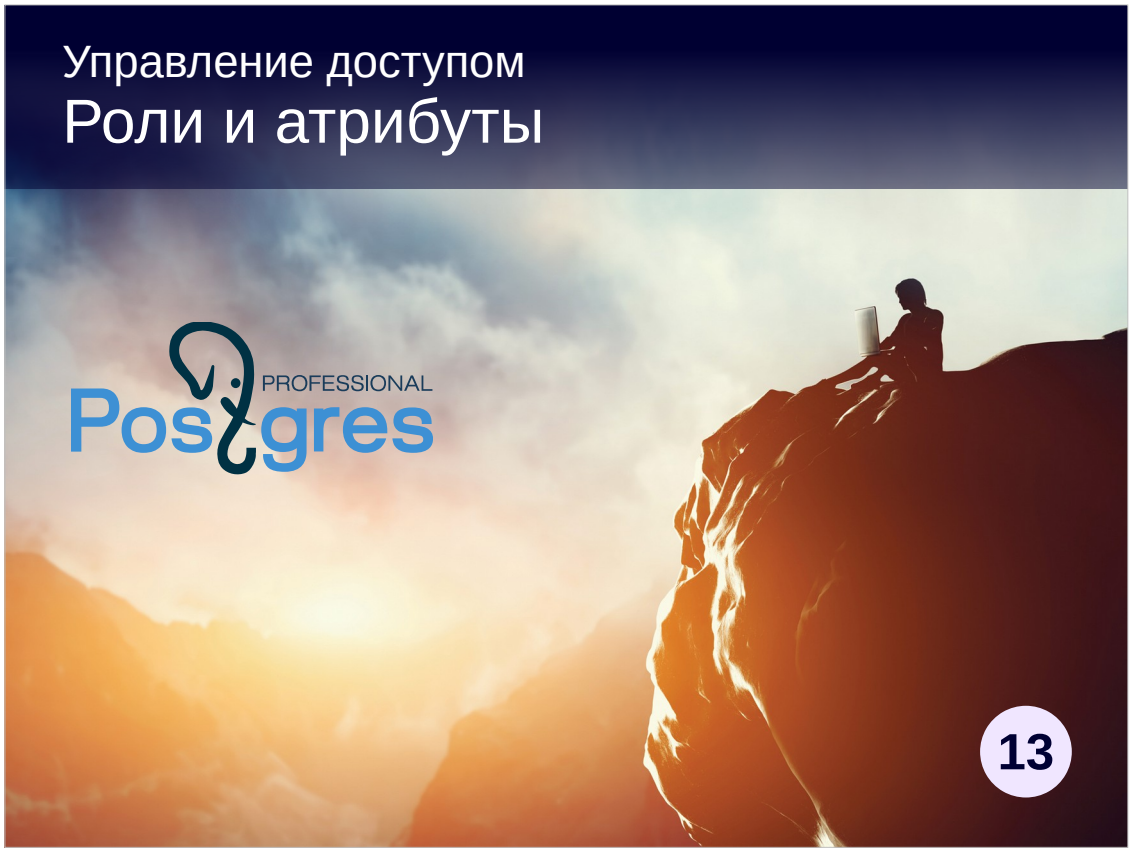


Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

Роли

Атрибуты

Участие в групповых ролях

Владельцы объектов

Роли

Роль — пользователь СУБД

Роль не связана с пользователем ОС

> хотя многие программы используют имя пользователя ОС как имя роли по умолчанию

Роль определяется на уровне кластера

В первом приближении роль — это пользователь СУБД. (Роль также может выступать в качестве группы пользователей, о чем говорится дальше в этой теме.)

Роли никак не связаны с именами пользователей ОС, хотя стандартные программы это предполагают, выбирая значения по умолчанию. Например, если при запуске psql не указать имя роли, на его место будет подставлено имя пользователя ОС.

Роли являются общими объектами кластера. Например, одна роль может подключаться к разным базам данных и быть владельцем объектов в разных БД.

https://postgrespro.ru/docs/postgresql/13/database-roles

Атрибуты

Атрибуты определяют свойства роли

`CREATE ROLE` *`роль`* `[WITH]` *`атрибут`* `[`*`атрибут`* `...]`

`LOGIN`	возможность подключения
`SUPERUSER`	суперпользователь
`CREATEDB`	возможность создавать базы данных
`CREATEROLE`	возможность создавать роли
`REPLICATION`	использование протокола репликации

и другие

Роль обладает некоторыми атрибутами, определяющими ее общие особенности и права (не связанные с правами доступа к объектам).

Обычно атрибуты имеют два варианта, например, CREATEDB (дает право на создание БД) и NOCREATEDB (не дает такого права). Как правило, по умолчанию выбирается ограничивающий вариант.

В таблице перечислены лишь некоторые из атрибутов. Атрибуты INHERIT и BYPASSRLS рассматривается дальше в этом модуле.

https://postgrespro.ru/docs/postgresql/13/role-attributes

https://postgrespro.ru/docs/postgresql/13/sql-createrole

Роли и атрибуты

В этом модуле приглашение будет показывать имя роли, от которой выполняется команда.

```
student=# CREATE DATABASE access_roles;
```

```
CREATE DATABASE
```

```
student=# \c access_roles
```

```
You are now connected to database "access_roles" as user "student".
```

Создадим роль для Алисы:

```
student=# CREATE ROLE alice LOGIN CREATEROLE;
```

```
CREATE ROLE
```

Алиса имеет возможность подключения (атрибут LOGIN) и создания других ролей (CREATEROLE).

Проверим это:

```
student=# \c - alice
```

```
You are now connected to database "access_roles" as user "alice".
```

```
alice=> CREATE ROLE bob LOGIN;
```

```
CREATE ROLE
```

Действительно, получилось и подключиться, и создать пользователя для Боба.

А вот Боб не сможет создать другую роль:

```
student$ /usr/lib/postgresql/13/bin/psql -U bob -d access_roles
```

```
bob=> CREATE ROLE charlie LOGIN;
```

```
ERROR:  permission denied to create role
```

Посмотреть роли, имеющиеся в кластере, можно так:

```
alice=> \du
```

```
                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 alice     | Create role                                                | {}
 bob       |                                                            | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 student   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
```

Кроме только что созданных ролей alice и bob есть еще две роли. Они имеют права суперпользователя:

- postgres — суперпользователь, созданный при инициализации кластера;
- student — роль создана специально для целей курса. Благодаря ей мы можем не указывать параметры подключения при запуске psql.

Существующие роли можно изменять. Например, Алиса может отобрать у Боба право входа:

```
alice=> ALTER ROLE bob NOLOGIN;
```

```
ALTER ROLE
```

Теперь Боб не сможет подключиться:

```
bob=> \c - bob
```

```
\connect: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: FATAL:  role "bob" is not permitted to log in
```

А у себя самой Алиса отберет CREATEROLE:

```
alice=> ALTER ROLE alice NOCREATEROLE;
```

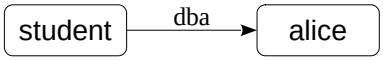
```
ALTER ROLE
```

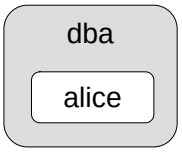
Такие пары, как LOGIN-NOLOGIN или CREATEROLE-NOCREATEROLE, есть и у других атрибутов.

Групповые роли

Включение роли в группу

```
student=> GRANT dba TO alice;
```





Исключение роли из группы

```
student=> REVOKE dba FROM alice;
```

Право управления участием в групповой роли

любая роль может включить другую роль в саму себя

роль с атрибутом SUPERUSER — любую роль в любую другую

роль с атрибутом CREATEROLE — любую роль в любую, кроме суперпользовательской

Роль может быть включена в другую роль подобно тому, как пользователь Unix может быть включен в группу.

PostgreSQL не делает различий между ролями-пользователями и ролями-группами. Поэтому любая роль может быть включена в любую другую. При этом возможно появление цепочек включений (но циклы не допускаются).

Смысл такого включения состоит в том, что для роли становятся доступны атрибуты (и привилегии, о которых пойдет речь дальше), которыми обладает групповая роль. Включение выполняется командой GRANT: возможности групповой роли *предоставляются* другой роли.

Важно, от имени какой роли выполняется команда GRANT. Правом на включение и исключение других ролей в данную роль обладают:

- сама эта роль;

- роли с атрибутом SUPERUSER;

- роли с атрибутом CREATEROLE (если данная роль — не является суперпользовательской).

Чтобы воспользоваться правами, которые дают атрибуты групповой роли, необходимо переключиться в нее командой SET ROLE.

https://postgrespro.ru/docs/postgresql/13/role-membership

Групповые роли

Алиса отобрала у себя атрибут CREATEROLE и теперь не может ни создавать новые роли, ни изменять атрибуты существующих:

```
alice=> ALTER ROLE bob LOGIN;
```

```
ERROR:  permission denied
```

Чтобы наделить Алису супервозможностями, включим ее в суперпользовательскую роль student. Такую команду можно выполнить или под самой ролью student, или под другой суперпользовательской ролью:

```
alice=> \c - postgres
```

```
You are now connected to database "access_roles" as user "postgres".
```

```
postgres=# GRANT student TO alice;
```

```
GRANT ROLE
```

```
postgres=# \du
```

```
                                 List of roles
 Role name |                         Attributes                         | Member of 
-----------+------------------------------------------------------------+-----------
 alice     |                                                            | {student}
 bob       | Cannot login                                               | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 student   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
```

Обратите внимание, что атрибут LOGIN в выводе команде \du не отображается в списке атрибутов. Но отображается его отсутствие.

А чтобы Алиса не злоупотребляла полномочиями, сделаем так, чтобы все ее команды попадали в журнал сообщений. Для этого мы используем еще один вариант установки конфигурационных параметров — значение присваивается параметру при подключении пользователя к серверу:

```
postgres=# ALTER ROLE alice SET log_min_duration_statement=0;
```

```
ALTER ROLE
```

Можно ограничить действие и конкретной базой данных:

```
postgres=# ALTER ROLE alice RESET log_min_duration_statement;
```

```
ALTER ROLE
```

```
postgres=# ALTER ROLE alice IN DATABASE access_roles SET log_min_duration_statement=0;
```

```
ALTER ROLE
```

Алиса не получает возможности групповой роли автоматически. Она может ими воспользоваться, только если переключится на эту роль:

```
postgres=# \c - alice
```

```
You are now connected to database "access_roles" as user "alice".
```

```
alice=> SET ROLE student;
```

```
SET
```

```
alice=> ALTER ROLE bob LOGIN;
```

```
ALTER ROLE
```

Это напоминает команду su в ОС Unix.

Чтобы понять, кем является пользователь на самом деле, и на какую роль он переключился, есть функции:

```
alice=> SELECT session_user, current_user;
```

```
 session_user | current_user 
--------------+--------------
 alice        | student
(1 row)
```

Вернемся к прежней роли:

```
alice=> RESET ROLE;
```

```
RESET
```

```
alice=> SELECT session_user, current_user;
```

```
 session_user | current_user 
--------------+--------------
 alice        | alice
(1 row)
```

И проверим, что команды попали в журнал сообщений:

```
student$ tail -n 5 /var/log/postgresql/postgresql-13-main.log
```

```
2022-05-11 13:16:49.224 MSK [34885] alice@access_roles LOG:  duration: 0.307 ms  statement: SET ROLE student;
2022-05-11 13:16:49.277 MSK [34885] alice@access_roles LOG:  duration: 5.291 ms  statement: ALTER ROLE bob LOGIN;
2022-05-11 13:16:49.350 MSK [34885] alice@access_roles LOG:  duration: 0.270 ms  statement: SELECT session_user, current_user;
2022-05-11 13:16:49.461 MSK [34885] alice@access_roles LOG:  duration: 0.312 ms  statement: RESET ROLE;
2022-05-11 13:16:49.518 MSK [34885] alice@access_roles LOG:  duration: 0.119 ms  statement: SELECT session_user, current_user;
```

Владельцы

Владелец объекта

роль, создавшая объект
(а также роли, включенные в нее)

может быть изменен командой `ALTER ... OWNER TO` *роль*

Когда роль создает в базе данных какие-либо объекты, она становится их *владельцем.* На самом деле владельцами считаются также и роли, включенные в создавшую объект роль.

При необходимости владельца объекта можно сменить. Для этого используется команда ALTER для соответствующего объекта с предложением OWNER TO.

Понятие владельца будет особенно важно для следующей темы этого модуля — привилегий.

Владельцы

Когда Алиса создает какой-либо объект в базе данных, она становится его владельцем.

```
alice=> CREATE TABLE test(id integer);
```

```
CREATE TABLE
```

Как в этом убедиться? Владелец указан в столбце owner:

```
alice=> \dt test
```

```
       List of relations
 Schema | Name | Type  | Owner
--------+------+-------+-------
 public | test | table | alice
(1 row)
```

Удаление ролей

Удалить роль можно, если нет объектов, которыми она владеет.

```
alice=> \c - student
```

```
You are now connected to database "access_roles" as user "student".
```

```
student=# DROP ROLE alice;
```

```
ERROR:  role "alice" cannot be dropped because some objects depend on it
DETAIL:  owner of table test
```

Чтобы удалить роль Алисы, можно передать ее объекты другой роли:

```
student=# REASSIGN OWNED BY alice TO bob;
```

```
REASSIGN OWNED
```

```
student=# \dt test
```

```
       List of relations
 Schema | Name | Type  | Owner
--------+------+-------+-------
 public | test | table | bob
(1 row)
```

```
student=# DROP ROLE alice;
```

```
DROP ROLE
```

Другой вариант — удалить объекты, принадлежащие роли:

```
student=# DROP OWNED BY bob;
```

```
DROP OWNED
```

```
student=# DROP ROLE bob;
```

```
DROP ROLE
```

Надо только иметь в виду, что роль может владеть объектами в разных базах данных.

Итоги

Роли объединяют концепции пользователей и групп

Атрибуты определяют свойства ролей

Роли можно включать друг в друга

У каждого объекта базы данных есть роль-владелец

Практика

1. Создайте роль creator без права входа в систему, но с правом создания баз данных и ролей.

   Создайте пользователя weak с правом входа в систему.
2. Убедитесь, что weak не может создать базу данных.
3. Включите пользователя weak в группу creator.

   Создайте новую базу данных под пользователем weak.

1. Создание ролей

```
student=# CREATE ROLE creator WITH CREATEDB CREATEROLE;
CREATE ROLE
student=# CREATE ROLE weak WITH LOGIN;
CREATE ROLE
```

2. Проверка возможности создания БД

```
student=# \c - weak
You are now connected to database "student" as user "weak".
weak=> CREATE DATABASE access_roles;
ERROR:  permission denied to create database
```

3. Включение в группу

```
weak=> \c - student
You are now connected to database "student" as user "student".
student=# GRANT creator TO weak;
GRANT ROLE
student=# \c - weak
You are now connected to database "student" as user "weak".
weak=> SET ROLE creator;
SET
weak=> CREATE DATABASE access_roles;
CREATE DATABASE
```

Практика+

1. Создайте роли alice и bob с правом входа в систему.

   Создайте таблицу под ролью alice.

2. Сделайте необходимые настройки так, чтобы обе роли могли изменять структуру таблицы (например, добавлять столбцы командой ALTER TABLE).

3. Изменять структуру могут только владельцы таблицы. Нужно сделать так, чтобы владельцем таблицы была не только Алиса, но и Боб.

1. Роли и таблица

```
student=# CREATE ROLE alice WITH LOGIN;
```

```
CREATE ROLE
```

```
student=# CREATE ROLE bob WITH LOGIN;
```

```
CREATE ROLE
```

```
student=# \c access_roles alice
```

```
You are now connected to database "access_roles" as user "alice".
```

```
alice=> CREATE TABLE test (id integer);
```

```
CREATE TABLE
```

2. Добавление владельца таблицы

Чтобы Боб мог изменять структуру таблицы, он должен стать ее владельцем. Для этого можно включить роль bob в роль alice. Такую команду может выполнить Алиса:

```
alice=> GRANT alice TO bob;
```

```
GRANT ROLE
```

```
alice=> \du alice|bob
```

```
           List of roles
 Role name | Attributes | Member of
-----------+------------+-----------
 alice     |            | {}
 bob       |            | {alice}
```

Теперь Боб может добавить столбец:

```
alice=> \c - bob
```

```
You are now connected to database "access_roles" as user "bob".
```

```
bob=> ALTER TABLE test ADD description text;
```

```
ALTER TABLE
```

И даже удалить таблицу:

```
bob=> DROP TABLE test;
```

```
DROP TABLE
```