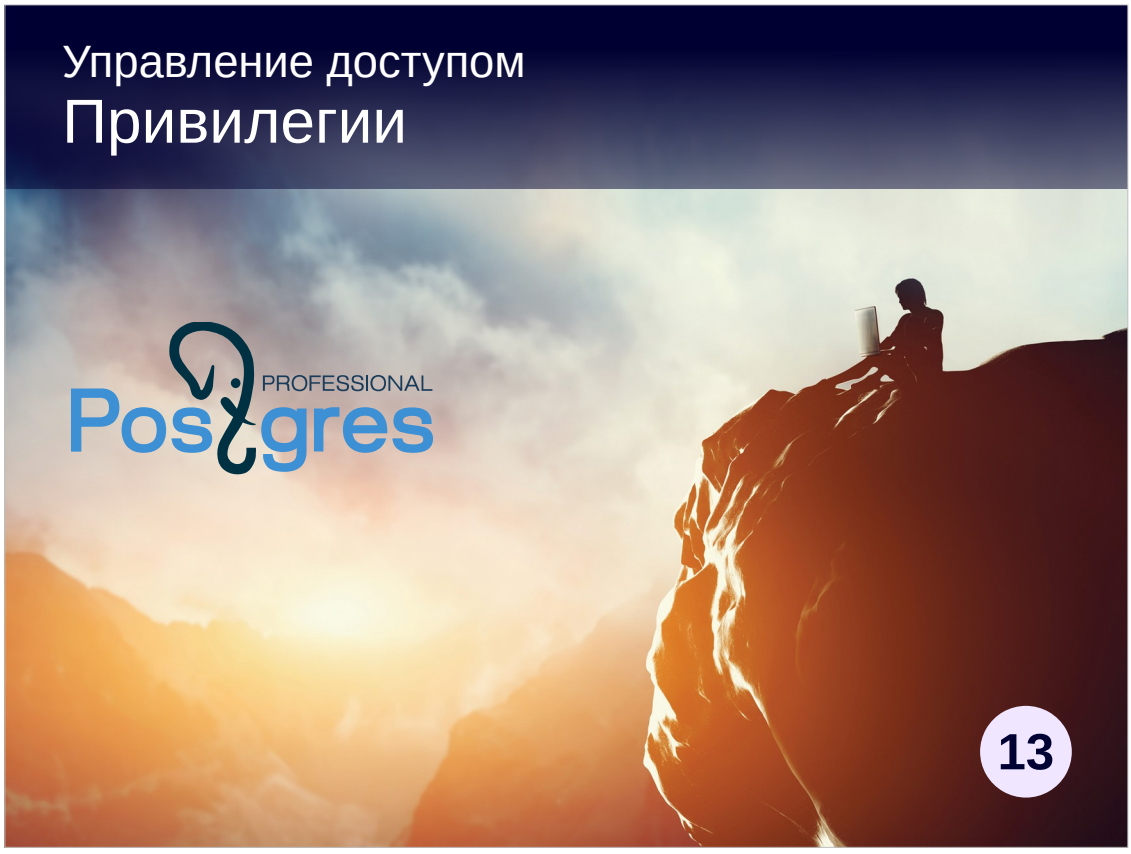




Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов



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

# Темы

Виды привилегий для разных объектов

Категории ролей с точки зрения управления доступом

Групповые привилегии

Выдача и отзыв привилегий и право передачи

Привилегии по умолчанию

Примеры управления доступом

# Привилегии

Привилегии определяют права доступа ролей к объектам

Таблицы

| | | |
|---|---|---|
| `SELECT` | чтение данных | можно на уровне столбцов |
| `INSERT` | вставка строк | |
| `UPDATE` | изменение строк | |
| `REFERENCES` | внешний ключ | |
| `DELETE` | удаление строк | |
| `TRUNCATE` | опустошение таблицы | |
| `TRIGGER` | создание триггеров | |

Представления

| | |
|---|---|
| `SELECT` | чтение данных |
| `TRIGGER` | создание триггеров |



Привилегии определяются на пересечении объектов кластера и ролей. Они ограничивают действия, доступные для ролей в отношении этих объектов.

Список возможных привилегий отличается для объектов различных типов. Привилегии для объектов основных типов приведены на этом и следующем слайдах.

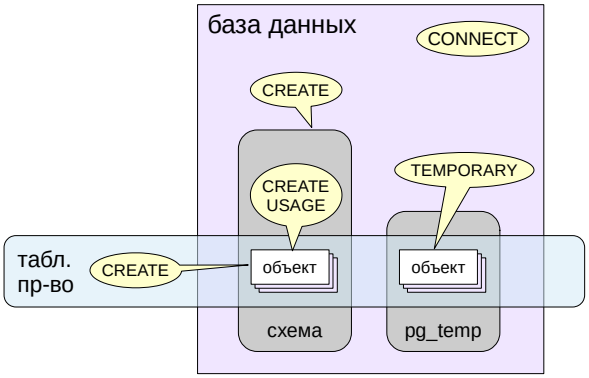
Больше всего привилегий определено для таблиц. Некоторые из них можно определить не только для всей таблицы, но и для отдельных столбцов.

https://postgrespro.ru/docs/postgresql/13/ddl-priv

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

# Привилегии

Табличные пространства,
базы данных, схемы




Последовательности

| | | | |
|---|---|---|---|
| SELECT | currval | | |
| UPDATE | | nextval | setval |
| USAGE | currval | nextval | |



Возможно, несколько неожиданный набор привилегий имеют последовательности. Выбирая нужные, можно разрешить или запретить доступ к трем управляющим функциям.

Для табличных пространств есть привилегия CREATE, разрешающая создание объектов в этом пространстве.

Для баз данных привилегия CREATE разрешает создавать схемы в этой БД, а для схемы привилегия CREATE разрешает создавать объекты в этой схеме.

Поскольку точное имя схемы для временных объектов заранее неизвестно, привилегия на создание временных таблиц вынесена на уровень БД (TEMPORARY).

Привилегия USAGE схемы разрешает обращаться к объектам в этой схеме.

Привилегия CONNECT базы данных разрешает подключение к этой БД.

## Категории ролей

Суперпользователи

полный доступ ко всем объектам — проверки не выполняются

Владельцы

доступ в рамках выданных привилегий
(изначально получает полный набор)

а также действия, не регламентируемые привилегиями,
например: удаление, выдача и отзыв привилегий и т. п.

Остальные роли

доступ исключительно в рамках выданных привилегий



В целом можно сказать, что доступ роли к объекту определяется привилегиями. Но можно выделить три категории ролей.

1. Проще всего с ролями с атрибутом суперпользователя. Такие роли могут делать все, что угодно — для них проверки разграничения доступа не выполняются.

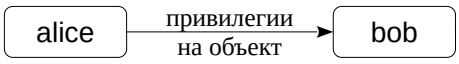
2. Владелец объекта сразу получает полный набор привилегий для этого объекта. В принципе, эти привилегии можно отозвать, но владелец объекта обладает также неотъемлемым правом совершать действия, не регламентируемые привилегиями. В частности, он может выдавать и отзывать привилегии (в том числе и себе самому), удалять объект и т. п.

3. Все остальные роли имеют доступ к объекту только в рамках выданных им привилегий.

# Управление привилегиями

Выдача привилегии

```
alice=> GRANT привилегии ON объект TO bob;
```




одна привилегия может быть независимо выдана разными ролями

Отзыв привилегии

```
alice=> REVOKE привилегии ON объект FROM bob;
```



Право выдачи и отзыва привилегий на объект имеет владелец этого объекта (и суперпользователь).

Синтаксис команд GRANT и REVOKE достаточно сложен и позволяет указывать как отдельные, так и все возможные привилегии; как отдельные объекты, так и группы объектов, входящие в определенные схемы и т. п.

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

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

### Привилегии

```
student=# CREATE DATABASE access_privileges;
```

```
CREATE DATABASE
```

```
student=# \c access_privileges
```

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

В нашем примере Алиса будет владельцем нескольких объектов в своей схеме.

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

```
CREATE ROLE
```

```
student=# CREATE SCHEMA alice;
```

```
CREATE SCHEMA
```

```
student=# GRANT CREATE, USAGE ON SCHEMA alice TO alice;
```

```
GRANT
```

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

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

Алиса создает пару таблиц.

```
alice=> CREATE TABLE t1(n integer);
```

```
CREATE TABLE
```

```
alice=> CREATE TABLE t2(n integer, m integer);
```

```
CREATE TABLE
```

Вторая роль, Боб, будет пытаться обращаться к объектам Алисы.

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

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

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

```
CREATE ROLE
```

Боб пробует обратиться к таблице t1.

```
student$ psql -U bob -d access_privileges
```

```
bob=> SELECT * FROM alice.t1;
```

```
ERROR:  permission denied for schema alice
LINE 1: SELECT * FROM alice.t1;
                      ^
```

В чем причина ошибки?

---

У Боба нет доступа к схеме, так как он не суперпользователь, не владелец схемы, и не имеет нужных привилегий.

```
student=# \dn+ alice
```

```
                  List of schemas
 Name  |  Owner  | Access privileges  | Description
-------+---------+--------------------+-------------
 alice | student | student=UC/student+|
       |         | alice=UC/student   |
(1 row)
```

В каждой строке access privileges отображается роль, ее привилегии и кем они были выданы:

роль=привилегии/кем_предоставлены

Названия привилегий обозначаются одной буквой. Привилегии для схем:

- U = usage
- C = create

---

Алисе надо выдать Бобу доступ к своей схеме.

```
student=# \c - alice

You are now connected to database "access_privileges" as user "alice".

alice=> GRANT CREATE, USAGE ON SCHEMA alice TO bob;

WARNING:  no privileges were granted for "alice"
GRANT
```

Почему привилегия не выдалась?

---

Потому что Алиса не является владельцем схемы.

```
alice=> \dn+ alice

                  List of schemas
 Name  |  Owner  | Access privileges  | Description
-------+---------+--------------------+-------------
 alice | student | student=UC/student+|
       |         | alice=UC/student   |
(1 row)
```

Сделаем Алису владельцем:

```
alice=> \c - student

You are now connected to database "access_privileges" as user "student".

student=# ALTER SCHEMA alice OWNER TO alice;

ALTER SCHEMA

student=# \dn+ alice

                 List of schemas
 Name  | Owner | Access privileges | Description
-------+-------+-------------------+-------------
 alice | alice | alice=UC/alice    |
(1 row)
```

Теперь Алиса сможет выдать доступ Бобу:

```
student=# \c - alice

You are now connected to database "access_privileges" as user "alice".

alice=> GRANT CREATE, USAGE ON SCHEMA alice TO bob;

GRANT
```

И Боб снова пытается прочитать таблицу:

```
bob=> SELECT * FROM alice.t1;

ERROR:  permission denied for table t1
```

В чем причина ошибки на этот раз?

---

Сейчас у Боба есть доступ к схеме, но нет доступа к самой таблице.

```
bob=> \dp alice.t1

                            Access privileges
 Schema | Name | Type  | Access privileges | Column privileges | Policies
--------+------+-------+-------------------+-------------------+----------
 alice  | t1   | table |                   |                   |
(1 row)
```

Пустое поле access privileges означает, что у владельца есть полный набор привилегий, а кроме него никто не имеет доступа.

---

Алиса должна дать Бобу доступ на чтение:

```
alice=> GRANT SELECT ON t1 TO bob;

GRANT
```

Посмотрим, как изменились привилегии:

```
alice=> \dp t1
```

```
                            Access privileges
 Schema | Name | Type  |  Access privileges  | Column privileges | Policies
--------+------+-------+---------------------+-------------------+----------
 alice  | t1   | table | alice=arwdDxt/alice+|                   |
        |      |       | bob=r/alice         |                   |
(1 row)
```

Теперь пустое поле «проявилось», и мы видим, что для Алисы в нем находится полный перечень привилегий. Вот обозначения привилегий для таблиц, не все из них вполне очевидные:

- a = insert
- r = select
- w = update
- d = delete
- D = truncate
- x = reference
- t = trigger

---

На этот раз у Боба все получается:

```
bob=> SELECT * FROM alice.t1;
```

```
 n
---
(0 rows)
```

А, например, добавить строку в таблицу он не сможет:

```
bob=> INSERT INTO alice.t1 VALUES (42);
```

```
ERROR:  permission denied for table t1
```

Некоторые привилегии можно выдать на определенные столбцы:

```
alice=> GRANT INSERT(n,m) ON t2 TO bob;
```

```
GRANT
```

```
alice=> GRANT SELECT(m) ON t2 TO bob;
```

```
GRANT
```

```
alice=> \dp t2
```

```
                            Access privileges
 Schema | Name | Type  | Access privileges | Column privileges | Policies
--------+------+-------+-------------------+-------------------+----------
 alice  | t2   | table |                   | n:                +|
        |      |       |                   |   bob=a/alice     +|
        |      |       |                   | m:                +|
        |      |       |                   |   bob=ar/alice     |
(1 row)
```

Теперь Боб может добавлять строки в t2:

```
bob=> INSERT INTO alice.t2(n,m) VALUES (1,2);
```

```
INSERT 0 1
```

А читать сможет только один столбец:

```
bob=> SELECT * FROM alice.t2;
```

```
ERROR:  permission denied for table t2
```

```
bob=> SELECT m FROM alice.t2;
```

```
 m
---
 2
(1 row)
```

Если необходимо, Алиса может выдать Бобу все привилегии, не перечисляя их явно.

```
alice=> GRANT ALL ON t1 TO bob;
```

```
GRANT
```

```
alice=> \dp t1
```

```
                            Access privileges
 Schema | Name | Type  |  Access privileges  | Column privileges | Policies
--------+------+-------+---------------------+-------------------+----------
 alice  | t1   | table | alice=arwdDxt/alice+|                   |
        |      |       | bob=arwdDxt/alice   |                   |
(1 row)
```

Теперь Бобу доступны все действия, например, удаление строк:

```
bob=> DELETE FROM alice.t1;
```

```
DELETE 0
```

А удаление самой таблицы?

```
bob=> DROP TABLE alice.t1;
```

```
ERROR:  must be owner of table t1
```

Удалить таблицу может только владелец (или суперпользователь), специальной привилегии для этого не существует.

# Групповые привилегии

Роль получает привилегии групповых ролей,
в которые она включена

- с атрибутом INHERIT привилегии наследуются автоматически
- с атрибутом NOINHERIT требуется явное переключение командой SET ROLE

Псевдороль public

- неявно включает все остальные роли



Роль может получать привилегии для доступа к объекту не только непосредственно, но и от групповых ролей, в которые она входит. Таким образом, для упрощения администрирования можно выдать групповой роли необходимый набор привилегий и затем выдавать пользователям весь этот набор целиком. Групповую роль можно рассматривать как особую привилегию; поэтому управление группами и выполняется теми же командами GRANT и REVOKE, что и управление привилегиями.

Роль с атрибутом INHERIT (по умолчанию) автоматически обладает привилегиями всех групп, в которые она входит. Это касается и псевдороли public, в которую неявно входят все роли.

Если же роль создана как NOINHERIT, то она может воспользоваться привилегиями группы, только выполнив команду SET ROLE и таким образом переключившись на эту групповую роль. Все действия, совершаемые ролью, будут совершаться от имени групповой роли (например, групповая роль будет владельцем созданных объектов).

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

# Преднастроенные роли

| | | |
|---|---|---|
| pg_signal_backend | — завершение сеансов и отмена запросов | |
| pg_read_all_settings | — чтение конфигурационных параметров | pg_monitor |
| pg_read_all_stats | — доступ к статистике | |
| pg_stat_scan_tables | — доступ к статистике, блокирующей доступ | |
| pg_read_server_files | — чтение файлов на сервере | |
| pg_write_server_files | — запись файлов на сервере | |
| pg_execute_server_programs | — выполнение программ на сервере | |



В PostgreSQL уже есть ряд преднастроенных ролей, обладающих рядом специальных привилегий для выполнения задач, которые обычно может совершать только суперпользователь. Три последние роли появилась в версии PostgreSQL 11.

Полный список всех ролей, включая предопределенные системные роли, можно посмотреть в psql командой \duS.

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

Аналогично можно создавать и собственные групповые роли, например, для управления резервным копированием и т. п.

### Групповые привилегии

Пусть Алиса выдаст роли public привилегию изменения t2:

```
alice=> GRANT UPDATE ON t2 TO public;
```

```
GRANT
```

```
alice=> \dp t2
```

```
                            Access privileges
 Schema | Name | Type  |  Access privileges  | Column privileges | Policies
--------+------+-------+---------------------+-------------------+----------
 alice  | t2   | table | alice=arwdDxt/alice+| n:               +|
        |      |       | =w/alice            |   bob=a/alice    +|
        |      |       |                     | m:               +|
        |      |       |                     |   bob=ar/alice    |
(1 row)
```

Пустая роль (слева от знака равенства) обозначает public.

---

Проверим, сможет ли Боб воспользоваться этой привилегией.

```
bob=> UPDATE alice.t2 SET n = n + 1;
```

```
ERROR:  permission denied for table t2
```

В чем причина ошибки?

---

Дело в том, что перед обновлением t2, сначала необходимо выбрать нужные строки, а для этого требуется привилегия чтения (как минимум, столбца n, который используется в условии). Но Боб имеет право читать только столбец m.

```
alice=> GRANT SELECT ON t2 TO bob;
```

```
GRANT
```

```
alice=> \dp t2
```

```
                            Access privileges
 Schema | Name | Type  |  Access privileges  | Column privileges | Policies
--------+------+-------+---------------------+-------------------+----------
 alice  | t2   | table | alice=arwdDxt/alice+| n:               +|
        |      |       | =w/alice           +|   bob=a/alice    +|
        |      |       | bob=r/alice         | m:               +|
        |      |       |                     |   bob=ar/alice    |
(1 row)
```

Вот теперь у Боба появилась возможность обновления таблицы:

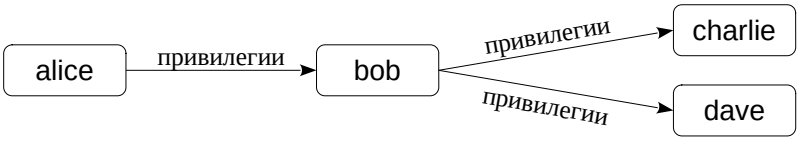
```
bob=> UPDATE alice.t2 SET n = n + 1;
```

```
UPDATE 1
```

# Передача права

Выдача привилегии с правом передачи

```
alice=> GRANT привилегии ON объект TO bob WITH GRANT OPTION;
```




Отзыв привилегии

```
alice=> REVOKE привилегии ON объект FROM bob CASCADE;
```

Отзыв права передачи

```
alice=> REVOKE GRANT OPTION FOR
        привилегии ON объект FROM bob CASCADE;
```

обязательно, если привилегия была передана другим ролям



При выдаче роли полномочия можно передать ей право дальнейшей передачи (и отзыва) этого полномочия. Это выполняется командой GRANT … WITH GRANT OPTION. Если роль воспользуется этим правом, образуется иерархия ролей.

Отозвать привилегию можно с помощью команды REVOKE. Роль может отозвать привилегию только у той роли, которой она его выдала. Например, в ситуации, показанной на слайде, alice не может отозвать привилегию непосредственно у charlie или dave.

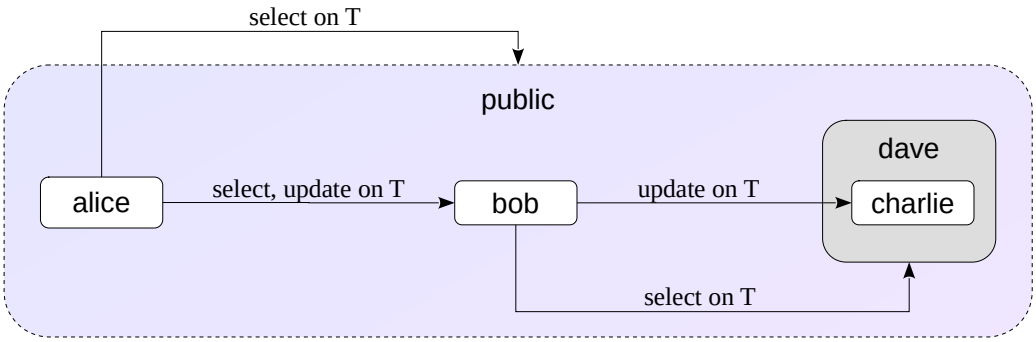
Однако при отзыве привилегии у bob привилегия будет автоматически отозвана у всех ролей в иерархии. Для этого надо указать ключевое слово CASCADE (если иерархия непуста, то без CASCADE будет ошибка).

Право передачи можно отозвать, не отзывая у роли саму привилегию. Это выполняется с помощью команды REVOKE GRANT OPTION FOR. Слово CASCADE имеет здесь такое же значение, как и при отзыве привилегии.

# Вопрос

Привилегии на таблицу T получены Бобом от Алисы.

Если Алиса выполнит
REVOKE ALL ON T FROM bob CASCADE,
какие привилегии останутся у Чарли и Дейва?






И у Чарли, и у Дейва останется только привилегия на чтение T, полученная от public. Все привилегии, выданные Бобом другим ролям, будут отозваны.

### Передача прав

Создадим третью роль, для Чарли. Боб попробует передать Чарли права на таблицу t1, принадлежащую Алисе.

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

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

```
student=# CREATE ROLE charlie LOGIN;
```

```
CREATE ROLE
```

У Боба есть полный доступ к t1:

```
bob=> \dp alice.t1
```

```
                            Access privileges
 Schema | Name | Type  |  Access privileges  | Column privileges | Policies
--------+------+-------+---------------------+-------------------+----------
 alice  | t1   | table | alice=arwdDxt/alice+|                   |
        |      |       | bob=arwdDxt/alice   |                   |
(1 row)
```

Но он не может передать свои привилегии Чарли:

```
bob=> GRANT SELECT ON alice.t1 TO charlie;
```

```
WARNING:  no privileges were granted for "t1"
GRANT
```

Чтобы это было возможно, Алиса должна выдать Бобу разрешение.

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

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

```
alice=> GRANT SELECT,UPDATE ON t1 TO bob WITH GRANT OPTION;
```

```
GRANT
```

```
alice=> \dp alice.t1
```

```
                            Access privileges
 Schema | Name | Type  |  Access privileges  | Column privileges | Policies
--------+------+-------+---------------------+-------------------+----------
 alice  | t1   | table | alice=arwdDxt/alice+|                   |
        |      |       | bob=ar*w*dDxt/alice |                   |
(1 row)
```

Звездочки справа от символа привилегии показывают право передачи.

---

Теперь Боб может поделиться с Чарли привилегиями, в том числе и правом передачи:

```
bob=> GRANT SELECT ON alice.t1 TO charlie WITH GRANT OPTION;
```

```
GRANT
```

```
bob=> GRANT UPDATE ON alice.t1 TO charlie;
```

```
GRANT
```

```
bob=> \dp alice.t1
```

```
                            Access privileges
 Schema | Name | Type  |  Access privileges  | Column privileges | Policies
--------+------+-------+---------------------+-------------------+----------
 alice  | t1   | table | alice=arwdDxt/alice+|                   |
        |      |       | bob=ar*w*dDxt/alice+|                   |
        |      |       | charlie=r*w/bob     |                   |
(1 row)
```

Роль может получить одну и ту же привилегию от разных ролей. Обратите внимание: если привилегия выдается суперпользователем, она выдается от имени владельца:

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

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

```
student=# GRANT UPDATE ON alice.t1 TO charlie;

GRANT

student=# \dp alice.t1

                            Access privileges
 Schema | Name | Type  |  Access privileges  | Column privileges | Policies
--------+------+-------+---------------------+-------------------+----------
 alice  | t1   | table | alice=arwdDxt/alice+|                   |
        |      |       | bob=ar*w*dDxt/alice+|                   |
        |      |       | charlie=r*w/bob    +|                   |
        |      |       | charlie=w/alice     |                   |
(1 row)
```

Роль может отозвать привилегии только у той роли, которой она их непосредственно выдала. Например, Алиса не сможет отозвать право передачи у Чарли, потому что она его не выдавала.

```
student=# \c - alice

You are now connected to database "access_privileges" as user "alice".

alice=> REVOKE GRANT OPTION FOR SELECT ON alice.t1 FROM charlie;

REVOKE
```

Никакой ошибки не фиксируется, но и права не изменяются:

```
alice=> \dp alice.t1

                            Access privileges
 Schema | Name | Type  |  Access privileges  | Column privileges | Policies
--------+------+-------+---------------------+-------------------+----------
 alice  | t1   | table | alice=arwdDxt/alice+|                   |
        |      |       | bob=ar*w*dDxt/alice+|                   |
        |      |       | charlie=r*w/bob    +|                   |
        |      |       | charlie=w/alice     |                   |
(1 row)
```

В то же время Алиса не может просто так отозвать привилегии у Боба, если он успел передать их кому-либо еще:

```
alice=> REVOKE GRANT OPTION FOR SELECT ON alice.t1 FROM bob;

ERROR:  dependent privileges exist
HINT:  Use CASCADE to revoke them too.
```

В таком случае привилегии надо отзывать по всей иерархии передачи с помощью CASCADE:

```
alice=> REVOKE GRANT OPTION FOR SELECT ON alice.t1 FROM bob CASCADE;

REVOKE

alice=> \dp alice.t1

                            Access privileges
 Schema | Name | Type  |  Access privileges  | Column privileges | Policies
--------+------+-------+---------------------+-------------------+----------
 alice  | t1   | table | alice=arwdDxt/alice+|                   |
        |      |       | bob=arw*dDxt/alice +|                   |
        |      |       | charlie=w/bob      +|                   |
        |      |       | charlie=w/alice     |                   |
(1 row)
```

Как видим, у Боба пропало право передачи привилегии, а у Чарли была отозвана и сама привилегия.

Аналогично можно отозвать по иерархии и привилегию.

```
alice=> REVOKE SELECT ON alice.t1 FROM bob CASCADE;

REVOKE

alice=> \dp alice.t1

                            Access privileges
 Schema | Name | Type  |  Access privileges  | Column privileges | Policies
--------+------+-------+---------------------+-------------------+----------
 alice  | t1   | table | alice=arwdDxt/alice+|                   |
        |      |       | bob=aw*dDxt/alice  +|                   |
        |      |       | charlie=w/bob      +|                   |
        |      |       | charlie=w/alice     |                   |
(1 row)
```

# Подпрограммы

Единственная привилегия для функций и процедур

| | |
|---|---|
| EXECUTE | выполнение |

Характеристики безопасности

| | |
|---|---|
| SECURITY INVOKER | выполняется с правами вызывающего (по умолчанию) |
| SECURITY DEFINER | выполняется с правами владельца |



Для подпрограмм (функций и процедур) есть единственная привилегия EXECUTE, разрешающая выполнение этой подпрограммы.

Тонкий момент состоит в том, от имени какого пользователя будет выполняться подпрограмма. Если подпрограмма объявлена как SECURITY INVOKER (по умолчанию), она выполняется с правами вызывающего пользователя. В этом случае операторы внутри подпрограммы смогут обращаться только к тем объектам, к которым у вызывающего пользователя есть доступ.

Если же указать фразу SECURITY DEFINER, подпрограмма работает с правами ее владельца. Так можно позволить другим пользователям выполнять определенные действия над объектами, к которым у них нет непосредственного доступа.

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

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

# Привилегии public

По умолчанию роль public получает ряд привилегий

| | |
|---|---|
| для баз данных | `CONNECT` (подключение)<br>`TEMPORARY` (создание временных таблиц) |
| для схемы public | `CREATE` (создание объектов)<br>`USAGE` (доступ к объектам) |
| для схем pg_catalog и information_schema | `USAGE` (доступ к объектам) |
| для подпрограмм | `EXECUTE` (выполнение) |

Удобно, но не безопасно



По умолчанию псевдороль public получает ряд привилегий (то есть, фактически, их получают все роли):

- подключение и создание временных таблиц для **всех** баз данных;
- использование схемы **public** и создание в ней объектов;
- использование схем **pg_catalog** и **information_schema**;
- выполнение **всех** функций и процедур.

(В PostgreSQL 15 роль public лишится права создания объектов.)

Такое поведение может оказаться нежелательным. В этом случае надо явно отозвать у public привилегии. Это можно сделать и в шаблонной базе данных template1, чтобы изменения распространялись на новые базы данных. Но для отзыва прав на подпрограммы надо пользоваться механизмом привилегий по умолчанию, о котором будет сказано дальше в этой теме.

## Подпрограммы

Алиса создает простую функцию, возвращающую число строк в таблице t1:

```
alice=> CREATE FUNCTION foo() RETURNS bigint AS $$
  SELECT count(*) FROM t1;
$$ LANGUAGE sql;

CREATE FUNCTION

alice=> INSERT INTO t1 VALUES (1);

INSERT 0 1

alice=> SELECT foo();

 foo
-----
   1
(1 row)
```

Псевдороль public автоматически получает привилегию EXECUTE для любой создаваемой функции. Поэтому, например, Боб может выполнить функцию, которую только что создала Алиса.

Отчасти это компенсируется тем, что по умолчанию (или при явном указании SECURITY INVOKER) функция выполняется с правами вызывающей роли:

```
bob=> SET search_path = public, alice;

SET

bob=> SELECT foo();

ERROR:  permission denied for table t1
CONTEXT:  SQL function "foo" statement 1
```

Поэтому Боб не сможет получить доступ к объектам, на которые ему не выданы привилегии.

---

Поскольку в функции не указана схема таблицы t1, Боб может создать свою таблицу t1 и функция будет работать с той из них, которая окажется в пути поиска первой:

```
bob=> CREATE TABLE t1(n numeric);

CREATE TABLE

bob=> SELECT foo();

 foo
-----
   0
(1 row)

alice=> SELECT foo();

 foo
-----
   1
(1 row)
```

---

Другой возможный вариант — объявить функцию как работающую с правами создавшего (SECURITY DEFINER):

```
alice=> ALTER FUNCTION foo() SECURITY DEFINER;

ALTER FUNCTION
```

В этом случае функция работает с правами создавшей ее роли, независимо от того, кто ее вызывает. Боб удаляет свою таблицу:

```
bob=> DROP TABLE t1;

DROP TABLE
```

И получает доступ к таблице Алисы:

```
bob=> SELECT foo();
```

```
 foo
-----
   1
(1 row)
```

В таком случае, конечно, надо внимательно следить за выданными привилегиями. Скорее всего, потребуется отозвать привилегию EXECUTE у роли public и выдавать ее явно только нужным ролям.

```
alice=> REVOKE EXECUTE ON ALL FUNCTIONS IN SCHEMA alice FROM public;
```

```
REVOKE
```

```
bob=> SELECT foo();
```

```
ERROR:  permission denied for function foo
```

# Привилегии по умолчанию

Способ выдать или отозвать привилегии при создании объекта

```
ALTER DEFAULT PRIVILEGES
[ FOR ROLE список_целевых_ролей ]
[ IN SCHEMA схема ]
  GRANT привилегии ON класс_объектов TO роль;

ALTER DEFAULT PRIVILEGES
  REVOKE привилегии ON класс_объектов FROM роль;
```




Можно настроить дополнительные привилегии, выдаваемые или отзываемые при создании объекта. Для этого используется команда ALTER DEFAULT PRIVILEGES.

Механизм срабатывает, когда *целевая_роль* (по умолчанию — текущий пользователь) создает объект, принадлежащий указанному *классу_объектов* (например, таблицы или функции) в указанной *схеме* (по умолчанию — в любой). Предложение GRANT говорит о том, что на созданный объект надо выдать указанные привилегии для указанной роли. Предложение REVOKE, наоборот, используется для отзыва привилегий.

Для роли public механизм привилегий по умолчанию выдает право выполнения при создании подпрограмм. Чтобы public не получал такое право, надо выдать команду

ALTER DEFAULT PRIVILEGES
FOR ROLE …
REVOKE EXECUTE ON ROUTINES FROM public;

При этом в предложении FOR ROLE надо указать все роли, которые могут создать подпрограммы.

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

### Привилегии по умолчанию

Чтобы псевдороль public не получала право на выполнение новых функций, такое право надо у нее отозвать.

Проверим текущие настройки:

```
alice=> \ddp
```

```
         Default access privileges
 Owner | Schema | Type | Access privileges
-------+--------+------+-------------------
(0 rows)
```

Таблица выглядит пустой, но мы знаем, что за пустыми полями скрываются значения по умолчанию. Такое значение можно найти в документации или получить следующим образом:

```
alice=> SELECT acldefault('function', 'alice'::regrole);
```

```
        acldefault
--------------------------
 {=X/alice,alice=X/alice}
(1 row)
```

Это значение включает право выполнения для public. Отзовем его:

```
alice=> ALTER DEFAULT PRIVILEGES
REVOKE EXECUTE ON ROUTINES FROM public;
```

```
ALTER DEFAULT PRIVILEGES
```

Теперь в «проявившемся» поле право для public отсутствует:

```
alice=> \ddp
```

```
           Default access privileges
 Owner | Schema |   Type   | Access privileges
-------+--------+----------+-------------------
 alice |        | function | alice=X/alice
(1 row)
```

Теперь, когда Алиса будет создавать новые функции, Боб не сможет их выполнять:

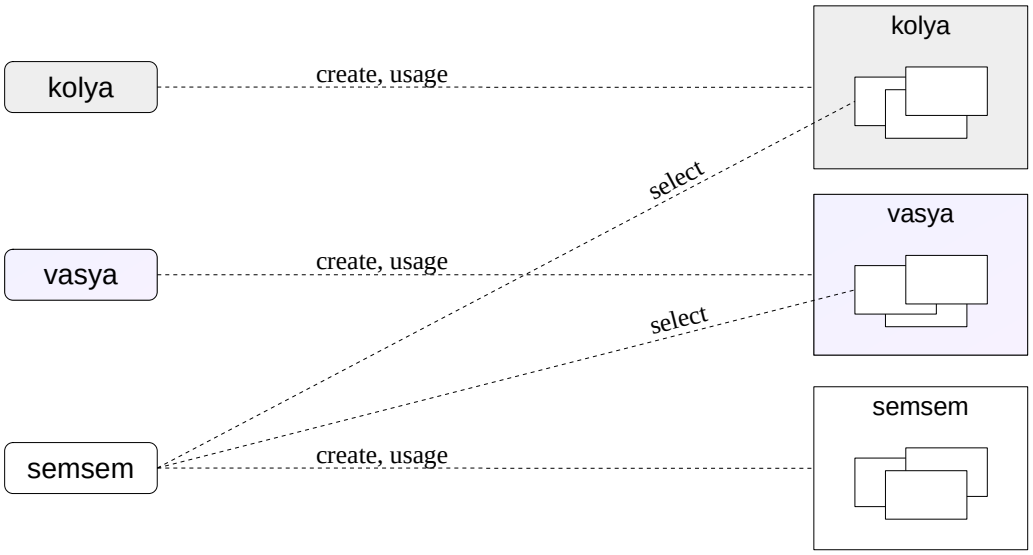
```
alice=> CREATE FUNCTION bar() RETURNS integer AS $$
SELECT 1;
$$ LANGUAGE sql SECURITY DEFINER;
```

```
CREATE FUNCTION
```

```
bob=> SELECT bar();
```

```
ERROR:  permission denied for function bar
```

# Пример 1






Механизмы управления доступом (роли, атрибуты и привилегии, а также схемы) весьма гибки и позволяют в разных случаях организовать работу удобным образом. Приведем несколько примеров.

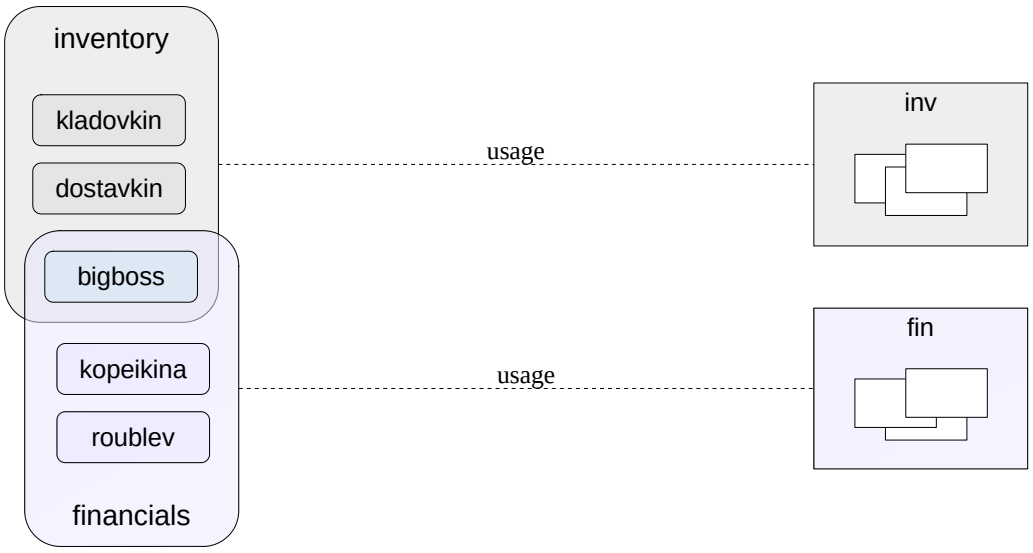
Пример простого использования.

Семен Семенович и его студенты Коля и Вася занимаются научными исследованиями и хранят результаты измерений в базе данных.

Администратор создал на факультетском сервере БД каждому из них своего пользователя и одноименную схему. Путь поиска он оставил по умолчанию. Групповые роли не используются.

Каждый пользователь является владельцем объектов, которые он создает в своей схеме. Кроме того, Коля и Вася дают доступ к некоторым своим таблицам Семену Семеновичу, чтобы он мог посмотреть их результаты.

# Пример 2






Более сложный пример.

На выделенный сервер БД установлена система автоматизации предприятия. Она состоит из двух модулей — складского и финансового.

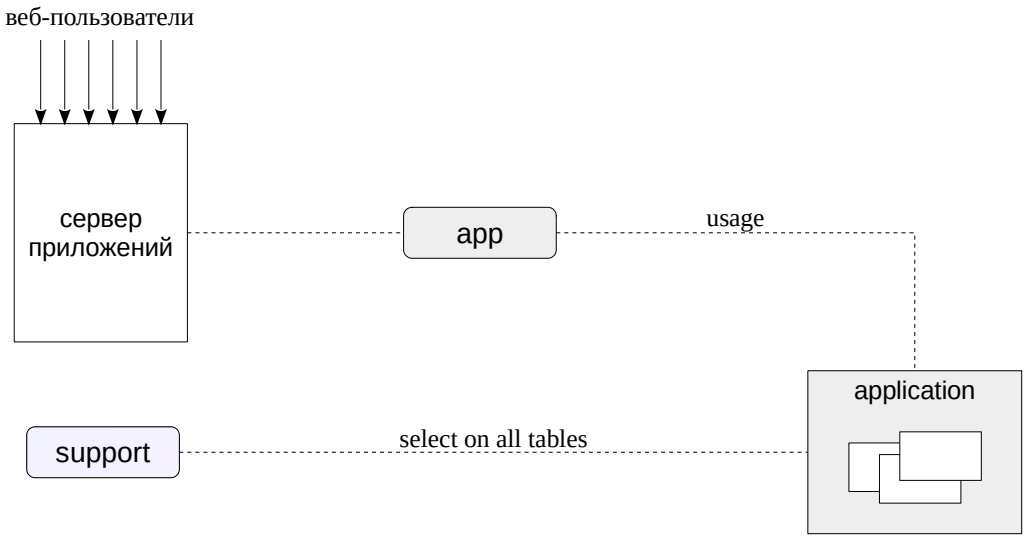
Для каждого из модулей создана своя схема (inv для склада и fin для финансов) и своя групповая роль (inventory для склада и financials для финансов). Групповая роль является владельцем всех объектов в своей схеме.

В складском отделе работают Кладовкин и Доставкин. Для каждого создан свой пользователь и включен в группу inventory.

В финансовом отделе работают Копейкина и Рублев. Для каждого создан свой пользователь и включен в группу financials.

Для генерального директора предприятия на всякий случай создан пользователь, включенный в обе группы, хотя вряд ли он когда-либо воспользуется системой.

# Пример 3






Очень часто бывает так, что аутентификация пользователей информационной системы происходит не в СУБД, а на сервере приложений. Действительно, если у системы тысячи пользователей и есть возможность онлайн-регистрации, нет никакого смысла управлять ими в СУБД.

В этом случае сервер приложений подключается к базе данных под одной заранее созданной ролью, а информацию о пользователе передает при необходимости в виде какого-то контекста.

Но даже и в этом случае в СУБД наверняка понадобятся несколько ролей. Например, специальная роль для технической поддержки, сотрудники которой смогут читать таблицы основного сервера для быстрого решения проблем.

# Итоги

Привилегии определяют права доступа ролей к объектам

Роли, атрибуты и привилегии, схемы — гибкий механизм, позволяющий по-разному организовать работу

- можно легко разрешить все всем
- можно строго разграничить доступ, если это необходимо

# Практика

Настройте привилегии таким образом, чтобы одни пользователи имели полный доступ к таблицам, а другие могли только запрашивать, но не изменять информацию.

1. Создайте новую базу данных и две роли: writer и reader.
2. Отзовите у роли public все привилегии на схему public, выдайте роли writer обе привилегии, а роли reader — только usage.
3. Настройте привилегии по умолчанию так, чтобы роль reader получала доступ на чтение к таблицам, принадлежащим writer в схеме public.
4. Создайте пользователей w1 в группе writer и r1 в группе reader.
5. Под пользователем writer создайте таблицу.
6. Убедитесь, что r1 имеет доступ к таблице только на чтение, а w1 имеет к ней полный доступ, включая удаление.

## 1. База данных и роли

```
=> CREATE DATABASE access_privileges;
```

```
CREATE DATABASE
```

```
=> CREATE USER writer;
```

```
CREATE ROLE
```

```
=> CREATE USER reader;
```

```
CREATE ROLE
```

## 2. Привилегии

```
=> \c access_privileges
```

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

```
=> REVOKE ALL ON SCHEMA public FROM public;
```

```
REVOKE
```

```
=> GRANT ALL ON SCHEMA public TO writer;
```

```
GRANT
```

```
=> GRANT USAGE ON SCHEMA public TO reader;
```

```
GRANT
```

## 3. Привилегии по умолчанию

```
=> ALTER DEFAULT PRIVILEGES
FOR ROLE writer
IN SCHEMA public
GRANT SELECT ON TABLES TO reader;
```

```
ALTER DEFAULT PRIVILEGES
```

## 4. Пользователи

Пишущая роль:

```
=> CREATE ROLE w1 LOGIN IN ROLE writer;
```

```
CREATE ROLE
```

Конструкция IN ROLE сразу же добавляет новую роль в указанную. То есть такая команда эквивалентна двум:

```
CREATE ROLE w1 LOGIN;

GRANT writer TO w1;
```

Читающая роль:

```
=> CREATE ROLE r1 LOGIN IN ROLE reader;
```

```
CREATE ROLE
```

## 5. Таблица

```
=> \c - writer
```

```
You are now connected to database "access_privileges" as user "writer".
```

```
=> CREATE TABLE t(n integer);
```

```
CREATE TABLE
```

## 6. Проверка

Роль w1 может вставлять строки:

```
=> \c - w1
```

```
You are now connected to database "access_privileges" as user "w1".
```

```
=> INSERT INTO t VALUES (42);
```

```
INSERT 0 1
```

Роль r1 может читать таблицу:

```
=> \c - r1
```

```
You are now connected to database "access_privileges" as user "r1".
```

```
=> SELECT * FROM t;
```

```
 n
----
 42
(1 row)
```

Но не может изменить:

```
=> UPDATE t SET n = n + 1;
```

```
ERROR:  permission denied for table t
```

Роль w1 может удалить таблицу:

```
=> \c - w1
```

```
You are now connected to database "access_privileges" as user "w1".
```

```
=> DROP TABLE t;
```

```
DROP TABLE
```

В PostgreSQL 14 будет доступна преднастроенная роль pg_read_all_data, автоматически дающая возможность чтения любых данных.

# Практика+

1. Создайте роль alice. Создайте таблицу.

   Выдайте роли alice привилегию на чтение таблицы и привилегию на изменение с правом передачи.

2. Проверьте права доступа к созданной таблице с помощью представления table_privileges информационной схемы. Сравните с выводом команды \dp.

3. Проверьте права доступа к созданной таблице с помощью функции has_table_privileges.



2. Другие представления информационной схемы:
https://postgrespro.ru/docs/postgresql/13/information-schema

3. Другие функции для проверки привилегий:
https://postgrespro.ru/docs/postgresql/13/functions-info#FUNCTIONS-INFO-ACCESS-TABLE

## 1. Роль, таблица и привилегии

```
=> CREATE DATABASE access_privileges;

CREATE DATABASE

=> CREATE USER alice;

CREATE ROLE

=> \c access_privileges

You are now connected to database "access_privileges" as user "student".

=> CREATE TABLE test(id integer);

CREATE TABLE

=> GRANT SELECT ON test TO alice;

GRANT

=> GRANT UPDATE ON test TO alice WITH GRANT OPTION;

GRANT
```

## 2. Привилегии в информационной схеме

```
=> SELECT grantee, grantor, privilege_type, is_grantable
FROM information_schema.table_privileges
WHERE table_name = 'test';

 grantee | grantor | privilege_type | is_grantable
---------+---------+----------------+--------------
 student | student | INSERT         | YES
 student | student | SELECT         | YES
 student | student | UPDATE         | YES
 student | student | DELETE         | YES
 student | student | TRUNCATE       | YES
 student | student | REFERENCES     | YES
 student | student | TRIGGER        | YES
 alice   | student | SELECT         | NO
 alice   | student | UPDATE         | YES
(9 rows)
```

Команда psql показывает ту же информацию, но в другом виде:

```
=> \dp test

                              Access privileges
 Schema | Name | Type  |    Access privileges    | Column privileges | Policies
--------+------+-------+-------------------------+-------------------+----------
 public | test | table | student=arwdDxt/student+|                   |
        |      |       | alice=rw*/student       |                   |
(1 row)
```

## 3. Функции для проверки привилегий

```
=> SELECT has_table_privilege('alice', 'test', 'SELECT') AS has_select,
          has_table_privilege('alice', 'test', 'UPDATE') AS has_update,
          has_table_privilege('alice', 'test', 'DELETE') AS has_delete;

 has_select | has_update | has_delete
------------+------------+------------
 t          | t          | f
(1 row)
```