Co monitorujemy w bazie danych

> Proces silnika widoczny w liście procesów systemu operacyjnego mówi wyłącznie, że coś nasłuchuje na porcie — nic nie mówi o tym, czy baza ma jeszcze miejsce na dysku, czy replika nie zostaje w tyle o godzinę, ani czy backup sprzed tygodnia w ogóle się powiódł.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
MVCC	Multi-Version Concurrency Control: baza trzyma kilka wersji wiersza, żeby czytający nie blokował piszącego
OOM	Out Of Memory: brak pamięci; OOM killer to mechanizm jądra zabijający proces, gdy jej zabraknie
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Fiszka

TYLKO ODCZYT

Liczba połączeń wobec limitu: PostgreSQL

```
SELECT count(*), (SELECT setting FROM pg_settings WHERE name = 'max_connections') AS
limit
FROM pg_stat_activity;
```

Przykład:

```
 count | limit
-------+-------
   142 | 200
```

```
(1 row)
```

🔒 TYLKO ODCZYT

Najdłuższa otwarta transakcja: PostgreSQL

```
SELECT pid, now() - xact_start AS czas_trwania, query
FROM pg_stat_activity
WHERE xact_start IS NOT NULL
ORDER BY xact_start
LIMIT 5;
```

Przykład:

```
  pid  | czas_trwania    | query
-------+-----------------+---------------------------------
 28471 | 00:47:12.884213 | UPDATE zamowienia SET status = ...
(1 row)
```

🔒 TYLKO ODCZYT

Wiek najstarszego niezamrożonego wiersza (wraparound): PostgreSQL

```
SELECT datname, age(datfrozenxid) AS wiek_transakcji
FROM pg_database
ORDER BY wiek_transakcji DESC
LIMIT 5;
```

Przykład:

```
 datname   | wiek_transakcji
-----------+-----------------
 produkcja |       142398201
(1 row)
```

Wartość porównujemy z parametrem `autovacuum_freeze_max_age` (domyślnie 200 milionów) — zbliżanie się do tej granicy jest sygnałem, że `autovacuum` nie nadąża z zamrażaniem starych

wierszy.

□□ TYLKO ODCZYT

Liczba sesji i blokad oczekujących: SQL Server

```
SELECT blocking_session_id, session_id, wait_type, wait_time
FROM sys.dm_exec_requests
WHERE blocking_session_id <> 0;
```

Przykład:

```
blocking_session_id   session_id   wait_type   wait_time
--------------------- ------------ ----------- ----------
54                    61           LCK_M_X     4213
```

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

□□ TYLKO ODCZYT

Wolne miejsce w pliku danych: SQL Server

```
SELECT name, size/128.0 AS rozmiar_mb,
       size/128.0 - CAST(FILEPROPERTY(name, 'SpaceUsed') AS int)/128.0 AS wolne_mb
FROM sys.database_files;
```

Przykład:

```
name           rozmiar_mb   wolne_mb
-------------- ------------ ----------
produkcja      8192.00      612.25
produkcja_log  2048.00      1290.75
```

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

□□ ZMIENIA SYSTEM

Sesje aktywne i długo działające: Oracle

```
SELECT sid, serial#, username, status, last_call_et
FROM v$session
WHERE status = 'ACTIVE' AND last_call_et > 600;
```

`last_call_et` w sekundach — filtr powyżej 600 pokazuje sesje aktywne dłużej niż dziesięć minut.

Przykład:

```
   SID SERIAL# USERNAME       STATUS   LAST_CALL_ET
------ ------- -------------- -------- ------------
   142    5821 APLIKACJA      ACTIVE           1847
```

TYLKO ODCZYT

Wolne miejsce w tablespace: Oracle

```
SELECT tablespace_name, round(sum(bytes)/1024/1024) AS wolne_mb
FROM dba_free_space
GROUP BY tablespace_name;
```

Przykład:

```
TABLESPACE_NAME    WOLNE_MB
------------------ ----------
USERS                    1842
DANE                      412
```

TYLKO ODCZYT

Procesy i wątki oczekujące na blokadę: MySQL

```
SELECT trx_id, trx_state, trx_started, trx_query
FROM information_schema.innodb_trx
WHERE trx_state = 'LOCK WAIT';
```

Przykład:

```
+--------+-----------+--------------------+--------------------------+
| trx_id | trx_state | trx_started        | trx_query                |
+--------+-----------+--------------------+--------------------------+
| 482910 | LOCK WAIT | 2026-09-25 14:02:11 | UPDATE zamowienia SET ... |
+--------+-----------+--------------------+--------------------------+
```

TYLKO ODCZYT

Wolne miejsce w tablespace InnoDB: MySQL

```
SELECT table_schema, round(sum(data_free)/1024/1024) AS wolne_mb
FROM information_schema.tables
GROUP BY table_schema;
```

Przykład:

```
+--------------+----------+
| table_schema | wolne_mb |
+--------------+----------+
| sklep        |      184 |
+--------------+----------+
```

Po co to istnieje

Pojęcie: monitoring bazy danych. Ciągłe zbieranie i ocena metryk opisujących stan silnika, po to żeby wykryć problem, zanim zauważą go użytkownicy aplikacji, albo zanim problem stanie się nieodwracalny. Sprawdzenie „czy proces działa” (`systemctl status`, `ps aux | grep postgres`) odpowiada tylko na jedno, bardzo wąskie pytanie — proces może działać bez zarzutu i jednocześnie odmawiać nowych połączeń, bo limit połączeń jest wyczerpany, może mieć pełny dysk pod tablespace, może mieć replikę opóźnioną o godzinę, albo może od tygodnia nie mieć ani jednego udanego backupu. Żaden z tych stanów nie zmienia statusu procesu w oczach systemu operacyjnego — każdy z nich wymaga osobnej, świadomej metryki.

Cztery warstwy monitoringu. Monitoring bazy danych dzieli się naturalnie na cztery warstwy, z których każda odpowiada na inne pytanie: czy baza w ogóle przyjmuje połączenia (dostępność),

czy starcza jej miejsca i zasobów, żeby działać dalej (pojemność), czy działa wystarczająco szybko dla obecnego obciążenia (wydajność), i czy procesy działające w tle — backup, replikacja, statystyki — wykonują się zgodnie z planem (poprawność procesów tła). Kompletny monitoring pokrywa wszystkie cztery warstwy naraz — monitoring skupiony wyłącznie na dostępności regularnie przegapia powolnie narastające problemy pojemności, które kończą się awarią dostępności dopiero w chwili, gdy jest już za późno na spokojną reakcję.

Jak to działa

Poniższa lista metryk pokrywa wszystkie cztery warstwy z konkretnymi progami, od których warto zaczynać konfigurację alertów — progi te dostosowujemy do własnego środowiska, ale rząd wielkości pozostaje zbliżony niezależnie od silnika.

Metryka	Warstwa	Typowy próg alertu
liczba aktywnych połączeń wobec limitu (`max_connections`)	dostępność	powyżej 80% limitu
najdłuższa otwarta transakcja	dostępność / pojemność	dłuższa niż kilka–kilkanaście minut, zależnie od profilu obciążenia
opóźnienie repliki	dostępność	powyżej kilkudziesięciu sekund dla asynchronicznej
wiek najstarszego udanego backupu	poprawność procesów tła	starszy niż zakładane RPO/RTO dopuszcza
wolne miejsce w tablespace / pliku danych	pojemność	poniżej 15–20% wolnego miejsca
trafienia w cache (bufor stron)	wydajność	poniżej 90–95%, zależnie od profilu
blokady i oczekiwania na blokady	wydajność	sesja czekająca na blokadę dłużej niż kilka sekund
kolejka operacji I/O / czas odpowiedzi dysku	wydajność	rosnąca kolejka albo odpowiedź powyżej kilkunastu milisekund
wiek statystyk optymalizatora	wydajność	brak odświeżenia od dużej zmiany danych

Dodatkowo w PostgreSQL: bloat i wraparound. Pojęcie: bloat. Narastająca ilość martwych, niewidocznych już żadnej transakcji wersji wierszy pozostawionych przez mechanizm MVCC (temat strony MVCC: dlaczego czytający nie blokuje piszącego), które nie zostały jeszcze odzyskane przez `VACUUM` — tabela i jej indeksy fizycznie rosną szybciej niż rośnie liczba żywych wierszy, co spowalnia zarówno odczyty, jak i zapisy. **Pojęcie: wraparound (zawinięcie licznika transakcji).** Zagrożenie wynikające ze skończonej długości licznika identyfikatorów transakcji w PostgreSQL — jeśli `VACUUM` nie zdąży „zamrozić” wystarczająco starych wierszy, zanim licznik dogoni sam siebie, silnik z własnej inicjatywy przechodzi w tryb tylko do odczytu, żeby zapobiec

utracie spójności danych. Monitorowanie wieku najstarszej niezamrożonej transakcji (`age(datfrozenxid)`) jest w PostgreSQL metryką bez odpowiednika w pozostałych trzech silnikach, bo żaden z nich nie zarządza widocznością wierszy przez numerowanie transakcji w ten sam sposób.

Zdarzenia z logu kończące się zawsze alertem

Niektóre zdarzenia, gdy tylko pojawią się w logu silnika, są na tyle poważne, że nie czekamy na próg cykliczny, tylko wysyłamy alert natychmiast po ich wystąpieniu:

- zakleszczenie (deadlock) — sygnał problemu w kolejności blokowania zasobów przez aplikację, temat strony Zakleszczenie (deadlock): jak powstaje i jak je czytać z logu
- brak miejsca na dysku pod plikami danych albo logiem transakcyjnym — silnik przestaje przyjmować zapisy w ciągu sekund od wyczerpania miejsca
- nieudany backup — brak alertu o niepowodzeniu backupu jest dokładnie sytuacją opisaną na stronie Backup pełny, różnicowy, przyrostowy, logi, w której backup, którego nikt nie odtworzył ani nie doglądał, przestaje istnieć w praktyce na długo przed tym, jak ktokolwiek to zauważy
- nieoczekiwany restart instancji — każdy restart poza zaplanowanym oknem serwisowym jest sygnałem awarii sprzętu, zabrakło pamięci (OOM killer), albo błędu samego silnika, wymagającym wyjaśnienia niezależnie od tego, czy baza wstała z powrotem bez interwencji

Konto monitorujące

Konto, którym łączy się system monitoringu, potrzebuje wyłącznie prawa odczytu widoków i tabel systemowych opisujących stan silnika — liczbę sesji, statystyki, blokady, rozmiary — a nie potrzebuje żadnego dostępu do tabel z danymi biznesowymi. W PostgreSQL odpowiada temu rola `pg_monitor` (od wersji 10), łącząca uprawnienia do odczytu większości widoków statystycznych bez dostępu do zawartości tabel użytkownika. W SQL Server rolę serwerową `##MS_ServerStateReader##` albo dedykowane uprawnienie `VIEW SERVER STATE` bez żadnych praw do baz z danymi. W Oracle rolę `SELECT_CATALOG_ROLE` łączoną z dedykowanymi widokami `V$`. Nadanie kontu monitorującemu szerszych uprawnień niż odczyt stanu silnika jest naruszeniem tej samej zasady najmniejszych uprawnień, opisanej na stronie Konta, role i uprawnienia: GRANT, REVOKE, zasada najmniejszych uprawnień — narzędzie monitorujące, które umie tylko patrzeć, nie ma potrzeby umieć czegokolwiek więcej.

Częste błędy

Monitoring tylko dostępności. Sprawdzanie wyłącznie tego, czy port bazy odpowiada, przegapia narastające problemy pojemności i wydajności, które objawiają się awarią dostępności dopiero wtedy, gdy jest już za późno — pełny dysk pod tablespace kończy się dokładnie tym samym efektem co padnięty proces, tylko dociera do niego dłuższą, wcześniej widoczną drogą.

Progi skopiowane z innego środowiska bez dostosowania. Próg alertu na liczbę połączeń albo na opóźnienie repliki, przeniesiony wprost z innej bazy o zupełnie innym profilu obciążenia, generuje albo lawinę fałszywych alertów, albo, gorzej, milczy dokładnie tam, gdzie dla tego konkretnego systemu problem już się zaczął.

Brak alertu na nieudany backup jako oddzielne zdarzenie. Monitoring, który sprawdza tylko, czy proces backupu w ogóle wystartował, a nie czy zakończył się sukcesem i wyprodukował plik nadający się do odtworzenia, jest tożsamy z brakiem monitoringu backupu w ogóle.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — dobór warstwy metryki

Klasyfikujemy trzy zdarzenia do jednej z czterech warstw monitoringu (dostępność, pojemność, wydajność, poprawność procesów tła): rosnąca kolejka oczekujących zapytań do dysku, tablespace zapełniony w 97%, backup logów nie wykonał się od trzech dni.

Odpowiedź

Rosnąca kolejka I/O to warstwa wydajności — system fizycznie zwalnia, choć wciąż działa. Tablespace zapełniony w 97% to warstwa pojemności — miejsce się kończy, choć jeszcze nie skończyło. Brak backupu logów od trzech dni to warstwa poprawności procesów tła — proces, który miał się wykonywać cyklicznie, przestał, i nikt tego jeszcze nie odczuł jako awarii, dopóki nie przyjdzie dzień odtwarzania.

Pytania kontrolne

1. Dlaczego sprawdzenie „proces bazy działa” nie jest wystarczającym monitoringiem?

Odpowiedź

Bo obejmuje tylko jedną, bardzo wąską warstwę — dostępność na poziomie samego procesu — i nie mówi nic o pojemności (miejsce na dysku, limit połączeń), wydajności (blokady, kolejka I/O, nieaktualne statystyki) ani o poprawności procesów tła (backup, replikacja). Proces może działać bez zarzutu i jednocześnie odmawiać nowych połączeń albo mieć od tygodnia niedziałający backup, a monitoring ograniczony do samego procesu nigdy tego nie wykryje.

2. Dlaczego bloat i wraparound są metrykami specyficznymi dla PostgreSQL, bez odpowiednika w pozostałych trzech silnikach?

Odpowiedź

Obie metryki wynikają wprost z mechanizmu MVCC opartego na numerowaniu transakcji i pozostawianiu starych wersji wierszy zamiast nadpisywania ich w miejscu — mechanizmu, który PostgreSQL realizuje w sposób odróżniający go od pozostałych silników. Bloat to skutek nieodzyskanych starych wersji wierszy, a wraparound to ryzyko zawinięcia się skończonego licznika transakcji — oba zjawiska istnieją tylko tam, gdzie widoczność wierszy jest liczona przez porównywanie numerów transakcji, co jest specyfiką architektury PostgreSQL.

Powiązane: Zakleszczenie (deadlock): jak powstaje i jak je czytać z logu · MVCC: dlaczego czytający nie blokuje piszącego · Backup pełny, różnicowy, przyrostowy, logi · Konta, role i uprawnienia: GRANT, REVOKE, zasada najmniejszych uprawnień · Statystyki i optymalizator kosztowy

Źródła: dokumentacja PostgreSQL, rozdział „Monitoring Database Activity” i „Routine Vacuuming” · Microsoft Docs, „Dynamic Management Views” · Oracle Database Reference, widoki `V$SESSION` i `V$DATAGUARD_STATS` · dokumentacja MySQL, rozdział „InnoDB Monitors” i „The INFORMATION_SCHEMA INNODB_TRX Table”

Revision #43
Created 2026-09-24 23:07:49 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:59 UTC by Tomasz Konieczny