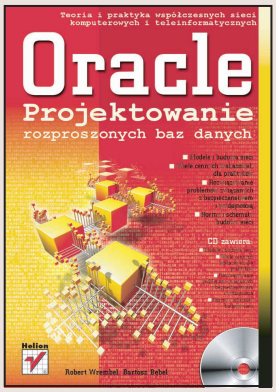
Oracle. Projektowanie rozproszonych baz danych



Autorzy: Robert Wrembel, Bartosz Bębel
ISBN: 83-7197-951-7
Format: B5, stron: 304

W typowych zastosowaniach systemów baz danych wykorzystuje się architekturę scentralizowaną, w której system zarządzania bazą i wszystkie dane znajdują się w tym samym węźle sieci informatycznej. Istnieje jednak wiele zastosowań, w których scentralizowane bazy danych nie zapewniają wymaganej funkcjonalności i efektywności pracy. W takich przypadkach stosuje się tzw. rozproszone bazy danych.

Wiele problemów związanych z projektowaniem i zarządzaniem scentralizowanymi bazami danych, m.in. projektowanie struktury bazy, przetwarzanie i optymalizacja zapytań, zarządzanie współbieżnością transakcji staje się znacznie trudniejsze w przypadku baz rozproszonych.

Najpopularniejszymi systemami umożliwiającymi tworzenie rozproszonych baz danych są systemy firmy Oracle. Funkcjonalność Oracle pociąga za sobą dużą złożoność oprogramowania. Niniejsza książka stanowi kompendium wiedzy niezbędnej do projektowania rozproszonych baz danych, opartych na Oracle, a także potrzebnej do administrowania takimi bazami.

Tematy poruszone w książce to m.in.:

- Architektura rozproszonych baz danych
- Oprogramowanie komunikacji sieciowej Oracle Net
- Procesy komunikacji sieciowej Oracle
- Zarządzanie transakcjami rozproszonymi
- Replikacja danych: migawki i zaawansowane sposoby replikacji
- Oracle9i Lite – rozproszona baza danych dla urządzeń mobilnych
- Optymalizacja zapytań rozproszonych
- Partycjonowanie tabel i indeksów

Książka jest adresowana do administratorów rozproszonych baz danych (nie tylko systemu Oracle), szerokiego grona informatyków, zajmujących się projektowaniem rozproszonych systemów informatycznych opartych na bazach danych, studentów kierunków związanych z informatyką oraz wszystkich tych, którzy interesują się problematyką rozproszonych baz danych.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 3.

Procesy Oracle komunikacji sieciowej

Komunikacja między każdą aplikacją użytkownika a bazą danych odbywa się za pośrednictwem tzw. **procesu usługowego** (ang. *server process*). Jego rolą jest obsługa żądań użytkowników. Do jego zadań m.in. należy:

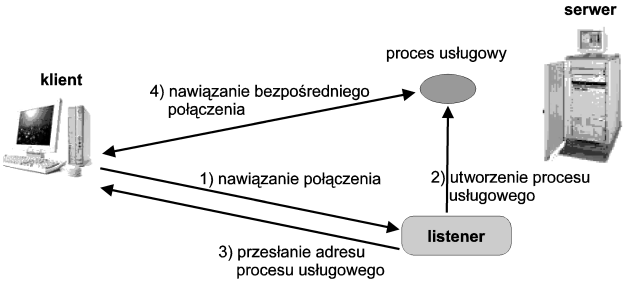
- wykonywanie analizy składniowej i optymalizacja poleceń SQL;
- wykonywanie poleceń SQL;
- odczytywanie żądanych danych z dysku i umieszczanie ich w pamięci *SGA* — por. [WJZ99, LoKo02, LoTh02, O9Con];
- przekazywanie wyników poleceń SQL do aplikacji użytkowników.

W *SZBD Oracle* procesy usługowe mogą pracować w trzech następujących konfiguracjach: procesów dedykowanych, procesów czuwających i procesów współdzielonych. Wszystkie trzy typy procesów omówione zostaną w niniejszym rozdziale.

Dedykowany proces usługowy

W standardowej konfiguracji klient-serwer, dla każdej aplikacji użytkownika jest tworzony jeden proces usługowy, za pomocą którego realizowany jest dla tej aplikacji dostęp do bazy danych. Jest to tzw. architektura systemu z **dedykowanymi procesami usługowymi** (ang. *dedicated servers*). Została ona przedstawiona na rysunku 3.1.

W powyższej architekturze komunikacja między aplikacją użytkownika a bazą danych przebiega w czterech następujących krokach. W kroku 1. przez aplikację użytkownika zostaje nawiązane połączenie z konkretnym procesem nasłuchującym — *listener*. Po odebraniu żądania *listener* tworzy nowy dedykowany proces usługowy do obsługi żądań użytkownika (krok 2.). W kroku 3. *listener* przesyła do aplikacji adres i numer



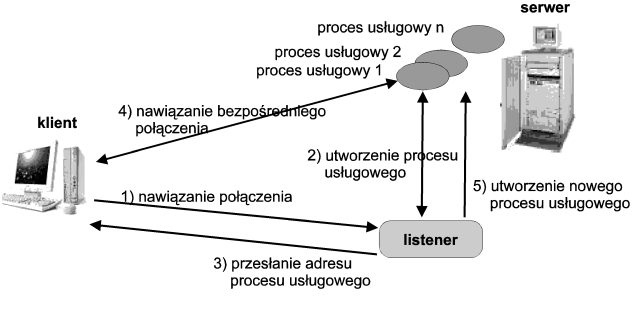
Rysunek 3.1. *Komunikacja między aplikacją użytkownika a bazą danych w architekturze z dedykowanymi procesami usługowymi*

portu komunikacyjnego dedykowanego procesu usługowego. Po otrzymaniu tego adresu zostaje nawiązane przez aplikację bezpośrednie połączenie ze wskazanym procesem usługowym (krok 4.).

Standardowa konfiguracja *listenera*, omówiona w rozdziale 2., zapewnia pracę w architekturze dedykowanych procesów usługowych.

Czuwający proces usługowy

Rozszerzenie konfiguracji systemu z serwerami dedykowanymi stanowi tzw. architektura z **czuwającymi procesami usługowymi** (ang. *prespawned servers*). W tym przypadku *listener*, po jego uruchomieniu, tworzy pulę aktywnych procesów usługowych, gotowych do obsługi żądań użytkowników. W momencie pojawienia się żądania jeden z wolnych procesów czuwających przydzielany jest do obsługi tego żądania. Ponieważ proces czuwający jest natychmiast gotowy do pracy, odpowiedź systemu jest szybsza. Przykładowa architektura systemu z czuwającymi procesami usługowymi została przedstawiona na rysunku 3.2.



Rysunek 3.2. *Komunikacja między aplikacją użytkownika a bazą danych w architekturze czuwających procesów usługowych*

Komunikacja między aplikacją użytkownika a bazą danych przebiega podobnie, jak w przypadku przedstawionym na rysunku 3.1. W kroku 1. przez aplikację użytkownika zostaje nawiązane połączenie z procesem *listener*. Po odebraniu żądania przez proces *listener* zostaje wybrany jeden z dostępnych procesów usługowych (krok 2.), a jego adres i numer portu jest przekazywany do aplikacji (krok 3.). Po otrzymaniu tego adresu przez aplikację zostaje nawiązane bezpośrednie połączenie ze wskazanym procesem usługowym (krok 4.). W kroku 5. przez proces *listener* tworzony jest nowy proces usługowy, który zostaje dodany do puli dostępnych aktywnych procesów.

Konfigurowanie procesu nasłuchu

W celu skonfigurowania *listenera* w architekturze z czuwającymi procesami usługowymi należy w pliku *listener.ora* umieścić zbiór parametrów, określających dla każdego protokołu m.in. maksymalną liczbę procesów usługowych, pulę aktywnych procesów i czas istnienia procesu po jego wykorzystaniu. Przykładową zawartość pliku *listener.ora* przedstawiono poniżej.

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = dcs-rw-toshiba)(PORT = 1521))
      )
    )
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = LAB92.II.PP)
      (ORACLE_HOME = C:\oracle\ora92)
      (SID_NAME = LAB92)
      (PRESPAWN_MAX = 20)
      (PRESPAWN_LIST =
        (PRESPAWN_DESC =
           (PROTOCOL = TCP)
           (POOL_SIZE = 10)
           (TIMEOUT = 1)
        )
      )
    )
  )
```

W powyższym przykładzie dla instancji *LAB92* wyspecyfikowano maksymalnie 20 procesów czuwających (parametr `PRESPAWNED_MAX=20`). Dla połączeń za pomocą protokołu TCP/IP pula aktywnych procesów wynosi 10 (parametr `POOL_SIZE=10`). Parametr `TIMEOUT=1` oznacza czas istnienia (w minutach) procesu usługowego po jego wykorzystaniu przez aplikację użytkownika. Po upłynięciu tego czasu proces jest usuwany z systemu.

Liczba czuwających procesów usługowych, tworzonych w momencie startowania procesu nasłuchowego, jest widoczna na ekranie po wydaniu polecenia `lsnrctl start nazwa_procesu`. Poniżej przedstawiono fragment informacji wyświetlanych w czasie startowania *listenera*, skonfigurowanego jak wyżej. Można zauważyć, że po uruchomieniu *listenera* tworzonych jest 11 procesów usługowych, z których 10 jest czuwających.

```
C:\>lsnrctl start
...
Rozpoczęcie tnslsnr: proszę czekać ...
TNSLSNR for 32-bit Windows: Version 9.2.0.1.0 - Production
Plik parametrów systemowych jest C:\oracle\ora92\network\admin\listener.ora
STAN NASŁUCHU
------------------------
Alias                     LISTENER
```

```
...
Nasłuch punktów końcowych - podsumowanie...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=dcs-rw-toshiba)(PORT=1521)))
Podsumowanie usług...
Usługa "LAB92.II.PP" ma liczbę instancji równą 1.
Instancja "LAB92", stan UNKNOWN, ma dla tej usługi 11 procedur(-ę, -y) obsługi.
```

Szczegółową informację o procesach usługowych można uzyskać, wydając polecenie:

```
lsnrclt services
```

Architektura systemu z dedykowanymi procesami czuwającymi powinna być stosowana dla niewielkiej i średniej liczby użytkowników. Liczba użytkowników, przy której konfiguracja ta zapewnia efektywny dostęp do bazy danych, zależy od mocy obliczeniowej i pamięci RAM węzła. Każdy nowy proces użytkownika wymaga uruchomienia odrębnego procesu usługowego, który zajmuje czas procesora i alokuje swoją pamięć. Oznacza to, że nawet silny obliczeniowo węzeł dla dużej liczby równocześnie pracujących użytkowników wykorzysta wszystkie swoje zasoby, co z kolei obniży efektywność systemu. Z tych względów *Oracle* obsługuje tak zwaną architekturę ze współdzielonymi procesami usługowymi.

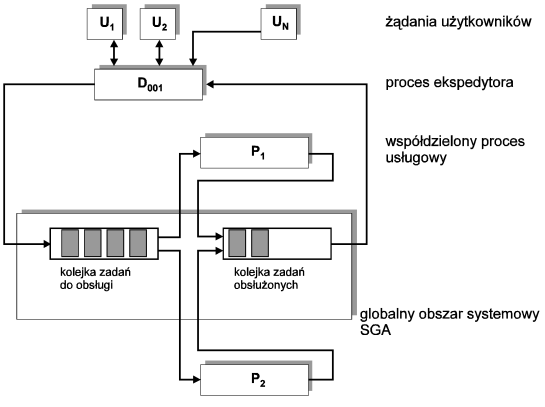
Współdzielony proces usługowy

Instancję bazy danych można również uruchomić w konfiguracji ze **współdzielonymi procesami usługowymi** (ang. *shared servers*), zwanej wcześniej **wielowątkową** (ang. *multi-threaded servers*). W konfiguracji tej pojedynczy proces usługowy obsługuje żądania wielu aplikacji użytkowników, natomiast aplikacja łączy się najpierw z tzw. **procesem ekspedytora** (ang. *dispatcher*). Zadaniem ekspedytora jest kierowanie żądań aplikacji (np. wykonanie polecenia SQL, wywołanie procedury składowanej) do wolnego procesu usługowego.

W konfiguracji współdzielonej serwer może wykorzystywać wiele ekspedytorów i wiele współdzielonych procesów usługowych. Liczba współdzielonych procesów usługowych jest dostosowywana dynamicznie przez *SZBD*, w zależności od liczby żądań użytkowników.

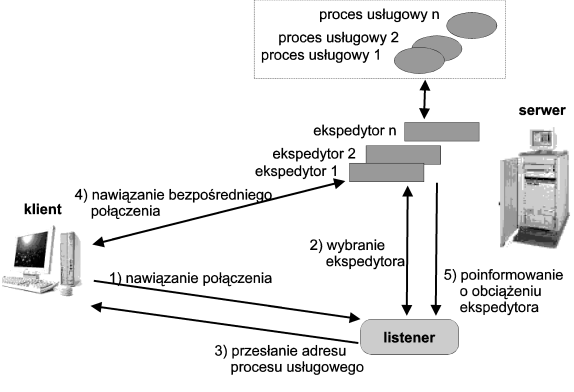
Żądania użytkowników, które zostały przyjęte przez ekspedytor, trafiają do tzw. **kolejki zadań do obsługi** (ang. *request queue*). Dla danej instancji bazy danych istnieje tylko jedna taka kolejka i jest ona współdzielona przez wszystkie ekspedytory. Zadania z tej kolejki są następnie pobierane i obsługiwane przez współdzielone procesy usługowe. Kolejka tych zadań jest obsługiwana zgodnie z algorytmem *FIFO* (*first-in-first-out*), czyli zadania są obsługiwane w kolejności ich pojawiania się w kolejce. Obsłużone zadania (np. wyniki zapytania) trafiają do odpowiednich **kolejek zadań obsłużonych** (ang. *response queues*). Każdy ekspedytor posiada prywatną kolejkę takich zadań. Informacje z kolejki zadań obsłużonych są następnie przesyłane przez ekspedytor do odpowiedniego procesu użytkownika. Kolejka zadań do obsługi, jak i kolejka zadań obsłużonych znajduje się w pamięci *SGA* instancji bazy danych.

Rysunek 3.3 przedstawia przykładową architekturę pracy współdzielonych procesów usługowych. Żądania użytkowników są kierowane do współdzielonych procesów usługowych P_1 i P_2 przez jeden ekspedytor D_{001}.



Rysunek 3.3. *Współdzielone procesy usługowe*

Komunikacja między procesem użytkownika a bazą danych została zilustrowana na rysunku 3.4. W czasie uruchamiania instancji bazy danych — por. [WJZ99, LoKo02, LoTh02, O9Con] — uruchamiane są procesy ekspedytorów i tworzona jest pula procesów usługowych. Adresy dostępnych ekspedytorów są rejestrowane w procesie *listener*. W kroku 1. żądanie użytkownika jest odbierane przez proces *listener*. Po odebraniu żądania przez proces *listener* zostaje wybrany najmniej obciążony ekspedytor (krok 2.), a następnie do procesu klienta zostaje wysłane żądanie przełączenia się na wskazany ekspedytor wraz z jego adresem i numerem portu komunikacji (krok 3.). W kroku 4. zostaje przez aplikację użytkownika nawiązane bezpośrednie połączenie ze wskazanym ekspedytorem. Każdorazowo po nawiązaniu połączenia z nową aplikacją użytkownika do *listenera* zostaje przez ekspedytora wysłana informacja o jego obciążeniu (krok 5.).



Rysunek 3.4. *Komunikacja między aplikacją użytkownika a bazą danych w architekturze współdzielonych procesów usługowych*

Konfigurowanie współdzielonych procesów usługowych

W celu skonfigurowania instancji bazy danych w architekturze ze współdzielonymi procesami usługowymi należy w pliku konfiguracyjnym instancji bazy danych *initSID.ora* — por. [O9Dba, WJZ99] — umieścić zbiór dodatkowych parametrów.

Przykładowy zbiór tych parametrów przedstawiono poniżej. Pierwszy wiersz definiuje zbiór 4 ekspedytorów (DISPATCHERS=4) dla protokołu TCP/IP (PROTOCOL=TCP). Każdy z ekspedytorów może obsługiwać maksymalnie 100 połączeń (CONNECTIONS=100).

Drugi wiersz definiuje 1 ekspedytor dla protokołu SPX/IPX. Ekspedytor może obsługiwać maksymalnie 20 połączeń. W wersjach wcześniejszych niż *Oracle9i* zamiast parametru DISPATCHERS należało wykorzystywać MTS_DISPATCHERS.

```
DISPATCHERS="(PROTOCOL=TCP) (DISPATCHERS=4) (CONNECTIONS=100)"
DISPATCHERS="(PROTOCOL=SPX) (DISPATCHERS=1) (CONNECTIONS=20)"
```

Dodatkowo, specyfikując parametry ekspedytora można jawnie podać adres komputera, na którym ekspedytor będzie pracował, i jawnie zaalokować dla niego numer portu. Poniższy listing definiuje trzy ekspedytory pracujące na komputerze o nazwie *dcs-rw-toshiba*. Każdy z nich posiada jednak inny port komunikacji: D000 pracuje na porcie 3400, D001 na 3401, a D002 na 3402.

```
DISPATCHERS="(ADDRESS=(PROTOCOL=tcp)(HOST=dcs-rw-
toshiba)(PORT=3400))(DISPATCHERS=1)(CONNECTIONS=100)"
DISPATCHERS="(ADDRESS=(PROTOCOL=tcp)(HOST=dcs-rw-
toshiba)(PORT=3401))(DISPATCHERS=1)(CONNECTIONS=100)"
DISPATCHERS="(ADDRESS=(PROTOCOL=tcp)(HOST=dcs-rw-
toshiba)(PORT=3402))(DISPATCHERS=1)(CONNECTIONS=100)"
```

W pliku *initSID.ora* poszczególne sekcje DISPATCHERS muszą być pisane w jednym wierszu.

Liczba współdzielonych procesów usługowych, wykorzystywanych przez instancję, jest kontrolowana za pomocą dwóch parametrów inicjujących: SHARED_SERVERS (MTS_SERVERS w wersjach wcześniejszych niż *Oracle9i*) i MAX_SHARED_SERVERS (MTS_MAX_SERVERS w wersjach wcześniejszych niż *Oracle9i*). Pierwszy z nich określa minimalną liczbę procesów usługowych, tworzonych w czasie startowania instancji bazy danych i utrzymywanych przez nią w czasie pracy. Drugi parametr określa maksymalną liczbę tych procesów. Maksymalną liczbę ekspedytorów określa parametr MAX_DISPATCHERS (MTS_MAX_DISPATCHERS w wersjach wcześniejszych niż *Oracle9i*). Przykładowo, poniższe parametry definiują 40 procesów usługowych jako minimum i 120 jako maksimum, a maksymalna liczba ekspedytorów ustalona jest na 10.

```
shared_servers=40           # mts_servers
max_shared_servers=120      # mts_max_servers
max_dispatchers=10          # mts_max_dispatchers
```

Rozmiar pamięci procesów usługowych

W architekturze współdzielonych procesów usługowych standardowo każdy proces alokuje pamięć w *obszarze współdzielonym* (ang. *shared pool*) pamięci *SGA* — por. [O9Con, O9Dba, LoKo02, LoTh02]. Rozwiązanie takie nie jest jednak zalecane, ponie-

waż wpływa na zmniejszenie efektywności działania bazy danych. Z tego powodu dla procesów usługowych *Oracle Corp.* zaleca alokowanie odrębnego obszaru pamięci poza *SGA*. Rozmiarem tego obszaru steruje parametr konfiguracyjny instancji LARGE_POOL_SIZE (LARGE_POOL w wersjach wcześniejszych niż *Oracle9i*). Rozmiar tego obszaru dobiera się szacunkowo na podstawie dotychczasowego wykorzystania pamięci przez procesy usługowe i liczby równoczesnych dołączeń do bazy danych.

Informacje na temat wykorzystania pamięci przez procesy usługowe można odczytać za pomocą poniższego zapytania, skierowanego do dynamicznych tabel systemowych *V$SESSION*, *V$SESSTAT* i *V$STATNAME*.

```
select sess.sid, sess.username,
       sname.name, stat.value
from v$session sess,
     v$sesstat stat,
     v$statname sname
where sess.sid=stat.sid
and   stat.statistic#=sname.statistic#
and   sname.name in ('session uga memory', 'session uga memory max')
and   sess.username is not null
order by sess.sid;
```

Przykładowy wynik zapytania przedstawiono poniżej.

```
SID        USERNAME   NAME                          VALUE
---------- ---------- ------------------------- ----------
         9 SYS        session uga memory           338816
         9 SYS        session uga memory max       469744
        12 PUCHATEK   session uga memory            76960
        12 PUCHATEK   session uga memory max        76960
        15 SHREK      session uga memory            43568
        15 SHREK      session uga memory max       572756
        17 BOLEK      session uga memory            59604
        17 BOLEK      session uga memory max       139784
```

SID oznacza identyfikator sesji, *USERNAME* — nazwę użytkownika bazy danych, *NAME* — opis wartości kolumny *VALUE*. *VALUE* reprezentuje rozmiar zaalokowanej pamięci, mierzony w bajtach. *session uga memory* reprezentuje aktualnie wykorzystywany przez proces usługowy rozmiar pamięci, natomiast *session uga memory max* reprezentuje maksymalny rozmiar pamięci, jaki wykorzystano w ramach bieżącej sesji.

Na podstawie powyższego wyniku widzimy, że sesja o numerze 15 wykorzystała maksymalnie 572756B pamięci, natomiast aktualnie wykorzystuje 43568B. Jeżeli w systemie spodziewanych jest 100 równoczesnych sesji o podobnej charakterystyce, wówczas rozmiar LARGE_POOL_SIZE należy wyliczyć jako: 572756 ∗ 100.

Minimalną liczbę ekspedytorów można ustalić na podstawie maksymalnej liczby połączeń obsługiwanych przez jeden proces usługowy i na podstawie oczekiwanej liczby równoczesnych połączeń. Maksymalna liczba połączeń obsługiwanych przez proces usługowy zależy od systemu operacyjnego. Można ją odczytać po uruchomieniu instancji w trybie współdzielonych procesów usługowych, korzystając z polecenia lsnrctl services. Jeżeli w sekcji DISPATCHERS nie umieścimy parametru CONNECTIONS, wówczas przyjęta zostanie maksymalna dozwolona liczba połączeń w systemie operacyjnym.

Przykładowo, w poniższy listingu maksymalna liczba procesów wynosi 1002 (wartość dla Windows XP). Zaleca się jednak unikanie maksymalnego obciążania ekspedytorów.

```
C:\>lsnrctl services

LSNRCTL for 32-bit Windows: Version 9.2.0.1.0 - Production ....
.......
.......
    Procedura(-y) obsługi:
      "DEDICATED" ustanowiono:0 odmówiono:0 stan:ready
         LOCAL SERVER
      "D000" ustalono:0 odmówiono:0 bieżące:0 maks.:1002 stan:ready
         DISPATCHER <machine: DCS-RW-TOSHIBA, pid: 2100>
         (ADDRESS=(PROTOCOL=tcp)(HOST=dcs-rw-toshiba)(PORT=3400))
```

Informacje o aktualnej architekturze pracy instancji bazy danych

Informacje o aktualnej architekturze pracy instancji można uzyskać na kilka sposobów. Pierwszy z nich wymaga dołączenia się do bazy danych z poziomu *SQL*Plus* jako użytkownik z uprawnieniami DBA. Następnie należy wydać polecenie `show parameters dispatchers`. Jeżeli nie skonfigurowano instancji w architekturze współdzielonych procesów usługowych, wówczas wartość parametru inicjującego `DISPATCHERS` będzie pusta, jak przedstawiono w poniższym listingu. Parametry `MAX_DISPATCHERS` i `MTS_MAX_DISPATCHERS` przyjmują w takim przypadku wartości domyślne.

```
SQL> show parameters dispatchers
NAME                                 TYPE        VALUE
------------------------------------ ----------- --------
dispatchers                          string
max_dispatchers                      integer     5
mts_dispatchers                      string
mts_max_dispatchers                  integer     5
```

W drugim sposobie wykorzystywane jest polecenie `lsnrctl services`, jak podano poniżej. W poniższym listingu znajduje się przykładowy opis czterech ekspedytorów o nazwach D000, D001, D002, D003. Pierwszy z nich obsługuje 2 połączenia (*"D000" ustalono:2*). Kolejne 2 połączenia są obsługiwane przez procesy dedykowane (*"DEDICATED" ustanowiono:2*).

```
LSNRCTL> services
Łączy się z (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=dcs-rw-toshiba)
(PORT=1521)))
Podsumowanie usług...
Usługa "LAB92.II.PP" ma liczbę instancji równą 1.
  Instancja "LAB92", stan UNKNOWN, ma dla tej usługi 1 procedur(-ę, -y)
  obsługi...

    Procedura(-y) obsługi:
      "DEDICATED" ustalono:0 odmówiono:0
         LOCAL SERVER
  Instancja "LAB92", stan READY, ma dla tej usługi 5 procedur(-ę, -y)
```

```
obsługi...
  Procedura(-y) obsługi:
    "D003" ustalono:0 odmówiono:0 bieżące:0 maks.:100 stan:ready
      DISPATCHER <machine: DCS-RW-TOSHIBA, pid: 2392>
      (ADDRESS=(PROTOCOL=tcp)(HOST=dcs-rw-toshiba)(PORT=3440))
    "D002" ustalono:0 odmówiono:0 bieżące:0 maks.:100 stan:ready
      DISPATCHER <machine: DCS-RW-TOSHIBA, pid: 3040>
      (ADDRESS=(PROTOCOL=tcp)(HOST=dcs-rw-toshiba)(PORT=3439))
    "D001" ustalono:0 odmówiono:0 bieżące:0 maks.:100 stan:ready
      DISPATCHER <machine: DCS-RW-TOSHIBA, pid: 2412>
      (ADDRESS=(PROTOCOL=tcp)(HOST=dcs-rw-toshiba)(PORT=3438))
    "D000" ustalono:2 odmówiono:0 bieżące:0 maks.:100 stan:ready
      DISPATCHER <machine: DCS-RW-TOSHIBA, pid: 2124>
      (ADDRESS=(PROTOCOL=tcp)(HOST=dcs-rw-toshiba)(PORT=3436))
    "DEDICATED" ustanowiono:2 odmówiono:0 stan:ready
      LOCAL SERVER
Polecenie zakończone powodzeniem
```

W systemach Unix wydanie polecenia `ps` (np. `ps -ef`) umożliwia wyświetlenie procesów systemowych. Instancja w architekturze współdzielonych procesów usługowych będzie posiadała procesy ekspedytorów o nazwach *ora_dxxx_SID*.

Kolejnym źródłem informacji na temat architektury współdzielonych procesów usługowych są dynamiczne tabele systemowe, m.in. *V$DISPATCHER* i *V$SHARED_SERVER* — por. [O9Ref]. Pierwsza z nich udostępnia nazwy i parametry ekspedytorów, a druga — nazwy i parametry współdzielonych procesów usługowych. Przykładowe zapytanie do *V$DISPATCHER* i jego wynik przedstawiono poniżej.

```
SQL> select name, network, status, conf_indx from v$dispatcher;

NAME NETWORK                        STATUS   CONF_INDX
---- ------------------------------ ------- ----------
D000 (ADDRESS=(PROTOCOL=tcp)(HOST= WAIT             0
     dcs-rw-toshiba)(PORT=3400))

D001 (ADDRESS=(PROTOCOL=tcp)(HOST= WAIT             1
     dcs-rw-toshiba)(PORT=3401))

D002 (ADDRESS=(PROTOCOL=tcp)(HOST= WAIT             2
     dcs-rw-toshiba)(PORT=3402))
```

Atrybut *name* oznacza nazwę ekspedytora, *network* przechowuje parametry wyspecyfikowane w omówionej wcześniej sekcji `ADDRESS`, *status* przechowuje aktualny stan pracy ekspedytora. *status* może przyjąć m.in. jedną z następujących wartości: *WAIT* — ekspedytor oczekuje na żądanie połączenia, *SEND* — ekspedytor wysyła wiadomość, *RECEIVE* — ekspedytor otrzymuje żądanie, *CONNECT* — nawiązywane jest połączenie z aplikacją użytkownika, *DISCONNECT* — połączenie jest przerywane. Wartość atrybutu *CONF_INDX* określa numer parametru inicjującego `DISPATCHERS`, który posłużył do utworzenia ekspedytora. Ekspedytor utworzony na podstawie wartości pierwszego w kolejności parametru otrzymuje wartość równą 0, ekspedytor utworzony na podstawie drugiego parametru otrzymuje wartość 1 itp. Powyższy wynik otrzymano dla parametrów `DISPATCHERS` określonych w sposób przedstawiony poniżej.

```
DISPATCHERS="(ADDRESS=(PROTOCOL=tcp)(HOST=dcs-rw-
➥toshiba)(PORT=3400))(DISPATCHERS=1)(CONNECTIONS=100)"
```

```
DISPATCHERS="(ADDRESS=(PROTOCOL=tcp)(HOST=dcs-rw-
➥toshiba)(PORT=3401))(DISPATCHERS=1)(CONNECTIONS=100)"
DISPATCHERS="(ADDRESS=(PROTOCOL=tcp)(HOST=dcs-rw-
➥toshiba)(PORT=3402))(DISPATCHERS=1)(CONNECTIONS=100)"
```

Zarządzanie parametrami pracy instancji w architekturze współdzielonych procesów usługowych

W czasie pracy instancji bazy danych można dynamicznie modyfikować liczbę działających współdzielonych procesów usługowych i ekspedytorów. Służy do tego celu polecenie `alter system`. Przykładowo w *Oracle9i* za pomocą poniższego polecenia aktualna liczba procesów usługowych ustalana jest na `30`. We wcześniejszych wersjach systemu `shared_servers` należy zastąpić przez `mts_servers`.

```
SQL> alter system set shared_servers=30;
```

Liczbę i parametry pracy ekspedytorów modyfikuje się w *Oracle9i* poleceniem:

```
alter system set dispatchers = '(INDEX=conf_indx)(parametry);
```

We wcześniejszych wersjach systemu `dispatchers` należy zastąpić przez `mts_dispatchers`.

Przykładowo, poniższe polecenie dodaje do systemu jeden ekspedytor o indeksie 3.

```
SQL> alter system set dispatchers=
  2     '(INDEX=3)
  3      (ADDRESS=(PROTOCOL=TCP)(HOST=dcs-rw-toshiba)(PORT=3403))
  4      (DISPATCHERS=1)';
```

Dodany ekspedytor pojawia się w systemie i jest widoczny za pomocą *V$DISPATCHER*.

```
SQL> select name, network, status, conf_indx from v$dispatcher;

NAME NETWORK                       STATUS   CONF_INDX
---- ----------------------------- ------- ----------
D000 (ADDRESS=(PROTOCOL=tcp)(HOST= WAIT             0
     dcs-rw-toshiba)(PORT=3400))

D001 (ADDRESS=(PROTOCOL=tcp)(HOST= WAIT             1
     dcs-rw-toshiba)(PORT=3401))

D002 (ADDRESS=(PROTOCOL=tcp)(HOST= WAIT             2
     dcs-rw-toshiba)(PORT=3402))

D003 (ADDRESS=(PROTOCOL=TCP)(HOST= WAIT             3
     dcs-rw-toshiba)(PORT=3403))
```

Kolejne przykładowe polecenie umożliwia usunięcie ekspedytora o indeksie 3. Po jego wykonaniu ekspedytor zostanie fizycznie usunięty z systemu z pewnym opóźnieniem.

```
SQL> alter system set dispatchers=
  2  '(INDEX=3)(PROTOCOL=TCP)(DISPATCHERS=0)';
```

Wybór typu procesu usługowego

Jeżeli instancja pracuje w trybie współdzielonych procesów usługowych, wówczas przez aplikację może zostać wskazane, czy do jej obsługi ma być wykorzystany dedykowany proces usługowy, czy współdzielony. Wyboru tego dokonuje się, konfigurując plik *tnsnames.ora*. Opisany wcześniej parametr `SERVER` z wartością *DEDICATE* lub *SHARED* steruje wyborem procesu usługowego. Poprzez wprowadzenie do pliku *tnsnames.ora* dwóch różnych nazw usług, wskazujących na tę samą bazę danych, lecz z różną wartością parametru `SERVER`, może być dokonany wybór procesu obsługującego aplikację właśnie przez wskazanie odpowiedniej usługi. Przykładowy fragment pliku *tnsnames.ora* zawierający taką konfigurację przedstawiono poniżej.

```
LAB92-shared.II.PP =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = dcs-rw-toshiba)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = SHARED)
      (SERVICE_NAME = LAB92)
    )
  )

LAB92-dedicated.II.PP =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = dcs-rw-toshiba)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = LAB92)
    )
  )
```

Jeżeli w pliku *sqlnet.ora* zostanie umieszczony parametr `USE_DEDICATED_SERVER=ON`, wówczas dla wszystkich połączeń aplikacji będą alokowane procesy dedykowane, nawet jeśli w definicji usługi parametr `SERVER` przyjmuje wartość *SHARED*. Brak parametru `USE_DEDICATED_SERVER` lub wpis `USE_DEDICATED_SERVER=OFF` powoduje alokowanie procesów usługowych zgodnie z konfiguracją instancji i wartością parametru `SERVER`.

Wykrywanie nieaktywnych połączeń

Raz nawiązane połączenie między procesem usługowym a aplikacją wykorzystuje zasoby systemowe. W przypadku zerwania połączenia z aplikacją, np. na skutek braku połączenia sieciowego, awarii komputera użytkownika lub aplikacji, proces ją obsługujący staje się nieaktywny (ang. *dead*) i powinien zostać usunięty z systemu. Często w przypadku tego typu awarii dane w bazie są blokowane przez transakcję użytkownika. Usunięcie nieaktywnego procesu powoduje wycofanie transakcji i zwolnienie wszystkich wykorzystywanych przez nią blokad.

Proces testowania aktywności aplikacji użytkownika (ang. *dead connection detection*) jest realizowany przez *Oracle Net*. Oprogramowanie to wysyła pakiety testowe za pomocą każdego z połączeń. Częstotliwość próbkowania jest określona parametrem konfiguracyjnym `SQLNET.EXPIRE_TIME` pliku *sqlnet.ora*. Jego wartość podaje się w minutach. Przykładowo, wpis `SQLNET.EXPIRE_TIME=10` oznacza próbkowanie co 10 minut. Jest to wartość zalecana w dokumentacji *Oracle* [O9NAG]. W domyślnej konfiguracji plik *sqlnet.ora* nie zawiera parametru `SQLNET.EXPIRE_TIME`, a więc nieaktywne połączenia nie są wykrywane.