Użytkownicy, role i uprawnienia

Poziom: podstawy **Dotyczy: PostgreSQL 9.6–17** **Czas: ~25 min**

Konto Zabbixa do monitoringu, konto do backupu przez replikację i konto aplikacji do jednego schematu — trzy różne role, trzy różne zestawy uprawnień, ten sam mechanizm `CREATE ROLE` i `GRANT` pod spodem.

Fiszka

ZMIENIA SYSTEM

Konfiguracja usera zbx_monitor dla Zabbixa: PostgreSQL 10+

```
CREATE USER zbx_monitor WITH PASSWORD 'HasloUzytkownika' INHERIT;
```

Tworzymy dedykowane konto monitorujące dla Zabbixa — samo `CREATE USER` jeszcze nie daje mu żadnych uprawnień poza domyślnymi.

Przykład:

```
CREATE ROLE
```

Nadajemy rolę wbudowaną `pg_monitor`, dostępną od wersji 10 wzwyż:

```
GRANT pg_monitor TO zbx_monitor;
```

Przykład:

```
GRANT ROLE
```

Doprecyzowujemy dodatkowo odczyt widoku statystyk bazodanowych, jeśli konkretny szablon monitoringu tego wymaga:

```
GRANT SELECT ON pg_stat_database TO zbx_monitor;
```

Przykład:

```
GRANT
```

Na koniec dopisujemy w `pg_hba.conf`, opisanym na stronie postgresql.conf i pg_hba.conf, regułę pozwalającą temu kontu się zalogować z hosta, na którym stoi serwer Zabbixa:

```
host    all             zbx_monitor     10.20.10.0/24   scram-sha-256
```

Po dopisaniu reguły przeładowujemy konfigurację:

```
SELECT pg_reload_conf();
```

◫◫ ZMIENIA SYSTEM

Konfiguracja usera zbx_monitor dla Zabbixa: PostgreSQL starszy niż 10

```
CREATE USER zbx_monitor WITH PASSWORD 'HasloUzytkownika' INHERIT;
```

Na wersjach sprzed 10, gdzie rola `pg_monitor` jeszcze nie istnieje, jedynym sposobem na dostęp do pełnego zakresu widoków statystyk jest nadanie kontu monitorującemu pełnych uprawnień superusera:

```
ALTER USER zbx_monitor WITH SUPERUSER;
```

Przykład:

```
ALTER ROLE
```

Uważaj: to rozwiązanie zostaje na stałe kontem o pełnych uprawnieniach administracyjnych — przy migracji instancji na wersję 10 lub nowszą warto od razu zamienić je na węższy zestaw z `pg_monitor`, zamiast przenosić `SUPERUSER` dalej bez potrzeby.

◫◫ ZMIENIA SYSTEM

Dodanie usera backupu z prawem replikacji

```
CREATE USER base_backup_user WITH REPLICATION LOGIN;
```

Tworzymy dedykowane konto do backupu fizycznego, wykorzystywane przez narzędzia takie jak `pg_basebackup` albo pgBackRest: podstawowe komendy — atrybut `REPLICATION` daje prawo nawiązania strumienia replikacyjnego bez nadawania jakichkolwiek uprawnień do samych danych w tabelach.

Przykład:

```
CREATE ROLE
```

Dopisujemy w `pg_hba.conf` regułę zezwalającą temu kontu na połączenia replikacyjne:

```
host    replication     base_backup_user    127.0.0.1/32    scram-sha-256
```

Sprawdzamy, że rola rzeczywiście ma atrybut replikacji:

```
\du+ base_backup_user
```

Przykład:

```
                                   List of roles
    Role name     |            Attributes             | Member of | Description
------------------+-----------------------------------+-----------+-------------
 base_backup_user  | Replication                       | {}        |
(1 row)
```

□□ ZMIENIA SYSTEM

Nadanie uprawnień do schematu i tabel: GRANT

```
GRANT USAGE ON SCHEMA NazwaSchematu TO NazwaUsera;
```

Samo `USAGE` na schemacie nie daje dostępu do żadnej tabeli w środku — pozwala jedynie „wejść" do schematu, czyli odwoływać się do jego obiektów po nazwie. Bez tego uprawnienia rola nie widzi tabel schematu nawet wtedy, gdy ma na nie nadane `SELECT`.

Przykład:

```
GRANT
```

Nadanie odczytu i zapisu na konkretnej tabeli:

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

Przykład:

```
GRANT
```

Nadanie tego samego zestawu na wszystkich tabelach istniejących już w schemacie, jednym poleceniem:

```
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA NazwaSchematu TO NazwaUsera;
```

Przykład:

```
GRANT
```

◫◫ ZMIENIA SYSTEM

Uprawnienia domyślne dla przyszłych obiektów: ALTER DEFAULT PRIVILEGES

> **Pojęcie: ALTER DEFAULT PRIVILEGES.** `GRANT ... ON ALL TABLES` z poprzedniego przykładu obejmuje wyłącznie tabele istniejące w chwili wykonania polecenia — każda nowa tabela utworzona później w tym samym schemacie znowu nie ma żadnych uprawnień dla naszej roli, dopóki ktoś nie nada ich ręcznie. `ALTER DEFAULT PRIVILEGES` rozwiązuje to na przyszłość, definiując uprawnienia, jakie mają być nadawane automatycznie każdej nowo tworzonej tabeli w danym schemacie.

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

Przykład:

```
ALTER DEFAULT PRIVILEGES
```

Uważaj: reguła obowiązuje tylko dla obiektów tworzonych przez tę samą rolę, która wykonała `ALTER DEFAULT PRIVILEGES` — jeśli tabele w schemacie tworzy inne konto, regułę trzeba ustawić z klauzulą `FOR ROLE NazwaTworcy`.

◫ TYLKO ODCZYT

Sprawdzenie nadanych uprawnień: information_schema

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

Sprawdzamy, jakie uprawnienia konkretna rola faktycznie ma nadane na poszczególnych tabelach — przydatne przy audycie dostępu albo przy diagnozowaniu, dlaczego aplikacja dostaje `permission denied`.

Przykład:

```
    grantee    | table_schema | table_name  | privilege_type
---------------+--------------+-------------+----------------
 nazwa_usera   | sprzedaz     | zamowienia  | SELECT
 nazwa_usera   | sprzedaz     | zamowienia  | INSERT
 nazwa_usera   | sprzedaz     | zamowienia  | UPDATE
(3 rows)
```

Role a użytkownicy

> **Pojęcie: rola.** W PostgreSQL nie ma osobnego pojęcia „użytkownik" i osobnego „grupa" tak, jak w niektórych innych silnikach — jest jeden obiekt, rola, a to, czy działa jak konto do logowania, czy jak grupa uprawnień, zależy wyłącznie od jej atrybutów. Rola z atrybutem `LOGIN` może się nią zalogować klient — to jest to, co potocznie nazywamy użytkownikiem. Rola bez `LOGIN` służy wyłącznie jako kontener uprawnień, do którego przypisujemy inne role przez `GRANT NazwaRoli TO NazwaUsera` — to jest odpowiednik grupy.

`CREATE USER` i `CREATE ROLE` tworzą dokładnie ten sam obiekt — jedyna różnica to domyślne atrybuty: `CREATE USER` dokłada `LOGIN` automatycznie, `CREATE ROLE` nie. Dlatego `psql` po obu poleceniach zwraca ten sam komunikat, `CREATE ROLE`, co widać na stronie Podstawowe operacje na bazie i użytkownikach.

CREATE ROLE z opcjami

```
CREATE ROLE NazwaUsera WITH LOGIN PASSWORD 'HasloUzytkownika' CONNECTION LIMIT 10;
```

Tworzymy rolę do logowania z hasłem i limitem jednoczesnych połączeń — `CONNECTION LIMIT` chroni instancję przed jednym kontem aplikacyjnym, które przez błąd w puli połączeń otwiera ich nieskończenie wiele.

Przykład:

```
CREATE ROLE
```

Najczęściej używane atrybuty roli, którymi doprecyzowujemy `CREATE ROLE`:

Atrybut	Znaczenie
`LOGIN` / `NOLOGIN`	czy rolą można się zalogować
`SUPERUSER` / `NOSUPERUSER`	pełne uprawnienia administracyjne w klastrze
`CREATEDB`	prawo tworzenia nowych baz danych
`CREATEROLE`	prawo tworzenia i modyfikowania innych ról
`REPLICATION`	prawo nawiązywania połączeń replikacyjnych
`INHERIT` / `NOINHERIT`	czy rola automatycznie dziedziczy uprawnienia ról, do których należy, bez `SET ROLE`
`CONNECTION LIMIT n`	maksymalna liczba jednoczesnych połączeń tej roli
`VALID UNTIL 'data'`	data wygaśnięcia konta

Role wbudowane

> **Pojęcie: rola wbudowana (predefined role).** Zestaw ról tworzonych automatycznie przez `initdb`, dających dostęp do konkretnego, wąskiego zakresu uprawnień administracyjnych bez konieczności nadawania pełnego `SUPERUSER`. Wprowadzone w PostgreSQL 10 właśnie po to, żeby konta monitorujące i serwisowe nie musiały być superuserami.

Rola wbudowana	Co daje
`pg_monitor`	odczyt wszystkich widoków i funkcji monitorujących — łączy w sobie trzy poniższe role
`pg_read_all_stats`	odczyt wszystkich widoków statystyk (`pg_stat_*`), łącznie z tymi wymagającymi normalnie uprawnień superusera

Rola wbudowana	Co daje
`pg_read_all_settings`	odczyt wszystkich parametrów konfiguracyjnych, także tych ukrytych dla zwykłych ról
`pg_signal_backend`	prawo wywołania `pg_cancel_backend` i `pg_terminate_backend` na cudzych sesjach, opisanych na stronie Sesje, długie zapytania, kasowanie kwerendy
`pg_read_server_files`	odczyt plików z dysku serwera przez funkcje takie jak `COPY FROM`
`pg_write_server_files`	zapis plików na dysku serwera
`pg_execute_server_program`	uruchamianie programów zewnętrznych z poziomu serwera

Uważaj: role wbudowane istnieją dopiero od PostgreSQL 10. Na starszych wersjach jedynym sposobem na nadanie kontu szerokiego dostępu do statystyk monitorujących bywało tymczasowe nadanie `SUPERUSER` — rozwiązanie gorsze z punktu widzenia bezpieczeństwa, ale jedyne dostępne na tamtych wersjach, co widać w przykładzie konfiguracji `zbx_monitor` niżej.

Z życia wzięte

> **Z ŻYCIA WZIĘTE**
>
> Konto `zbx_monitor` na pg01 od lat ma nadany `SUPERUSER`, bo Zabbixa konfigurowano jeszcze na starszej wersji PostgreSQL. Przy okazji przeglądu uprawnień zamieniamy je na rolę `pg_monitor`, węższą i wystarczającą do tego samego monitoringu. Zabbix dalej widzi wszystkie potrzebne statystyki, ale bez pełnych uprawnień administracyjnych na koncie serwisowym.

Częste błędy

Nadanie GRANT na tabele bez wcześniejszego USAGE na schemacie. Aplikacja dostaje `permission denied for schema`, mimo że uprawnienia na samej tabeli wyglądają poprawnie — brakującym ogniwem jest `USAGE` na schemacie nadrzędnym.

Zakładanie, że uprawnienia obejmą też przyszłe tabele. `GRANT ... ON ALL TABLES IN SCHEMA` jest migawką stanu w chwili wykonania — bez `ALTER DEFAULT PRIVILEGES` każda kolejna migracja dodająca nowe tabele wymaga ponownego, ręcznego `GRANT`.

Powiązane: Podstawowe operacje na bazie i użytkownikach · postgresql.conf i pg_hba.conf · pgBackRest: podstawowe komendy · Sesje, długie zapytania, kasowanie kwerendy

Źródła: dokumentacja PostgreSQL, rozdział „Database Roles" i „Predefined Roles"

Revision #43
Created 2026-09-24 23:08:07 UTC by Tomasz Konieczny
Updated 2026-10-26 14:12:27 UTC by Tomasz Konieczny