Rozpoznanie bazy

Rozmiary, obiekty, aktywne sesje i długie zapytania.

- Rozpoznanie bazy: rozmiary, obiekty, aktywność
- Sesje, długie zapytania, kasowanie kwerendy

Rozpoznanie bazy: rozmiary, obiekty, aktywność

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

Zestaw zapytań, od których zaczynamy pracę na nieznanej wcześniej instancji — co w niej siedzi, ile zajmuje i ile osób jest akurat podłączonych.

Fiszka

TYLKO ODCZYT

Wylistowanie największych tabel i indeksów: pg_class

```
SELECT
    nspname || '.' || relname AS "Object Name", relkind AS "Object Type",
    pg_size_pretty(pg_relation_size(C.oid)) AS "size"
FROM pg_class C
LEFT JOIN pg_namespace N ON (N.oid = C.relnamespace)
WHERE nspname NOT IN ('pg_catalog', 'information_schema')
ORDER BY pg_relation_size(C.oid) DESC
LIMIT 20;
```

Sprawdzamy, co w bazie faktycznie zajmuje miejsce na dysku — pierwsze zapytanie na nowej, nieznanej instancji, zanim jeszcze zaczniemy szukać przyczyn wolnego działania. Kolumna `relkind` odróżnia zwykłą tabelę (`r`) od indeksu (`i`), widoku materializowanego (`m`) czy sekwencji (`S`). Odfiltrowanie `pg_catalog` i `information_schema` odcina katalog systemowy, żeby lista pokazywała wyłącznie obiekty naszych danych.

Przykład:

```
             Object Name              | Object Type |  size
--------------------------------------+-------------+---------
 public.zamowienia                    | r           | 2145 MB
```

```
 public.idx_zamowienia_klient_id       | i           | 412 MB
 public.zamowienia_pkey                | i           | 380 MB
 public.klienci                        | r           | 98 MB
 public.pozycje_zamowien               | r           | 76 MB
(5 rows)
```

TYLKO ODCZYT

Sprawdzenie indeksów danej tabeli: pg_indexes

```
SELECT * FROM pg_indexes WHERE tablename = 'NazwaTabeli' AND schemaname =
'NazwaSchematu';
```

Sprawdzamy, jakie indeksy faktycznie istnieją na konkretnej tabeli i jaka jest ich pełna definicja — wygodniejsze do dalszego przetwarzania w skryptach niż metakomenda `\d`, bo zwraca gotowy tekst `CREATE INDEX` w kolumnie `indexdef`.

Przykład:

```
 schemaname |    tablename    |          indexname          | tablespace
|                              indexdef
------------+-----------------+-----------------------------+------------
+-------------------------------------------------------------------
 public     | zamowienia      | zamowienia_pkey             |            | CREATE UNIQUE
INDEX zamowienia_pkey ON public.zamowienia USING btree (id)
 public     | zamowienia      | idx_zamowienia_klient_id    |            | CREATE INDEX
idx_zamowienia_klient_id ON public.zamowienia USING btree (klient_id)
(2 rows)
```

TYLKO ODCZYT

Sprawdzenie rozmiaru bazy danych: pg_database_size

```
SELECT pg_size_pretty(pg_database_size('NazwaBazy'));
```

Sprawdzamy łączny rozmiar bazy na dysku, wliczając wszystkie jej tabele, indeksy i pozostałe obiekty — pierwszy numer, o który pyta się przy planowaniu pojemności wolumenu albo przy szacowaniu czasu backupu opisanego na stronie pg_dump, pg_dumpall, pg_restore.

Przykład:

```
 pg_size_pretty
----------------
 4823 MB
(1 row)
```

Rozmiary wszystkich baz w klastrze naraz, bez podawania nazwy, zwraca `\l+` opisane na stronie psql: klient wiersza poleceń.

TYLKO ODCZYT

Sprawdzenie ostatniej daty modyfikacji baz danych: pg_stat_file

```
SELECT (pg_stat_file('base/'||oid||'/PG_VERSION')).modification, datname
FROM pg_database;
```

> **Pojęcie: pg_stat_file.** Funkcja zwracająca metadane systemu plików (rozmiar, czas modyfikacji) dla wskazanej ścieżki wewnątrz `PGDATA` — tutaj wykorzystana jako pośredni sposób sprawdzenia, kiedy ostatnio dotknięty był katalog danej bazy, przez datę modyfikacji jej pliku `PG_VERSION`. To przybliżenie, nie precyzyjny znacznik ostatniej transakcji, ale wystarczające, żeby odróżnić bazy używane na bieżąco od takich, których nikt nie ruszał od lat.

Przykład:

```
       modification        |  datname
---------------------------+-----------
 2017-08-24 11:02:13+02    | template0
 2022-04-14 09:41:55+02    | template1
 2023-03-22 14:12:08+02    | archiwum
 2026-09-24 22:10:03+02    | produkcja
(4 rows)
```

TYLKO ODCZYT

Sprawdzenie liczby aktywnych połączeń: pg_stat_activity

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

Sprawdzamy, ile połączeń jest akurat otwartych do klastra, niezależnie od tego, czy coś w danym momencie wykonują — przydatne razem z limitem skonfigurowanym w `max_connections`, opisanym na stronie postgresql.conf i pg_hba.conf, żeby ocenić, jak blisko limitu jesteśmy.

Przykład:

```
 count
-------
    34
(1 row)
```

Porównanie z limitem w jednym zapytaniu:

```
SELECT count(*), current_setting('max_connections') FROM pg_stat_activity;
```

Przykład:

```
 count | current_setting
-------+-----------------
    34 | 100
(1 row)
```

Z życia wzięte

Z ŻYCIA WZIĘTE

Przejmujemy po odejściu poprzedniego administratora bazę cennik na pg01, nie wiedząc, co w niej w ogóle siedzi. Zapytanie do `pg_class` pokazuje tabelę `zamowienia_archiwum` ważącą więcej niż cała reszta bazy razem wzięta. Okazuje się, że nikt nigdy nie czyścił jej z danych sprzed pięciu lat.

Częste błędy

Liczenie rozmiaru tabeli bez wliczenia jej indeksów. `pg_relation_size` zwraca rozmiar samej tabeli albo samego indeksu — do łącznego rozmiaru obiektu wraz z jego indeksami i danymi TOAST służy `pg_total_relation_size`, dająca zwykle znacząco większą liczbę niż sama tabela.

Wnioskowanie o realnej aktywności bazy wyłącznie z daty modyfikacji PG_VERSION. Ta data zmienia się rzadko i nie odzwierciedla bieżących operacji `SELECT`/`INSERT` — do sprawdzenia faktycznej aktywności służy `pg_stat_activity` albo statystyki z `pg_stat_user_tables`, temat rozwinięty na stronie Sesje, długie zapytania, kasowanie kwerendy.

Powiązane: psql: klient wiersza poleceń · Sesje, długie zapytania, kasowanie kwerendy · Indeksy i plany wykonania w PostgreSQL · postgresql.conf i pg_hba.conf

Źródła: dokumentacja PostgreSQL, rozdział „System Catalogs" i „System Administration Functions"

Sesje, długie zapytania, kasowanie kwerendy

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

Zapytanie, które wisi od kwadransa i blokuje inne, to najczęstszy powód, dla którego ktoś w ogóle loguje się na produkcyjną instancję poza planowaną pracą — ta strona pokazuje, jak taką sesję znaleźć i jak ją bezpiecznie zakończyć.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
PID	Process ID: numer procesu w systemie

Fiszka

TYLKO ODCZYT

Kasowanie uruchomionej kwerendy: pg_cancel_backend

```
SELECT pg_cancel_backend(NumerPID);
```

Przerywamy wykonywane akurat zapytanie, zostawiając samą sesję i połączenie otwarte — klient dostaje błąd `ERROR: canceling statement due to user request` i może wysłać kolejne zapytanie w tym samym połączeniu.

Przykład:

```
 pg_cancel_backend
-------------------
 t
(1 row)
```

TYLKO ODCZYT

Zakończenie całej sesji: pg_terminate_backend

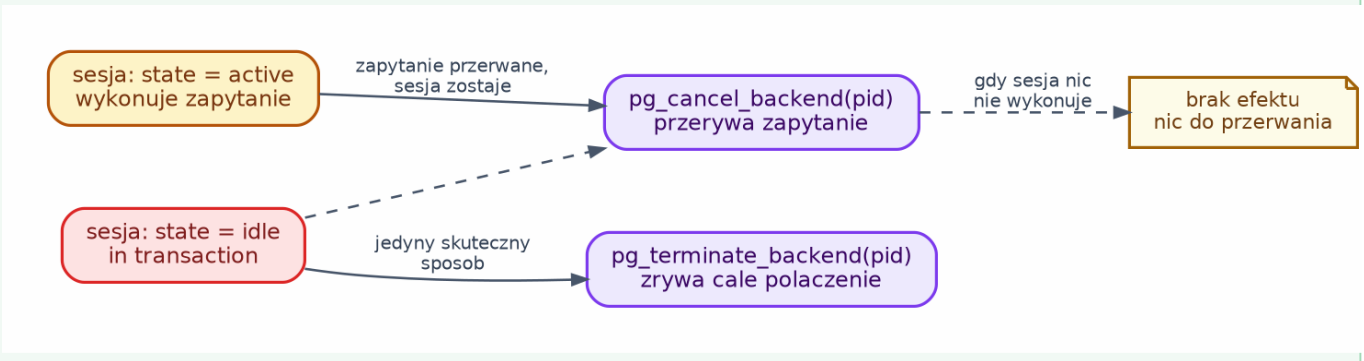
```
SELECT pg_terminate_backend(NumerPID);
```

> **Pojęcie: pg_cancel_backend kontra pg_terminate_backend.** `pg_cancel_backend` przerywa tylko bieżące zapytanie i zostawia sesję żywą — jeśli sesja jest w stanie `idle in transaction` i nic akurat nie wykonuje, `pg_cancel_backend` nie ma czego przerwać i nie da żadnego efektu. `pg_terminate_backend` zamyka cały proces backendu, zrywając połączenie klienta — jedyny skuteczny sposób na pozbycie się sesji zawieszonej w `idle in transaction` albo takiej, która ignoruje samo anulowanie zapytania.

Przykład:

```
 pg_terminate_backend
----------------------
 t
(1 row)
```

Uważaj: `pg_terminate_backend` zrywa połączenie w sposób, jaki aplikacja klienta odbiera jako nagły błąd sieci, nie jako czyste zakończenie transakcji — jeśli aplikacja nie ma logiki ponawiania połączenia, użytkownik po drugiej stronie zobaczy błąd.



Sesja aktywna traci tylko zapytanie po pg_cancel_backend, sesja idle in transaction wymaga pg_terminate_backend (kliknięcie otwiera pełny rozmiar)

Z życia wzięte

Z ŻYCIA WZIĘTE

Piątek, 16:05, panel Oli wisi, bo jedna sesja utknęła w stanie `idle in transaction` po błędzie w aplikacji. `pg_cancel_backend` nic nie daje, bo sesja niczego akurat nie wykonuje, więc sięgamy po `pg_terminate_backend` na znalezionym PID. Aplikacja Oli łapie błąd połączenia i sama się reconnectuje, panel wraca w kilka sekund.

Widok pg_stat_activity

Pojęcie: pg_stat_activity. Widok systemowy pokazujący jeden wiersz na każde aktywne połączenie do klastra — proces backendu opisany na stronie Architektura PostgreSQL: klaster, baza, schemat, procesy, aktualnie wykonywane zapytanie, jego stan i czas trwania. To pierwsze miejsce, w które zaglądamy, kiedy ktoś zgłasza, że baza „się zawiesiła".

Podejrzenie aktywnych sesji: pg_stat_activity

```
SELECT pid, usename, state, now() - query_start AS czas_trwania, query
FROM pg_stat_activity
WHERE state != 'idle'
ORDER BY czas_trwania DESC;
```

Sortujemy po czasie trwania, żeby najdłużej działające zapytanie znalazło się na górze wyniku — zwykle to ono jest przyczyną zgłoszenia.

Przykład:

```
  pid  |   usename   | state  | czas_trwania |                    query
-------+-------------+--------+--------------+-----------------------------------------------
 24501 | nazwa_usera | active | 00:14:32     | UPDATE zamowienia SET status = 'zamkniete' ...
 24488 | zbx_monitor | active | 00:00:01     | SELECT * FROM pg_stat_database
(2 rows)
```

> **Pojęcie: stan sesji (`state`).** Kolumna mówiąca, co backend robi w tej chwili — `active` oznacza wykonywanie zapytania, `idle` oznacza połączenie otwarte, ale bez zapytania w toku, `idle in transaction` oznacza otwartą transakcję, w której klient nic już nie wysyła, ale i nie zamknął jej `COMMIT`-em ani `ROLLBACK`-iem — ten ostatni stan jest szczególnie niebezpieczny, bo transakcja wisząca w tle trzyma otwarte blokady i uniemożliwia `VACUUM` sprzątanie wierszy, tematu rozwiniętego na stronie VACUUM, autovacuum i bloat.

Znalezienie sesji blokujących: pg_blocking_pids

```
SELECT pid, pg_blocking_pids(pid) AS blokowany_przez, query
FROM pg_stat_activity
WHERE pg_blocking_pids(pid) != '{}';
```

Sprawdzamy, które sesje czekają na zwolnienie blokady i przez jaki `PID` są zablokowane — mechanizm blokad, na którym to się opiera, opisuje ogólnie strona Blokady: współdzielone, wyłączne, eskalacja. Zamiast ręcznie przeglądać `pg_locks` i dopasowywać wiersze po `relation` i `pid`, ta funkcja od razu zwraca gotową listę PID-ów blokujących każdą oczekującą sesję.

Przykład:

```
  pid  | blokowany_przez |                    query
-------+-----------------+---------------------------------------------
 24519 | {24501}         | SELECT * FROM zamowienia WHERE id = 100
(1 row)
```

W tym przykładzie sesja `24519` czeka na sesję `24501` — dokładnie tę widoczną w poprzednim przykładzie jako najdłużej działająca. Naturalnym następnym krokiem jest sprawdzenie, czy sesja blokująca rzeczywiście musi jeszcze trwać, czy nadaje się do zakończenia w sposób opisany niżej.

Automatyczne limity czasu

Zamiast ręcznie polować na wiszące sesje po fakcie, oba typy problemu — zbyt długie zapytanie i zbyt długo otwarta, bezczynna transakcja — da się ograniczyć z góry parametrami konfiguracyjnymi z postgresql.conf i pg_hba.conf, ustawianymi globalnie albo na poziomie

pojedynczej roli.

Ustawienie limitu czasu zapytania: statement_timeout

```
SET statement_timeout = '30s';
```

Każde zapytanie w tej sesji, które przekroczy podany czas, zostaje automatycznie przerwane tak, jakby ktoś ręcznie wywołał `pg_cancel_backend` — bez czekania na interwencję administratora.

Przykład:

```
SET
```

Ustawienie limitu czasu bezczynnej transakcji: idle_in_transaction_session_timeout

```
SET idle_in_transaction_session_timeout = '5min';
```

Sesja, która otworzyła transakcję i nie wysłała żadnego kolejnego polecenia przez podany czas, zostaje automatycznie zamknięta przez instancję — dokładnie ten mechanizm, który ręcznie odtwarza `pg_terminate_backend` na sesji znalezionej przez `pg_blocking_pids`.

Przykład:

```
SET
```

Oba parametry ustawiamy zwykle nie pojedynczą komendą `SET` w sesji, tylko na stałe dla roli albo bazy:

```
ALTER ROLE NazwaUsera SET statement_timeout = '30s';
```

Przykład:

```
ALTER ROLE
```

Częste błędy

Wywołanie pg_cancel_backend na sesji idle in transaction i oczekiwanie efektu. Skoro sesja niczego akurat nie wykonuje, nie ma czego anulować — potrzebny jest `pg_terminate_backend`.

Ustawienie zbyt agresywnego statement_timeout globalnie, dla całej instancji. Krótki, uniwersalny limit czasu potrafi przerywać legalne, długie raporty analityczne — bezpieczniej ustawiać go dla konkretnej roli aplikacyjnej, a nie dla całego klastra naraz.

Powiązane: Rozpoznanie bazy: rozmiary, obiekty, aktywność · Blokady: współdzielone, wyłączne, eskalacja · VACUUM, autovacuum i bloat · postgresql.conf i pg_hba.conf

Źródła: dokumentacja PostgreSQL, rozdział „The Statistics Collector" i „System Administration Functions"