openGauss-安全管理机制项目报告

> 主要内容：
>
> openGauss数据库权限管理模型、权限规划示例、权限识别项目具体实施方法

1. 项目信息

1.1 openGauss数据库权限管理模型

由于数据库中存储着大量重要数据和各类敏感信息，并且为持有不同权限的合法用户提供数据共享服务，这就要求数据库具备完善的安全防御机制来抵抗来自内部和外部的恶意攻击，以保障数据不丢失、隐私不泄露以及数据不被篡改等。当前openGauss数据库已经构建了纵深防御的安全体系，保障数据库在应用中的安全。完善的权限管理机制可以有效阻断恶意用户的越权操作。

常见的权限控制模型有三种：**基于策略的访问控制模型，基于角色的访问控制模型以及基于会话和角色的访问控制模型。** openGauss数据库采用**基于角色的权限访问控制模型(RBAC)**，利用角色来组织和管理权限，能够大大简化对权限的授权管理。借助角色机制，当给一组权限相同的用户授权时，只需将权限授予角色，再将角色授予这组用户即可，不需要对用户逐一授权。而且利用角色权限分离可以很好地控制不同用户拥有不同的权限，相互制约达到平衡。

1.1.1 基于角色的权限访问控制模型(RBAC)

`RBAC` 思想简单地说，一个用户拥有若干角色，每一个角色拥有若干权限，每一个角色拥有若干个菜单，这样，就构造成“用户-角色-权限”、“角色-菜单” 的授权模型。在这种模型中，用户与角色、角色与权限、角色与菜单之间构成了多对多的关系。

在openGauss数据库中，用户和角色是基本相同的概念，唯一的区别是在创建角色的时默认没有LOGIN权限，也不会自动创建同名的模 ，也就是说一个拥有LOGIN权限的角色可以被认为是一个用户。在以下的介绍中我们统一通过用户(USER) 来连接、访问数据库以及执行SQL，通过角色(ROLE) 来组织和管理权限。我们通过将不同的权限打包成角色授予用户，使得用户获得该角色中的所有权限。同时通过改变角色的权限，该角色所包含的所有成员的权限也会被自动修改。

在openGauss数据库系统中权限分为两种：**系统权限和对象权限。**

- 系统权限是指系统规定用户使用数据库的权限，比如登录数据库、创建数据库、创建用户/角色、创建安全策略等。
- 对象权限是指在数据库、模式、表、视图、函数等数据库对象上执行特殊动作的权限，不同的对象类型与不同的权限相关联，比如数据库的连接权限，表的查看、更新、插入等权限，函数的执行权限等。基于特定的对象来描述对象权限才是有意义的。

系统权限

系统权限又称用户属性，具有特定属性的用户会获得指定属性所对应的权限。**系统权限无法通过角色（ROLE）被继承。**在创建用户或角色时可以通过SQL语句CREATE ROLE/USER指定用户具有某些属性，或者通过ALTER ROLE/USER的方式给用户/角色添加用户属性或取消用户属性。

openGauss数据库支持如下系统权限的授予和回收：

系统权限	权限范围
SYSADMIN	允许用户创建数据库，创建表空间 允许用户创建用户/角色 允许用户查看、删除审计日志 允许用户查看其它用户的数据
MONADMIN	允许用户对系统模式dbe_perf及该模式下的监控视图或函数进行查看和权限管理
OPRADMIN	允许用户使用Roach工具执行数据库备份和恢复
POLADMIN	允许用户创建资源标签、创建动态数据脱敏策略和统一审计策略
AUDITADMIN	允许用户查看、删除审计日志
CREATEDB	允许用户创建数据库
USEFT	允许用户创建外表
CREATEROLE	允许用户创建用户/角色
INHERIT	允许用户继承所在组的角色的权限
LOGIN	允许用户登录数据库
REPLICATION	允许用户执行流复制相关操作

openGauss提供SQL语句CREATE/ALTER ROLE/USER实现系统权限的授予和回收，示例如下：

```
#例1：创建角色role1，同时授予role1创建数据库的权限
openGauss=# CREATE ROLE role1 WITH CREATEDB password 'openGauss@2021';
CREATE ROLE

#例2：授予角色role1监控管理员的权限，同时取消创建数据库的权限
openGauss=# ALTER ROLE role1 WITH MONADMIN NOCREATEDB;
ALTER ROLE

#例3：查看系统表pg_authid或系统视图pg_roles获取角色role1的相关信息
openGauss=# SELECT rolname,rolcreatedb,rolmonitoradmin FROM pg_authid WHERE rolname=
'role1';
 rolname | rolcreatedb | rolmonitoradmin
---------+-------------+-----------------
 role1   | f           | t
(1 row)
```

对象权限

对象所有者缺省具有该对象上的所有操作权限，比如修改、删除对象的权限，查看对象的权限，将对象的操作权限授予其他用户，或撤销已经授予的操作权限等。其中对象的ALTER、 DROP、COMMENT、INDEX、VACUUM以及对象的可再授予权限属于所有者固有的权限，隐式拥有。但对象所有者可以撤消自己的普通权限，例如，使表对自己以及其他人都只可读。

对象权限可以通过角色（ROLE）被继承，这样方便用户将这些单个的权限打包成一个角色进行权限管理。 openGauss数据库针对每一类数据库对象支持如下对象权限：

对象	权限	权限说明
TABLESPACE	CREATE	允许用户在指定的表空间中创建表
	ALTER	允许用户对指定的表空间执行ALTER语句修改属性
	DROP	允许用户删除指定的表空间
	COMMENT	允许用户对指定的表空间定义或修改注释
DATABASE	CONNECT	允许用户连接到指定的数据库
	TEMP	允许用户在指定的数据库中创建临时表
	CREATE	允许用户在指定的数据库里创建模式
	ALTER	允许用户对指定的数据库执行ALTER语句修改属性
	DROP	允许用户删除指定的数据库
	COMMENT	允许用户对指定的数据库定义或修改注释
SCHEMA	CREATE	允许用户在指定的模式中创建新的对象
	USAGE	允许用户访问包含在指定模式内的对象
	ALTER	允许用户对指定的模式执行ALTER语句修改属性
	DROP	允许用户删除指定的模式
	COMMENT	允许用户对指定的模式定义或修改注释
FUNCTION	EXECUTE	允许用户使用指定的函数
	ALTER	允许用户对指定的函数执行ALTER语句修改属性
	DROP	允许用户删除指定的函数
	COMMENT	允许用户对指定的函数定义或修改注释
TABLE	INSERT	允许用户对指定的表执行INSERT语句插入数据
	DELETE	允许用户对指定的表执行DELETE语句删除表中数据
	UPDATE	允许用户对指定的表执行UPDATE语句
	SELECT	允许用户对指定的表执行SELECT语句
	TRUNCATE	允许用户执行TRUNCATE语句删除指定表中的所有记录
	REFERENCES	允许用户对指定的表创建一个外键约束
	TRIGGER	允许用户在指定的表上创建触发器
	ALTER	允许用户对指定的表执行ALTER语句修改属性
	DROP	允许用户删除指定的表
	COMMENT	允许用户对指定的表定义或修改注释

对象	权限	权限说明
	INDEX	允许用户在指定表上创建索引，并管理指定表上的索引
	VACUUM	允许用户对指定的表执行ANALYZE和VACUUM操作

openGauss提供SQL语句GRANT/REVOKE实现对象权限的授予和回收：

```
#创建连接
[omm@localhost root]$ gsql -d postgres -r
gsql ((openGauss 3.1.0 build 4e931f9a) compiled at 2022-09-29 14:40:01 commit 0 last
mr  release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.
#创建测试数据
openGauss=# create database testdb;
CREATE DATABASE
openGauss=# \c testdb
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "testdb" as user "omm".
testdb=# create user test identified by 'test@123';
NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
testdb=# create user user1 identified by 'test@123';
NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
testdb=# alter database testdb owner to test;
ALTER DATABASE
testdb=# set search_path to test;
SET
testdb=# create table tbl1 (id int);
CREATE TABLE
testdb=# insert into tbl1 values(1),(2),(3);
INSERT 0 3


#例1：将对表tbl1进行select的权限以及将select再赋权的权限授予用户user1,
#赋权后用户user1有权对tbl执行select操作且user1有权限将select权限再赋予其他用户

[omm@home ~]$ gsql -d testdb -c "GRANT select ON TABLE test.tbl1 TO user1 WITH GRANT
OPTION"
GRANT
[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "select * from test.tbl1"
ERROR:  permission denied for schema test
LINE 1: select * from test.tbl1
                      ^
DETAIL:  N/A
#如上因为user1没有test模式的usage权限，所以即便给他授权了模式下的表的select权限也访问不了
gsql -d testdb -c "GRANT usage ON schema test TO user1"
[omm@home ~]$ gsql -d testdb -c "GRANT usage ON schema test TO user1"
GRANT
[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "select * from test.tbl1"
 id
----
  1
```

```
  2
  3
(3 rows)

#此时user1没有对表alter、drop的权限
[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "drop table test.tbl1"
ERROR:  permission denied for relation tbl1
DETAIL:  N/A
[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "alter table test.tbl1 add
column name text"
ERROR:  permission denied for relation tbl1
DETAIL:  N/A

#例2：将对表tbl1进行alter和drop的权限赋给用户user1
#赋权后用户user1有权对tbl1进行修改（ALTER）和删除（DROP）操作
[omm@home ~]$ gsql -d testdb -c "GRANT alter, drop ON TABLE test.tbl1 TO user1;"
GRANT
[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "alter table test.tbl1 add
column name text"
ALTER TABLE

#例3：撤销用户user1对表tbl进行select的权限
[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "select * from test.tbl1"
 id | name
----+------
  1 |
  2 |
  3 |
(3 rows)

[omm@home ~]$ gsql -d testdb -c "REVOKE select ON test.tbl1 FROM user1"
REVOKE
#撤销后用户user1对tbl进行select操作会报错：
[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "select * from test.tbl1"
ERROR:  permission denied for relation tbl1
DETAIL:  N/A
```

本次项目的目的就在于区分对象权限的情况：

```
opengauss=# \du
                                    List of roles
    Role name    |                         Attributes                         | Member of
-----------------+------------------------------------------------------------+-----------
admin_department | Sysadmin                                                   | {}
omm              | Sysadmin, Create role, Create DB, Replication, Administer audit, Monitoradmin, Operatoradmin, Policyadmin, UseFT | {}
opengauss        | Sysadmin                                                   | {}
test             |                                                            | {}
user1            |                                                            | {}
```

> `CREATE USER`
>
> 通过CREATE USER创建的用户，默认具有LOGIN权限。
> 通过CREATE USER创建用户的同时，系统会在执行该命令的数据库中，为该用户创建一个同名的SCHEMA。
> 系统管理员在普通用户同名schema下创建的对象，所有者为schema的同名用户（非系统管理员）。

`CREATE ROLE`

角色是拥有数据库对象和权限的实体。在不同的环境中角色可以认为是一个用户，一个组或者兼顾两者。
在数据库中添加一个新角色，角色无登录权限。
创建角色的用户必须具备CREATE ROLE的权限或者是系统管理员。

`schema`

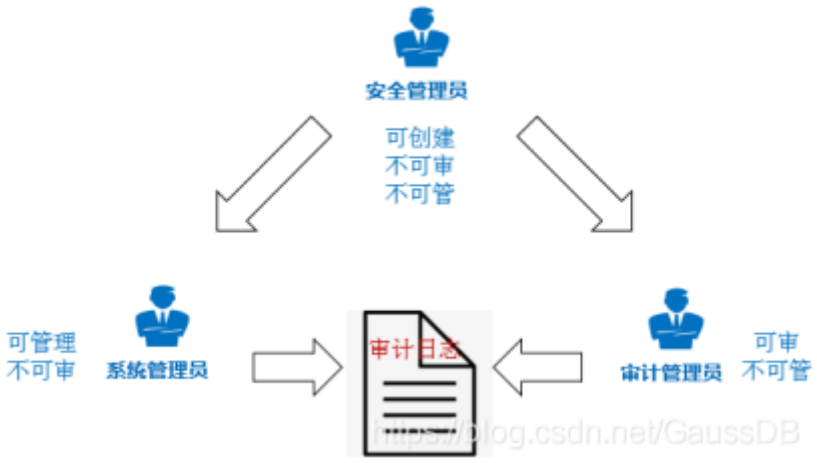
Schema又称作模式。通过管理Schema，允许多个用户使用同一数据库而不相互干扰，可以将数据库对象组织成易于管理的逻辑组，同时便于将第三方应用添加到相应的Schema下而不引起冲突。

1.1.2 三权分立机制

伴随着数据库的发展以及所面向业务场景的扩展，对数据库权限分离以及权限管理的细粒度划化提出了更高的要求，为了满足多样化用户的业务安全要求，openGauss数据库针对权限模型进行了更细粒度的权限划分，使得用户可以更灵活地依据实际业务进行用户权限分配和管理，除了基本系统权限和对象权限的划分外，还有一些高阶的权限管理机制用来满足客户的业务诉求，比如三权分立机制。

openGauss安装完成后会得到一个具有最高权限的超级用户。数据库超级用户的高权限意味着该用户可以做任何系统管理操作和数据管理操作,甚至可以修改数据库对象,包括接下来将要介绍的审计日志信息。对于企业管理来说，手握超级用户权限的管理人员可以在无人知晓的情况下改变数据行为，这带来的后果是不可想象的。

为了很好地解决权限高度集中的问题，在openGauss系统中引入三权分立角色模型，如图所示。三权分立角色模型最关键的**三个角色为安全管理员、系统管理员和审计管理员**。其中，安全管理员用于创建数据管理用户；系统管理员对创建的用户进行赋权；审计管理员则审计安全管理员、系统管理员、普通用户实际的操作行为。



通过三权分立角色模型实现权限的分派，且三个管理员角色独立行使权限，相互制约制衡。使得整个系统的权限不会因为权限集中而引入安全的风险。

事实上，产品使用过程中的安全是技术本身与组织管理双重保障的结果，在系统实现三权分立模型后，需要有三个对应的产品自然人分别握有对应的账户信息，以达到真正权限分离的目的。

三权分立是对系统权限管理机制的补充，核心思想是将管理数据库对象的权限、管理用户的权限和管理审计日志的权限分离，从而避免一个管理员拥有过度集中的权利带来的高风险。通过将GUC参数`enableSeparationOfDuty`设置为on来打开三权分立开关。

```
openGauss=# select name,setting,unit,context from pg_settings where name ~
'enableSeparationOfDuty';
          name          | setting | unit |  context
------------------------+---------+------+------------
 enableSeparationOfDuty | off     |      | postmaster
(1 row)
```

```
ALTER SYSTEM SET enableSeparationOfDuty TO 'on';
-- 开启分离
```

三权分立开关打开后， SYSADMIN的权限范围将缩小，不再包括允许创建用户/角色的权限，也不再包括允许查看、删除数据库审计日志的权限。SYSADMIN，CREATEROLE，AUDITADMIN三种系统权限的权限范围互相隔离，互不影响，而且一个用户仅能被赋予其中一个属性。

三权分立打开后的权限范围如下：

系统权限	权限范围
SYSADMIN	允许用户创建数据库，创建表空间
CREATEROLE	允许用户创建用户/角色
AUDITADMIN	允许用户查看、删除审计日志

```
SHOW audit_enabled; #查询审计是否开启
```

列级访问控制

在一些业务场景中，数据表中的某些列存储了重要的信息，需要对用户不可见，但其他列的数据又需要用户能够查看或操作，此时就需要针对数据表的特定列做访问控制，实现针对用户的列级别的访问控制。

openGauss提供SQL语句GRANT/REVOKE实现针对列对象的权限授予和回收：

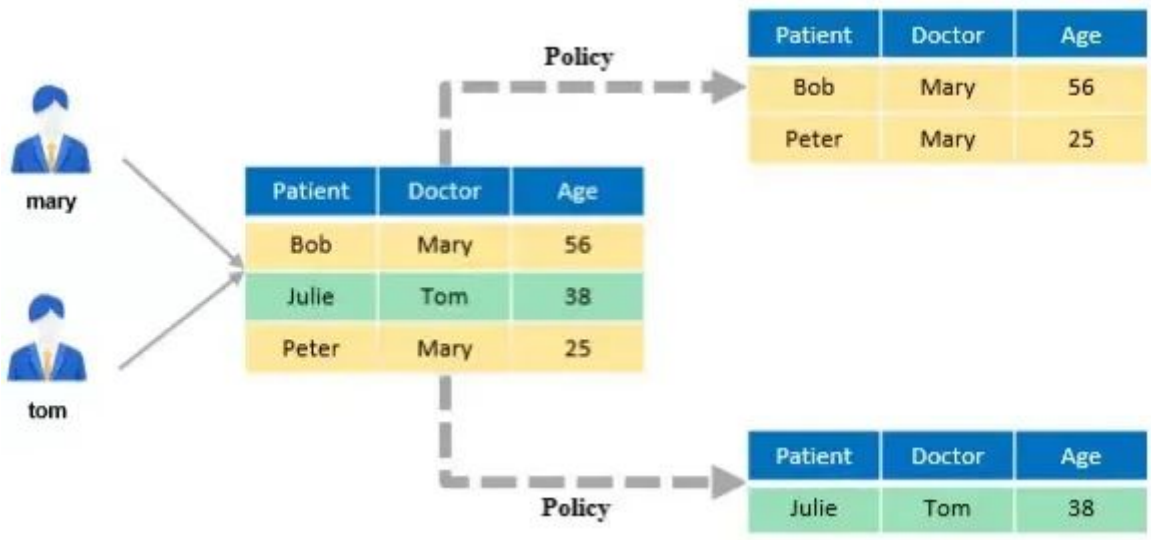
```
#创建测试数据
gsql -d postgres -r
create database testdb;
\c testdb
create user test identified by 'test@123';
create user user1 identified by 'test@123';
alter database testdb owner to test;
set search_path to test;
create table tbl (id int,name varchar(20));
insert into tbl values(1,'test1'),(2,'test2'),(3,'test3');


#例1：将对表tbl的第一列(id)进行select的权限和对表tbl的第二列(name)进行update的权限授予用户user1
#赋权后用户user1有权对tbl的第一列执行select操作和对第二列执行update操作
[omm@home ~]$ gsql -d testdb -c "GRANT select(id),update(name) ON TABLE test.tbl TO
user1;"
GRANT
[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "select id from test.tbl"
ERROR:  permission denied for schema test
```

```
LINE 1: select id from test.tbl
                      ^
DETAIL:  N/A
#如上因为user1没有test模式的usage权限，所以即便给他授权了模式下的表的select权限也访问不了
[omm@home ~]$ gsql -d testdb -c "GRANT usage ON schema test TO user1"
GRANT
[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "select id from test.tbl"
 id
----
  1
  2
  3
(3 rows)
[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "select name from test.tbl"
ERROR:  permission denied for relation tbl
DETAIL:  N/A

[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "update test.tbl set name =
'haha' where id=3"
UPDATE 1
[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "update test.tbl set id = 4
where id=3"
ERROR:  permission denied for relation tbl
DETAIL:  N/A

#例2：撤销用户user1对表tbl的第一列id进行select的权限
#撤销后用户user1不再具有查看表tbl的第一列id数据的权限
[omm@home ~]$ gsql -d testdb -c "REVOKE select(id) ON test.tbl FROM user1"
REVOKE
[omm@home ~]$ gsql -d testdb -U user1 -W test@123  -c "select id from test.tbl"
ERROR:  permission denied for relation tbl
DETAIL:  N/A
```

行级访问控制

在实际业务中还存在另外一种场景，同一张数据表，只允许用户查看满足特定条件的行数据，此时就需要将访问控制精确到数据表的行级别，使得不同用户执行相同的SQL查询、更新或删除操作，读取到的结果是不同的。

用户可以在数据表上创建行级访问控制(row level security)策略，该策略是针对特定数据库用户、特定SQL操作生效的表达式。当数据库用户访问数据表时，满足策略条件的行对用户可见，不满足条件的行对用户不可见，从而实现针对用户的行级别的访问控制。



openGauss提供SQL语句CREATE/ALTER/DROP ROW LEVEL SECURITY进行行级访问权限策略的创建/修改/删除操作：

```
#创建测试数据
gsql -d postgres -r
create database testdb;
\c testdb
create user test identified by 'test@123';
create user mary identified by 'test@123';
create user tom identified by 'test@123';
alter database testdb owner to test;

set search_path to test;
#步骤1：创建信息表pat_info记录医院病人的个人信息：
create table pat_info(patience varchar(20),doctor varchar(20),age int);
insert into pat_info values('peter','mary',25),('bob','mary',56),('julie','tom',38)

#查询表数据
[omm@home ~]$ gsql -d testdb -U test -W test@123  -c "select * from test.pat_info"
 patience | doctor | age
----------+--------+-----
 peter    | mary   |  25
 bob      | mary   |  56
 julie    | tom    |  38
(3 rows)



#步骤2：创建行级访问控制策略，使得医生只能查看属于自己的病人信息：
[omm@home ~]$ gsql -d testdb -c "CREATE ROW LEVEL SECURITY POLICY rls_select ON
test.pat_info FOR select USING(doctor=current_user)"
CREATE ROW LEVEL SECURITY POLICY



#步骤3：打开信息表pat_info上的行级访问控制开关
[omm@home ~]$ gsql -d testdb -c "ALTER TABLE test.pat_info ENABLE ROW LEVEL
SECURITY;"
ALTER TABLE
```

```
#步骤4：将信息表pat_info的查看权限赋予所有人
[omm@home ~]$ gsql -d testdb -c "grant select on table test.pat_info to public;"
GRANT


#步骤5：Mary医生的查看结果：
[omm@home ~]$ gsql -d testdb -U mary -W test@123  -c "select * from test.pat_info"
ERROR:  permission denied for schema test
LINE 1: select * from test.pat_info
                      ^
DETAIL:  N/A
[omm@home ~]$
[omm@home ~]$ gsql -d testdb -c "GRANT usage ON schema test TO mary"
GRANT
[omm@home ~]$ gsql -d testdb -U mary -W test@123  -c "select * from test.pat_info"
 patience | doctor | age
----------+--------+-----
 peter    | mary   |  25
 bob      | mary   |  56
(2 rows)



#Tom医生的查看结果：
[omm@home ~]$ gsql -d testdb -c "GRANT usage ON schema test TO tom"
GRANT
[omm@home ~]$ gsql -d testdb -U tom -W test@123  -c "select * from test.pat_info"
```

1.2 权限查询命令示例

> vmware启动后opengauss可能没有打开，需要人工开启：
>
> ```
> su - omm  #有空格 有'-'
> ```

1、查询当前数据所有用户及其权限：\du

```
opengauss=# \du
                                   List of roles
    Role name    |                         Attributes                         | Member of
-----------------+------------------------------------------------------------+-----------
 admin_department | Sysadmin                                                   | {}
 omm              | Sysadmin, Create role, Create DB, Replication, Administer audit, Monitoradmin, Operatoradmin, Policyadmin, UseFT | {}
 opengauss        | Sysadmin                                                   | {}
 test             |                                                            | {}
 user1            |                                                            | {}
```

或者通过 `PG_USER\pg_authid` 可以查看数据库中所有用户的列表，还可以查看用户ID（USESYSID）和用户权限。

```
SELECT * FROM pg_user;
SELECT * FROM pg_authid;
```

2、查看数据库中包含的表

例如，在 `PG_TABLES` 系统表中查看public schema中包含的所有表。

```
SELECT distinct(tablename) FROM pg_tables WHERE SCHEMANAME = 'public';
```

```
create user testUser identified by 'test@123';
-- 查询用户
SELECT * FROM pg_user;
-- 查询角色
SELECT * FROM pg_roles;
-- 在 PostgreSQL 中，创建一个用户（user）时，会自动创建一个与该用户同名的角色（role）。
-- 用户和角色在 PostgreSQL 中是密切相关的概念，可以说用户就是一个具有登录能力的角色。
```

3、查询用户数据库的权限：

```
opengauss=# select a.datname,b.rolname,string_agg(a.pri_t,',') from (select datname,
(aclexplode(COALESCE(datacl, acldefault('d'::"char",datdba")))).grantee as grantee,
(aclexplode(COALESCE(datacl, acldefault('d'::"char", datdba")))).privilege_type as
pri_t from pg_database where datname not like 'template%') a,pg_roles b where
(a.grantee=b.oid or a.grantee=0) and b.rolname='opengauss' group by
a.datname,b.rolname;

    datname    |  rolname  |                string_agg
---------------+-----------+-------------------------------------------
 testdb        | opengauss | TEMPORARY,CONNECT
 db_department | opengauss | TEMPORARY,CONNECT
 postgres      | opengauss | TEMPORARY,CONNECT
 opengauss     | opengauss | TEMPORARY,CONNECT,CREATE,TEMPORARY,CONNECT
(4 rows)
```

显示用户opengauss对于opengauss数据库具有 `TEMPORARY,CONNECT,CREATE,TEMPORARY,CONNECT` 等权限

4、根据用户名查询table 权限，可以通过视图 `information_schema.table_privileges` 来查看，为了方便展示，sql如下

```
opengauss=# select table_name,table_schema,grantee,string_agg(privilege_type,',')
from information_schema.table_privileges where grantee='opengauss' group by
table_name,table_schema,grantee;
```

具体解释如下：

- `SELECT table_name, table_schema, grantee, string_agg(privilege_type, ',')`：选择 `table_name`、`table_schema`、`grantee` 和使用逗号分隔的权限类型（privilege_type）的聚合结果。
- `FROM information_schema.table_privileges`：从 `information_schema` 系统表中查询表级别的权限信息。
- `WHERE grantee='test'`：筛选出授权对象（grantee）为 `test` 的记录。
- `GROUP BY table_name, table_schema, grantee`：按照 `table_name`、`table_schema` 和 `grantee` 进行分组。

通过执行这条查询语句，可以获取用户或角色 `test` 对每个表的权限信息。查询结果将包括表的名称、所属的模式（schema）、授权对象（即 `test`）、以逗号分隔的权限类型列表。这可以帮助管理员查看特定用户或角色在数据库中有权访问哪些表，并了解其所具有的权限类型。

```
opengauss=# select table_name,table_schema,grantee,string_agg(privilege_type,',') from information_schema.table_privileges where grantee='omm' group by table_
name,table_schema,grantee;
          table_name          |    table_schema    | grantee |                    string_agg
------------------------------+--------------------+---------+-----------------------------------------------
 sql_sizing                   | information_schema | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 global_stat_bad_block        | dbe_perf           | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 triggers                     | information_schema | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 usage_privileges             | information_schema | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 attributes                   | information_schema | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 summary_statio_user_tables   | dbe_perf           | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 global_config_settings       | dbe_perf           | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 session_memory               | dbe_perf           | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 global_statio_all_tables     | dbe_perf           | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 view_column_usage            | information_schema | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 domain_udt_usage             | information_schema | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 pg_shadow                    | pg_catalog         | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 role_table_grants            | information_schema | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 gs_labels                    | pg_catalog         | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 global_transactions_prepared_xacts | dbe_perf     | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 statio_all_indexes           | dbe_perf           | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 summary_workload_sql_elapse_time | dbe_perf       | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 user_login                   | dbe_perf           | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 workload_transaction         | dbe_perf           | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 replication_stat             | dbe_perf           | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 memory_node_detail           | dbe_perf           | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 summary_stat_user_indexes    | dbe_perf           | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 character_sets               | information_schema | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 check_constraint_routine_usage | information_schema | omm   | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
 check_constraints            | information_schema | omm     | INSERT,SELECT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER
```

如图所示，omm对于各个表都具有权限。

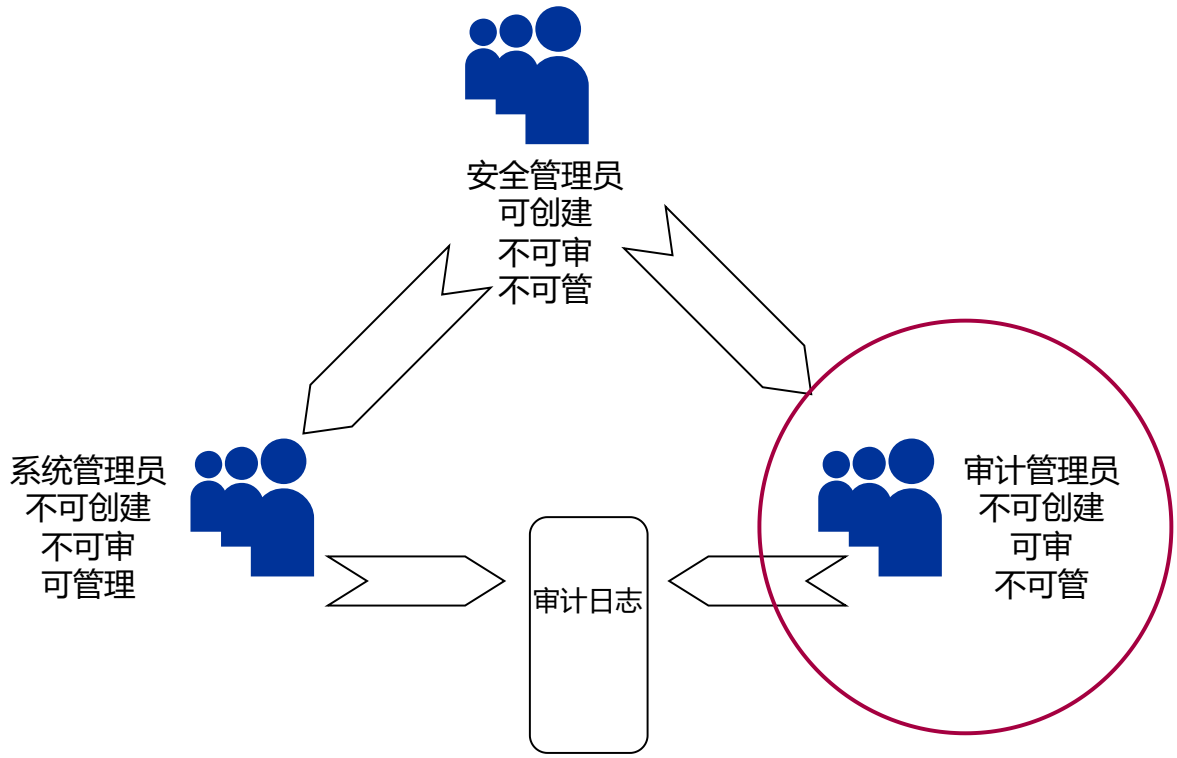
1.3 方案描述

需求：

设计用户权限检查项，开发扫描程序，实现数据库权限扫描功能，扫描数据库中是否有违规操作的可能和安全隐患，完成设计文档。

1.3.1 三权分立权限安全策略

扫描当前数据库中安全管理员、系统管理员和审计管理员，三权分立开启之后，权限管理更加严格，如图无法给审计管理员赋予系统管理员的权限。



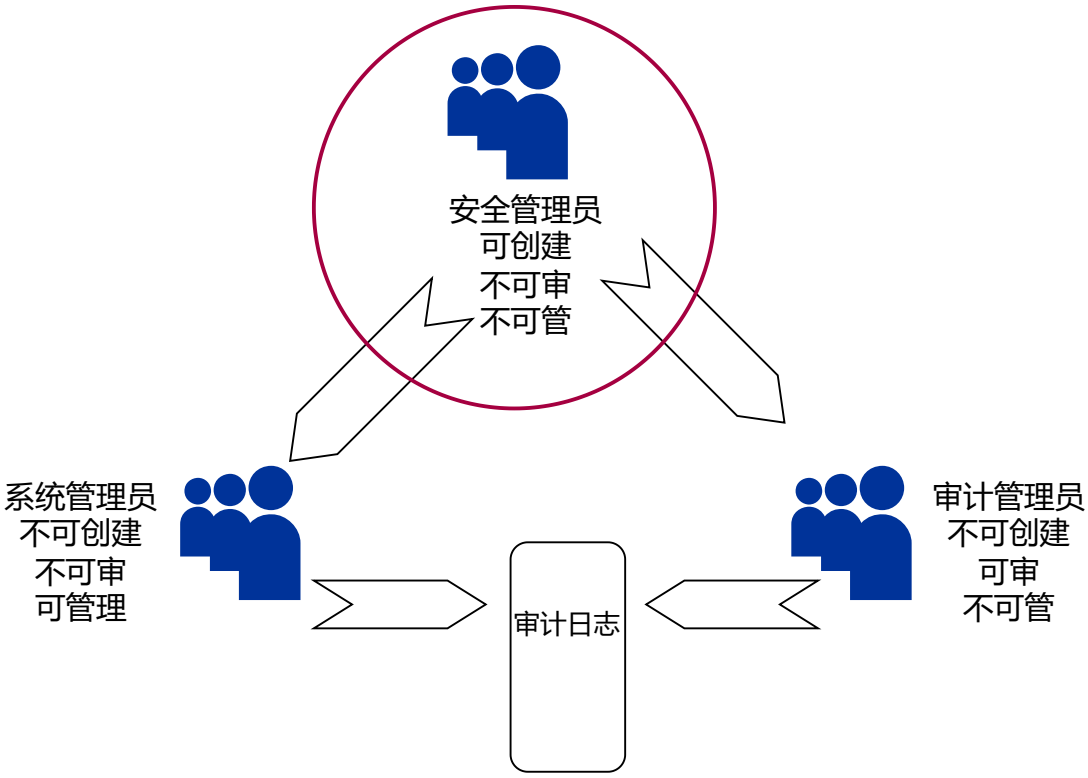
1）安全管理员rolcreaterole：

```
CREATE USER poladmin WITH CREATEROLE password "gauss@123";
# 决定一个角色是否可以创建新角色（也就是执行CREATE ROLE和CREATE USER）。 一个拥有CREATEROLE权限的角色也可以修改和删除其他角色。

SELECT rolname,rolsuper,rolcreaterole,rolsystemadmin,rolauditadmin FROM pg_roles
WHERE rolcreaterole = 'true';

SELECT *
FROM pg_roles
WHERE rolcreaterole = 'true' AND rolname ≠ 'omm';
-- 如果有查询结果表明存在违规
```

非**三权分立**时，只有系统管理员和具有CREATEROLE属性的系统管理员用户才能创建、修改或删除角色。三权分立下，只有初始用户和具有CREATEROLE属性的安全管理员的用户才能创建、修改或删除角色，下面以安全管理员为例：

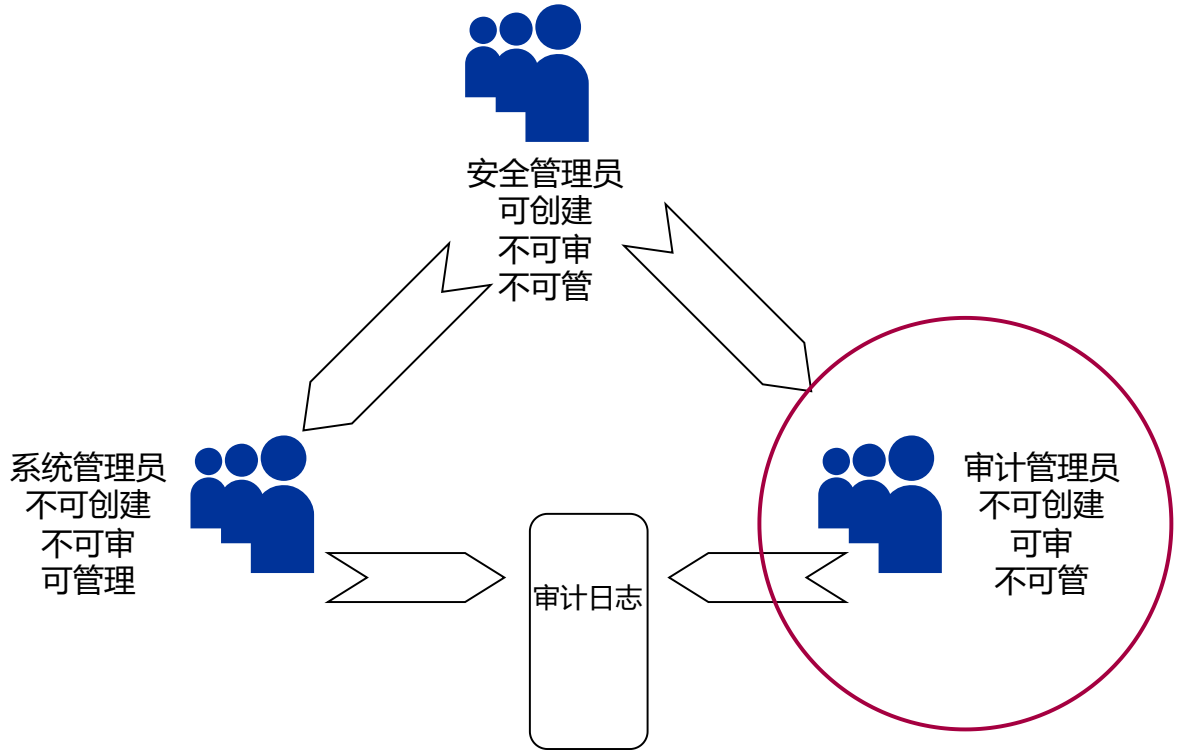
- 查询createrole的用户，只能是具有该createrole权限，不能具有审计和系统管理权限。

```
select table_name,table_schema,grantee,string_agg(privilege_type,',') from
information_schema.table_privileges where grantee='audadmin' group by
table_name,table_schema,grantee;
```



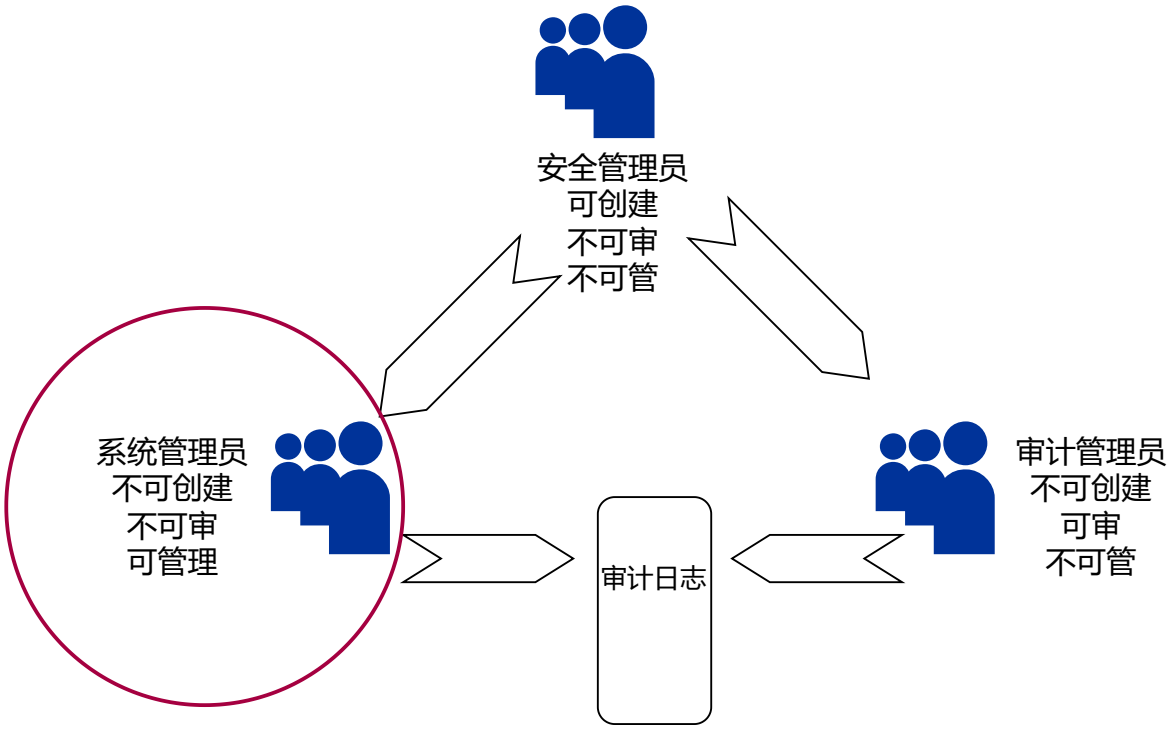
2） 审计管理员rolauditadmin

```
CREATE USER poladmin WITH AUDITADMIN password "gauss@123";

# AUDITADMIN | NOAUDITADMIN  定义角色是否有审计管理属性。
SELECT *
FROM pg_roles
WHERE rolauditadmin = 'true' AND rolname ≠ 'omm';
```



3）系统管理员rolsystemadmin

```
CREATE USER poladmin WITH SYSADMIN password "gauss@123";

# SYSADMIN | NOSYSADMIN  决定一个新角色是否为“系统管理员”，具有SYSADMIN属性的角色拥有系统最高权限。缺省为NOSYSADMIN。
SELECT *
FROM pg_roles
WHERE rolsystemadmin = 'true' AND rolname ≠ 'omm';
```



1.3.2 表格、数据库权限安全策略

1）查询用户对各个表的权限

实现思路：

根据SQL语句：

```
SELECT grantee AS rol_name, table_name, string_agg(privilege_type, ', ') AS
privileges FROM information_schema.role_table_grants GROUP BY grantee,table_name;
```

设计SpringBoot的restful接口，使用mybtais负责dao，编写entity层，entity层的名字叫`Rol_Table_Privilege`，编写service层给出controller，编写对应的网页的layui风格的使用这个接口的表格的ajax代码.

展示结果为：

还可以根据指定用户查询所有数据表或者指定数据表的权限：

SQL代码为：

```
 select table_name,table_schema,grantee,string_agg(privilege_type,',') from
information_schema.table_privileges where grantee='admin_department' group by
table_name,table_schema,grantee;
-- 查询当前数据库下当前用户对表的权限
```

2）查询用户对各个数据库的权限

实现思路：

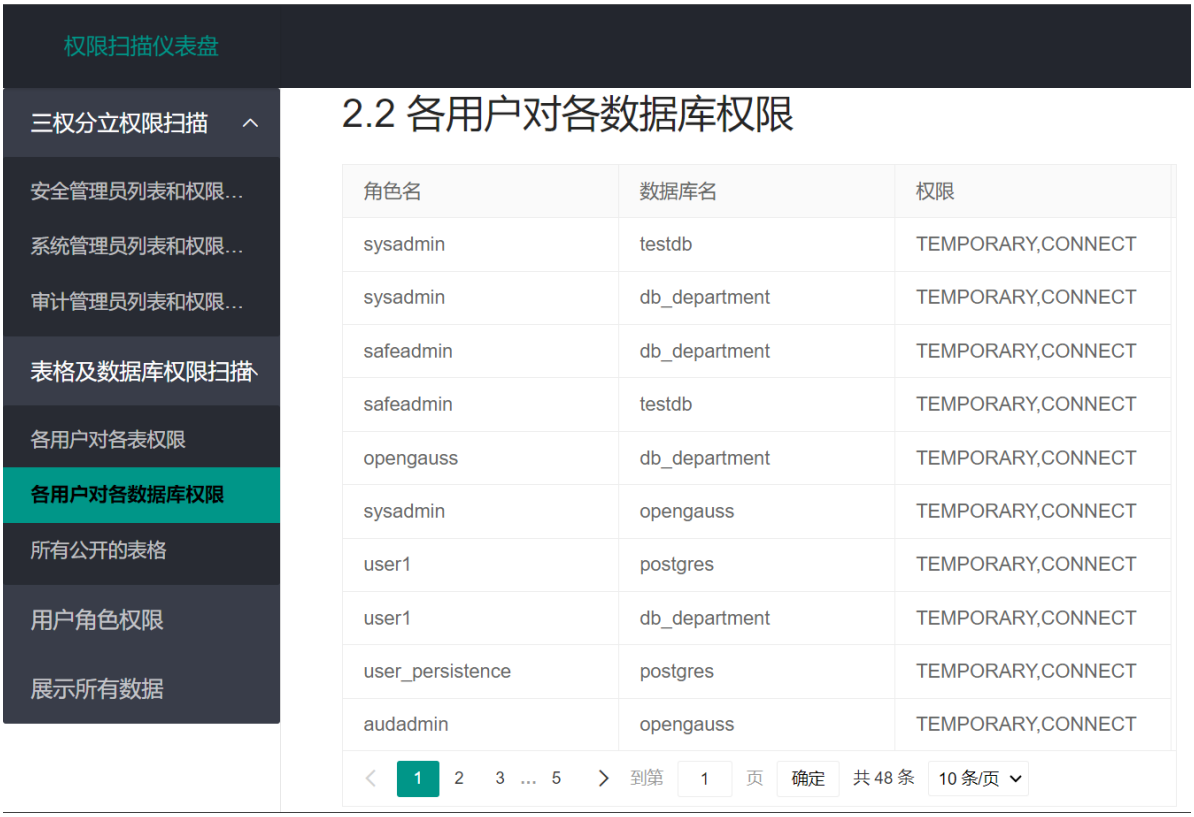
根据SQL语句：

```
SELECT b.rolname, a.datname, string_agg(a.pri_t, ',') AS privileges
FROM (
    SELECT datname, grantee, privilege_type AS pri_t
```

```
    FROM (
        SELECT
            datname,
            (aclexplode(COALESCE(datacl, acldefault('d'::"char", datdba")))).grantee
AS grantee,
            (aclexplode(COALESCE(datacl, acldefault('d'::"char",
datdba")))).privilege_type AS privilege_type
        FROM pg_database
        WHERE datname NOT LIKE 'template%'
    ) subquery
) a
JOIN pg_roles b ON a.grantee = b.oid OR a.grantee = 0
WHERE b.rolname NOT LIKE 'gs%'
GROUP BY a.datname, b.rolname;

-- 优化为一行的版本为:
SELECT b.rolname, a.datname, string_agg(a.pri_t, ',') AS privileges FROM (SELECT
datname, grantee, privilege_type AS pri_t FROM (SELECT datname,
(aclexplode(COALESCE(datacl, acldefault('d'::"char", datdba")))).grantee AS grantee,
(aclexplode(COALESCE(datacl, acldefault('d'::"char", datdba")))).privilege_type AS
privilege_type FROM pg_database WHERE datname NOT LIKE 'template%') subquery) a JOIN
pg_roles b ON a.grantee = b.oid OR a.grantee = 0 WHERE b.rolname NOT LIKE 'gs%'
GROUP BY a.datname, b.rolname;
```

设计SpringBoot的restful接口，使用mybtais负责dao，编写entity层，entity层的名字叫`Rol_Table_Privilege`，编写service层给出controller，编写对应的网页的layui风格的使用这个接口的表格的ajax代码.

展示结果为:



同理还可以根据指定用户查询所有数据库或者指定数据库的权限。

3）当前数据库的表格

实现思路：

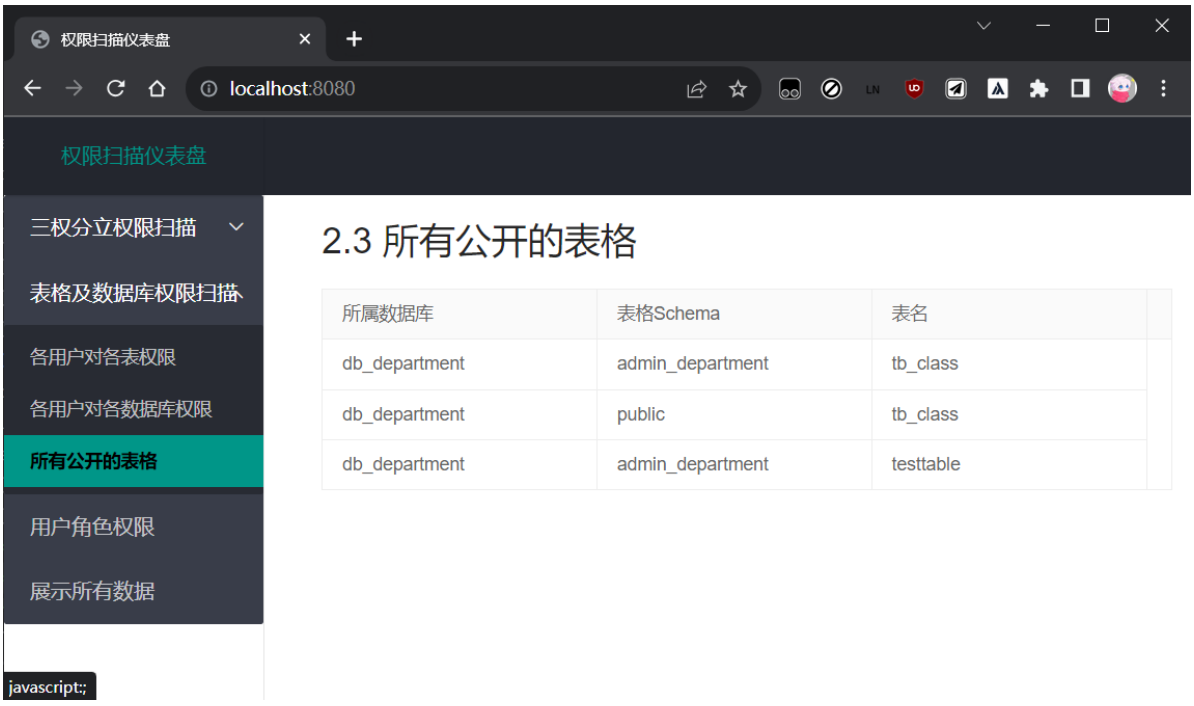
根据SQL语句：

```
SELECT table_catalog AS database,table_schema,table_name
FROM information_schema.tables
WHERE table_type = 'BASE TABLE'
    AND table_schema NOT IN ('pg_catalog',
'information_schema','dbe_pldeveloper','db4ai');
-- 查询当前所有公开的表格，包括：所在数据库，所在schema，名字 Table
```

设计SpringBoot的restful接口，使用mybtais负责dao，编写entity层，entity层的名字叫`Table`，编写service层给出controller，编写对应的网页的layui风格的使用这个接口的表格的ajax代码.

```

GRANT SELECT ON SCHEMA public TO testuser;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO testuser;
-- 授予 testuser 用户在 public 模式下所有表的查询权限，包括 tb_class 表

 GRANT SELECT ON tb_class TO testuser;
-- 授予 testuser 用户 tb_class 表的查询权限。
```

效果：



1.3.3 角色、用户权限安全策略

1）管理员可以有多个，但是安全管理员只能有一个

```
SELECT rolname FROM pg_roles WHERE rolsuper = true;
-- 查询初始用户也就是超级管理员
```

实现思路：

根据所有用户的列表中，我们可以查找超级管理员的数据，如果超过一个，就存在安全隐患，展示出隐患信息。

2）查询所有用户、角色信息：

实现思路：

根据SQL语句：

```
SELECT usesysid,usename,usesuper
FROM pg_user;
```

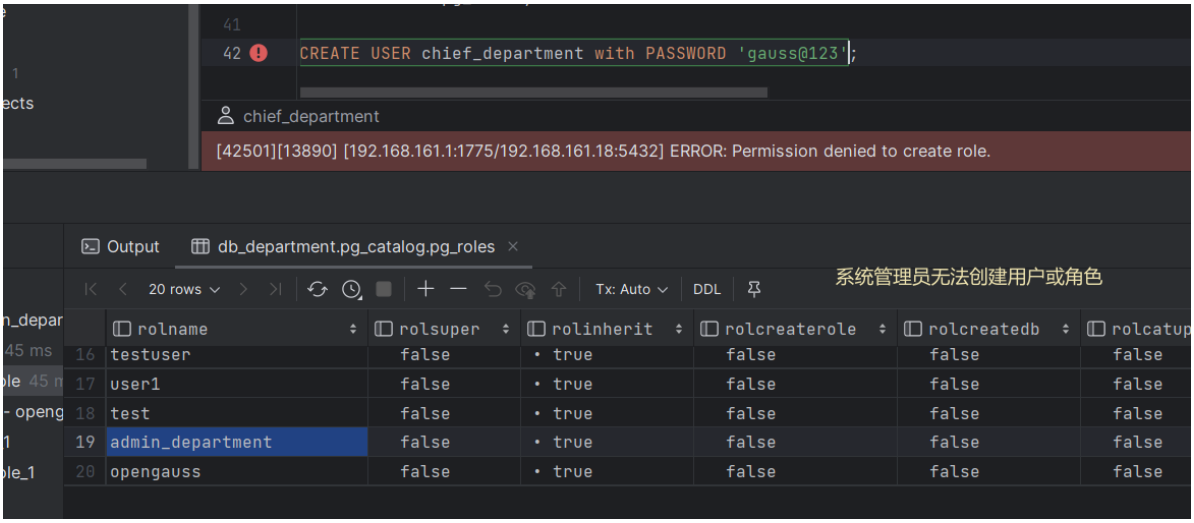
设计SpringBoot的restful接口，使用mybtais负责dao，编写entity层，entity层的名字叫`PgUser`，编写service层给出controller，编写对应的网页的layui风格的使用这个接口的表格的ajax代码.

展示结果为：

3）禁止新建以“gs_role ”开头的用户/角色，也禁止将已有的用户/角色重命名为以“gs_role ”开头；

```
SELECT * FROM pg_user;
-- 查询的结果不能包含gs_role开头，否则存在违规

SELECT *
FROM pg_user
WHERE usename LIKE 'gs_role%';
-- 如果有查询结果表明存在违规
```

4）非三权分立下，openGauss用户帐户只能由系统管理员rolsystemadmin或拥有CREATEROLE属性的安全管理员创建和删除。三权分立时，用户帐户只能由初始用户omm和安全管理员rolcreaterole创建。



如上图，Data的连接账号的是admin_department，是一个系统管理员(sysadmin)，不是安全管理员，因此无法创建账户，如果没有开启三权分立，那么就可以创建账号。

2. 项目进度

2.1 已完成工作

根据原定方案和时间规划，完成一篇技术文档， 完成一个主要是能识别数据库三权分立的管理员权限，识别用户或角色对数据库和表格的操作权限，识别所有用户列表的项目，已经将项目提交PR到 https://gitee.com/opengauss/examples/pulls/55 。

2.2 遇到的问题及解决方案

问题：访问数据库用户和角色权限

解决方案：通过SQL查询系统表格得到数据。

问题：openGauss数据安全策略设计

解决方案：研读三权分立的原理和三权分立的基本要求，通过SQL查询得到的权限数据，评判数据库权限安全。

2.3 后续工作安排

继续完善安全策略的设计，或许可以将安全评判的标准设计进入数据库内部，集成为一个功能。

附录：

参考文档：

1、 **https://blog.csdn.net/myneth/article/details/129036436**

2、 **https://www.kancloud.cn/sinkiang/skadmin_document/1267757**

3、schema介绍： **https://www.jb51.net/article/275164.htm**

4、 **RDS实践**

5、 **sks-admin**

6、 **postgresql查询权限**

7、 **gauss5.0手册**