Połączenie i konfiguracja

psql, pliki konfiguracyjne, uwierzytelnianie i podstawowe operacje na bazie.

- psql: klient wiersza poleceń
- Podstawowe operacje na bazie i użytkownikach
- postgresql.conf i pg_hba.conf

psql: klient wiersza poleceń

Poziom: podstawy **Dotyczy: PostgreSQL 9.6-17** **Czas: ~20 min**

`psql` jest jedynym narzędziem, którego naprawdę trzeba się nauczyć w PostgreSQL — cała reszta administracji rozgrywa się wewnątrz tej samej sesji, metakomendami i zapytaniami do katalogu systemowego.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
CSV	w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami
SQL	Structured Query Language: język zapytań do relacyjnych baz danych
TCP	Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia

Fiszka

TYLKO ODCZYT

Wylistowanie wszystkich baz danych: \l

```
\l
```

Zwracamy listę baz danych w klastrze wraz z właścicielem, kodowaniem i uprawnieniami dostępu.

Przykład:

```
                                  List of databases
    Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
------------+----------+----------+-------------+-------------+-----------------------
 postgres   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
```

```
 produkcja  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
            |          |          |             |             | postgres=CTc/postgres
 template1  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
            |          |          |             |             | postgres=CTc/postgres
(4 rows)
```

Wariant z plusem, `\l+`, dokłada rozmiar każdej bazy i tablespace, w którym leży.

TYLKO ODCZYT

Wylistowanie wszystkich schematów: \dn

```
\dn
```

Zwracamy schematy istniejące w bieżącej bazie, do której jesteśmy podłączeni.

Przykład:

```
        List of schemas
    Name    |       Owner
------------+--------------------
 public     | postgres
 sprzedaz   | nazwa_usera
(2 rows)
```

TYLKO ODCZYT

Wylistowanie tabel z bieżącego schematu: \dt

```
\dt
```

Bez dodatkowego argumentu `\dt` pokazuje tabele wyłącznie z pierwszego schematu na ścieżce przeszukiwania (`search_path`) — zwykle `public`.

Przykład:

```
          List of relations
 Schema |    Name    | Type  |  Owner
```

```
--------+------------+-------+----------
 public | klienci    | table | postgres
 public | zamowienia | table | postgres
(2 rows)
```

◫◫ TYLKO ODCZYT

Wylistowanie tabel z konkretnego schematu: \dt NazwaSchematu.*

```
\dt sprzedaz.*
```

Wskazujemy schemat wprost, gdy interesują nas tabele spoza domyślnej ścieżki przeszukiwania.

Przykład:

```
             List of relations
  Schema  |     Name      | Type  |   Owner
----------+---------------+-------+------------
 sprzedaz | faktury       | table | nazwa_usera
 sprzedaz | pozycje_faktur| table | nazwa_usera
(2 rows)
```

◫◫ TYLKO ODCZYT

Wylistowanie tabel ze wszystkich schematów: \dt .

```
\dt *.*
```

Rozszerzamy wzorzec na wszystkie schematy w bazie, łącznie z `pg_catalog` i `information_schema`, jeśli nie odfiltrujemy ich ręcznie.

Przykład:

```
               List of relations
   Schema   |      Name       | Type  |   Owner
------------+-----------------+-------+------------
 public     | klienci         | table | postgres
 public     | zamowienia      | table | postgres
```

```
 sprzedaz   | faktury         | table | nazwa_usera
 sprzedaz   | pozycje_faktur  | table | nazwa_usera
(4 rows)
```

🔒 TYLKO ODCZYT

Wyświetlenie definicji tabeli: \d NazwaTabeli

```
\d zamowienia
```

> **Pojęcie: definicja obiektu w `\d`.** Ta metakomenda zwraca nie dane z tabeli, tylko jej strukturę — kolumny z typami, indeksy założone na tabeli, ograniczenia (`NOT NULL`, `FOREIGN KEY`, `CHECK`) i triggery, jeśli jakieś na tabeli wiszą.

Przykład:

```
                                  Table "public.zamowienia"
   Column    |  Type   | Collation | Nullable |                Default
-------------+---------+-----------+----------+---------------------------------------
 id          | integer |           | not null | nextval('zamowienia_id_seq'::regclass)
 klient_id   | integer |           | not null |
 data        | date    |           | not null |
 wartosc     | numeric |           |          |
Indexes:
    "zamowienia_pkey" PRIMARY KEY, btree (id)
    "idx_zamowienia_klient_id" btree (klient_id)
Foreign-key constraints:
    "zamowienia_klient_id_fkey" FOREIGN KEY (klient_id) REFERENCES klienci(id)
```

🔒 TYLKO ODCZYT

Wyświetlenie wszystkich tabel bieżącego schematu przez samo \d

```
\d
```

Bez argumentu `\d` działa identycznie jak `\dt`, ale dokłada do wyniku też widoki i sekwencje — pełny obraz obiektów relacyjnych schematu, nie tylko tabel.

Przykład:

```
               List of relations
 Schema |         Name          |   Type   |  Owner
--------+-----------------------+----------+----------
 public | klienci               | table    | postgres
 public | zamowienia            | table    | postgres
 public | zamowienia_id_seq     | sequence | postgres
(3 rows)
```

TYLKO ODCZYT

Wylistowanie wszystkich indeksów: \di

```
\di
```

Zwracamy indeksy niezależnie od tego, do której tabeli należą — przydatne przy szukaniu indeksu po nazwie, bez pamiętania, na jakiej tabeli leży.

Przykład:

```
                   List of relations
 Schema |           Name            | Type  |  Owner   |    Table
--------+---------------------------+-------+----------+-------------
 public | idx_zamowienia_klient_id  | index | postgres | zamowienia
 public | zamowienia_pkey           | index | postgres | zamowienia
 public | klienci_pkey              | index | postgres | klienci
(3 rows)
```

TYLKO ODCZYT

Wylistowanie wszystkich ról i użytkowników: \du

```
\du
```

W PostgreSQL role i użytkownicy to ten sam obiekt — różnicę i konsekwencje tego rozwija strona Użytkownicy, role i uprawnienia.

Przykład:

```
                                   List of roles
  Role name  |                         Attributes                         | Member of
-------------+------------------------------------------------------------+-----------
 nazwa_usera | Create DB                                                  | {}
 postgres    | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 zbx_monitor |                                                            |
{pg_monitor}
(3 rows)
```

Wariant z plusem, `\du+`, dokłada opis (`Description`) roli, jeśli został ustawiony komendą `COMMENT ON ROLE`.

◫◫ TYLKO ODCZYT

Wylistowanie wszystkich funkcji: \df

```
\df
```

Zwracamy funkcje i procedury widoczne na ścieżce przeszukiwania, wraz z typem zwracanym i listą argumentów.

Przykład:

```
                          List of functions
 Schema |     Name      | Result data type | Argument data types |  Type
--------+---------------+------------------+---------------------+--------
 public | licz_rabat    | numeric          | klient_id integer   | func
(1 row)
```

◫◫ TYLKO ODCZYT

Wylistowanie wszystkich widoków: \dv

```
\dv
```

Analogicznie do `\dt`, ale dla widoków — wariant z gwiazdką (`\dv *.*`) rozszerza wynik na wszystkie schematy.

Przykład:

```
              List of relations
 Schema |        Name         | Type |  Owner
--------+---------------------+------+----------
 public | v_zamowienia_klient  | view | postgres
(1 row)
```

🔒 TYLKO ODCZYT

Otwarcie edytora tekstowego dla bieżącego polecenia: \e

```
\e
```

Otwieramy w zewnętrznym edytorze (zmienna `EDITOR`, domyślnie `vi`) bufor z ostatnio wpisanym poleceniem — wygodne przy pisaniu dłuższych zapytań wielolinijkowych, których niewygodnie poprawia się bezpośrednio w terminalu. Po zapisaniu i zamknięciu edytora `psql` wykonuje zawartość bufora.

🔒 TYLKO ODCZYT

Edycja treści funkcji w edytorze: \ef

```
\ef NazwaFunkcji
```

Otwieramy w edytorze definicję istniejącej funkcji, gotową do modyfikacji — po zapisaniu `psql` sam wykonuje `CREATE OR REPLACE FUNCTION` z poprawioną treścią.

🔒 TYLKO ODCZYT

Przełączenie formatu wyniku na rozszerzony: \x

```
\x
```

> **Pojęcie: format rozszerzony.** Zamiast szerokiej tabeli z kolumnami obok siebie, każdy wiersz wyniku wypisywany jest pionowo, jedna kolumna pod drugą — nieocenione przy tabelach z wieloma kolumnami albo z długą treścią w jednej z nich, które w zwykłym formacie łamią się na ekranie.

Przykład:

```
\x
Expanded display is on.
```

```
SELECT * FROM zamowienia WHERE id = 1;
```

```
-[ RECORD 1 ]-------
id        | 1
klient_id | 42
data      | 2026-01-15
wartosc   | 1250.00
```

Wariant `\x auto` włącza format rozszerzony tylko wtedy, gdy wynik nie mieści się na szerokości terminala, i jest tym, co warto wpisać do `.psqlrc`.

TYLKO ODCZYT

Włączenie pomiaru czasu wykonania: \timing

```
\timing
```

Po włączeniu każde kolejne polecenie wypisuje na końcu czas, jaki zajęło jego wykonanie — pierwsza rzecz, którą włączamy przy diagnozowaniu wolnych zapytań opisanym na stronie Sesje, długie zapytania, kasowanie kwerendy.

Przykład:

```
Timing is on.
```

```
SELECT count(*) FROM zamowienia;
```

```
 count
-------
 48213
(1 row)

Time: 12.481 ms
```

TYLKO ODCZYT

Przełączenie się na inną bazę: \c

```
\c NazwaBazy
```

Zamykamy bieżące połączenie i otwieramy nowe, do innej bazy w tym samym klastrze — jedyny sposób w `psql`, żeby zobaczyć obiekty innej bazy, bo pojedyncze połączenie zawsze celuje w jedną konkretną bazę.

Przykład:

```
\c produkcja
```

```
You are now connected to database "produkcja" as user "nazwa_usera".
```

Metakomenda przyjmuje też host, port i użytkownika, dzięki czemu można nią też przełączyć się na inny serwer bez wychodzenia z `psql`:

```
\c produkcja nazwa_usera pg02 5432
```

◫◫ TYLKO ODCZYT

Wykonanie komendy powłoki bez wychodzenia z psql: \!

```
\! komenda
```

Uruchamiamy dowolną komendę systemową w podpowłoce, nie zamykając sesji `psql` — wygodne, żeby na przykład sprawdzić rozmiar pliku dumpa leżącego obok, bez przełączania terminali.

Przykład:

```
\! ls -lh /var/lib/pgsql/backup/
```

```
-rw-------. 1 postgres postgres 842M lut 12 03:00 produkcja_full.dump
```

◫◫ TYLKO ODCZYT

Wykonanie skryptu SQL z pliku: \i

```
\i SciezkaDoPliku
```

Wczytujemy i wykonujemy zawartość wskazanego pliku tak, jakby jego treść została wpisana ręcznie w sesji — standardowy sposób odtworzenia zrzutu w formacie `plain` z `pg_dump`, opisanego na stronie pg_dump, pg_dumpall, pg_restore.

Przykład:

```
\i /path/to/file_dump.sql
```

```
CREATE TABLE
COPY 48213
ALTER TABLE
```

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

TYLKO ODCZYT

Import lub eksport danych przez klienta: \copy

```
\copy NazwaTabeli TO 'SciezkaDoPliku' WITH CSV HEADER
```

> **Pojęcie: `\copy` kontra `COPY`.** Polecenie SQL `COPY` czyta i zapisuje plik z perspektywy procesu serwera, więc plik musi leżeć na dysku serwera i być dostępny dla konta systemowego `postgres`. Metakomenda `\copy` robi to samo, ale plik czyta i zapisuje klient `psql` na swoim własnym dysku, przesyłając dane przez połączenie — dzięki temu eksport czy import działa też wtedy, gdy łączymy się z odległej maszyny i nie mamy dostępu do systemu plików serwera.

Przykład eksportu tabeli do pliku CSV:

```
\copy zamowienia TO '/home/nazwa_usera/zamowienia.csv' WITH CSV HEADER
```

```
COPY 48213
```

SQL = Structured Query Language: język zapytań do relacyjnych baz danych · **CSV** = w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami

Z życia wzięte

Z ŻYCIA WZIĘTE

O 10:20 Ola prosi o dostęp do bazy cennik tylko na chwilę, żeby sprawdzić jedną tabelę. Łączymy się `psql -h pg01 -U ola.wisniewska cennik` i pokazujemy jej `\dt` zamiast dawać hasło do konta administracyjnego. Po skończonej robocie Ola wychodzi z sesji, a jej konto zostaje z uprawnieniami tylko do jednego schematu.

Czym jest psql

Pojęcie: psql. Interaktywny klient wiersza poleceń dostarczany razem z PostgreSQL, łączący w sobie zwykły interpreter SQL i zestaw własnych poleceń — metakomend zaczynających się od `\` — które nie są SQL-em, tylko skrótami wykonującymi za nas zapytania do katalogu systemowego albo zmieniającymi zachowanie samej sesji. Metakomenda `\dt` nie istnieje jako polecenie serwera — `psql` po jej wpisaniu sam buduje i wysyła odpowiednie `SELECT` do `pg_catalog`, a wynik formatuje w czytelną tabelę.

Parametry połączenia

Połączenie z instancją opisują cztery parametry, wszystkie opcjonalne — pominięte przyjmują wartości domyślne (host lokalny przez gniazdo uniksowe, port `5432`, użytkownik systemowy, baza o tej samej nazwie co użytkownik):

```
psql -h NazwaSerwera -p NumerPortu -U NazwaUsera -d NazwaBazy
```

Przykład połączenia do zdalnej instancji:

```
psql -h pg01 -p 5432 -U nazwa_usera -d produkcja
```

Po podaniu `-h` klient łączy się przez TCP, nawet jeśli serwer i klient siedzą na tym samym hoście — pominięcie `-h` wymusza gniazdo uniksowe, co jest tym, czego używa `sudo -u postgres psql` opisane na stronie Instalacja PostgreSQL z własnym katalogiem danych.

Plik .pgpass — hasło bez interaktywnego wpisywania

> **Pojęcie: .pgpass.** Plik w katalogu domowym użytkownika systemowego, trzymający hasła do połączeń — dzięki niemu skrypty i zadania `cron` łączą się z bazą bez interaktywnego pytania o hasło i bez wpisywania go wprost w treści skryptu. Każda linia to jeden wzorzec połączenia:

```
pg01:5432:produkcja:nazwa_usera:HasloUzytkownika
```

Gwiazdka w dowolnym polu dopasowuje każdą wartość. Plik musi mieć uprawnienia `0600` — z szerszymi uprawnieniami `libpq` go ignoruje, traktując jak nieistniejący.

Zmienne środowiskowe połączenia

Zamiast powtarzać flagi `-h`/`-p`/`-U`/`-d` w każdym wywołaniu, `psql` (a właściwie biblioteka `libpq`, z której korzysta też każde inne narzędzie CLI silnika, w tym `pg_dump`) czyta je ze zmiennych środowiskowych, jeśli flagi nie zostały podane:

Zmienna	Odpowiednik flagi
`PGHOST`	`-h`
`PGPORT`	`-p`
`PGUSER`	`-U`
`PGDATABASE`	`-d`
`PGPASSWORD`	hasło — odradzane, trafia do listy procesów; wolimy `.pgpass`

Plik .psqlrc — konfiguracja sesji

> **Pojęcie: .psqlrc.** Plik w katalogu domowym, wykonywany automatycznie przy każdym starcie `psql` — miejsce na metakomendy, które i tak wpisywalibyśmy ręcznie po każdym połączeniu, jak `\timing on` albo rozszerzony format wyniku. Plik `.psqlrc` przyjmuje dokładnie te same metakomendy, które opisuje reszta tej strony:

```
\timing on
\x auto
\set PROMPT1 '%n@%/%R%# '
```

Częste błędy

Podawanie hasła w PGPASSWORD w skrypcie widocznym dla innych userów. Zmienna środowiskowa procesu jest widoczna w `/proc/PID/environ` dla właściciela procesu i roota — `.pgpass` z uprawnieniami `0600` jest tu bezpieczniejszym rozwiązaniem.

Mylenie `\d` z `SELECT * FROM`. `\d NazwaTabeli` pokazuje strukturę, nie dane — nowa osoba czasem oczekuje zawartości tabeli i jest zaskoczona listą kolumn zamiast wierszy.

Powiązane: Podstawowe operacje na bazie i użytkownikach · postgresql.conf i pg_hba.conf · Sesje, długie zapytania, kasowanie kwerendy · pg_dump, pg_dumpall, pg_restore

Źródła: dokumentacja PostgreSQL, rozdział „psql" · `man 1 psql`

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`

postgresql.conf i pg_hba.conf

Poziom: podstawy **Dotyczy: PostgreSQL 9.6–17** **Czas: ~20 min**

Dwa pliki tekstowe decydują o wszystkim, co instancja robi i kogo do siebie wpuszcza — ta strona pokazuje, gdzie ich szukać, co w nich zmieniamy najczęściej i kiedy zmiana wymaga restartu, a kiedy wystarczy przeładowanie.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
SIGHUP	sygnał wysyłany, gdy zamykamy terminal, w którym proces wystartował, dawniej oznaczał rozłączenie linii telefonicznej
SQL	Structured Query Language: język zapytań do relacyjnych baz danych
TCP	Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia
TLS	Transport Layer Security: szyfrowanie połączeń, np. w HTTPS

Fiszka

TYLKO ODCZYT

Sprawdzenie ścieżki pliku postgresql.conf: SHOW config_file

```
SHOW config_file;
```

Pytamy działającą instancję wprost o ścieżkę do jej własnego pliku konfiguracyjnego — pewniejsze niż zgadywanie na podstawie dystrybucji, bo instancja mogła zostać zainstalowana z niestandardowym `PGDATA`, tematem strony Instalacja PostgreSQL z własnym katalogiem danych.

Przykład:

```
               config_file
-----------------------------------------
 /postgres/15/data/postgresql.conf
(1 row)
```

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

TYLKO ODCZYT

Wyświetlenie wszystkich parametrów konfiguracyjnych: SHOW ALL

```
SHOW ALL;
```

Zwracamy komplet parametrów wraz z bieżącą wartością i krótkim opisem — wygodne do przeszukania, gdy szukamy parametru po fragmencie nazwy.

Przykład:

```
              name               | setting |                        description
---------------------------------+---------
+------------------------------------------------------------
 max_connections                 | 100     | Sets the maximum number of concurrent
connections.
 shared_buffers                  | 128MB   | Sets the number of shared memory buffers
used by the server.
 work_mem                        | 4MB     | Sets the maximum memory to be used for
query workspaces.
(3 rows)
```

Równoważny wynik, w formie do dalszego przetwarzania w SQL, zwraca zapytanie do widoku katalogowego:

```
SELECT * FROM pg_settings;
```

SQL = Structured Query Language: język zapytań do relacyjnych baz danych · **show** = FortiOS: wypisuje konfigurację, w podstawowej postaci tylko ustawienia różniące się od domyślnych, a z wariantem full-configuration także wartości domyślne

TYLKO ODCZYT

Sprawdzenie wartości jednego parametru: current_setting()

```
SELECT current_setting('max_connections');
```

Pytamy o wartość dokładnie jednego parametru — czytelniejsze w skryptach niż filtrowanie wyniku `SHOW ALL`.

Przykład:

```
 current_setting
-----------------
 100
(1 row)
```

□□ TYLKO ODCZYT

Przeładowanie konfiguracji bez restartu: pg_reload_conf()

```
SELECT pg_reload_conf();
```

Nakazujemy instancji ponownie wczytać `postgresql.conf` i `pg_hba.conf` bez zrywania istniejących połączeń — działa dla większości parametrów, ale nie dla wszystkich, co opisuje tabela niżej. Odpowiednikiem z poziomu systemu operacyjnego, bez logowania do bazy, jest wysłanie sygnału `SIGHUP` do procesu postmastera albo komenda serwisowa:

```
sudo systemctl reload postgresql-15.service
```

Przykład:

```
 pg_reload_conf
----------------
 t
(1 row)
```

□□ **SIGHUP** = sygnał wysyłany, gdy zamykamy terminal, w którym proces wystartował, dawniej oznaczał rozłączenie linii telefonicznej · **systemctl** = system control: sterowanie usługami systemd

Z życia wzięte

Z ŻYCIA WZIĘTE

Zabbix na pg01 zgłasza coraz więcej odrzuconych połączeń tuż przed zamknięciem miesiąca, więc podnosimy `max_connections` w `postgresql.conf`. Przeładowanie sygnałem SIGHUP nic nie zmienia, bo ten konkretny parametr wymaga pełnego restartu instancji. Restart planujemy na wieczór, żeby nie przerywać pani Halinie pracy w księgowości w środku dnia.

Czym jest postgresql.conf

Pojęcie: postgresql.conf. Główny plik konfiguracyjny instancji, leżący w katalogu `PGDATA` opisanym na stronie Architektura PostgreSQL: klaster, baza, schemat, procesy. Każdy parametr to jedna linia `nazwa = wartosc`, z komentarzem `#` opisującym jednostkę i zakres. Plik jest wspólny dla całego klastra — dotyczy wszystkich baz w środku, nie da się mieć osobnego `postgresql.conf` dla jednej bazy (choć część parametrów da się nadpisać na poziomie pojedynczej bazy albo roli komendą `ALTER DATABASE ... SET` czy `ALTER ROLE ... SET`).

Najważniejsze parametry postgresql.conf

Poniższa tabela nie jest pełną listą — `SHOW ALL` zwraca kilkaset pozycji — tylko zestawem, który realnie zmieniamy w codziennej eksploatacji.

Parametr	Do czego służy	Wymaga restartu
`listen_addresses`	interfejsy sieciowe, na których instancja przyjmuje połączenia TCP	tak
`port`	port TCP nasłuchu	tak
`max_connections`	maksymalna liczba jednoczesnych połączeń klienckich	tak
`shared_buffers`	pamięć dzielona na bufor stron danych, zwykle 25% RAM serwera	tak
`work_mem`	pamięć na operację sortowania i haszowania w jednym zapytaniu	nie

Parametr	Do czego służy	Wymaga restartu
`maintenance_work_mem`	pamięć dla operacji utrzymaniowych — `VACUUM`, `CREATE INDEX`, `ALTER TABLE`	nie
`wal_level`	ilość informacji zapisywanej do WAL — `minimal`, `replica`, `logical`	tak
`archive_mode`	włączenie archiwizacji segmentów WAL	tak
`archive_command`	polecenie kopiujące zamknięty segment WAL do archiwum	nie
`checkpoint_timeout`	maksymalny odstęp czasu między checkpointami	nie
`log_min_duration_statement`	próg czasu wykonania, od którego zapytanie trafia do logu	nie
`log_destination`, `log_directory`, `log_filename`	dokąd i w jakich plikach trafiają logi serwera	częściowo (zależnie od parametru)

Uważaj: `work_mem` jest limitem na jedną operację sortującą albo haszującą wewnątrz jednego zapytania, nie na całe zapytanie ani na całą sesję — zapytanie z kilkoma sortowaniami i złączeniami hashowymi może w szczycie zużyć wielokrotność tej wartości, co jest częstym źródłem nieoczekiwanego zużycia pamięci przy zbyt optymistycznie podkręconym `work_mem`.

`shared_buffers`, `wal_level` i `max_connections` należą do parametrów, które zmieniają rozmiar segmentów pamięci dzielonej alokowanych przy starcie procesu — stąd wymóg restartu, nie samego przeładowania.

Czym jest pg_hba.conf

> **Pojęcie: pg_hba.conf.** Plik kontrolujący, kto i w jaki sposób może się uwierzytelnić w klastrze — leży w tym samym katalogu `PGDATA` co `postgresql.conf` i jest odczytywany jeszcze przed sprawdzeniem hasła. Nazwa pochodzi od „host-based authentication". Każda linia to jedna reguła, sprawdzana od góry pliku w dół — pierwsza pasująca reguła decyduje, żadna kolejna linia nie jest już brana pod uwagę, nawet jeśli pasowałaby lepiej.

Format linii to pięć kolumn: typ połączenia (`local` dla gniazda uniksowego, `host` dla TCP, `hostssl` dla TCP wymuszającego TLS), nazwa bazy, nazwa roli, adres klienta (pominięty dla `local`) i metoda uwierzytelniania.

Przykład fragmentu pliku, otwierającego dostęp replikacyjny i dostęp konta monitorującego z sieci lokalnej:

```
# TYPE  DATABASE        USER            ADDRESS         METHOD
local   all             postgres                        peer
host    replication     base_backup_user 127.0.0.1/32   trust
host    all             zbx_monitor     10.20.10.0/24   scram-sha-256
host    all             all             0.0.0.0/0       reject
```

Metody uwierzytelniania w pg_hba.conf

> **Pojęcie: metoda uwierzytelniania.** Sposób, w jaki instancja potwierdza tożsamość klienta pasującego do danej reguły — od całkowitego zaufania po hasło przesyłane w postaci nieodwracalnie zaszyfrowanej.

Metoda	Jak działa	Kiedy używamy
`trust`	wpuszcza bez żadnego sprawdzenia hasła	wyłącznie połączenia lokalne, zaufane, np. narzędzia utrzymaniowe na tym samym hoście — nigdy z sieci publicznej
`peer`	porównuje nazwę konta systemowego z nazwą roli bazodanowej	połączenia lokalne przez gniazdo uniksowe, typowo konto `postgres`
`ident`	jak `peer`, ale przez zewnętrzny serwer ident po sieci	połączenia TCP z zaufanej sieci, rzadko spotykane dziś
`md5`	hasło porównywane po zahaszowaniu MD5	starsze instalacje, kompatybilność wsteczna
`scram-sha-256`	hasło porównywane silniejszym mechanizmem SCRAM, domyślne od PostgreSQL 10+	zalecany standard dla połączeń hasłowych
`reject`	odrzuca połączenie bez sprawdzania czegokolwiek	reguła zamykająca plik, blokująca to, co nie pasowało do żadnej wcześniejszej, bardziej szczegółowej reguły

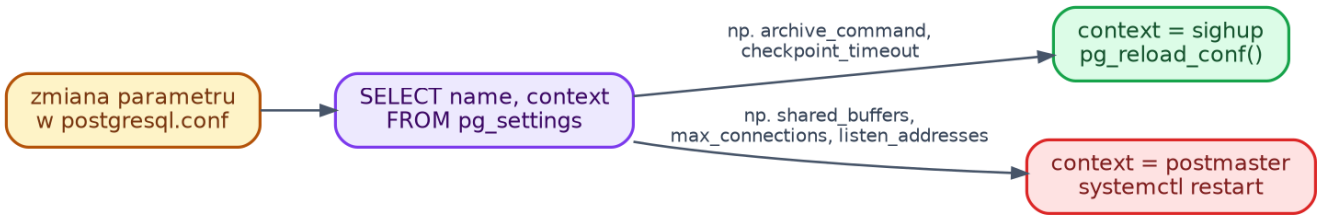
Uważaj: `trust` na regule pasującej do adresu szerszego niż pojedynczy host lokalny (`0.0.0.0/0` albo cała sieć firmowa) oznacza logowanie bez hasła dla każdego, kto potrafi nawiązać połączenie TCP do instancji — to jest błąd konfiguracyjny, nie skrót, nawet jeśli chwilowo wygodny przy pracach serwisowych, jak przy upgrade opisanym na stronie Upgrade silnika PostgreSQL.

Przeładowanie a restart

Zmiana w `pg_hba.conf` zawsze wystarcza przeładować — `pg_reload_conf()` albo `systemctl reload`, bez zrywania istniejących sesji. Zmiana w `postgresql.conf` wymaga przeładowania dla większości parametrów, ale nie dla tych związanych z rozmiarem pamięci dzielonej i topologią nasłuchu, wypisanych w tabeli wyżej — dla nich jedynym sposobem jest pełny restart usługi:

```
sudo systemctl restart postgresql-15.service
```

Sposobem na sprawdzenie z wyprzedzeniem, czy dany parametr wymaga restartu, jest kolumna `context` w `pg_settings`:



Kolumna context w pg_settings rozstrzyga, czy zmiana parametru wystarczy przeładować, czy wymaga pełnego restartu (kliknięcie otwiera pełny rozmiar)

```
SELECT name, context FROM pg_settings WHERE name = 'shared_buffers';
```

Przykład:

```
      name      |  context
----------------+------------
 shared_buffers  | postmaster
(1 row)
```

Wartość `postmaster` w kolumnie `context` oznacza, że parametr jest czytany tylko raz, przy starcie procesu nadzorczego — stąd konieczność restartu. Wartość `sighup` oznacza, że wystarczy przeładowanie sygnałem `SIGHUP`, czyli dokładnie to, co robi `pg_reload_conf()`.

Częste błędy

Edycja postgresql.conf i oczekiwanie efektu bez przeładowania. Sama zmiana pliku na dysku nic nie robi, dopóki instancja go nie wczyta ponownie — `pg_reload_conf()` albo restart, zależnie od parametru.

Zbyt szeroka reguła trust w pg_hba.conf zostawiona po pracach serwisowych. Reguła otwarta tymczasowo na czas upgrade czy migracji, o czym przypomina strona Upgrade silnika PostgreSQL, zapomniana i pozostawiona w produkcji, jest realną luką bezpieczeństwa, nie kosmetycznym niedopatrzeniem.

Powiązane: Architektura PostgreSQL: klaster, baza, schemat, procesy · Użytkownicy, role i uprawnienia · Logi WAL: włączenie i archiwizacja · Upgrade silnika PostgreSQL

Źródła: dokumentacja PostgreSQL, rozdział „Server Configuration" i „Client Authentication"