

- Why were default permissions not applied?!

```
alter default privileges in schema orders for role orders_owner grant
select, insert, update, delete on tables to orders_read_write;
```

Challenge #5: The wonders of ownership!

NO!

Challenges #6, #7, #8… Lots of weird things!

```
grant select orders.sales_points to role_ro;
grant insert, update, delete on orders.sales_points to role_app;
grant role_ro to your_user;
grant role_app to your_user;
revoke delete on orders.sales_points from your_user;
```

Will this work?

- It won’t, and moreover, errors won’t be reported:
    REVOKE of permissions which are not granted
    GRANT permissions which are already granted except for roles
- You can’t drop user that has any privileges
- You can’t `drop role cascade`
- **And there is no easy way to see what permissions a given user has!**

Now imagine you have not five, not ten, but 300+ databases, and new requests are coming each day!

A separate instance for each new project – possible, but expensive.

Security Models Overview

Principles and implementation

DRW

Basic principles

The only security model to support multi-tenancy within one PostgreSQL database

Principle of least privilege

- A user is given the minimum levels of access needed to perform their job functions.

Durability

- Non-superuser users do not have a way to bypass security settings

Flexibility

- One package supports four security models with different permissions hierarchy.

Key features

Security levels matrix

- Schema owner TRUE/FALSE
- Account owner TRUE/FALSE

Database level security

- Security modal is set up on the database level

Security-definer functions

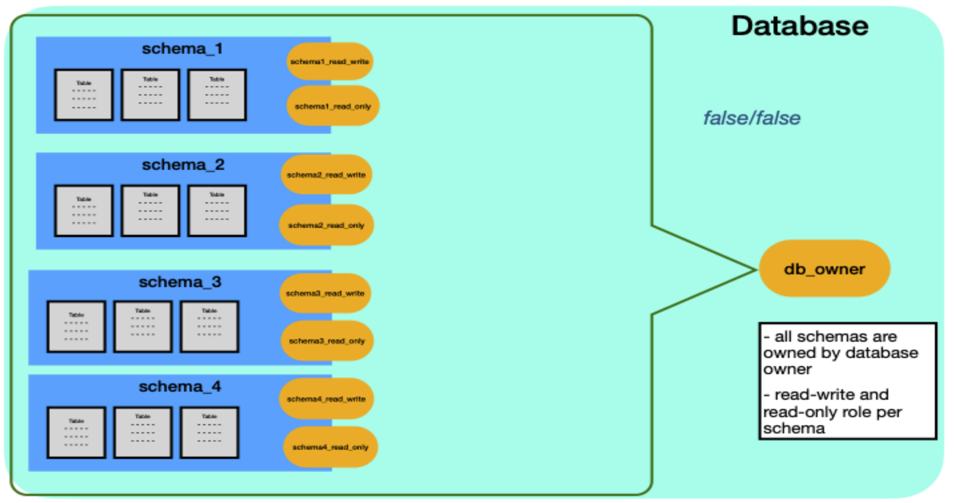
- Schemas and roles creation/deletion are performed using security definer functions

Event trigger

- Forces all objects in each schema to be owned by the schema owner role and assigns default privileges

Four Models

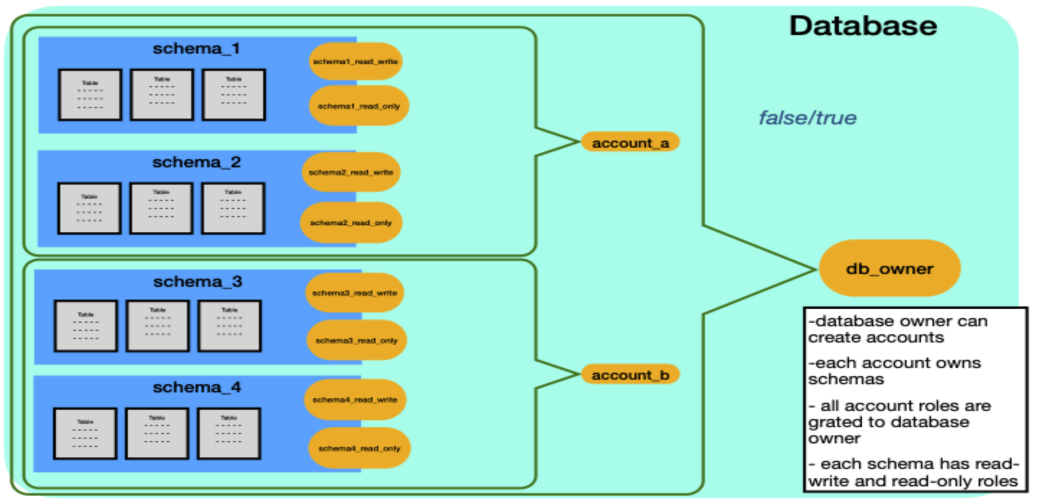
Single owner



- all schemas are owned by database owner

- read-write and read-only role per schema

Account owner



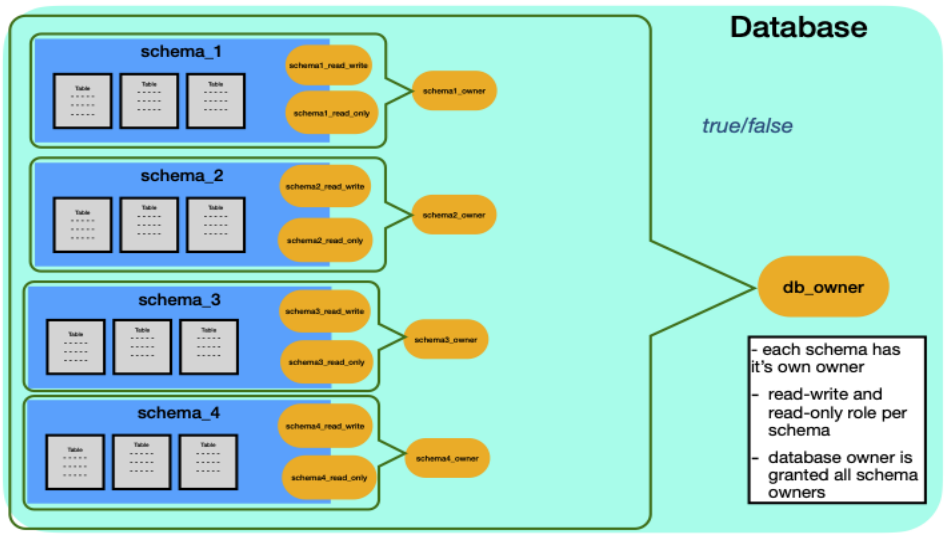
-database owner can create accounts

-each account owns schemas

- all account roles are grated to database owner

- each schema has read-write and read-only roles

Schema owner

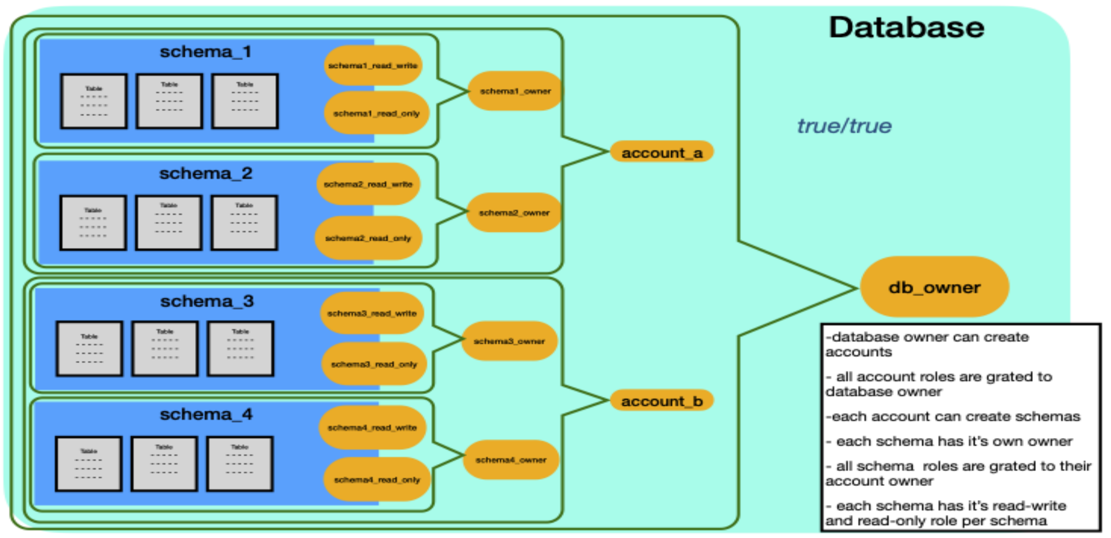


- each schema has it's own owner

- read-write and read-only role per schema

- database owner is granted all schema owners

Account and schema owner



-database owner can create accounts

- all account roles are grated to database owner

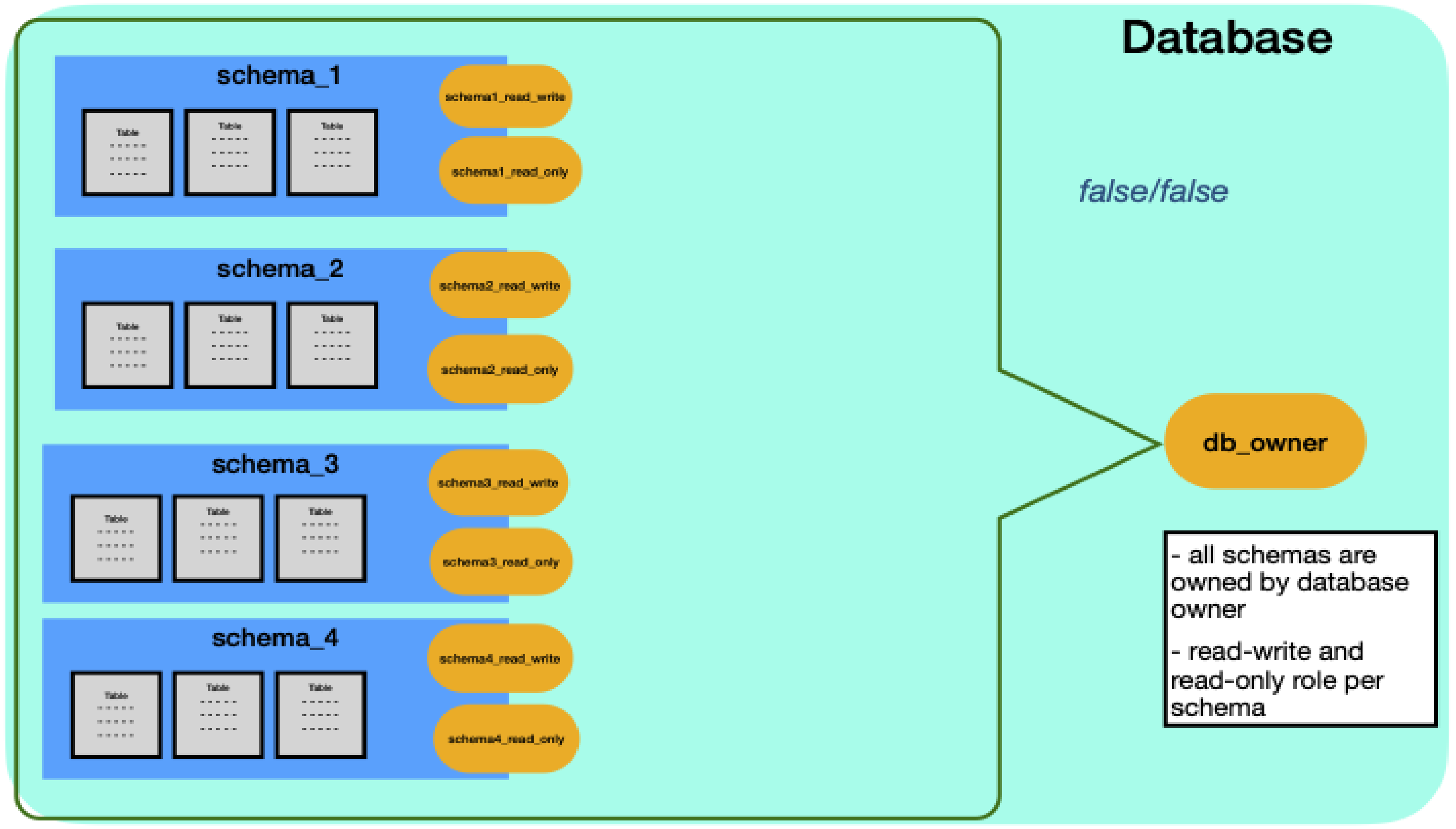
-each account can create schemas

- each schema has it's own owner

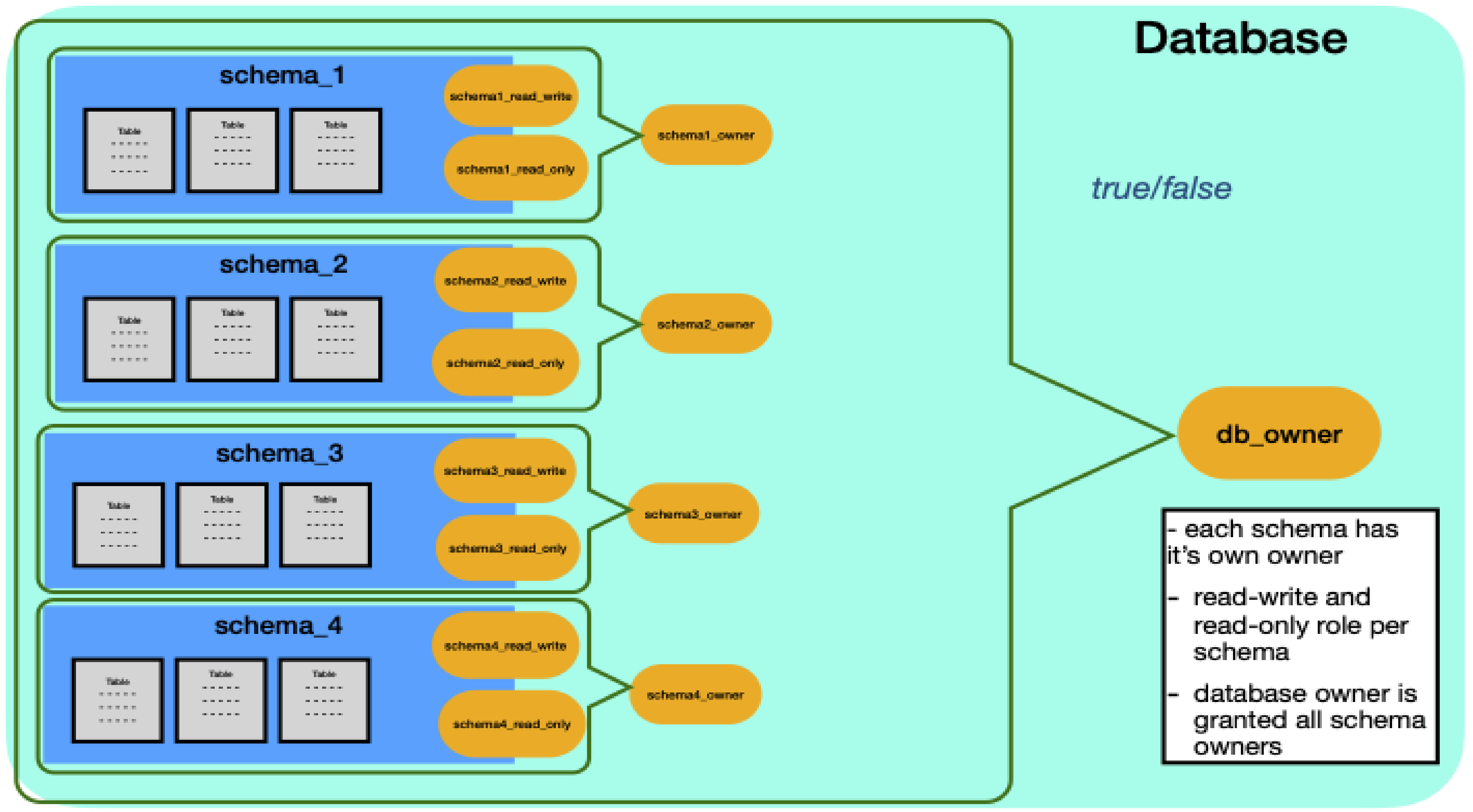
- all schema roles are grated to their account owner

- each schema has it's read-write and read-only role per schema

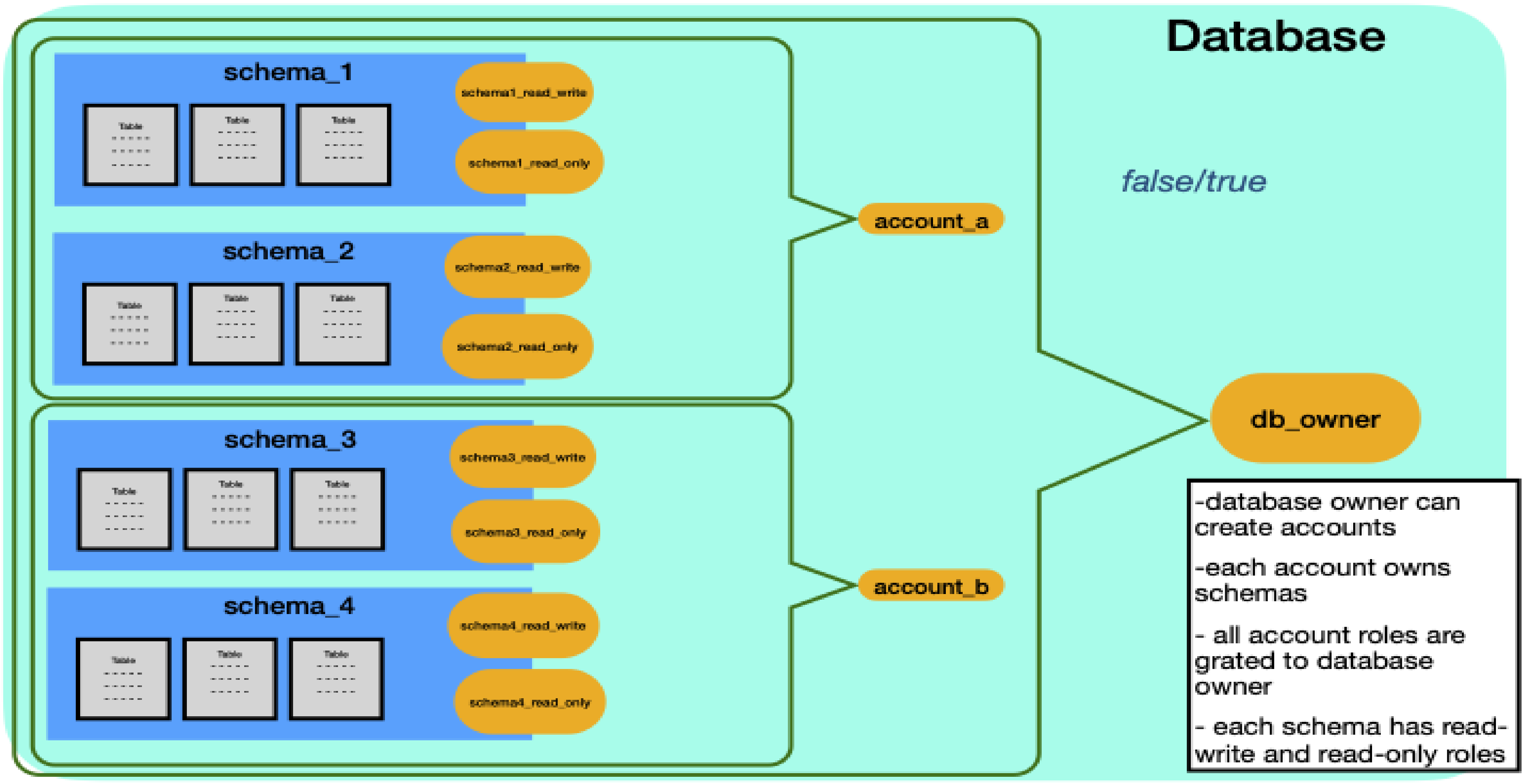
Single owner



Schema owner



Account owner



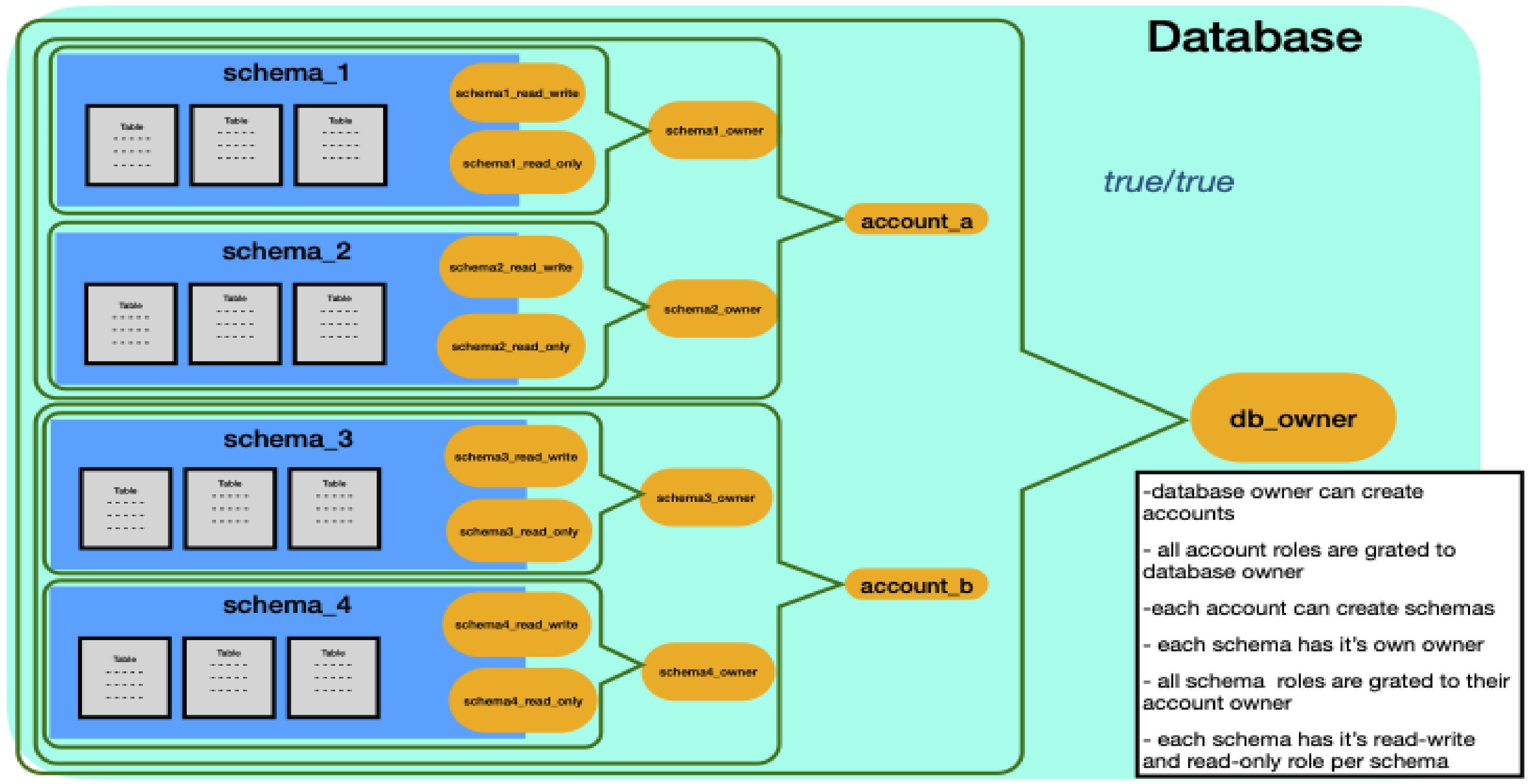
-database owner can create accounts

-each account owns schemas

- all account roles are grated to database owner

- each schema has read-write and read-only roles

Account and schema owner



-database owner can create accounts

- all account roles are grated to database owner

-each account can create schemas

- each schema has it's own owner

- all schema roles are grated to their account owner

- each schema has it's read-write and read-only role per schema

Enabling security model

- Deploy the package
- If the package was previously deployed, the previous security settings will be used:

  Changing existing settings is manual
- If that’s the first deployment run

```
select * from grant_create_schema_users(Boolean, Boolean)
```

This will

  - record security setting in the database
  - enable event trigger
  - grant execute on all security-definer functions to the database owner role

Functions

create_schema_roles

Input parameters:

- schema_name
- app_user_name (opt)
- app_user_password (opt)
- ro_user_name (opt)
- ro_user_password (opt)
- ddl_user_name (opt)
- ddl_user_password (opt)
- account_owner (opt, default = current user)

Actions:

- creates schema (ownership is driven by security settings)
- creates read_write role
- creates read_only role
- creates owner role (if applicable)
- creates/assigns app, ro and owner users

drop_schema_roles

Input parameters:

- schema_name

Actions:

- revokes read_only role from all users
- revokes read_write from all users
- revokes owner role (if applicable)
- drops all associated roles
- drops schema

assign_schema_owner_user

Input parameters:

- schema_name
- ddl_user_name
- ddl_user_password (opt)

Actions:

- creates user ddl_user_name if it does not exist
- changes password if user exists & password provided
- grants schema owner role to ddl_user_name

assign_schema_app_user

Input parameters:

- schema_name
- app_user_name
- app_user_password (opt)

Actions:

- creates user app_user_name if it does not exist
- changes password if user exists & password provided
- grants schema read_write role to app_user_name

assign_schema_ro_user

Input parameters:

- schema_name
- ro_user_name
- ro_user_password (opt)

Actions:

- creates user ro_user_name if it does not exist
- changes password if user exists & password provided
- grants schema read_only role to ro_user_name

Revoke functions

- `revoke_schema_owner_role`
- `revoke_schema_app_role`
- `revoke_schema_ro_role`

Additional security definer functions

- `select_all_privileges()`: all privileges on the current db
- `blocking_processes()`: blocking query with superuser privileges
- `pg_stat_activity()`: pg_stat_activity with superuser privileges

Code details

Event trigger forces new object ownership and permissions to the schema owner

```
FOR v_obj IN
  SELECT * FROM
pg_event_trigger_ddl_commands ()
 order by object_type desc
LOOP
 <fix perm>
END LOOP
```

Code details

Check whether the `current_user`:has an ownership role for this schema (`grant execute` is not enough)

```
select
 exists (
   with recursive x as
   (
     select member::regrole,
            roleid::regrole as role
      from pg_auth_members as m
      union all
     select x.member::regrole,
            m.roleid::regrole
      from pg_auth_members as m
      join x on m.member = x.role
     )
     select 1
     from x
     where
        (member::text = current_user
        and role = (select nspowner::regrole from pg_namespace
                                   where nspname=p_schema_name)
                           or current_user=     (select (nspowner::regrole)::text from pg_namespace
                                   where nspname=p_schema_name)
                             )     );
```

Code details

Checking the execution stack inside security definer function

```
if not
   perm_check_stack(
'dba_tools.perm_drop_schema_roles')
  then
  raise exception 'You are not allowed
to drop schema %', p_schema_name;
end if;
```

What is there for the users?

- No need for a new database when you start a new project
- Create new schemas
- Create new users
- Assign and revoke users’ privileges
- Change users’ passwords

You are in control!

Future work

And can we make it all happen in Postgres???

- Reporting
- Unit tests
- Conversion automation

DRW

Where to find me

Prairie Postgres

prairiepostgres.org

LinkedIn:

https://www.linkedin.com/in/henrietta-dombrovskaya-367b26/

GitHub:

https://github.com/hettie-d

Illinois Prairie PostgreSQL User Group

https://www.meetup.com/illinois-prairie-postgresql-user-group

Q&A

Hettie Dombrovskaya
Database Architect DRW

hdombrovska@drwholdings.com

www.drw.com