Die Definition des User-Models in PostgreSQL-Datenbank

Group Roles:

Es werden zwei Rollen (Gruppen) definiert:

- gr_planer
- gr_viewer

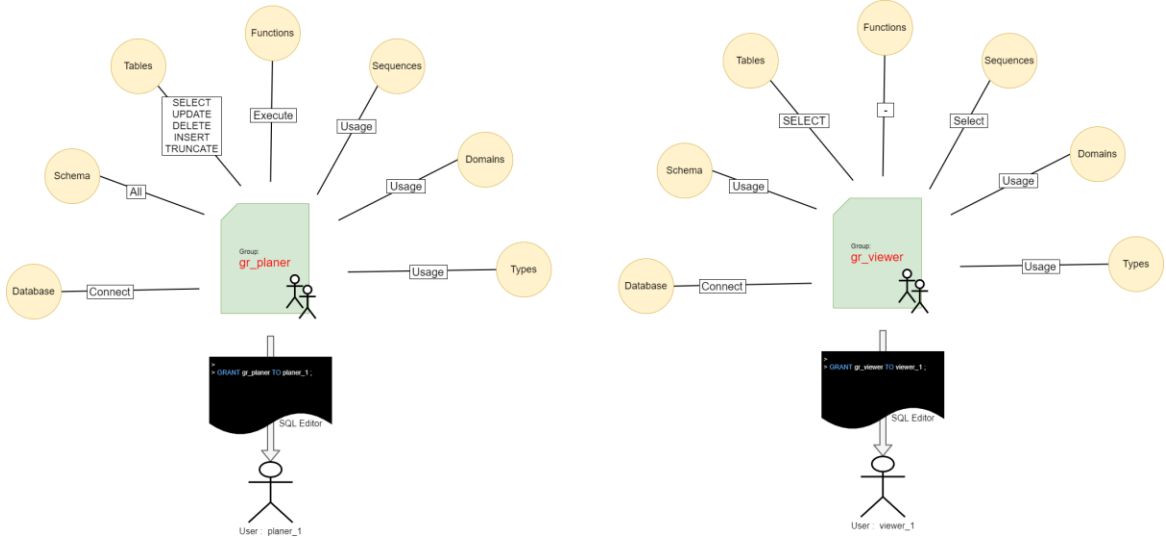
Sie wird nur einmal für den gesamten Cluster durchgeführt. Beachten Sie, dass wir es nicht noch einmal definieren sollten.

□ SQL Editor

```
CREATE ROLE gr_planer WITH NOSUPERUSER NOCREATEDB NOLOGIN NOREPLICATION;
CREATE ROLE gr_viewer WITH NOSUPERUSER NOCREATEDB NOLOGIN NOREPLICATION;
```

Das User-Modell:

Das entworfene Modell sieht wie folgt aus:



Definition der Privilegien für eine neue Datenbank:

Wenn eine Datenbank erstellt wird, kopieren Sie den angehängten SQL-Code in einen SQL-Editor und führen Sie ihn aus. Sie brauchen für diese Datenbank nichts mehr zu ändern. Alles wird von den definierten Triggern automatisch erledigt. Sie können leicht Benutzer hinzufügen oder entfernen.

Definition der Privilegien auf einer vorhandenen Datenbank:

Wenn eine Datenbank bereits vorhanden ist, kopieren Sie den angehängten SQL-Code in einen SQL-Editor und führen Sie ihn aus. Dabei werden die Privilegien auf bestehende Objekte angewendet. Sie brauchen für diese Datenbank nichts mehr zu ändern. Alles wird von den definierten Triggern automatisch erledigt. Sie können leicht Benutzer hinzufügen oder entfernen.

Einen neuen Benutzer hinzufügen:

Wann immer wir einen neuen Mitarbeiter als Planer haben, definieren wir einen Benutzer wie folgt (zum Beispiel *john* mit dem Passwort *1234*)

□ SQL Editor

```
CREATE ROLE john WITH NOSUPERUSER NOCREATEDB LOGIN NOREPLICATION    PASSWORD
'1234';
```

Mit dem folgenden Befehl werden dem Benutzer *john* die Privilegien für die Gruppe *gr_planer* zugewiesen. Wir müssen nicht für jeden Benutzer die Privilegien auf jeder Datenbank ändern. Wir tun dies nur für Gruppen und ordnen dann einen Benutzer diesen Gruppen zu.

□ SQL Editor

```
GRANT gr_planer TO john;
```

Widerrufen von Privilegien von einem Benutzer:

Wenn wir einen Benutzer behalten, aber die Privilegien einschränken wollen, gehen wir wie folgt vor:

□ SQL Editor

```
REASSIGNED OWNED BY john TO admin;

DROP OWNED BY john;

REVOKE gr_planer FROM john;
```

Benutzer löschen:

Wenn wir einen Benutzer entfernen wollen, müssen wir zunächst die Objekte, die er besitzt, an einen anderen Benutzer, z.B. den Benutzer *admin*, weitergeben. Dann löschen wir den Benutzer.

□ SQL Editor

```
REASSIGNED OWNED BY john TO admin;

DROP USER john;
```

Der SQL-Code des User-Modells:

```
/*
Definition of Privileges on a Database.

This code should be run on a database to give the privileges to the following roles(groups):
		gr_planer
		gr_viewer

with the following definitions:
		CREATE ROLE gr_planer WITH NOSUPERUSER NOCREATEDB NOLOGIN NOREPLICATION;
		CREATE ROLE gr_viewer WITH NOSUPERUSER NOCREATEDB NOLOGIN NOREPLICATION;

If the database already exists, it applies the privileges on the existin objects.
last Edit: 02-07-2020
Sayidi
*/

DO $$
DECLARE
		dbnam text; /* The name of database*/
		sch text; /* Temporary value*/
		schs text; /* The names of schemas*/
BEGIN
		SELECT current_database() INTO dbnam;
		FOR sch IN
			select schema_name from information_schema.schemata
			where schema_name <> 'information_schema' and schema_name !~ E'^pg_'
		LOOP
			IF schs is null then
			SELECT sch into schs;
			else
			SELECT concat(schs ,', ', sch) into schs;
			end if;
		END LOOP;
		/* User : gr_viewer*/
		EXECUTE 'GRANT CONNECT ON DATABASE "'||dbnam||'" TO gr_viewer;';
		EXECUTE 'GRANT SELECT ON ALL TABLES IN SCHEMA '|| schs || ' TO gr_viewer;';
		EXECUTE 'GRANT USAGE ON SCHEMA '||schs||' TO gr_viewer;';
		/* User : gr_planer*/
		EXECUTE 'GRANT CONNECT ON DATABASE "'||dbnam||'" TO gr_planer;';
		EXECUTE 'GRANT SELECT, UPDATE, DELETE, INSERT, TRUNCATE ON ALL TABLES IN SCHEMA '|| schs || ' TO gr_planer;';
		EXECUTE 'GRANT CREATE ON SCHEMA '||schs||' TO gr_planer;';
		EXECUTE 'GRANT SELECT ON ALL SEQUENCES IN SCHEMA '||schs||' TO gr_planer;';
		EXECUTE 'GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO gr_planer;';
		raise notice 'schs of the database %: %', dbnam, schs;
END$$;



-----------------------------------------------------------------------------------------------------------------------
/*Triggers:	*/
DROP EVENT TRIGGER IF EXISTS newdomain;
DROP EVENT TRIGGER IF EXISTS newfunction;
DROP EVENT TRIGGER IF EXISTS newschema;
DROP EVENT TRIGGER IF EXISTS newsequence;
DROP EVENT TRIGGER IF EXISTS newtable;
DROP EVENT TRIGGER IF EXISTS newtype;
-----------------------------------------------------------------------------------------------------------------------
--Schema :
CREATE OR REPLACE FUNCTION newschema() RETURNS event_trigger AS $schemacreation$
DECLARE
	obj record;
BEGIN
		raise notice 'new schema was created!';
		FOR obj IN SELECT * FROM pg_event_trigger_ddl_commands()
	LOOP
		RAISE NOTICE 'tag:%	classid:%	obj.object_type:%	obj.schema_name:%,	object_identity:%',
				tg_tag,
		  obj.classid,
				obj.object_type,
				obj.schema_name,
		  obj.object_identity;
		EXECUTE 'GRANT USAGE ON SCHEMA '||obj.object_identity||' TO gr_viewer;';
		EXECUTE 'GRANT ALL ON SCHEMA '||obj.object_identity||' TO gr_planer;';
	END LOOP;
END
$schemacreation$ LANGUAGE plpgsql;


CREATE EVENT TRIGGER newschema ON ddl_command_end WHEN TAG IN ('CREATE SCHEMA') EXECUTE PROCEDURE newschema();
-----------------------------------------------------------------------------------------------------------------------
-- Tables:
CREATE OR REPLACE FUNCTION newtable() RETURNS event_trigger AS $$
DECLARE
	obj record;
sequ text;
BEGIN
raise notice 'new Table was created!';
FOR obj IN SELECT * FROM pg_event_trigger_ddl_commands()
	LOOP
		RAISE NOTICE 'tag:%	classid:%	obj.object_type:%	obj.schema_name:%,	object_identity:%',
				tg_tag,
  obj.classid,
				obj.object_type,
				obj.schema_name,
  obj.object_identity;
IF obj.object_type in ('table', 'view', 'materialized view') THEN
EXECUTE 'GRANT SELECT ON TABLE '||obj.object_identity||' TO gr_viewer;';
EXECUTE 'GRANT SELECT, UPDATE, DELETE, INSERT, TRUNCATE ON TABLE '||obj.object_identity||' TO gr_planer;';
END IF;
	END LOOP;
END
$$ LANGUAGE plpgsql;
```

```
CREATE EVENT TRIGGER newtable ON ddl_command_end WHEN TAG IN ('CREATE TABLE','CREATE VIEW' , 'CREATE TABLE AS', 'CREATE MATERIALIZEd VIEW', 'CREATE FOREIGH TABLE') EXECUTE PROCEDURE newtable();
-----------------------------------------------------------------------------------------------------------------
-- sequences
CREATE OR REPLACE FUNCTION newsequence() RETURNS event_trigger AS $body$
DECLARE
	obj record;
BEGIN
		raise notice 'new Sequence was created!';
		FOR obj IN SELECT * FROM pg_event_trigger_ddl_commands()
	LOOP
		RAISE NOTICE 'tag:%	classid:%	obj.object_type:%	obj.schema_name:%,	object_identity:%',
				tg_tag,
			obj.classid,
				obj.object_type,
				obj.schema_name,
			obj.object_identity;
		EXECUTE 'GRANT USAGE ON SEQUENCE '||obj.object_identity||' TO gr_planer;';
		EXECUTE 'GRANT SELECT ON SEQUENCE '||obj.object_identity||' TO gr_viewer;';
	END LOOP;
END
$body$ LANGUAGE plpgsql;

CREATE EVENT TRIGGER newsequence ON ddl_command_end WHEN TAG IN ('CREATE SEQUENCE') EXECUTE PROCEDURE newsequence();
-----------------------------------------------------------------------------------------------------------------
-- functions
CREATE OR REPLACE FUNCTION newfunction() RETURNS event_trigger AS $body$
DECLARE
	obj record;
BEGIN
		raise notice 'new function was created!';
		FOR obj IN SELECT * FROM pg_event_trigger_ddl_commands()
	LOOP
		RAISE NOTICE 'tag:%	classid:%	obj.object_type:%	obj.schema_name:%,	object_identity:%',
				tg_tag,
			obj.classid,
				obj.object_type,
				obj.schema_name,
			obj.object_identity;
		IF obj.schema_name = 'public' THEN
				EXECUTE 'GRANT EXECUTE ON FUNCTION '||obj.object_identity||' TO gr_viewer;';
		END IF;
		EXECUTE 'GRANT EXECUTE ON FUNCTION '||obj.object_identity||' TO gr_planer;';
	END LOOP;
END
$body$ LANGUAGE plpgsql;

CREATE EVENT TRIGGER newfunction ON ddl_command_end WHEN TAG IN ('CREATE FUNCTION') EXECUTE PROCEDURE newfunction();
-----------------------------------------------------------------------------------------------------------------
-- Domain
CREATE OR REPLACE FUNCTION newdomain() RETURNS event_trigger AS $body$
DECLARE
	obj record;
BEGIN
		raise notice 'new domain was created!';
		FOR obj IN SELECT * FROM pg_event_trigger_ddl_commands()
	LOOP
		RAISE NOTICE 'tag:%	classid:%	obj.object_type:%	obj.schema_name:%,	object_identity:%',
				tg_tag,
			obj.classid,
				obj.object_type,
				obj.schema_name,
			obj.object_identity;
		EXECUTE 'GRANT USAGE ON DOMAIN '||obj.object_identity||' TO public;';
	END LOOP;
END
$body$ LANGUAGE plpgsql;

CREATE EVENT TRIGGER newdomain ON ddl_command_end WHEN TAG IN ('CREATE DOMAIN') EXECUTE PROCEDURE newdomain();
-----------------------------------------------------------------------------------------------------------------
-- Types
CREATE OR REPLACE FUNCTION newtype() RETURNS event_trigger AS $body$
DECLARE
	obj record;
BEGIN
		raise notice 'new type was created!';
		FOR obj IN SELECT * FROM pg_event_trigger_ddl_commands()
	LOOP
		RAISE NOTICE 'tag:%	classid:%	obj.object_type:%	obj.schema_name:%,	object_identity:%',
				tg_tag,
			obj.classid,
				obj.object_type,
				obj.schema_name,
			obj.object_identity;
		EXECUTE 'GRANT USAGE ON TYPE '||obj.object_identity||' TO public;';
	END LOOP;
END
$body$ LANGUAGE plpgsql;

CREATE EVENT TRIGGER newtype ON ddl_command_end WHEN TAG IN ('CREATE TYPE') EXECUTE PROCEDURE newtype();

 -----------------------------------------------------------------------------------------------------------------
```