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

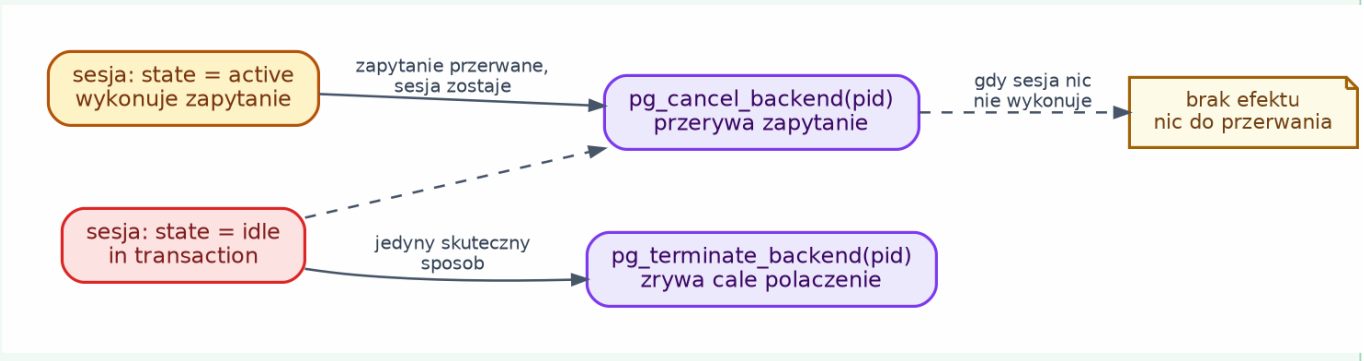
```
SELECT pg_terminate_backend(NumerPID);
```

> **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"

Revision #43
Created 2026-09-24 23:08:06 UTC by Tomasz Konieczny
Updated 2026-10-26 14:12:25 UTC by Tomasz Konieczny