Создание роли и привелегий серверного уровня

**Объекты роли**

В PostgreSQL у роли (ролью) является группа прав и привилегий, которые определяют, что пользователь или роль имеет право делать в базе данных. Объекты, к которым могут быть применены различные права доступа для роли, включают в себя следующие:

1. Таблицы: Роли могут иметь права на чтение, запись, обновление и удаление данных из таблиц.

2. Представления: Роли могут иметь право просматривать данные из представлений.

3. Индексы: Роли могут иметь право создавать, удалять или использовать индексы для оптимизации запросов.

4. Схемы: Роли могут иметь права доступа к определенным схемам в базе данных.

5. Функции: Роли могут иметь право запускать функции в базе данных.

6. Процедуры: Роли могут иметь права на выполнение хранимых процедур.

7. Триггеры: Роли могут иметь право создавать и управлять триггерами.

8. Расширения: Роли могут иметь доступ к различным расширениям в PostgreSQL.

9. Базы данных: Роли могут иметь права доступа к определенным базам данных.

10. Экспорт/Импорт данных: Роли могут иметь право экспортировать и импортировать данные.

Эти объекты представляют различные аспекты базы данных, к которым роли могут иметь доступ и на которые могут быть наложены различные права доступа. Управление объектами, к которым имеет доступ роль, является важной частью безопасности базы данных и управления пользователями и их привилегиями в PostgreSQL.

В PostgreSQL можно создавать пользователя у которого может быть много ролей, это позволять целую иерархию ролей с их привелегиями.

-- запрос который даёт практически все привелегии серверного уровня.

CREATE ROLE name LOGIN SUPERUSER CREATEDB CREATEROLE REPLICATION;

привилегия SUPERUSER в PostgreSQL предоставляет пользователю или роли все привилегии в системе. Это самая высокая привилегия, доступная в PostgreSQL, и она позволяет пользователю выполнять любые действия, включая:

• Создание и удаление баз данных

• Создание и удаление пользователей

• Назначение привилегий другим пользователям

• Изменение системных параметров

• Выполнение любых команд SQL

Пользователи с привилегией SUPERUSER по сути имеют неограниченный доступ ко всем объектам и данным в системе PostgreSQL.

Следует соблюдать осторожность при назначении привилегии SUPERUSER, поскольку она может представлять значительный риск для безопасности. Эта привилегия должна назначаться только авторизованному персоналу и использоваться в соответствии с принципом наименьших привилегий, чтобы ограничить доступ только к необходимым функциям.

-- Создать роль

CREATE ROLE role\_name;

-- Грант привилегий на все таблицы в схеме

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO role\_name;

-- Грант привилегий на все последовательности в схеме

GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO role\_name;

-- Грант привилегий на все функции в схеме

GRANT ALL PRIVILEGES ON ALL FUNCTIONS IN SCHEMA public TO role\_name;

-- Грант привилегий на все схемы

GRANT USAGE ON ALL SCHEMAS TO role\_name;

-- Грант привилегий на создание и удаление объектов в базе данных

GRANT CREATE, DROP ON DATABASE db\_name TO role\_name;

-- Грант привилегий на создание и выполнение процедур

GRANT CREATE, EXECUTE ON PROCEDURE proc\_name TO role\_name;

-- Грант привилегий на создание и изменение пользователей

GRANT CREATE, ALTER ON USER u\_name TO role\_name;

-- Грант привилегий серверного уровня

GRANT CREATE DATABASE TO role\_name;

GRANT ALTER TABLESPACE TO role\_name;

GRANT CONNECT TO role\_name;

-- Предоставьте пользователю stanleyig права на использование схемы public — это позволит

-- просматривать объекты в этой схеме

GRANT USAGE ON SCHEMA public TO stanleyig;

-- право на создание таблиц

GRANT CREATE ON SCHEMA public TO stanleyig;

Если просто создать роль CREATE ROLE role\_name;, то по умолчанию все привелегии будут с префиксом NO. Например NO LOGIN.

РОЛЬ postgres это единственная роль созданная по умолчанию с привелегией SUPERUSER

CREATE USER user\_name;

Это тоже самое что и

CREATE ROLE role\_name LOGIN;

Обычно пользователь создаётся с паролем

CREATE USER user\_name WITH PASSWORD ‘password’

CREATE ROLE role\_name создаёт «бесправную» роль (NOLOGIN, NOSUPERUSER и т.д.)

ЛУЧШИЕ ПРАКТИКИ КАСАЮЩИЕСЯ УПРАВЛЕНИЯ БЕЗОПАСНОСТЬЮ

* Не работать с кластером из-под SUPERUSER(postgres). Создавать пользователя с правами(привилегиями) CREATEDB и CREATEROLE и работать из-под него.
* Необходимо создать отдельные роли для отдельных приложений по сценариям использования: readonly, readwrite.
* Давать минимальные права, требующиеся ролями. Это согласуется с принципом, «все запрещено, что не разрешено», а не «всё разрешено, что не запрещено»
* Следовательно, более безопасная стратегия это «все запрещено, что не разрешено». Т.е. сначала надо изъять все права, а потом уже наделять конкретными и нужными.
* Создавать пользователей по акторам приложения: hr\_user, reporting\_user, admin\_stuff.
* Если на пользователя назначено 2 роли и одна запрещает создание БД, а другая разрешает, то разрешение в приоритете.
* При создании БД создаётся схема public и скрытая роль public c доступом на запись в схему public
* Всем новым ролям/пользователям неявно даются привилегии роли public. Таким образом при создании роли/пользователя неявно унаследоваются привилегии роли public на создание, чтобы это исключить надо запустить команду REVOKE CREATE ON SCHEMA public ON public; а так-же REVOKE ALL ON DATABASE db\_name FROM public; -запрещает конектится к указанной базе данных из под роли public
* Осторожно c привилегией CREATEROLE. Если юзер имеет эту привилегию, но не CREATEDB, то он может создать новую роль с правами CREATEDB! А CREATEROLE это почти SUPERUSER потому что юзер с такой привилегией может создать роль с правами CREATEDB и уже обладать правами CREATEROLE И CREATEDB, хотя он по сути ещё не SUPERUSER, НО УЖЕ ОЧЕНЬ МОГУЩЕСТВЕННЫЙ! ☺
* Прежде чем запустить эти команды (REVOKE), для начало надо явно выдать права на конект к базе, GRANT CONNECT ON DATABASE db\_name TO user;. Т.е. сначала дать нужные права явно, а потом изъять из public.

Доступ к таблице

**GRANT** *operation* **ON TABLE** *table\_name* **TO** *role*

***Operation:***

**1. SELECT**

**2. INSERT**

**3. UPDATE**

**4. DELETE**

**5. TRUNCATE**

**6. REFERENCES – создание внешних ключей**

**7. TRIGGER**

**Допустим надо предоставить доступ ко всем таблицам в определённой схеме**

**GRANT** operation **ON ALL TABLES IN SCHEMA** schema\_name **TO** role

Операции можно перечислять через запятую, если надо дать привилегии на несколько операций.

**Доступ к колонкам**

**GRANT** opration(columns) **ON TABLE** table\_name **TO** role

***Operation:***

**1. SELECT**

**2. INSERT**

**3. UPDATE**

**4. REFERENCES**

**Доступ к строкам**

**ALTER TABLE** table **ENABLE ROW LEVEL SECURITY;**

После включения role-level security доступ к данным запрещён.

Необходимо дать доступ явно через политику:

**CREATE POLICY** policy\_name **ON** table

**FOR** operation **TO** role

**USING** (expression)

Две и более политики работают как запросы **UNION** между ними. Т.е. в результирующий набор будут включаться все данные из этих политик, даже если они по смыслу друг другу противоречат.

Удалить политику:

**DROP POLICY** policy\_name **ON** table\_name

**Удаление роли**

Чтобы удалить роль, нужно перед этим изъять все права и переназначить все объекты на другую роль которая владела удаляемая роль.

**ALTER TABLE** table\_name **OWNER TO** other\_role

Незначимые объекты, которыми владела удаляемая роль можно удалить с помощью команды:

**DROP OWNED BY** deleting\_role [**CASCADE**][ **RESTRICT**], restrict is default

Снять все привилегии с таблиц у удаляемой роли:

**REVOKE ALL PRIVILEGES ON table1, table2.. FROM deleteting\_role**

Снять доступ к базе данных для удаляемой роли:

**REVOKE ALL ON DATABASE** db\_name **FROM** deleting\_role

Снять со схемы:

**REVOKE ALL ON SCHEMA** schema\_name **FROM** deleting\_role

Таким образом удаляем все зависимые объекты и изымаем все права.

И только после этого удаляем роль:

**DROP ROLE[IF EXISTS]** deleting\_role