

## Лабораторна робота 7. Привілеї.

### Теоретичні відомості до лабораторної роботи 7.

Ролі бази даних глобальні для всієї установки кластера бази даних (не для окремої бази даних). Для створення ролі використовується команда SQL:

```
CREATE ROLE ім'я [ [ WITH ] параметр [ ... ] ]
```

Тут *ім'я* відповідає правилам іменування ідентифікаторів SQL: або просте, без особливих знаків, або в подвійних лапках. (На практиці до команди зазвичай додаються інші вказівки, такі як LOGIN. Докладніше про це нижче.) *Параметри* описані нижче.

Для видалення ролі використовується команда:

```
DROP ROLE ім'я;
```

Для отримання списку існуючих ролей:

```
SELECT rolname FROM pg_roles;
```

### Атрибути ролей

Роль бази даних може мати атрибути, що визначають її повноваження та взаємодію із системою автентифікації клієнтів.

#### Право підключення: LOGIN | NOLOGIN

Ці пропозиції визначають, чи дозволяється новій ролі вхід на сервер; тобто, ця роль може стати початковим авторизованим ім'ям при підключенні клієнта. Можна вважати, що роль атрибуту LOGIN відповідає користувачу. Ролі без цього атрибуту бувають корисні для управління доступом у базі даних, але це не користувачі у звичайному розумінні. За замовчуванням мається на увазі варіант NOLOGIN, за винятком виклику CREATE ROLE у вигляді CREATE USER.

Для створення такої ролі можна використовувати будь-який з варіантів:

```
CREATE ROLE ім'я LOGIN;
```

```
CREATE USER ім'я;
```

(Команда CREATE USER еквівалентна CREATE ROLE за виключенням того, що CREATE USER за замовчуванням включає атрибут LOGIN, в той час як CREATE ROLE — ні.) Пропозиція USER вважається застарілим написанням пропозиції ROLE.

#### Статус суперкористувача: SUPERUSER | NOSUPERUSER

Суперкористувач бази даних обходить всі перевірки прав доступу, за виключенням права на вхід в систему. Це небезпечний привілей і він не повинен використовуватися недбало. Найкраще виконувати більшу частину роботи не як суперкористувач. Для створення нового суперкористувача використовується

```
CREATE ROLE ім'я SUPERUSER
```

(Потрібно виконати з під ролі, яка також є суперкористувачем). За замовчуванням мається на увазі NOSUPERUSER.

#### Створення бази даних: CREATEDB | NOCREATEDB

Роль повинна явно мати дозвіл на створення бази даних. За замовчуванням мається на увазі NOCREATEDB. Для створення такої ролі використовується

```
CREATE ROLE ім'я CREATEDB.
```

#### Створення ролі: CREATEROLE | NOCREATEROLE

Ці пропозиції визначають, чи зможе роль створювати нові ролі (тобто виконувати CREATE ROLE). Роль із правом CREATEROLE може також змінювати та видаляти інші ролі. За замовчуванням мається на увазі NOCREATEROLE. Для створення такої ролі використовується

```
CREATE ROLE ім'я CREATEROLE.
```

#### Пароль: PASSWORD 'пароль' | PASSWORD NULL

Задає пароль ролі. Якщо автентифікація за паролем не буде використовуватися, цей параметр можна опустити. При вказівці порожнього значення буде встановлено пароль

NULL, що не дозволить даному користувачеві пройти автентифікацію за паролем. За бажанням пароль NULL можна встановити явно, вказавши PASSWORD NULL. Пароль вказується під час створення ролі:

```
CREATE ROLE ім'я PASSWORD 'рядок'.
```

#### ADMIN ім'я\_ролі

Пропозиція ADMIN подібна до ROLE, але перелічені в ньому ролі включаються в нову роль з атрибутом WITH ADMIN OPTION, що дає їм право включати в цю роль інші ролі.

```
INHERIT | NOINHERIT
```

Ці пропозиції визначають, чи роль «успадковуватиме» права ролей, членом яких вона є. Без INHERIT членство в іншій ролі дозволяє лише виконати SET ROLE і перейти на цю роль; правами, призначеними іншій ролі, можна буде користуватися лише після цього. За умовчанням мається на увазі INHERIT.

```
VALID UNTIL 'дата_час'
```

Пропозиція VALID UNTIL встановлює дату і час, після якого пароль ролі перестане діяти. Якщо ця пропозиція відсутня, термін дії пароля буде необмеженим.

```
IN ROLE ім'я_ролі
```

У пропозиції IN ROLE перераховуються одна чи кілька існуючих ролей, в які буде негайно включена нова роль. (Зверніть увагу, що додати нову роль з правами адміністратора таким чином не можна; для цього потрібно окремо виконати команду GRANT.)

```
ROLE ім'я_ролі
```

У пропозиції ROLE перераховуються одна або декілька ролей, які автоматично стають членами створюваної ролі. (По суті таким чином нова роль стає "групою".)

Атрибути ролей можуть бути змінені після створення командою ALTER ROLE.

### Членство в ролі

Часто буває зручно згрупувати користувачів для спрощення управління правами: права можна видавати для всієї групи та у всієї групи забирати. У PostgreSQL при цьому створюється роль, що представляє групу, а після членство у цій групі видається ролям індивідуальних користувачів.

Для налаштування групової ролі спочатку необхідно створити саму роль:

```
CREATE ROLE ім'я;
```

Зазвичай групова роль не має атрибуту LOGIN, хоча за бажання його можна встановити.

Після того, як групова роль створена, до неї можна додавати або видаляти членів, використовуючи команди GRANT і REVOKE:

```
GRANT групова_роль TO роль1, ... ;
```

```
REVOKE групова_роль FROM роль1, ... ;
```

Членом ролі може бути й інша групова роль (бо насправді немає жодних відмінностей між груповими і груповими ролями). При цьому база даних не допускає замикання членства у колі.

### Видалення ролей

Оскільки ролі можуть володіти об'єктами баз даних і мати права доступу до інших об'єктів, видалення ролі не зводиться до негайної дії DROP ROLE. Спочатку мають бути видалені та передані іншим власникам всі об'єкти, що належать ролі; також мають бути відкликані всі права, надані ролі.

Володіння об'єктами можна передавати в індивідуальному порядку, застосовуючи команду ALTER, наприклад:

```
ALTER TABLE bobs_table OWNER TO alice;
```

Крім того, для перепризначення будь-якої іншої ролі володіння відразу всіма об'єктами, які належать ролі, яку потрібно видалити, можна застосувати команду

REASSIGN OWNED. Оскільки REASSIGN OWNED не може звертатися до об'єктів в інших базах даних, її необхідно виконати в кожній базі, яка містить об'єкти, що належать цій ролі.

Після того, як всі цінні об'єкти будуть передані новим власникам, всі об'єкти, що залишаються, що належать ролі, яку потрібно видалити, можуть бути видалені за допомогою команди DROP OWNED. І ця команда не може звертатися до об'єктів в інших базах даних, тому її потрібно запускати в кожній базі, яка містить об'єкти, що належать ролі.

DROP OWNED також видаляє всі права, які дані цільовій ролі для об'єктів, що не належать їй. Так як REASSIGN OWNED такі об'єкти не торкається, зазвичай необхідно запустити і REASSIGN OWNED, і DROP OWNED (у цьому порядку!), щоб повністю ліквідувати залежність ролі, що видаляється.

З урахуванням цього, загальний рецепт видалення ролі, яка володіла об'єктами, коротко такий:

```
REASSIGN OWNED BY doomed_role TO successor_role;
DROP OWNED BY doomed_role;
-- повторити попередні команди для кожної бази у кластері
DROP ROLE doomed_role;
```

Якщо не всі об'єкти потрібно передати одному новому власнику, краще спочатку вручну відпрацювати винятки, а на завершення виконати наведені вище дії.

При спробі виконати DROP ROLE для ролі, в якій зберігаються залежні об'єкти, буде видано повідомлення, які говорять, які об'єкти потрібно передати іншому власнику чи видалити.

Встановити ідентифікатор поточного користувача в рамках сеансу можна за допомогою команди

**SET ROLE ім'я\_ролі.**

Ім'я ролі може бути записане у вигляді ідентифікатора або строкової константи. Після SET ROLE права доступу для команд SQL перевіряються так, якби сеанс спочатку був встановлений з цим ім'ям ролі.

Вказуючи певне ім'я\_ролі, поточний користувач повинен бути членом цієї ролі. (Якщо користувач сеансу є суперкористувачем, він може вибрати будь-яку роль.)

Форми NONE та RESET скидають ідентифікатор поточного користувача, тому активним стає ідентифікатор користувача сеансу. Ці форми можуть виконувати будь-які користувачі

Приклади:

```
SELECT SESSION_USER, CURRENT_USER;
session_user | current_user
-----+-----
peter       | peter
SET ROLE 'paul';
SELECT SESSION_USER, CURRENT_USER;
session_user | current_user
-----+-----
peter       | paul
```

### **Привілеї**

**Привілеї** – це те, що визначає, чи може вказаний користувач виконати дану команду. Є кілька типів привілеїв, що відповідають декільком типам операцій. Привілеї даються і скасовуються двома командами SQL: GRANT (ДОПУСК) і REVOKE (СКАСУВАННЯ).

Кожен користувач в базі даних має набір привілеїв. Ці привілеї можуть змінюватися з часом – нові додаватися, старі видалятися.

Привілеї надаються певному користувачеві у зазначеній таблиці, або базовій таблиці, або представленні. Користувач, який створив таблицю (будь-якого виду), є власником цієї

таблиці, тобто має всі привілеї в цій таблиці і може передавати привілеї іншим користувачам в цій таблиці.

Привілеї, які можна назначити користувачу представлені в таблиці.

Таблиця – Привілеї, які можна назначити користувачу

Привілеї	Надані можливості
SELECT	Користувач з цим привілеєм може виконувати запити в таблиці.
INSERT	Користувач з цим привілеєм може виконувати команду INSERT в таблиці.
UPDATE	Користувач з цим привілеєм може виконувати команду UPDATE в таблиці. Ви можете обмежити цей привілей для певних стовпців таблиці.
DELETE	Користувач з цим привілеєм може виконувати команду DELETE в таблиці.
REFERENCES	Користувач з цим привілеєм може визначити зовнішній ключ, який використовує один або більше стовпців цієї таблиці, як батьківський ключ. Ви можете обмежити цей привілей для певних стовпців.

Крім того, можливі такі нестандартні привілеї об'єкта, як INDEX, який дає право створювати індекс в таблиці, SYNONYM – дає право створювати синонім для об'єкта, і ALTER – дає право виконувати команду ALTER TABLE в таблиці.

Команда GRANT *надає привілеї*:

GRANT привілея(і) ON таблиця TO користувач(і);

Списки привілеїв або користувачів, відокремлюваних комами, є абсолютно прийнятними, необов'язково застосовувати одиночні привілеї окремому користувачеві командою GRANT.

Всі привілеї об'єкта використовують один і той же синтаксис, крім команд UPDATE і REFERENCES в яких можна вказувати *імена стовпців*:

GRANT UPDATE (атрибут1, атрибут2) ON таблиця TO користувач(і);

Ця команда дозволить зазначеному користувачу змінювати значення тільки в атрибут1, атрибут2 зазначеної таблиці.

REFERENCES використовується аналогічно. Надання привілею REFERENCES іншому користувачеві, дозволяє йому створювати зовнішні ключі, що посилаються на стовпці таблиці як на батьківські ключі.

SQL підтримує два аргументи для команди GRANT, які мають спеціальне значення: **ALL PRIVILEGES** (всі привілеї) або просто ALL і **PUBLIC** (загальні).

ALL використовується замість імен привілеїв в команді GRANT, щоб віддати все привілеї в таблиці.

GRANT ALL ON таблиця TO користувач(і);

PUBLIC – більше схожий на тип аргументу «захопити всі (catch-all)», ніж на призначену для користувача привілею. У такому випадку всі користувачі автоматично отримують задану привілею. Найбільш часто, це застосовується для привілеї SELECT в певних базових таблицях або представленнях, які необхідно зробити доступними для будь-якого користувача.

GRANT привілея(і) ON таблиця TO PUBLIC;

PUBLIC не обмежений в його передачі тільки поточним користувачам. Будь-який новий користувач, який додається до системи, автоматично отримає всі привілеї, призначені раніше всім, так що, якщо необхідно буде обмежити доступ до таблиці всім,

зараз або в майбутньому, найкраще надати інші привілеї, крім SELECT, для індивідуальних користувачів.

Іноді, власнику таблиці потрібно, щоб інші користувачі могли мати такі ж привілеї в його таблиці. Зазвичай це робиться в системах, де один або більше людей створюють кілька (або всі) базові таблиці в базі даних, а потім *передають відповідальність* за них тим, хто буде фактично з ними працювати. SQL дозволяє робити це за допомогою WITH GRANT OPTION

GRANT привілея(ї) ON таблиця TO користувач(і) WITH GRANT OPTION;

Користувач за допомогою GRANT OPTION в особливий привілеї для даної таблиці, може, в свою чергу, надати цей привілеї до тієї ж таблиці, з або без GRANT OPTION, будь-якого іншого користувача. Це не змінює приналежності самої таблиці, як і раніше таблиця належать її власнику. Тому користувачі, які отримали права, повинні встановлювати префікс у вигляді імені власника, коли посилаються на ці таблиці

GRANT привілея(ї) ON власник.таблиця TO користувач(і);

Можна зробити дії привілеїв більш точними, використовуючи представлення. Кожного разу, коли передаються привілеї в базовій таблиці користувачеві, вони автоматично поширюються на всі рядки, а при використанні можливих винятків UPDATE і REFERENCES, на всі стовпці таблиці. Створюючи представлення, яке посилається на основну таблицю, і потім переносить привілеї на представлення, а не на таблицю, можна обмежувати ці привілеї будь-якими виразами в запиті, що містяться в представленні. Це значно покращує базисні можливості команди GRANT.

Щоб створювати представлення, необхідно мати привілеї SELECT у всіх таблицях, на які є посилання в представленні. Якщо представлення модифікуються, будь-які привілеї, INSERT, UPDATE або DELETE, які є для базової таблиці, будуть автоматично передаватися представленням. Якщо відсутній привілеї на модифікацію в базових таблицях, він буде відсутній і в представленнях, які створили, навіть якщо самі ці представлення модифікуються. Так як зовнішні ключі не використовуються в представленнях, привілеї REFERENCES ніколи не використовуються при створенні представлень.

**Видалення привілеїв** виконується командою REVOKE. Синтаксис команди REVOKE схожий на GRANT, але має протилежне значення

REVOKE привілея(ї) ON таблиця FROM користувач(і);

Списки привілеїв або користувачів, відокремлюваних комами, є абсолютно прийнятними, аналогічно команді GRANT.

Привілеї скасовуються користувачем, який їх надав, і скасування буде каскадуватися, тобто воно буде автоматично поширюватися на всіх користувачів, які отримали від нього цей привілеї.

Для **створення користувача** адміністратор бази даних надає користувачу привілею CONNECT:

GRANT CONNECT TO ім'я\_користувача;

Для **видалення користувача** необхідно використовувати для REVOKE привілеї CONNECT. Якщо зробити спробу видалити привілеї CONNECT користувача, який має створені ним таблиці, команда буде відхилена, тому що її дія залишить таблицю без власника, а це не дозволяється. Перш ніж видалити його привілеї CONNECT, необхідно видалити всі таблиці, створені цим користувачем. Якщо ці таблиці не порожні, то можна передати їх дані в інші таблиці за допомогою команди INSERT, яка використовує запит.

### **Перевірка встановлених привілеїв**

Для перевірки наданих привілеїв можна встановити ідентифікатор поточного користувача в рамках сеансу за допомогою команди

### SET ROLE ім'я\_ролі

та виконати дії, що дозволені та недозволені певному користувачу.








Наприклад, користувачу us1 надано привілеї на перегляд та зміну даних тільки в таблиці Дисципліни. Всі інша привілеї не надано. Перевіряємо:

```
SET ROLE 'us1';
```

```
SELECT SESSION_USER, CURRENT_USER;
```

Результат	План выполнения	Сообщения	Notifications
session_user name	current_user name		
1 postgres	us1		

```
SELECT * from Дисциплина where id=3;
```

Результат	План выполнения		Сообщения	Notifications		
 id [PK] integer	 шифрОП character (20)	 названиеДисциплины character varying	 колКредитовВсего double precision	 кчЗаУчебнПланом integer	 кчФактическиВыделено integer	
1	3	ЗП 0.02	Вища математика	11	480	480

```
delete from Дисциплина;
```

Результат	План выполнения	Сообщения	Notifications
ERROR: ОШИБКА: нет доступа к таблице Дисциплина			
SQL-состояние: 42501			

Або створити в PgAdmin сервер з ім'ям та паролем створеного користувача:

Правою кнопкою на PostgreSQL XX -> Створити -> Сервер

На вкладці Загальні задаємо будь-яке ім'я серверу.

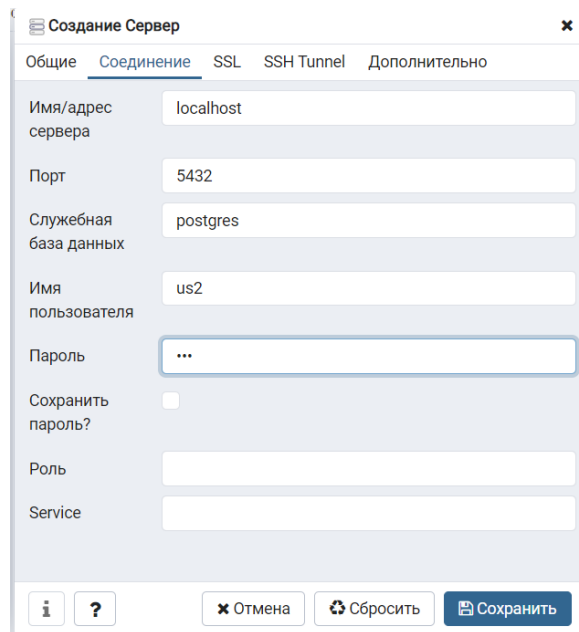
На вкладці З'єднання задаємо:

Ім'я/адреса сервера – localhost (якщо працюєте з локальною версією)

Ім'я користувача – ім'я створеного вами користувача

Пароль – пароль створеного вами користувача

Зберегти



З'явиться сервер, на якому ви можете спробувати виконати дії, що дозволені та недозволені певному користувачу.

#### Обозреватель

- ▼ Servers (2)
  - > PostgreSQL 12
  - > testuser

#### Завдання до лабораторної роботи 7

1. Створіть трьох користувачів.
2. Надайте право першому користувачу на зміну декількох стовпців будь-якої таблиці.
3. Надайте право всім користувачам системи на перегляд будь-якої таблиці
4. Надайте право другому користувачу вставляти або модифікувати значення таблиці з правом передавати іншим користувачам вказані права.
5. Надайте третьому користувачу права адміністратора (всі права на всі таблиці).
6. Зніміть привілей INSERT для третього користувача для будь-якої таблиці.
7. Надайте право першому користувачу на доступ тільки до певних рядків будь-якої таблиці.
8. Розподіліть інші привілеї на свій розсуд.

#### Структура звіту до лабораторної роботи

Для кожного з запитів:

- 1) постановка задачі, що вирішується;
- 2) SQL-код рішення;
- 3) скриншот отриманого результату.