Author : Chris Drawater
Date : 19/02/2007
Version : 1.0

PostgreSQL 8.2.1 – A User Management Example

Abstract

PostgreSQL account management can be quite daunting. A worked example can provided a useful source of ideas and 'how to do' codes examples.

Document Status

Introduction

This paper documents how PostgreSQL user accounts might be handled with PostgreSQL 8.2. It is made available as a source of ideas for those formulating their user account management strategy or policy.

It assumes that a production quality PostgreSQL databases has been successfully deployed (see Reference 1).

User categories

In the below examples, the basic idea is for a single PostgreSQL cluster to support only 3 user categories =>

1 'super user' → the DBA acct - eg *postgres* acct , which can create databases
1 'power user' → to own/manage the tables etc - eg *cxd* in these examples
n * 'end users' → to access data only through the applications

Background

Note that

- user accts are roles with logins
- groups can be considered roles with no login
- roles are cluster wide (across all databases)

and

- schemas are collections of databases objects within an individual database
- 1 databases can have n * schemas
- 1 role can own n * schemas
- Unless a specific named schema is specified, all objects are created in the default schema "public"
- The default schema (object search/creation) path for a role is *$user, public*

Creating the roles, schemas, paths and permissions

The below example creates 1 'power user' with its own schema which owns ALL the application tables, plus a N * 'end users' who can only access data via the aforementioned application tables (and cannot create their own objects). No tables will be created in the public schema.

So, for the below

cxd → 'power user'
endusers → group of 'end users'
enduser1 → 1 of the 'end user' group
db9 → the application database
postgres → the only 'super user'

First create the 'power user' →

$ psql db9 postgres

```
create user  cxd  with password 'abc';
grant create on tablespace appdata to cxd;
create schema authorization cxd;          -- create schema with same name as user – the default schema
```

Create the group role (and define appropriate schema permissions for that group) →

$ psql db9 postgres

```
 create role  endusers;                          -- ie group
grant usage  on schema cxd  to endusers;    -- can access ( subject to permissions) objects in cxd schema;
revoke  create on  schema  public  from public;     -- no object creation on public schema for any user
revoke create on schema  cxd  from  endusers;       -- no object creation on cxd schema
```

Define an ' end user' role (and make it a member of the *endusers* group role) →

$ psql db9 postgres

```
create user  enduser1  with password 'xyz';          -- ie a role with login
alter role enduser1 set search_path to cxd,public;   -- need login type role to  set search path
grant endusers to enduser1;                  --  same as  ALTER GROUP  endusers ADD USER enduser1;
```

Now a little verification →

$ psql db9 postgres

```
\du                              -- list users
select * from pg_roles;
\dn+                             -- list schema & privs
```

Now switch to the owner of the tables – the ' power user' – and create the tables/grants etc →

$ psql db9 cxd

create table t1 (val varchar(10)) tablespace appdata; -- created in cxd schema by default

grant select on t1 to group endusers;
-- revoke select on t1 from group endusers;

\dt+ -- list tables per schema
\dp *-- list table privs*

Verification

Log in as an 'enduser' account and verify the setup created in the previous section →

$ psql db9 enduser1

SHOW search_path; -- should show *cxd,public*

select has_schema_privilege('cxd', 'create'); -- should return false
select has_schema_privilege('cxd', 'usage'); -- should return true
select has_schema_privilege('public', 'create'); -- should return false
select has_tablespace_privilege ('appdata','create'); -- should return false

create table t2 (val varchar(10)); -- ERROR: permission denied for schema cxd
create table t2 (val varchar(10)) tablespace appdata; -- ERROR: permission denied for schema cxd

create table public.t2 (val varchar(10)); -- ERROR: permission denied for schema public

insert into t1 values ('test'); -- ERROR ERROR: permission denied for relation t1

*select * from t1;* -- success
*select * from cxd.t1;* -- success

The lazy way…

A simpler but less secure method is to keep all tables for all users etc in the *public* schema

Create a single 'power user' to create/own/control the tables →

$ psql db9 postgres

```
create user  cxd  with password 'abc';
grant create on tablespace appdata to cxd;
```

then create n* 'end user' accounts to work against the data →

$psql db9 postgres

```
create role  endusers;                      -- group of end users accounts
create user  enduser1  with password 'xyz'; -- create 1 end user login account/role
grant endusers  to enduser1;                -- ie ALTER GROUP endusers ADD USER enduser1;
```

Once the tables/indexes/procedures etc have been created by 'power user' *cxd* above, grants (ie access permissions) similar to below can then be granted to the 'end user' group.

$ psql db9 cxd

```
create table xt1 ( val varchar(10) ) tablespace appdata;
grant select  on  xt1 to group endusers;
```

Concluding Remarks

Keep your application data secure by careful control of user accounts and the scope of what those account are permitted to do or access. These worked examples have hopefully provided a few ideas on how to achieve that goal.

Chris Drawater has been working with DBMSs since 1987 and the JDBC API since late 1996, and can be contacted at *chris.drawater@ericssonservices.co.uk* or *drawater@btinternet.com* .

References

1. Drawater (2006), PostgreSQL 8.1 on Solaris 10 – Deployment Guidelines v1.1
   @ *http://www.postgresql.org/docs/techdocs.33*