Rozpoznanie i diagnostyka

Wersja, parametry, sesje, blokady, obciążenie i scheduler.

- Rozpoznanie instancji: wersja, parametry, komponenty
- Sesje: aktywne, nieaktywne, najdłuższe
- Blokady i obciążenie
- Scheduler i zadania w bazie

Rozpoznanie instancji: wersja, parametry, komponenty

Poziom: podstawy **Dotyczy: Oracle Database 19c** **Czas: ~20 min**

Zanim dotkniemy nieznanej instancji, sprawdzamy trzy rzeczy: jaka to wersja, czy wszystkie komponenty słownika danych są sprawne i z jakiego pliku parametrów instancja aktualnie żyje.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
SGA	System Global Area: wspólna pamięć instancji Oracle
SQL	Structured Query Language: język zapytań do relacyjnych baz danych
WMS	Warehouse Management System: system zarządzania magazynem
XML	eXtensible Markup Language: format tekstowy do zapisu ustrukturyzowanych danych w znacznikach, tutaj używany przez PowerShell do zapisu zrzutu obiektów poleceniem Export-Clixml

Fiszka

TYLKO ODCZYT

Sprawdzenie nazwy, statusu i wersji instancji: v$instance

```
SELECT instance_name, status, version FROM v$instance;
```

Przykład:

```
INSTANCE_NAME    STATUS       VERSION
---------------- ------------ -----------------
WMSPRD           OPEN         19.0.0.0.0
```

TYLKO ODCZYT

Sprawdzenie pełnej wersji komponentów bazy: v$version

```
SELECT * FROM v$version;
```

`v$instance.version` podaje samą wersję silnika, natomiast `v$version` wypisuje pełny banner ze wszystkimi komponentami — przydatne przy rozstrzyganiu, czy mamy do czynienia z edycją Standard czy Enterprise.

Przykład:

```
BANNER
--------------------------------------------------------------------------------
Oracle Database 19c Standard Edition 2 Release 19.22.0.0.0 - Production
PL/SQL Release 19.22.0.0.0 - Production
CORE 19.22.0.0.0 Production
TNS for Linux: Version 19.22.0.0.0 - Production
```

TYLKO ODCZYT

Sprawdzenie komponentów niesprawnych: DBA_REGISTRY

```
SELECT comp_name, version, status FROM dba_registry WHERE status <> 'VALID';
```

Zapytanie, które w zdrowej instancji nie zwraca żadnego wiersza — jeśli zwraca, to mamy konkretny trop, który komponent naprawić, zanim zajmiemy się czymkolwiek innym.

Przykład:

```
no rows selected
```

TYLKO ODCZYT

Sprawdzenie pełnej wersji wszystkich komponentów

```
COLUMN comp_id FOR A10
COLUMN version_full FOR A12
COLUMN status FOR A10
COLUMN modified FOR A21
SELECT comp_id, version_full, status, modified FROM dba_registry;
```

Przykład:

```
COMP_ID    VERSION_FULL STATUS     MODIFIED
---------- ------------ ---------- --------------------
CATALOG    19.22.0.0.0  VALID      25-SEP-26 07.10.02 AM
CATPROC    19.22.0.0.0  VALID      25-SEP-26 07.11.44 AM
JAVAVM     19.22.0.0.0  VALID      25-SEP-26 07.13.20 AM
XDB        19.22.0.0.0  VALID      25-SEP-26 07.14.05 AM
APEX       19.22.0.0.0  VALID      25-SEP-26 07.18.51 AM
```

TYLKO ODCZYT

Przekompilowanie obiektów nieprawidłowych: utlrp.sql

```
@$ORACLE_HOME/rdbms/admin/utlrp.sql
```

Skrypt dostarczany razem z instalacją Oracle, który przechodzi po wszystkich obiektach PL/SQL ze statusem `INVALID` w całej bazie i próbuje je przekompilować równolegle. Uruchamiamy go zwykle zaraz po upgrade'zie wersji albo po nałożeniu patcha — te operacje z reguły zostawiają część pakietów i widoków w stanie `INVALID`, dopóki nie zostaną przekompilowane wobec nowego słownika.

Przykład:

```
SQL> @$ORACLE_HOME/rdbms/admin/utlrp.sql
...
PL/SQL procedure successfully completed.

Timestamp: 25-SEP-2026 08:45:12

DOC> The following query reports the number of invalid objects remaining.
DOC> If the number is not 0, run utlrp.sql again.
```

```
OBJ_COUNT
----------
         0
```

TYLKO ODCZYT

Zrzut wszystkich parametrów bazy: v$parameter

```
SELECT name, value FROM v$parameter;
```

Kompletny dump wszystkich parametrów instancji — przydatny, gdy chcemy przenieść konfigurację na inny serwer albo porównać dwie instancje różnica po różnicy, zamiast pytać o parametry pojedynczo przez `SHOW PARAMETER`.

Przykład:

```
NAME                                 VALUE
------------------------------------ ------------------------------
processes                            2000
sessions                             3305
sga_target                           45G
pga_aggregate_target                 11G
db_block_size                        8192
open_cursors                         300
...
```

TYLKO ODCZYT

Sprawdzenie roli i trybu otwarcia bazy: v$database

```
SELECT 'DATABASE_ROLE', database_role FROM v$database
UNION ALL
SELECT 'OPEN_MODE', open_mode FROM v$database;
```

`database_role` odróżnia bazę podstawową (`PRIMARY`) od standby w konfiguracji Data Guard, a `open_mode` mówi, czy baza jest otwarta do odczytu i zapisu (`READ WRITE`), tylko do odczytu, czy dopiero zamontowana bez otwarcia.

Przykład:

```
'DATABASE_ROLE'                DATABASE_ROLE
------------------------------ ------------------------------
DATABASE_ROLE                   PRIMARY
OPEN_MODE                       READ WRITE
```

□□ TYLKO ODCZYT

Sprawdzenie, czy instancja startuje z pfile czy spfile

```
SELECT DECODE(value, NULL, 'PFILE', 'SPFILE') "Init File Type"
FROM sys.v_$parameter WHERE name = 'spfile';
```

Jeśli parametr `spfile` ma wartość `NULL`, instancja została uruchomiona z tradycyjnego pliku tekstowego (`pfile`) — inaczej wartością jest ścieżka do faktycznie użytego `spfile`.

Przykład:

```
Init File Type
--------------
SPFILE
```

□□ ZMIENIA SYSTEM

Utworzenie pfile na podstawie aktualnego spfile

```
CREATE PFILE FROM SPFILE;
```

Zrzuca bieżące parametry ze `spfile` binarnego do zwykłego pliku tekstowego w domyślnej lokalizacji `$ORACLE_HOME/dbs/initNazwaSID.ora` — przydaje się, gdy chcemy ręcznie przejrzeć albo zedytować parametry poza sesją SQL, albo mieć czytelną kopię zapasową konfiguracji przed większą zmianą.

Przykład:

```
SQL> CREATE PFILE FROM SPFILE;


File created.
```

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

□□ TYLKO ODCZYT

Sprawdzenie aktualnego limitu procesów

```
SELECT value FROM v$parameter WHERE name = 'processes';
```

Przykład:

```
VALUE
------------------------------
2000
```

□□ ZMIENIA SYSTEM

Ustawienie maksymalnej liczby procesów

```
ALTER SYSTEM SET processes = 2000 SCOPE=SPFILE;
```

`PROCESSES` nie da się zmienić w locie — zmiana wymaga `SCOPE=SPFILE` i restartu instancji, bo liczba procesów decyduje o rozmiarze struktur alokowanych w SGA już przy starcie.

Przykład:

```
SQL> ALTER SYSTEM SET processes = 2000 SCOPE=SPFILE;


System altered.
```

□□ **SGA** = System Global Area: wspólna pamięć instancji Oracle · **SET** = Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP

□□ TYLKO ODCZYT

Sprawdzenie bieżącego i maksymalnego wykorzystania sesji oraz procesów: v$resource_limit

```
SELECT resource_name, current_utilization, max_utilization, limit_value
FROM v$resource_limit
WHERE resource_name IN ('sessions', 'processes');
```

`v$resource_limit` różni się od samego sprawdzenia parametru tym, że pokazuje nie tylko limit, ale i to, ile z niego faktycznie wykorzystano — `max_utilization` to szczyt od ostatniego restartu instancji, przydatny sygnał, czy zbliżamy się do limitu `PROCESSES` w normalnym ruchu, czy tylko incydentalnie.

Przykład:

```
RESOURCE_NAME        CURRENT_UTILIZATION MAX_UTILIZATION LIMIT_VALUE
-------------------- ------------------- -------------- ------------
sessions                             412            890         3305
processes                            298            605         2000
```

Gdy `MAX_UTILIZATION` zbliża się do `LIMIT_VALUE`, kolejna sesja aplikacyjna kończy się błędem `ORA-00018: maximum number of sessions exceeded` albo `ORA-00020: maximum number of processes exceeded` — sygnał, że parametr `PROCESSES` wymaga podniesienia, zanim, a nie po tym, jak zacznie brakować.

Z życia wzięte

□□ Z ŻYCIA WZIĘTE

Po nocnej przerwie prądu Tomek loguje się na ora01 i nie wie, czy instancja WMSPRD w ogóle wstała poprawnie po awarii. Zapytanie do `v$instance` pokazuje status OPEN i wersję 19.22, ale `DBA_REGISTRY` zwraca jeden wiersz ze statusem INVALID po nieudokończonym patchu sprzed miesiąca. Marek dzwoni pół godziny później, że raporty WMS liczą się wolniej niż zwykle, a przyczyną okazuje się właśnie ten sam, nieprzekompilowany komponent.

Wersja i tożsamość instancji

Pojęcie: v$instance. Widok dynamiczny opisujący samą instancję — jej nazwę, status, wersję silnika, czas uruchomienia. W odróżnieniu od `v$database`, który opisuje bazę danych

jako zbiór plików, `v$instance` mówi o tym, co żyje akurat w pamięci.

Walidacja i wersje komponentów słownika

Pojęcie: DBA_REGISTRY. Widok słownikowy trzymający listę wszystkich komponentów Oracle zainstalowanych w bazie (Java, XML DB, APEX, Spatial i tak dalej) razem z ich statusem. Status `VALID` oznacza komponent sprawny; `INVALID` oznacza obiekty PL/SQL komponentu wymagające przekompilowania — typowy skutek niedokończonego upgrade'u albo patcha.

Plik parametrów: spfile czy pfile

Sposób, w jaki instancja wystartowała, ma znaczenie praktyczne przy każdej zmianie parametru — opisane szerzej przy okazji `SCOPE` w `ALTER SYSTEM` na stronie Architektura Oracle: instancja, baza, SGA, PGA, procesy tła.

Limity procesów i sesji

Pojęcie: PROCESSES. Parametr instancji określający maksymalną liczbę procesów systemu operacyjnego, jakie instancja może otworzyć jednocześnie — każda sesja dedykowana to jeden proces, więc ten parametr w praktyce ogranicza liczbę równoległych połączeń.

Pojęcie: SESSIONS. Maksymalna liczba jednoczesnych sesji logicznych. Domyślnie silnik wylicza go automatycznie z `PROCESSES` (w przybliżeniu `1,5 x PROCESSES + 22`), ale można go też ustawić ręcznie.

Częste błędy

Podniesienie tylko SESSIONS bez PROCESSES. `SESSIONS` jest pochodną `PROCESSES` — samo podniesienie `SESSIONS` bez odpowiedniego zapasu w `PROCESSES` nic nie da, bo to liczba procesów systemowych jest twardym ograniczeniem fizycznym.

Oczekiwanie, że ALTER SYSTEM SET processes zadziała od razu. Bez restartu instancji zmiana siedzi tylko w `spfile` — `SHOW PARAMETER processes` w bieżącej sesji nadal pokaże starą wartość, dopóki instancja nie zostanie zrestartowana.

Ignorowanie DBA_REGISTRY po upgrade'zie. Komponent w stanie `INVALID` po migracji wersji potrafi cicho psuć funkcjonalność zależną od niego (np. `XDB`, `APEX`) bez żadnego komunikatu błędu w aplikacji — dopóki ktoś nie sprawdzi rejestru wprost.

Powiązane: Architektura Oracle: instancja, baza, SGA, PGA, procesy tła · SQL*Plus: klient i jego konfiguracja · Sesje: aktywne, nieaktywne, najdłuższe

Źródła: notatki własne, dokumentacja Oracle Database Reference 19c

Sesje: aktywne, nieaktywne, najdłuższe

Poziom: podstawy **Dotyczy: Oracle Database 19c** **Czas: ~20 min**

Zgłoszenie „baza się zawiesiła" prawie zawsze zaczyna się od tego samego widoku — `v$session` — i kończy tym samym poleceniem — `ALTER SYSTEM KILL SESSION`. Ta strona pokazuje całą drogę pomiędzy tymi dwoma punktami.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
PID	Process ID: numer procesu w systemie
RMAN	Recovery Manager: narzędzie Oracle do kopii zapasowych
SGA	System Global Area: wspólna pamięć instancji Oracle
WMS	Warehouse Management System: system zarządzania magazynem

Fiszka

TYLKO ODCZYT

Sprawdzenie aktywnych sesji

```
SELECT username FROM v$session WHERE username IS NOT NULL ORDER BY username ASC;
```

Lista wszystkich zalogowanych użytkowników — pierwszy, najprostszy rzut oka na to, kto w ogóle jest połączony do instancji, zanim zejdziemy do szczegółów pojedynczej sesji.

Przykład:

```
USERNAME
------------------------------
```

```
APP_USER
APP_USER
REPORTS_USER
SYSTEM
ZABBIX
```

ASC = Automatic Self-Calibration: automatyczna samokalibracja czujnika CO2, która zakłada, że co tydzień czujnik zobaczy czyste powietrze o stężeniu około 400 ppm

TYLKO ODCZYT

Sprawdzenie SID i SERIAL# wszystkich sesji

```
SELECT sid, serial#, username, command, program FROM v$session;
```

Przykład:

```
       SID    SERIAL# USERNAME             COMMAND PROGRAM
---------- ---------- -------------------- ------- ------------------------------
       142       5501 APP_USER                     3 JDBC Thin Client
       201       8832 REPORTS_USER                 3 sqlplus@app01
       305         12 ZABBIX                        3 zbx_monitor
```

SID = w Oracle: System Identifier, nazwa instancji; w Windows: Security Identifier, numer konta

TYLKO ODCZYT

Sprawdzenie sesji nieaktywnych

```
SELECT sid, serial#, status, program, username FROM v$session WHERE status = 'INACTIVE';
```

Sesja `INACTIVE` ma otwarte połączenie, ale w danym momencie nic nie wykonuje — z reguły to aplikacja trzymająca połączenie w puli, nie problem sam w sobie. Dużo takich sesji naraz bywa jednak sygnałem źle skonfigurowanego connection poolingu po stronie aplikacji.

Przykład:

```
       SID    SERIAL# STATUS   PROGRAM                          USERNAME
---------- ---------- -------- ------------------------------ --------------------
        88       2210 INACTIVE JDBC Thin Client                APP_USER
```

```
       112       4471 INACTIVE JDBC Thin Client                APP_USER
```

◫◫ TYLKO ODCZYT

Wyświetlenie informacji o procesie na podstawie SID i SERIAL

```
SELECT p.spid, s.sid, s.serial#, s.username, s.machine, s.program
FROM v$session s
JOIN v$process p ON (s.paddr = p.addr)
WHERE s.sid = NumerSID AND s.serial# = NumerSerial;
```

Łączymy `v$session` z `v$process` przez `paddr`/`addr`, żeby dojść od logicznej sesji Oracle do fizycznego procesu systemu operacyjnego, który ją obsługuje — `spid` w wyniku to właśnie systemowy PID, ten sam, który zobaczymy w wyniku `ps -ef`.

Przykład:

```
SPID          SID    SERIAL# USERNAME             MACHINE          PROGRAM
--------- ---------- ---------- -------------------- ----------------
------------------------------
28471            142       5501 APP_USER             app01        JDBC Thin Client
```

◫◫ **SID** = w Oracle: System Identifier, nazwa instancji; w Windows: Security Identifier, numer konta · **PID** = Process ID: numer procesu w systemie · **join** = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

◫◫ TYLKO ODCZYT

Sprawdzenie najdłużej wykonujących się kwerend

```
COLUMN sql_id FORMAT A15
COLUMN username FORMAT A15
COLUMN sql_text FORMAT A40
COLUMN event FORMAT A20
SELECT s.sid, s.serial#, s.username, s.sql_id, s.last_call_et AS elapsed_seconds,
       s.status, s.event, q.sql_text
FROM v$session s
JOIN v$sql q ON s.sql_id = q.sql_id
WHERE s.status = 'ACTIVE'
ORDER BY s.last_call_et DESC
```

```
FETCH FIRST 10 ROWS ONLY;
```

`last_call_et` to liczba sekund, jaka minęła od ostatniej interakcji sesji z instancją — dla sesji `ACTIVE` w praktyce oznacza to czas trwania bieżącego zapytania. Sortujemy malejąco, żeby najdłużej trwające zapytanie znalazło się na górze wyniku.

Przykład:

```
       SID    SERIAL# USERNAME        SQL_ID          ELAPSED_SECONDS STATUS
EVENT                SQL_TEXT
---------- ---------- -------------- -------------- -------------- --------
------------------- ----------------------------------------
       142       5501 APP_USER        7fzg1k3mn82qw        872 ACTIVE   db file
sequential read UPDATE zamowienia SET status = 'zamkniete'...
```

join = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

ZMIENIA SYSTEM

Sprawdzenie najdłużej działających programów: v$session_longops

```
SELECT s.sid, s.serial#, s.username, s.osuser, s.program, s.machine,
       l.elapsed_seconds, l.start_time, l.last_update_time, l.target_desc, l.sofar,
l.totalwork
FROM v$session s, v$session_longops l
WHERE s.sid = l.sid
  AND s.username IS NOT NULL
  AND l.elapsed_seconds > 60
ORDER BY l.elapsed_seconds DESC;
```

> **Pojęcie: v$session_longops.** Widok dedykowany operacjom, które z natury trwają długo i potrafią raportować postęp w trakcie wykonania — backup RMAN, przebudowa indeksu, `CREATE TABLE ... AS SELECT` na dużej tabeli. `sofar` i `totalwork` to jednostki postępu specyficzne dla danej operacji — ich iloraz razy 100 daje procent ukończenia.

Przykład:

```
       SID    SERIAL# USERNAME  OSUSER  PROGRAM         MACHINE     ELAPSED_SECONDS
TARGET_DESC          SOFAR TOTALWORK
```

```
---------- ---------- --------- ------- --------------- ----------- ---------------
------------------- ----- ---------
       305         12 SYSTEM    oracle  rman@ora01   ora01               4120 RMAN: full
datafile  71230    112000
```

RMAN = Recovery Manager: narzędzie Oracle do kopii zapasowych

TYLKO ODCZYT

Sprawdzenie historii aktywności konkretnych sesji: v$active_session_history

```
SELECT a.sid, a.sql_id, a.sql_text
FROM v$active_session_history a
WHERE a.sid IN (NumerSID)
ORDER BY a.sid, a.sample_time;
```

`v$active_session_history` próbkuje stan aktywnych sesji co sekundę i trzyma krótką historię w pamięci — użyteczne, gdy sesja już się zakończyła, ale chcemy odtworzyć, co dokładnie robiła tuż przed problemem, zanim zniknęła z `v$session`.

Przykład:

```
       SID SQL_ID          SQL_TEXT
---------- --------------- ----------------------------------------
       142 7fzg1k3mn82qw   UPDATE zamowienia SET status = 'zamkniete'...
       142 7fzg1k3mn82qw   UPDATE zamowienia SET status = 'zamkniete'...
```

ZMIENIA SYSTEM

Zakończenie sesji: ALTER SYSTEM KILL SESSION z IMMEDIATE

```
ALTER SYSTEM KILL SESSION 'NumerSID,NumerSerial' IMMEDIATE;
```

Przykład:

```
SQL> ALTER SYSTEM KILL SESSION '142,5501' IMMEDIATE;


System altered.
```

⚠ ZMIENIA SYSTEM

Zakończenie sesji bez IMMEDIATE

```
ALTER SYSTEM KILL SESSION 'NumerSID,NumerSerial';
```

Sesja zostaje oznaczona do zamknięcia, ale faktyczne zwolnienie zasobów następuje dopiero, gdy sesja sama dojdzie do bezpiecznego punktu — do tego czasu w `v$session.status` widzimy `KILLED`, a nie natychmiastowe zniknięcie wiersza.

Przykład:

```
SQL> ALTER SYSTEM KILL SESSION '142,5501';


System altered.
```

⚠ ZMIENIA SYSTEM

Rozłączenie sesji: ALTER SYSTEM DISCONNECT SESSION

```
ALTER SYSTEM DISCONNECT SESSION 'NumerSID,NumerSerial' IMMEDIATE;
```

> **Pojęcie: KILL SESSION kontra DISCONNECT SESSION.** `KILL SESSION` zamyka sesję logiczną, ale przy sesjach dedykowanych proces systemowy po stronie serwera potrafi jeszcze przez chwilę istnieć, zanim faktycznie zniknie. `DISCONNECT SESSION` idzie krok dalej i zamyka też samo połączenie sieciowe do klienta — używamy go, gdy `KILL SESSION` nie wystarcza, bo sesja jest zawieszona na poziomie samego połączenia (np. martwy socket po awarii sieci), a nie na poziomie wykonywanego zapytania.

Przykład:

```
SQL> ALTER SYSTEM DISCONNECT SESSION '142,5501' IMMEDIATE;


System altered.
```

⚠ ZMIENIA SYSTEM

Zakończenie wszystkich sesji danego użytkownika

```
BEGIN
  FOR r IN (SELECT sid, serial# FROM v$session WHERE username = 'NazwaUsera') LOOP
    EXECUTE IMMEDIATE 'ALTER SYSTEM KILL SESSION ''' || r.sid || ',' || r.serial# || '''
IMMEDIATE';
  END LOOP;
END;
/
```

Blok PL/SQL zamiast pojedynczej komendy, bo `ALTER SYSTEM KILL SESSION` przyjmuje dokładnie jedną parę `SID,SERIAL#` na wywołanie — pętla po `v$session` odfiltrowanym do jednego użytkownika zabija wszystkie jego sesje naraz, bez ręcznego przepisywania każdej pary z osobna.

Przykład:

```
SQL> BEGIN
  2    FOR r IN (SELECT sid, serial# FROM v$session WHERE username = 'APP_USER') LOOP
  3      EXECUTE IMMEDIATE 'ALTER SYSTEM KILL SESSION ''' || r.sid || ',' || r.serial#
|| ''' IMMEDIATE';
  4    END LOOP;
  5  END;
  6  /

PL/SQL procedure successfully completed.
```

□□ **SID** = w Oracle: System Identifier, nazwa instancji; w Windows: Security Identifier, numer konta · **execute** = FortiOS: polecenia, które coś wykonują na urządzeniu, na przykład execute backup config wysyła kopię konfiguracji, a execute ha manage otwiera sesję na drugiej zaporze w parze

□□ **TYLKO ODCZYT**

Sprawdzenie systemowego PID procesu bazy danych

```
SELECT p.spid
FROM v$session s JOIN v$process p ON s.paddr = p.addr
WHERE s.sid = NumerSID;
```

Przykład:

```
SPID
------------------------
28471
```

□□ **PID** = Process ID: numer procesu w systemie

□□ OSTROŻNIE · NIEODWRACALNE

Ubicie procesu systemowego: kill -9

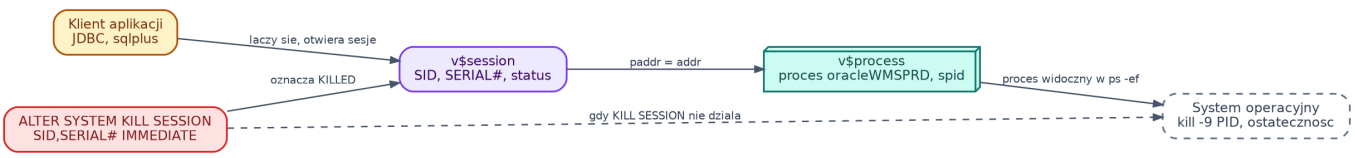
```
kill -9 NumerPID
```

Sięgamy po `kill -9` na procesie systemowym Oracle tylko wtedy, gdy `ALTER SYSTEM KILL SESSION IMMEDIATE` i `ALTER SYSTEM DISCONNECT SESSION` obie zawiodły, a proces nadal wisi — to znaczy, że sam mechanizm zamykania sesji wewnątrz instancji nie działa, nie że proces jest po prostu wolny. `kill -9` nie daje procesowi Oracle szansy na uporządkowane zwolnienie swoich struktur w SGA — proces PMON musi potem posprzątać po nim ręcznie, co dla pojedynczej sesji zwykle przebiega bez problemu, ale w skrajnym przypadku, przy uszkodzonych strukturach pamięci współdzielonej, potrafi doprowadzić do niestabilności całej instancji, a nie tylko jednej sesji. Traktujemy to jako ostateczność, nie pierwszy odruch.

□□ **SGA** = System Global Area: wspólna pamięć instancji Oracle · **kill** = wyślij sygnał do procesu, domyślnie SIGTERM proszący o zamknięcie

Widok v$session

Pojęcie: v$session. Widok dynamiczny z jednym wierszem na każde połączenie do instancji — identyfikator sesji (`SID`), numer seryjny (`SERIAL#`), użytkownik, program kliencki, status i wiele innych atrybutów. To jest odpowiednik `pg_stat_activity` z PostgreSQL, tylko z inną nazwą i inną kolumną identyfikującą sesję.



Sesja logiczna z v$session łączy się z fizycznym procesem systemowym przez v$process, a KILL SESSION działa najpierw na sesji, dopiero potem na procesie (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: SID i SERIAL#.** `SID` to numer sesji, unikalny w danym momencie, ale wielokrotnego użytku — po zakończeniu sesji Oracle prędzej czy później przydzieli ten sam `SID` nowemu połączeniu. `SERIAL#` to numer seryjny, który razem z `SID` jednoznacznie identyfikuje konkretną sesję nawet po tym, jak stary `SID` zostanie ponownie użyty. Każda komenda administracyjna działająca na konkretnej sesji — zakończenie, sprawdzenie procesu — wymaga obu wartości naraz, właśnie po to, żeby nie trafić przypadkiem w zupełnie inną, nowszą sesję.

Kończenie sesji

> **Pojęcie: ALTER SYSTEM KILL SESSION.** Polecenie administracyjne, które oznacza sesję do zamknięcia. Bez `IMMEDIATE` Oracle czeka na naturalny punkt zakończenia bieżącej operacji sesji i oznacza ją jako `KILLED` do tego czasu — sesja może w tym stanie wisieć jeszcze przez chwilę. Z `IMMEDIATE` wycofuje bieżącą transakcję i zwalnia zasoby sesji od razu.

Częste błędy

KILL SESSION bez IMMEDIATE i oczekiwanie natychmiastowego efektu. Sesja przechodzi w stan `KILLED`, ale nie znika z `v$session` od razu — dopiero gdy sama dojdzie do punktu, w którym instancja może bezpiecznie zwolnić jej zasoby. Przy sesji zawieszonej w oczekiwaniu na blokadę to może nigdy nie nastąpić bez `IMMEDIATE`.

Pomylenie starego SID z nowym. `SID` jest wielokrotnego użytku — komenda wykonana na `SID` bez sprawdzenia aktualnego `SERIAL#` potrafi trafić w zupełnie inną, świeżo nawiązaną sesję, jeśli między odczytaniem `SID` a wykonaniem komendy minęło zbyt dużo czasu.

Odruchowe `kill -9` zamiast ALTER SYSTEM KILL SESSION. Zabicie procesu systemowego z pominięciem mechanizmów instancji jest zawsze rozwiązaniem awaryjnym, nie standardową procedurą kończenia sesji — najpierw próbujemy zamknąć sesję na poziomie samej bazy.

Z życia wzięte

□□ Z ŻYCIA WZIĘTE

O 6:10 Marek zgłasza, że skanery w Swarzędzu wiszą przy logowaniu do WMS. Tomek sprawdza `v$session` na WMSPRD i widzi kilkadziesiąt sesji APP_USER w stanie ACTIVE, z jedną trzymającą to samo zapytanie od dwudziestu minut. Po znalezieniu jej systemowego PID przez `v$process` i zabiciu jej poleceniem `ALTER SYSTEM KILL SESSION IMMEDIATE` kolejka skanerów rusza z miejsca.

Powiązane: Rozpoznanie instancji: wersja, parametry, komponenty · Blokady i obciążenie · Scheduler i zadania w bazie · Architektura Oracle: instancja, baza, SGA, PGA, procesy tła

Źródła: notatki własne, dokumentacja Oracle Database Reference 19c

Blokady i obciążenie

Poziom: podstawy **Dotyczy: Oracle Database 19c** **Czas: ~20 min**

Zapytanie, które nagle wisi bez żadnego oczywistego powodu, prawie zawsze czeka na blokadę trzymaną przez inną sesję — a sesja zjadająca cały procesor rzadko zgłasza się sama, więc obie rzeczy sprawdzamy zapytaniem, nie domysłem.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
CPU	Central Processing Unit: procesor
RAC	Real Application Clusters: klaster Oracle, kilka serwerów na jedną bazę
WMS	Warehouse Management System: system zarządzania magazynem

Fiszka

TYLKO ODCZYT

Sprawdzenie blokujących sesji przez blocking_session

```
SELECT sid, serial#, username, blocking_session, status
FROM v$session
WHERE blocking_session IS NOT NULL;
```

Kolumna `blocking_session` w `v$session` to najszybsza droga do znalezienia, kto na kogo czeka — Oracle wylicza ją sam, bez potrzeby ręcznego łączenia `v$lock` z samą sobą. Prawidłowy wynik w zdrowej bazie to `no rows selected`.

Przykład:

```
       SID    SERIAL# USERNAME             BLOCKING_SESSION STATUS
---------- ---------- -------------------- ---------------- --------
       201       8832 REPORTS_USER                      142 ACTIVE
```

Zdarza się, że to zapytanie nie zwraca żadnych blokad, mimo że problem z blokowaniem faktycznie występuje — dzieje się tak, gdy sesja blokująca należy do innej instancji w konfiguracji RAC, albo gdy blokada jest typu, którego `blocking_session` nie wylicza wprost. W takich przypadkach sięgamy po pełniejszy, klasyczny łańcuch blokad niżej.

RAC = Real Application Clusters: klaster Oracle, kilka serwerów na jedną bazę

ZMIENIA SYSTEM

Łańcuch blokad przez v$lock

```
SELECT s1.username || ' (SID=' || s1.sid || ') is blocking ' ||
       s2.username || ' (SID=' || s2.sid || ')' AS blokada
FROM v$lock l1
JOIN v$session s1 ON s1.sid = l1.sid
JOIN v$lock l2 ON l1.id1 = l2.id1 AND l1.id2 = l2.id2 AND l2.request > 0
JOIN v$session s2 ON s2.sid = l2.sid
WHERE l1.block = 1;
```

Pojęcie: v$lock. Widok niskopoziomowy trzymający jeden wiersz na każdą blokadę trzymaną albo żądaną przez sesję — kolumna `block = 1` oznacza, że dana sesja blokuje kogoś innego, a `request > 0` oznacza, że dana sesja czeka na blokadę. Łączymy tabelę samą ze sobą po identyfikatorach zasobu (`id1`, `id2`), żeby sparować sesję trzymającą blokadę z sesją na nią czekającą.

Przykład:

```
BLOKADA
--------------------------------------------------------------------------------
APP_USER (SID=142) is blocking REPORTS_USER (SID=201)
```

SID = w Oracle: System Identifier, nazwa instancji; w Windows: Security Identifier, numer konta · **join** = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

TYLKO ODCZYT

Sprawdzenie obiektów zablokowanych przez daną sesję

```
SELECT b.sid, b.serial#, b.username, c.owner, c.object_name, c.object_type
FROM v$locked_object a, v$session b, dba_objects c
```

```
WHERE a.session_id = b.sid
  AND a.object_id = c.object_id
  AND a.session_id = NumerSID;
```

> **Pojęcie: v$locked_object.** Widok łączący blokadę z konkretnym obiektem bazy — tabelą, indeksem — zamiast z samym identyfikatorem zasobu jak `v$lock`. Odpowiadamy tym zapytaniem na pytanie „na jakich dokładnie tabelach ta sesja trzyma blokadę", nie tylko „czy kogoś blokuje".

Przykład:

```
       SID    SERIAL# USERNAME             OWNER      OBJECT_NAME          OBJECT_TYPE
---------- ---------- -------------------- ---------- -------------------- ------------
       142       5501 APP_USER             APP_OWNER  ZAMOWIENIA           TABLE
```

TYLKO ODCZYT

Sprawdzenie sesji zużywających najwięcej CPU

```
SELECT s.sid, s.serial#, s.username, s.program, s.status, ROUND(st.value/100) AS
cpu_usage_seconds
FROM v$session s, v$sesstat st, v$statname sn
WHERE s.sid = st.sid
  AND st.statistic# = sn.statistic#
  AND sn.name LIKE '%CPU used by this session%'
ORDER BY st.value DESC
FETCH FIRST 20 ROWS ONLY;
```

> **Pojęcie: v$sesstat i v$statname.** `v$sesstat` trzyma wartości liczników statystycznych dla każdej sesji, ale same liczniki są zakodowane numerycznie (`statistic#`) — `v$statname` tłumaczy ten numer na czytelną nazwę, tu akurat na `CPU used by this session`. Wartość jest wyrażona w setnych częściach sekundy, stąd dzielenie przez 100 w zapytaniu.

Przykład:

```
       SID    SERIAL# USERNAME             PROGRAM                          STATUS
CPU_USAGE_SECONDS
---------- ---------- -------------------- -------------------------------- --------
-----------------
       142       5501 APP_USER             JDBC Thin Client
ACTIVE                   842
       305         12 ZABBIX               zbx_monitor
ACTIVE                     4
```

Ta lista jest naturalnym punktem wyjścia, gdy monitoring albo `top` na poziomie systemu operacyjnego pokazuje wysokie zużycie CPU przez procesy Oracle (`oracleNazwaSID`), ale nie mówi wprost, która sesja logiczna za tym stoi — łączymy `spid` z wyniku `ps` z `v$process`/`v$session`, tak jak opisano na stronie Sesje: aktywne, nieaktywne, najdłuższe, albo idziemy od razu przez to zapytanie, jeśli podejrzewamy problem na poziomie samej bazy.

CPU = Central Processing Unit: procesor

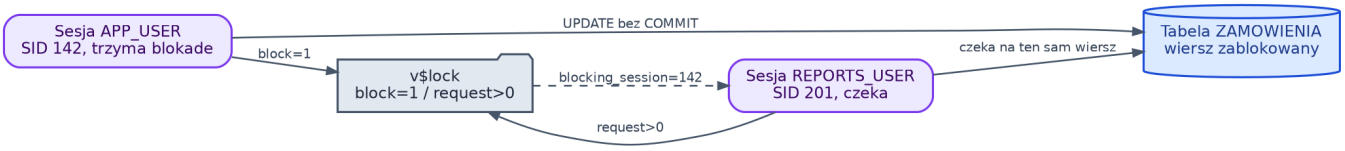
Z życia wzięte

Z ŻYCIA WZIĘTE

Kwadrans po starcie zmiany w Swarzędzu Marek pisze, że WMS zawiesza się przy zamykaniu zamówień. Tomek odpytuje `blocking_session` w `v$session` na WMSPRD i znajduje sesję REPORTS_USER czekającą na APP_USER, który od rana trzyma otwartą transakcję bez commitu. Po sprawdzeniu, że to nie jest legalna operacja wsadowa, zespół kończy sesję blokującą i kolejka zamówień odblokowuje się w kilka sekund.

Blokady w bazie

Pojęcie: blokada (lock). Mechanizm, przez który Oracle chroni spójność danych — sesja modyfikująca wiersz trzyma na nim blokadę wyłączną, dopóki nie wykona `COMMIT` albo `ROLLBACK`, a każda inna sesja próbująca zmodyfikować ten sam wiersz musi czekać. Ogólny mechanizm blokad, ich typy i eskalację opisuje strona Blokady: współdzielone, wyłączne, eskalacja — tu zajmujemy się tym, jak taką sytuację zdiagnozować w konkretnej instancji Oracle.



Sesja APP_USER trzyma blokadę na wierszu tabeli ZAMOWIENIA, sesja REPORTS_USER czeka na ten sam wiersz, a v$lock pozwala sparować obie strony blokady (kliknięcie otwiera pełny rozmiar)

Częste błędy

Poleganie wyłącznie na blocking_session. Kolumna działa dobrze dla zwykłych blokad wierszowych w pojedynczej instancji, ale nie pokrywa wszystkich przypadków (RAC, niektóre blokady systemowe) — jeśli podejrzewamy blokadę, a `blocking_session` milczy, sięgamy po pełne zapytanie na `v$lock`.

Zabicie sesji blokującej bez sprawdzenia, co robi. Sesja trzymająca blokadę często jest w środku długiej, ale legalnej transakcji wsadowej — `ALTER SYSTEM KILL SESSION` na niej cofnie całą pracę tej transakcji. Zanim zakończymy sesję blokującą, sprawdzamy jej zapytanie i czas trwania metodami ze strony Sesje: aktywne, nieaktywne, najdłuższe.

Traktowanie wysokiego CPU_USAGE_SECONDS jako wartości bieżącej. To licznik narastający od początku sesji, nie chwilowe zużycie — sesja otwarta od tygodnia z dużą wartością niekoniecznie zużywa dużo CPU teraz, tylko zdążyła nazbierać tę wartość w czasie.

Powiązane: Blokady: współdzielone, wyłączne, eskalacja · Sesje: aktywne, nieaktywne, najdłuższe · Zakleszczenie (deadlock): jak powstaje i jak je czytać z logu

Źródła: notatki własne, dokumentacja Oracle Database Reference 19c

Scheduler i zadania w bazie

Poziom: podstawy **Dotyczy: Oracle Database 19c** **Czas: ~15 min**

Zadanie, które przestało się uruchamiać, zwykle nie ma nic wspólnego z samym zadaniem — winny jest koordynator schedulera, który trzeba zrestartować, albo parametr `JOB_QUEUE_PROCESSES` ustawiony na zero.

Fiszka

TYLKO ODCZYT

Sprawdzenie listy zadań schedulera

```
SELECT job_name, state, enabled, last_start_date, next_run_date
FROM dba_scheduler_jobs
ORDER BY job_name;
```

Przykład:

```
JOB_NAME                       STATE           ENABLED
LAST_START_DATE                   NEXT_RUN_DATE
------------------------------ --------------- -------
---------------------------------- ----------------------------------
JOB_EKSPORT_RAPORTOW             SCHEDULED       TRUE    25-SEP-26 02.00.11.000000 AM
+02:00 26-SEP-26 02.00.00.000000 AM +02:00
JOB_CZYSZCZENIE_LOGOW            RUNNING         TRUE    25-SEP-26 08.00.02.000000 AM
+02:00 26-SEP-26 08.00.00.000000 AM +02:00
```

TYLKO ODCZYT

Sprawdzenie zadań aktualnie uruchomionych

```
SELECT job_name, session_id, running_instance, elapsed_time
FROM dba_scheduler_running_jobs;
```

Przykład:

```
JOB_NAME                       SESSION_ID RUNNING_INSTANCE ELAPSED_TIME
------------------------------ ---------- ---------------- ---------------
JOB_CZYSZCZENIE_LOGOW                 305                1 +00 00:12:34.0
```

TYLKO ODCZYT

Sprawdzenie czasu działania konkretnego joba przez v$session_longops

```
SELECT s.sid, s.serial#, s.username, l.elapsed_seconds, l.sofar, l.totalwork
FROM v$session s, dba_scheduler_running_jobs j, v$session_longops l
WHERE s.sid = j.session_id
  AND s.sid = l.sid
  AND j.job_name = 'NazwaJobu';
```

Łączymy trzy widoki naraz, żeby dla konkretnej, znanej z góry nazwy joba dostać nie tylko fakt, że działa, ale i szczegółowy postęp z `v$session_longops`, opisanego szerzej na stronie Sesje: aktywne, nieaktywne, najdłuższe.

Przykład:

```
       SID    SERIAL# USERNAME             ELAPSED_SECONDS      SOFAR  TOTALWORK
---------- ---------- -------------------- --------------- ---------- ----------
       305         12 SYSTEM                         1284      84200     120000
```

TYLKO ODCZYT

Zatrzymanie zadania: DBMS_SCHEDULER.STOP_JOB

```
EXEC DBMS_SCHEDULER.STOP_JOB('NazwaJobu');
```

Przerywa działające zadanie i ustawia jego stan na `STOPPED` — job pozostaje zdefiniowany, tylko przestaje się wykonywać do czasu ponownego uruchomienia albo najbliższego uruchomienia wg harmonogramu.

Przykład:

```
SQL> EXEC DBMS_SCHEDULER.STOP_JOB('JOB_CZYSZCZENIE_LOGOW');

PL/SQL procedure successfully completed.
```

□□ **exec** = zastąp bieżący proces powłoki wskazanym programem, bez tworzenia nowego procesu

□□ **TYLKO ODCZYT**

Uruchomienie zadania ręcznie: DBMS_SCHEDULER.RUN_JOB

```
EXEC DBMS_SCHEDULER.RUN_JOB('NazwaJobu');
```

Przykład:

```
SQL> EXEC DBMS_SCHEDULER.RUN_JOB('JOB_CZYSZCZENIE_LOGOW');

PL/SQL procedure successfully completed.
```

□□ **exec** = zastąp bieżący proces powłoki wskazanym programem, bez tworzenia nowego procesu

□□ **TYLKO ODCZYT**

Wyłączenie schedulera: SCHEDULER_DISABLED na TRUE

```
EXEC DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('SCHEDULER_DISABLED', 'TRUE');
```

Przykład:

```
SQL> EXEC DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('SCHEDULER_DISABLED', 'TRUE');

PL/SQL procedure successfully completed.
```

□□ **exec** = zastąp bieżący proces powłoki wskazanym programem, bez tworzenia nowego procesu

□□ **TYLKO ODCZYT**

Włączenie schedulera: SCHEDULER_DISABLED na FALSE

```
EXEC DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('SCHEDULER_DISABLED', 'FALSE');
```

Przykład:

```
SQL> EXEC DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('SCHEDULER_DISABLED', 'FALSE');


PL/SQL procedure successfully completed.
```

exec = zastąp bieżący proces powłoki wskazanym programem, bez tworzenia nowego procesu

ZMIENIA SYSTEM

Restart koordynatora: wyzerowanie JOB_QUEUE_PROCESSES

```
ALTER SYSTEM SET job_queue_processes = 0;
```

> **Pojęcie: JOB_QUEUE_PROCESSES.** Parametr instancji określający maksymalną liczbę procesów wykonujących zadania schedulera równolegle. Ustawienie go na `0` zatrzymuje cały mechanizm — żaden nowy job się nie uruchomi, dopóki nie przywrócimy wartości dodatniej.

Przykład:

```
SQL> ALTER SYSTEM SET job_queue_processes = 0;


System altered.
```

SET = Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP

ZMIENIA SYSTEM

Restart koordynatora: przywrócenie JOB_QUEUE_PROCESSES

```
ALTER SYSTEM SET job_queue_processes = 1000;
```

Powrót do wartości roboczej po chwili — sam ten cykl `0` → wartość docelowa wymusza, żeby proces koordynatora zakończył się i wystartował od nowa, co w praktyce jest jego restartem bez zatrzymywania całej instancji.

Przykład:

```
SQL> ALTER SYSTEM SET job_queue_processes = 1000;


System altered.
```

□□ **SET** = Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP

□□ **ZMIENIA SYSTEM**

Ustawienie trwałej wartości JOB_QUEUE_PROCESSES

```
ALTER SYSTEM SET job_queue_processes = 1000 SCOPE=BOTH;
```

Trwałą wartość tego parametru, przeżywającą restart instancji, ustawiamy z `SCOPE=BOTH`.

Przykład:

```
SQL> ALTER SYSTEM SET job_queue_processes = 1000 SCOPE=BOTH;


System altered.
```

□□ **SET** = Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP

□□ **TYLKO ODCZYT**

Dodatkowe wyłączenie starszych zadań DBMS_JOB

```
EXEC DBMS_IJOB.SET_ENABLED(FALSE);
```

Instancje z zadaniami zdefiniowanymi jeszcze starszym pakietem `DBMS_JOB` mają osobny wyłącznik na poziomie konkretnego zadania — `DBMS_IJOB.SET_ENABLED(FALSE)` wyłącza je, `TRUE` włącza z powrotem. Sięgamy po to tylko w bazach, które nadal mają odziedziczone zadania sprzed migracji na `DBMS_SCHEDULER`.

Przykład:

```
SQL> EXEC DBMS_IJOB.SET_ENABLED(FALSE);
```

```
PL/SQL procedure successfully completed.
```

exec = zastąp bieżący proces powłoki wskazanym programem, bez tworzenia nowego procesu

ZMIENIA SYSTEM

Wyczyszczenie shared pool przy zawieszonym schedulerze

```
ALTER SYSTEM FLUSH SHARED_POOL;
```

Krok pomocniczy, po który sięgamy, gdy restart koordynatora przez `JOB_QUEUE_PROCESSES` sam w sobie nie wystarcza — w shared pool, opisanym na stronie Architektura Oracle: instancja, baza, SGA, PGA, procesy tła, potrafi utkwić plan wykonania powiązany z zablokowanym zadaniem.

Przykład:

```
SQL> ALTER SYSTEM FLUSH SHARED_POOL;


System altered.
```

SGA = System Global Area: wspólna pamięć instancji Oracle · **PGA** = Program Global Area: prywatna pamięć procesu serwera Oracle

TYLKO ODCZYT

Sprawdzenie historii uruchomień joba

```
SELECT log_date, job_name, status, error#, run_duration
FROM dba_scheduler_job_run_details
WHERE job_name = 'NazwaJobu'
ORDER BY log_date DESC
FETCH FIRST 10 ROWS ONLY;
```

`error#` różny od zera wskazuje, że zadanie zakończyło się błędem — konkretny numer odpowiada standardowemu kodowi błędu Oracle (`ORA-nnnnn`), a `run_duration` pokazuje, ile faktycznie trwało wykonanie, niezależnie od tego, czy zakończyło się sukcesem.

Przykład:

```
LOG_DATE                              JOB_NAME              STATUS     ERROR# RUN_DURATION
------------------------------------- --------------------- ---------- ------ ------------
25-SEP-26 02.00.14.000000 AM +02:00 JOB_EKSPORT_RAPORTOW  SUCCEEDED       0 +00 00:04:22
24-SEP-26 02.00.09.000000 AM +02:00 JOB_EKSPORT_RAPORTOW  FAILED       1017 +00 00:00:02
```

Wiersz ze `STATUS = FAILED` i `ERROR# = 1017` mówi wprost, że przyczyną był `ORA-01017: invalid username/password` — zwykle skutek wygasłego hasła konta, pod którym zadanie się łączy, opisanego na stronie Użytkownicy, role i uprawnienia.

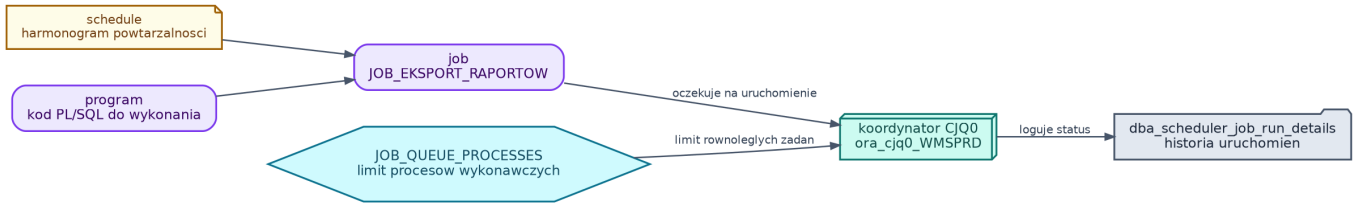
Z życia wzięte

Z ŻYCIA WZIĘTE

W poniedziałek rano Marek dzwoni, że nocny raport zapasów magazynowych z WMSPRD nie doszedł do jego skrzynki. Tomek sprawdza `dba_scheduler_jobs` i widzi, że `JOB_EKSPORT_RAPORTOW` ma status SCHEDULED, ale nigdy się nie uruchomił, bo ktoś wcześniej ustawił `JOB_QUEUE_PROCESSES` na zero przy diagnozowaniu innego problemu. Po przywróceniu wartości roboczej koordynator schedulera CJQ0 wstaje ponownie i zaległy raport odpala się ręcznie poleceniem `RUN_JOB`.

DBMS_SCHEDULER

Pojęcie: DBMS_SCHEDULER. Wbudowany pakiet PL/SQL do planowania i uruchamiania zadań wewnątrz bazy — następca starszego `DBMS_JOB`, wciąż obecnego w bazie dla kompatybilności wstecznej. Cztery pojęcia opisują jego model pracy:



Schedule i program razem tworzą job, job czeka na koordynatora CJQ0, a JOB_QUEUE_PROCESSES ogranicza liczbę zadań wykonywanych naraz (kliknięcie otwiera pełny

rozmiar)

> **Pojęcie: job.** Pojedyncze zadanie — co ma się wykonać (kod PL/SQL, procedura, polecenie systemowe) i kiedy.

> **Pojęcie: program.** Zdefiniowana osobno akcja do wykonania, wielokrotnego użytku — kilka jobów może odwoływać się do jednego programu z różnym harmonogramem.

> **Pojęcie: schedule.** Harmonogram — reguła powtarzalności, niezależna od konkretnego joba, również wielokrotnego użytku.

> **Pojęcie: window.** Okno czasowe z przypisanym poziomem priorytetu zasobów (przez Resource Manager), w którym zadania mogą się wykonywać z określonym limitem zasobów.

Restart schedulera programów

> **Pojęcie: SCHEDULER_DISABLED.** Atrybut schedulera, którym wyłączamy uruchamianie zadań całkowicie — inaczej niż zatrzymanie pojedynczego joba, ten wyłącznik dotyczy wszystkich zadań schedulera naraz.

> **Pojęcie: koordynator schedulera (CJQ0).** Proces tła odpowiedzialny za faktyczne odpalanie zaplanowanych zadań — widoczny w systemie jako `ora_cjq0_NazwaSID`. Gdy zadania przestają się uruchamiać mimo poprawnego harmonogramu i `SCHEDULER_DISABLED = FALSE`, podejrzanym jest właśnie ten proces, nie definicja samego joba. Restart koordynatora wymuszamy pośrednio, przez zmianę liczby procesów kolejki zadań.

Częste błędy

Zatrzymanie joba przez STOP_JOB i pozostawienie go tak na stałe. `STOP_JOB` nie zmienia harmonogramu — job wystartuje ponownie przy najbliższym zaplanowanym uruchomieniu, chyba że dodatkowo go wyłączymy przez `DBMS_SCHEDULER.DISABLE`.

Mylenie SCHEDULER_DISABLED z JOB_QUEUE_PROCESSES = 0. Pierwsze wyłącza scheduler na poziomie logicznym, drugie odbiera mu procesy do wykonywania zadań na poziomie instancji — oba dają podobny efekt widoczny z zewnątrz, ale to dwa różne przełączniki i warto sprawdzić oba, jeśli zadania nie startują.

Powiązane: Sesje: aktywne, nieaktywne, najdłuższe · Architektura Oracle: instancja, baza, SGA, PGA, procesy tła · Użytkownicy, role i uprawnienia

Źródła: notatki własne, dokumentacja Oracle Database PL/SQL Packages and Types Reference 19c