Konta, role i uprawnienia: GRANT, REVOKE, zasada najmniejszych uprawnień

> Konto aplikacji, które potrafi wykonać `DROP TABLE`, nie jest wygodne — jest niewybuchłą bombą czekającą na literówkę w skrypcie migracyjnym albo na przejęty klucz dostępu.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Fiszka

ZMIENIA SYSTEM

Utworzenie konta i roli: PostgreSQL

```
CREATE ROLE NazwaUsera LOGIN PASSWORD 'HasloDoUstawienia';
CREATE ROLE NazwaSchematu_odczyt;
```

Tworzymy podmiot z prawem logowania (`LOGIN` odróżnia konto od roli czysto grupującej, która sama nie loguje się nigdy) oraz osobną rolę bez prawa logowania, służącą wyłącznie jako grupa uprawnień do nadania innym kontom.

Przykład:

```
produkcja=# CREATE ROLE nazwa_usera LOGIN PASSWORD 'HasloDoUstawienia';
CREATE ROLE
```

ZMIENIA SYSTEM

Nadanie uprawnień do tabeli: GRANT

```
GRANT SELECT, INSERT, UPDATE ON NazwaSchematu.NazwaTabeli TO NazwaUsera;
```

Nadajemy jednemu kontu trzy konkretne uprawnienia do jednej tabeli — bez `DELETE` i bez `DROP`, jeśli konto nie ma ich potrzebować.

Przykład:

```
produkcja=# GRANT SELECT, INSERT, UPDATE ON sklep.zamowienia TO nazwa_usera;
GRANT
```

ZMIENIA SYSTEM

Nadanie uprawnień do całego schematu: GRANT

```
GRANT USAGE ON SCHEMA NazwaSchematu TO NazwaUsera;
GRANT SELECT ON ALL TABLES IN SCHEMA NazwaSchematu TO NazwaUsera;
```

`USAGE` na schemacie jest warunkiem koniecznym, żeby podmiot w ogóle widział cokolwiek wewnątrz niego — bez niego nawet uprawnienie do konkretnej tabeli nie wystarcza, bo konto nie ma prawa „wejść” do schematu, w którym ta tabela leży. Drugie polecenie nadaje odczyt do wszystkich tabel istniejących w schemacie w chwili wykonania — nowe tabele utworzone później tego uprawnienia automatycznie nie dziedziczą.

Przykład:

```
produkcja=# GRANT USAGE ON SCHEMA sklep TO nazwa_usera;
GRANT
produkcja=# GRANT SELECT ON ALL TABLES IN SCHEMA sklep TO nazwa_usera;
GRANT
```

ZMIENIA SYSTEM

Uprawnienia domyślne dla nowych obiektów: ALTER DEFAULT PRIVILEGES

```
ALTER DEFAULT PRIVILEGES IN SCHEMA NazwaSchematu
GRANT SELECT ON TABLES TO NazwaUsera;
```

Ustawiamy regułę, która automatycznie nada odczyt każdej nowej tabeli utworzonej w tym schemacie od tego momentu w przód — rozwiązuje dokładnie lukę z poprzedniej fiszki, gdzie `GRANT ... ON ALL TABLES` obejmował tylko obiekty istniejące w chwili wykonania.

Przykład:

```
produkcja=# ALTER DEFAULT PRIVILEGES IN SCHEMA sklep GRANT SELECT ON TABLES TO
nazwa_usera;
ALTER DEFAULT PRIVILEGES
```

□□ ZMIENIA SYSTEM

Odebranie uprawnień: REVOKE

```
REVOKE INSERT, UPDATE ON NazwaSchematu.NazwaTabeli FROM NazwaUsera;
```

Odbieramy dwa z wcześniej nadanych uprawnień, zostawiając samo `SELECT` — przydatne, gdy konto miało szersze prawa niż faktycznie potrzebuje, wykryte dopiero po fakcie.

Przykład:

```
produkcja=# REVOKE INSERT, UPDATE ON sklep.zamowienia FROM nazwa_usera;
REVOKE
```

□□ ZMIENIA SYSTEM

Nadanie roli innemu kontu: GRANT roli

```
GRANT NazwaSchematu_odczyt TO NazwaUsera;
```

Dołączamy konto do roli grupującej uprawnienia — od tego momentu konto dziedziczy wszystko, co ma ta rola, bez potrzeby powtarzania pojedynczych `GRANT`ów.

Przykład:

```
produkcja=# GRANT sklep_odczyt TO nazwa_usera;
GRANT ROLE
```

🔍 TYLKO ODCZYT

Sprawdzenie uprawnień użytkownika: PostgreSQL

```
SELECT grantee, table_schema, table_name, privilege_type
FROM information_schema.role_table_grants
WHERE grantee = 'nazwa_usera';
```

Przykład:

```
   grantee    | table_schema | table_name |  privilege_type
--------------+--------------+------------+-----------------
 nazwa_usera  | sklep        | zamowienia | SELECT
(1 row)
```

⚠️ ZMIENIA SYSTEM

Utworzenie użytkownika i nadanie uprawnień: Oracle

```
CREATE USER NazwaUsera IDENTIFIED BY HasloDoUstawienia;
GRANT CONNECT TO NazwaUsera;
GRANT SELECT, INSERT, UPDATE ON NazwaSchematu.NazwaTabeli TO NazwaUsera;
```

`CREATE USER` tworzy konto, ale samo w sobie nie daje mu prawa nawet się połączyć — dopiero rola `CONNECT` (albo odpowiadające jej uprawnienie systemowe `CREATE SESSION`) na to pozwala.

Przykład:

```
SQL> CREATE USER nazwa_usera IDENTIFIED BY HasloDoUstawienia;

User created.

SQL> GRANT CONNECT TO nazwa_usera;

Grant succeeded.
```

🔍 TYLKO ODCZYT

Sprawdzenie uprawnień użytkownika: Oracle

```
SELECT grantee, table_name, privilege
FROM dba_tab_privs
WHERE grantee = 'NAZWA_USERA';
```

Przykład:

```
GRANTEE       TABLE_NAME      PRIVILEGE
------------- --------------- ----------
NAZWA_USERA   ZAMOWIENIA      SELECT
NAZWA_USERA   ZAMOWIENIA      INSERT
```

□□ ZMIENIA SYSTEM

Utworzenie loginu i użytkownika: SQL Server

```
CREATE LOGIN NazwaUsera WITH PASSWORD = 'HasloDoUstawienia';
USE NazwaBazy;
CREATE USER NazwaUsera FOR LOGIN NazwaUsera;
GRANT SELECT, INSERT, UPDATE ON NazwaSchematu.NazwaTabeli TO NazwaUsera;
```

Tworzymy login na poziomie instancji, potem użytkownika powiązanego z tym loginem wewnątrz konkretnej bazy — dopiero temu użytkownikowi nadajemy uprawnienia do obiektów tej bazy.

Przykład:

```
Command(s) completed successfully.
```

□□ **SQL** = Structured Query Language: język zapytań do relacyjnych baz danych

□□ TYLKO ODCZYT

Sprawdzenie uprawnień użytkownika: SQL Server

```
SELECT dp.permission_name, o.name AS obiekt
FROM sys.database_permissions dp
JOIN sys.objects o ON dp.major_id = o.object_id
JOIN sys.database_principals pr ON dp.grantee_principal_id = pr.principal_id
WHERE pr.name = 'NazwaUsera';
```

Przykład:

```
permission_name   obiekt
----------------- --------------
SELECT            Zamowienia
INSERT            Zamowienia
UPDATE            Zamowienia
```

SQL = Structured Query Language: język zapytań do relacyjnych baz danych ·**join** = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

ZMIENIA SYSTEM

Utworzenie użytkownika i nadanie uprawnień: MySQL

```
CREATE USER 'NazwaUsera'@'%' IDENTIFIED BY 'HasloDoUstawienia';
GRANT SELECT, INSERT, UPDATE ON NazwaBazy.NazwaTabeli TO 'NazwaUsera'@'%';
```

Znak `%` w nazwie konta oznacza dowolny host, z którego wolno się łączyć — w środowisku produkcyjnym zamieniamy go na konkretny adres albo podsieć (`'NazwaUsera'@'10.20.10.%'`), zamiast pozwalać na logowanie z dowolnego miejsca w sieci.

Przykład:

```
Query OK, 0 rows affected (0.02 sec)
Query OK, 0 rows affected (0.01 sec)
```

TYLKO ODCZYT

Sprawdzenie uprawnień użytkownika: MySQL

```
SHOW GRANTS FOR 'NazwaUsera'@'%';
```

Przykład:

```
+-------------------------------------------------------------------+
| Grants for NazwaUsera@%                                           |
+-------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `NazwaUsera`@`%`                            |
```

```
| GRANT SELECT, INSERT, UPDATE ON `sklep`.`zamowienia` TO `NazwaUsera`@`%` |
+--------------------------------------------------------------------------+
```

show = FortiOS: wypisuje konfigurację, w podstawowej postaci tylko ustawienia różniące się od domyślnych, a z wariantem full-configuration także wartości domyślne

Po co to istnieje

Pojęcie: podmiot (login, użytkownik, rola). Byt, któremu silnik bazy danych może przypisać prawo do wykonania jakiejś operacji — osoba logująca się interaktywnie, aplikacja łącząca się z bazy poświadczeniami zapisanymi w konfiguracji, albo zestaw uprawnień nadawany zbiorczo innym podmiotom pod jedną nazwą. Rozróżnienie między tymi trzema słowami różni się między silnikami, ale sama koncepcja — „ktoś albo coś, co dostaje prawa” — jest wspólna dla wszystkich czterech.

Pojęcie: uprawnienie (privilege, grant). Zgoda na wykonanie konkretnego typu operacji na konkretnym obiekcie bazy — odczyt z tabeli, zapis do niej, wykonanie procedury, utworzenie nowej tabeli w schemacie. Uprawnienie zawsze wiąże trzy elementy: kto je ma, co wolno mu zrobić, i na czym — samo słowo „uprawnienie” bez wskazania obiektu, którego dotyczy, nic nie znaczy.

Login kontra użytkownik bazy danych. W SQL Server i w Oracle rozróżnienie między loginem a użytkownikiem bazy nie jest kosmetyczne. **Pojęcie: login (SQL Server).** Podmiot uwierzytelniający się na poziomie całej instancji serwera — sam login nie ma jeszcze żadnych praw wewnątrz konkretnej bazy danych, dopóki nie zostanie z nim powiązany użytkownik wewnątrz tej bazy. **Pojęcie: użytkownik bazy danych (database user, SQL Server).** Podmiot istniejący wewnątrz jednej konkretnej bazy, powiązany z loginem na poziomie instancji, i to jemu, nie loginowi wprost, nadaje się uprawnienia do obiektów tej bazy. Ten sam login może być powiązany z użytkownikami o różnych nazwach i różnych uprawnieniach w różnych bazach tej samej instancji. Oracle rozróżnia analogicznie konto uwierzytelniające na poziomie bazy (`CREATE USER`) od jego uprawnień w konkretnym schemacie, choć tam schemat i użytkownik są ze sobą utożsamione ściślej niż w SQL Server. PostgreSQL i MySQL nie mają tego rozróżnienia w tej samej postaci — rola w PostgreSQL i użytkownik w MySQL łączą obie funkcje w jednym bycie.

Jak to działa

Poziomy uprawnień. Uprawnienia nadaje się na kilku poziomach hierarchii obiektów bazy, od najszerszego do najwęższego, a uprawnienie nadane wyżej zwykle obejmuje wszystko, co jest niżej.

Poziom	Co obejmuje	Przykład
Instancja	całą instalację silnika, wszystkie bazy naraz	prawo tworzenia nowych baz, zarządzania kontami
Baza danych	jedną konkretną bazę	prawo połączenia się z bazą, tworzenia w niej schematów
Schemat	grupę obiektów wewnątrz bazy	prawo tworzenia tabel w danym schemacie
Obiekt (tabela, widok, procedura)	jeden konkretny obiekt	`SELECT`, `INSERT`, `UPDATE`, `EXECUTE` na tej tabeli czy procedurze
Kolumna	pojedynczą kolumnę wskazanej tabeli	odczyt wszystkich kolumn poza jedną zawierającą dane wrażliwe

Konta techniczne

Trzy typy kont technicznych pojawiają się na niemal każdej bazie produkcyjnej, i każde z nich potrzebuje innego, ściśle dobranego zestawu uprawnień.

Konto aplikacji. Potrzebuje `SELECT`, `INSERT`, `UPDATE` (a czasem `DELETE`) wyłącznie na tabelach, z których aplikacja faktycznie korzysta — nigdy praw do zmiany struktury (`CREATE`, `ALTER`, `DROP`), bo aplikacja w normalnej pracy nie tworzy ani nie kasuje tabel. Migracje schematu wykonuje osobne konto administracyjne, używane ręcznie albo z procesu wdrożeniowego, nie konto, którym łączy się produkcyjny kod aplikacji dwadzieścia cztery godziny na dobę.

Konto backupu. Potrzebuje prawa odczytu wszystkich obiektów bazy oraz, w zależności od silnika, specjalnej roli systemowej dedykowanej backupowi (jak `pg_read_all_data` w PostgreSQL czy uprawnienie `BACKUP DATABASE` w SQL Server) — nie potrzebuje prawa zapisu do jakiejkolwiek tabeli z danymi produkcyjnymi, bo backup wyłącznie czyta.

Konto monitorujące. Potrzebuje odczytu widoków i tabel systemowych opisujących stan silnika — aktywne sesje, statystyki, blokady — a nie potrzebuje żadnego dostępu do samych danych biznesowych. Rozwinięcie tego konta znajduje się na stronie Co monitorujemy w bazie danych.

Zasada najmniejszych uprawnień. Pojęcie: zasada najmniejszych uprawnień (principle of least privilege). Reguła nadawania każdemu podmiotowi wyłącznie tych uprawnień, które są mu rzeczywiście potrzebne do wykonywania jego roli — nic ponad to, „na wszelki wypadek” czy „żeby nie musieć wracać do tego później”. Konto aplikacji sklepu internetowego nie potrzebuje `DROP TABLE`, bo aplikacja w swojej normalnej pracy nigdy nie kasuje tabel — nadanie tego uprawnienia nie przyspiesza niczego i nie ułatwia niczego w typowej pracy, za to w razie błędu w kodzie, literówki w zapytaniu generowanym dynamicznie, albo przejęcia poświadczeń aplikacji przez atakującego, otwiera drogę do zniszczenia całej struktury danych, zamiast ograniczyć szkodę do pojedynczych wierszy w jednej tabeli.

Częste błędy

Konta domyślne i puste hasła po instalacji. Świeżo zainstalowany silnik bazy danych bywa dostarczany z kontem administracyjnym o znanej, domyślnej nazwie i albo bez hasła, albo z hasłem publicznie znanym z dokumentacji instalacyjnej. Pozostawienie takiego konta aktywnego po instalacji, bez zmiany hasła albo bez jego wyłączenia, jest jednym z najczęściej skanowanych i wykorzystywanych automatycznie wektorów ataku na bazy danych wystawione choćby przypadkiem do sieci szerszej niż zamierzano.

Nadmiarowe uprawnienia „na wszelki wypadek”. Opisane w sekcji o zasadzie najmniejszych uprawnień — nadanie kontu aplikacji roli administratora bazy, bo „tak jest szybciej, nie trzeba się zastanawiać, czego dokładnie potrzeba”, zamienia każdą podatność w aplikacji w podatność całej bazy danych.

Wspólne konto dla wielu aplikacji albo wielu administratorów. Jedno konto techniczne, którego hasło zna kilka zespołów albo kilka niezależnych systemów, uniemożliwia jednoznaczne przypisanie, kto (albo co) wykonał daną operację — przy incydencie bezpieczeństwa czy błędnie wykonanej zmianie logi silnika pokazują tylko nazwę konta, nie faktycznego sprawcę.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — dobór uprawnień dla konta aplikacji

Nowa aplikacja ma czytać i zapisywać zamówienia w tabeli `zamowienia` schematu `sklep`, nigdy nie tworzy ani nie kasuje tabel, i nie powinna widzieć żadnego innego schematu na tej samej instancji. Jakie polecenia w PostgreSQL nadadzą dokładnie takie uprawnienia, bez niczego ponad to?

Odpowiedź

```
CREATE ROLE aplikacja_sklep LOGIN PASSWORD 'HasloDoUstawienia';
GRANT USAGE ON SCHEMA sklep TO aplikacja_sklep;
GRANT SELECT, INSERT, UPDATE ON sklep.zamowienia TO aplikacja_sklep;
```

Konto dostaje prawo logowania, prawo „wejścia” do schematu `sklep` przez `USAGE`, i dokładnie trzy uprawnienia do jednej tabeli — bez `DELETE`, bez `CREATE`, bez dostępu do jakiegokolwiek innego schematu, bo nic mu tego nie nadaje.

Pytania kontrolne

1. Dlaczego samo `GRANT SELECT ON NazwaSchematu.NazwaTabeli TO NazwaUsera` w PostgreSQL może nie wystarczyć, żeby użytkownik faktycznie odczytał tę tabelę?

Odpowiedź

Uprawnienie do konkretnego obiektu nie zastępuje uprawnienia `USAGE` na schemacie, w którym ten obiekt leży — bez `USAGE` podmiot nie ma prawa nawet „zobaczyć” zawartości schematu, więc zapytanie do tabeli kończy się błędem braku uprawnień, mimo że `SELECT` na samą tabelę został nadany.

2. Czym różni się login od użytkownika bazy danych w SQL Server?

Odpowiedź

Login uwierzytelnia się na poziomie całej instancji serwera i sam z siebie nie ma żadnych uprawnień wewnątrz konkretnej bazy. Użytkownik bazy danych istnieje wewnątrz jednej konkretnej bazy, jest powiązany z loginem, i to jemu nadaje się uprawnienia do obiektów tej bazy — ten sam login może mieć różnych, osobno uprawnionych użytkowników w różnych bazach tej samej instancji.

Powiązane: Co monitorujemy w bazie danych · Backup pełny, różnicowy, przyrostowy, logi · Katalog systemowy: skąd baza wie, co w sobie ma

Źródła: dokumentacja PostgreSQL, rozdział „Database Roles and Privileges” · Oracle Database Security Guide, rozdział „Configuring Privilege and Role Authorization” · Microsoft Docs, „Principals” i „Permissions (Database Engine)” · dokumentacja MySQL, rozdział „Access Control and Account Management”

Revision #44
Created 2026-09-24 23:07:47 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:57 UTC by Tomasz Konieczny