权限管控，还可以再简单点

原创 xiongcc 2024-05-25 401

前言

PostgreSQL 中的权限设计略微复杂，属于层次结构，对于新手不太友好，老鸟也会经常云里雾里。举个栗子，假如你需要查询某一行数据的话，让我捋捋，你大概需要经过这么多层关卡：

1. 先通过数据库防火墙
2. 能够登录对应数据库 (LOGIN)
3. 表所在数据库的连接权限 (CONNECT)
4. 表所在模式的使用权限 (USAGE)
5. 表本身的查看权限 (SELECT)
6. 列级权限 (SELECT)
7. 行级策略 (RLS)

实名劝退不少新手小白，还有角色 (ROLE)、用户 (USER)和组 (GROUP) 这些十分类似的概念，更不用说还可以角色套娃了。关于权限我也写了不少文章：

1. 刨根问底 | 如何删除用户最优？
2. 又被权限搞晕了？拿捏！
3. 再唠唠晕乎的权限体系

今天向各位分享几个实用插件，简单易懂，快速上手，帮助各位轻松拿捏这个晕乎的权限体系。

权限查询

crunchy_check_access

第一个推荐的插件是 crunchy_check_access

> Functions and views to facilitate PostgreSQL object access inspection

在 MySQL 中，如果要获取一个用户的所有权限，很方便，一条语句就可以搞定，但是在 PostgreSQL 中，很不幸，你需要拼接大量的表才能获取出可能并不完整的权限列表，crunchy_check_access 便可以解决这样的难题。

```
postgres=# create extension check_access ;
CREATE EXTENSION
postgres=# \dx+ check_access
   Objects in extension "check_access"
            Object description
-----------------------------------------
 function all_access()
 function all_access(boolean)
 function all_grants()
 function all_grants(boolean)
 function check_access(text,boolean)
 function check_access(text,boolean,text)
 function check_grants(text,boolean)
 function check_grants(text,boolean,text)
 function my_privs()
 function my_privs_sys()
 view my_privs
 view my_privs_sys
(12 rows)
```

比如你想 postgres 用户对于表的相关权限

```
postgres=# SELECT * FROM all_access() WHERE base_role = 'postgres' and objtype
 role_path | base_role | as_role  | objtype | objid  | schemaname |   objname
-----------+-----------+----------+---------+--------+------------+------------
 postgres  | postgres  | postgres | table   | 167849 | public     | enrollments
 postgres  | postgres  | postgres | table   | 167849 | public     | enrollments
 postgres  | postgres  | postgres | table   | 167849 | public     | enrollments
 postgres  | postgres  | postgres | table   | 167849 | public     | enrollments
 postgres  | postgres  | postgres | table   | 167849 | public     | enrollments
(5 rows)
```

对于库的相关权限

```
postgres=# SELECT * FROM all_access() WHERE base_role = 'postgres' and objtype
 role_path | base_role | as_role  | objtype  | objid | schemaname | objname  |
-----------+-----------+----------+----------+-------+------------+----------+-
 postgres  | postgres  | postgres | database | 33405 |            | postgres |
 postgres  | postgres  | postgres | database | 33405 |            | postgres |
 postgres  | postgres  | postgres | database | 33405 |            | postgres |
 postgres  | postgres  | postgres | database | 33405 |            | postgres |
 postgres  | postgres  | postgres | database | 33405 |            | postgres |
(5 rows)
```

更多用法各位读者自行尝试吧，还是十分方便的！

pg_permissions

第二个插件是 pg_permissions

> This extension allows you to review object permissions on a PostgreSQL database.

pg_permissions 支持**权限对比**，DBA 在管理权限时只需要在 permission_target 表中录入相应的权限，后期开发人员或 DBA 在开发阶段可能会随意的分配权限，比如权限超了，只需要查询结果还可以写入等，那么 DBA 只需要运行对比函数就能发现权限是否符合设计

要求，防患于未然。

```
postgres=# \dx+ pg_permissions
Objects in extension "pg_permissions"
         Object description
------------------------------------
 function permission_diffs()
 function permissions_trigger_func()
 sequence permission_target_id_seq
 table permission_target
 type obj_type
 type perm_type
 view all_permissions
 view column_permissions
 view database_permissions
 view function_permissions
 view schema_permissions
 view sequence_permissions
 view table_permissions
 view view_permissions
(14 rows)
```

权限对比参照它自己展示的例子即可：

```
SELECT * FROM public.permission_diffs();
```

复制

missing	role_name	object_type	schema_name	object_name	column_name
f	laurenz	VIEW	appschema	appview	
t	appuser	TABLE	appschema	apptable	

(2 rows)

> That means that `appuser` is missing the `DELETE` privilege on `appschema.apptable` which should be granted, while user `laurenz` has the additional `SELECT` privilege on `appschema.appview` (`missing` is `FALSE`).

另外一个功能类似于 crunchy_check_access，可以查询某个用户的权限列表，对应于 *_permissions相对应的视图，比如

```
postgres=# select * from table_permissions limit 2;
 object_type |     role_name      | schema_name |   object_name    | column_name
-------------+--------------------+-------------+------------------+-------------
 TABLE       | pg_database_owner | public      | pgbench_branches |
 TABLE       | pg_database_owner | public      | pgbench_history  |
(2 rows)

postgres=# select * from column_permissions limit 2;
 object_type |     role_name      | schema_name |  object_name   | column_name |
-------------+--------------------+-------------+----------------+-------------+-
 COLUMN      | pg_database_owner | public      | ptab01_202303 | id          |
 COLUMN      | pg_read_all_data  | public      | ptab01_202303 | id          |
(2 rows)
```

灰常好用！

实用 SQL

如果各位觉得装插件麻烦的话，那有没有懒人专用的 SQL 呢？Sure！

```
WITH server_permissions AS (
        SELECT
            r.rolname,
            'Server_Permissions' AS "Level",
            r.rolsuper,
            r.rolinherit,
            r.rolcreaterole,
            r.rolcreatedb,
            r.rolcanlogin,
            ARRAY(
                SELECT b.rolname
                FROM pg_catalog.pg_auth_members m
                JOIN pg_catalog.pg_roles b ON m.roleid = b.oid
                WHERE m.member = r.oid
```

```
            ) AS memberof,
            r.rolbypassrls
        FROM pg_catalog.pg_roles r
        WHERE r.rolname !~ '^pg_'
    ),

    db_ownership AS (
        SELECT
            r.rolname,
            'DB_Ownership' AS "Level",
            d.datname
        FROM pg_catalog.pg_database d, pg_catalog.pg_roles r
        WHERE d.datdba = r.oid
    ),

    schema_permissions AS (
        SELECT
            'Schema Permissions' AS "Level",
            r.rolname AS role_name,
            nspname AS schema_name,
            pg_catalog.has_schema_privilege(r.rolname, nspname, 'CREATE') AS cr
            pg_catalog.has_schema_privilege(r.rolname, nspname, 'USAGE') AS usa
        FROM pg_namespace pn, pg_catalog.pg_roles r
        WHERE array_to_string(nspacl, ',') LIKE '%' || r.rolname || '%'
              AND nspowner > 1
    ),

    table_ownership AS (
        SELECT
            'Table Ownership' AS "Level",
            tableowner,
            schemaname,
            tablename
        FROM pg_tables
        GROUP BY tableowner, schemaname, tablename
    ),

    object_permissions AS (
        SELECT
```

```
            'Object Permissions' AS "Level",
            COALESCE(NULLIF(s[1], ''), 'public') AS rolname,
            n.nspname,
            relname,
            CASE
                WHEN relkind = 'm' THEN 'Materialized View'
                WHEN relkind = 'p' THEN 'Partitioned Table'
                WHEN relkind = 'S' THEN 'Sequence'
                WHEN relkind = 'I' THEN 'Partitioned Index'
                WHEN relkind = 'v' THEN 'View'
                WHEN relkind = 'i' THEN 'Index'
                WHEN relkind = 'c' THEN 'Composite Type'
                WHEN relkind = 't' THEN 'TOAST table'
                WHEN relkind = 'r' THEN 'Table'
                WHEN relkind = 'f' THEN 'Foreign Table'
            END AS "Object Type",
            s[2] AS privileges
        FROM
            pg_class c
            JOIN pg_namespace n ON n.oid = relnamespace
            JOIN pg_roles r ON r.oid = relowner,
            UNNEST(COALESCE(relacl::text[], FORMAT('{%s=arwdDxt/%s}', rolname,
            REGEXP_SPLIT_TO_ARRAY(acl, '=|/') s
        WHERE relkind <> 'i' AND relkind <> 't'
    )

    SELECT
        "Level",
        rolname AS "Role",
        'N/A' AS "Object Name",
        'N/A' AS "Schema Name",
        'N/A' AS "DB Name",
        'N/A' AS "Object Type",
        'N/A' AS "Privileges",
        rolsuper::text AS "Is SuperUser",
        rolinherit::text,
        rolcreaterole::text,
        rolcreatedb::text,
        rolcanlogin::text,
        memberof::text,
```

```
        rolbypassrls::text
    FROM server_permissions

    UNION

    SELECT
        dow."Level",
        dow.rolname,
        'N/A',
        'N/A',
        datname,
        'N/A',
        'N/A',
        'N/A',
        'N/A',
        'N/A',
        'N/A',
        'N/A',
        'N/A',
        'N/A'
    FROM db_ownership AS dow

    UNION

    SELECT
        "Level",
        role_name,
        'N/A',
        schema_name,
        'N/A',
        'N/A',
        CASE
            WHEN create_grant IS TRUE AND usage_grant IS TRUE THEN 'Usage+Creat
            WHEN create_grant IS TRUE AND usage_grant IS FALSE THEN 'Create'
            WHEN create_grant IS FALSE AND usage_grant IS TRUE THEN 'Usage'
            ELSE 'None'
    END,
    'N/A',
    'N/A',
    'N/A',
```

```
        'N/A',
        'N/A',
        'N/A',
        'N/A'
    FROM schema_permissions

    UNION

    SELECT
        "Level",
        tableowner,
        tablename,
        schemaname,
        'N/A',
        'N/A',
        'N/A',
        'N/A',
        'N/A',
        'N/A',
        'N/A',
        'N/A',
        'N/A'
    FROM table_ownership

    UNION

    SELECT
        "Level",
        rolname,
        relname,
        nspname,
        'N/A',
        "Object Type",
        privileges,
        'N/A',
        'N/A',
        'N/A',
        'N/A',
        'N/A',
```

```
        'N/A',
        'N/A'
    FROM object_permissions
    ORDER BY "Role";
```

权限限制

set_user

原生 PostgreSQL 权限有很多"漏洞"，比如无法阻止 alter system，copy on program 搞破坏 (之前还有 CVE) 等等

> This PostgreSQL extension allows switching users and optional privilege escalation with enhanced logging and control. It provides an additional layer of logging and control when unprivileged users must escalate themselves to superuser or object owner roles in order to perform needed maintenance tasks.

- The current effective user becomes `rolename`.
- The role transition is logged, with a specific notation if `rolename` is a superuser.
- `log_statement` setting is set to "all", meaning every SQL statement executed while in this state will also get logged.
- If `set_user.block_alter_system` is set to "on", `ALTER SYSTEM` commands will be blocked.
- If `set_user.block_copy_program` is set to "on", `COPY PROGRAM` commands will be blocked.
- If `set_user.block_log_statement` is set to "on", `SET log_statement` and variations will be blocked.
- If `set_user.block_log_statement` is set to "on" and `rolename` is a database superuser, the current `log_statement` setting is changed to "all", meaning every SQL statement executed

- If `set_user.superuser_audit_tag` is set, the string value will be appended to `log_line_prefix` upon superuser escalation. All logs after superuser escalation will be tagged with the value of `set_user.superuser_audit_tag`. This value defaults to `'AUDIT'`.
- If `set_user.exit_on_error` is set to "on", the backend process will exit on ERROR during calls to set_session_auth().
- Post-execution hook for `set_user` is called if it is set.

这个插件就是提供了一些此类限制，用法很简单。

pg_restrict

pg_restrict 类似，多了个禁止删库这个操作，原生 PostgreSQL 是无法阻止删库的 (除非你设置为模板数据库)

- `pg_restrict.alter_system` (boolean): restrict ALTER SYSTEM command to master roles (`pg_restrict.master_roles` parameter). Default is *false*.
- `pg_restrict.copy_program` (boolean): restrict COPY ... PROGRAM command to master roles (`pg_restrict.master_roles` parameter). Default is *false*.
- `pg_restrict.master_roles` (string): Roles that are allowed to execute the restricted commands. If there is more than one role, separate them with comma. Default is *postgres*.
- `pg_restrict.nonremovable_databases` (string): restrict DROP databases listed here to a master role (even if the current role is the database owner or superuser). Default is *postgres, template1, template0*.
- `pg_restrict.nonremovable_roles` (string): restrict DROP roles listed here to a master role (even if the current role has CREATEROLE privilege or is a superuser). Default is *postgres*.

pg_sulog

pg_sulog 主要针对的是超级用户，让超级用户也不能肆无忌惮的搞破坏了！但是很久不维护了，有点脱裤子放屁的味道，与其这样，还给超级用户的权限做什么。

- 'BLOCK', super user role's all operation is blocked.
- 'MAINTENANCE', Other than the following commands, super user operation will be blocked.
    - VACUUM, REINDEX, ANALYZE, CLUSTER
- 'LOGGING', super user role's all operation is logged.

小结

权限还是需要多动手，多折腾几下，其实也就那么回事。

墨力计划

「喜欢这篇文章，您的关注和赞赏是给作者最好的鼓励」

关注作者　赞赏

分享你的看法，一起交流吧～

相关阅读

腾讯iOA企业级安全办公解决方案

2025年8月中国数据库排行榜：双星竞入三甲榜，TDSQL 连跃位次升

【DBA坦白局】第三期：作为DBA，你加过最晚的班是到几点？在干什么？

2025年7月国产数据库大事记：GoldenDB创千万级大单，可信数据库大会召开，openGauss HyBench打榜第一，电科金仓举办2025产品发布会……

IDC报告：2024中国金融行业集中式事务型数据库市场破11.6亿元，Oracle领跑、达梦强势追赶

优炫数据库在山东省寿光市人民检察院成功应用！

2025年7月国产数据库中标情况一览：长沙银行千万采购GoldenDB，秦皇岛银行近七百万采购TDSQL！

重磅发布：Oracle ADG 一键自动化搭建脚本

重磅 | 万里数据库GreatDB亮相上合组织数字经济论坛 以硬核科技共绘“数字丝路”新图景

中国信通院2025上半年“可信数据库”新增标准解读