Typowe awarie i co robimy

Typ: referencja **Dotyczy: PostgreSQL 9.6–17** **Czas: ~20 min**

Dziewięć sytuacji, z którymi administrator PostgreSQL styka się najczęściej — dla każdej: co widać w logu, jak potwierdzić diagnozę, co robimy i czego świadomie nie robimy.

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
LVM	Logical Volume Manager: warstwa w Linuksie składająca dyski w elastyczne wolumeny
WAL	Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych

Fiszka

TYLKO ODCZYT

Sprawdzenie liczby połączeń względem limitu: pg_stat_activity

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

Zapytanie i próg alarmowy przed osiągnięciem limitu opisuje strona Monitoring instancji.

TYLKO ODCZYT

Sprawdzenie zajętości dysku i katalogów PGDATA: df i du

```
df -h /var/lib/pgsql
du -sh /var/lib/pgsql/15/data/*
```

Sprawdzamy zarówno ogólne zapełnienie wolumenu, jak i to, który podkatalog `PGDATA` faktycznie rośnie — często okazuje się, że to nie same dane, tylko `pg_wal`.

df = disk free: ile miejsca wolnego na dyskach · **du** = disk usage: ile miejsca zajmuje katalog

□□ TYLKO ODCZYT

Sprawdzenie porzuconych slotów replikacyjnych: pg_replication_slots

```
SELECT slot_name, active, restart_lsn, wal_status FROM pg_replication_slots;
```

Slot z `active = f` i `wal_status` innym niż `reserved` (na przykład `lost`) wskazuje na porzuconą replikę albo subskrypcję, opisaną na stronach Replikacja strumieniowa i Replikacja logiczna — silnik przechowuje dla niego WAL bezterminowo, dopóki ktoś slotu nie usunie. Rozmiar samego katalogu sprawdzamy poleceniem z fiszki na stronie Rozmiary, tablespace i miejsce na dysku.

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

□□ TYLKO ODCZYT

Sprawdzenie martwych krotek i ostatniego autovacuum: pg_stat_user_tables

```
SELECT relname, n_dead_tup, last_autovacuum FROM pg_stat_user_tables ORDER BY n_dead_tup DESC LIMIT 5;
```

Pełny mechanizm martwych krotek i progów autovacuum opisuje strona VACUUM, autovacuum i bloat. Sprawdzamy też `pg_stat_activity` pod kątem długo otwartych transakcji, opisanych na stronie Monitoring instancji — jedna zapomniana transakcja `idle in transaction` potrafi zablokować sprzątanie na wielu tabelach naraz.

□□ TYLKO ODCZYT

Sprawdzenie planu zapytania: EXPLAIN (ANALYZE, BUFFERS)

```
EXPLAIN (ANALYZE, BUFFERS) SELECT ...;
```

Sposób czytania planu opisuje strona Indeksy i plany wykonania w PostgreSQL. Sprawdzamy trzy najczęstsze przyczyny po kolei: `last_analyze` w `pg_stat_user_tables` (nieaktualne statystyki po dużym imporcie, temat rozwinięty na stronie VACUUM, autovacuum i bloat), obecność indeksu oznaczonego `INVALID` po przerwanej budowie `CONCURRENTLY`, i procent martwych krotek na tabeli.

Z życia wzięte

> **Z ŻYCIA WZIĘTE**
>
> Po nagłej przerwie w zasilaniu serwerowni w środku nocy baza Zabbixa na pg01 nie odpowiada na połączenia i dział monitoringu zaczyna dzwonić. Log pokazuje `automatic recovery in progress`, więc zamiast restartować usługę w kółko, obserwujemy `tail -f` i czekamy, aż pozycja redo w kolejnych liniach faktycznie rośnie. Baza wraca sama po kilku minutach, bez utraty żadnej zatwierdzonej transakcji.

Tabela: objaw → przyczyna → pierwsze kroki

Objaw	Przyczyna	Pierwsze kroki
Baza nie startuje po nagłym wyłączeniu zasilania	przerwane odtwarzanie po awarii (crash recovery) w toku albo uszkodzony `pg_control`	`tail -f` na logu, cierpliwość dla trwającego redo
`FATAL: sorry, too many clients already`	wyczerpany `max_connections`, zwykle wyciek połączeń w aplikacji	`pg_stat_activity` wobec `max_connections`
`could not extend file: No space left on device`	wolumen z danymi albo z `pg_wal` zapełniony	`df -h`, `du -sh` na największych katalogach
katalog pg_wal urósł i nie maleje	porzucony slot replikacyjny albo zepsuty `archive_command`	`pg_replication_slots`, log pod kątem błędów archiwizacji
`FATAL: database is starting up` trwające długo	instancja wciąż w trybie odtwarzania, odmawia połączeń do jego zakończenia	tempo postępu redo w logu, obciążenie dysku
autovacuum nie nadąża i tabela puchnie	długo otwarta transakcja blokująca sprzątanie albo zbyt wysokie progi dla dużej tabeli	`pg_stat_user_tables`, `pg_stat_activity` pod kątem starych transakcji
`ERROR: canceling statement due to conflict with recovery` na replice	konflikt zapytania z odtwarzaniem WAL	ustawienia `max_standby_streaming_delay` i `hot_standby_feedback`
zapytanie, które wczoraj działało sekundę, dziś trwa minuty	nieaktualne statystyki, bloat albo uszkodzony indeks	`EXPLAIN (ANALYZE, BUFFERS)`, porównanie planu
uszkodzona strona danych	błąd dysku albo pamięci na poziomie systemu operacyjnego	komunikat `invalid page` w logu, stan sprzętu

Baza nie startuje po nagłym wyłączeniu zasilania

Co widzimy w logu:

```
LOG:  database system was interrupted; last known up at 2026-09-24 22:10:03
LOG:  database system was not properly shut down; automatic recovery in progress
LOG:  redo starts at 0/5A00028
```

Jak potwierdzamy diagnozę: obecność linii `automatic recovery in progress` oznacza, że silnik sam odtwarza się z WAL od ostatniego checkpointu — to jest normalna, wbudowana ścieżka po awarii, nie stan wymagający interwencji. Sprawdzamy, czy pozycja redo w kolejnych liniach logu faktycznie rośnie, co potwierdza, że proces posuwa się do przodu, a nie utknął.

Co robimy: czekamy na komunikat `database system is ready to accept connections`, obserwując log przez `tail -f`. Jeśli odtwarzanie trwa bardzo długo, sprawdzamy obciążenie dysku (`iostat`) — czas redo jest proporcjonalny do ilości WAL zapisanej od ostatniego checkpointu, opisanego na stronie Logi WAL: włączenie i archiwizacja, i do prędkości dysku. Jeśli log zamiast redo pokazuje `FATAL: could not read file "global/pg_control"`, plik kontrolny jest uszkodzony — dopiero to jest sytuacja wymagająca `pg_resetwal` jako absolutnej ostateczności, bo narzędzie to zeruje pozycję WAL i może skutkować utratą ostatnich transakcji.

Czego nie robimy: nie kasujemy ręcznie plików z `pg_wal` w nadziei na przyspieszenie startu i nie uruchamiamy `initdb` w tym samym katalogu danych — obie czynności niszczą możliwość odtworzenia stanu sprzed awarii.

FATAL: sorry, too many clients already

Co widzimy w logu:

```
FATAL:  sorry, too many clients already
```

Komunikat trafia do logu za każdym razem, gdy kolejne połączenie zostaje odrzucone, bo limit jest już wyczerpany.

Co robimy: identyfikujemy źródło połączeń przez `pg_stat_activity` pogrupowane po `usename` i `application_name` — najczęstszą przyczyną jest aplikacja, która otwiera nowe połączenie zamiast oddawać je do puli. Rozwiązaniem docelowym jest pula połączeń (`pgbouncer` albo pula wbudowana w aplikację), nie samo podniesienie `max_connections`.

Czego nie robimy: nie podnosimy `max_connections` odruchowo, bez sprawdzenia przyczyny — parametr ten wymaga restartu, opisanego na stronie postgresql.conf i pg_hba.conf, i zwiększa zapotrzebowanie na pamięć dzieloną niezależnie od tego, czy nowe połączenia są faktycznie potrzebne.

could not extend file: No space left on device

Co widzimy w logu:

```
ERROR:  could not extend file "base/16400/24601": No space left on device
HINT:  Check free disk space.
```

Co robimy: zwalniamy miejsce w bezpieczny sposób — usuwamy stare, już zarchiwizowane kopie backupu spoza `PGDATA`, albo rozszerzamy wolumen, jeśli leży na LVM, zgodnie z procedurą powiększania przestrzeni dyskowej. Po zwolnieniu miejsca instancja sama wznawia normalną pracę, bez potrzeby restartu.

Czego nie robimy: nie kasujemy ręcznie żadnych plików wewnątrz `PGDATA`, w tym segmentów WAL — nawet plik wyglądający na stary może być wciąż potrzebny do spójności bazy, a jego usunięcie kończy się uszkodzeniem klastra, nie odzyskaniem miejsca w bezpieczny sposób.

Katalog pg_wal urósł i nie maleje

Co widzimy w logu: zwykle brak jawnego komunikatu błędu — sygnałem jest sam rozmiar katalogu rosnący ponad `max_wal_size` bez odpowiadającego wzrostu ruchu transakcyjnego. Jeśli przyczyną jest zepsuty `archive_command`, w logu powtarza się:

```
LOG:  archive command failed with exit code 1
DETAIL:  The failed archive command was: cp %p /backup/db_log/%f
```

Co robimy: po potwierdzeniu, że slot rzeczywiście nikt nie używa, usuwamy go: `SELECT pg_drop_replication_slot('NazwaSlotu')`. Jeśli przyczyną jest zepsuty `archive_command`, naprawiamy

polecenie albo docelowy katalog archiwum, zgodnie ze stroną Logi WAL: włączenie i archiwizacja — po naprawie zaległe segmenty archiwizują się same, jeden po drugim.

Czego nie robimy: nie usuwamy plików z `pg_wal` ręcznie z poziomu systemu plików, nawet gdy jesteśmy pewni, które są „stare" — silnik sam decyduje, które segmenty są jeszcze potrzebne, i ręczna ingerencja może usunąć segment wymagany do spójnego odtworzenia.

FATAL: database is starting up

Co widzimy w logu klienta:

```
FATAL:  the database system is starting up
```

Sam serwer w tym czasie normalnie kontynuuje odtwarzanie w swoim własnym logu, bez żadnego komunikatu o błędzie — to klient dostaje odmowę, bo instancja jeszcze nie skończyła startu.

Jak potwierdzamy diagnozę: ten sam mechanizm co w pierwszym przypadku tej strony — sprawdzamy w logu serwera, czy pozycja redo w kolejnych liniach faktycznie się przesuwa, co odróżnia normalnie trwające odtwarzanie od zawieszonego procesu.

Co robimy: czekamy, monitorując tempo postępu w logu i obciążenie dysku — długie odtwarzanie po niekontrolowanym zamknięciu jest kosztem, jaki płacimy raz, proporcjonalnym do ilości niewykonanego jeszcze redo.

Czego nie robimy: nie restartujemy usługi w kółko, licząc na szybszy start — każdy restart w trakcie trwającego odtwarzania przerywa postęp i przy kolejnym uruchomieniu proces zaczyna się od tej samej pozycji, wydłużając łączny czas do gotowości bazy.

Autovacuum nie nadąża i tabela puchnie

Co widzimy w logu: przy włączonym `log_autovacuum_min_duration`, opisanym na stronie Logi serwera: konfiguracja i czytanie, pojawiają się wpisy o coraz dłuższym czasie trwania kolejnych przebiegów na tej samej tabeli, albo autovacuum w ogóle nie pojawia się dla niej od dłuższego czasu.

Co robimy: kończymy blokującą transakcję, jeśli ją znajdziemy, uruchamiamy ręczne `VACUUM (VERBOSE, ANALYZE)` na tabeli w oknie o niższym ruchu, i dostrajamy progi `autovacuum_vacuum_scale_factor` dla tej konkretnej, dużej tabeli zamiast dla całej instancji.

Czego nie robimy: nie uruchamiamy `VACUUM FULL` na produkcji bez okna serwisowego — blokada wyłączna na czas jego trwania jest kosztem, którego zwykłe `VACUUM` nie ma.

ERROR: canceling statement due to conflict with recovery na replice

Co widzimy w logu repliki:

```
ERROR:  canceling statement due to conflict with recovery
DETAIL:  User query might have needed to see row versions that must be removed.
```

Jak potwierdzamy diagnozę: sprawdzamy wartości `max_standby_streaming_delay` i `hot_standby_feedback` na replice — mechanizm konfliktu i rolę obu parametrów opisuje strona Replikacja strumieniowa. Powtarzający się konflikt na tej samej porze dnia zwykle pokrywa się z uruchomieniem konkretnego raportu na replice.

Co robimy: wydłużamy `max_standby_streaming_delay`, żeby dać zapytaniom więcej czasu przed przerwaniem, albo włączamy `hot_standby_feedback`, świadomie akceptując, że w zamian instancja źródłowa będzie dłużej trzymać martwe krotki potrzebne trwającym na replice zapytaniom.

Czego nie robimy: nie ignorujemy notorycznie powtarzającego się konfliktu jako „normalnego zachowania repliki" — częsty konflikt oznacza źle dobrane parametry względem realnego wzorca zapytań, nie cechę architektury, z którą trzeba żyć.

Zapytanie, które wczoraj działało sekundę, dziś trwa minuty

Co widzimy w logu: przy `log_min_duration_statement`, opisanym na stronie Logi serwera: konfiguracja i czytanie, samo zapytanie pojawia się z realnym czasem wykonania, ale log sam z siebie nie mówi dlaczego czas się zmienił.

Co robimy: odświeżamy statystyki ręcznym `ANALYZE NazwaTabeli`, porównujemy plan przed i po, i jeśli przyczyną jest indeks `INVALID`, usuwamy go i budujemy ponownie.

Czego nie robimy: nie zgadujemy przyczyny na podstawie samego odczucia „coś jest wolniejsze" bez porównania rzeczywistego planu wykonania — te same objawy dają zupełnie różne przyczyny,

wymagające różnych napraw.

Uszkodzona strona danych

Co widzimy w logu:

```
ERROR:  invalid page in block 128401 of relation base/16400/24601
```

Czasem poprzedzone komunikatem systemu operacyjnego o błędzie odczytu z dysku w `dmesg`.

Jak potwierdzamy diagnozę: numer relacji i bloku z komunikatu pozwala zidentyfikować obiekt przez `pg_relation_filepath` odwrócone na nazwę tabeli. Sprawdzamy stan sprzętu (`smartctl`, `dmesg`) niezależnie od samej bazy — uszkodzona strona jest zwykle objawem awarii dysku albo pamięci, nie błędem samego PostgreSQL. Sprawdzamy też, czy replika, opisana na stronie Replikacja strumieniowa, ma tę samą stronę nieuszkodzoną.

Co robimy: jeśli istnieje czysta kopia — backup sprzed uszkodzenia albo replika bez tego samego problemu — odtwarzamy dane stamtąd, co jest zawsze rozwiązaniem preferowanym. Dopiero gdy żadna czysta kopia nie istnieje, a odczyt reszty tabeli jest ważniejszy niż utrata tego jednego fragmentu, sięgamy po `zero_damaged_pages = on` tymczasowo, w jednej sesji, żeby przejść przez uszkodzoną stronę i odczytać resztę danych.

> **Pojęcie: zero_damaged_pages.** Parametr każący silnikowi zastąpić nieczytelną stronę stroną wypełnioną zerami zamiast przerywać działanie błędem — pozwala kontynuować pracę, ale wiersze, które były na tej stronie, znikają bezpowrotnie, bez żadnej możliwości ich odzyskania.

Czego nie robimy: nie włączamy `zero_damaged_pages` jako pierwszego odruchu i nie zostawiamy go trwale włączonego globalnie w `postgresql.conf` — ukrywa to kolejne uszkodzenia zamiast je sygnalizować i przy każdym użyciu trwale gubi dane, więc jest ostatecznością po wyczerpaniu opcji odtworzenia z czystej kopii, nigdy pierwszym krokiem.

Powiązane: VACUUM, autovacuum i bloat · Monitoring instancji · Logi serwera: konfiguracja i czytanie · Replikacja strumieniowa · Rozmiary, tablespace i miejsce na dysku · Indeksy i plany wykonania w PostgreSQL

Źródła: dokumentacja PostgreSQL, rozdział „Reliability and the Write-Ahead Log", „Routine Vacuuming" i „Recovery Configuration"

Revision #40
Created 2026-09-25 00:33:30 UTC by Tomasz Konieczny
Updated 2026-10-26 14:12:59 UTC by Tomasz Konieczny