Безопасность в Postgresql

Грачев Д.Г.

Привет! Я Апачай и сегодня Апачай будет вас учить защищаться!

Апачай потратил много времени в раздумьях, как вас научить так, чтобы не травмировать. Пробовал и так и этак. Надеюсь вы оцените старания Апачая.

Правда у Апачая ничего не получилось, как ни старался, камень которому он это показывал - разваливался. Надеюсь вы не обидитесь на Апачая если вдруг ваша нервная система не выдержит.

Что ж, давайте не думать о плохом
и начнем нашу тренировку!

Немного теории

Postgres Pro использует концепцию ролей (*roles*) для управления разрешениями на доступ к базе данных. Роль можно рассматривать как пользователя базы данных или как группу пользователей, в зависимости от того, как роль настроена. Роли могут владеть объектами базы данных (например, таблицами и функциями) и выдавать другим ролям разрешения на доступ к этим объектам, управляя тем, кто имеет доступ и к каким объектам. Кроме того, можно предоставить одной роли *членство* в другой роли, таким образом одна роль может использовать права других ролей.

Концепция ролей включает в себя концепцию пользователей («users») и групп («groups»). До версии 8.1 в PostgreSQL пользователи и группы были отдельными сущностями, но теперь есть только роли. Любая роль может использоваться в качестве пользователя, группы, и того и другого.

Немного теории

Другими словами: Роли могут объединять пользователей в группы, хотя это понятие с версии 8.1 более не используется, при этом права доступа **НАСЛЕДУЮТСЯ.**

Вообще на текущий момент нет разделения на пользователей/группы/роли. Есть только роли, но для простоты и для поддержки есть синонимы.

```
CREATE USER student WITH PASSWORD 'student123';

CREATE ROLE student WITH LOGIN PASSWORD 'student123';


CREATE GROUP editors;
CREATE ROLE editors WITH NOLOGIN;
```

Немного теории

Роль может быть:

- LOGIN — может входить в систему (пользователь)
- NOLOGIN — не может входить (группа/роль)
- SUPERUSER — всё может (администратор)

```
-- Создание пользователя (роль с LOGIN)
CREATE ROLE accountUser WITH LOGIN PASSWORD
'12345678';

-- Создание группы (роль без LOGIN)
CREATE ROLE user WITH NOLOGIN;

-- Создание суперпользователя
CREATE ROLE AccountSuperadmin WITH SUPERUSER
LOGIN PASSWORD '12345678';
```

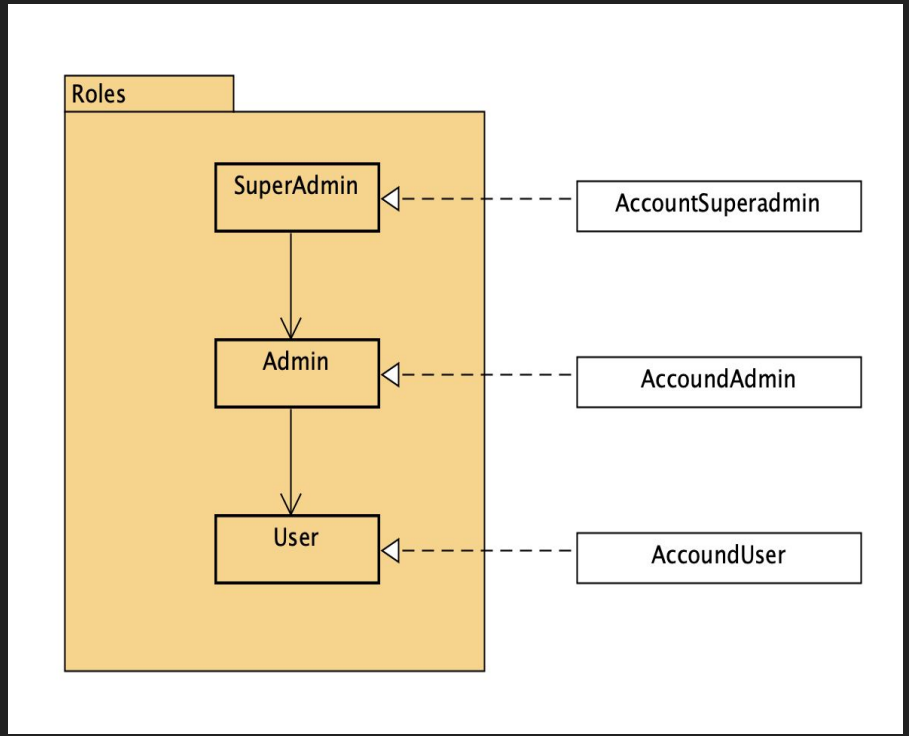
👥 Пользователи vs Группы

Пользователь (LOGIN)	Группа (NOLOGIN)
Может подключаться к БД	Не может подключаться
Имеет личные объекты (таблицы)	Используется для управления правами
Принадлежит группам	Объединяет пользователей
`CREATE USER`	`CREATE ROLE` или `CREATE GROUP`

Базовый синтаксис

```
CREATE ROLE role_name WITH
    LOGIN | NOLOGIN                -- может ли входить в систему
    SUPERUSER | NOSUPERUSER         -- суперпользователь
    CREATEDB | NOCREATEDB           -- может создавать БД
    CREATEROLE | NOCREATEROLE       -- может создавать роли
    INHERIT | NOINHERIT             -- наследует ли права групп
    REPLICATION                     -- для репликации
    PASSWORD 'string'               -- пароль
    VALID UNTIL 'timestamp'         -- срок действия
      CONNECTION LIMIT n;             -- лимит подключений
```

Иерархия наследования ролей (Неправильно)

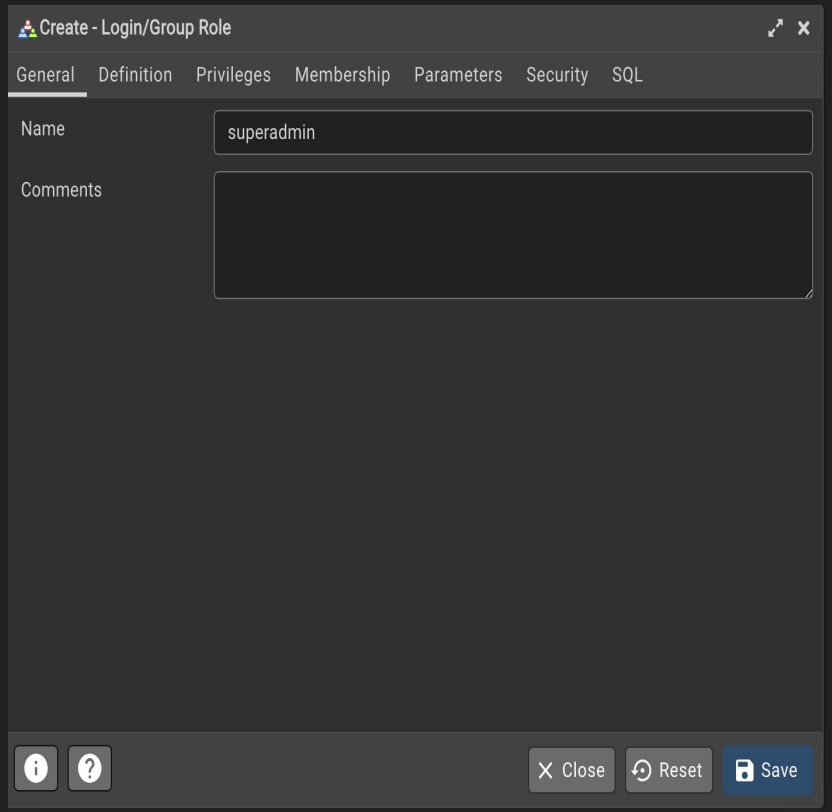


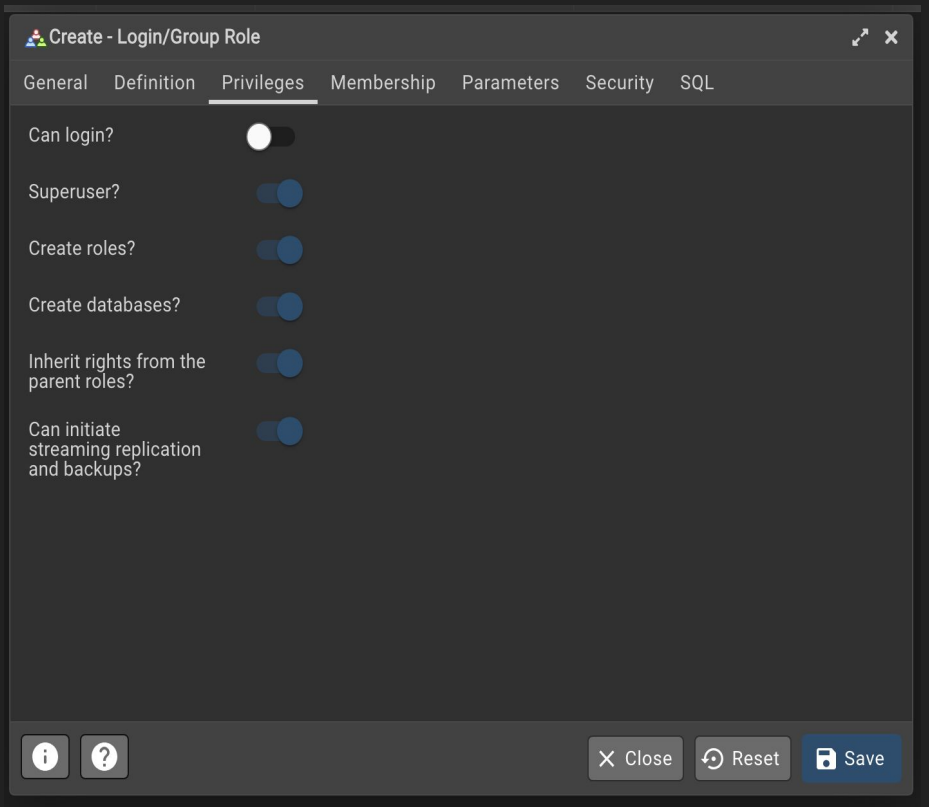
Предположим вам пришла задача создать такую зависимость ролей и учетный записей пользователей. Если “взять” в работу такую схему, то есть два пути:

1) Поправить согласно фактической реализации и согласовать с аналитиком, мол у себя поправь, правда сроки.

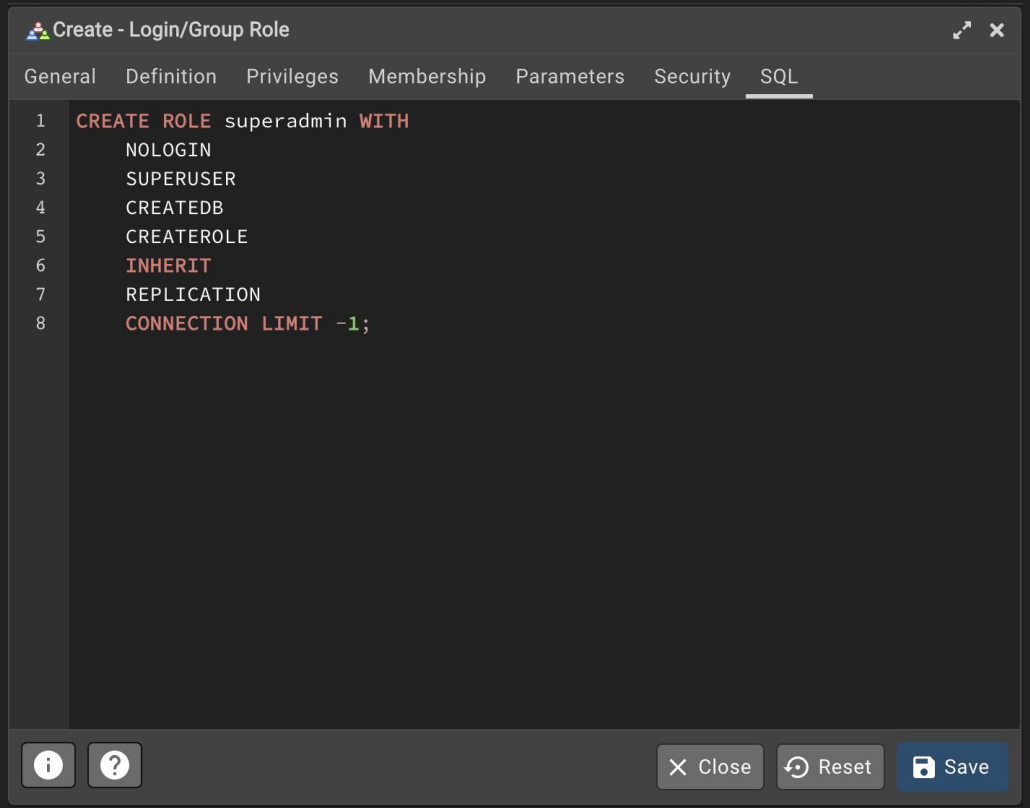
2) Делать как написано, тогда - все пропало.

Иерархия наследования ролей (Неправильно)



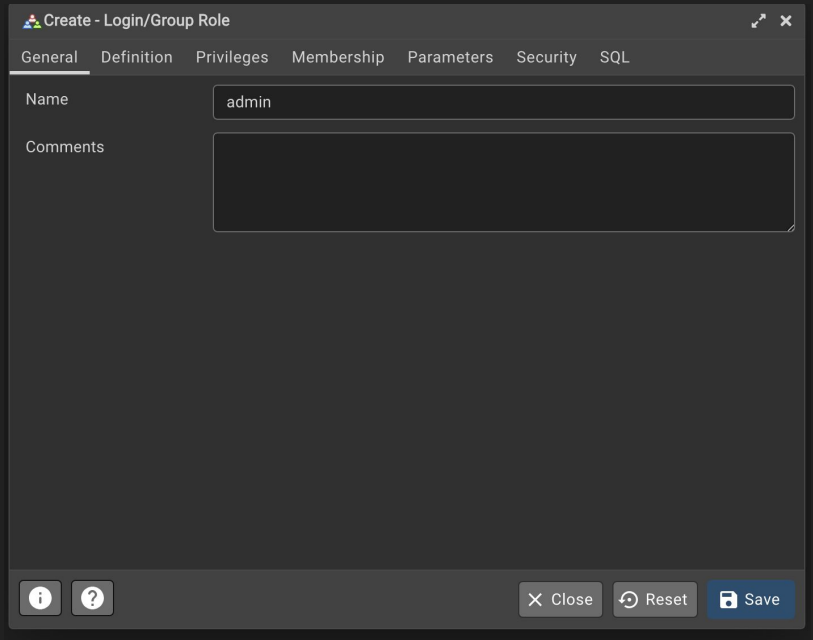


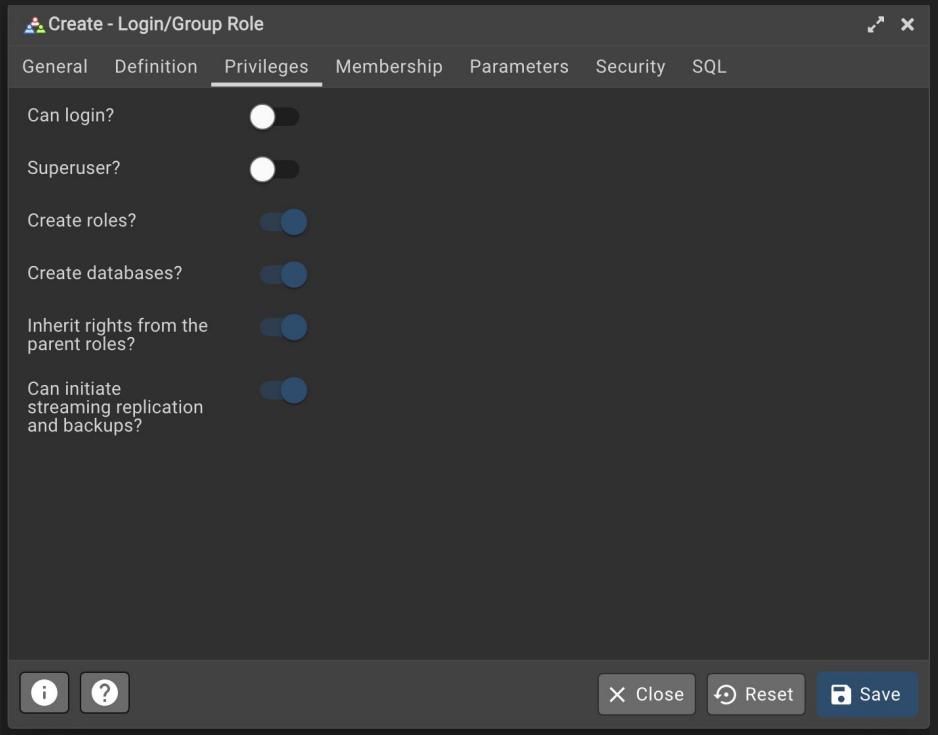
Иерархия наследования ролей (Неправильно)



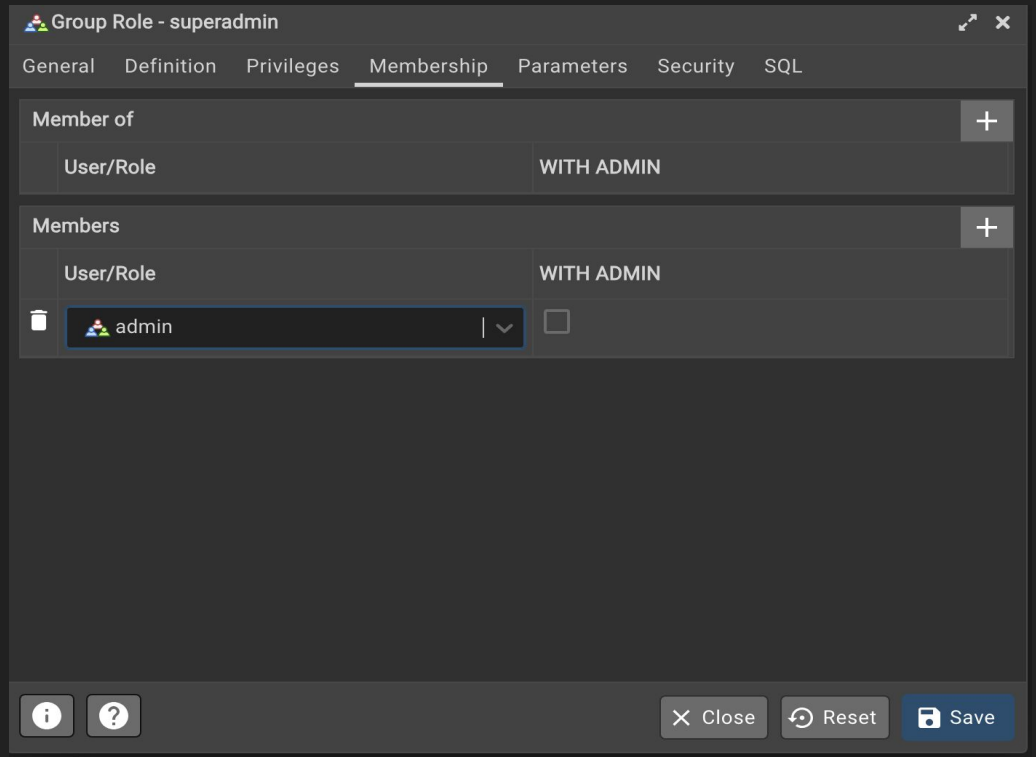
Обратите внимание на кол-во подключений. Лучше выставлять. Также для текущей роли выдали максимальные права, что в рамках суперпользователя нормально, но лучше все же разграничивать и не сваливать все в одну корзину.

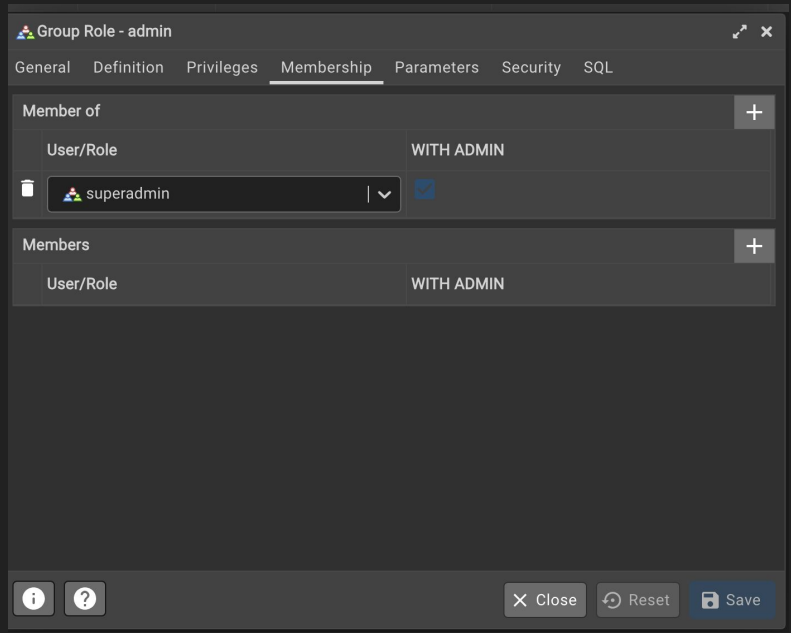
Иерархия наследования ролей (Неправильно)



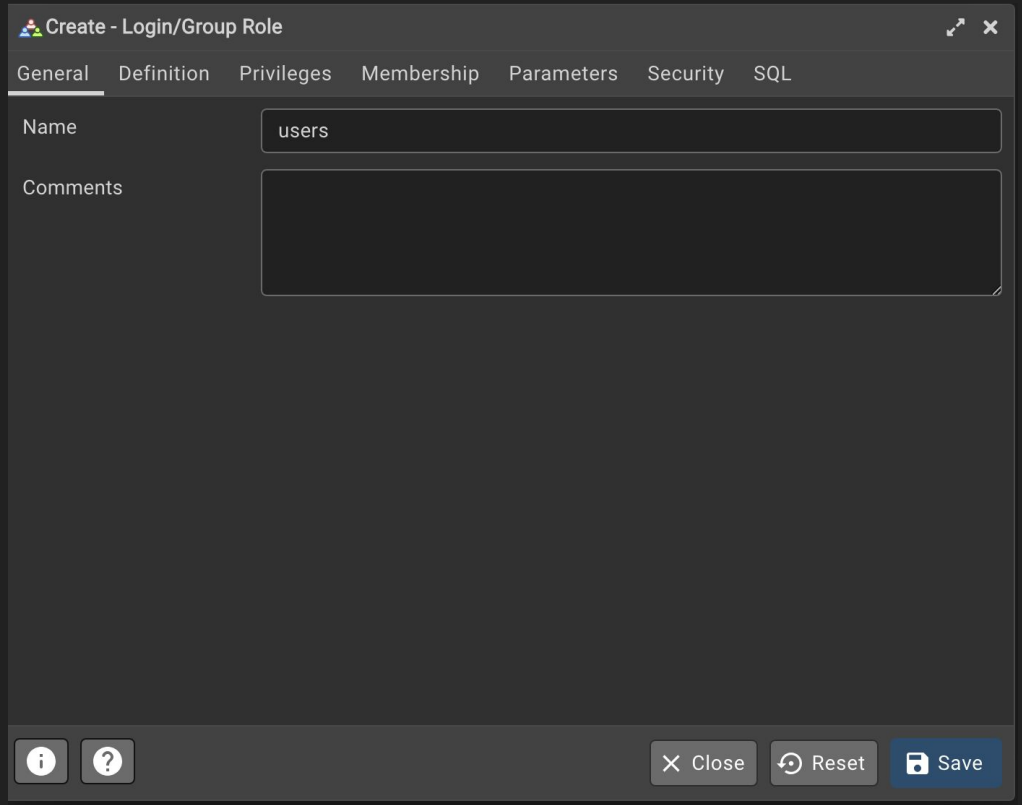


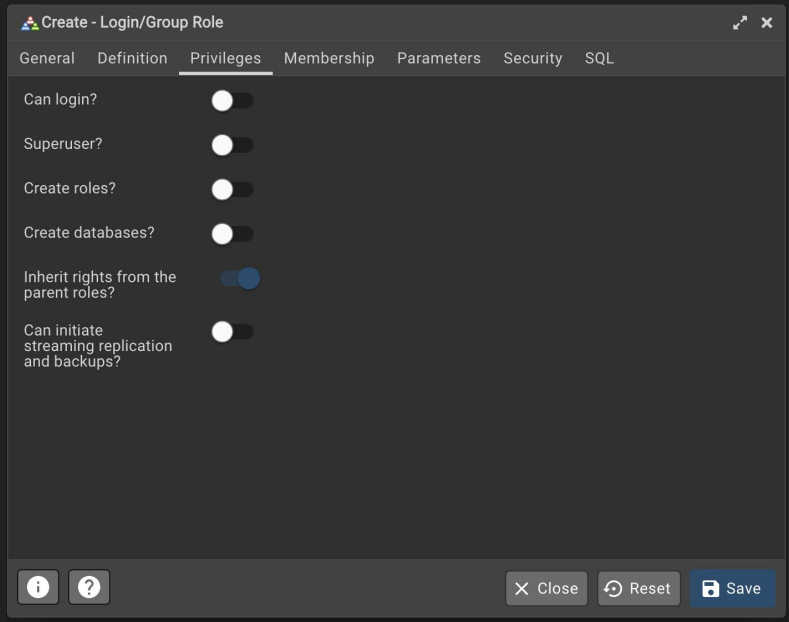
Иерархия наследования ролей (Неправильно)



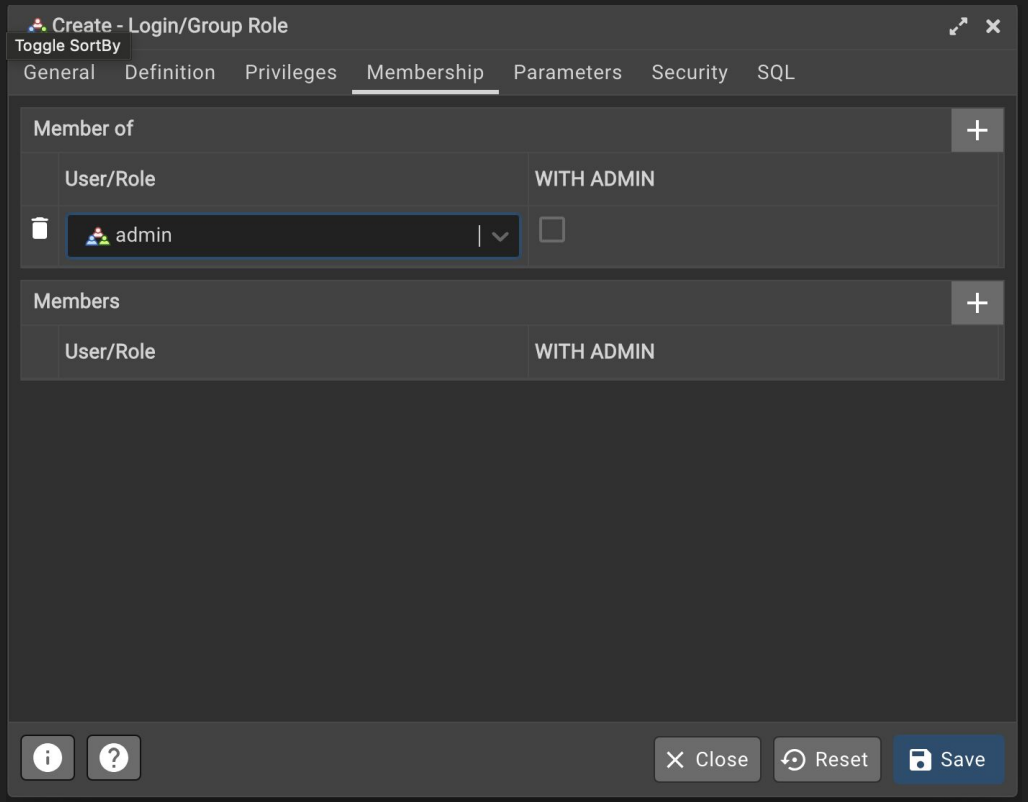


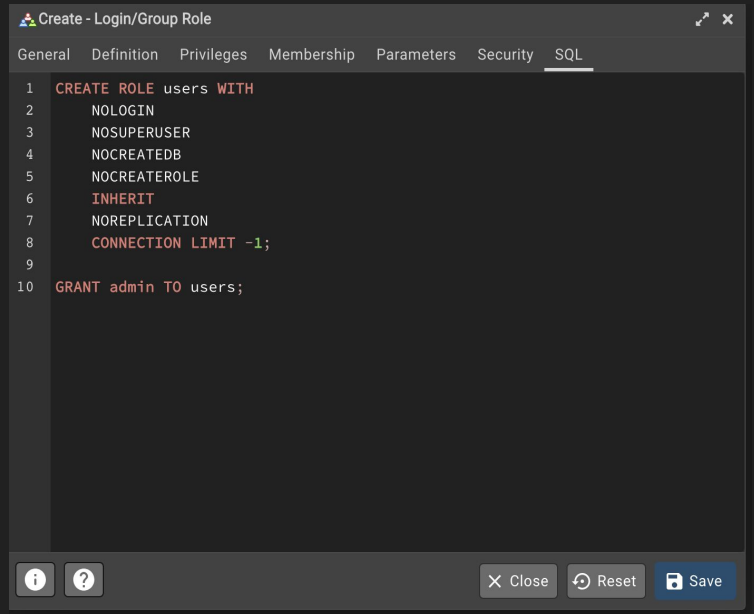
Иерархия наследования ролей (Неправильно)



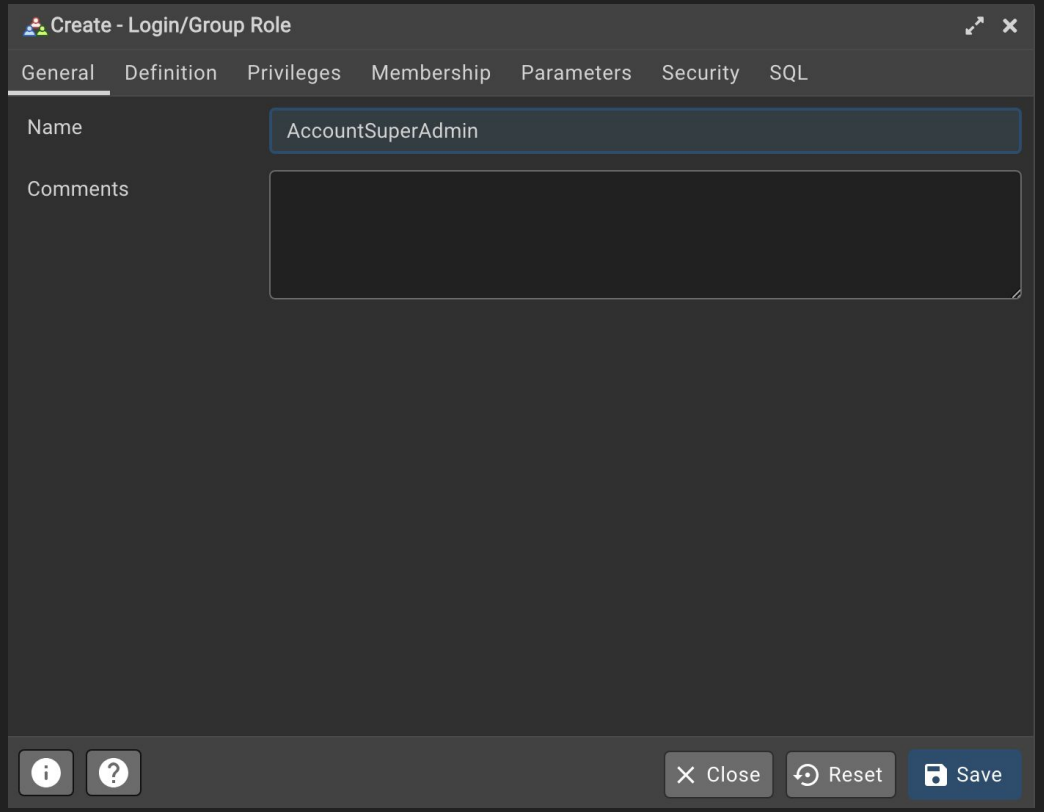


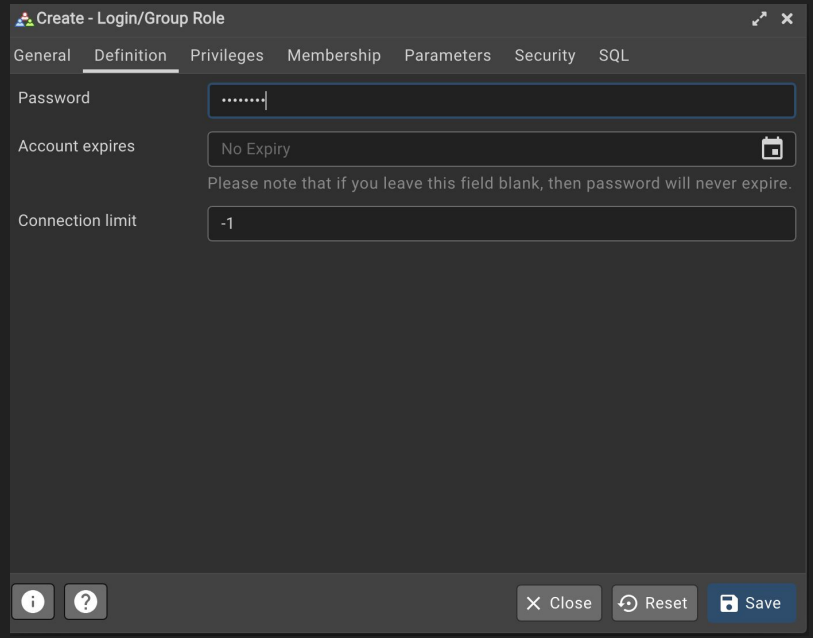
Иерархия наследования ролей (Неправильно)



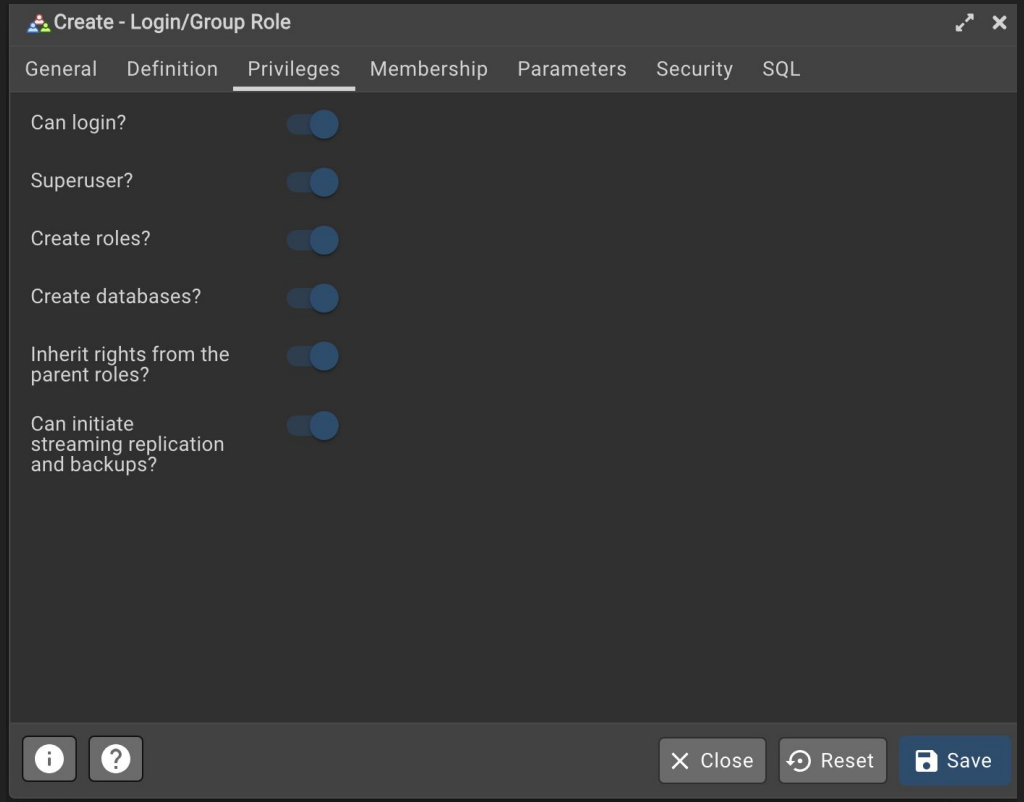


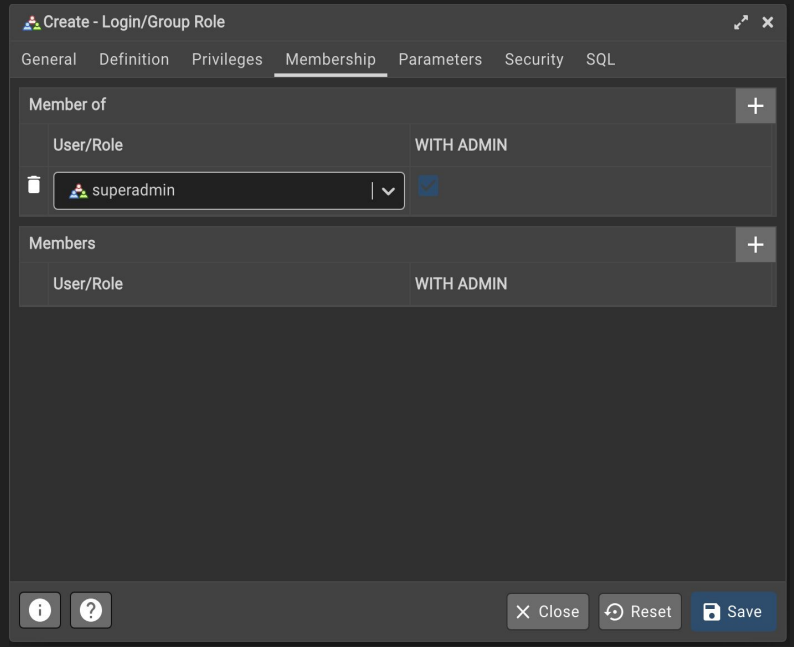
Создаем пользователей (AccountSuperAdmin)



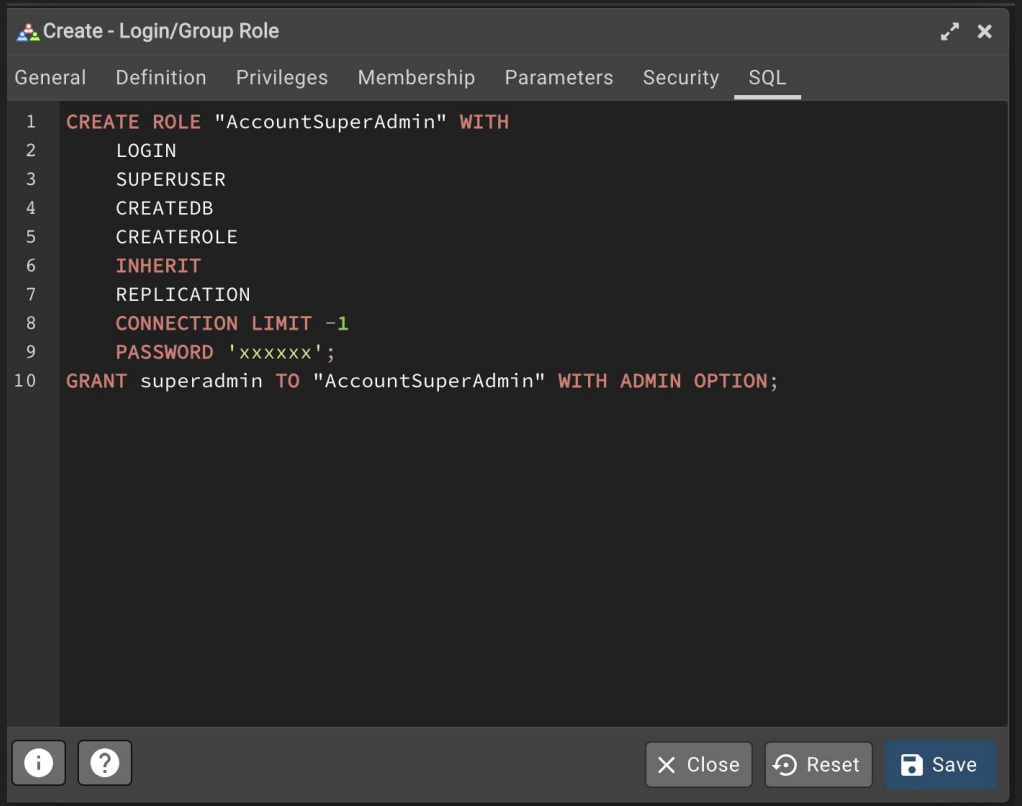


Создаем пользователей (AccountSuperAdmin)

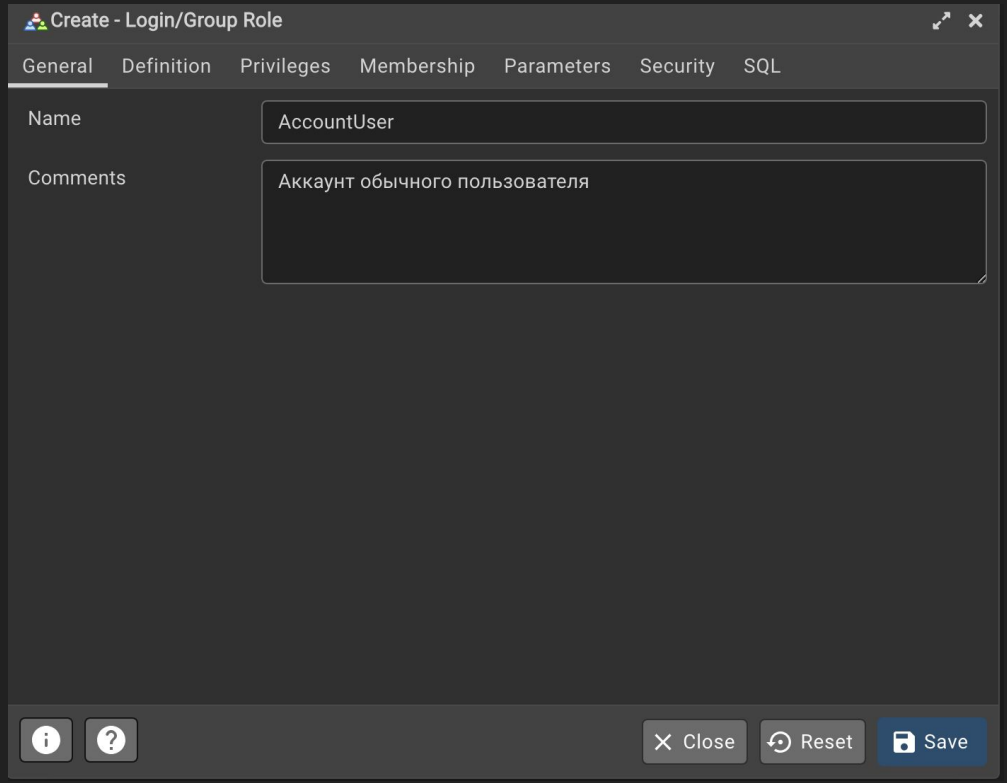


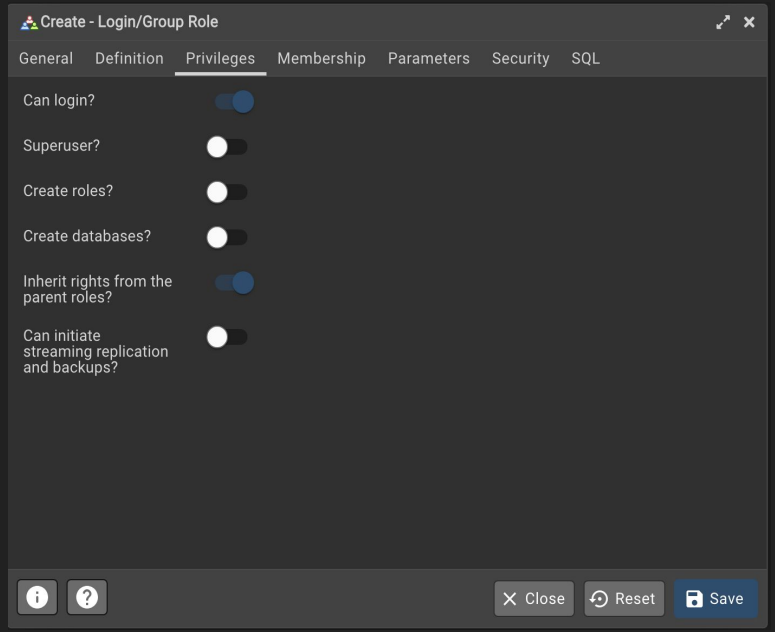


Создаем пользователей (AccountSuperAdmin)

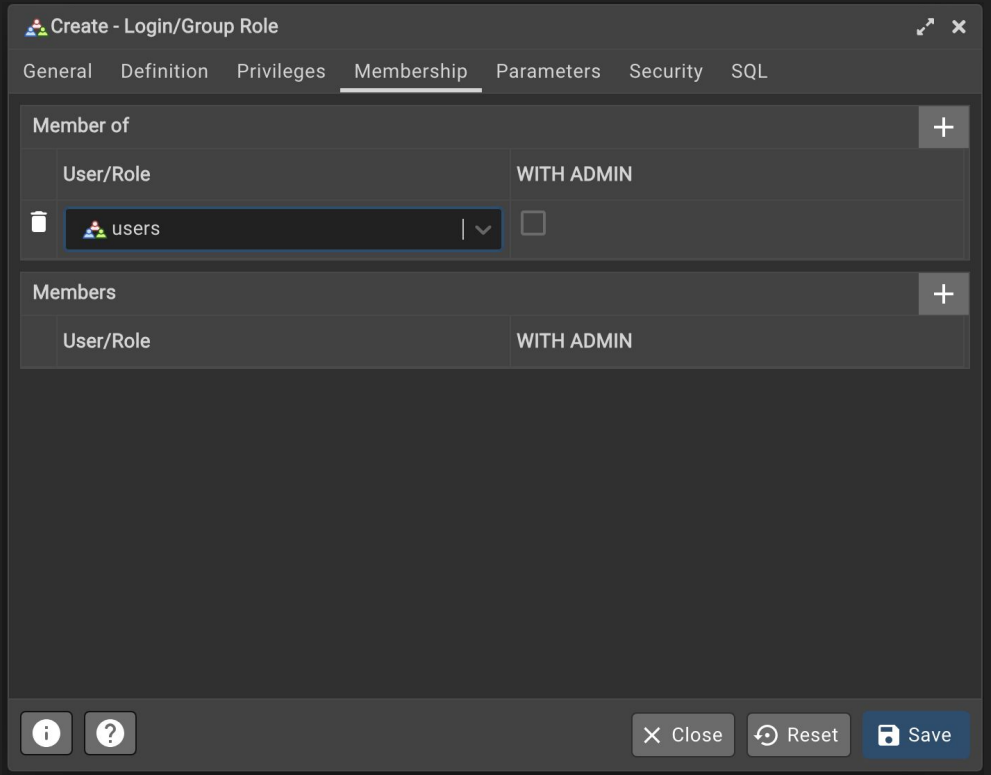


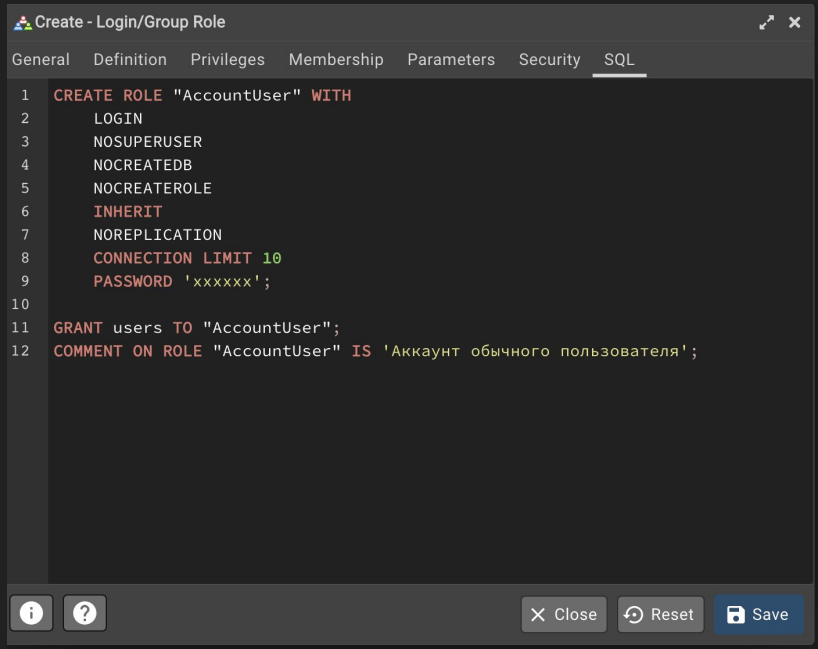
Создаем пользователей (AccountUser)





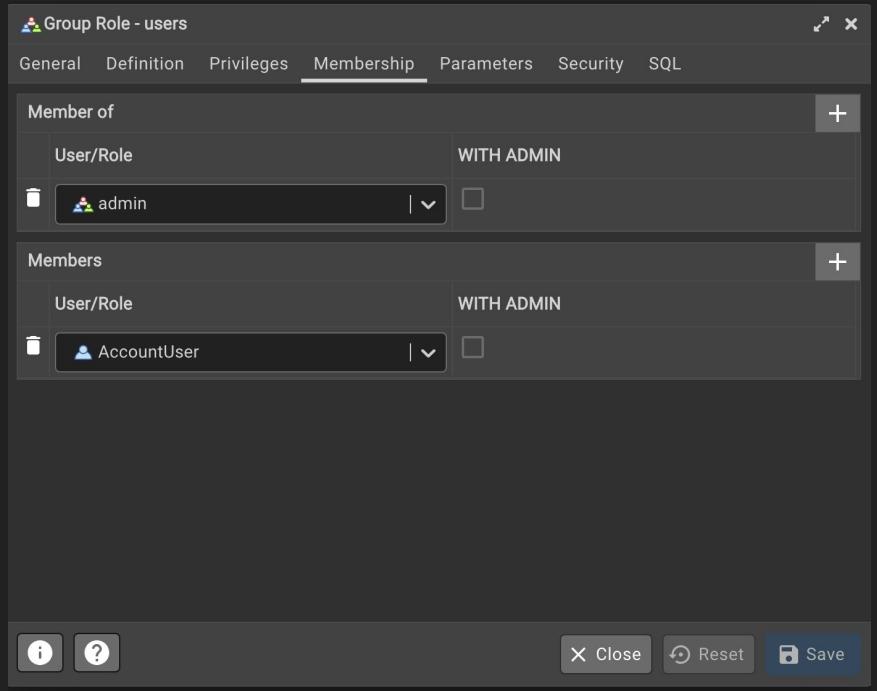
Создаем пользователей (AccountUser)



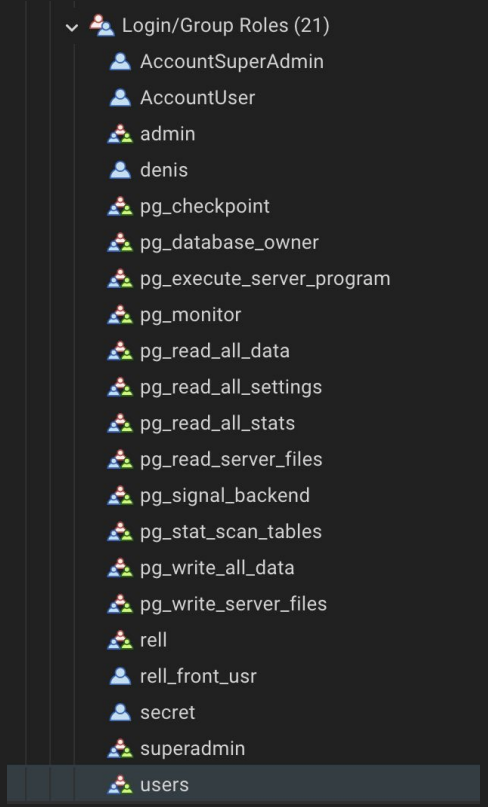


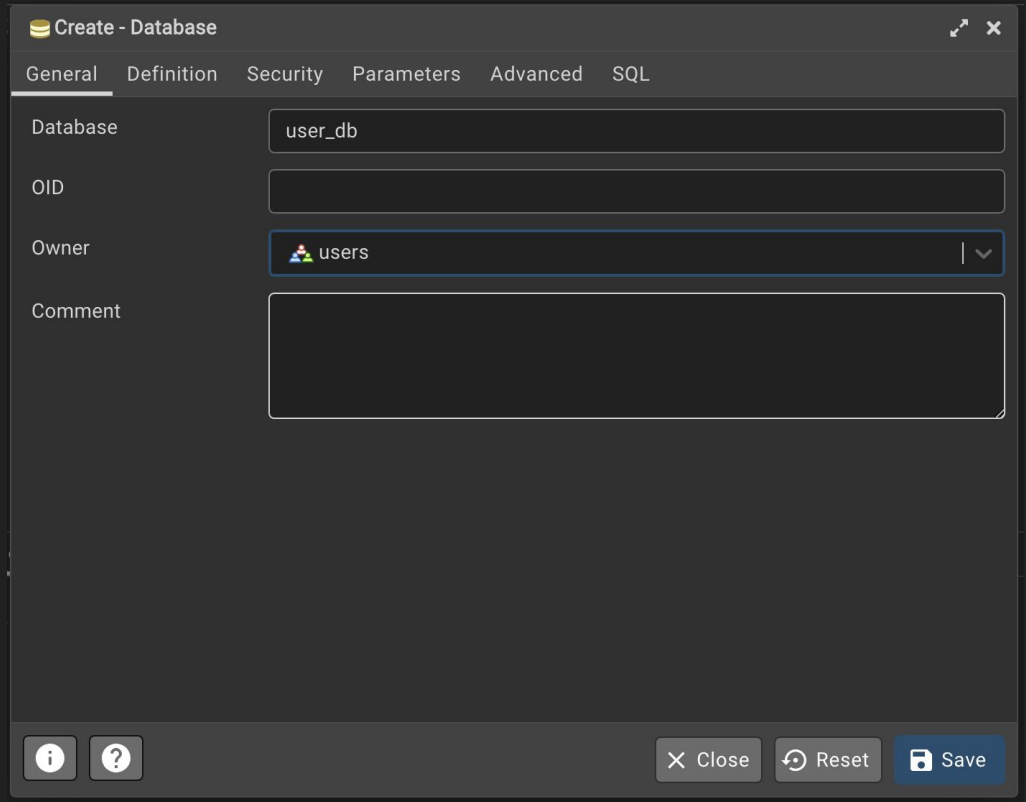
Просмотр текущих зависимостей

Теперь если просмотреть связи ролей, к примеру у пользователей можно увидеть следующее:
1) Роль входит в состав роли админов
2) Имеет одного пользователя “AccountUser”

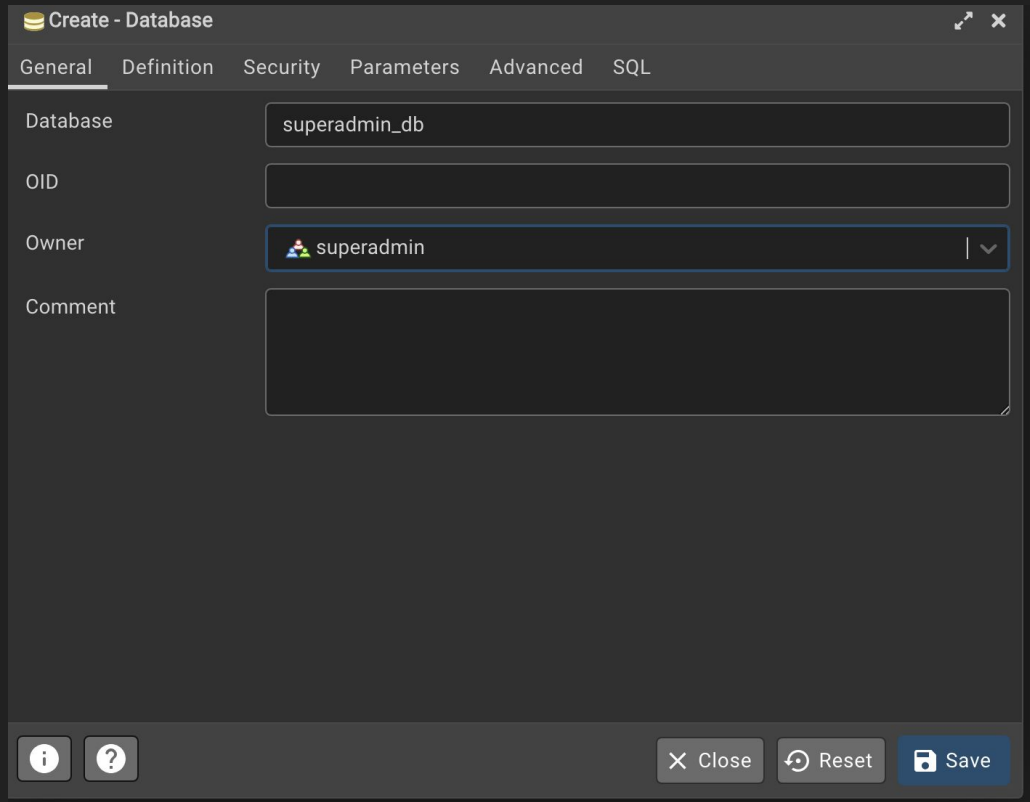


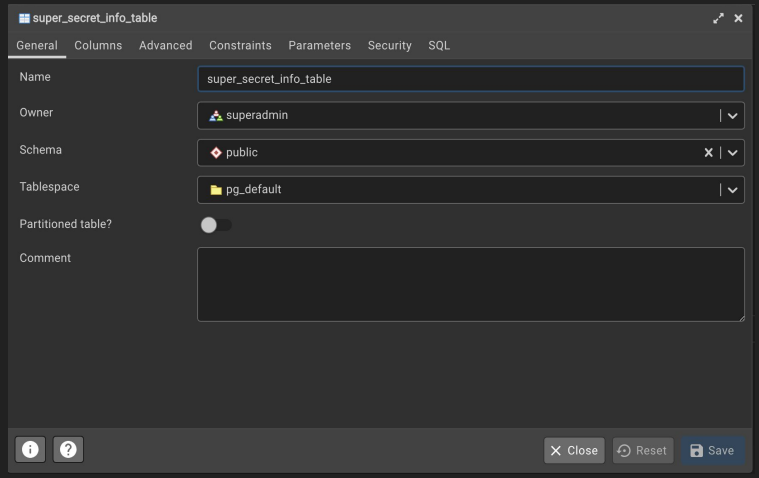
Создаем базу данных для ролей





Создаем базу данных для ролей



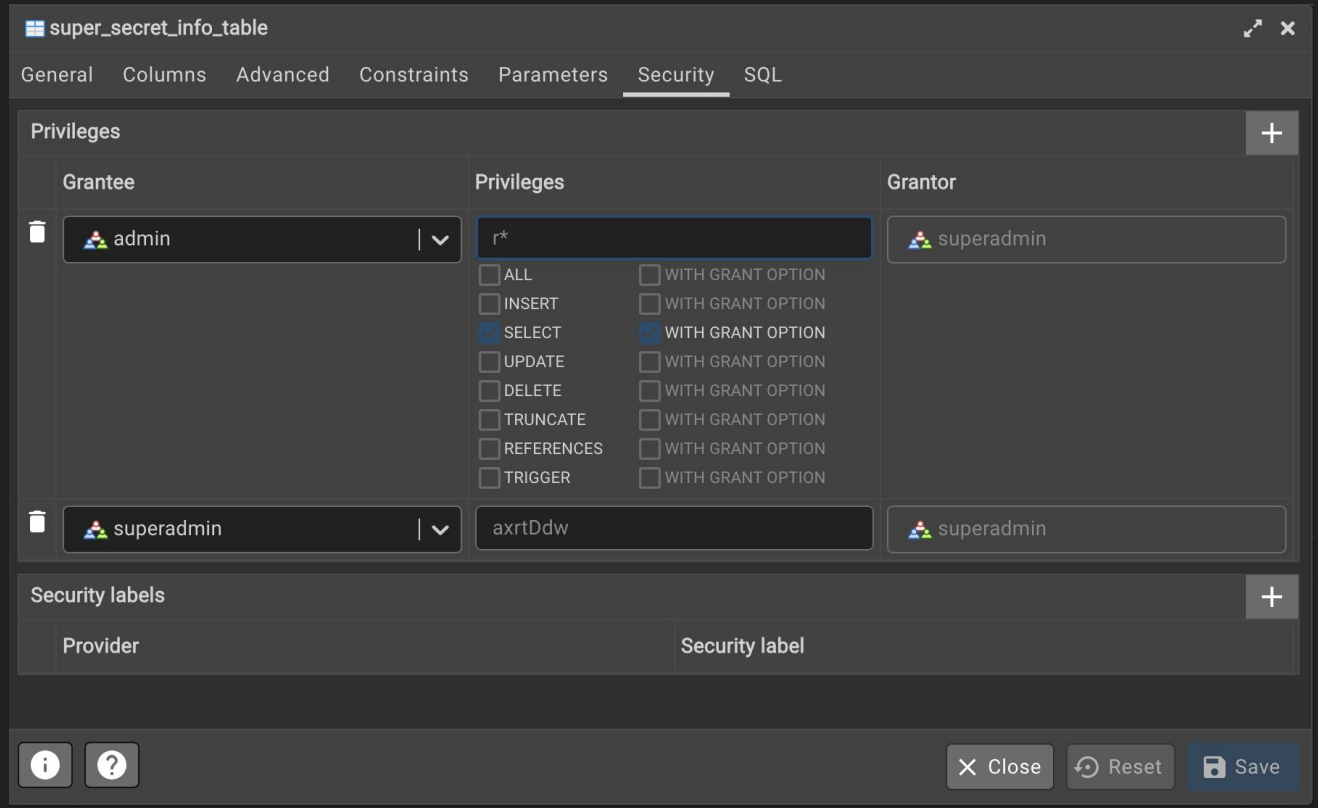


Привилегии (GRANT)

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
    [, ...] | ALL [PRIVILEGES] }
    ON { [TABLE] table_name [, ...] | ALL TABLES IN SCHEMA schema_name }
      TO role_name;
```

- Уровень БД
```
GRANT CONNECT ON DATABASE chirper TO readers;
GRANT CREATE ON DATABASE chirper TO developers;
GRANT TEMP ON DATABASE chirper TO analysts;
```
- Уровень схемы БД
```
GRANT USAGE ON SCHEMA public TO readers;
GRANT CREATE ON SCHEMA public TO writers;
GRANT ALL ON SCHEMA public TO admins;
```
- Уровень таблицы
```
GRANT SELECT ON products TO readers;
GRANT INSERT, UPDATE ON products TO writers;
GRANT ALL ON products TO admins;
```
- Уровень колонки
```
GRANT SELECT (id, name, price) ON products TO sales;
GRANT UPDATE (price) ON products TO managers;
```

Назначение прав доступа другим ролям



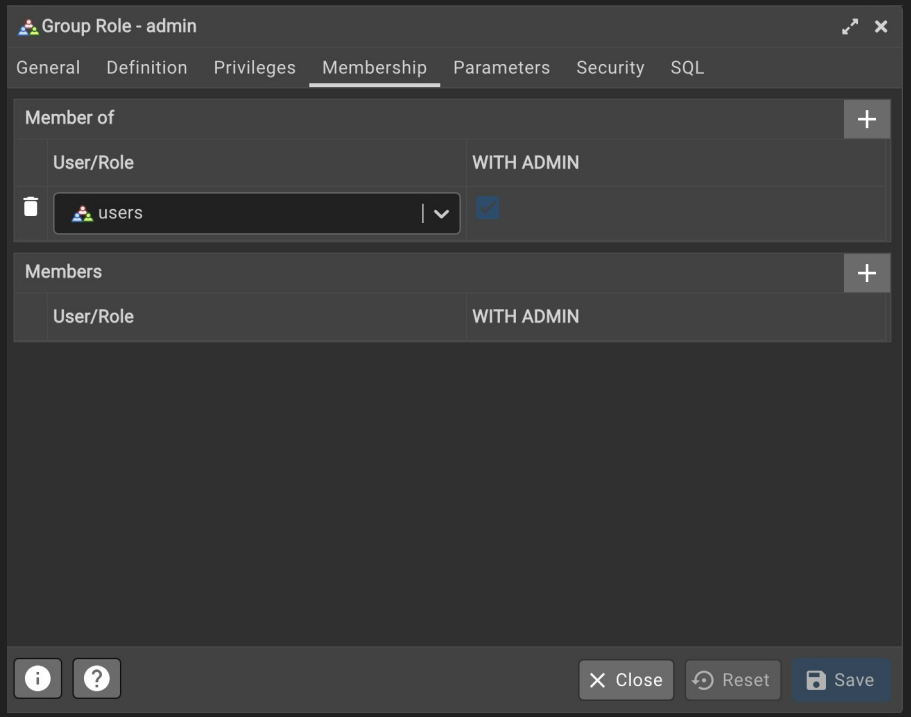
А все ли правильно?

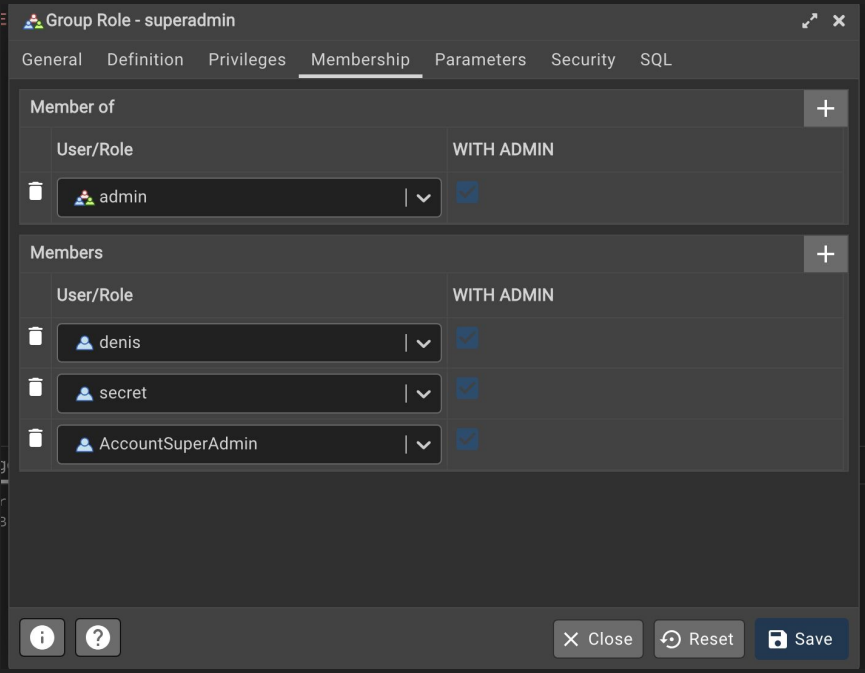
```
Вы подключены к базе данных "public" как пользователь "AccountUser".
public=> \c superadmin_db
Пароль:
Вы подключены к базе данных "superadmin_db" как пользователь "AccountUser".
superadmin_db=> \dt
                  Список отношений
 Схема  |           Имя            |   Тип   |  Владелец
--------+--------------------------+---------+------------
 public | super_secret_info_table | таблица | superadmin
(1 строка)

superadmin_db=> select * from super_secret_info_table;
--
(0 строк)

superadmin_db=>
```

Причина

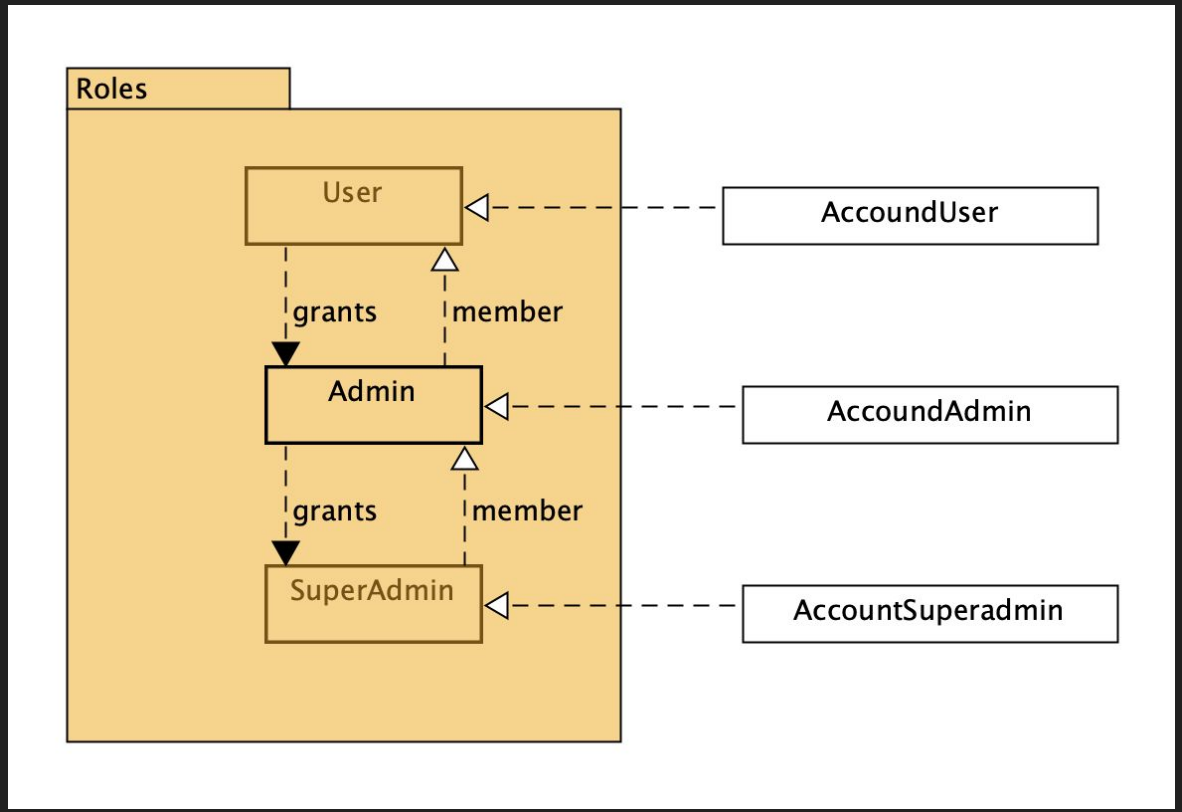
Так как роли наследуют права доступа от родителя, то выходит что users унаследовали права доступа superadmin через admin! Следовательно чтобы решить проблему доступов необходимо развернуть зависимости с точностью наоборот. Те логически расширяем права наследуемых ролей.

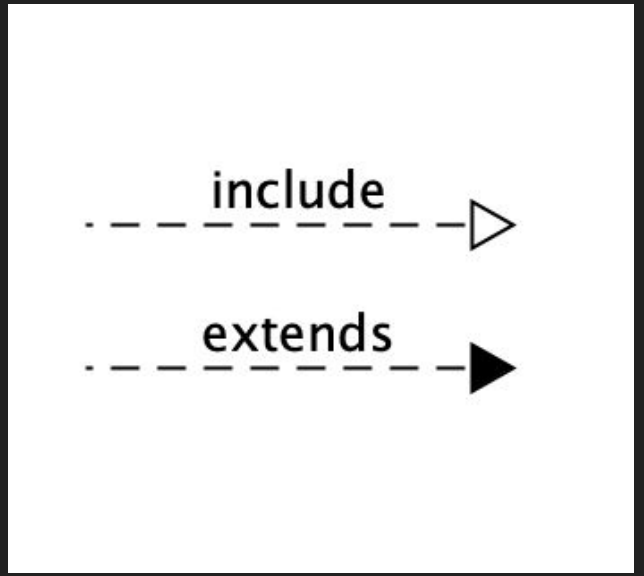
Иерархия наследования ролей (Правильно)





В чем разница?





После

Как только изменили иерархию ролей во всех сессиях сразу подцепились изменения.

```
--
(0 строк)

superadmin_db=> select * from super_secret_info_table;
ERROR:  permission denied for table super_secret_info_table
superadmin_db=>
```

Row Level Security (RLS) - безопасность на уровне строк

```sql
-- Включаем RLS на таблице
ALTER TABLE products ENABLE ROW LEVEL SECURITY;

-- Создаем политику: менеджеры видят всё, остальные только свои продукты
CREATE POLICY products_policy ON products
    USING (                                    -- FOR SELECT
        current_user = 'admin'
        OR current_user IN (SELECT username FROM managers)
        OR created_by = current_user
    );

-- Политика для INSERT
CREATE POLICY products_insert_policy ON products
    FOR INSERT WITH CHECK (
        price > 0  -- Проверка при вставке
    );

-- Политика для UPDATE
CREATE POLICY products_update_policy ON products
    FOR UPDATE USING (
        current_user = created_by  -- только автор может менять
    );
```

А как забрать?

```
-- Забрать конкретные права
REVOKE DELETE ON products FROM sales;

-- Забрать все права
REVOKE ALL ON products FROM sales;

-- Забрать права с каскадным эффектом
REVOKE SELECT ON products FROM sales CASCADE;

-- Забрать право входить в группу
REVOKE sales_dept FROM john;
```

Безопасность на уровне соединения (pg_hba.conf)

```
# TYPE  DATABASE    USER        ADDRESS         METHOD

# Локальные подключения

local   all         all                         peer

local   chirper     ivan,petr                   md5


# Удаленные подключения

host    chirper     readers     192.168.1.0/24  scram-sha-256

host    chirper     admins      10.0.0.0/8      ldap

  hostssl all         all         0.0.0.0/0       md5
```

Поздравляем!

Если вы еще здесь, значит готовы
к чему-то более серьезному!

Многоуровневая система безопасности PostgreSQL

```
└── Сеть/Хост (pg_hba.conf)
    └── Аутентификация (md5, scram-sha-256, peer...)
        └── Роли / Пользователи / Группы
            └── Привилегии (GRANT)
                └── Row Level Security (RLS)
                    └── SSL/TLS шифрование
```

Где мы?)

Немного юмора

Dark Souls	PostgreSQL
Ты умираешь от первого же моба	Ты падаешь от первой же ошибки
Нужно выучить паттерны боссов	Нужно выучить все настройки
Каждая победа - катарсис	Каждый запуск - праздник
"Это сложно, но честно"	"Это сложно, но надежно"
NG+ с новыми врагами	Новые версии с новыми фичами

SSL/TLS в PostgreSQL — базовая защита соединения

SSL (Secure Sockets Layer) и его преемник TLS (Transport Layer Security) — это протоколы шифрования данных между клиентом и сервером

```
— Клиент (приложение) —[Шифрование]—> Сервер PostgreSQL
    (Данные защищены от перехвата)
```

Зачем?

1. Защита от прослушивания (Eavesdropping) — никто в сети не может прочитать передаваемые данные
2. Защита от подмены сервера (MITM) — клиент уверен, что соединяется с настоящим сервером
3. Защита от подделки клиента (Impersonation) — сервер может проверить, кто подключается

Включаем в postgresql.conf

```
# Включаем SSL
ssl = on

# Файлы сертификатов
ssl_cert_file = 'server.crt'      # сертификат сервера
ssl_key_file = 'server.key'        # приватный ключ сервера
ssl_ca_file = 'root.crt'           # корневые сертификаты ЦС (для проверки клиентов)

# Минимальная версия TLS (рекомендуется TLSv1.2+)
  ssl_min_protocol_version = 'TLSv1.2'
```

Настройка клиента (pg_hba.conf)

```
# Требовать SSL для всех подключений

hostssl all all 0.0.0.0/0 md5


# Требовать SSL + клиентский сертификат

  hostssl all all 0.0.0.0/0 md5 clientcert=verify-full
```

Режимы SSL для клиента

Режим	Защита	Описание
`disable`	❌ Нет	Без шифрования
`allow`	⚠️ Минимальная	SSL если сервер требует
`prefer`	⚠️ Средняя	SSL если сервер поддерживает
`require`	✅ Шифрование	Требует шифрование, но не проверяет сертификат
`verify-ca`	✅✅ Проверка ЦС	Шифрование + проверка сертификата сервера
`verify-full`	✅✅✅ Полная	Шифрование + проверка + совпадение имени хоста

Пример подключения

```
psql "sslmode=verify-full host=db.example.com dbname=chirper user=alice"
```

sslinfo — "датчик" SSL-соединения

sslinfo — это расширение PostgreSQL, которое предоставляет функции для получения информации о текущем SSL-соединении и сертификате клиента .

Простыми словами: Это как приборная панель, которая показывает: "Соединение защищено? Каким шифром? Кто подключился по сертификату?"

```
CREATE EXTENSION sslinfo; -- Это расширение требует установки!
```

Быстрая справка

Функция	Что возвращает
ssl_is_used()	Используется ли SSL (true/false)
ssl_version()	Версия протокола (TLSv1.2, TLSv1.3)
ssl_cipher()	Название шифра (AES256-GCM и т.д.)
ssl_client_cert_present()	Есть ли клиентский сертификат
ssl_client_serial()	Серийный номер сертификата
ssl_client_dn()	Distinguished Name клиента
ssl_issuer_dn()	Издатель сертификата
ssl_client_dn_field('CN')	Конкретное поле сертификата (например, Common Name)

Пример использования

```
36
37
38  -- Проверка безопасности текущего соединения
39  SELECT
40      ssl_is_used() AS "SSL активно",
41      ssl_version() AS "Протокол",
42      ssl_cipher() AS "Шифр";
43
44  -- Получение информации о клиенте (если есть сертификат)
45  SELECT
46      ssl_client_dn() AS "Клиент",
47      ssl_client_serial() AS "Серийный номер";
48
49  -- Блокировка небезопасных операций
50  CREATE OR REPLACE FUNCTION secure_payment(amount DECIMAL)
51  RETURNS TEXT AS $$
52  BEGIN
53      IF NOT ssl_is_used() THEN
54          RETURN '❌ Платеж отклонен: нет SSL!';
55      END IF;
56
57      IF ssl_version() < 'TLSv1.2' THEN
58          RETURN '⚠️ Обновите клиент до TLS 1.2+';
59      END IF;
60
61      -- Выполняем платеж
62      RETURN '✅ Платеж проведен';
63  END;
64  $$ LANGUAGE plpgsql;
```

Сложно?

Мы уже близки к PCI DSS!

Но перед этим …

sepgsql — мандатное управление доступом

sepgsql — это расширение PostgreSQL, которое интегрируется с SELinux (Security-Enhanced Linux) для обеспечения мандатного контроля доступа (MAC) на основе меток безопасности .

Простыми словами: Это как система допуска к секретным документам — даже если у вас есть права на чтение таблицы, вы не увидите строки с грифом "Совершенно секретно", если у вас нет соответствующего допуска.

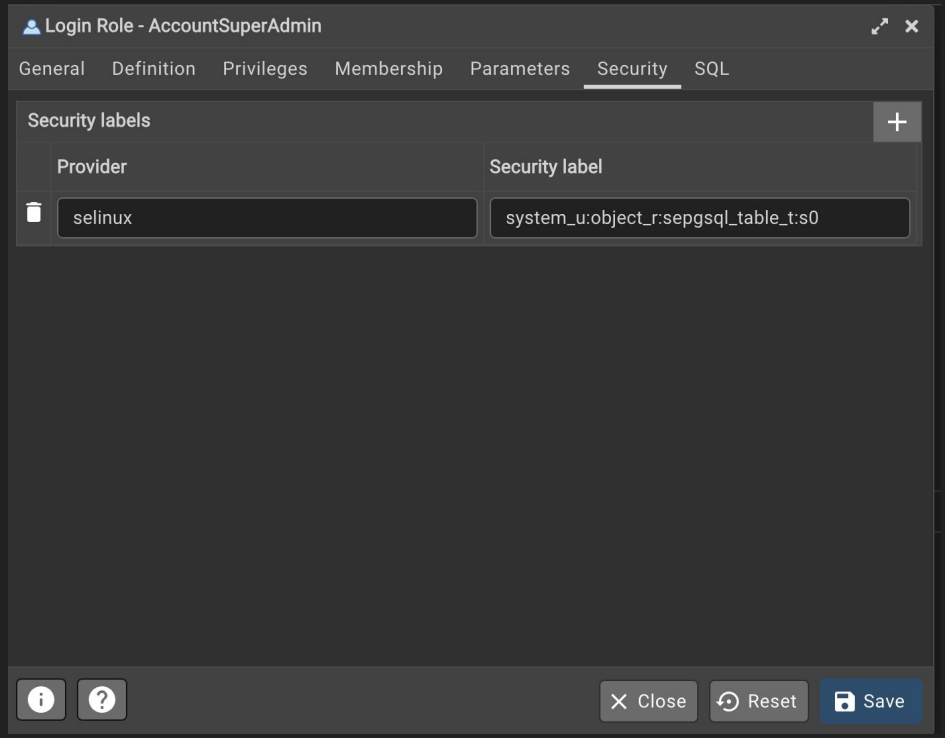
🔧 Как это работает?

1. Метки безопасности — каждому объекту (таблице, колонке, строке) присваивается метка вида `system_u:object_r:sepgsql_table_t:s0`
2. Правила SELinux — политики, которые определяют, кто с какой меткой может что делать
3. Проверка при каждом доступе — sepgsql проверяет метку пользователя и метку объекта
4. Работает только на Linux с включенным SELinux
5. Требует специальной настройки политик SELinux
6. Имеет значительные ограничения (не все действия контролируются)
7. Экспериментальная функциональность

Пример создания

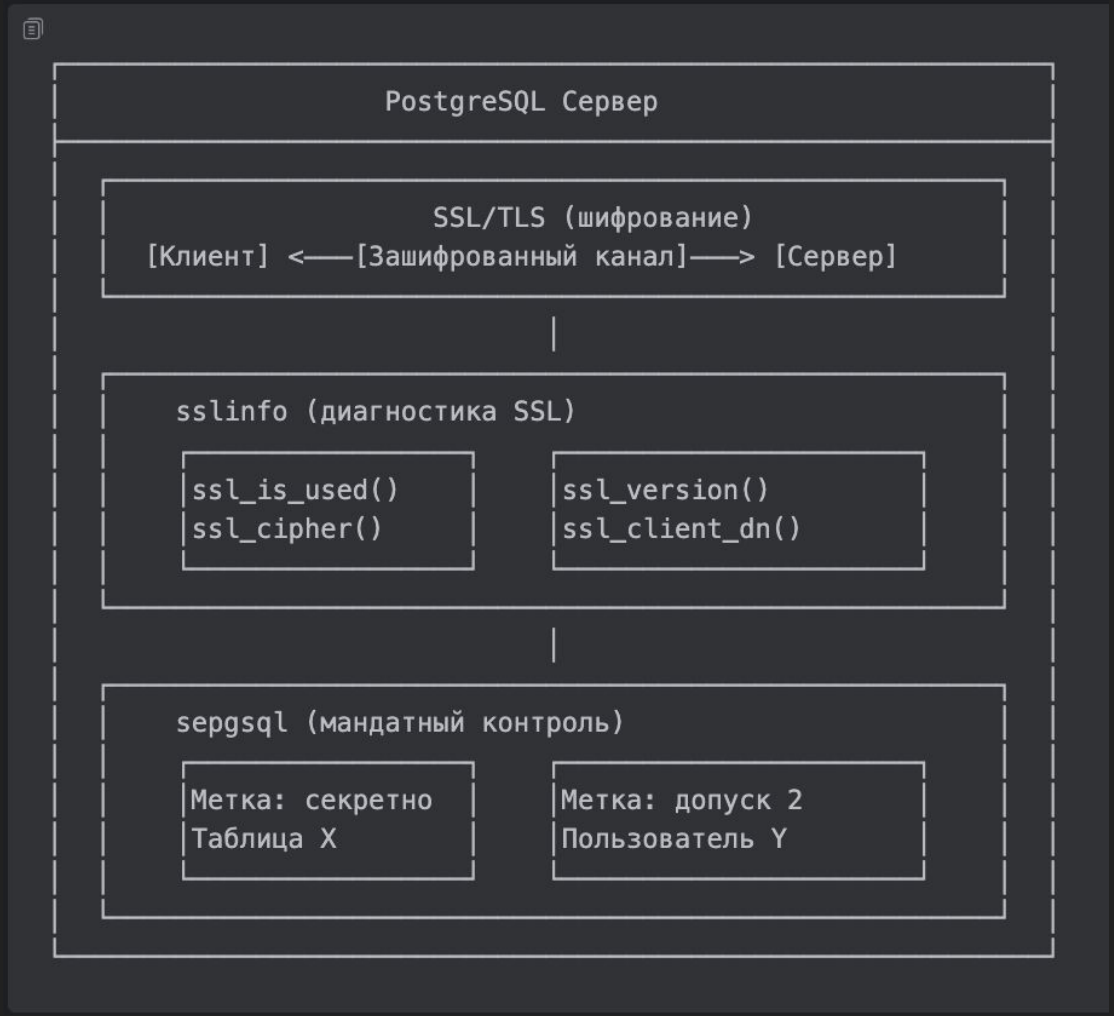
```
-- Назначение метки безопасности таблице
SECURITY LABEL ON TABLE super_secret_info_table IS
'system_u:object_r:sepgsql_secret_table_t:s0';
```

```
-- Назначение метки пользователю
SECURITY LABEL ON ROLE AccountSuperAdmin IS
'system_u:object_r:sepgsql_user_t:s0';
```



```sql
-- 1. Включаем SSL на сервере (postgresql.conf)
-- ssl = on
-- ssl_cert_file = 'server.crt'

-- 2. Устанавливаем sslinfo для мониторинга
CREATE EXTENSION sslinfo;

-- 3. Создаем функцию для проверки безопасности
CREATE OR REPLACE FUNCTION check_security_level()
RETURNS TEXT AS $$
DECLARE
    ssl_status BOOLEAN;
    tls_version TEXT;
    client_cert BOOLEAN;
BEGIN
    -- Проверяем SSL
    ssl_status := ssl_is_used();

    IF NOT ssl_status THEN
        RETURN '⚠️ НЕТ SSL! Данные передаются открыто!';
    END IF;

    -- Проверяем версию TLS
    tls_version := ssl_version();
    client_cert := ssl_client_cert_present();

    RETURN format('✅ SSL активен (%s), клиентский сертификат: %s',
                  tls_version,
                  CASE WHEN client_cert THEN 'есть' ELSE 'нет' END);
END;
$$ LANGUAGE plpgsql;

-- 4. Если нужно сверхсекретно — включаем sepgsql
-- shared_preload_libraries = 'sepgsql'
-- CREATE EXTENSION sepgsql;

-- Назначаем метки
SECURITY LABEL ON COLUMN products.supplier_info
    IS 'system_u:object_r:sepgsql_confidential_t:s0';

-- 5. Проверка
SELECT check_security_level();
```

Аспект	SSL/TLS	sslinfo	sepgsql
Что делает	Шифрует канал связи	Показывает информацию о шифровании	Контролирует доступ по меткам
Уровень	Сеть/транспорт	Соединение	Объекты БД (таблицы, строки)
Защита от	Перехвата, MITM	Нет, только диагностика	Несанкционированного доступа
Требования	OpenSSL	OpenSSL	SELinux, Linux
Когда использовать	Всегда в продакшене	Для аудита и мониторинга	Для сверхсекретных данных
Сложность	Средняя	Низкая	Очень высокая


PostgreSQL Сервер
SSL/TLS (шифрование)
[Клиент] <——[Зашифрованный канал]——> [Сервер]
sslinfo (диагностика SSL)
ssl_is_used()
ssl_cipher()
ssl_version()
ssl_client_dn()
sepgsql (мандатный контроль)
Метка: секретно
Таблица X
Метка: допуск 2
Пользователь Y

Итого

Технология	Аналогия
SSL/TLS	Запечатанный конверт вместо открытки
sslinfo	Проверка: "Конверт действительно запечатан?"
sepgsql	Сейф с грифом "Сов. секретно" внутри здания

Для 99% проектов достаточно SSL/TLS + sslinfo для контроля. sepgsql нужен только если вы работаете с гостайной или в банке с особо строгими требованиями.

Заметки

- В современных системах небольшого масштаба все крутится в кластере контейнеров, таких как k8s которые по умолчанию могут в безопасность
- Если ресурс небольшой, то обычно для базы данных закрывают все внешние подключения кроме localhost
- Если внешние порты закрыть, а нужно срочно подключиться - можно использовать мост через ssh (если он настроен).
- Чем больше безопасности - тем медленнее работает =-)
- Старайтесь любый подключения закрывать сертификатами.
- Помните, у сертификатов имеется срок годности, - будет плохо если интеграция между сервисами упадет из-за просрочки.

Пришло время теперь узнать на что мы способны на самом деле!

Практическое задание

```
CREATE DATABASE shop;
\c shop

CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    price DECIMAL,
    supplier_info TEXT,  -- конфиденциально
    internal_notes TEXT  -- только для сотрудников
);

CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    product_id INT REFERENCES products,
    user_id INT,
    quantity INT,
    created_at TIMESTAMP
);

CREATE TABLE suppliers (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    contact_info TEXT,
    bank_details TEXT  -- очень конфиденциально
);
```

```
-- Группы (роли без логина)
CREATE ROLE customers;      -- клиенты
CREATE ROLE sales;          -- продавцы
CREATE ROLE managers;       -- менеджеры
CREATE ROLE accountants;    -- бухгалтеры
CREATE ROLE admins;         -- администраторы

-- Пользователи
CREATE USER ivan WITH PASSWORD 'ivan123';
CREATE USER petr WITH PASSWORD 'petr123';
CREATE USER anna WITH PASSWORD 'anna123';
CREATE USER admin WITH PASSWORD 'admin123' SUPERUSER;

-- Назначаем роли
GRANT customers TO ivan;
GRANT sales TO petr;
GRANT accountants TO anna;
  GRANT admins TO admin, petr;  -- Петр еще и админ
```

```sql
-- 1. Базовые права для всех
GRANT CONNECT ON DATABASE shop TO customers, sales,
managers, accountants;

-- 2. Права на схему
GRANT USAGE ON SCHEMA public TO customers, sales, managers,
accountants;

-- 3. КЛИЕНТЫ (только видят товары)
GRANT SELECT ON products TO customers;
GRANT SELECT (id, name, price) ON products TO customers;
GRANT SELECT, INSERT ON orders TO customers;  -- могут
создавать заказы

-- 4. ПРОДАВЦЫ (видят больше, могут обновлять)
GRANT SELECT, UPDATE ON products TO sales;
GRANT SELECT, UPDATE ON orders TO sales;
GRANT SELECT (name, contact_info) ON suppliers TO sales;  --
только имя и контакт

-- 5. МЕНЕДЖЕРЫ (почти всё, кроме финансов)
GRANT ALL ON products, orders TO managers;
GRANT SELECT ON suppliers TO managers;

-- 6. БУХГАЛТЕРЫ (финансовая информация)
GRANT SELECT (id, name, price) ON products TO accountants;
GRANT SELECT, UPDATE ON orders TO accountants;
GRANT SELECT (name, bank_details) ON suppliers TO
accountants;  -- банковские реквизиты

-- 7. АДМИНЫ (всё)
  GRANT ALL ON ALL TABLES IN SCHEMA public TO admins;
```

```sql
-- Подключаемся как Иван (клиент)
\c shop ivan
SELECT * FROM products;  -- ✅ OK
SELECT supplier_info FROM products;  -- ❌ ОШИБКА!
(нет прав)
INSERT INTO products VALUES (...)  -- ❌ ОШИБКА!
SELECT * FROM suppliers;  -- ❌ ОШИБКА!

-- Подключаемся как Петр (продавец + админ)
\c shop petr
SELECT * FROM products;  -- ✅ OK
UPDATE products SET price = 100 WHERE id = 1;  -- ✅
OK
  SELECT * FROM suppliers;  -- ✅ OK (админ видит

  всё)
```

Практическое задание

```sql
-- Создаем иерархию ролей
CREATE ROLE employees;           -- все сотрудники
CREATE ROLE sales_dept;           -- отдел продаж
CREATE ROLE accounting_dept;      -- бухгалтерия

-- Наследование: младшие роли входят в старшие
GRANT employees TO sales_dept, accounting_dept;
GRANT sales_dept TO sales_managers, sales_staff;
GRANT accounting_dept TO accountants;

-- Создаем сотрудников с наследованием прав
CREATE USER john WITH PASSWORD 'john123' IN ROLE sales_staff;
CREATE USER jane WITH PASSWORD 'jane123' IN ROLE accountants;

-- Джон автоматически имеет права:
-- sales_staff -> sales_dept -> employees
```

```sql
-- Включаем RLS на таблице
ALTER TABLE products ENABLE ROW LEVEL SECURITY;

-- Создаем политику: менеджеры видят всё, остальные только свои продукты
CREATE POLICY products_policy ON products
    USING (                                    -- FOR SELECT
        current_user = 'admin'
        OR current_user IN (SELECT username FROM managers)
        OR created_by = current_user
    );

-- Политика для INSERT
CREATE POLICY products_insert_policy ON products
    FOR INSERT WITH CHECK (
        price > 0  -- Проверка при вставке
    );

-- Политика для UPDATE
CREATE POLICY products_update_policy ON products
    FOR UPDATE USING (
        current_user = created_by  -- только автор может менять
      );
```

Практическое задание SEPGSQL (по желанию)

- Установить Linux с поддержкой SeLinux (виртулка/хост)
- Установить на нее Postgresql
- В Postgresql создать базу и установить расширение segpsql
- Создать произвольную таблицу и на ее некоторые колонки установить необходимые метки для доступы (security label)
- Создать две роли. Одна с меткой другая без, но обе с доступом к схеме.
- Заполнить таблицу данными
- Из под двух учеток произвести select * … и проверить результат.

Рекомендация

- Изучить как выпускать самому подписанные сертификаты
- Научиться настраивать и устанавливать защищенные соединения между клиентом и базой данных

Ресурсы

- https://postgrespro.ru/docs/postgrespro/18/client-authentication - авторизация пользователей
- https://postgrespro.ru/docs/postgrespro/18/user-manag - роли/пользователи управление
- https://www.postgresql.org/docs/current/sql-security-label.html - security label sepsql/SeLInux/dummy