Podstawowe operacje na bazie i użytkownikach

Poziom: podstawy **Dotyczy: PostgreSQL 9.6–17** **Czas: ~15 min**

Garść operacji, które wykonujemy zaraz po pierwszym zalogowaniu do świeżej instancji — zanim jeszcze dojdzie do konfiguracji WAL czy backupów.

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
SQL	Structured Query Language: język zapytań do relacyjnych baz danych
WAL	Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych

Fiszka

ZMIENIA SYSTEM

Utworzenie nowej bazy danych: CREATE DATABASE

```
CREATE DATABASE NazwaBazy;
```

Tworzymy nową bazę danych w bieżącym klastrze — nowy podkatalog w `base/`, opisanym na stronie Architektura PostgreSQL: klaster, baza, schemat, procesy, i kopię wzorca `template1` jako punkt startowy.

Przykład:

```
CREATE DATABASE
```

ZMIENIA SYSTEM

Utworzenie superusera: CREATE USER ... WITH SUPERUSER

```
CREATE USER NazwaUsera WITH SUPERUSER PASSWORD 'HasloUzytkownika';
```

Tworzymy rolę z prawem logowania i pełnymi uprawnieniami administracyjnymi w klastrze — rolę superusera nadajemy tylko kontom faktycznie odpowiedzialnym za administrację instancji, nigdy kontom aplikacyjnym. Pojęcie roli i różnicę między rolą a userem rozwija strona Użytkownicy, role i uprawnienia.

Przykład:

```
CREATE ROLE
```

Uważaj: `psql` po wykonaniu `CREATE USER` zwraca komunikat `CREATE ROLE`, nie `CREATE USER` — w PostgreSQL to dokładnie ta sama operacja, `CREATE USER` jest tylko historycznym synonimem `CREATE ROLE ... LOGIN`.

□□ ZMIENIA SYSTEM

Zmiana hasła użytkownika: ALTER USER

```
ALTER USER NazwaUsera WITH PASSWORD 'NoweHaslo';
```

Nadpisujemy hasło istniejącej roli — działa identycznie dla zwykłego użytkownika i dla superusera.

Przykład:

```
ALTER ROLE
```

Równoważnym skrótem, wygodnym bo pyta o nowe hasło interaktywnie zamiast wpisywać je jawnie w poleceniu (gdzie zostałoby zapisane w historii `psql`), jest metakomenda:

```
\password NazwaUsera
```

Przykład:

```
Enter new password for user "nazwa_usera":
Enter it again:
```

◫ TYLKO ODCZYT

Sprawdzenie userów oraz ich ról: \du oraz pg_user

```
\du
```

Metakomenda została opisana szczegółowo na stronie psql: klient wiersza poleceń. Równoważny wynik, w postaci zwykłego zapytania SQL zamiast metakomendy, daje odpytanie widoku systemowego:

```
SELECT * FROM pg_user;
```

Przykład:

```
   usename   | usesysid | usecreatedb | usesuper | userepl | usebypassrls |  passwd  |
valuntil |          useconfig
-------------+----------+-------------+----------+---------+--------------+----------
+----------+-----------------------------
 postgres    |       10 | t           | t        | t       | t            | ********
|          |
 nazwa_usera |    16384 | t           | f        | f       | f            | ********
|          |
 zbx_monitor |    16391 | f           | f        | f       | f            | ********
|          |
(3 rows)
```

Widok `pg_user` pokazuje tylko role z prawem logowania — pełną listę ról, łącznie z rolami grupowymi bez prawa logowania, zwraca `\du` albo szerszy widok `pg_roles`.

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

◫ ZMIENIA SYSTEM

Nadanie uprawnień do bazy danych: GRANT

```
GRANT ALL PRIVILEGES ON DATABASE NazwaBazy TO NazwaUsera;
```

Nadajemy wskazanej roli pełne uprawnienia na poziomie całej bazy — najczęściej pierwszy krok po utworzeniu konta aplikacyjnego, zanim doprecyzujemy uprawnienia na poziomie schematu i tabel. Pełny mechanizm `GRANT`, role wbudowane i domyślne uprawnienia dla nowo tworzonych obiektów rozwija strona Użytkownicy, role i uprawnienia.

Przykład:

```
GRANT
```

◫◫ OSTROŻNIE · NIEODWRACALNE

Usunięcie całej zawartości tabeli: TRUNCATE TABLE

```
TRUNCATE TABLE NazwaTabeli;
```

> **Pojęcie: TRUNCATE.** Usuwamy wszystkie wiersze tabeli bez skanowania ich pojedynczo, w odróżnieniu od `DELETE FROM` bez `WHERE`, które usuwa wiersz po wierszu i zapisuje każde usunięcie do WAL osobno. `TRUNCATE` zwalnia strony dyskowe od razu i resetuje też powiązane sekwencje `IDENTITY`, jeśli nie podamy inaczej — różnicę między `DELETE`, `TRUNCATE` i `DROP` rozwija strona DELETE vs TRUNCATE vs DROP.

Przykład:

```
TRUNCATE TABLE
```

Wariant z `ONLY` obcina wyłącznie wskazaną tabelę, pomijając tabele potomne dziedziczące po niej strukturę (partycje albo klasyczne dziedziczenie tabel):

```
TRUNCATE ONLY NazwaTabeli;
```

Przykład:

```
TRUNCATE TABLE
```

Uważaj: `TRUNCATE` wymaga blokady wyłącznej na tabeli (`ACCESS EXCLUSIVE`) — na tabeli, do której akurat ktoś się odpytuje, polecenie czeka na zwolnienie blokady, a samo, dopóki czeka, blokuje nowe odczyty ustawiane w kolejce za nim.

◫◫ **WAL** = Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych

Z życia wzięte

Z ŻYCIA WZIĘTE

Nowa aplikacja zespołu Oli potrzebuje własnej bazy na pg01, zanim ktokolwiek zdąży ustalić konwencję nazewnictwa kont. Zakładamy `CREATE DATABASE` i osobne konto z `GRANT` ograniczonym do jednej bazy, zamiast dawać aplikacji `SUPERUSER` na szybko przed demo dla pana Zbigniewa. Demo wypada o czasie, a konto zostaje bez zbędnych uprawnień na później.

Częste błędy

Nadawanie SUPERUSER kontom aplikacyjnym zamiast precyzyjnych uprawnień. Konto z prawem `SUPERUSER` omija wszystkie mechanizmy `GRANT` i `RLS` — jeśli aplikacja potrzebuje tylko odczytu i zapisu w jednym schemacie, dostaje dokładnie to, nie więcej, zgodnie z zasadą najmniejszych uprawnień rozwiniętą na stronie Użytkownicy, role i uprawnienia.

Użycie DELETE FROM zamiast TRUNCATE przy czyszczeniu dużej tabeli. Dla tabeli liczącej miliony wierszy `DELETE FROM bez WHERE` generuje ogromny ruch WAL i zostawia po sobie martwe krotki wymagające `VACUUM`, opisane na stronie VACUUM, autovacuum i bloat — `TRUNCATE` tego problemu nie ma.

Powiązane: psql: klient wiersza poleceń · Użytkownicy, role i uprawnienia · DELETE vs TRUNCATE vs DROP · VACUUM, autovacuum i bloat

Źródła: dokumentacja PostgreSQL, rozdział „SQL Commands" — `CREATE ROLE`, `ALTER ROLE`, `TRUNCATE`

Revision #43
Created 2026-09-24 23:08:02 UTC by Tomasz Konieczny
Updated 2026-10-26 14:12:18 UTC by Tomasz Konieczny