ADMINISTRACJA BAZAMI DANYCH

Rozdziały 1-29

Rok akademicki – 2009/2010

Notatki do przedmiotu „Administracja bazami danych”

PLAN WYKŁADU

1. **Architektura systemu Oracle.**
2. **Podstawowe obiekty bazy danych i zarządzanie nimi.**
3. **Administracja użytkownikami:**
    - **Tworzenie użytkowników.**
    - **Przywileje systemowe i obiektowe, role.**
    - **Ograniczenia (profile).**
4. **Monitorowanie pracy użytkowników.**
5. **Eksport i import danych.**
6. **Tworzenie nowej bazy.**
7. **Optymalizacja zapytań.**
8. **Backup i Recovery.**
9. **Archiwizacja.**
10. **Data Guard.**
11. **Strojenie instancji.**

BIBLIOGRAFIA

[1] **Dokumentacja ORACLE 10g/11g**

[2] **Zasoby http://www.orafaq.com**

[3] Theriaut M., Carmichael R., Viscusi J.: Oracle 9i - Administrowanie bazami danych od podstaw. Helion 2003

[4] **Loney Kelvin, Oracle 10g. Kompendium administratora. Helion 2005.**

SPIS TREŚCI

1. ROLA ADMINISTRATORA

§ 1.1. Rodzaje użytkowników

- **Użytkownik końcowy:**
  - Korzysta z wcześniej przygotowanych interfejsów.
  - Posiada uprawnienia do odczytywania i modyfikowania wybranych danych.
- **Projektant/Programista:**
  - Korzysta z narzędzi do projektowania aplikacji.
  - Posiada uprawnienia do tworzenia bazy, schematów, tabel, …
- **Administrator:**
  - Opiekuje się gotowymi aplikacjami, instaluje je.
  - Zarządza użytkownikami, nadaje im uprawnienia.
  - Odpowiada za ciągłość pracy.
  - Odpowiada za wydajność.
  - Tworzy kopie bezpieczeństwa.
  - …

§ 1.2. Zadania administratora

- Instalacja systemu.
- Zapewnienie ciągłej pracy aplikacji.
- Startowanie bazy.
- Tworzenie kopii zapasowych.
- Zapewnienie optymalnych warunków dostępu do bazy.
- Dbanie o wydajność systemu.
- Monitorowanie systemu.
- Zapewnienie dostępu do bazy na zasadach bezpieczeństwa określonych odpowiednią polityką (tajność danych, zgodność z ustawami, …).
- Tworzenie lub usuwanie użytkowników.
- Nadawanie lub odbieranie uprawnień użytkownikom.
- Przywracanie bazy po awarii.

2. ARCHITEKTURA SYSTEMU ZARZĄDZANIA BAZĄ DANYCH ORACLE

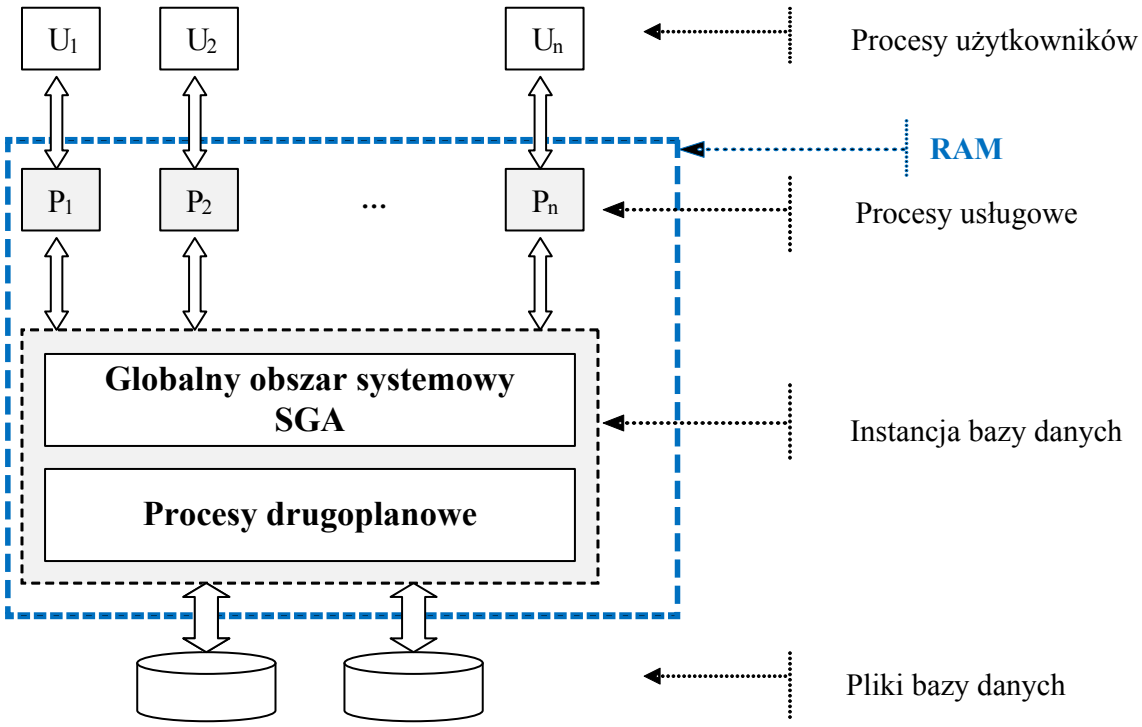
§ 2.1. Pojęcie instancji

Należy rozróżnić pojęcie bazy danych (pliki do przechowywania danych) od pojęcia instancji. Dostęp do danych w tych plikach następuje poprzez instancję.

Pojęcie instancji bazy danych można zilustrować następująco:

- **Baza danych** - komplet plików przechowujących dane i zarządzanych przez specjalną aplikację.
- **Instancja** - uruchomiona aplikacja do zarządzania bazą danych składająca się ze wspólnej pamięci SGA oraz zestawu procesów drugoplanowych.

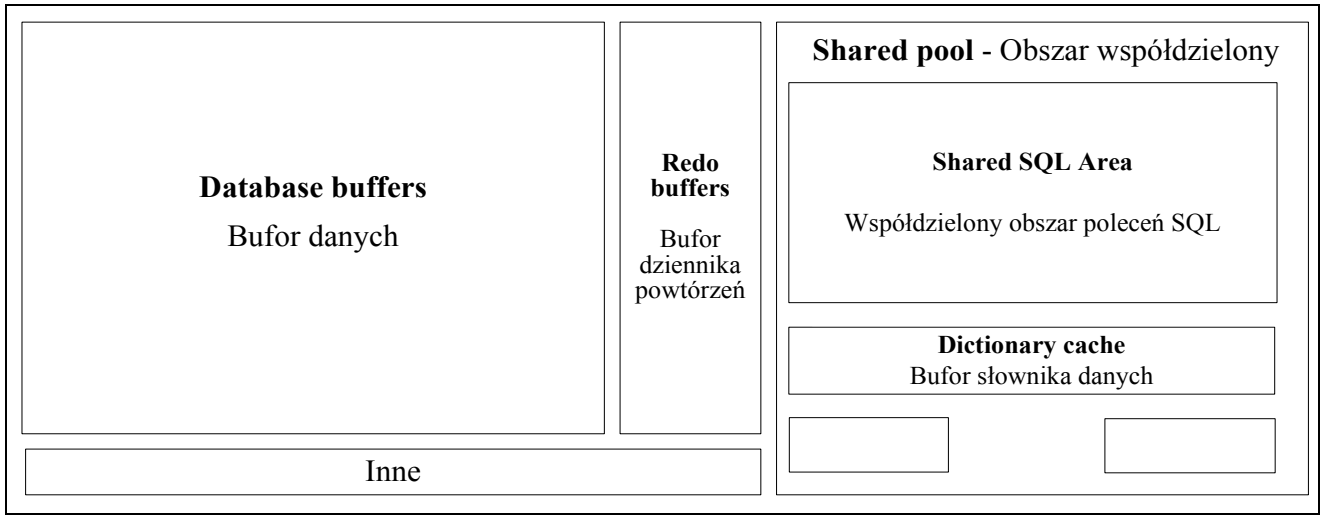
Dostęp do danych odbywa się poprzez instancję.



Rysunek 2.1.1. Pojęcie instancji.

§ 2.2. Struktura pamięci SGA

Obszar **SGA** (System Global Area) jest tworzony w pamięci RAM w momencie startu **instancji**. Można go zilustrować następująco:



Rysunek 2.2.1. Globalny obszar systemowy - SGA.

gdzie

- **Bufor danych** - przechowuje dane odczytane z dysku.
- **Bufor dziennika powtórzeń** - przechowuje informacje o zmianach wprowadzonych do bazy danych.
- **Obszar współdzielony** - przeznaczony do obsługi poleceń SQL i PL/SQL:
  - *Współdzielony obszar poleceń SQL* - przechowuje informacje o poleceniach SQL użytkowników (np. ich optymalizację).
  - *Bufor słownika danych* - przechowuje informacje ze słownika danych.
  - …
- …

```
Total System Global Area  285212672 bytes
Fixed Size                1287016 bytes ..... /* Inne */
Variable Size             100666520 bytes     /* Shared pool */
Database Buffers          180355072 bytes
Redo Buffers              2904064 bytes
```

Dane te można otrzymać z perspektywy dynamicznej `V$SGA` poleceniem:

```
SQL> SELECT * FROM v$sga;
```

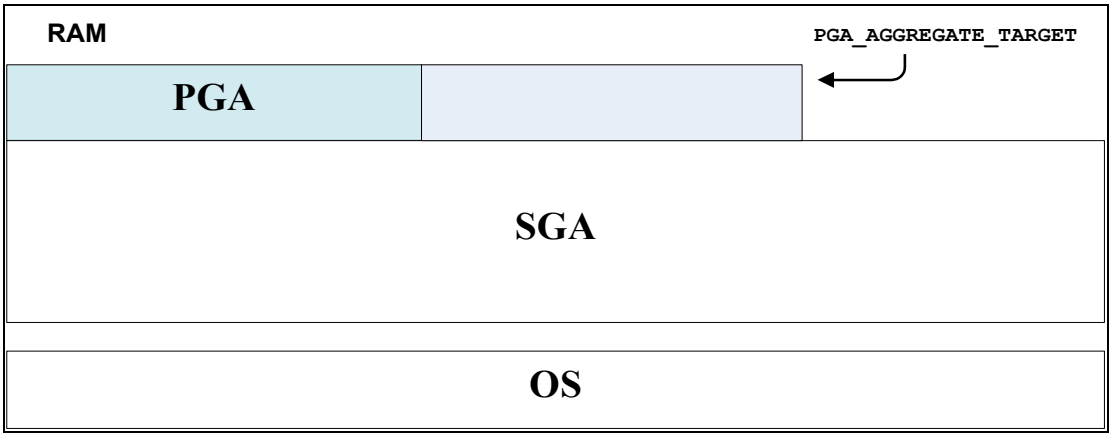
lub poleceniem edytora `sqlplus`:

```
SQL> SHOW sga
```

§ 2.3. Proces Global Area

Każdy proces usługowy i drugoplanowy ma przydzielony obszar pamięci operacyjnej zwany pamięcią procesu.

Obszary te tworzą obszar pamięci nazywanym **PGA** – Proces Global Area, którego maksymalny rozmiar jest zgodny z parametrem instancji `PGA_AGGREGATE_TARGET`.


RAM
PGA_AGGREGATE_TARGET
PGA
SGA
OS

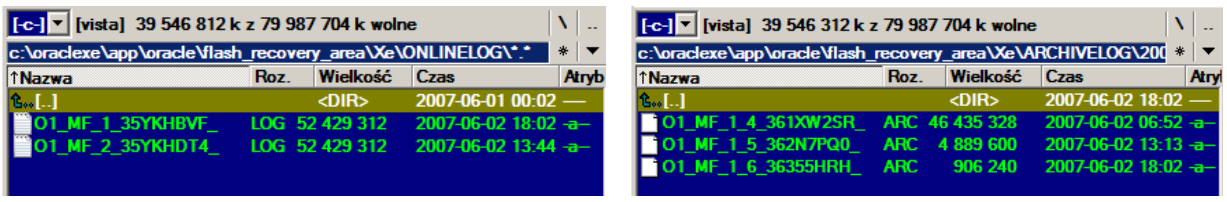
3. PLIKI SYSTEMU ORACLE

System Zarządzania Bazą Danych Oracle wykorzystuje pliki kodu i bazy danych, tj. pliki danych, dziennika powtórzeń, kontrolne, konfiguracyjne, haseł i rejestrujące sposób pracy bazy danych.

- Pliki kodu (**oracle code files**) zawierają oprogramowanie systemu zarządzania bazą danych, programy usługowe i skrypty z poleceniami SQL wykorzystywane m.in. do administrowania systemem, tworzenia bazy danych i różnego rodzaju dodatkowych obiektów systemowych. Pliki kodu znajdują się np. w następujących podkatalogach katalogu domowego Oracle: *dbs*, *bin*, *rdbms*.
- Pliki danych (**data files**) służą do przechowywania danych systemowych i danych wprowadzanych przez użytkownika. Zgodnie z konwencją pliki bazy danych posiadają rozszerzenie *dbf*.
- Pliki kontrolne (**control files**) zawierają informacje o bazie danych, m.in. nazwy i położenie plików danych, plików dziennika powtórzeń, informacje o dostępności poszczególnych plików oraz dane wykorzystywane w czasie odtwarzania bazy danych po awarii. Przyjmuje się, że pliki te posiadają rozszerzenia *ctl* lub *dbf*.

Rysunek 3.1.1. Pliki kontrolne, danych i wycofywania.

- Pliki dziennika powtórzeń (**redo log files**) rejestrują wszystkie zatwierdzone operacje wykonywane na bazie danych. Pliki te są wykorzystywane m.in. do odtwarzania bazy danych po awarii. Wyróżnia się dwa rodzaje plików dziennika powtórzeń: aktywne (**online redo log files**) i zarchiwizowane (**archived redo log files**). Zgodnie z konwencją aktywne pliki dziennika powtórzeń posiadają rozszerzenia *log*, a zarchiwizowane *arc*.



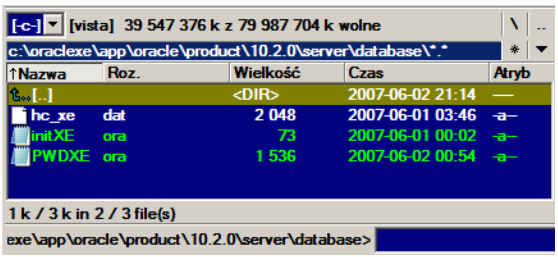
Rysunek 3.1.2. Pliki dziennika powtórzeń i ich archiwizacje.

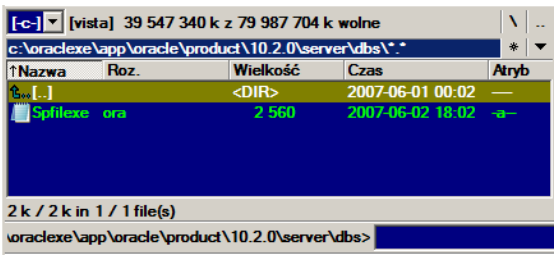
- Pliki kontrolne (**control files**) zawierają informacje o bazie danych, m.in. nazwy i położenie plików danych, plików dziennika powtórzeń, informacje o dostępności poszczególnych plików oraz dane wykorzystywane w czasie odtwarzania bazy danych po awarii. Przyjmuje się, że pliki te posiadają rozszerzenia *ctl* lub *dbf*.

Rysunek 3.1.3. Pliki kontrolne, danych, tymczasowe i wycofywania.

- Pliki konfiguracyjne (init files) zawierają parametry konfiguracyjne bazy danych. Są one odczytywane przez system w czasie uruchamiania bazy danych. Pliki te posiadają nazwy:
  - `init<SID>.ora` lub `spfile<SID>.ora`,
  - `pwd<SID>.ora`,
  - …

  gdzie `<SID>` oznacza nazwę instancji bazy danych, np. `initXE.ora`, `pwdXE.ora` lub `spfileXE.ora`.

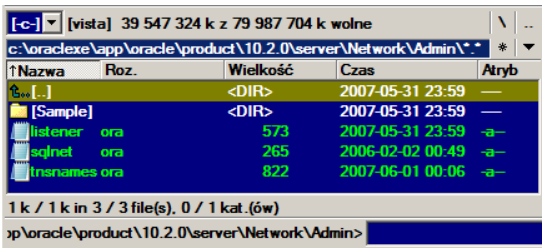




Rysunek 3.1.4. Pliki konfiguracyjne bazy.

- Pliki zawierają parametry konfiguracyjne sieci. Są one wykorzystywane przy połączeniach z instancją. Są to pliki np.

  `tnsnames.ora`, `listener.ora`, `sqlnet.ora` ….



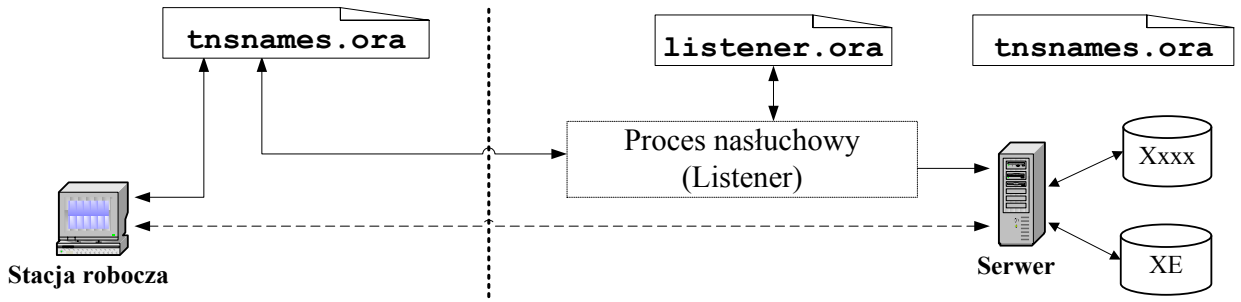
Rysunek 3.1.5. Sieciowe pliki konfiguracyjne.

4. POŁĄCZENIE Z INSTANCJĄ

§ 4.1. Sposób łączenia się z instancją

```
# TNSNAMES.ORA Network Configuration File:
#.............\network\admin\tnsnames.ora
# Generated by Oracle configuration tools.
XE =
 (DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)
    (HOST = c208_1)(PORT = 1521))
  )
  (CONNECT_DATA =
   (SERVER = DEDICATED)
   (SERVICE_NAME = XE)     /*nazwa
                                instancji*/
  )
 )
BAZA =
 (DESCRIPTION =
  (ADDRESS_LIST =
   (ADDRESS = (PROTOCOL = TCP)
   (HOST = zysknt.math.uni.lodz.pl)
                    (PORT = 1521))
   )
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = baza)      /*nazwa
                                  instancji*/
   )
 )
```

```
# LISTENER.ORA Network Configuration File:
# ……..\network\admin\listener.ora
# Generated by Oracle configuration tools.
LISTENER =
 (DESCRIPTION_LIST =
   (DESCRIPTION =
       (ADDRESS_LIST =
       (ADDRESS = (PROTOCOL = TCP)
       (HOST = c208_1)(PORT = 1521))
     )
   )
 )
SID_LIST_LISTENER =
 (SID_LIST =
   (SID_DESC =
     (GLOBAL_DBNAME = xe) (ORACLE_HOME =
C:\oraclexe\app\oracle\product\10.2.0\server)
        (SID_NAME = xe)
    )
    (SID_DESC =
        (GLOBAL_DBNAME = xxxx)
        (ORACLE_HOME =
C:\oraclexe\app\oracle\product\10.2.0\server)
        (SID_NAME = xxxx)
    )
 )
```



Rysunek 4.1.1. Proces nasłuchowy (Listener).

- **Na stacji roboczej musi być zainstalowany klient Oracle.**
- Na serwerze jest zainstalowany serwer Oracle i uruchomiony jest proces nasłuchowy **`Listener`**.
- Użytkownik łączy się podając nazwę użytkownika, hasło i identyfikator połączenia.
- Narzędzie klienckie (np. **`sqlplus`**) odczytuje parametry połączenia z lokalnego pliku **`tnsnames.ora`**.
- Sygnał idzie do programu nasłuchującego, który sprawdza:
  - Czy instancja serwera jest uruchomiona?
  - Czy identyfikator użytkownika i hasło jest poprawne?
- Proces nasłuchowy tworzy proces serwera odpowiedzialny za realizację żądań użytkownika.
- Do użytkownika zwracany jest adres i port (TCP/IP) tak utworzonego procesu.

Uwaga. Instancji korzystających z tych samych plików danych może być wiele. Umożliwia to równomierne rozłożenie obciążeń żądań wielu użytkowników.

§ 4.2. Klient Oracle

Podamy teraz przykład instalacji najprostszego klienta Oracle. W zasadzie każde narzędzie firmy Oracle zawiera klienta.

Z **http://www.oracle.com/technology/software/index.html** `Instant Client` można pobrać pliki

```
Instantclient-basic-win32-10.2.0.1-20050930.zip
InstantClientPackage-SQL*Plus.
```

- Pliki te należy rozpakować np. do katalogu **`c:\instantclient.`**
- W **`Mój Komputer->Właściwości->Zaawansowane->Zmienne środowiskowe`**

  Zmienne środowiskowe:
  - Edytujemy ścieżkę **`PATH`** i na początku dodajemy: **`c:\instantclient`**;
  - Dodajemy nową**`:`** **`SQLPATH`** o wartości **`c:\instantclient`**;
  - Dodajemy nową: **`TNS_ADMIN`** o wartości **`c:\instantclient`**;
  - Dodajemy nową: **`NLS_LANG`** o wartości POLISH_POLAND.WE8MSWIN1252
  - Plik tekstowy **`tnsnames.ora`** umieszczamy w katalogu **`c:\instantclient`**

Przykład pliku `tnsnames.ora`:

```
BAZA =
 (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)
                  (HOST = zysknt.math.uni.lodz.pl)(PORT = 1521))
   (CONNECT_DATA =
     (SERVER = DEDICATED)
     (SERVICE_NAME = baza)
   )
 )
```

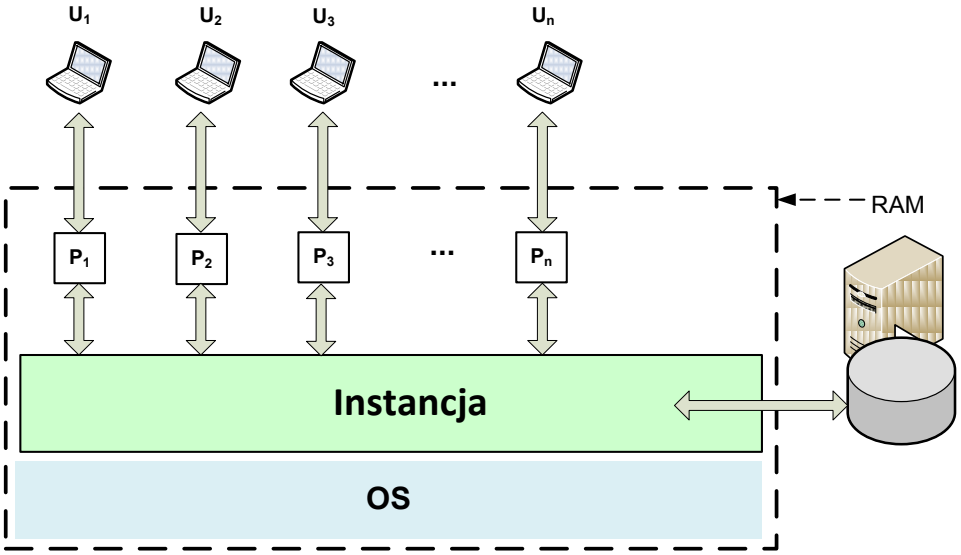
Po takiej instalacji dostępny jest z linii poleceń program `sqlplus`.

Poza tym można wykorzystywać wszystkie programy, które wymagają klienta `Oracle`.

Uwaga. Na serwerze, gdzie zainstalowany jest Oracle zainstalowany jest również klient Oracle.

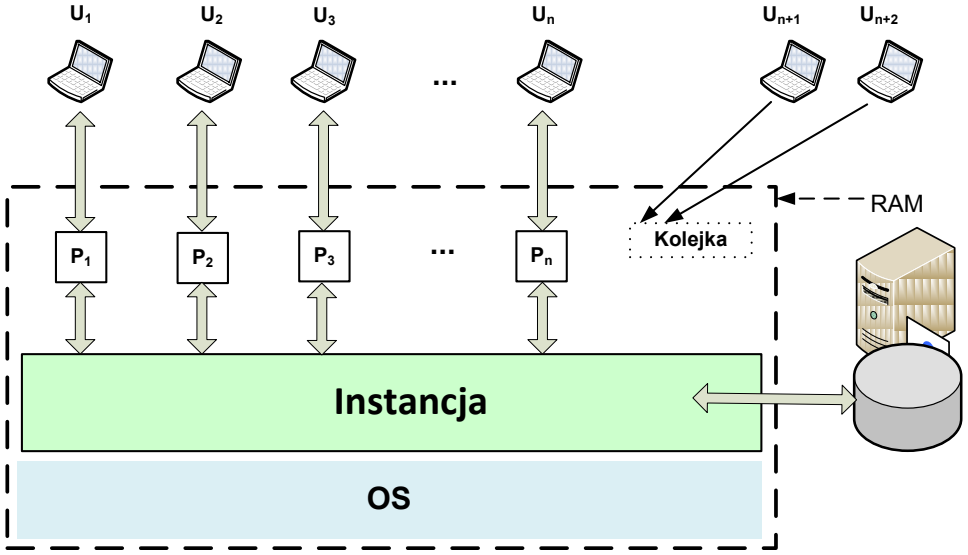
§ 4.3. Rodzaje serwerów baz danych Oracle

- Serwer dedykowany
  - Konfiguracja dla 1-100 użytkowników.
  - Dla każdego użytkownika tworzony jest oddzielny proces serwera wraz z oddzielnym obszarem pamięci (PGA).



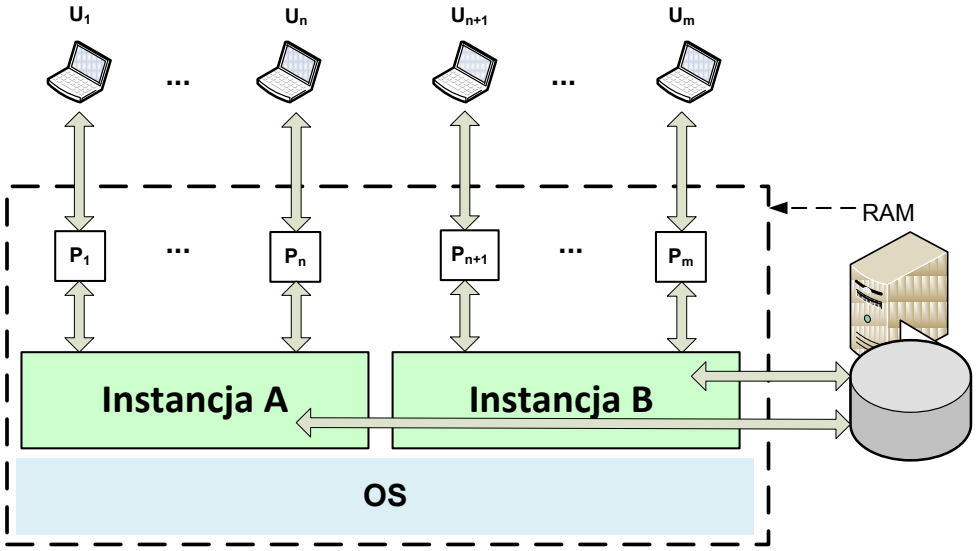
Rysunek 4.3.1. Serwer dedykowany.

- Serwer wielokanałowy
  - Konfiguracja dla 100-300 użytkowników.
  - Przy uruchamianiu systemu tworzona jest stała liczba procesów serwera. Żądanie użytkownika przydzielane jest do wolnego procesu lub ustawiane jest w kolejkę.



Rysunek 4.3.2. Serwer wielokanałowy.

- Serwer wieloinstancyjny
  - Dla jednego zbioru plików danych tworzonych jest kilka instancji.



Rysunek 4.3.3. Serwer wieloinstancyjny.

5. PROCESY BAZY DANYCH

§ 5.1. Rodzaje procesów

System Oracle wykorzystuje następujące trzy rodzaje procesów:

- Drugoplanowe.
- Usługowe.
- Użytkowników.

§ 5.2. Procesy drugoplanowe

Każda instancja bazy danych może posiadać od … do … różnych procesów zwanych drugoplanowymi (**backgroup processes**). Ich liczba jest zależna od konfiguracji, w jakiej pracuje baza danych.

Instancja Oracle składa się między innymi z następujących procesów drugoplanowych:

- Proces monitorowania procesów (ang. process monitor process – **PMON**). **Dokonuje on porządkowania po nieprawidłowym zakończeniu procesu użytkownika.** Wycofuje pozostawione niezatwierdzone transakcje i zwalnia zasoby zajęte przez nieistniejący już proces.
- Proces zapisujący do bazy danych (ang. database writer process – **DBWR**). Dla zapewnienia wydajnego i równoczesnego operowania na danych, **Oracle nie zezwala procesowi użytkownika na bezpośrednie modyfikowanie bloku danych na dysku**. Bloki, które muszą być zmodyfikowane lub te, do których wstawiane są dane, są najpierw przenoszone do **Bufora danych w SGA** (Database buffers). Bloki te są następnie zapisywane partiami na dysk przez proces drugoplanowy **DBWR**. **Tak, więc DBWR jest jedynym procesem, który ma prawa zapisu do plików Oracle**.
- Proces zapisujący do plików dziennika powtórzeń (ang. log writer process – **LGWR**). Za każdym razem, gdy proces Oracle modyfikuje blok danych, zapisuje również zatwierdzone zmiany do **Bufora dziennika powtórzeń w SGA** (Redo buffers). Proces **LGWR** jest odpowiedzialny za zapisywanie buforów **Bufora dziennika powtórzeń** do pliku dziennika powtórzeń. Proces ten odczytuje partiami zawartość buforów **Bufora dziennika powtórzeń** i zapisuje je sekwencyjnie do aktualnego pliku dziennika powtórzeń. Należy mieć na uwadze to, że **LGWR** jest jedynym procesem zapisującym do plików dziennika powtórzeń.
- Proces archiwizujący (ang. archiver process – **ARCH**). Proces ten jest uruchamiany, gdy baza danych znajduje się **w trybie archiwizowania dziennika powtórzeń** i włączone jest automatyczne archiwizowanie. **Kopiuje ostatnio zapełniony plik dziennika powtórzeń w miejsce przydzielone na kopię zapasową.**

- Proces monitorujący system (ang. system monitor process – **SMON**). Ten proces drugoplanowy wykonuje operacje takie, jak zwalnianie miejsca i scalanie przyległych wolnych obszarów w jeden duży obszar. **SMON jest również odpowiedzialny za odtwarzanie transakcji podczas odtwarzania instancji** (w czasie uruchamiania instancji po awarii lub po zamknięciu w trybie przerwania - ABORT).

- Proces kontrolny (ang. checkpoint process – **CKPT**). **W momencie kontrolnym proces DBWR zapisuje wszystkie zmodyfikowane bloki na dysk**.

- ……

§ 5.3. Procesy usługowe i procesy użytkowników

Użytkownik komunikuje się z instancją za pomocą procesu tworzonego w momencie uruchomienia aplikacji.

Dla każdego procesu użytkownika tworzony jest **jeden proces usługowy** (w przypadku, gdy serwer Oracle pracuje w trybie serwera dedykowanego), który wykonuje następujące operacje:

- Analizuje i optymalizuje składnię poleceń SQL.
- Wykonuje polecenia SQL.
- Odczytuje żądane dane z dysku.
- Przekazuje wyniki poleceń SQL do procesów użytkowników.

Każdy proces usługowy ma przydzielony obszar pamięci operacyjnej w **PGA** – Proces Global Area.

6. URUCHAMIANIE I ZAMYKANIE INSTANCJI

§ 6.1. Uruchamianie i zatrzymywanie serwisu nasłuchowego

Uruchamianie i zatrzymywanie serwisu nasłuchowego (`Listener`) można w środowisku Windows dokonać w następujący sposób:

Uruchamianie:

```
C:\lsnrctl.exe
LSNRCTL> start
```

lub

```
C:\net start nazwa_serwisu_listenera
```

Zatrzymywanie:

```
C:\lsnrctl.exe
LSNRCTL> stop
```

lub

```
C:\net stop nazwa_serwisu_listenera
```

§ 6.2. Uruchamianie i zatrzymywanie serwisu i instancji

Do uruchamiania serwisu instancji i instancji można użyć programu **`Oradim`** z parametrami:

```
C:\oradim -STARTUP -SID sid
                   [-STARTTYPE srvc | inst | srvc,inst]
                   [-PFILE filename | SPFILE]
```

gdzie:

`sid`	-	nazwa instancji,
`filename`	-	nazwa pliku konfiguracyjnego `init<SID>.ora`.

W przypadku braku parametru `-PFILE filename` lub `SPFILE` pliki `konfiguracyjne` szukane są w katalogach domyślnych, najpierw plik z parametrami typu `SPFILE` a w przypadku jego braku plik typu `PFILE`.

§ 6.3. Otwieranie serwisu instancji przy pomocy programu Oradim

Sam serwis instancji można uruchomić następująco:

```
C:\oradim -STARTUP -SID xe -STARTTYPE srvc
```

§ 6.4. Otwieranie instancji przy pomocy programu Oradim

Samą instancję można uruchomić następująco (**wcześniej musi być uruchomiony serwis**):

```
C:\oradim -STARTUP -SID xe -STARTTYPE inst
```

lub jednocześnie z serwisem:

```
C:\oradim -STARTUP -SID xe -STARTTYPE srvc,inst
```

```
lub jeszcze dodatkowo ze wskazaniem pliku z parametrami:
```

```
C:\oradim -STARTUP -SID xe -STARTTYPE srvc,inst
                                  -PFILE c:\initXE.ora
```

§ 6.5. Zatrzymywanie serwisu instancji i instancji

Zatrzymanie serwisu instancji i samej instancji można wykonać następująco:

```
C:\oradim -SHUTDOWN -SID xe -SHUTTYPE srvc,inst
```

§ 6.6. Otwieranie instancji przy pomocy SQL*Plus

Otwieranie samej instancji realizuje się za pomocą polecenia **STARTUP** lub **STARTUP OPEN**.

Poniższa sekwencja poleceń prowadzi do otwarcia instancji (**musi być uruchomiony proces nasłuchowy i serwis instancji**):

- Wywołanie programu SQL*Plus z opcją /nolog.

```
C:\sqlplus /nolog
```

- Dołączenie się do instancji jako użytkownik SYS lub SYSTEM z uprawnieniami SYSOPER lub SYSDBA – predefiniowany administrator bazy danych.

```
SQL> CONNECT sys/password@nazwa_instancji AS SYSDBA
```

lub

```
SQL> CONNECT sys AS SYSDBA
```

- Otwarcie instancji:

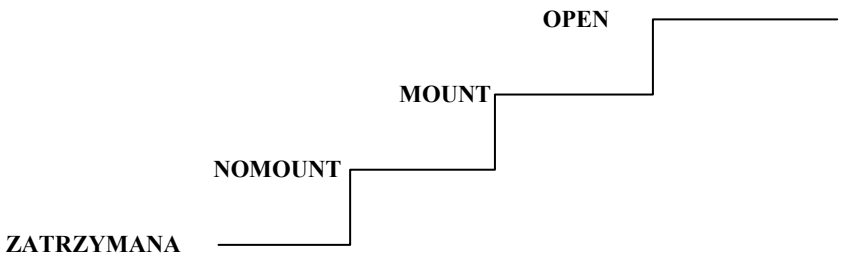
```
SQL> STARTUP
```

lub

```
SQL> STARTUP OPEN
Instancja ORACLE została uruchomiona.
Baza danych została zamontowana.
Baza danych została otwarta.
```

W wyniku wydania polecenia **STARTUP OPEN** lub **STARTUP** instancja przechodzi przez trzy tryby pracy: **NOMOUNT**, **MOUNT** i **OPEN**.

Administrator bazy danych może ją również uruchomić w trybie **NOMOUNT** i **MOUNT**.



Rysunek 6.6.1. Etapy uruchamiania instancji

Tryb NOMOUNT jest wykorzystywany m.in. do tworzeni bazy danych i plików kontrolnych.

W tym trybie Oracle:

- Odczytuje pliki parametrów konfiguracyjnych instancji init<SID>.ora lub spfile<SID>.ora. W szczególności znajduje nazwy i lokalizacje plików kontrolnych.
- Tworzy i inicjuje obszar pamięci **SGA**.
- Uruchamia procesy drugoplanowe.
- ….

Do uruchomienia instancji w trybie **NOMOUNT** służy polecenie:

```
SQL> STARTUP NOUMOUNT
```

Tryb MOUNT jest wykorzystywany m.in. do zmiany położenia plików danych, zmiany położenia, tworzenia nowych i usuwania istniejących plików dziennika powtórzeń, tworzenie kopi (backup), odtwarzania bazy danych (recovery).

W tym trybie Oracle:

- Odczytuje plik kontrolny w celu zlokalizowania plików danych i dziennika powtórzeń.
- Przyłącza pliki danych i pliki dziennika powtórzeń.

Do uruchomienia instancji z bazą danych w trybie **MOUNT** służy polecenie

```
SQL> STARTUP MOUNT
```

lub z trybu NOMOUNT poleceniem

```
SQL> ALTER DATABASE MOUNT;
```

W trybie **OPEN** są otwierane pliki danych i pliki dziennika powtórzeń, co powoduje, że baza danych staje się dostępna dla użytkowników. Tryb ten jest również wykorzystywany do odtwarzania części bazy danych po awarii.

Do uruchomienia instancji z bazą danych w trybie **OPEN** służy polecenie

```
SQL> STARTUP [OPEN]
```

lub gdy baza jest w trybie MOUNT poleceniem

```
SQL> ALTER DATABASE OPEN;
```

§ 6.7. Zamykanie instancji

Zamykanie instancji przebiega w trzech fazach. W fazie pierwszej są zamykane pliki danych i pliki dziennika powtórzeń, w drugiej instancja odłącza te pliki i zamyka pliki kontrolne. W fazie trzeciej są usuwane procesy bazy danych i zwalniana jest pamięć zaalokowana dla obszaru SGA.

Instancję można zamknąć w czterech trybach:

NORMAL, TRANSACTIONAL, IMMEDIATE, ABORT.

- Zamknięcie instancji w trybie **NORMAL** wykonuje się za pomocą polecenia

```
SQL> SHUTDOWN [NORMAL]
```

Zamknięcie w tym trybie czeka na zakończenie wszystkich sesji (blokuje otwarcie nowych sesji).

- Zamknięcie instancji w trybie **TRANSACTIONAL** wykonuje się za pomocą polecenia

```
SQL> SHUTDOWN TRANSACTIONAL
```

Zamknięcie w tym trybie czeka na zakończenie wszystkich aktualnych transakcji użytkownika (blokuje rozpoczęcie nowych transakcji) i odłącza.

- Do zamknięcia instancji w trybie **IMMEDIATE** służy polecenie

```
SQL> SHUTDOWN IMMEDIATE
```

Tryb ten różni się od trybu NORMAL tym, że aktualnie wykonywane transakcje są przerywane i wszystkie aktywne transakcje są natychmiast wycofywane (blokuje rozpoczęcie nowych transakcji i otwarcie nowych sesji).

- W trybie **ABORT** instancję zamyka się za pomocą polecenia

```
SQL> SHUTDOWN ABORT
```

Polecenie to stosuje się, gdy instancji nie można zamknąć ani w trybie NORMAL ani IMMEDIATE. W wyniku wykonania polecenia SHUTDOWN ABORT wszystkie aktualnie wykonywane polecenia są natychmiast przerywane, ale transakcje nie są wycofywane.

§ 6.8. Przykłady uruchamiania i zamykania instancji

Instancja ma nazwę **xe**.

```
C:\sqlplus /nolog
SQL*Plus: Release 10.2.0.1.0 - Production on N Mar 11 18:18:17 2007
Copyright (c) 1982, 2005, Oracle.  All rights reserved.
SQL> CONNECT sys AS SYSDBA
Enter password:******  - Proszę podać hasło: ******
Connected.             - Połączono
SQL> SHUTDOWN
Database closed.            - Baza danych została zamknięta.
Database dismounted.        - Baza danych została zdemontowana.
ORACLE instance shut down.  - Instancja ORACLE została zamknięta.
SQL> desc all_users
ERROR: ORA-...: ORACLE not available
SQL> STARTUP
ORACLE instance started.                - Instancja ORACLE została uruchomiona.
   Total System Global Area  285212672 bytes
   Fixed Size                  1287016 bytes
   Variable Size             100666520 bytes
   Database Buffers          180355072 bytes
   Redo Buffers                2904064 bytes
Database mounted.                       - Baza danych została zamontowana.
Database opened.                        - Baza danych została otwarta.
SQL> SHUTDOWN
Database closed.            - Baza danych została zamknięta.
Database dismounted.        - Baza danych została zdemontowana.
ORACLE instance shut down.  - Instancja ORACLE została zamknięta.
```

Może się zdarzyć, że musimy wykonać jakieś operacje na otwartej bazie i nie chcemy, aby inni użytkownicy logowali się do bazy. Wtedy należy otworzyć bazę z opcją RESTRICT.

```
SQL> STARTUP RESTRICT
ORACLE instance started.                - Instancja ORACLE została uruchomiona.
   Total System Global Area  285212672 bytes
   ...
   Redo Buffers                2904064 bytes
Database mounted.                       - Baza danych została zamontowana.
Database opened.                        - Baza danych została otwarta.
SQL> CONNECT kadry/kadry
   ERROR: ORA-XXXXX: ORACLE only available to users with RESTRICTED SESSION privilege
```

W tym trybie otwarcia bazy tylko użytkownicy posiadający uprawnienie systemowe RESTRICTED SESSION mogą się z nią połączyć.

```
SQL> CONNECT sys AS SYSDBA
Enter password:******  - Proszę podać hasło: ******
Connected.             - Połączono
SQL> GRANT RESTRICTED SESSION TO kadry;
   Grant succeeded.             - Przyznanie uprawnień zakończone powodzeniem.
SQL> CONNECT kadry/kadry
```

```
Connected.                    - Połączono.
SQL> CONNECT sys AS SYSDBA
Enter password:******  - Proszę podać hasło: ******
Connected.             - Połączono
SQL> REVOKE RESTRICTED SESSION FROM kadry;
Revoke succeeded.             - Pozbawienie uprawnień zakończone powodzeniem.
SQL> SHUTDOWN
Database closed.              - Baza danych została zamknięta.
Database dismounted.          - Baza danych została zdemontowana.
ORACLE instance shut down.    - Instancja ORACLE została zamknięta.
SQL> STARTUP OPEN
ORACLE instance started.              - Instancja ORACLE została uruchomiona.
Total System Global Area  285212672 bytes
...
Redo Buffers              2904064 bytes
Database mounted.                     - Baza danych została zamontowana.
Database opened.                      - Baza danych została otwarta.
SQL> SHUTDOWN IMMEDIATE
Database closed.              - Baza danych została zamknięta.
Database dismounted.          - Baza danych została zdemontowana.
ORACLE instance shut down.    - Instancja ORACLE została zamknięta.
SQL> STARTUP MOUNT
ORACLE instance started.              - Instancja ORACLE została uruchomiona.
Total System Global Area  285212672 bytes
...
Redo Buffers              2904064 bytes
Database mounted.                     - Baza danych została zamontowana.
SQL> STARTUP
ORA-XXXXX: cannot start already-running ORACLE-shut it down first
ORA-XXXXX: nie można startować już aktywnej ORACLE - należy ja najpierw zamknąć
SQL> SHUTDOWN
ORA- XXXXX: baza danych nie jest zamontowana
Instancja ORACLE została zamknięta.
SQL> STARTUP NOMOUNT
ORACLE instance started.              - Instancja ORACLE została uruchomiona.
Total System Global Area  285212672 bytes
...
Redo Buffers              2904064 bytes
```

Z poziomu **MOUNT** można przejść do poziomu **OPEN** otwarcia poleceniem:

```
SQL> ALTER DATABASE OPEN;
Database altered.
```

Z poziomu **NOMOUNT** nie można tym poleceniem przejść do poziomu **OPEN**. Można jednak wykonać następujące polecenia:

```
SQL> ALTER DATABASE MOUNT;
Database altered.
SQL> ALTER DATABASE OPEN;
Database altered.
```

Uwaga. Poleceniem ALTER DATABASE nie można obniżać poziomu otwarcia.

Od poziomu otwarcia NOMOUNT można odczytać poziom otwarcia z perspektywy dynamicznej V$INSTANCE poleceniem:

```
SQL> SELECT status FROM V$INSTANCE;
```

Od poziomu otwarcia MOUNT można odczytać poziom otwarcia z perspektywy dynamicznej V$DATABASE poleceniem:

```
SQL> SELECT name, open_mode FROM V$DATABASE;
```

Uwaga. Niektóre perspektywy dynamiczne dostępne są już od poziomu NOMOUNT (np.VSGA, VPARAMETER, V$INSTANCE…).

7. PARAMETRY BAZY DANYCH ORACLE

§ 7.1. Zmiany parametrów

W najnowszych wersjach Oracle uruchamiany system korzysta z binarnego piku parametrów **spfile<SID>.ora**. Jest możliwość wystartowania z parametrami z odpowiednika, pliku tekstowego **init<SID>.ora**.

W pliku binarnym SPFILE możemy zmieniać wartość parametrów z poziomu SQL.

W pliku tekstowym PFILE jest to niemożliwe.

Sposoby tworzenia tych plików pokazują następne przykłady (z konta użytkownika z uprawnieniami SYSDBA).

Poleceniem

```
SQL> CREATE SPFILE=
    'c:\oraclexe\app\oracle\product\10.2.0\server\dbs\spfileXE.ora'
     FROM PFILE=
'c:\oraclexe\app\oracle\product\10.2.0\server\database\initXE.ora';
```

można utworzyć nowy plik **spfileXE.ora** i restartować instancję.

Użyty zostanie plik parametrów **spfileXEt.ora** (w przypadku jego braku użyty będzie plik **initXE.ora**).

Poleceniem

```
SQL> CREATE PFILE=
 'c:\oraclexe\app\oracle\product\10.2.0\server\database\initxe.ora'
     FROM SPFILE=
  'c:\oraclexe\app\oracle\product\10.2.0\server\dbs\spfilexe.ora';
```

można odtworzyć plik **initXE.ora** i zmodyfikować go. Następnie znowu utworzyć plik **spfileXE.ora** i restartować instancję.

W czasie pracy instancji zmiana parametrów systemu może być zmieniona w pliku **spfile<sid>.ora.**

```
SQL> ALTER SYSTEM SET RESOURCE_LIMIT=TRUE SCOPE=BOTH;
                                          /*MEMORY|SPFILE|BOTH*/
```

Uwaga. Nie wszystkie parametry można zmienić z opcją MEMORY. Tak więc dla niektórych parametrów polecenie ze SCOPE=BOTH lub SCOPE=MEMEORY zakończy się błędem.

Przykład 7.1.1. Zmiana parametru SGA_MAX_SIZE określającego maksymalny rozmiar obszaru SGA z opcją **SCOPE=BOTH** zakończy się błędem.

```
SQL> ALTER SYSTEM SET sga_max_size=536870912 SCOPE=BOTH;
     ERROR at line 1:
     ORA-02095: specified initialization parameter cannot be modified
```

Przykład 7.1.2. Sposób zmiany pewnych parametrów z poziomu SQL pokazują następujące przykłady:

```
SQL> ALTER SYSTEM SET sql_trace=TRUE SCOPE=MEMORY;
      System altered.
```

Po restarcie systemu obowiązuje poprzednia wartość.

```
SQL> ALTER SYSTEM SET sql_trace =TRUE SCOPE=SPFILE;
      System altered.
```

Ustawienia obowiązują dopiero po restarcie systemu.

```
SQL> ALTER SYSTEM SET sql_trace =TRUE SCOPE=BOTH;
      System altered.
```

Ustawienia obowiązują natychmiast i po restarcie.

§ 7.2. Przykładowy plik init<SID>.ora

W pliku **init<SID>.ora** (**spfile<SID>.ora**) ustawiane są między innymi następujące parametry:

DB_NAME	-	Identyfikator bazy danych (maks. 8 znaków).
CONTROL_FILES	-	**Nazwy plików sterujących.**
OPEN_CURSORS		**Ilość jednocześnie otwartych kursorów w czasie sesji.**
DB_BLOCK_SIZE	-	Rozmiar bloku bazy danych w bajtach.
SHARED_POOL_SIZE	-	**Rozmiar współdzielonej puli w bajtach.**
BACKGROUND_DUMP_DEST	-	Miejsce, w którym przechowywane będą pliki do śledzenia procesów drugoplanowych.
USER_DUM_DEST	-	Miejsce, w którym przechowywane będą pliki do śledzenia procesów użytkownika.
COMPATIBILE	-	Wersja serwera, z którą zgodna jest instancja.
PROCESSES	-	Maksymalna liczba procesów SO mogących jednocześnie łączyć się z instancją.
LOG_ARCHIVE_START	-	**Automatyczne archiwizowanie.**
LOG_ARCHIVE_FORMAT	-	Format nazwy plików archiwum.
LOG_ARCHIVE_DEST	-	Miejsce archiwizowania plików rejestru.
UNDO_MANAGEMENT	-	**Metoda zarządzania wycofywaniem transakcji.**
HASH_JOIN_ENABLED		Dostęp do metody Hash Join w optymalizatorze zapytań.
…		…

Przykładowy plik **initXE.ora** (wersja z instalacji Expres Edition).

```
db_cache_size=180355072
java_pool_size=4194304
large_pool_size=8388608
shared_pool_size=88080384
streams_pool_size=0
audit_file_dest='c:\oraclexe\app\oracle\admin\xe\adump'
background_dump_dest='c:\oraclexe\app\oracle\admin\xe\bdump'
compatible='10.2.0.1.0'
control_files='c:\oraclexe\oradata\xe\control.dbf'
core_dump_dest='c:\oraclexe\app\oracle\admin\xe\cdump'
db_name='XE'
DB_RECOVERY_FILE_DEST_SIZE=10G
DB_RECOVERY_FILE_DEST='c:\oraclexe\app\oracle\flash_recovery_area'
dispatchers='(PROTOCOL=TCP) (SERVICE=xeXDB)'
job_queue_processes=4
open_cursors=300
os_authent_prefix=''
pga_aggregate_target=90M
remote_login_passwordfile='EXCLUSIVE'
```

```
sessions=20
sga_target=270M
shared_servers=4
undo_management='AUTO'
undo_tablespace='UNDO'
user_dump_dest='c:\oraclexe\app\oracle\admin\xe\udump'
```

Wartość wszystkich parametrów (jest ich w wersji 10/11g około 250) można odczytać z dynamicznej perspektywy **V$PARAMETER** już od stanu **NOMOUNT**.

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

Uwaga. Wartości parametrów nie występujących w pliku init<SID>.ora lub spfile<SID>.ora przyjmują wartości domyślne.

8. ZARZĄDZANIE BEZPIECZEŃSTWEM

Zapewnienie bezpieczeństwa systemu jest jednym z najważniejszych zadań stawianych przed administratorem bazy danych.

§ 8.1. Użytkownicy

Konta użytkowników tworzy się poleceniem **CREATE USER**.

Składnia polecenia **CREATE USER:**

```
SQL> CREATE USER nazwa_użytkownika IDENTIFIED BY hasło
     [DEFAULT TABLESPACE nazwa_przestrzeni]
     [TEMPORARY TABLESPACE nazwa_przestrzeni]
     [QUOTA {liczba [K | M] | UNLIMITED} ON nazwa_przestrzeni]...
     [PROFILE nazwa_profilu]
     [PASSWORD EXPIRE]
     [ACCOUNT { LOCK | UNLOCK }];
```

Do zmiany parametrów użytkownika bazy służy polecenie **ALTER USER**.

Składnia polecenia **ALTER USER:**

```
SQL> ALTER USER nazwa_użytkownika [ IDENTIFIED BY hasło ]
     [DEFAULT TABLESPACE nazwa_przestrzeni]
     [TEMPORARY TABLESPACE nazwa_przestrzeni]
     [QUOTA {liczba [K | M] | UNLIMITED} ON nazwa_przestrzeni]...
     [PROFILE nazwa_profilu]
     [PASSWORD EXPIRE]
     [ACCOUNT { LOCK | UNLOCK }];
```

gdzie:

nazwa_użytkownika	-	Nazwa tworzonego (zmienianego) użytkownika, składająca się z maksymalnie 30 znaków, których wielkość liter nie jest istotna.
hasło	-	Hasło podawane w przypadku uwierzytelniania poprzez system zarządzania bazą danych.
	-	Składa się maksymalnie z 30 znaków, w których wielkość liter nie jest istotna. Przechowywane w postaci zaszyfrowanej w kolumnie **PASSWORD** tabeli **SYS.USER$,** jest również udostępniane przez kolumnę **PASSWORD** perspektywy **DBA_USERS**.
DEFAULT TABLESPACE	-	Określa domyślną przestrzeń tabel, w której będą tworzone obiekty użytkownika. W przypadku pominięcia tego parametru domyślną przestrzenią tabel dla użytkownika będzie **SYSTEM.** Parametr ten nie ogranicza możliwości tworzenia obiektów w innych przestrzeniach tabel.
TEMPORARY TABLESPACE	-	Określa tymczasową przestrzeń tabel, w której powstają segmenty tymczasowe użytkownika wykorzystywane podczas operacji sortowania dużej ilości danych.
QUOTA	-	Pozwala określić limit miejsca dostępnego dla użytkownika w podanej przestrzeni tabel, wyrażony w bajtach (domyślnie), kilobajtach K, megabajtach M. Użycie słowa **UNLIMITED** powoduje przyznanie nieograniczonego limitu.
nazwa_profilu	-	Przypisuje użytkownikowi profil. Pominięcie tej klauzuli powoduje przyznanie profilu **DEFALUT**.
PASSWORD EXPIRE	-	Ustawia status konta oznaczający wygaśnięcie hasła.
ACCOUNT	-	Blokuje (**LOCK**) lub odblokowuje (**UNLOCK**) konto użytkownika (zmienia status konta).

Uwagi.

- Użytkownik, który tworzy użytkowników musi posiadać uprawnienie systemowe **CREATE USER**.
- Do zmiany użytkowników wymagane jest posiadanie uprawnienia **ALTER USER**.
- Użytkownik nieposiadający przyznanego uprawnienia `ALTER USER` może jednak zmienić swoje hasło poleceniem `ALTER USER`.
- W poleceniu `CREATE USER` parametry `DEFAULT TABLESPACE`, `TEMPORARY TABLESPACE` są opcjonalne, jednak zaleca się ich używać i przypisywać inne przestrzenie tabel niż domyślna `SYSTEM`.
- W momencie tworzenia użytkownika tworzony jest jednocześnie schemat użytkownika o tej samej nazwie.
- Schemat to nazwany zestaw obiektów takich jak: tabele, indeksy, ograniczenia, wyzwalacze, perspektywy, pakiety, procedury, funkcje, sekwencje. Schemat jest ściśle związany z użytkownikiem, dlatego często się go z nim utożsamia.
- Usuwanie użytkowników wykonuje się poleceniem `DROP USER`, do jego wykonywania wymagane jest uprawnienie systemowe **DROP USER**.

Składnia polecenia **DROP USER:**

```
SQL> DROP USER nazwa_użytkownika [CASCADE];
```

gdzie:

`CASCADE`	-	Użycie tej opcji powoduje usunięcie wszystkich obiektów użytkownika przed usunięciem jego samego.

§ 8.2. Przykłady

Przykład 8.2.1. Poniższe polecenie tworzy użytkownika `kowalski` z hasłem `jan`, który będzie miał przypisaną domyślną przestrzeń tabel **users** oraz tymczasową **temp**. Użytkownik ten będzie mógł tworzyć obiekty (po przyznaniu odpowiednich uprawnień systemowych) w przestrzeniach **users** i **system**, ilość zajętej przez niego przestrzeni nie może przekroczyć 1 MB, w każdej z nich.

```
SQL> CREATE USER kowalski
     IDENTIFIED BY jan
     DEFAULT TABLESPACE users
     TEMPORARY TABLESPACE temp
     QUOTA 1M ON users
     QUOTA 1M ON system;
```

Jeżeli w schemacie `kowalski` nie istnieją żadne obiekty, to usuniemy go poleceniem

```
SQL> DROP USER kowalski;
```

Użytkownik zostanie usunięty, w przeciwnym razie zostanie zgłoszony błąd:

```
ORA-01922: potrzebna specyfikacja CASCADE, aby usunąć 'KOWALSKI'.
```

Wtedy usuniemy go poleceniem

```
SQL> DROP USER kowalski CASCADE;
```

Przykład 8.2.2. Przykład ten pokazuje jak można połączyć się jako dowolny użytkownik nie znając jego hasła. Aby tego dokonać trzeba posiadać uprawnienie systemowe **ALTER USER** oraz prawo odczytywania perspektywy **DBA_USERS**. Załóżmy, że chcemy połączyć się jako użytkownik **kadry**. Korzystając z perspektywy **DBA_USERS** sprawdzamy zakodowane hasło użytkownika.

```
SQL> SELECT password
     FROM dba_users
     WHERE username = 'KADRY';

PASSWORD
----------
728513B2EDA48756
```

Zmieniamy hasło użytkownika **kadry** i łączymy się jako **kadry** z nowym hasłem.

```
SQL> ALTER USER kadry IDENTIFIED BY nowe_hasło;
Użytkownik został zmieniony.
SQL> CONNECT kadry/nowe_hasło;
Połączony.
```

A teraz korzystając z nieudokumentowanej opcji **IDENTIFIED BY VALUES** polecenia ALTER USER … przywracamy poprzednie hasło użytkownika.

```
SQL> ALTER USER kadry IDENTIFIED BY VALUES '728513B2EDA48756';
```

Perspektywa **DBA_USERS** pokazuje to samo zakodowane hasło, a użytkownik może się łączyć przy użyciu swojego starego hasła.

```
SQL> SELECT password
     FROM dba_users
     WHERE username = 'KADRY';

PASSWORD
----------
728513B2EDA48756

SQL> CONNECT kadry/stare_hasło
Połączony.
```

§ 8.3. Zakończenie sesji użytkownika

Zakończenia sesji użytkownika podłączonego do bazy danych:

- Uniemożliwia użytkownikowi wykonanie dalszych poleceń w bazie.
- Zwalnia zablokowane zasoby.
- Wyświetli użytkownikowi komunikat.
- Wymaga uprawnienia ALTER SYSTEM.

Sesję użytkownika należy zakończyć gdy:

- Użytkownik przetrzymuje zasoby pilnie potrzebne innemu użytkownikowi.
- DBA musi zamknąć bazę danych.

Sesję użytkownika kończy się wydając polecenie

```
SQL> ALTER SYSTEM KILL SESSION 'sid, serial#';
```

gdzie:

sid	- określa nr sesji użytkownika (SESSION ID),
serial#	- określa nr seryjny użytkownika.

Polecenie **ALTER SYSTEM KILL SESSION …:**

- **Wycofuje aktualną transakcję użytkownika.**
- Zwalnia wszystkie aktualnie posiadane przez użytkownika blokady na tabelach i wierszach.
- Zwalnia wszystkie zasoby zarezerwowane dla użytkownika.

Nr identyfikacyjny sesji i nr seryjny sesji użytkownika można odszukać w perspektywie V$SESSION.

Przykład 8.3.1. Wykonanie kroków potrzebnych do zakończenia sesji użytkownika kadry.

```
SQL> SELECT sid, serial#, username FROM v$session
     WHERE username='KADRY';
      SID      SERIAL#    USERNAME
      8          103        KADRY
SQL> ALTER SYSTEM KILL SESSION '8,103';
```

§ 8.4. Profile

Profile są nazwanymi zbiorami limitów, dostarczającymi mechanizmu pozwalającego:

- Ograniczyć przydzielone zasoby systemu (np. liczba jednoczesnych sesji użytkownika, czas bezczynności, …).
- Prowadzić pewną politykę zarządzania hasłami.

Uwaga. Profili może być wiele, ale jeden użytkownik posiada tylko jeden profil. Po utworzeniu bazy danych istnieje jeden profil o nazwie **DEFAULT (prawie bez żadnych ograniczeń)**.

Aby kontrola limitów zdefiniowanych przez profile był dokonywana, wymagane jest włączenie jej poleceniem

```
SQL> ALTER SYSTEM SET RESOURCE_LIMIT = TRUE;
```

lub ustawienie wartości parametru inicjalizacji (w **init<SID>.ora** lub **spfile<SID>.ora**)

```
RESOURCE_LIMIT = TRUE
```

co zapewnia jej automatyczne włączenie również przy kolejnym uruchomieniu bazy.

Do tworzenia profili służy polecenie **CREATE PROFILE**. Aby je wykonywać wymagane jest posiadanie uprawnienia systemowego **CREATE PROFILE**.

Składnia polecenia **CREATE PROFILE:**

```
CREATE PROFILE profile LIMIT
[SESSIONS_PER_USER          {liczba | UNLIMITED | DEFAULT}]
[CPU_PER_SESSION            {liczba | UNLIMITED | DEFAULT}]
[CPU_PER_CALL               {liczba | UNLIMITED | DEFAULT}]
[CONNECT_TIME               {liczba | UNLIMITED | DEFAULT}]
[IDLE_TIME                  {liczba | UNLIMITED | DEFAULT}]
[LOGICAL_READS_PER_SESSION  {liczba | UNLIMITED | DEFAULT}]
[LOGICAL_READS_PER_CALL     {liczba | UNLIMITED | DEFAULT}]
[COMPOSITE_LIMIT            {liczba | UNLIMITED | DEFAULT}]
```

```
  [PRIVATE_SGA                 {liczba [K|M]| UNLIMITED | DEFAULT}]
  [FAILED_LOGIN_ATTEMPTS       {liczba | UNLIMITED | DEFAULT}]
  [PASSWORD_LIFE_TIME          {liczba | UNLIMITED | DEFAULT}]
  [{PASSWORD_REUSE_TIME | PASSWORD_REUSE_MAX}
                                  { liczba | UNLIMITED | DEFAULT}]
  [PASSWORD_LOCK_TIME          { liczba | UNLIMITED | DEFAULT}]
  [PASSWORD_GRACE_TIME         { liczba | UNLIMITED | DEFAULT}]
  [PASSWORD_VERIFY_FUNCTION    {function | NULL | DEFAULT}];
```

Profile można zmieniać poleceniem **ALTER PROFILE** wymagającego uprawnienia systemowego **ALTER PROFILE**.

Składnia polecenia **ALTER PROFILE**:

```
 ALTER PROFILE profile LIMIT
  [SESSIONS_PER_USER           {liczba | UNLIMITED | DEFAULT}]
  [CPU_PER_SESSION             {liczba | UNLIMITED | DEFAULT}]
  [CPU_PER_CALL                {liczba | UNLIMITED | DEFAULT}]
  [CONNECT_TIME                {liczba | UNLIMITED | DEFAULT}]
  [IDLE_TIME                   {liczba | UNLIMITED | DEFAULT}]
  [LOGICAL_READS_PER_SESSION   {liczba | UNLIMITED | DEFAULT}]
  [LOGICAL_READS_PER_CALL      {liczba | UNLIMITED | DEFAULT}]
  [COMPOSITE_LIMIT             {liczba | UNLIMITED | DEFAULT}]
  [PRIVATE_SGA                 {liczba [K|M]| UNLIMITED | DEFAULT}]
  [FAILED_LOGIN_ATTEMPTS       {liczba | UNLIMITED | DEFAULT}]
  [PASSWORD_LIFE_TIME          {liczba | UNLIMITED | DEFAULT}]
  [{PASSWORD_REUSE_TIME | PASSWORD_REUSE_MAX}
                                  { liczba | UNLIMITED | DEFAULT}]
  [PASSWORD_LOCK_TIME          { liczba | UNLIMITED | DEFAULT}]
  [PASSWORD_GRACE_TIME         { liczba | UNLIMITED | DEFAULT}]
  [PASSWORD_VERIFY_FUNCTION    {function | NULL | DEFAULT}];
```

gdzie:

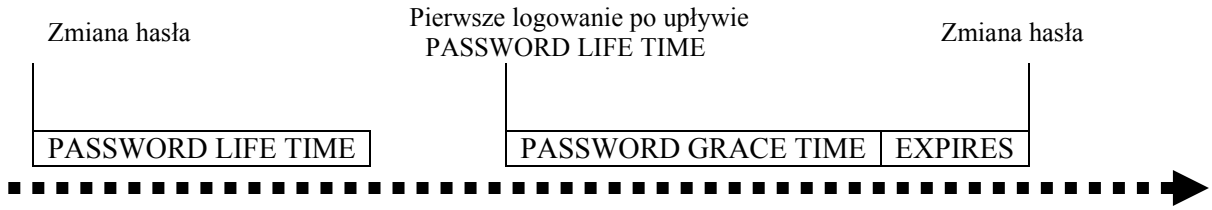
profile	-	Nazwa tworzonego / zmienianego profilu.
SESSIONS_PER_USER	-	Ogranicza liczbę równocześnie otwartych sesji użytkownika.
CPU_PER_SESSION	-	Ogranicza czas procesora dla sesji, podany w setnych częściach sekundy.
CPU_PER_CALL	-	Ogranicza czas procesora dla wywołania (parse, execute, fetch), podany w setnych częściach sekundy.
CONNECT_TIME	-	Ogranicza czas trwania sesji, podany w minutach.
IDLE_TIME	-	Ogranicza okres ciągłej bezczynności podczas sesji, podany w minutach. Długotrwałe operacje (np. zapytania) nie są brane pod uwagę przy ustalaniu czasu bezczynności, nawet, jeśli aplikacja nie wykonuje żadnych operacji.
LOGICAL_READS_PER_SESSION	-	Ogranicza liczbę odczytanych bloków danych w sesji. Pod uwagę brane są bloki odczytane, z dysku jak i z pamięci operacyjnej.
LOGICAL_READS_PER_CALL	-	Ogranicza liczbę odczytanych bloków (z dysku i z pamięci) podczas jednego wywołania polecenia SQL.
COMPOSITE_LIMIT	-	Ogranicza wykorzystanie całkowitego kosztu zasobów dla sesji. System oblicza całkowity koszt zasobów jako sumę ważoną wartości: CPU_PER_SESSION, CONNECT_TIME, LOGICAL_READS_PER_SESSION, PRIVATE_SGA, którą można określić poleceniem ALTER RESOURCE COST.
PRIVATE_SGA	-	Ogranicza ilość prywatnej przestrzeni sesji przydzielonej w SHARED POOL obszaru SGA. Wielkość podana w bajtach, kilobajtach (K), lub megabajtach (M). Ograniczenie ma sens jedynie w przypadku serwera wielowątkowego.

FAILED_LOGIN_ATTEMPTS	- Określa ilość nieudanych prób logowania na konto użytkownika, po których nastąpi zablokowanie konta na czas zależny od wartości PASSWORD_LOCK_TIME.
PASSWORD_LIFETIME	- Ograniczają okres ważności hasła.
PASSWORD_GRACE_TIME	- Po upłynięciu PASSWORD_LIFETIME dni od ostatniej zmiany hasła, przy pierwszym logowaniu zaczyna być obliczany okres PASSWORD_GRACE_TIME, po którym hasło wygasa. Jednocześnie w okresie PASSWORD_GRACE_TIME, użytkownik jest informowany o liczbie dni pozostałych do wygaśnięcia.
PASSWORD_REUSE_TIME	- Określa liczbę dni, przez jaką hasło nie może być ponownie wykorzystane podczas zmiany hasła przez użytkownika. Jeżeli wartość PASSWORD_REUSE_TIME ma wartość liczbową, to wartość PASSWORD_REUSE_MAX musi być ustalona na UNLIMITED.
PASSWORD_REUSE_MAX	- Określa liczbę zmian haseł, po której hasło może być ponownie wykorzystane podczas zmiany hasła przez użytkownika. Jeżeli wartość PASSWORD_REUSE_MAX ma wartość liczbową, to wartość PASSWORD_REUSE_TIME musi być ustalona na UNLIMITED.
PASSWORD_LOCK_TIME	- Określa liczbę dni, przez jakie konto będzie zablokowane po określonej przez FAILED_LOGIN_ATTEMPTS nieudanych próbach logowania.
PASSWORD_VERIFY_FUNCTION	- Nazwa funkcji weryfikacji hasła. Funkcja ta musi być umieszczona w schemacie użytkownika SYS i posiadać nagłówek następującej postaci: **FUNCTION nazwa_funkcji** **(** **nazwa_użytkownika IN VARCHAR2,** **haslo IN VARCHAR2,** **poprzednie_hasło IN VARCHAR2** **)** **RETURN BOOLEAN**

Uwaga. Zmiany dokonane poleceniem ALTER PROFILE nie wpływają na już rozpoczęte sesje.

Rysunek poniżej przedstawia znaczenie parametrów:

PASSWORD_LIFETIME i **PASSWORD_GRACE_TIME.**



Rysunek 8.4.1. Znaczenie parametrów **PASSWORD_LIFETIME** i **PASSWORD_GRACE_TIME**

Ustawienia wybranego profilu (np. DEFAULT) w bazie można otrzymać poleceniem:

```
SQL> SELECT * FROM dba_profiles WHERE profile='DEFAULT';
```

PROFILE	RESOURCE_NAME	RESOURCE_TYPE	LIMIT
DEFAULT	COMPOSITE_LIMIT	KERNEL	UNLIMITED
DEFAULT	SESSIONS_PER_USER	KERNEL	UNLIMITED
DEFAULT	CPU_PER_SESSION	KERNEL	UNLIMITED
DEFAULT	CPU_PER_CALL	KERNEL	UNLIMITED
DEFAULT	LOGICAL_READS_PER_SESSION	KERNEL	UNLIMITED
DEFAULT	LOGICAL_READS_PER_CALL	KERNEL	UNLIMITED
DEFAULT	IDLE_TIME	KERNEL	UNLIMITED
DEFAULT	CONNECT_TIME	KERNEL	UNLIMITED
DEFAULT	PRIVATE_SGA	KERNEL	UNLIMITED
DEFAULT	FAILED_LOGIN_ATTEMPTS	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_LIFE_TIME	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_REUSE_TIME	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_REUSE_MAX	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_VERIFY_FUNCTION	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_LOCK_TIME	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_GRACE_TIME	PASSWORD	UNLIMITED

Do usuwania profili służy polecenie **DROP PROFILE**, do wykonywania, którego wymagane jest posiadanie uprawnienia systemowego **DROP PROFILE**.

Składnia polecenia **DROP PROFILE:**

```
SQL> DROP PROFILE nazwa_profilu [CASCADE];
```

gdzie:

nazwa_profilu	-	Nazwa usuwanego profilu.
CASCADE	-	Pozwala usunąć profil, do którego są przypisani użytkownicy. Przed usunięciem profilu użytkownicy są przypisywani automatycznie do profilu DEFAULT.

Uwaga. Nie można usunąć profilu **DEFAULT**.

§ 8.5. Przykłady związane z zarządzaniem z poziomu profili

Przykład 8.5.1. Z użytkownika SYSTEM utwórzmy profil p1, a następnie przypiszmy użytkownika kadry do utworzonego profilu.

```
SQL> CREATE PROFILE p1 LIMIT
     PASSWORD_LIFE_TIME 10
     PASSWORD_GRACE_TIME 20
      SESSIONS_PER_USER 1
      FAILED_LOGIN_ATTEMPTS 2
      PASSWORD_LOCK_TIME 3;
SQL> ALTER USER kadry PROFILE p1;
```

Uwaga. Ustawienia dotyczące hasła obowiązują dopiero po zmianie na nowe hasło. Pozostałe ustawienia obowiązują od następnej sesji.

Z użytkownika kadry zmieniamy hasło.

```
SQL> CONNECT kadry/kadry
SQL> ALTER USER kadry IDENTIFIED BY kadry;
```

Po 3 dniach użytkownik loguje się do bazy.

```
SQL> CONNECT kadry/kadry
 Połączony.
```

Po kolejnych 8 (razem jest to 11 dni od zmiany hasła) dniach użytkownik loguje się do bazy.

```
SQL> CONNECT kadry/kadry
BŁĄD:
ORA-XXXXX: hasło wygaśnie w ciągu 20 dni
Połączony.
```

Po kolejnych 21 dniach, użytkownik kadry próbuje się połączyć z bazą i jest zmuszony do zmiany hasła.

```
SQL> CONNECT kadry/kadry
 BŁĄD:
 ORA-28001: hasło wygasło
 Zmiana hasła dla kadry
 Stare hasło: ....
```

Data wygaśnięcia hasła konta **kadry** można odczytać poleceniem:

```
SQL> SELECT username, expiry_date, profile
     FROM dba_users
     WHERE username = 'KADRY';
```

USERNAME	EXPIRY_DATE	PROFILE
U3	2004-08-13 23:43:56	P1

Spróbujmy usunąć profil p1.

```
SQL> DROP PROFILE P1;
    BŁĄD w linii 1:
```

```
ORA-02382: profil P1 ma przypisanego użytkownika, nie można go usunąć bez
CASCADE
```

Poniższym zapytaniem możemy sprawdzić, jacy użytkownicy mają przypisany profil `p1`.

```
SQL> SELECT username
     FROM dba_users
     WHERE profile = 'P1';
```

USERNAME
KADRY

```
SQL> DROP PROFILE p1 CASCADE;
```

```
Profil został usunięty.
```

A teraz sprawdźmy, do jakiego profilu jest przypisany użytkownik, który był przypisany do profilu usuniętego z opcją `CASCADE`.

```
SQL> SELECT username, profile
     FROM dba_users
     WHERE username = 'KADRY';
```

USERNAME	PROFILE
KADRY	DEFAULT

Przykład 8.5.2. Często, tworząc nowe konto dla użytkownika chcemy, wymusić zmianę hasła przy pierwszym logowaniu. Możemy to zrobić używając parametru `PASSWORD EXPIRE`, co pokazują poniższe instrukcje.

```
SQL> CREATE USER kowalski IDENTIFIED BY jan
     DEFAULT TABLESPACE users
     TEMPORARY TABLESPACE temp
     QUOTA 1M ON users
     QUOTA 1M ON system
     PASSWORD EXPIRE;
SQL> GRANT create session,create table TO kowalski;
```

Aplikacje powinny żądać zmiany hasła przy pierwszym logowaniu.

§ 8.6. Weryfikacja hasła

System ORACLE dostarcza bardzo elastyczny system weryfikacji trudności hasła. Możliwości są właściwie nieograniczone. Można np. sprawdzić czy użytkownicy nie używają w swoich hasłach danych ogólnie dostępnych jak numery telefonu itp.

Przykładowa funkcja tworzona jest przez skrypt `utlpwdmg.sql`, który znajduje się w katalogu `...\RDBMS\ADMIN`.

Przykład 8.6.1. Prosta funkcja weryfikacji hasła.

```
CREATE OR REPLACE FUNCTION verify_function
(username VARCHAR2,password VARCHAR2, old_password VARCHAR2)
RETURN BOOLEAN
IS
BEGIN
   IF LENGTH(password) < 5 THEN
        RAISE_APPLICATION_ERROR (-20001,
        'Hasło musi posiadać co najmniej 5 znaków');
   END IF;
   RETURN (TRUE);
END;
/
```

Modyfikacja profilu:

```
ALTER PROFILE default LIMIT
PASSWORD_VERIFY_FUNCTION verify_function;
```

Modyfikacja użytkownika (o ile miał inny profil):

```
ALTER USER kadry PROFILE default;
```

Spróbować zmienić hasło nie spełniające warunku z funkcji `verify_function` poleceniem

```
ALTER USER kadry IDENTIFIED BY nowe_haslo REPLACE stare_haslo;
```

Przykład 8.6.2. Przykładowa funkcja weryfikacji hasła (dla zainteresowanych).

```
CREATE OR REPLACE FUNCTION czy_haslo_trudne(
  username       IN   VARCHAR2,
  password       IN   VARCHAR2,
  old_password   IN   VARCHAR2 ) RETURN BOOLEAN
IS
  v_zglaszany_dbms_error     CONSTANT INTEGER  NOT NULL := -20997;
  v_cyfry                    CONSTANT VARCHAR2(10) NOT NULL := '0123456789';
  v_litery                   CONSTANT VARCHAR2(35) NOT NULL
    := 'qwertyuiopasdfghjklzxcvbnm';
  v_symbole                  CONSTANT VARCHAR2(31) NOT NULL := '~`!@#$%^&*()-_=+[{]};:"\|,<.>/?';
  v_min_dlugosc              CONSTANT PLS_INTEGER          := 5;
  v_wymaga_cyfr              CONSTANT BOOLEAN      NOT NULL := TRUE;
  v_wymaga_liter             CONSTANT BOOLEAN      NOT NULL := TRUE;
  v_wymaga_symboli           CONSTANT BOOLEAN      NOT NULL := TRUE;
  v_rozne_od_nazwy           CONSTANT BOOLEAN      NOT NULL := TRUE;
  v_nie_zawiera_starego      CONSTANT BOOLEAN      NOT NULL := TRUE;
  v_nie_jest_czescia_starego CONSTANT BOOLEAN      NOT NULL := TRUE;
  v_nie_wystepuje_w_slowniku CONSTANT BOOLEAN      NOT NULL := TRUE;

BEGIN
  IF LENGTH(password) < v_min_dlugosc THEN
    raise_application_error( v_zglaszany_dbms_error,
                          sło musi składać się z co najmniej ' || v_min_dlugosc || ' znaków.');
  END IF;

  IF  v_rozne_od_nazwy AND password = username THEN
    raise_application_error( v_zglaszany_dbms_error,
                'Hasło jest łatwe do odgadnięcia (nazwa użytkownika i hasło są identyczne).');
  END IF;

  IF  v_nie_zawiera_starego AND INSTR(password, old_password) != 0 THEN
    raise_application_error( v_zglaszany_dbms_error,
                                          'Nowe hasło nie może zwierać poprzedniego hasła.');
  END IF;

  IF  v_nie_jest_czescia_starego AND INSTR(old_password, password) != 0 THEN
    raise_application_error( v_zglaszany_dbms_error,
                                      'Nowe hasło nie może być wycinkiem poprzedniego hasła.');
  END IF;

  IF  v_wymaga_cyfr AND LENGTH(password)!=LENGTH(TRANSLATE(password, 'X' || v_cyfry, 'X')) THEN
    raise_application_error( v_zglaszany_dbms_error, 'Hasło musi zawierać cyfrę.');
  END IF;

  IF  v_wymaga_symboli AND LENGTH(password)!=LENGTH(TRANSLATE(password, 'X' || v_symbole, 'X') )
  THEN
    raise_application_error( v_zglaszany_dbms_error,
                                       'Hasło musi zawierać jeden z symboli: ' || v_symbole );
  END IF;

  IF  v_wymaga_liter
      AND LENGTH(NLS_LOWER(password) ) != LENGTH(TRANSLATE(password, 'X' || v_litery, 'X') )
  THEN
    raise_application_error( v_zglaszany_dbms_error, 'Hasło musi zawierać literę');
  END IF;
  IF v_nie_wystepuje_w_slowniku THEN
    DECLARE
      v_haslo VARCHAR2(30);
    BEGIN
      SELECT s.haslo
      INTO   v_haslo
      FROM   sl_hasel s
      WHERE  s.haslo = NLS_UPPER(password);
      raise_application_error(
        v_zglaszany_dbms_error,
        'Hasło jest łatwe do odgadnięcia (znajduje się w słowniku niedozwolonych haseł).'
      );
```

```
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        NULL;
    END;
  END IF;

  RETURN TRUE;
END czy_haslo_trudne;
/
```

§ 8.7. Autoryzacja przez plik haseł

Użytkownicy w Oracle standardowo autoryzowani są przez bazę danych. W pewnych sytuacjach można skorzystać z autoryzacji przez plik haseł.

Autoryzacja przez plik haseł dostępna jest dla użytkowników posiadających uprawnienia systemowe SYSOPER lub SYSDBA.

Uprawnienia te dają użytkownikowi możliwość wykonywania między innymi następujących operacji:

SYSDBA

- Wykonywanie poleceń **STARTUP** i **SHUTDOWN.**
- Wykonywanie polecenia **ALTER DATABASE** z klauzulami**: OPEN, MOUNT, BACKUP.**
- Wykonywanie polecenia **CREATE DATABASE.**
- Wykonywanie poleceń **ARCHIVELOG** oraz **RECOVERY.**
- Tworzenie binarnego pliku parametrów poleceniem **CREATE SPFILE.**
- Zawierają uprawnienie **RESTRICTED SESSION.**
- ...

SYSOPER

- Wykonywanie poleceń **STARTUP** i **SHUTDOWN.**
- Wykonywanie polecenia **ALTER DATABASE** z klauzulami**: OPEN, MOUNT, BACKUP.**
- Wykonywanie poleceń **ARCHIVELOG** oraz **RECOVERY.**
- Tworzenie binarnego pliku parametrów poleceniem **CREATE SPFILE.**
- Zawierają uprawnienie **RESTRICTED SESSION.**
- **Brak dostępu do danych użytkowników i brak uprawnień do tworzenia np. tabel, użytkowników itp.**
- ...

Plik haseł znajduje się w katalogu:

...\database\pwd<SID>.ora

Nowy plik haseł można utworzyć programem **orapwd** z systemowej linii poleceń:

```
c:\orapwd FILE=plik PASSWORD=hasło ENTRIES= liczba
```

gdzie:

plik	– Parametr powinien zawierać pełną ścieżkę do nowego pliku haseł wraz z jego nazwą.
hasło	– Hasło, które będzie obowiązywało użytkownika SYS do autoryzacji poza bazą danych.
liczba	– Ilość wpisów zarezerwowanych dla użytkowników z uprawnieniami SYSOPER oraz SYSDBA. Po osiągnięciu limitu konieczne jest wygenerowanie nowego pliku.

Uwaga. Przed utworzeniem nowego pliku haseł należy zamknąć bazę danych i zatrzymać usługę, która uruchamia bazę danych w środowisku Windows.

W celu skorzystania z autoryzacji przez plik haseł należy połączyć się jako użytkownik z klauzulą `AS SYSOPER` lub `AS SYSDBA`, np.

```
SQL> CONNECT sys AS SYSOPER;
```

Uwaga. Podczas instalacji Oracle w systemie operacyjnym Windows tworzona jest grupa użytkowników `ORA_DBA`. Użytkownicy systemu operacyjnego dopisani do tej grupy, którzy posiadają uprawnienie systemowe `SYSOPER` lub `SYSDBA` nie są proszeni o podawanie hasła podczas autoryzacji przez plik haseł.

Uwaga. Informacje, którzy użytkownicy posiadają uprawnienia `SYSDBA` lub `SYSOPER` można znaleźć w perspektywie `V_$PWFILE_USERS`.

```
SQL> SELECT * FROM v_$pwfile_users;

USERNAME                       SYSDB SYSOPER
------------------------------ ----- -------
SYS                            TRUE  TRUE
SYSTEM                         TRUE  FALSE
```

9. UPRAWNIENIA

To, jakie operacje mogą wykonywać użytkownicy jest regulowane poprzez nadane im uprawnienia. Nowo utworzony użytkownik nie posiada żadnych uprawnień, czyli nawet nie może uzyskać połączenia z bazą (utworzyć sesji).

Rodzaje uprawnień:

- Uprawnienia systemowe.
- Uprawnienia obiektowe.

§ 9.1. Przykładowe uprawnienia systemowe

Uprawnienia i role	
GRANT ANY PRIVILEGE	- Pozwala przyznawać i odbierać dowolne uprawnienie systemowe (nie istnieje analogiczne uprawnienie obiektowe, czyli użytkownik A nie może przyznać bezpośrednio uprawnień do obiektów użytkownika B, jeżeli nie posiada do nich uprawnienia obiektowego z opcją GRANT).
CREATE ROLE	**- Pozwala tworzyć rolę.**
ALTER ANY ROLE	- Pozwala zmieniać dowolną rolę.
DROP ANY ROLE	- Pozwala zmieniać dowolną rolę.
GRANT ANY ROLE	- Pozwala przyznawać dowolną rolę.
Użytkownicy	
CREATE USER	**- Pozwala tworzyć użytkowników.**
ALTER USER	- Pozwala zmieniać użytkowników. Użytkownik nieposiadający tego uprawnienia może wykonać polecenie ALTER USER, aby zmienić swoje hasło.
DROP USER	- Pozwala usuwać użytkowników.
Profile użytkowników	
CREATE PROFILE	- Pozwala tworzyć profile użytkowników.
ALTER PROFILE	- Pozwala zmieniać profile użytkowników.
DROP PROFILE	- Pozwala usuwać profile użytkowników.
Tabele, indeksy, ograniczenia	
CREATE TABLE	**- Pozwala tworzyć, usuwać, zmieniać tabele i widoki we własnym schemacie oraz związane z nimi indeksy i ograniczenia.**
CREATE ANY TABLE	**- Pozwala tworzyć tabele w dowolnym schemacie oraz związane z nimi indeksy i ograniczenia.**
ALTER ANY TABLE	- Pozwala zmieniać tabele i kompilować widoki z dowolnego schematu oraz związane z nimi indeksy i ograniczenia.
BACKUP ANY TABLE	- Pozwala użyć narzędzi eksportu dla tabel z dowolnego schematu.
DROP ANY TABLE	**- Pozwala usuwać i obcinać tabele z dowolnego schematu.**
LOCK ANY TABLE	- Pozwala blokować dowolną tabelę lub widok w bazie.
SELECT ANY TABLE	**- Pozwala zadać zapytania do dowolnej tabeli, widoku, migawki.**
INSERT ANY TABLE	**- Pozwala wstawiać wiersze do dowolnej tabeli lub widoku.**
UPDATE ANY TABLE	**- Pozwala zmieniać wiersze w dowolnej tabeli lub widoku.**
DELETE ANY TABLE	**- Pozwala usuwać dowolną tabelę lub widok.**
CREATE ANY INDEX	**- Pozwala tworzyć indeks w dowolnym schemacie dla dowolnej tabeli.**
ALTER ANY INDEX	**- Pozwala zmieniać dowolny indeks w bazie.**
DROP ANY INDEX	**- Pozwala usunąć dowolny indeks w bazie.**
WYZWALACZE	
CREATE TRIGGER	**- Pozwala tworzyć, zmieniać i usuwać wyzwalacze we własnym schemacie.**
CREATE ANY TRIGGER	- Pozwala tworzyć wyzwalacze w dowolnym schemacie.
ALTER ANY TRIGGER	- Pozwala zmieniać wyzwalacze w dowolnym schemacie (włączać, wyłączać, kompilować).
DROP ANY TRIGGER	- Pozwala usuwać wyzwalacze znajdujące się w dowolnym schemacie.
PERSPEKTYWY	
CREATE VIEW	**- Pozwala tworzyć, zmieniać i usuwać widoki we własnym schemacie.**
CREATE ANY VIEW	- Pozwala tworzyć widoki w dowolnym schemacie. Dodatkowo wymagane jest posiadanie uprawnień do obiektów wykorzystywanych przez tworzony widok.
DROP ANY VIEW	- Pozwala usuwać widoki z każdego schematu.
Migawki	
CREATE SNAPSHOT	**- Pozwala tworzyć migawki we własnym schemacie (wymagane jest również posiadanie uprawnienia CREATE TABLE).**
CREATE ANY SNAPSHOT	- Pozwala tworzyć migawki w dowolny schemacie (dodatkowo wymagane jest uprawnienie CREATE ANY TABLE).
ALTER SNAPSHOT	- Pozwala zmieniać migawki własnego schematu.
DROP ANY SNAPSHOT	- Pozwala usuwać migawki ze wszystkich schematów.

Klastry

CREATE CLUSTER	- Pozwala tworzyć, zmieniać i usuwać klaster we własnym schemacie.
CREATE ANY CLUSTER	- Pozwala tworzyć klaster w dowolnym schemacie.
ALTER ANY CLUSTER	- Pozwala zmieniać dowolny klaster w bazie.
DROP ANY CLUSTER	- Pozwala usuwać dowolny klaster w bazie.

SEKWENCJE

CREATE SEQUENCE	**- Pozwala tworzy, zmieniać i usuwać sekwencje we własnym schemacie.**
CREATE ANY SEQUENCE	- Pozwala tworzyć sekwencje w dowolnym schemacie.
ALTER ANY SEQUENCE	- Pozwala zmieniać sekwencje znajdujące się w dowolnym schemacie.
DROP ANY SEQUENCE	- Pozwala usuwać sekwencje z dowolnego schematu.
SELECT ANY SEQUENCE	**- Pozwala korzystać z sekwencji znajdującej się w dowolnym schemacie.**

PROCEDURY, FUNKCJE, PAKIETY

CREATE PROCEDURE	**- Pozwala tworzyć, zmieniać, usuwać procedury, funkcje, pakiety we własnym schemacie.**
CREATE ANY PROCEDURE	- Pozwala tworzyć procedury, funkcje i pakiety w dowolnym schemacie (Wymagane jest, aby użytkownik posiadał również uprawnienia ALTER ANY TABLE, BACKUP ANY TABLE, DROP ANY TABLE, SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE).
ALTER ANY PROCEDURE	- Pozwala kompilować dowolną procedurę, funkcję, pakiet z dowolnego schematu.
DROP ANY PROCEDURE	- Pozwala usuwać dowolną procedurę, funkcję, pakiet z dowolnego schematu.
EXECUTE ANY PROCEDURE	- Pozwala wywoływać procedurę, funkcję (samodzielną lub znajdującą się w pakiecie) oraz korzystać z publicznych zmiennych pakietów.

LINKI BAZODANOWE

CREATE DATABASE LINK	**- Pozwala usuwać i tworzyć prywatne linki do innych baz własnym schemacie.**
CREATE PUBLIC DATABASE LINK	**- Pozwala tworzyć publiczne linki do innych baz danych.**
DROP PUBLIC DATABASE LINK	- Pozwala usuwać publiczne linki do innych baz danych.

SYNONIMY

CREATE PUBLIC SYNONYM	- Pozwala tworzyć publiczne synonimy do obiektów.
DROP PUBLIC SYNONYM	- Pozwala usuwać publiczne synonimy.
CREATE SYNONYM	**- Pozwala tworzyć i usuwać synonimy prywatne we własnym schemacie.**
CREATE ANY SYNONYM	- Pozwala tworzyć synonimy prywatne w dowolnym schemacie.
DROP ANY SYNONYM	- Pozwala usuwać synonimy prywatne z dowolnego schematu.

SESJE

CREATE SESSION	- Pozwala utworzyć sesję, czyli połączyć się z bazą danych.
ALTER SESSION	- Pozwala wywoływać polecenie ALTER SESSION.
RESTRICTED SESSION	- Pozwala połączyć się z bazą, gdy została ona uruchomiona przy użyciu polecenia STARTUP RESTRICT.

BAZA DANYCH

ALTER DATABASE	- Pozwala zmieniać cechy bazy danych, dodawać pliki z poziomu bazy, bez względu na posiadane uprawnienia w systemie operacyjnym.

PRZESTRZENIE TABEL

CREATE TABLESPACE	- Pozwala tworzyć przestrzenie tabel.
ALTER TABLESPACE	**- Pozwala zmieniać przestrzenie tabel.**
MANAGE TABLESPACE	- Pozwala przełączać przestrzenie tabel w tryb ONLINE/OFLINE oraz wykonywać backup przestrzeni tabel.
DROP TABLESPACE	- Pozwala usuwać przestrzenie tabel.
UNLIMITED TABLESPACE	**- Pozwala na zapisywanie we wszystkich przestrzeniach tabel. Uprawnienie to jest nadrzędne w stosunku do „quota" nałożonych na użytkowników i może być przyznawane tylko bezpośrednio użytkownikowi (bez pośrednictwa roli).**

SEGMENTY WYCOFANIA

CREATE ROLLBACK SEGMENT	- Pozwala tworzyć segmenty wycofania.
ALTER ROLLBACK SEGMENT	- Pozwala zmieniać segmenty wycofania.
DROP ROLLBACK SEGMENT	- Pozwala usuwać segmenty wycofania.

SYSTEM

ALTER SYSTEM	**- Pozwala wywoływać polecenie ALTER SYSTEM.**

ANALIZOWANIE

ANALYZE ANY	- Pozwala analizować (wykonać polecenia ANALYZE) dowolną tabelę, indeks lub klaster.

ŚLEDZENIE

AUDIT ANY	**- Pozwala włączać i wyłączać śledzenie dowolnego obiektu.**
AUDIT SYSTEM	**- Pozwala włączać i wyłączać śledzenie poleceń i uprawnień.**

§ 9.2. Przykłady

Przykład 9.2.1. Utwórzmy użytkownika `u` z hasłem `u` i z uprawnieniami **`create session, create table`** oraz z prawem do tworzenia obiektów w przestrzeniach `users i system i aby używał przestrzeni tymczasowej temp`.

```
SQL> CREATE USER u IDENTIFIED BY u
     DEFAULT TABLESPACE users
     TEMPORARY TABLESPACE temp
     QUOTA 1m ON users
     QUOTA 1m ON system;

SQL> GRANT create session, create table TO u;
```

Jako nowo utworzony użytkownik utwórzmy tabelę, indeks a następnie usuńmy utworzone obiekty.

```
SQL> CREATE TABLE t1(k1 NUMBER);
   Tabela została utworzona.
SQL> CREATE INDEX t1_k1 ON t1(k1) TABLESPACE users;
   Indeks został utworzony.
SQL> DROP INDEX t1_k1;
   Indeks został usunięty.
SQL> DROP TABLE t1;
   Tabela została usunięta.
```

Jak widać uprawnienie `create table` pozwala nie tylko tworzyć tabele, ale również je usuwać oraz tworzyć i usuwać indeksy i ograniczenia (constraints).

Przykład 9.2.2. Utwórzmy użytkownika `u_baza` w bazie danych poleceniem

```
SQL> CREATE USER u_baza IDENTIFIED BY u_baza
     DEFAULT TABLESPACE users
     TEMPORARY TABLESPACE temp;
```

Spróbujmy teraz połączyć się z bazą jako użytkownik `u_baza`.

```
SQL> CONNECT u_baza/u_baza
   BŁĄD:
   ORA-XXXX: użytkownik u_baza nie ma uprawnienia CREATE SESSION; odmowa
   rejestracji
```

Próba nie powiodła się, ponieważ nie przyznaliśmy użytkownikowi uprawnienia systemowego **`create session`**.

Ponieważ użytkownik `u_baza` ma być użytkownikiem, który będzie właścicielem aplikacji, zatem poza uprawnieniem pozwalającym na połączenie z systemem, musi mieć uprawnienia pozwalające mu na tworzenie obiektów w jego schemacie. Przyznajmy mu uprawnienia do tworzenia tabel, perspektyw, wyzwalaczy, procedur, funkcji, sekwencji.

```
SQL> GRANT
       create session,
       create table,
       create view,
       create trigger,
       create procedure,
       create sequence
     TO u_baza;
```

Spróbujmy połączyć się jako użytkownik `u_baza`, a następnie utworzyć prostą tabelę.

```
SQL> CONNECT u_baza/u_baza
```

```
Połączony.
SQL> CREATE TABLE t1 (k1 NUMBER);
BŁĄD w linii 1:
ORA-XXXXX: brak uprawnień na przestrzeni tabel 'USERS'
```

Jak widać próba połączenia zakończyła się sukcesem, natomiast nie udało się utworzyć tabeli w domyślnej przestrzeni tabel. Użytkownikowi `u_baza` powinniśmy nadać uprawnienia do przestrzeni `users` i `baza_i` (`baza_i` przestrzeń tabel przeznaczona dla indeksów).

```
SQL> ALTER USER u_baza
     QUOTA UNLIMITED ON baza
     QUOTA UNLIMITED ON baza_i;
Użytkownik został zmieniony.
```

Teraz jako użytkownik `u_baza` możemy spróbować utworzyć tabelę przechowywaną w domyślnej przestrzeni tabel z indeksem w przestrzeni `baza_i`.

```
SQL> CREATE TABLE t1
    (
      id NUMBER(4) PRIMARY KEY USING INDEX TABLESPACE baza_i,
      k1 VARCHAR2(100)
    );
Tabela została utworzona.
```

A teraz spróbujmy utworzyć tabelę w przestrzeni tabel `SYSTEM`.

```
SQL> CREATE TABLE t2(k1 VARCHAR2(100)) TABLESPACE system;
BŁĄD w linii 1:
ORA-XXXXX: brak uprawnień na przestrzeni tabel 'SYSTEM'
```

Jak widać dopóki nie przyznamy uprawnienia do przestrzeni tabel, użytkownik nie może tworzyć w niej obiektów.

§ 9.3. Uprawnienia obiektowe

Użytkownik nieposiadający uprawnień systemowych typu **ANY** może wykonywać operacje na obiekcie innego schematu tylko wtedy, gdy posiada do tego obiektu odpowiednie uprawnienie obiektowe. To, jakie uprawnienia można przyznać do obiektu, jest uzależnione od typu obiektu.

Typ obiektu / Uprawnienie obiektowe	TABELA	PERSPEKTYWA	SEKWENCJA	PROCEDURA, FUNKCJA, PAKIET
SELECT	•	•	•	
INSERT	•	•		
UPDATE	•	•		
DELETE	•	•		
ALTER	•		•	
EXECUTE				•
INDEX	•			
REFERENCES	•			
...				

Rysunek 9.3.1. Uprawnienia obiektowe

Uwaga. Uprawnienie `INDEX`, `REFERENCES` nie mogą być przyznane roli.

Uwaga. Użytkownik z przyznanym uprawnieniem `INDEX` nie może utworzyć indeksu na atrybutach już zaindeksowanych.

Skrót ALL (lub ALL PRIVILEGES) zastępuje wszystkie dostępne dla danego obiektu uprawnienia obiektowe. ALL nie jest uprawnieniem, a jedynie mechanizmem pozwalającym przydzielać wszystkie dozwolone uprawnienia obiektowe.

Przykład 9.3.2. Utwórzmy dwóch użytkowników u1, u2 i nadajmy im uprawnienia systemowe CREATE SESSION i CREATE TABLE oraz uprawnienia do tworzenia obiektów w przestrzeni tabel users.

```
SQL> CREATE USER u1 IDENTIFIED BY u1
     DEFAULT TABLESPACE users
     TEMPORARY TABLESPACE temp
     QUOTA 1M ON users

SQL> GRANT create session, create table TO u1;

SQL> CREATE USER u2 IDENTIFIED BY u2
     DEFAULT TABLESPACE users
     TEMPORARY TABLESPACE temp
     QUOTA 1M ON users

SQL> GRANT connect, create table TO u2;
```

Jako użytkownik u1 utwórzmy tabelę t1 wstawmy do niej wiersz oraz przyznajmy do niej uprawnienie obiektowe REFERENCES użytkownikowi u2.

```
SQL> CREATE TABLE t1
     (
      id NUMBER CONSTRAINT t1_pk PRIMARY KEY
                                  USING INDEX TABLESPACE users,
      k1 VARCHAR2(100)
      );

SQL> INSERT INTO t1 (id, k1) VALUES (1, 'a');

SQL> GRANT REFERENCES ON t1 TO u2;
```

Jako użytkownik u2 utwórzmy tabelę t2 związaną z tabelą u1.t1 obowiązkowym kluczem obcym z opcją usuwania kaskadowego, a następnie wstawmy do niej wiersz.

```
SQL> CREATE TABLE t2
     (
       id NUMBER CONSTRAINT t2_pk  PRIMARY KEY
                                    USING INDEX TABLESPACE users,
       k1 VARCHAR2(100),
       u1_t1_id NUMBER CONSTRAINT t2_u1_t1_fk REFERENCES u1.t1(id)
                                                ON DELETE CASCADE
      );

SQL> INSERT INTO t2 (id, k1, u1_t1_id) VALUES (1, 'b', 1);

SQL> COMMIT;
```

Jak widać udało się wstawić wiersz podrzędny, mimo że użytkownik u2 nie posiada uprawnień do wybierania z tabeli u1.t1.

Próba wykonania polecenia `SELECT` z tabeli `u1.t1` powoduje zgłoszenie wyjątku `ORA-.....`: niewystarczające uprawnienia. Gdyby użytkownik `u2` nie posiadał żadnych uprawnień do tabeli `u1.t1`, próba pobrania z niej wierszy zakończyłaby się błędem `ORA-.....`: tabela lub perspektywa nie istnieje.

Spróbujmy teraz jako użytkownik `u1` usunąć wiersz z tabeli `t1`, dla którego istnieje wiersz podrzędny w tabeli `u2.t2`.

```
SQL> DELETE FROM t1 WHERE id = 1;
    1 wiersz został usunięty.
SQL> COMMIT;
```

Jak widzimy został usunięty wiersz z tabeli `u1.t1` oraz kaskadowo wiersz z tabeli `u2.t2`, do której usuwający użytkownik nie ma żadnych uprawnień, a próba wykonania przez niego polecenia `SELECT` dla tej tabeli kończy się błędem ORA-……: tabela lub perspektywa nie istnieje.

10. ROLE

Mechanizm ról pozwala w łatwy i kontrolowany sposób zarządzać uprawnieniami systemowymi i objektowymi. **Role są nazwanymi grupami uprawnień**.

Roli można przyznać:

- inne role,
- dowolne uprawnienie systemowe poza `UNLIMITED TABLESPACE`,
- uprawnienie do obiektów poza `INDEX`, `REFERENCES`.

Rolę można przyznać użytkownikowi lub innej roli.

§ 10.1. Tworzenie roli

Do tworzenia roli służy polecenie `CREATE ROLE`. Aby utworzyć rolę trzeba posiadać uprawnienie systemowe `CREATE ROLE`.

```
SQL> CREATE ROLE nazwa_roli;
```

```
SQL> CREATE ROLE test;
```

Podczas tworzenia roli system nadaje ją z opcją `ADMIN` tworzącemu.

Do usuwania ról służy polecenie `DROP ROLE`. Użytkownik usuwający rolę musi posiadać ją z opcją `ADMIN OPTION` lub posiadać uprawnienie systemowe `DROP ANY ROLE`.

```
SQL> DROP ROLE nazwa_roli;
```

Uwaga. W momencie usuwania roli system automatycznie odbiera ją wszystkim użytkownikom.

Podczas instalacji systemu tworzone są specjalne przywileje systemowe:

- `SYSOPER`,
- `SYSDBA`.

Są one potrzebne do wykonywania operacji na bazie, która nie jest zamontowana, czyli wtedy, kiedy słownik danych nie jest dostępny. Przywileje te są używane przy połączeniu typu `AS SYSDBA` i `AS SYSOPER`.

Przywilej `SYSOPER` uprawnia do wykonywania następujących poleceń na bazie danych:

- `STARTUP`,
- `SHUTDOWN`,
- `ALTER DATABASE OPEN/MOUNT`,
- `ALTER DATABASE BACKUP CONTROLFILE`,
- `ALTER TABLESPACE BEGIN/END BACKUP`,
- `ARCHIVE LOG`,
- `RECOVER`,
- ...

Przywilej `SYSDBA` zawiera przywilej `SYSOPER` z opcją `ADMIN`. **Jest to przywilej wykorzystywany podczas tworzenia bazy danych.**

Przywileje `SYSDBA` i `SYSOPER` posiadają przywilej systemowy `RESTRICTED SESSION`.

Podczas tworzenia bazy danych tworzone są między innymi role:

- `CONNECT`,
- `RESOURCE`,

- DBA.

Role te są rolami definiowanym dla zachowania zgodności z wcześniejszymi wersjami i mogą być dowolnie modyfikowane.

Rola CONNECT ma następujące uprawnienia systemowe:

- CREATE SESSION,
- ...,

Rola RESOURCE posiada następujące uprawnienia systemowe

- CREATE CLUSTER,
- CREATE PROCEDURE,
- CREATE SEQUENCE,
- CREATE TABLE,
- CREATE VIEW,
- CREATE TRIGGER.

Rola DBA obejmuje wszystkie uprawnienia systemowe.

UWAGA! Podczas nadawania użytkownikowi roli RESOURCE lub DBA przyznawane jest również uprawnienie UNLIMITED TABLESPACE.

Przy tworzeniu bazy tworzone są role:

- **EXP_FULL_DATABASE,**
- **IMP_FULL_DATABASE.**

Są to role pozwalające na wykonywanie pełnego importu i eksportu bazy danych narzędziami Imp, Exp.

§ 10.2. Przyznawanie uprawnień systemowych i ról

Użytkownik przyznający rolę musi posiadać przyznawaną rolę z opcją ADMIN lub posiadać uprawnienie systemowe GRANT ANY ROLE. **Roli nie można przyznać jej samej**.

Użytkownik przyznający uprawnienie systemowe musi posiadać je z opcją ADMIN lub mieć przyznane uprawnienie systemowe GRANT ANY PRIVILEGE.

```
SQL> GRANT
        {system_priv | role}     [, { system_priv | role}]
     TO
        {user | role* | PUBLIC}   [, { user | role* | PUBLIC}]
     [WITH ADMIN OPTION];
```

gdzie:

system_priv	-	Przyznawane uprawnienie systemowe.
role	-	Przyznawana rola.
user	-	Nazwa użytkownika, któremu przyznawane są uprawnienia.
role*	-	Nazwa roli, której przyznawane są uprawnienia.
PUBLIC	-	Powoduje, że uprawnienia zostają przyznane wszystkim użytkownikom.
WITH ADMIN OPTION	-	Przyznaje uprawnienia z opcją administracyjną, która pozwala przyznać lub odebrać otrzymane uprawnienie innym użytkownikom lub rolom. Rola przyznana z tą opcją może zostać zmieniona lub usunięta przez otrzymującego uprawnienie. Aby odebrać opcję ADMIN należy odebrać uprawnienie lub rolę, a następnie przyznać ją bez opcji ADMIN.

Uwaga. Użytkownik, który posiada uprawnienie do obiektu przyznane pośrednio poprzez role, nie może z niego korzystać w swoich obiektach składowanych w bazie, takich jak perspektywy, procedury, funkcje, wyzwalacze. Aby tworzyć obiekty wykorzystujące obiekty innych użytkowników, wymagane jest posiadanie uprawnień nadanych bezpośrednio przez właściciela wykorzystywanych obiektów. Zatem jeżeli użytkownik **a** chce odwoływać się we własnym obiekcie do obiektu **b.o** użytkownika **b**, to uprawnienie do obiektu **b.o musi zostać przyznane użytkownikowi a bez pośrednictwa roli.**

Uwaga. Jeżeli użytkownik otrzymał uprawnienie systemowe wraz z opcją administracyjną i przekazał je innym użytkownikom, to odebranie mu tego prawa nie spowoduje odebrania go użytkownikom, którzy je od niego otrzymali.

UPRAWNIENIA SYSTEMOWE NIE SĄ ODBIERANE KASKADOWO !

§ 10.3. Przyznawanie uprawnień obiektowych

Aby przyznawać uprawnienie do obiektu innemu użytkownikowi lub roli trzeba być jego właścicielem lub posiadać uprawnienia do niego z opcją **GRANT**.

```
SQL> GRANT {object_priv | All [PRIVILEGES]}
     [( column [, column] ...)]
     [, {object_priv | All [PRIVILEGES]}
     [ ( column [, column] ...) ] ]
     ON [schema.]object
     TO {user | role* | Public}
     [WITH GRANT OPTION];
```

gdzie:

object_priv	-	Przyznawane uprawnienie obiektowe.
ALL [PRIVILEGES]	-	Skrót zastępujący wszystkie uprawnienia jakie może przyznać użytkownik nadający uprawnienie. Nie jest to nazwa uprawnienia.
column	-	W przypadku przyznawania uprawnienia REFERENCES, UPDATE do tabeli lub widoku możliwe jest wyspecyfikowanie kolumn, do których ma być ograniczone przyznane uprawnienie. **Nie mają zastosowania przy SELECT**.
[schema.]object	-	Nazwa obiektu poprzedzona opcjonalnie nazwą schematu, do którego następuje przyznanie uprawnień. Nazwę obiektu można zastąpić synonimem do niego. Przyznanie uprawnienia do synonimu jest równoznaczne z przyznaniem uprawnienia do obiektu, do którego odnosi się synonim i odwrotnie.
user	-	Nazwa użytkownika, któremu przyznawane są uprawnienia.
role*	-	Nazwa roli, której przyznawane są uprawnienia.
PUBLIC	-	Powoduje, że uprawnienia zostają przyznane wszystkim użytkownikom.
WITH GRANT OPTION	-	Pozwala przyznać uprawnienia do obiektów, z prawem przyznania ich dalej. Opcji tej nie można stosować przyznając prawa roli.

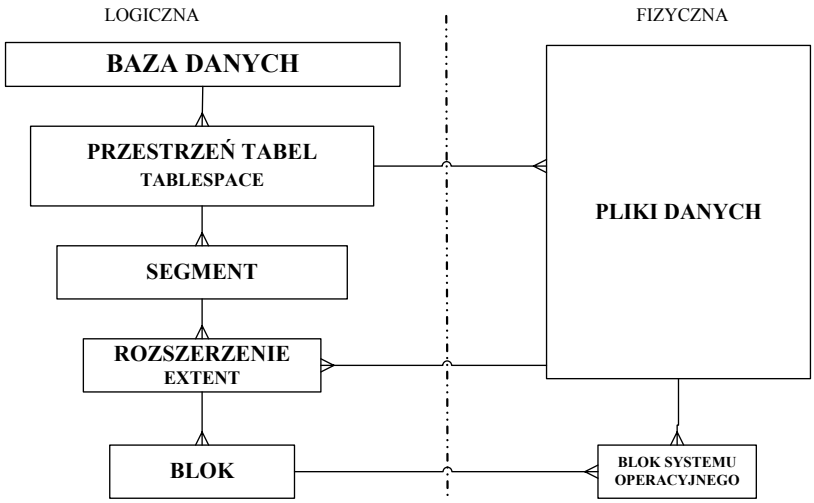
Uwaga. Jeżeli użytkownik otrzymał uprawnienie do obiektu wraz z opcją administracyjną i przekazał je innym użytkownikom, to odebranie mu tego prawa spowoduje odebranie go użytkownikom, którzy je od niego otrzymali.

UPRAWNIENIA DO OBIEKTÓW SĄ ODBIERANE KASKADOWO !

11. STRUKTURA PRZECHOWYWANIA

§ 11.1. Warstwa fizyczna i logiczna przechowywania danych

Strukturę przechowywania bazy danych ORACLE można podzielić na fizyczną i logiczną.

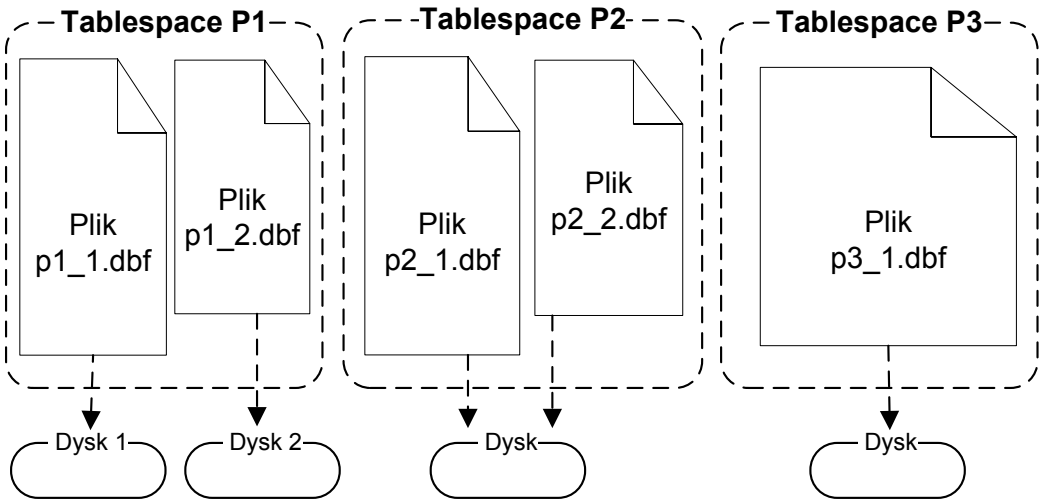


Rysunek 11.1.1. Warstwa logiczna i fizyczna danych.

Rozdzielenie warstw fizycznej i logicznej uelastycznia zarządzanie systemem.

Fizycznie baza składa się z plików danych.

Logicznie baza składa się z przestrzeni tabel (`tablespace`). Dla danej przestrzeni tabel musi istnieć, co najmniej jeden plik danych.

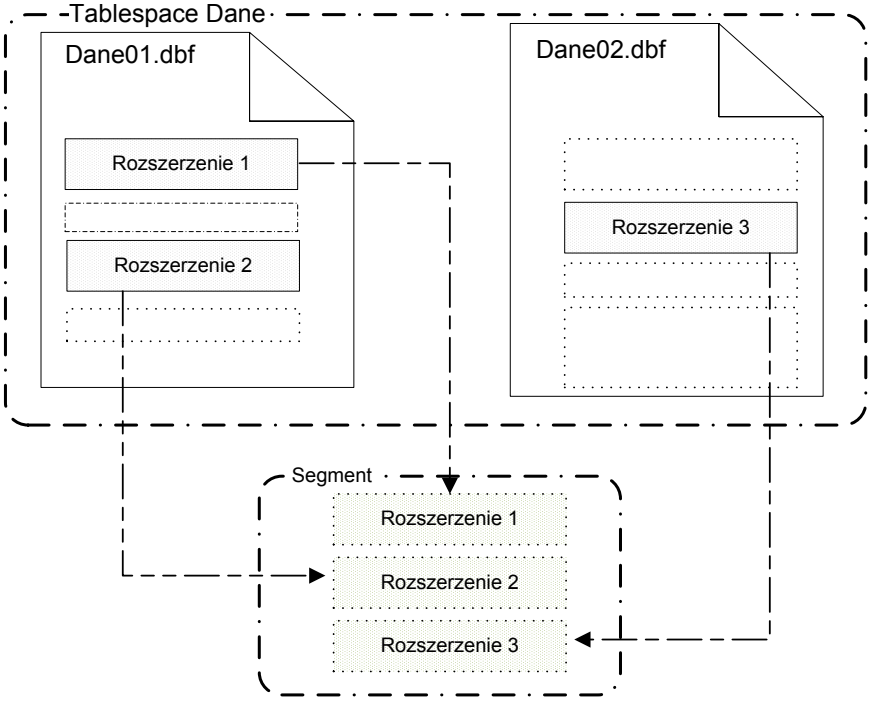


Rysunek 11.1.2. Przestrzenie danych.

Każdy plik danych jest logicznie podzielony na mniejsze jednostki. Najmniejszą jednostką logiczną pliku danych jest **blok**.

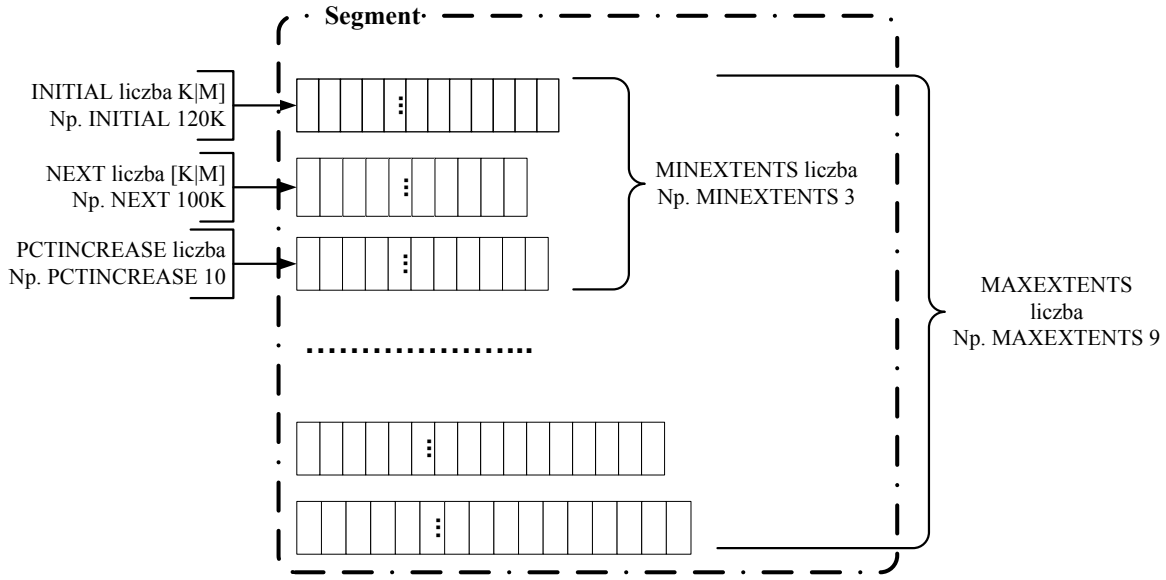
Z **bloków** stanowiących ciągły obszar zbudowane są **rozszerzenia** (`extent`).

Segment to zbiór **rozszerzeń**.



Rysunek 11.1.3. Rozszerzenia i segment.

Następny rysunek ilustruje budowę segmentu (parametry będą omówione później).



Rysunek 11.1.4. Budowa segmentu.

Blok jest najmniejszą jednostką alokacji przestrzeni dyskowej dla bazy Oracle, składający się z jednego lub wielu bloków systemu operacyjnego.

Rozmiar bloku jest ustawiany podczas tworzenia bazy przez parametr `DB_BLOCK_SIZE`.

Blok można podzielić na:

- Nagłówek.
- Wolną przestrzeń.
- Przestrzeń danych.

Rysunek 11.1.5. Budowa bloku

Nagłówek bloku (ang. block header) przechowuje między innymi:

- informacje o typie segmentu, w skład, którego wchodzi,
- dane o transakcjach, w których blok jest używany.
- …

Wolna przestrzeń bloku znajduje się pomiędzy nagłówkiem i przestrzenią danych. Konstrukcja taka pozwala **na rozrastanie się nagłówka, jak i przestrzeni danych**. Początkowo wolna przestrzeń bloku jest spójnym obszarem jednak operacje wstawiania i aktualizacji mogą być przyczyną fragmentacji wolnej przestrzeni bloku.

Defragmentacja wolnej przestrzeni w bloku jest przeprowadzana przez system automatycznie.

Klauzula przechowywania (ang. `storage clause`) może być określona na poziomie:

- **przestrzeni tabel,**
- **lub segmentu (tabeli).**

Jeżeli parametry przechowywania nie są jawnie ustawione na poziomie segmentu, to stosowane są parametry jego przestrzeni tabel (poza **`MINIMUM EXTENT`**).

Jeżeli przestrzeń tabel nie ma jawnie określonych parametrów przechowywania przyjmowane są wartości domyślne systemu.

Po zmianie parametru przechowywania, nowa wartość jest stosowana jedynie dla nowo przydzielanych zakresów (extents).

Nowe zakresy są przydzielane, gdy segment jest:

- tworzony,
- rozszerzany,
- zmieniany.

Zakresy są zwalniane, gdy segment jest:

- usuwany,
- zmieniany,
- …,
- automatycznie zmniejszany (tylko segment wycofania).

Po utworzeniu przestrzeni tabel plik danych składa się z bloku nagłówka, który jest pierwszym blokiem pliku oraz z wolnego zakresu będącego pozostałą częścią pliku danych. Podczas tworzenia segmentów, przydzielana jest im przestrzeń należąca do wolnych zakresów.

Zakresy zwalniane przez segmenty są dodawane do puli wolnych zakresów w przestrzeni tabel.

Spójny fragment przestrzeni wykorzystywany przez segment nazywany jest wykorzystanym zakresem (ang. `used extent`).

Częste przydzielanie i zwalnianie zakresów może prowadzić do fragmentacji przestrzeni w pliku danych.

W przestrzeni tabel może powstać ciągły obszar wolnej przestrzeni złożonej z kilku zakresów, który można złączyć (ang. `coalesed`) w jeden zakres. Takie złączenie w jeden zakres następuje, gdy:

- `SMON` zainicjuje transakcję przestrzeni (ang. `space transaction`),
- system próbuje przydzielić zakres, który potrzebuje więcej przestrzeni niż przestrzeń jednego z sąsiednich zakresów,
- wydane zostanie polecenie

```
SQL> ALTER TABLESPACE nazwa_przestrzeni_tabel COALESCE;
```

`Proces SMON` łączy zakresy w przestrzeniach tabel, w których `PCTINCREASE > 0`, zatem, aby złączenia następowały automatycznie w przestrzeniach zawierających obiekty użytkowników należy w domyślnej klauzuli przechowywania przestrzeni tabel ustawić `PCTINCREASE = 1`.

Przykład 11.1.2. Poniższe zapytanie pokazuje jak sprawdzić, które przestrzenie zawierają zakresy mogące podlegać złączeniu.

```
SQL> SELECT tablespace_name, total_extents,
           percent_extents_coalesced
     FROM dba_free_space_coalesced
     WHERE percent_extents_coalesced != 100;
```

TABLESPACE_NAME	TOTAL_EXTENTS	PERCENT_EXTENTS_COALESCED
USERS	7	42,857142857

Jak widać możemy złączyć zakresy w przestrzeni **USERS**.

```
SQL> ALTER TABLESPACE users COALESCE;
```

Różne typy segmentów mają różną skłonność do fragmentacji, dlatego zalecane jest umieszczanie ich w osobnych przestrzeniach tabel.

§ 11.2. Segmenty danych

Segmenty to obiekty bazy danych, które zajmują jej przestrzeń. Tabele są podstawowymi obiektami relacyjnych baz danych. **Dane tabel są przechowywane w segmentach danych**.

Dane wierszy są przechowywane w bazie w rekordach zmiennej długości. Kolumny wiersza są zazwyczaj przechowywane w takim porządku, w jakim występują kolumny w tabeli.

Każdy wiersz tabeli posiada:

- Nagłówek wiersza – wykorzystywany do przechowywania liczby kolumn wiersza, oraz informacji o łańcuchu i statusie blokady wiersza.
- Dane wiersza – zawierające: długości kolumn, wartości kolumn.

Dane znakowe mogą być przechowywane w bazie jako łańcuchy stałej lub zmiennej długości. Typy znakowe stałej długości (**CHAR**, **NCHAR**) są przechowywane w zadeklarowanej długości.

Typy znakowe zmiennej długości (**VARCHAR2**, **NVARCHAR2**) wykorzystują tylko przestrzeń potrzebną do zapisania bieżącej wartości kolumny. Wiersze tej samej kolumny mogą różnić się długością.

Liczby w bazie danych ORACLE są zawsze (bez względu na definicję) przechowywane jako dane zmiennej długości. Mogą one mieć 38 cyfr znaczących.

Typ danych **DATE** jest siedmiobajtowym typem stałej długości przechowującym datę i czas.

Baza ORACLE udostępnia następujące typy danych do przechowywania wielkich obiektów (**LOB** - Large Object):

- **CLOB** – przeznaczony do przechowywania dużych ilości danych z zestawem znaków stałej szerokości,
- **NCLOB** – przeznaczony do przechowywania dużej ilości danych znakowych stałej szerokości,
- **BLOB** – przeznaczony do przechowywania danych bez struktury,
- **BFILE** – przeznaczony do przechowywania danych w plikach systemu operacyjnego.

W bazie ORACLE istnieje specjalny typ danych dla identyfikatorów wierszy – **ROWID.**

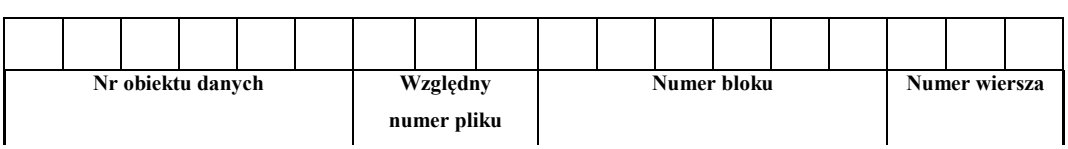
ROWID jest pseudokolumną typu **ROWID**, posiadającą następujące właściwości:

- Jest unikalnym identyfikatorem każdego wiersza w całej bazie danych.
- Nie jest jawnie przechowywana jak inne wartości kolumn.
- Zapewnia najszybszą metodę dostępu do wiersza.
- Jest wykorzystywana w indeksach do wyszukiwania wiersza z określoną wartością klucza.

`Przykładowy ROWID` mógłby składać się z następujących części (w zależności od wersji Oracle):

- **Numeru obiektu danych** (ang. Data Object Number), który jest unikalnym w całej bazie numerem przypisanym do każdego obiektu danych,
- **Względnego numeru pliku** (ang. Relative File Number), który jest unikalnym numerem pliku w ramach przestrzeni tabel,
- **Numeru bloku** (ang. Block Number), który reprezentuje pozycję bloku zawierającego dany wiersz w pliku danych,

- **Numeru wiersza** (ang. Row Number), który identyfikuje numer pozycji wiersza w katalogu wierszy nagłówka bloku.



Rysunek 11.2.1. Budowa ROWID

Przykład 11.2.2. Poniższe zapytanie wybiera z tabeli **kadry.osoby** kolumny **ROWID** i **id**.

```
SQL> SELECT ROWID, id FROM kadry.osoby;
```

ROWID	ID
AAABo6AAD**AAAAL7**AAA	1
AAABo6AADAAAAL7AAB	2

Jak widzimy dla wybranych wierszy

- numer obiektu danych to AAABo6,
- względny numer pliku to AAD
- numer bloku to AAAAL7

Natomiast wiersz o `id = 1` ma numer wiersza w katalogu wierszy równy AAA.
Ścieżka lokalizacji wiersza przez **ROWID**:

- Na podstawie numeru obiektu danych system może określić przestrzeń tabel, ponieważ segment może być umieszczony tylko w jednej przestrzeni tabel.
- Znając przestrzeń tabel i względny numer pliku danych w przestrzeni tabel można określić plik danych.
- Dalej, znając plik danych i numer bloku, można odszukać blok zawierający wiersz.
- Po zlokalizowaniu bloku, znając numer wiersza w katalogu wierszy umieszczonym w nagłówku bloku, można odczytać pozycję wiersza w bloku.

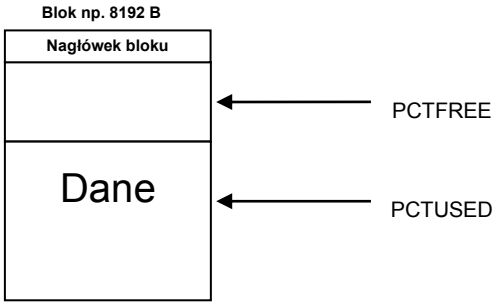
Parametry składowania **INITRANS** i **MAXTRANS** oznaczają odpowiednio początkową i maksymalną liczbę obszarów transakcji tworzonych w bloku. Obszary transakcji przechowują informacje o wszystkich transakcjach dokonujących w danym momencie zmian w bloku (**może to powodować zwiększenie rozmiaru nagłówka bloku**).

Dla każdej transakcji przeznaczony jest tylko jeden obszar, bez względu na to ile wierszy modyfikuje.

Wartość domyślna parametru **INITRANS** dla segmentów danych wynosi 1.

Parametr **MAXTRANS** ustala limit współbieżnych transakcji dokonujących zmian w bloku. Wartość domyślna parametru **MAXTRANS** wynosi 255. Niska wartość **MAXTRANS** ogranicza wykorzystywanie wolnej przestrzeni na obszary transakcji i w ten sposób zwiększa przestrzeń przeznaczoną dla danych.

Parametr **PCTFREE** określa procentowo (w stosunku do rozmiaru bloku pomniejszonego o nagłówek) wielkość obszaru zarezerwowanego dla aktualizacji wierszy już znajdujących się w bloku. **Jeżeli wartość parametru PCTFREE nie jest ustawiona przyjmowana jest wartość domyślna.**



Rysunek 11.2.2. Parametry PCTFREE i PCTUSED

Wstawianie rekordów do bloku może następować dopóty, dopóki parametr `PCTFREE` nie zostanie przekroczony.

Niski współczynnik `PCTFREE`:

- Powoduje rezerwowanie małej ilości miejsca na aktualizację wierszy.
- Zapewnia lepsze wypełnienie bloku.
- Może powodować wysoki koszt przetwarzania (np. przy operacji `UPDATE` może zabraknąć miejsca na nowe wartości).

Wysoki współczynnik `PCTFREE`:

- Powoduje rezerwowanie dużej przestrzeni dla aktualizacji wierszy.
- Zmniejsza wykorzystanie przestrzeni.
- Może zmniejszyć koszt przetwarzania (np. przy operacji `UPDATE`) .

Dużą wartość `PCTFREE` należy ustawiać, jeżeli tabela zawiera wiersze, które w wyniku aktualizacji będą zwiększały rozmiar.

Małą wartość `PCTFREE` należy ustawiać, jeżeli tabela zawiera wiersze, które nie będą zmieniane.

Parametr **`PCTUSED`** jest podawany procentowo i określa minimalny procent wykorzystanej przestrzeni, jaki system stara się zachować w każdym bloku. Blok jest ponownie umieszczany na liście bloków wolnych, kiedy wykorzystana w nim przestrzeń spadnie poniżej **`PCTUSED`**, czyli wtedy, gdy istnieje wystarczająca przestrzeń na wstawienie średniej wielkości wiersza. Lista bloków wolnych w segmencie, to lista bloków gotowych do przyjęcia kolejnych wierszy. Jeżeli wartość parametru **`PCTUSED`** nie jest ustawiona przyjmowana jest wartość domyślna.

Przy wstawieniu wiersza system przegląda kolejne bloki z listy wolnych i szuka bloku z wystarczającą przestrzenią. Poprawne ustawienie **`PCTUSED`** powoduje, że system szybko znajduje blok mający wystarczająco dużo wolnej przestrzeni dla wstawianego wiersza.

Niski współczynnik `PCTUSED`

- Powoduje gorsze wykorzystanie przestrzeni.
- Zmniejsza koszt operacji `UPDATE` i `DELETE`, przez rzadkie przenoszenie bloku na listę wolnych.
- Zmniejsza koszt operacji `INSERT` bo szybciej znajduje bloki dostępne do wstawiania.

Wysoki współczynnik `PCTUSED`

- Powoduje leprze wykorzystanie przestrzeni.
- Zwiększa koszt np. operacji `INSERT (wzrasta czas znalezienia bloku, do którego może być wstawiany wiersz).`

Jeżeli wiersz jest tak duży, że nie mieści się w jednym bloku, to system dzieli taki wiersz na mniejsze części zwane elementami wiersza. Każdy element wiersza jest przechowywany w osobnym bloku wraz ze wskaźnikiem do następnej części wiersza. **Wiersz taki nazywany jest wierszem w łańcuchu.**

§ 11.3. Segmenty wycofania

Segment wycofania (**rollback segment**) przechowuje dane sprzed ich zmodyfikowaniem przez niezatwierdzoną transakcję. **Każda baza danych musi posiadać przynajmniej jeden segment wycofania.** Segment ten (o nazwie **`SYSTEM`**) jest tzw. systemowym segmentem wycofania.

Segment ten jest wykorzystywany m.in. przez transakcje zapisujące informacje do słownika bazy danych. Oracle zaleca, aby każda baza danych posiadała dodatkowe – niesystemowe segmenty wycofania. Każdy taki segment jest tworzony w określonej przestrzeni tabel, podanej w poleceniu tworzącym segment wycofania.

Zadania segmentów wycofania:

- **Wycofywanie transakcji** – Kiedy transakcja dokonuje modyfikacji wiersza tabeli poprzednie wartości tego wiersza są przechowywane w segmencie wycofania, aż do zakończenia transakcji. W przypadku wycofywania transakcji następuje przywrócenie wartości wiersza na podstawie danych zapisanych w segmencie wycofania.
- **Zapewnienie spójności odczytu** – Transakcje nie widzą zmian dokonanych przez transakcje, które nie zostały zatwierdzone, a polecenia nie widzą nawet zmian zatwierdzonych już po ich rozpoczęciu. Potrzebny jest często obraz danych sprzed pewnego czasu. Informacji do stworzenia takiego spójnego obrazu dostarczają segmenty wycofania.
- **Odtwarzanie transakcji** – Po awarii systemu z rozpoczętymi transakcjami przy ponownym otwieraniu bazy serwer musi wycofać niezatwierdzone transakcje.

Przykład 11.3.1. Listę istniejących segmentów wycofywania można otrzymać poleceniem:

```
SQL> SELECT segment_name,owner,tablespace_name
     FROM dba_rollback_segs;

SEGMENT_NAME                   OWNER  TABLESPACE_NAME
------------------------------ ------ ------------------------------
SYSTEM                         SYS    SYSTEM
_SYSSMU1$                      PUBLIC UNDO
_SYSSMU2$                      PUBLIC UNDO
_SYSSMU3$                      PUBLIC UNDO
_SYSSMU4$                      PUBLIC UNDO
_SYSSMU5$                      PUBLIC UNDO
_SYSSMU6$                      PUBLIC UNDO
_SYSSMU7$                      PUBLIC UNDO
```

```
_SYSSMU8$                      PUBLIC UNDO
_SYSSMU9$                      PUBLIC UNDO
_SYSSMU10$                     PUBLIC UNDO
```

Wyróżnia się następujące typy segmentów wycofania:

- Systemowy (**SYSTEM**) – tworzony podczas tworzenia bazy danych w przestrzeni tabel **SYSTEM**, może być wykorzystywany tylko przy zmianach obiektów z tej przestrzeni tabel.
- Niesystemowe prywatne – wykorzystywane tylko przez jedną instancję.
- . . .

Każda transakcja musi mieć przydzielony segment wycofania. Jeżeli nie wystąpi jawne żądanie określonego segmentu

```
SQL> SET TRANSACTION USE ROLLBACK SEGMENT nazwa_segmentu
     NAME nazwa_transakcji;
```

serwer przydzieli segment z najmniejszą liczbą transakcji.

Segmenty wycofania tworzy się poleceniem **CREATE ROLLBACK SEGMENT**. Do wykonywania tego polecenia wymagane jest posiadanie uprawnienia systemowego o tej samej nazwie.

Segmenty wycofania są tworzone przy pomocy polecenia o następującej składni:

```
SQL> CREATE [PUBLIC] ROLLBACK SEGMENT nazwa_segmentu
     TABLESPACE nazwa_przestrzeni
     STORAGE
             (
              [INITIAL rozmiar [K|M]]
              [NEXT rozmiar [K|M]]
              [MINEXTENTS liczba]
              [MAXEXTENTS liczba]
              [OPTIMAL rozmiar [K|M]| NULL]
             );
```

gdzie:

OPTIMAL	-	Pozwala wyspecyfikować optymalny rozmiar segmentu wycofania. System automatycznie dba o zmniejszanie segmentu do podanej wielkości.
OPTIMAL NULL	-	Opcja ta powoduje, że system nie będzie automatycznie zmniejszał rozmiaru segmentu.

Przykład 11.3.2. Przykładowo poniższe polecenie tworzy segment wycofania o nazwie **rb01** w przestrzeni tabel **users**. Rozmiar pierwszego rozszerzenia wynosi 256KB. Rozmiar drugiego i każdego następnego rozszerzenia również wynosi 256KB. Minimalna liczba rozszerzeń została określona jako 2, a maksymalna na 20. Oznacza to, że maksymalny rozmiar segmentu wycofania może osiągnąć wartość 20*256KB=5120KB.

```
SQL> CREATE ROLLBACK SEGMENT rb01
     TABLESPACE users
     STORAGE
      (
         INITIAL 256K
         NEXT 256K
```

```
    MINEXTENTS 2
    MAXEXTENTS 20
    OPTIMAL 512K
);
```

Po utworzeniu segment wycofania jest nieaktywny, tj. posiada status `offline`. W celu jego uaktywnienia (włączenia) stosuje się polecenie:

```
SQL> ALTER ROLLBACK SEGMENT nazwa_segmentu ONLINE;
```

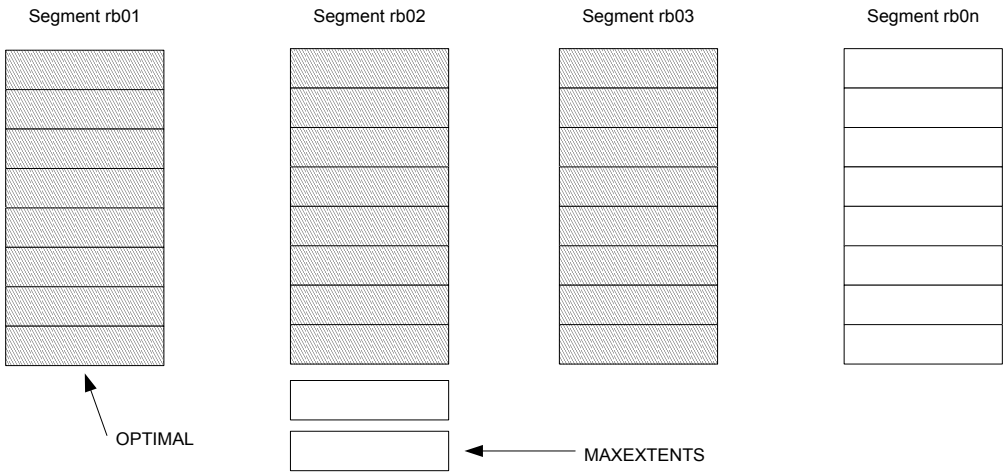
Do wyłączenia segmentu wycofania służy polecenie:

```
SQL> ALTER ROLLBACK SEGMENT nazwa_segmentu OFFLINE;
```

Opis utworzonych segmentów wycofania można uzyskać wydając zapytania do perspektyw systemowych:

`DBA_SEGMENTS` i `DBA_ROLLBACK_SEGS`.

Transakcje wykorzystują zakresy segmentu wycofania w sposób cykliczny. W momencie zapełnienia aktualnego zakresu wycofania wskaźnik zapisów jest przesuwany na następny zakres.



Rysunek 11.3.1. Rollback segment

Kiedy ostatni dostępny zakres jest zapełniony, wskaźnik może zostać przeniesiony na początek pierwszego zakresu tylko wtedy, gdy jest on wolny. Wskaźnik nie może przeskoczyć zajętego zakresu i przenieść się na pierwszy wolny. Jeżeli po wypełnieniu aktualnego zakresu następny jest zajęty, to następuje rozszerzenie segmentu wycofania, przez dodanie dodatkowego zakresu. Segment wycofania może tak rosnąć aż do uzyskania maksymalnej liczby zakresów, wyspecyfikowanej w parametrze `MAXEXTENTS`.

Dla segmentów wycofania można podać parametr przechowywania `OPTIMAL`, który określa w bajtach rozmiar, do którego segment powinien powrócić, gdy będzie to możliwe.

W środowisku bazy transakcyjnej, w którym zwykle mają miejsce krótkie transakcje, zalecanych jest istnienie wielu małych segmentów wycofania (np. jeden segment wycofania na 4 współbieżne transakcje). Duża liczba segmentów wycofania zapobiega rywalizacji o nagłówek segmentu wycofania.

W przypadku środowiska, w który wykonuje się głównie przetwarzania wsadowe dokonujące wielu zmian przy małej liczbie współbieżnych transakcji, zalecane jest utworzenie małej liczby dużych

segmentów wycofania, które powinny być umieszczone w dużych przestrzeniach tabel, aby umożliwić im rozrastanie się.

Rozmiar segmentu wycofania zależy od:

- Typu wykonywanych instrukcji (wstawianie, modyfikacja, usuwanie).
- Rozmiaru przetwarzanych rekordów.

Wstawienie rekordu do tabeli generuje mniej danych wycofania, niż usunięcie tego samego rekordu. **Dzieje się tak, dlatego, że usunięcie wymaga zapisania w segmencie wycofania całego wiersza, natomiast wstawienie zapisuje tylko identyfikator `ROWID`.**

Uwagi.

- Po utworzeniu segmentu wycofania jest on w trybie **`OFFLINE`**.
- Dla segmentu wycofania parametr **`MINEXTENTS`** musi być większy lub równy 2.
- Wartość parametru **`OPTIMAL`** musi wynosić przynajmniej tyle, co inicjalny rozmiar segmentu wyznaczony przez parametry **`INITIAL`**, **`NEXT`** i **`MINEXTENTS`**.
- Dla segmentów wycofania wskazane jest, aby rozmiar wszystkich zakresów był równy, czyli wartość parametru **`INITIAL`** powinna być równa wartości parametru **`NEXT`**.
- Należy unikać ustawiania parametru **`MAXEXTENTS`** na **`UNLIMITED`**, ponieważ może to powodować nadmierny wzrost segmentów wycofania i plików danych w przypadku wystąpienia błędów aplikacji.
- **W celu minimalizowania rywalizacji i fragmentacji należy umieszczać segmenty wycofania w specjalnie dla nich wydzielonej przestrzeni tabel.**

Aby segment wycofania był automatycznie włączany przez instancję musi być wyspecyfikowany w pliku parametrów, przez parametr **`ROLLBACK_SEGMENTS=(rb01,rb02,rb03)`**.

Liczba segmentów wycofania, które mogą być włączone przez instancję jest ustalana parametrem **`MAX_ROLLBACK_SEGMENT`**. Zaleca się ustawiać go na liczbę o jeden większą od liczby wymaganych w instancji niesystemowych segmentów wycofania.

W momencie otwierania bazy danych segmenty wycofania są przydzielane instancji.

Do usuwania segmentów wycofania służy polecenie **`DROP ROLLBACK SEGMENT`**. Jego użycie wymaga posiadania uprawnienia **`DROP ROLLBACK SEGMENT`**.

Składnia polecenia **`DROP ROLLBACK SEGMENT:`**

```
SQL> DROP ROLLBACK SEGMENT nazwa_segmentu;
```

Przykład 11.3.3. Wyłączenie i usunięcie segmentu wycofania **`rb01.`**

```
SQL> ALTER ROLLBACK SEGMENT rb01 OFFLINE;
 Segment wycofania został zmieniony.
SQL> DROP ROLLBACK SEGMENT rb01;
 Segment wycofania został usunięty.
```

§ 11.4. Automatyczne zarządzanie wycofywaniem transakcji

Od wersji 9i wprowadzono **możliwość automatycznego sposobu wycofywania transakcji `Undo Management`**.

W pliku `init<SID>.ora` lub `spfile<SID>.ora` musi znajdować się wtedy wpis:

```
undo_management=AUTO
undo_retention=10800
undo_tablespace=undotbs01
```

gdzie parametr `undo_retention` zaleca systemowi przechowanie starych wersji danych przez minimum 10800 sekund.

System w przestrzeni tabel `UNDOTBS01` tworzy kilka segmentów wycofywania i sam nimi zarządza.

Uwaga. Przy tworzeniu bazy danych nazwa przestrzeni wycofywania musi być zgodna z nazwą w pliku `init<SID>.ora` lub `spfile<SID>.ora` tzn. w naszym przykładzie `UNDOTBS01`.

§ 11.5. Zarządzanie przestrzeniami tabel (permanent tablespace) i plikami danych

Właściwości przestrzeni tabel:

- Należy tylko do jednej bazy.
- Składa się z jednego lub wielu plików systemu operacyjnego.
- Może być przyłączona i odłączona do bazy w trakcie jej pracy (poza przestrzenią **`SYSTEM`**).
- Może mieć status **`READ ONLY`** - „tylko odczyt” lub **`READ WRITE`** - „odczyt i zapis”, który można zmieniać podczas pracy bazy.

Mechanizm przestrzeni tabel umożliwia między innymi:

- Rozmieszczenie przechowywanych danych na różnych urządzeniach fizycznych, co zwiększa wydajności operacji zapisu i odczytu przez zmniejszenie rywalizacji dostępu do dysku.
- Dokonanie częściowych archiwizacji oraz częściowego odtworzenia archiwum.
- Przechowywanie statycznych danych na urządzeniach „tylko do odczytu”.

Baza danych zawsze zawiera przestrzeń tabel **`SYSTEM`**, która przechowuje słownik danych oraz segment wycofania **`SYSTEM.`**

Do tworzenia przestrzeni tabel służy polecenie **`CREATE TABLESPACE`**. Do jego użycia potrzebne jest uprawnienie systemowe **`CREATE TABLESPACE`**.

Do tworzenia przestrzeni tabel służy polecenie **`CREATE TABLESPACE`** o następującej składni:

```
SQL> CREATE TABLESPACE nazwa_przestrzeni_tabel
     DATAFILE 'plik' SIZE rozmiar [K|M]
       [ AUTOEXTEND ON NEXT rozmiar [K|M] [MAXSIZE rozmiar [K|M]] ]
     [ONLINE|OFFLINE]
```

```
      DEFAULT STORAGE
            (
              [INITIAL rozmiar [K|M]]
              [NEXT rozmiar [K|M]]
              [MINEXTENTS liczba]
              [MAXEXTENTS liczba]
              [PCTINCREASE liczba]
             );
```

gdzie:

plik	-	oznacza nazwę pliku wraz z pełnią ścieżką dostępu
rozmiar	-	oznacza rozmiar pliku podany w bajtach, kilobajtach (K) lub megabajtach (M)
ONLINE	-	powoduje utworzenie aktywnej przestrzeni tabel
OFFLINE	-	powoduje utworzenie nieaktywnej przestrzeni tabel

Przykład 11.5.1. Poniższa instrukcja tworzy przestrzeń tabel **baza** składającą się z dwóch plików danych.

```
SQL> CREATE TABLESPACE baza
     DATAFILE 'c:\oraclexe\oradata\xe\baza_1.dbf' SIZE 1M
                                  AUTOEXTEND ON NEXT 1M MAXSIZE 50M,
              'c:\oraclexe\oradata\xe\baza_2.dbf' SIZE 1M
                                  AUTOEXTEND ON NEXT 1M MAXSIZE 50M
     DEFAULT STORAGE
        (
           INITIAL 10K
           NEXT 50K
           MINEXTENTS 1
           MAXEXTENTS 999
           PCTINCREASE 10
         )
     ONLINE;
```

Przestrzenie tabel można zmieniać poleceniem **ALTER TABLESPACE**.

Składnia polecenia **ALTER TABLESPACE**:

```
SQL> ALTER TABLESPACE nazwa_przestrzeni_tabel
    { ADD DATAFILE 'plik' [autoextend_clause]
                [ , 'plik' [autoextend_clause] ]
     | RENAME DATAFILE 'plik' [,'plik']
       TO 'plik' [,'plik']
     | COALESCE
     | DEFAULT STORAGE storage_clause };
```

Przestrzeń tabel można usunąć z bazy poleceniem **DROP TABLESPACE.**

Składnia polecenia **DROP TABLESPACE**:

```
SQL> DROP TABLESPACE nazwa_przestrzeni_tabel
     [INCLUDING CONTENTS [CASCADE CONSTRAINTS] ];
```

gdzie:

INCLUDING CONTENTS	-	Powoduje usunięcie wszystkich segmentów (tabel z danymi) przed jej usunięciem.
CASCADE CONSTRAINTS	-	Powoduje usunięcie więzów spójności referencyjnej z tabel przechowywanych w innych przestrzeniach, które odwołują się do tabel usuwanej przestrzeni.

Rozmiar przestrzeni tabel można zwiększyć przez:

- Dodanie nowego pliku danych (**ALTER TABLESPACE**),

- Zwiększenie rozmiaru już istniejącego pliku.

Zmiana rozmiaru pliku danych może następować w wyniku:

- Wydania polecenia **ALTER DATABASE** (tzw. ręczna zmiana rozmiaru pliku),
- Automatycznego rozszerzenia.

Poniższe polecenie zwiększa rozmiar dwóch plików danych przestrzeni `baza`.

```
SQL> ALTER DATABASE
     DATAFILE 'c:\oraclexe\oradata\xe\baza_1.dbf',
              'c:\oraclexe\oradata\xe\baza_2.dbf'
     RESIZE 2M;
```

Automatyczne zwiększenie rozmiaru pliku danych może następować, jeżeli opcja pliku danych **AUTOEXTEND** jest włączona, co może być spowodowane użyciem następujących poleceń zawierających klauzulę **autoextend_clause**:

- `CREATE DATABASE` (dla tworzonych plików danych),
- `ALTER DATABASE` (dla istniejących plików danych),
- `CREATE TABLESPACE` (dla tworzonych plików danych),
- `ALTER TABLESPACE` (dla istniejących plików danych).

Składnia klauzuli `autoextend_clause`:

```
AUTOEXTEND
{OFF
     |ON [NEXT liczba [K|M]] [MAXSIZE {UNLIMITED|liczba [K|M]}]}
```

Przykład 11.5.2. Zobaczmy, jakie pliki danych przestrzeni tabel **baza** mogą ulegać automatycznemu rozszerzeniu.

```
SQL> SELECT file_name, autoextensible
     FROM dba_data_files
     WHERE tablespace_name = 'BAZA';
```

FILE_NAME	AUTOEXTENSIBLE
C:\ORACLEXE\ORADATA\XE\BAZA_1.DBF	YES
C:\ORACLEXE\ORADATA\XE\BAZA_2.DBF	YES

Jak widać wszystkie mogą być rozszerzane automatycznie przez system.

Odłączmy tę właściwość.

```
SQL> ALTER DATABASE
     DATAFILE 'c:\oraclexe\oradata\xe\baza_2.dbf',
              'c:\oraclexe\oradata\xe\baza_1.dbf'
     AUTOEXTEND OFF;
```

```
SQL> SELECT file_name, autoextensible
     FROM dba_data_files
     WHERE tablespace_name = 'BAZA';
```

FILE_NAME	AUTOEXTENSIBLE
C:\ORACLEXE\ORADATA\XE\BAZA_1.DBF	NO
C:\ORACLEXE\ORADATA\XE\BAZA_2.DBF	NO

Przestrzeń tabel może mieć ustawiony następujący status:

- ONLINE
- READ ONLY
- READ WRITE
- OFFLINE

Do przestrzeni tabel, która ma status **OFFLINE**, użytkownicy nie mają dostępu (tzn. nie mają dostępu do już istniejących obiektów oraz nie mogą tworzyć w niej nowych obiektów).

Podczas przełączania przestrzeni w stan **OFFLINE** system wykonuje punkt kontrolny (będzie o nim mowa w dalszej części) na wszystkich plikach bazy oraz odnotowuje to zdarzenie w słowniku danych i w pliku kontrolnym.

Zamknięcie i uruchomienie bazy nie zmienia stanów przestrzeni tabel.

Przykład 11.5.3. Sprawdźmy status przestrzeni tabel `users`.

```
SQL> SELECT tablespace_name, status
     FROM dba_tablespaces
     WHERE tablespace_name = 'USERS';

TABLESPACE_NAME  STATUS
USERS            ONLINE
```

Przykład 11.5.4. Przełączmy teraz przestrzeń tabel `users` w stan **OFFLINE**.

```
SQL> ALTER TABLESPACE users OFFLINE;

SQL> SELECT tablespace_name, status
     FROM dba_tablespaces
     WHERE tablespace_name = 'USERS';

TABLESPACE_NAME  STATUS
USERS            OFFLINE
```

Przykład 11.5.5. Poniższe polecenie pokazuje, że w odłączonej przestrzeni tabel nie można tworzyć obiektów.

```
SQL> CREATE TABLE t1 (id NUMBER) TABLESPACE users;
BŁĄD w linii 1:
ORA-XXXXX: przestrzeń tabel 'USERS' jest offline, nie można przydzielić w niej miejsca
```

Przestrzeń tabel ze statusem **READ ONLY** jest niedostępna dla operacji modyfikujących jej pliki danych. System ORACLE nigdy nie modyfikuje plików danych przestrzeni **READ ONLY**, co pozwala umieścić je na urządzeniach tylko do odczytu np. CD-ROM.

Aby możliwe było przełączenie przestrzeni tabel w tryb **READ ONLY** musi ona spełniać następujące warunki:

- musi być włączona (**ONLINE**),
- nie mogą być z nią związane żadne aktywne transakcje,
- nie może zawierać aktywnego segmentu wycofania,
- nie może podlegać w tym momencie gorącej archiwizacji (będzie opisane później).

Zalecanym sposobem przełączenia przestrzeni tabel w tryb **READ ONLY** jest uruchomienie instancji w trybie ograniczonego dostępu (**RESTRICT**).

Przełączenie przestrzeni tabel w tryb **READ ONLY** skutkuje wykonaniem punktów kontrolnych (zapis bloków z SGA z zatwierdzonymi zmianami do plików) na plikach danych tej przestrzeni.

Przy przechodzeniu przestrzeni ze stanu **READ ONLY** w tryb pozwalający na zapis, muszą być włączone wszystkie pliki danych tej przestrzeni. Przejście do stanu pełnego dostępu wykonuje się poleceniem:

```
SQL> ALTER TABLESPACE nazwa_przestrzeni_tabel READ WRITE;
```

Do przeniesienia plików danych można wykorzystać polecenia: **ALTER TABLESPACE** lub **ALTER DATABASE**. Polecenia te faktycznie sprawdzają tylko istnienie plików docelowych, przed ich wykonaniem należy utworzyć pliki docelowe przez skopiowanie plików źródłowych.

Przykład 11.5.6. Przykład zmiany położenia plików danych. Utwórzmy tabelę w przestrzeni tabel **baza**, aby po zakończeniu przeniesienia móc zaprezentować poprawność operacji.

```
SQL> CREATE TABLE t(k NUMBER) TABLESPACE baza;
SQL> INSERT INTO t VALUES (1);
SQL> COMMIT;
```

Poniższe zapytanie pokazuje, że istnieją dwa pliki danych przestrzeni **baza**.

```
SQL> SELECT file_name, status
     FROM dba_data_files
     WHERE tablespace_name = 'BAZA';
```

FILE_NAME	STATUS
C:\ORACLEXE\ORADATA\XE\BAZA_1.DBF	AVAILABLE
C:\ORACLEXE\ORADATA\XE\BAZA_2.DBF	AVAILABLE

Przełączmy przestrzeń baza w tryb tylko do odczytu.

```
SQL> ALTER TABLESPACE baza READ ONLY;
```

```
SQL> SELECT tablespace_name, status
     FROM dba_tablespaces
     WHERE tablespace_name = 'BAZA';
```

TABLESPACE_NAME	STATUS
BAZA	READ ONLY

Dalej wykorzystamy narzędzie **SQL*Plus**, aby zatrzymać, a następnie wystartować bazę w trybie **RESTRICT** (z bazą mogą łączyć się tylko użytkownicy posiadający uprawnienia **RESTRICTED SESSION**).

```
SQL> CONNECT sys AS SYSDBA
Hasło:
Połączony.
SQL> SHUTDOWN
Baza danych zamknięta.
Baza danych zdemontowana.
Instancja ORACLE zamknięta.
SQL> STARTUP RESTRICT
Instancja ORACLE wystartowała.
Baza danych zamontowana.
Baza danych otwarta.
```

Przełączamy przestrzeń w tryb **OFFLINE**.

```
SQL> ALTER TABLESPACE baza OFFLINE;
Instrukcja przetworzona.
```

Kopiujemy plik przestrzeni tabel do katalogu **C:\baza** oraz wykonujemy poniższe polecenie.

```
SQL> ALTER TABLESPACE baza
     RENAME DATAFILE
               'c:\oraclexe\oradata\xe\baza_1.dbf',
               'c:\oraclexe\oradata\xe\baza_2.dbf'
     TO
          'c:\baza\baza_1.dbf',
          'c:\baza\baza_2.dbf';
Instrukcja przetworzona.
```

Przełączamy przestrzeń w tryb **ONLINE** i ponownie restartujemy bazę.

```
SQL> ALTER TABLESPACE baza ONLINE;
Instrukcja przetworzona.

SQL> SHUTDOWN
Baza danych zamknięta.
Baza danych zdemontowana.
Instancja ORACLE zamknięta.

SQL> STARTUP
Instancja ORACLE wystartowała.
Baza danych zamontowana.
Baza danych otwarta.
```

Poniższe zapytanie pokazuje, że pliki zostały przeniesione i obiekty przestrzeni są dostępne.

```
SQL> SELECT file_name, status
     FROM dba_data_files
     WHERE tablespace_name = 'BAZA';
```

FILE_NAME	STATUS
C:\BAZA\BAZA_1.DBF	AVAILABLE
C:\BAZA\BAZA_2.DBF	AVAILABLE

```
SQL> SELECT * FROM t;
```

§ 11.6. Zarządzanie przestrzeniami tymczasowymi (temporary tablespace)

Tymczasowe przestrzenie (temporary tablespace) służą do przechowywania pośrednich wyników. Nie można w niej tworzyć obiektów (np. tabel, indeksów).

W bazie może występować wiele takich przestrzeni.

Użytkownik może mieć prawo do wielu takich przestrzeni.

Do tworzenia tymczasowej przestrzeni służy polecenie **CREATE TEMPORARY TABLESPACE** o następującej składni:

```
SQL> CREATE TABLESPACE nazwa_przestrzeni_tabel
     TEMPFILE 'plik' SIZE rozmiar [K|M]
       [ AUTOEXTEND ON NEXT rozmiar [K|M] [MAXSIZE rozmiar [K|M]] ]
     [ONLINE|OFFLINE]
     DEFAULT STORAGE
          (
            [INITIAL rozmiar [K|M]]
            [NEXT rozmiar [K|M]]
            [MINEXTENTS liczba]
            [MAXEXTENTS liczba]
            [PCTINCREASE liczba]
```

```
        );
```

Wiele poleceń z grupy **ALTER TABLESPACE** … stosuje się i do tych przestrzeni.

§ 11.7. Zarządzanie przestrzeniami wycofywania (undo tablespace)

Przestrzenie wycofywania (undo tablespace) służą do przechowywania segmentów wycofywania przy automatcznym zarządzaniu wycofywaniem (automanagement). Nie można w niej tworzyć obiektów (np. tabel, indeksów).

Do tworzenia przestrzeni wycofywania służy polecenie **CREATE UNDO TABLESPACE** o następującej składni:

```
SQL> CREATE UNDO TABLESPACE nazwa_przestrzeni_tabel
     DATAFILE 'plik' SIZE rozmiar [K|M]
       [ AUTOEXTEND ON NEXT rozmiar [K|M] [MAXSIZE rozmiar [K|M]] ]
     [ONLINE|OFFLINE]
     DEFAULT STORAGE
          (
            [INITIAL rozmiar [K|M]]
            [NEXT rozmiar [K|M]]
            [MINEXTENTS liczba]
            [MAXEXTENTS liczba]
            [PCTINCREASE liczba]
          );
```

Nazwa tej przestrzeni musi być zgodna z nazwą występującą w parametrze

```
undo_tablespace=undotbs01
```

12. OBIEKTY BAZY DANYCH (POWTÓRZENIE)

§ 12.1. Tabela

Tabela bazodanowa jest to obiekt, w którym logicznie przechowywane są dane. Polecenie tworzące tabele wygląda następująco:

```
SQL> CREATE TABLE nazwa_tabeli
     (atrybut typ (DEFAULT wyrażenie] [,..])
     [PCTFREE liczba]
     [PCTUSED liczba]
     [INITRANS liczba]
     [MAXTRANS liczba]
     [TABLESPACE przestrzen_tabel]
     [STORAGE (
               [INITIAL liczba [K/M]]
               [NEXT liczba [K/M]
               [PCTINCREAE liczba]
               [MINEXTENTS liczba]
               [MAXEXTENTS liczba]
              )];
```

gdzie:

PCTFREE liczba	-	Ilość wolnej przestrzeni pozostawionej w blokach.
PCTUSED liczba	-	Procent wypełnienia bloku, poniżej którego blok traktowany jest za dostępny.
przestrzen_tabel	-	Nazwa przestrzeni tabel, w której składowane będą dane.
INITIAL liczba	-	Rozmiar pierwszego rozszerzenia, jakie zostanie przydzielony tabeli.
NEXT liczba	-	Rozmiar drugiego rozszerzenia.
PCTINCREASE liczba	-	Procentowy przyrost rozmiaru każdego kolejnego rozszerzenia.
MINEXTENTS liczba	-	Liczba rozszerzeń, jakie zostaną przydzielone w chwili tworzenia.
MAXEXTENTS liczba	-	Maksymalna liczba rozszerzeń, z jakich może składać się tabela.

W przypadku braku któregoś parametru przyjmowana jest wartość domyślna.

Przesuwanie tabeli do innej przestrzeni tabel:

```
ALTER TABLE nazwa_tabeli MOVE  TABLESPACE nazwa_przestrzeni;
```

Można też przesuwać pojedyńcze partycje w przypadku tabel partycjonowanych.

Usuwanie tabeli:

```
SQL> DROP TABLE nazwa_tabeli [CASCADE CONSTRAINTS];
```

Informacje o istniejących tabelach można znaleźć w perspektywach:

DBA_TABLES, ALL_TABLES, USER_TABLES

§ 12.2. Indeksy

Są to struktury fizyczne tworzone na żądanie użytkownika, służą one do skrócenia czasu wyszukiwania rekordów spełniających warunki selekcji określone w zapytaniu.

```
SQL> CREATE [UNIQUE | BITMAP] INDEX nazwa_indeksu
     ON tabela(atrybut [,…])
     [PCTFREE liczba]
     [PCTUSED liczba]
     [INITRANS liczba]
     [MAXTRANS liczba]
     [TABLESPACE przestrzen_tabel]
     [STORAGE (
```

```
                [INITIAL liczba [K|M]]
                [NEXT liczba [K|M]]
                [PCTINCREAE liczba]
                [MINEXTENTS liczba]
                [MAXEXTENTS liczba]
               )];
```

Usuwanie indeksów:

```
SQL> DROP INDEX nazwa_indeksu;
```

Informacje o istniejących indeksach można znaleźć w perspektywach:

DBA_INDEXES,	USER_INDEXES,	ALL_IND_COLUMNS
ALL_INDEXES,	DBA_IND_COLUMNS,	USER_IND_COLUMNS

§ 12.3. Perspektywy

Struktury ograniczające zakres dostępnych danych do atrybutów krotek określonych w definicji; są definiowane na bazie, z co najmniej jednej relacji lub innej perspektywy; jest pamiętana wyłącznie w postaci definicji.

Perspektywy stosowane są w celu:

- Ograniczenia dostępu do tabel bazy danych.
- Uproszczenia zapytań w stosunku do zapytań kierowanych bezpośrednio do tabel.
- ….

```
SQL> CREATE [OR REPLACE] VIEW nazwa_perspektywy
     AS SELECT …
     [WITH CHECK OPTION];
```

Klauzula WITH CHECK OPTION powoduje, że w trakcie wstawiania i modyfikacji danych za pomocą perspektywy sprawdzane są warunki nałożone na nią i nie ma możliwości wstawienia niepasujących danych.

Usuwanie perspektyw:

```
SQL> DROP VIEW nazwa_perspektywy;
```

§ 12.4. Synonimy

Synonimy są to obiekty pozwalające zastąpić nazwy obiektów bazy danych innymi nazwami.

```
SQL> CREATE [PUBLIC] SYNONYM nazwa FOR obiekt;
```

Uwaga. Utworzenie publicznego synonimu nie daje jeszcze uprawnienia dostępu do obiektu, którego ten synonim dotyczy.

Przykład 12.4.1. Użytkownik kadry posiadający uprawnienie systemowe CREATE PUBLIC SYNONYM może utworzyć synonim **o** do tabeli osoby i przyznać uprawnienie SELECT do tabeli osoby użytkownikowi pbd.

```
SQL> CREATE PUBLIC SYNONYM o FOR osoby;
SQL> GRANT SELECT ON osoby TO pbd;
```

Po tym użytkownik **pbd** może wykonać polecenia:

```
SQL> SELECT * FROM kadry.osoby;
```

lub

```
SQL> SELECT * FROM o;
```

13. MONITOROWANIE PRACY UŻYTKOWNIKÓW

Dla administratora systemu Oracle *monitorowanie* (ang. *auditing*) jest procesem polegającym na zapisywaniu czynności dotyczących bazy danych.

W zależności od celu rozróżniane są trzy metody monitorowania pracy użytkowników:

- monitorowanie poleceń - **STATEMENT AUDITING**,
- monitorowanie uprawnień - **PRIVILEGE AUDITING**,
- monitorowanie obiektów - **OBJECT AUDITING**.

Włączenie możliwości monitorowania w czasie pracy systemu jest możliwe tylko wtedy, gdy w pliku konfiguracyjnym umieszczony jest parametr:

```
AUDIT_TRAIL= parametr
```

gdzie `parametr` może przybierać jedną z następujące wartości:

DB lub **TRUE**	- zapis następuje do **Dziennika Obserwacji** systemu ORACLE,
NONE	- wyłączenie monitorowania.

Monitorowanie użytkowników bardzo obciąża serwer bazy danych.

Zaleca się korzystanie z monitorowania tylko w konieczności.

Do wykonania poleceń **AUDIT** i **NOAUDIT** potrzebne jest uprawnienie systemowe **AUDIT SYSTEM.**

```
SQL> GRANT audit system TO nazwa_użytkownika;
```

§ 13.1. Monitorowanie poleceń

Włączenie monitorowania poleceń

```
SQL> AUDIT polecenie [,polecenie [...]]
     [BY użytkownik [,użytkownik [...]]
     [BY { SESSION | ACCESS} ]
     [WHENEVER [NOT] SUCCESSFUL];
```

gdzie:

`polecenie`	-	Nazwa monitorowanych poleceń. Np. `USER, VIEW, TABLE, ROLE, INDEX, PROCEDURE, SYSTEM AUDIT, TABLESPACE,...`
`użytkownik`	-	Nazwa monitorowanego użytkownika.
`BY SESSION`	-	Rejestrowany będzie tylko jeden rekord na sesję.
`BY ACCESS`	-	Rejestrowane będzie każde wykonane polecenie.
`SUCCESSFUL`	-	Monitorowane będą tylko poprawnie zakończone polecenia SQL.
`NOT SUCCESSFUL`	-	Monitorowane będą tylko nieudane polecenia SQL.

Informacje o aktualnie włączonym monitorowaniu można znaleźć w perspektywie

DBA_STMT_AUDIT_OPTS.

Wyłączenie monitorowania poleceń:

```
SQL> NOAUDIT polecenie [,polecenie [...]]
     [BY użytkownik [,użytkownik [...]]
     [WHENEVER [NOT] SUCCESSFUL];
```

Przykład 13.1.1. Włączenie i wyłączenie monitorowania poleceń.

Włączenie monitorowania poleceń.

```
SQL> AUDIT table, session
     BY kadry
     BY ACCESS
     WHENEVER SUCCESSFUL;
```

Przykładowo polecenie włącza monitorowanie poleceń kategorii **table** (CREATE TABLE, DROP TABLE, TRUNCATE TABLE, …) i **session** (przyłączenie do bazy i odłączenie) wykonywanych przez użytkownika **kadry**. Rejestrowane będzie każde pomyślne wykonanie polecenia.

Odczytanie informacji o aktualnie włączonym monitorowaniu poleceń:

```
SQL> SELECT * FROM dba_stmt_audit_opts;
```

Wyłączenie monitorowania poleceń.

```
SQL> NOAUDIT table
     BY kadry
     WHENEVER SUCCESSFUL;
```

Pozostaje monitorowanie polecenia kategorii session.

Przykłady monitorowanych poleceń:

Kategoria poleceń	Opis obejmowanych poleceń SQL
ALTER SYSTEM	ALTER SYSTEM
CLUSTER	CREATE CLUSTER, ALTER CLUSTER, TRUNCATE CLUSTER, DROP CLUSTER
INDEX	CREATE INDEX, ALTER INDEX, REBUILD INDEX
NOT EXISTS	Wszystkie polecenia SQL kończące się błędem w wyniku odwołania do nieistniejących obiektów
PROCEDURE	CREATE FUNCTION, CREATE PACKAGE, CREATE PACKAGE BODY, CREATE PROCEDURE, DROP FUNCTION, DROP PACKAGE, DROP PROCEDURE, …
ROLE	CREATE ROLE, DROP ROLE, …
SEQUENCE	CREATE SEQUENCE, DROP SEQUENCE, …
SESSION	Przyłączenie do bazy i odłączenie.
SYNONYM	CREATE SYNONYM, DROP SYNONYM,…
SYSTEM AUDIT	AUDIT, NO AUDIT, …
SYSTEM GRANT	Przyznanie i odebranie uprawnień systemowych.
TABLE	CREATE TABLE, ALTER TABLE, DROP TABLE, …
TABLESPACE	CREATE TABLESPACE, ALTER TABLESPACE, DROP TABLESPACE, …
TRIGGER	CREATE TRIGGER, ALTER TRIGGER ENABLE|DISABLE, ENABLE, DISABLE,…
USER	CREATE USER, ALTER USER, DROP USER, …
VIEW	CREATE VIEW, DROP VIEW, …
…	…

Rysunek 13.1.1. Przykłady monitorowanych poleceń.

Wykaz poleceń, które podlegają monitorowaniu można odczytać z widoku

STMT_AUDIT_OPTION_MAP

§ 13.2. Monitorowanie uprawnień

Monitorowanie uprawnień polega na automatycznym rejestrowaniu prób wykorzystania wybranych uprawnień systemowych posiadanych przez użytkownika.

Włączenie monitorowania uprawnień

```
SQL> AUDIT uprawnienie [,uprawnienie [...]]
     [BY użytkownik [,użytkownik [...]]
     [BY { SESSION | ACCESS} ]
     [WHENEVER [NOT] SUCCESSFUL];
```

gdzie:

uprawnienie	-	Nazwa monitorowanego uprawnienia systemowego. Np. `ALTER USER, CREATE VIEW, CREATE ANY TABLE,...`
użytkownik	-	Nazwa monitorowanego użytkownika.
BY SESSION	-	Rejestrowany będzie tylko jeden rekord na sesję.
BY ACCESS	-	Rejestrowane będzie każde wykonane polecenie wykorzystujące monitorowane uprawnienie.
SUCCESSFUL	-	Monitorowane będą tylko poprawnie zakończone polecenia SQL.
NOT SUCCESSFUL	-	Monitorowane będą tylko nieudane polecenia SQL.

Informacje o aktualnie włączonym monitorowaniu uprawnień można znaleźć w perspektywie `DBA_PRIV_AUDIT_OPTS`.

Wykaz uprawnień systemowych, które podlegają monitorowaniu można odczytać z widoku `SYSTEM_PRIVILEGE_MAP`.

Wyłączenie monitorowania uprawnień

```
SQL> NOAUDIT uprawnienie [,uprawnienie [...]]
     [BY użytkownik [,użytkownik [...]]
     [WHENEVER [NOT] SUCCESSFUL];
```

Przykład 13.2.1. Włączenie i wyłączenie monitorowania uprawnień.

Włączenie monitorowania uprawnień.

```
SQL> AUDIT create any table, create procedure
     BY kadry
     BY ACCESS
     WHENEVER SUCCESSFUL;
```

Odczytanie informacji o aktualnie włączonym monitorowaniu uprawnień:

```
SQL> SELECT * FROM dba_priv_audit_opts;
```

Wyłączenie monitorowania uprawnień.

```
SQL> NOAUDIT create any table
     BY kadry
     WHENEVER SUCCESSFUL;
```

Pozostaje monitorowanie uprawnienia `create procedure`.

§ 13.3. Monitorowanie obiektów

Monitorowanie obiektów stosuje się w sytuacji, gdy wymagane jest zawężenie monitorowania wyłącznie do wybranych obiektów bazy danych.

Włączenie monitorowania obiektów:

```
SQL> AUDIT operacja
    ON obiekt
     [BY { SESSION | ACCESS} ]
```

```
    [WHENEVER [NOT] SUCCESSFUL];
```

gdzie

operacja	- Nazwa monitorowanej operacji na danym obiekcie. Np. ALTER, DELETE, SELECT, UPDATE,...
użytkownik	- Nazwa monitorowanego użytkownika.
BY SESSION	- Rejestrowany będzie tylko jeden rekord na sesję.
BY ACCESS	- Rejestrowane będzie każde wykonane monitorowanej operacji na danym obiekcie.
SUCCESSFUL	- Monitorowane będą tylko poprawne dostępy do obiektu.
NOT SUCCESSFUL	- Monitorowane będą tylko nieudane dostępy do obiektu.

Informacje o aktualnie włączonym monitorowaniu obiektów można znaleźć w perspektywie

DBA_OBJ_AUDIT_OPTS.

Wyłączenie monitorowania uprawnień:

```
SQL> NOAUDIT operacja
     ON obiekt
     [BY { SESSION | ACCESS} ]
     [WHENEVER [NOT] SUCCESSFUL];
```

Przykład 13.3.1. Włączenie i wyłączenie monitorowania obiektów.

Włączenie monitorowania obiektów.

```
SQL> AUDIT update, delete
     ON kadry.wydzialy
     BY ACCESS
     WHENEVER SUCCESSFUL;
```

Odczytanie informacji o aktualnie włączonym monitorowaniu obiektów:

```
SQL> SELECT * FROM dba_obj_audit_opts;
```

Wyłączenie monitorowania obiektów.

```
SQL> NOAUDIT update
     ON kadry.wydzialy
     WHENEVER SUCCESSFUL;
```

Pozostaje monitorowanie operacji `delete` na objekcie `kadry.wydzialy`.

Monitorowanie obiektów może dotyczyć następujących operacji:

Operacja	Tabela	Widok	Licznik	Procedura
ALTER	X		X	
AUDIT	X	X	X	X
COMMENT	X	X		
DELETE	X	X		
EXECUTE				X
GRANT	X	X	X	X

Operacja	Tabela	Widok	Licznik	Procedura
INDEX	X			
INSERT	X	X		
LOCK	X	X		
RENAME	X	X		X
SELECT	X	X	X	
UPDATE	X	X		

Rysunek 13.3.1. Operacje monitorowanych obiektów.

§ 13.4. Odczytywanie dziennika obserwacji

Do uzyskania informacji o monitorowaniu można uzyskać w następujących perspektywach:

DBA_AUDIT_TRAIL, DBA_AUDIT_STATEMENT, DBA_AUDIT_SESSION,

DBA_AUDIT_OBJECT, DBA_AUDIT_EXISTS.

Każda z tych perspektyw posiada odpowiednik USER_*.

§ 13.5. Obsługa dziennika obserwacji

Obsługa dziennika obserwacji, polega głównie na periodycznym usuwaniu zapisów obserwacji i zerowaniu dziennika.

Zapisy z dziennika należy usuwać poprzez:

- Usunięcie wszystkich rekordów.
- Usunięcie wybranych rekordów.
- Archiwizację rekordów obserwacji do innej tabeli.

Przykład 13.5.1. Usunięcie z dziennika wszystkich rekordów sprzed trzech miesięcy.

```
SQL> DELETE FROM sys.aud$  WHERE ntimestamp#< SYSDATE-90;
```

Przykład 13.5.2. Usunięcie z dziennika obserwacji wszystkich rekordów.

```
SQL> TRUNCATE TABLE sys.aud$;
```

14. PLIKI ŚLADU (TRACE FILES) I PLIKI ALERTU (ALERT FILE)

§ 14.1. Pliki śladu

Pliki śladu (*trace files*) są plikami tekstowymi zawierającymi informacje o błędach wykrywanych przez proces serwera lub procesy tła, jakie wystąpiły w trakcie pracy instancji.

Administrator może wykorzystywać te informacje do analizowania i wykrywania przyczyn awarii.

Pliki śladu mogą zawierać także wszystkie polecenia realizowane w ramach sesji.

Wśród statystyk, które mogą być zbierane przez narzędzie `TRACE` są ilości czasu zajmowane przez pewne działania (opcja `timed_statistic=TRUE`).

Wartość tą można ustawić w `pliku init<SID>.ora/spfile<SID>.ora`, za pomocą polecenia `ALTER SYSTEM`, lub dla danej sesji (poleceniem `ALTER SESSION`).

Wzrost obciążenia po włączeniu tej opcji jest nieznaczny.

Parametry związane z plikami śladu:

BACKGROUND_DUMP_DEST	– Podaje ścieżkę na dysku, gdzie umieszczane są pliki śladu z opisem błędów wykrytych przez procesy tła.
USER_DUMP_DEST	– Podaje ścieżkę na dysku, gdzie umieszczane są pliki śladu z opisem błędów wykrytych przez proces serwera.
SQL_TRACE=TRUE	– **Podawany w pliku parametrów powoduje włączenie zapisu do plików śladu informacji na temat realizowanych poleceń SQL. Informacje o błędach zapisywane są niezależnie od wartości tego parametru.**
TIMED_STATISTICS=TRUE	– Podawany w pliku parametrów powoduje włączenie generowania statystyki dotyczącej ilości czasu potrzebnego do wykonania poleceń.

Parametry mogą być włączone dla bieżącej sesji poleceniem:

```
SQL> ALTER SESSION SET sql_trace = TRUE;
SQL> ALTER SESSION SET timed_statistics = TRUE;
```

lub na poziomie systemu poleceniami:

```
SQL> ALTER SYSTEM SET sql_trace = TRUE SCOPE=SPFILE;  /*wymaga restartu*/
SQL> ALTER SYSTEM SET timed_statistics = TRUE SCOPE=BOTH;
```

Przykład 14.1.1. Przykład fragmentu pliku śladu.

```
Dump file c:\oraclexe\app\oracle\admin\xe\udump\xe_ora_3348.trc

*** 2006-04-03 09:18:28.000
*** SESSION ID:(12.59) 2006-04-03 09:18:28.000
=====================
PARSING IN CURSOR #1 len=32 dep=0 uid=0 oct=42 lid=0 tim=1711438030 hv=1197935484 ad='66da73b0'
alter session set sql_trace=true
END OF STMT
EXEC #1:c=0,e=23427,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=1711429730
*** 2006-04-03 09:19:02.000
=====================
PARSE ERROR #1:len=33 dep=0 uid=0 oct=42 lid=0 tim=1797105600 err=922
alter session timed_statistics=2
=====================
PARSING IN CURSOR #1 len=23 dep=0 uid=0 oct=3 lid=0 tim=1838306445 hv=1731813439 ad='66da44f8'
select * from all_users
END OF STMT
PARSE #1:c=30043,e=146721,p=1,cr=34,cu=0,mis=1,r=0,dep=0,og=4,tim=1838306442
EXEC #1:c=0,e=19,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1838307082
FETCH #1:c=0,e=55,p=0,cr=7,cu=0,mis=0,r=1,dep=0,og=4,tim=1838307282
FETCH #1:c=0,e=159,p=0,cr=51,cu=0,mis=0,r=15,dep=0,og=4,tim=1838309344
FETCH #1:c=0,e=196,p=0,cr=49,cu=0,mis=0,r=15,dep=0,og=4,tim=1838466273
FETCH #1:c=0,e=15025,p=1,cr=4,cu=0,mis=0,r=0,dep=0,og=4,tim=1838521880
```

```
STAT #1 id=1 cnt=31 pid=0 pos=1 obj=0 op='NESTED LOOPS  '
STAT #1 id=2 cnt=31 pid=1 pos=1 obj=0 op='NESTED LOOPS  '
STAT #1 id=3 cnt=31 pid=2 pos=1 obj=22 op='TABLE ACCESS FULL OBJ#(22) '
STAT #1 id=4 cnt=31 pid=2 pos=2 obj=16 op='TABLE ACCESS CLUSTER OBJ#(16) '
STAT #1 id=5 cnt=31 pid=4 pos=1 obj=7 op='INDEX UNIQUE SCAN OBJ#(7) '
STAT #1 id=6 cnt=31 pid=1 pos=2 obj=16 op='TABLE ACCESS CLUSTER OBJ#(16) '
STAT #1 id=7 cnt=31 pid=6 pos=1 obj=7 op='INDEX UNIQUE SCAN OBJ#(7) '
..
```

Rysunek 14.1.1. Przykład fragmentu pliku śladu

Bardziej czytelną postać zawartości plików śladu można otrzymać przy pomocy programy `Tkprof`.

§ 14.2. Pliki alertu

Plik ostrzeżeń (*alert file*) jest plikiem tekstowym i zawiera chronologicznie uporządkowane komunikaty o błędach oraz inne informacje.

Parametr związany z plikiem ostrzeżeń:

`BACKGROUND_DUMP_DEST` - podaje ścieżkę na dysku, gdzie umieszczany jest plik ostrzeżeń.

Należy regularnie sprawdzać plik ostrzeżeń w celu wczesnego wykrycia ewentualnych problemów.

Plik ostrzeżeń posiada nazwę `alert_<SID>.log`.

Zawartość pliku ostrzeżeń:

- Komunikaty o błędach wewnętrznych.
- Błędy mówiące o uszkodzeniu bloków dyskowych.
- Błędy mówiące o zakleszczeniach.
- Informacje dotyczące operacji wykonywanych na bazie danych tzn.:
  - zamknięcie lub otwarcie bazy,
  - działania administratorskie przy pomocy poleceń DDL (zmiana struktury bazy danych), `ARCHIVELOG, RECOVER`,
  - wartości parametrów konfiguracyjnych z jakimi otwarto bazę danych.
- Błędy w odświeżaniu migawek,
- ….

Przykład 14.2.1. Przykład fragmentu pliku alertu

(**`BACKGROUND_DUMP_DEST=`** **`c:\oraclexe\app\oracle\admin\xe\bdump)`**.

```
Dump file c:\oraclexe\app\oracle\admin\xe\bdump\alert_xe.log

Wed Mar 28 01:43:02 2007
Starting ORACLE instance (normal)
LICENSE_MAX_SESSION = 0
LICENSE_SESSIONS_WARNING = 0
Picked latch-free SCN scheme 2
Using LOG_ARCHIVE_DEST_10 parameter default value as USE_DB_RECOVERY_FILE_DEST
Autotune of undo retention is turned on.
IMODE=BR
ILAT =10
LICENSE_MAX_USERS = 0
SYS auditing is disabled
ksdpec: called for event 13740 prior to event group initialization
Starting up ORACLE RDBMS Version: 10.2.0.1.0.
System parameters with non-default values:
  sessions                 = 49
  __shared_pool_size       = 83886080
```

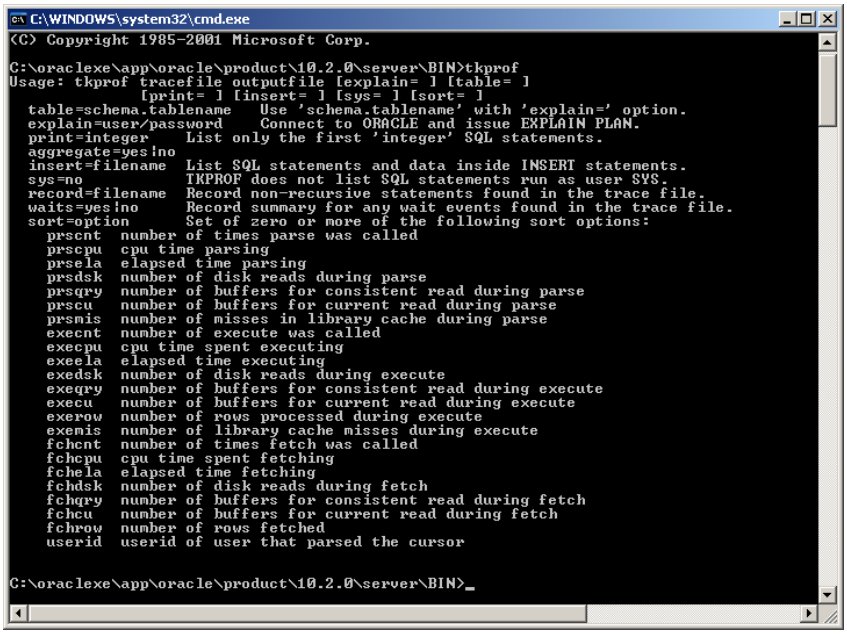
```
__large_pool_size        = 8388608
__java_pool_size         = 4194304
__streams_pool_size      = 0
spfile                   = C:\ORACLEXE\APP\ORACLE\PRODUCT\10.2.0\SERVER\DBS\SPFILEXE.ORA
sga_target               = 285212672
control_files            = C:\ORACLEXE\ORADATA\XE\CONTROL.DBF
__db_cache_size          = 184549376
compatible               = 10.2.0.1.0
db_recovery_file_dest    = C:\oraclexe\app\oracle\flash_recovery_area
db_recovery_file_dest_size= 10737418240
undo_management          = AUTO
undo_tablespace          = UNDO
remote_login_passwordfile= EXCLUSIVE
dispatchers              = (PROTOCOL=TCP) (SERVICE=XEXDB)
shared_servers           = 4
job_queue_processes      = 4
audit_file_dest          = C:\ORACLEXE\APP\ORACLE\ADMIN\XE\ADUMP
background_dump_dest     = C:\ORACLEXE\APP\ORACLE\ADMIN\XE\BDUMP
user_dump_dest           = C:\ORACLEXE\APP\ORACLE\ADMIN\XE\UDUMP
core_dump_dest           = C:\ORACLEXE\APP\ORACLE\ADMIN\XE\CDUMP
db_name                  = XE
open_cursors             = 300
os_authent_prefix        =
pga_aggregate_target     = 94371840
PMON started with pid=2, OS id=716
PSP0 started with pid=3, OS id=884
MMAN started with pid=4, OS id=2968
DBW0 started with pid=5, OS id=364
LGWR started with pid=6, OS id=2788
CKPT started with pid=7, OS id=3408
SMON started with pid=8, OS id=240
RECO started with pid=9, OS id=1248
CJQ0 started with pid=10, OS id=3412
MMON started with pid=11, OS id=2036
MMNL started with pid=12, OS id=2592
Wed Mar 28 01:43:02 2007
starting up 1 dispatcher(s) for network address '(ADDRESS=(PARTIAL=YES)(PROTOCOL=TCP))'...
starting up 4 shared server(s) ...
Oracle Data Guard is not available in this edition of Oracle.
Wed Mar 28 01:43:02 2007
ALTER DATABASE   MOUNT
Wed Mar 28 01:43:06 2007
Setting recovery target incarnation to 2
Wed Mar 28 01:43:07 2007
Successful mount of redo thread 1, with mount id 2499510630
Wed Mar 28 01:43:07 2007
Database mounted in Exclusive Mode
Completed: ALTER DATABASE   MOUNT
Wed Mar 28 01:43:07 2007
ALTER DATABASE OPEN
Wed Mar 28 01:43:07 2007
Thread 1 opened at log sequence 8
  Current            log#             1             seq#             8             mem#             0:
               C:\ORACLEXE\APP\ORACLE\FLASH_RECOVERY_AREA\XE\ONLINELOG\O1_MF_1_308HLVKN_.LOG
Successful open of redo thread 1
Wed Mar 28 01:43:07 2007
SMON: enabling cache recovery
Wed Mar 28 01:43:09 2007
Successfully onlined Undo Tablespace 1.
Wed Mar 28 01:43:09 2007
SMON: enabling tx recovery
Wed Mar 28 01:43:10 2007
Database Characterset is AL32UTF8
replication_dependency_tracking turned off (no async multimaster replication found)
Starting background process QMNC
QMNC started with pid=18, OS id=2248
Wed Mar 28 01:43:16 2007
Completed: ALTER DATABASE OPEN
Wed Mar 28 01:43:17 2007
db_recovery_file_dest_size of 10240 MB is 0.98% used. This is a
user-specified limit on the amount of space that will be used by this
database for recovery-related files, and does not reflect the amount of
space available in the underlying filesystem or ASM diskgroup.
```

Rysunek 14.2.1. Przykład fragmentu pliku alertu

15. CZĘSTO WYKORZYSTYWANE NARZĘDZIA

§ 15.1. Tkprof

Bardziej czytelną postać zawartości plików śladu można otrzymać przy pomocy programy `Tkprof`.



Rysunek 15.1.1. Pomoc dla programu `Tkprof`.

Przykład 15.1.2. Przykład wykorzystania programu `Tkprof`.

```
C:\TKPROF c:\oraclexe\app\oracle\admin\xe\udump\XE_ORA_13348.TRC
                                c:\oraclexe\xe_ora_3348.txt
```

Fragmenty pliku wynikowego:

```
TKPROF: Release 10.2.0.1.0 - Production on Sr Mar 28 01:10:06 2007
Copyright (c) 1982, 2005, Oracle.  All rights reserved.
Trace file: xe_ora_3348.trc
Sort options: default

********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
********************************************************************************
ALTER SESSION SET SQL_TRACE=TRUE

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute      1      0.00       0.01          0          0          0           0
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        1      0.00       0.01          0          0          0           0

Misses in library cache during parse: 0
Misses in library cache during execute: 1
Optimizer mode: ALL_ROWS
Parsing user id: SYS
********************************************************************************
ALTER SESSION SET TIMED_STATISTICS=TRUE

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
```

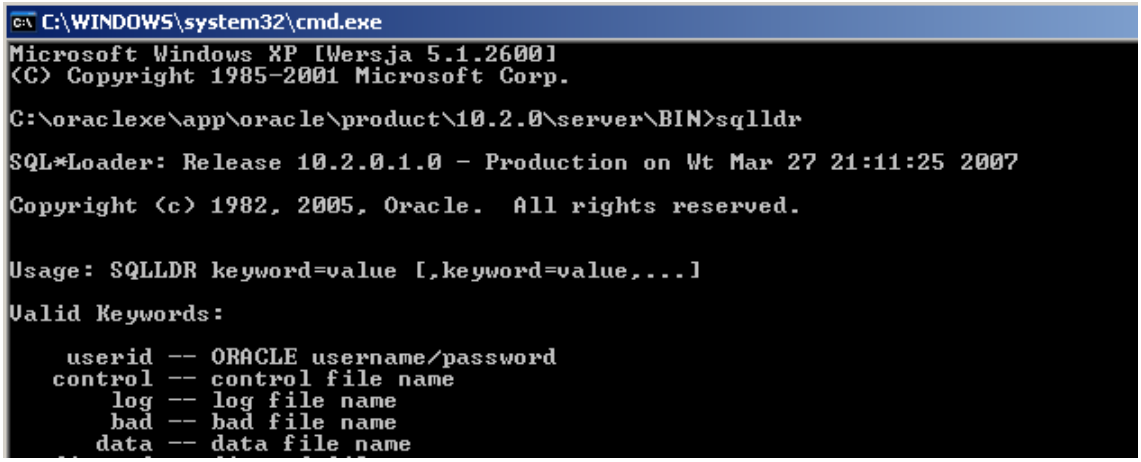
```
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2      0.01       0.00          0          0          0           0

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: SYS
********************************************************************************
select   obj#,type#,ctime,mtime,stime,status,dataobj#,flags,oid$,
spare1, spare2
from obj$ where owner#=:1 and name=:2 and namespace=:3 and
remoteowner is null and linkname is null and subname is null

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.00       0.00          0          0          0           0
Execute      4      0.00       0.00          0          0          0           0
Fetch        4      0.00       0.00          0         12          0           4
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       10      0.00       0.00          0         12          0           4

Misses in library cache during parse: 1
Misses in library cache during execute: 1
Optimizer mode: CHOOSE
Parsing user id: SYS   (recursive depth: 1)

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS BY INDEX ROWID OBJ$ (cr=3 pr=0 pw=0 time=67 us)
      1   INDEX RANGE SCAN I_OBJ2 (cr=2 pr=0 pw=0 time=39 us)(object id 37)

********************************************************************************
select /*+ rule */ bucket_cnt, row_cnt, cache_cnt, null_cnt,
timestamp#, sample_size, minimum, maximum, distcnt, lowval,
hival, density, col#, spare1, spare2, avgcln
from hist_head$ where obj#=:1 and intcol#=:2

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.00       0.00          0          0          0           0
Execute    110      0.01       0.00          0          0          0           0
Fetch      110      0.00       0.00          2        328          0         108
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      222      0.01       0.01          2        328          0         108

Misses in library cache during parse: 1
Misses in library cache during execute: 1
Optimizer mode: RULE
Parsing user id: SYS   (recursive depth: 2)

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS BY INDEX ROWID HIST_HEAD$ (cr=3 pr=0 pw=0 time=46 us)
      1   INDEX RANGE SCAN I_HH_OBJ#_INTCOL# (cr=2 pr=0 pw=0 time=26 us)(object id 257)
```

Rysunek 15.1.2. Przykład fragmentu pliku wynikowego z Tkprof

§ 15.2. SQL Loader

Narzędzie SQL Loader dostarczane wraz z oprogramowaniem bazy Oracle przeznaczone jest do masowego ładowania danych do bazy.

Program wywoływany z linii poleceń systemu operacyjnego jest jednym z najlepszych sposobów wstawiania danych do bazy i przy okazji jednym z najszybszych.



Rysunek 15.2.1. Pomoc dla programu `Sqlldr`.

Wywołanie programu **SQLLDR** bez parametrów wyświetla proste podpowiedzi.

W najprostszej konfiguracji do załadowania danych do bazy wystarczą dwa pliki:

- Plik z danymi,
- Plik kontrolny.

Przykład 15.2.2. Na koncie użytkownika `pbd` utworzona jest tabela `nazwy`.

```
SQL> CREATE TABLE nazwy (id NUMBER, nazwa VARCHAR2(10)UNIQUE);
```

Przykładowy plik z danymi **'c:\dane.txt'** może wyglądać następująco:

```
1,nazwa1
2,nazwa2
3,nazwa3
4,nazwa1
```

a plik kontrolny **'c:\kontrolny.txt'**:

```
load data
infile 'c:\dane.txt'
append into table nazwy
fields terminated by ','
(id, nazwa)
```

a wywołanie:

```
C:\SQLLDR pbd/pbd CONTROL=c:\kontrolny.txt
                                    LOG=c:\log.txt BAD=c:\zle.txt
```

Efektem takiego wywołania będzie załadowanie do tabeli **nazwy**, do kolumn o nazwach **id, nazwa**, trzech wierszy z pliku **c:\dane.txt**. Dane zostaną wstawione do tabeli **nazwy** znajdującej się w schemacie użytkownika `pbd` z hasłem `pbd`.

Poprawność wykonania całej operacji można weryfikować w pliku z rejestrem operacji, który zostanie utworzony (**c:\log.txt**) i w pliku z danymi nie wstawionymi do bazy (**zle.txt**), które można poprawić i ponownie spróbować załadować.

Plik **log.txt** wygląda następująco:

```
Control File:   d:\kontrolny.txt
```

```
Data File:      d:\dane.txt
  Bad File:     d:\zle.txt
  Discard File:  none specified

 (Allow all discards)
Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 256000 bytes
Continuation:    none specified
Path used:      Conventional

Table NAZWY, loaded from every logical record.
Insert option in effect for this table: APPEND
   Column Name              Position   Len  Term Encl Datatype
------------------------- ---------- ----- ---- ---- ---------------------
ID                             FIRST     *   ,       CHARACTER
NAZWA                           NEXT     *   ,       CHARACTER
Record 5: Rejected - Error on table NAZWY, column ID.
Column not found before end of logical record (use TRAILING NULLCOLS)
Record 4: Rejected - Error on table NAZWY.
ORA-00001: unique constraint (HR.SYS_C004969) violated

Table NAZWY:
  3 Rows successfully loaded.
  2 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.

Space allocated for bind array:                  33024 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:             5
Total logical records rejected:         2
Total logical records discarded:        0

Run began on Sr Mar 28 02:36:05 2007
Run ended on Sr Mar 28 02:36:06 2007

Elapsed time was:     00:00:00.63
CPU time was:         00:00:00.04
```

Rysunek 15.2.2. Przykład fragmentu pliku `log.txt` z programu SQL Loader

Plik `zle.txt` wygląda następująco:

```
4,nazwa3
```

Kompletne informacje na temat narzędzia `SQLLoader` i innych narzędzi można znaleźć w dokumentacji Oracle.

16. OPTYMALIZACJA ZAPYTAŃ (POWTÓRZENIE)

§ 16.1. Przykład (powtórzenie z Podstawy Baz Danych)

Mamy następujące relacje:

Dostawcy

Id	Nazwisko	Imie	…	…
1	Kowalski	Jan	…	…
2	Nowak	Anna	…	…
3	Norek	Tadeusz	…	…
…	…	…	…	…

Dostawy

…	Id_dostawcy	Data	Produkt	…
…	3	01-10-1990	P1	…
…	2	01-10-2000	P2	…
…	1	01-11-2001	P2	…
…	…	…	…	…

Przypuśćmy, że w relacji `dostawcy` jest **100 krotek**, a w relacji `dostawy` **10 000 krotek**, z których tylko **50** dotyczy produktu **P2**.

Podamy prosty przykład dający pewne pojęcie o tym, że dzięki optymalności można uzyskać wyraźną poprawę wydajności wykonania polecenia SQL.

Przykład 16.1.1. Chcemy wykonać zapytanie:

Podaj nazwiska dostawców dostarczających produkt P2.

Zapytanie ma postać:

```
SELECT nazwisko
FROM   dostawy d1 JOIN dostawcy d2 ON (d2.id = d1.id_dostawcy)
WHERE  d1.produkt = 'P2';
```

Pierwsza procedura wykonania zapytania:

1. Wykonanie operacji złączenia relacji `dostawy` i `dostawcy`.
   - Ten krok wymaga przeczytania 10 000 krotek z relacji dostawy.
   - Czytanie każdej ze 100 krotek z relacji dostawcy 10 000 razy (raz dla każdej krotki z relacji dostawy) .
   - Utworzenie pośredniego wyniku składającego się z 10 000 krotek i zapisanie tego wyniku na dysku (zakładamy, że w pamięci operacyjnej nie ma miejsca na składowanie 10000 pośrednich krotek.

- Wykonanie selekcji otrzymanych krotek w punkcie 1 do krotek zawierających produkt = ‘P2’.
  - Krok ten wymaga ponownego wczytania 10 000 krotek pośrednich.
  - Selekcja do krotek z produkt = ‘P2’ powoduje, że pozostaje 50 krotek pośrednich, które możemy przyjąć, że zmieszczą się w pamięci operacyjnej.

- Wykonanie operacji projekcji na wyniku z punktu 2.
  - Dokonanie projekcji wyniku otrzymanego w poprzednim etapie do atrybutu nazwisko (co najwyżej 50 krotek, które mogą pozostać w pamięci). Daje to końcowy wynik zapytania.

Druga procedura wykonania zapytania:

1. Wykonanie operacji selekcji na relacji `dostawy` do krotek dotyczących towar = ‘P2’.
   - Krok ten wymaga wczytania 10 000 krotek z relacji dostawy, ale w wyniku dostajemy relację, która zawiera tylko 50 krotek, którą możemy przechowywać w pamięci.

2. Wykonanie operacji złączenia relacji z kroku 1, która znajduje się w pamięci z relacją `dostawcy`.
   - Krok ten wymaga wczytania 100 krotek z relacji `dostawcy`, ale tylko raz, a nie po razie dla każdej krotki z kroku 1, gdyż wszystkie krotki znajdują się w pamięci.
   - Wynik zawiera 50 krotek i ciągle znajduje się w pamięci.
3. Wykonanie operacji projekcji na wyniku z punktu 2.
   - Dokonanie projekcji wyniku otrzymanego w poprzednim etapie do atrybutu nazwisko (co najwyżej 50 krotek, które mogą pozostać w pamięci). Daje to końcowy wynik zapytania.

Wniosek.

- Pierwsza procedura wymaga **1 020 000** operacji we-wy dla krotek, druga tylko **10 100**.
- Jeżeli weźmiemy jako miarę wydajności liczbę operacji we-wy (I/O), wówczas druga procedura byłaby około **100** razy lepsza niż druga.

§ 16.2. Podstawy optymalizacji zapytań (powtórzenie z Podstawy Baz Danych)

Zasadnicze etapy procesu optymalizacji (według C.J.Date):

1. Sformułowanie zapytania w jakiejś wewnętrznej postaci – niech będzie to algebra relacyjna (operacje `selekcji, projekcji, złączenie, union, minus, intersect,` ...).
2. Przekształcenie do postaci kanonicznej.
3. Wybór kandydatów do procedur niskiego poziomu.
4. Tworzenie różnych planów wykonania zapytania i wybór najtańszego rozwiązania.

Etap 1. Sformułowanie zapytania w jakiejś wewnętrznej postaci.

Jako wewnętrzną postać zapytania można wybrać pewien rodzaj **abstrakcyjnego drzewa składni**.

Rysunek 16.2.1. Abstrakcyjne drzewo składni.

Etap 2. Przekształcenie do postaci kanonicznej.

Na tym etapie optymalizator wykonuje szereg optymalizacji. Chodzi o to, że zapytanie możemy sformułować na wiele sposobów, pozornie odmiennych.

Przykład 16.2.2. Zapytanie „**Podaj nazwiska dostawców dostarczających produkt P2**” można sformułować na różne sposoby. Np.

```
SELECT nazwisko
FROM  dostawy d1, dostawcy d2
WHERE d2.id = d1.id_dostawcy  AND d1.produkt  IN ('P2');
```

Wydajność realizacji zapytania nie powinna zależeć od postaci jaką podał użytkownik.

Kolejnym krokiem jest przetworzenie pewnej postaci wewnętrznej do pewnej równoważnej postaci kanonicznej.

Definicja. Dany jest zbiór obiektów Q (np. zapytań) i pojęcie równoważności między obiektami (np. zapytanie `q1` jest równoważne zapytaniu `q2` wtedy i tylko wtedy gdy prowadzą do tego samego wyniku.

Definicja. Mówimy, że podzbiór C zbioru Q jest zbiorem postaci kanonicznych dla Q ze względu na podaną relację równoważności wtedy i tylko wtedy gdy każdy obiekt `q` ze zbioru Q jest równoważny tylko jednemu elementowi `c` ze zbioru C.

W celu przekształcenia wyniku z etapu 1 do równoważnej, ale bardziej wydajnej postaci, optymalizator używa pewnych dobrze określonych reguł transformacji.

Przykład 16.2.3. Wyrażenie

```
( A  JOIN  B ) WHERE selekcja-na-A
```

można zamienić na

```
( A  WHERE selekcja-na-A ) JOIN B
```

Tą transformację wykorzystaliśmy w naszym przykładzie.

Przykład 16.2.4. Inne reguły transformacji.

1. Wyrażenie

   ```
   ( A  WHERE selekcja-1 ) WHERE selekcja-2
   ```

   można zamienić na

   ```
   A  WHERE selekcja-1 AND selekcja-2
   ```

2. Wyrażenie

   ```
   ( A  [ projekcja-1 ] )  WHERE selekcja-1
   ```

   można zamienić na

   ```
   ( A  WHERE selekcja-1 )  [ projekcja-1 ]
   ```

5. Wyrażenia obliczeniowe i warunkowe.
   - `( a + b ) + c` jest równoważne `a + ( b + c )`
   - …

Uwaga. Wiele przekształceń opiera się na znajomości zależności funkcyjnych i kluczy kandydujących.

Etap 3. Wybór kandydatów do procedur niskiego poziomu.

- Po przekształceniu zapytania do bardziej korzystnej postaci (kanonicznej) optymalizator musi się zdecydować, jak wykonać przekształcone zapytanie.

- Na tym etapie rozpoczyna się analiza indeksów czy innych ścieżek dostępu (klastry, partycje, …).
- Zauważmy, że omawiając etap 1 i 2 nie zwracaliśmy na to uwagi.
- Strategia polega na spojrzeniu na wyrażenia stanowiące zapytanie jak na serię operacji „niskiego poziomu" dla których przygotowuje procedury implementacyjne.

Optymalizacja poleceń SQL jest procesem doboru odpowiednich struktur danych sposobu wykonywania tych poleceń.

Celem optymalizacji jest zminimalizowanie kosztu wykonania polecenia.

Przez **koszt** rozumie się czas zajętości procesora, urządzeń we/wy, urządzeń sieciowych, wykorzystanie innych zasobów systemu.

Optymalizacją zajmuje się specjalizowany moduł serwera bazy danych, zwany optymalizatorem (*optimizer*).

Optymalizator to część SZBD Oracle, która zajmuje się wybraniem sposobu odpowiedzi na wydane polecenia języka manipulowania danymi (DML).

§ 16.3. Metody optymalizacji

Optymalizacja jest procesem kosztownym, dlatego wykorzystuje się rożnego rodzaju metody przybliżone. Dwie podstawowe metody wykorzystywane w systemie Oracle to:

- **optymalizacja regułowa - oparta na pewnych zasadach (przestarzała);**
- **optymalizacja kosztowa.**

Metody te powiązane są z trybem pracy optymalizatora, odpowiednio: regułowym lub kosztowym.

Firma Oracle rekomenduje stosowanie metody kosztowej, która została wprowadzona w wersji 7.0 systemu i powinna być stosowana we wszystkich nowych aplikacjach.

§ 16.4. Optymalizacja kosztowa

Optymalizacja kosztowa opiera się o pewną funkcję kosztu, która umożliwia oszacowanie kosztu wykonania danego planu.

Optymalizator kosztowy, aby wybrać lepszy plan wykonania wykorzystuje informacje o strukturze i zawartości bazy, takie jak rozmiar tabeli, liczba wierszy, rozkład danych i warunki ich przechowywania w tabelach, klastrach, indeksach itp.

Większość tych informacji zwanych **statystykami** nie jest zbierana automatycznie przez system.

W celu umożliwienia pracy optymalizatora kosztowego, należy wyznaczyć statystyki za pomocą polecenia **`ANALYZE`**, lub korzystając z pakietu **`DBMS_STATS`**.

Zebrane statystyki służą do wyznaczenia optymalnego planu.

Optymalizator kosztowy opracowuje plan wykonania polecenia w trzech krokach:

- Generuje zbiór, wszystkich możliwych planów wykonania, podobnie jak dla metody regułowej.

- Dla każdego zapytania jest wyliczany koszt jego wykonania na podstawie zebranych statystyk. Na koszt składa się czas procesora, liczba operacji wejścia-wyjścia i pamięć potrzebna do wykonania zapytania.
- Optymalizator porównuje koszty i wybiera ten plan, który charakteryzuje się najmniejszym kosztem.

Domyślnym celem optymalizatora kosztowego jest wygenerowanie takiego planu wykonania, który zapewni uzyskanie najlepszej wydajności zapytania.

Można wyróżnić jeszcze dwa cele stosowania optymalizatora kosztowego:

- **Minimalna ilość zasobów** - celem optymalizacji jest zużycie najmniejszej ilości zasobów.
- **Najkrótszy czas odpowiedzi** - celem optymalizacji jest uzyskanie jak najkrótszego czasu odpowiedzi.

Odpowiedni wybór celu optymalizacji w danej aplikacji lub bazie pozwala dostroić zapytania SQL, stosownie do potrzeb i wymagań użytkownika.

§ 16.5. Cele optymalizacji

Celem optymalizacji może być:

1. **Czas odpowiedzi**. Optymalizacja czasu odpowiedzi jest szczególnie przydatna w aplikacjach interakcyjnych, gdzie najważniejsze jest zminimalizowanie czasu oczekiwania użytkownika na pojawienie się pierwszych wyników.
2. **Przepustowość**. Optymalizacja przepustowości stosowana jest zazwyczaj dla aplikacji typu wsadowego, np. raportów. W tym przypadku ważny jest czas wygenerowania całego raportu.

W systemie Oracle możliwy jest wybór optymalizatora i celu optymalizacji na trzech poziomach:

- **instancji,**
- **sesji użytkownika,**
- **pojedynczego polecenia SQL.**

Wybór optymalizatora na poziomie instancji dokonuje się za pomocą parametru konfiguracyjnego `OPTIMIZER_MODE` zapisanego w pliku `init<SID>.ora` lub `spfile<SID>.ora`.

Parametr ten może przyjąć następujące wartości:

`RULE, CHOOSE, FIRST_ROWS, ALL_ROWS.`

Wyboru optymalizatora i celu optymalizacji na poziomie instancji można również dokonać poleceniem:

```
SQL> ALTER SYSTEM SET optimizer_mode = all_rows;
```

Wyboru optymalizatora i celu optymalizacji na poziomie sesji można wykonać poleceniem:

```
SQL> ALTER SESSION SET optimizer_mode = first_rows(100);
```

Wyboru optymalizatora i celu optymalizacji na poziomie polecenia SQL dokonuje się za pomocą wskazówek:

RULE, CHOOSE, FIRST_ROWS, ALL_ROWS.

Poprzez stosownie wskazówek, można określić następujące elementy pracy optymalizatora:

1. **Najbardziej odpowiedni tryb pracy optymalizatora dla danego polecenia SQL.**
2. **Cel optymalizatora kosztowego.**
3. **Sposób uzyskiwania dostępu do danych.**
4. **Kolejność łączenia tabel.**
5. **Algorytm łączenia tabel.**
6. **Wybór przeglądu całej tabeli lub wykorzystania indeksów.**
7. **…**

Poniżej przedstawiono kilka przykładów sytuacji, w których wykorzystanie wskazówek może znacząco wpływać na poprawę wydajności systemu:

1. Indeks oparty na kolumnie, z dużą liczbą powtarzających się wartości. Nakazanie optymalizatorowi, by nie korzystał z indeksu w przypadku wartości, która się często powtarza, może być bardziej wydajne niż użycie indeksu.
2. Wyszukiwanie na podstawie indeksu bitmapowego. Jeżeli na tabeli zdefiniowano indeks bitmapowy oraz B*-tree, korzystniejsze jest użycie indeksu bitmapowego, mimo iż optymalizator preferuje indeks B*-tree.

§ 16.6. Wskazówki dla optymalizatora

Wskazówki przekazywane do optymalizatora, umieszczane są w komentarzu zapytania SQL zaraz po znaku „+”. Można je umieszczać jedynie za początkowym słowem kluczowym polecenia SQL: `SELECT, UPDATE, DELETE, INSERT`.

```
SELECT /*+ tekst wskazówki */  …
DELETE /*+ tekst wskazówki */  …
UPDATE /*+ tekst wskazówki */  …
INSERT /*+ tekst wskazówki */  …
```

W przypadku polecenia INSERT większość wskazówek jest ignorowana.

Dodanie znaku „+” na początku komentarza powoduje, że optymalizator traktuje komentarz, jako wskazówki.

Blok zapytania SQL może mieć tylko jeden komentarz.

Wskazówka umieszczona w komentarzu ma zastosowanie tylko dla bloku polecenia, w którym występuje.

Blok polecenia to:

1. Proste polecenie `SELECT, DELELE` lub `UPDATE`.
2. Główna część polecenia złożonego.
3. Podzapytanie polecenia złożonego.

Przykład 16.6.1. Przykład zastosowania wskazówki, która sugeruje optymalizatorowi wykonanie przeglądu całej tabeli `osoby`.

```
SELECT /*+ FULL(osoby) */ o.*
FROM osoby o
WHERE nazwisko='Kowalski';
```

W jednym komentarzu można umieścić klika wskazówek, o ile nie są wzajemnie sprzeczne. Wskazówki wielokrotne w wielu przypadkach są bardzo przydatne.

§ 16.7. Dostępne wskazówki

Pierwszą kategorią wskazówek są wskazówki określające tryb i cel optymalizatora, niezależnie od innych ustawień.

Cel określony we wskazówkach ma wyższy priorytet niż inne specyfikacje zdefiniowane za pomocą parametrów inicjalizacyjnych.

Do powyższej kategorii zaliczamy następujące wskazówki:

1. CHOOSE, 2. ALL_ROWS, 3. FIRST_ROWS, 4. RULE.

Wskazówka `RULE` wybiera optymalizator regułowy, a `CHOOSE` oznacza, że optymalizator sam wybiera tryb regułowy bądź kosztowy.

Składnia tej wskazówki jest następująca:

```
/*+ CHOOSE */
```

Przykład 16.7.1. Poniżej przedstawiono przykład zastosowania omawianej wskazówki.

```
SELECT /*+ CHOOSE*/ o.*,z.pensja
FROM osoby o, zatrudnienia z
WHERE o.id_os = z.id_os AND z.do IS NULL AND pensja <500;
```

Wskazówka `FIRST_ROWS` jest używana, aby ustawić cel optymalizatora kosztowego, na jak najkrótszy czas odpowiedzi dla pierwszych n rekordów.

Użycie tej wskazówki powoduje utworzenie planu wykonania, który przy jak najmniejszym wykorzystaniu zasobów zwraca n pierwszych rekordów.

Składnia tej wskazówki jest następująca:

```
/*+ FIRST_ROWS(n) */
```

Wskazówka `ALL_ROWS` oznacza, że optymalizator ma pracować w trybie kosztowym, uwzględniając ilość danych przetwarzanych w trakcie wykonywania całego zapytania.

Wykorzystując tą wskazówkę, plan wykonania polecenia SQL generowany jest w oparciu o jak najmniejszą ilość zasobów i największą ilość danych, jaka może zostać przetworzona.

Składnia tej wskazówki jest następująca:

```
/*+ ALL_ROWS */
```

Jeżeli w komentarzu pojawią się wskazówki dotyczące metod dostępu lub sposobów łączenia tabel, będą one miały wyższy priorytet nad `ALL_ROWS`.

Wskazówki `ALL_ROWS` i `FIRST_ROWS` są ignorowane w poleceniach `INSERT`, `UPDATE`, `DELETE` oraz w poleceniach `SELECT`, zawierających następujące konstrukcje:

- Operatory zbiorowe takie jak: `UNION, INTERSECT, MINUS`.
- Funkcje agregujące takie jak: `AVG, MIN, MAX, COUNT, SUM, STDEV lub VARIANCE`.
- Operator `DISTINCT`.

Zapytania z powyższymi konstrukcjami, nie mogą być zoptymalizowane do jak najkrótszego czasu odpowiedzi, ponieważ zastosowany operator wymaga odczytania wszystkich rekordów przed wyświetleniem wyników.

Wskazówka RULE włącza tryb regułowy optymalizatora. **Dodatkowo zastosowanie tej wskazówki powoduje ignorowanie wszystkich innych wskazówek dla danego bloku zapytania**.

Składnia tej wskazówki jest następująca:

```
/*+ RULE */
```

Przykład 16.7.2. Poniżej przedstawiono przykład zastosowania omawianej wskazówki.

```
SELECT /*+ RULE*/ o.*,z.pensja
FROM osoby o, zatrudnienia z
WHERE o.id_os = z.id_os AND z.do IS NULL AND pensja <500;
```

Stosując wskazówkę RULE nie możemy ocenić kosztu realizacji zapytania, gdyż wybrany za pomocą tej wskazówki tryb regułowy nie wyznacza kosztu wykonania dla danego polecenia.

Kolejną kategorią wskazówek są wskazówki specyfikujące sposób uzyskania dostępu do danych.

Do tej grupy zaliczamy następujące wskazówki:

1. `FULL`,
2. `INDEX`,
3. `INDEX_ASC`,
4. `INDEX_COMBINE`,
5. `INDEX_DESC`,
6. `NO_INDEX`,
7. `USE_CONCAT`
8. …

Na podstawie informacji o specyfice przechowywanych danych w bazie, możemy zdecydować o zastosowaniu przez optymalizator innej metody dostępu do danych niż domyślna. Zastosowana metoda, może okazać się bardziej efektywna.

Wskazówki są akceptowane tylko wtedy, gdy istnieją poprawne obiekty, do których się odwołują i gdy pozwala na to składnia użytego polecenia SQL. W przeciwnym przypadku, wskazówka jest ignorowana (np. brak indeksu).

Wskazując metodę dostępu, należy podać nazwę tabeli, do której będzie realizowany dostęp.

Wskazówka `FULL` wymusza użycie przeglądu całej tabeli dla podanej jako parametr. Wskazówka ta oznacza również, że pełny przegląd ma być przeprowadzony nawet wtedy, gdy w danej tabeli są zdefiniowane indeksy.

Składnia tej wskazówki jest następująca:

```
/*+ FULL(tabela) */
```

Przykład 16.7.3. Wskazówka zawarta w poniższym przykładzie wymusza na optymalizatorze pełny przegląd tabeli `wydzialy`.

```
SELECT /*+ FULL(wydzialy) */ w.nazwa
FROM wydzialy w
WHERE w.id_w > 4;
```

Wskazówka INDEX, INDEX_ASC, INDEX_DESC wymusza użycie indeksowanego przeglądu tabeli podanej jako parametr.

Wskazówka ta powoduje, że optymalizator użyje indeksu nawet wtedy, gdy normalnie indeks taki nie zostałby wykorzystany ze względu na małą liczbę różnych wartości.

Składnia tych wskazówek jest następująca:

```
/*+ INDEX(tabela index [index..index])*/
/*+ INDEX_ASC(tabela index [index..index])*/
/*+ INDEX_DESC(tabela index [index..index])*/
```

Wskazówka INDEX_COMBINE jawnie nakazuje, o ile istnieją, wykorzystanie indeksów bitmapowych w celu dostępu do danych w tabeli.

Wskazówka ta może dotyczyć wielu indeksów jednocześnie.

Składnia tej wskazówki jest następująca:

```
/*+ INDEX_COMBINE(tabela index [index..index])*/
```

Ostatnią wskazówką związaną z dostępem do danych przy wykorzystaniu indeksu jest NO_INDEX. Wskazówka ta wyklucza użycie podanych indeksów w planie wykonania danego zapytania. Indeksy, które nie zostaną wymienione we wskazówce są wykorzystywane.

Składnia wskazówki jest następująca:

```
/*+ NO_INDEX(tabela index [index..index])*/
```

Jeśli podana zostanie jedynie nazwa tabeli, nie będzie mógł być użyty żaden istniejący dla niej indeks.

Ostatnią wskazówką należącą do przedstawianej kategorii, należy USE_CONCAT.

Wymusza ona użycie przez optymalizator warunków OR w wyrażeniu WHERE i ich konwersję na operację zapytania UNION ALL.

Składnia omawianej wskazówki jest następująca:

```
/ *+ USE_CONCAT */
```

Przykład 16.7.4. Oto przykład zastosowania powyższej wskazówki.

```
SELECT /*+ USE_CONCAT */ nazwisko, plec
FROM osoby
WHERE id_os > 5 OR plec='K';
```

§ 16.8. Metody łączenia relacji

Można wyróżnić następujące sposoby łączenia relacji:

- **Nested – Loos**
- **Sort – merge**
- **Hash - join**

Metoda Nested - loops

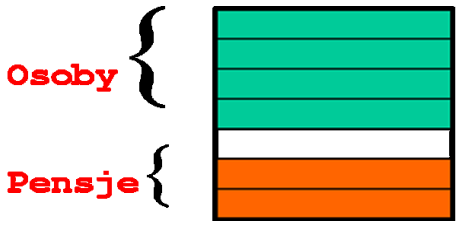
`Osoby`

ID	NAZWISKO	IMIE1	IMIE2	D_UR	PLEC
2	Kot	Adam	Marek	21/11/1980	M
3	Norek	Tadeusz		23/10/1982	M
4	Krawczyk	Adam		02/06/1982	M
1	Lis	Jan		21/10/1978	M

`Pensje`

ID	ID_OS	PENSJA	OD	DO
1	1	1100	01/03/1999	01/06/1999
3	3	1500	01/03/1999	31/05/2000
4	3	1800	01/06/2000	
2	2	1300	01/03/1999	

- Dla każdego wiersza z tabeli zewnętrznej (`Osoby`) znajdowane są wiersze z tabeli wewnętrznej (`Pensje`) spełniające warunek złączenia.
- Bloki tabeli zewnętrznej są wczytywane do RAM jednokrotnie a tabeli wewnętrznej wielokrotnie.



- Efektywność wykonania można zwiększyć stosując indeks do atrybutu złączeniowego w tabeli wewnętrznej.
- Metodę Nested – loops stosuje się do połączeń równościowych i nierównościowych.

Metoda Sort-merge

- W metodzie **sort-merge** pierwszym krokiem jest wstępne posortowanie łączonych tabel według atrybutów złączeniowych. Następnie są łączone wiersze spełniające warunek złączenia .

`Osoby`

ID	NAZWISKO	IMIE1	IMIE2	D_UR	PLEC
1	Lis	Jan		21/10/1978	M
2	Kot	Adam	Marek	21/11/1980	M
3	Norek	Tadeusz		23/10/1982	M
4	Krawczyk	Adam		02/06/1982	M

`Pensje`

ID	ID_OS	PENSJA	OD	DO
1	1	1100	01/03/1999	01/06/1999
2	2	1300	01/03/1999	
3	3	1500	01/03/1999	31/05/2000
4	3	1800	01/06/2000	

- System najczęściej stosuje metodę sort-merge gdy na atrybutach złączeniowych brak jest indeksów.

Metoda Hash-join

- W metodzie hash-join pierwszym krokiem jest budowa klastra haszowego używającego jako klucza atrybutów połączeniowych.

0	3 Norek Tadeusz 23/10/1982 M	3 3 1500 01/03/1999 31/05/2000 4 3 1800 01/06/2000
1	4 Krawczyk Adam 02/06/1982 M 1 Lis Jan 21/10/1978 M	1 1 1100 01/03/1999 01/06/1999
2	2 Kot Adam Marek 21/11/1980 M	2 2 1300 01/03/1999

Funkcja haszująca= *atrybut_połączeniowy* mod *3*

- System najczęściej stosuje metodę hash-join wyłącznie do połączeń równościowych.

Do określenia metody łączenia tabel służą wskazówki:

- **USE_NL - nested loops**
- **USE_MERGE - sort merge**
- **USE_HASH - hash merge**

Składnia wskazówki:

```
USE_metoda_połączenia(tabela [,tabela,…])
```

Przykład 16.8.1. Wymuszenie sposobu łączenia relacji.

```
SELECT /*+ USE_HASH(osoby o, zatrudnienia z)*/ o.nazwisko, z.pensja
FROM osoby o, zatrudnienia z
WHERE o.id_os = z.id_os AND z.do IS NULL;
```

Optymalizator kosztowy wybiera taką kolejność łączenia tabel, aby zminimalizować koszt tej operacji.

Do powyższej grupy zaliczamy następujące wskazówki:

- **ORDERED,**
- **STAR.**

Wskazówka `ORDERED` wymusza na optymalizatorze złączenie tabel w tej samej kolejności, w jakiej zostały wymienione w wyrażeniu `FROM`, od strony lewej do prawej.

Składnia tej wskazówki jest następująca:

```
/*+ ORDERED */
```

Zwykle jest zasadą to, że im więcej tabel podaje się w wyrażeniu `FROM`, tym większe korzyści daje zastosowanie tej wskazówki.

Wskazówka `STAR` wymusza umieszczenie największej tabeli jako ostatniej w kolejności złączania. Wskazówka ta ma zastosowanie tylko wówczas, gdy złączane są, co najmniej trzy tabele.

Składnia tej wskazówki jest następująca:

```
/*+ STAR */
```

17. ODCZYTYWANIE PLANU WYKONANIA POLECEŃ SQL

§ 17.1. Rola PLUSTRACE

Aby użytkownik mógł wyświetlać plan wykonania polecenia SQL musi posiadać uprawnienia zawarte w roli `PLUSTRACE`. Rolę tą można utworzyć uruchamiając skrypt `plustrce.sql` znajdującego się w katalogu `c:\oraclexe\...\sqlplus\admin`.

Należy to zrobić. z konta `sys as sysdba`.

Można to również zrobić następująco:

```
SQL> CREATE ROLE plustrace;
SQL> GRANT select ON v_$sesstat TO plustrace;
SQL> GRANT select ON v_$statname TO plustrace;
SQL> GRANT select ON v_$mystat TO plustrace;
SQL> GRANT plustrace TO dba WITH admin option;
```

§ 17.2. Plan wykonania poleceń DML

Plan wykonania może otrzymać tylko użytkownik posiadający przyznaną rolę `plustrace`.

Na koncie użytkownika posiadającego rolę `PLUSTRACE` należy poleceniem o składni

```
SQL> SET AUTOTRACE TRACE [EXPLAIN]
```

lub

```
SQL> SET AUTOTRACE ON [EXPLAIN]
```

włączyć wyświetlanie planu wykonywania poleceń SQL. Od tego momentu dla każdego polecenia `INSERT, UPDATE, DELETE` będzie wyświetlany plan wykonania.

W przypadku `ON` wyświetlany jest wynik zapytania i plan jego wykonania. W przypadku `TRACE` wyświetlany jest tylko plan wykonania. `EXPLAIN` powoduje, że wyświetlany jest tylko plan wykonania (bez statystyk).

Wyłączenie wyświetlania planów wykonania dokonuje się poleceniem

```
SQL> SET AUTOTRACE OFF
```

§ 17.3. Polecenie ANALYZE

Do zbierania statystyk w systemie Oracle wykorzystuje się polecenie `ANALYZE`.

Dzięki poleceniu `ANALYZE` można uzyskać dane statystyczne na temat systemu, obiektów znajdujących się w bazie danych, **które potrzebne są do wykonywania optymalizacji kosztowej**. Polecenie to zbiera informacje poprzez odczytywanie, niektórych lub też wszystkich danych w obiekcie oraz zachowuje informacje potrzebne do zrozumienia rozmieszczenia danych, takich jak największa i najmniejsza wartość w kolumnie, ilość wierszy w obiekcie oraz ilość bloków w obiekcie.

Sposób, w jaki zostanie wywołane polecenie zależy od tego, jaki rodzaj statystyki chcemy otrzymać. Polecenie może być wykorzystane w kilku trybach. Tryb ten zależy zarówno od rodzaju gromadzonych informacji, jak również od konfiguracji systemu.

Polecenie `ANALYZE` ma następującą składnie:

```
SQL> ANALYZE
```

```
      {
         TABLE [SCHEMAT.]nazwa_tabeli PARTITION(nazwa_partycji)]
        |INDEX [SCHEMAT.]nazwa_indeksu PARTITION(nazwa_partycji)]
        |CLUSTER [SCHEMAT.]nazwa_klastra
      }
      { COMPUTE STATISTICS
       |ESTIMATE STATISTICS [SAMPLE liczba {ROWS|PROCENT}]
       |DELETE STATISTICS
      };
```

gdzie:

TABLE,INDEX,CLUSTER	- Słowo kluczowe określające typ segmentu, który ma być analizowany.
schemat	- Parametr, który jest wymagany, jeśli segment należy do innego użytkownika.
nazwa_tabeli	- Nazwa przetwarzanej tabeli. Jest wymagana, jeśli w poleceniu zawarte jest słowo kluczowe TABLE.
nazwa_indeksu	- Nazwa przetwarzanego indeksu. Jest wymagana, jeśli w poleceniu, zawarte jest słowo kluczowe INDEX.
nazwa_klastra	- Nazwa przetwarzanego klastra. Jest wymagana, jeśli w poleceniu zawarte jest słowo kluczowe CLUSTER.
nazwa_partycji	- Parametr określający nazwę partycji, jeśli analizowana jest jedna partycja tabeli lub indeksu.
COMPUTE, ESTIMATE, DELETE	- Opcja określająca rodzaj wykonywanej operacji. Jedna z tych opcji jest wymagana, jeśli w poleceniu zawarte jest słowo kluczowe STATISTICS. COMPUTE wyznacza statystyki dokładne, ESTIMATE korzysta z próbki danych do wygenerowania statystyk, DELETE usuwa wszystkie wcześniej przeanalizowane statystyki.
SAMPLE liczba	- Opcja określająca rozmiar próbki używanej do oszacowania statystyk. Może być użyte tylko w połączeniu z opcją ESTIMATE. Jeśli klauzuli ESTIMATE użyje się bez opcji SAMPLE, rozmiar próbki zostanie określony domyślnie.
ROWS|PROCENT	- Parametr wskazujący czy wartość próbki powinna być traktowana jak liczba wierszy, czy jako procent rozmiaru tabeli. Może być użyte tylko w połączeniu z opcją ESTIMATE.

Polecenie ANALYZE umożliwia wyliczenie, oszacowanie lub usunięcie statystyk dla tabel, indeksów i klastrów. Przy wyliczaniu statystyk dla tabeli zostają również zebrane statystyki dla wszystkich indeksów skojarzonych z daną tabelą. Natomiast wyznaczając statystyki dla klastra zostają zbierane również statystyki dla wszystkich tabel wchodzących w skład tego klastra i wszystkich skojarzonych z tabelami indeksów. **Uprawnienia do wyznaczenia statystyk dla danego obiektu ma jego właściciel oraz inni użytkownicy posiadający uprawnienie systemowe ANALYZE ANY.**

Statystyki zbierane za pomocą polecenia ANALYZE są umieszczane w wewnętrznych tabelach Oracle. Mogą one być przeglądane poprzez perspektywy wydajnościowe. Każda perspektywa dostarcza w zasadzie tych samych informacji, różnice dotyczą zakresu prezentowanych danych.

Omawiane perspektywy poprzedzane są przedrostkami:

- **USER_** - Perspektywy te zawierają opis obiektów, których użytkownik jest właścicielem.
- **ALL_** - Perspektywy zawierające opis obiektów dostępnych dla danego użytkownika. Są to obiekty, do których użytkownik ma uprawnienia lub obiekty publiczne.
- **DBA_** - Perspektywy zawierające opis wszystkich obiektów występujących w systemie.

Perspektywy te dostarczają informacji o wielu rożnych obiektach bazy danych, takich jak: tabele, klastry, czy indeksy. Oto lista dostępnych perspektyw zawierających informację na temat wydajności systemu:

- Perspektywy zawierające opis tabel: USER_TABLES, ALL_TABLES, DBA_TABLES.

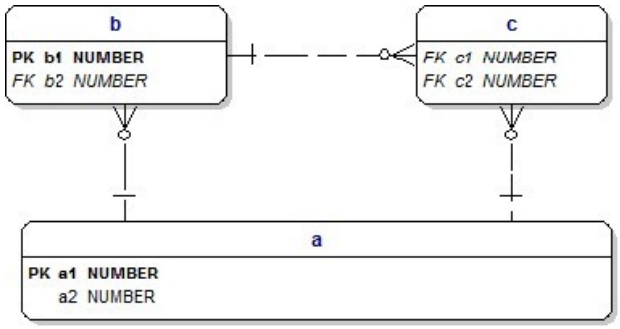
- Perspektywy zawierające opis indeksów: USER_INDEXES, ALL_INDEXES, DBA_INDEXES.
- Perspektywy zawierające opis klastrów: USER_CLUSTERS, ALL_CLUSTERS, DBA_CLUSTERS.

Informacje zawarte w wewnętrznych tabelach, obsługiwane przez te perspektywy, są wykorzystywane przez optymalizator kosztowy do wybrania optymalnego planu wykonania zapytania SQL.

Optymalizator podejmuje decyzje również na podstawie rozmiaru obiektów i zawartych w nim danych.

§ 17.4. Przykłady planów wykonania zapytań

Przykłady dotyczą zapytań do tabel o następującym diagramie encji:



Rysunek 17.4.1. Diagram encji do polecenia ANALYZE

Tabele utworzone zostały przy pomocy poleceń z **DODATKU A**.

```
SQL> SET AUTOTRACE trace explain
====KASOWANIE STATYSTYK ==================================================

SQL> ANALYZE TABLE a DELETE STATISTICS;
SQL> ANALYZE TABLE b DELETE STATISTICS;
SQL> ANALYZE TABLE c DELETE STATISTICS;
SQL> ANALYZE INDEX ind_a_a1 DELETE STATISTICS;
SQL> ANALYZE INDEX ind_b_b1 DELETE STATISTICS;
====PRZYKŁADOWE POLECENIA ================================================

SQL> SELECT * FROM a WHERE a1>60;
SQL> SELECT * FROM a WHERE a1=60;
SQL> SELECT * FROM a WHERE a1+0=60;
SQL> SELECT * FROM a WHERE a2=60;
SQL> SELECT a1 FROM a,b;
SQL> SELECT a2 FROM a,b WHERE a1=b2;
SQL> SELECT b1 FROM a,b WHERE a1=b2;
SQL> SELECT b1 FROM a,b WHERE a1=b1;
SQL> SELECT a1,b2 FROM a,b WHERE a2=b2;
SQL> SELECT a1,b2,c2 FROM a,b,c WHERE a2=b1 and b1=c1;
SQL> SELECT a1,b1,c2 FROM a,b,c WHERE a2=b2 and b2=c1;
SQL> SELECT a2,b2,c1 FROM a,b,c WHERE a1=c2 and b1=c2;
```

```
Plan wykonywania 1
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'A'
   2    1     INDEX (RANGE SCAN) OF 'IND_A_A1' (UNIQUE)
```

Przeszukiwanie liści indeksu IND_A_A1 i po znalezieniu odpowiednich wartości atrybutu a1 spełniających warunek WHERE (o ile istnieją) i ROWIDu rekordów znalezienie ich w tabeli a (bez przeszukiwania sekwencyjnego tabeli).

```
Plan wykonywania 2
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'A'
   2    1     INDEX (UNIQUE SCAN) OF 'IND_A_A1' (UNIQUE)
```

Wyszukanie w indeksie IND_A_A1 i po znalezieniu odpowiedniej wartości atrybutu a1 (o ile istnieje) i ROWIDu rekordu znalezienie go w tabeli a (bez przeszukiwania sekwencyjnego tabeli).

```
Plan wykonywania 3
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (FULL) OF 'A'
```

Przegląd sewencyjny tabeli a.

```
Plan wykonywania 4
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (FULL) OF 'A'
```

Przegląd sewencyjny tabeli a.

```
Plan wykonywania 5
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   NESTED LOOPS
   2    1     TABLE ACCESS (FULL) OF 'B'
   3    1     TABLE ACCESS (FULL) OF 'A'
```

Algorytm NESTED LOOPS – dla każdego rekordu z tabeli zewnętrznej b (sekwencyjnie) poszukiwane są rekordy pasujące w tabeli wewnętrznej a (sekwencyjnie) – nie obciąża procesora, ale dużo operacji wejścia-wyjścia.

```
Plan wykonywania 6
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   NESTED LOOPS
   2    1     TABLE ACCESS (FULL) OF 'B'
   3    1     TABLE ACCESS (BY INDEX ROWID) OF 'A'
   4    3       INDEX (UNIQUE SCAN) OF 'IND_A_A1' (UNIQUE)
```

Dla każdego wiersza z b poszukiwany jest w indeksie IND_A_A1 ROWIDu odpowiadającego wiersza w a, a następnie wybranie po ROWID wiersza z a.

```
Plan wykonywania 7
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   NESTED LOOPS
   2    1     TABLE ACCESS (FULL) OF 'B'
   3    1     INDEX (UNIQUE SCAN) OF 'IND_A_A1' (UNIQUE)
```

Dla każdego wiersza z b poszukiwana jest w indeksie IND_A_A1 odpowiadająca wartość.

```
Plan wykonywania 8
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   NESTED LOOPS
   2    1     TABLE ACCESS (FULL) OF 'B'
   3    1     INDEX (UNIQUE SCAN) OF 'IND_A_A1' (UNIQUE)
```

```
Plan wykonywania 9
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   MERGE JOIN
   2    1     SORT (JOIN)
   3    2       TABLE ACCESS (FULL) OF 'B'
   4    1     SORT (JOIN)
   5    4       TABLE ACCESS (FULL) OF 'A'
```

Algorytm SORT MERGE – sortuje obie tabele i dla każdego rekordu szuka sekwencyjnie odpowiedniego rekordu w drugiej – nie obciąża procesora, ale dużo operacji wejścia-wyjścia.

```
Plan wykonywania 10
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   MERGE JOIN
   2    1     SORT (JOIN)
   3    2       NESTED LOOPS
   4    3         TABLE ACCESS (FULL) OF 'C'
```

```
   5    3         TABLE ACCESS (BY INDEX ROWID) OF 'B'
   6    5           INDEX (UNIQUE SCAN) OF 'IND_B_B1' (UNIQUE)
   7    1     SORT (JOIN)
   8    7       TABLE ACCESS (FULL) OF 'A'
```

Tabele b i c łączone będą metodą NESTED LOOPS i wynik tego połączenia będzie łączony z tabelą a metodą SORT MERGE.

```
Plan wykonywania 11
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   MERGE JOIN
   2    1     MERGE JOIN
   3    2       SORT (JOIN)
   4    3         TABLE ACCESS (FULL) OF 'C'
   5    2       SORT (JOIN)
   6    5         TABLE ACCESS (FULL) OF 'B'
   7    1     SORT (JOIN)
   8    7       TABLE ACCESS (FULL) OF 'A'
```

```
Plan wykonywania 12
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   NESTED LOOPS
   2    1     NESTED LOOPS
   3    2       TABLE ACCESS (FULL) OF 'C'
   4    2       TABLE ACCESS (BY INDEX ROWID) OF 'B'
   5    4         INDEX (UNIQUE SCAN) OF 'IND_B_B1' (UNIQUE)
   6    1     TABLE ACCESS (BY INDEX ROWID) OF 'A'
   7    6       INDEX (UNIQUE SCAN) OF 'IND_A_A1' (UNIQUE)
```

====TWORZENIE STATYSTYK ==========

```
SQL> ANALYZE TABLE a COMPUTE STATISTICS;
SQL> ANALYZE TABLE b COMPUTE STATISTICS;
SQL> ANALYZE TABLE c COMPUTE STATISTICS;
SQL> ANALYZE INDEX ind_a_a1 COMPUTE STATISTICS;
SQL> ANALYZE INDEX ind_b_b1 COMPUTE STATISTICS;
```

Po zebraniu statystyk wykonane zostały ponownie polecenia SQL.

```
==============================================================================================
Plan wykonywania 1
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=96 Bytes=576)
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'A' (Cost=2 Card=96 Bytes=576)
   2    1     INDEX (RANGE SCAN) OF 'IND_A_A1' (UNIQUE) (Cost=1 Card=9 Bytes6)
```

```
Plan wykonywania 2
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=1 Card=1 Bytes=6)
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'A' (Cost=1 Card=1 Bytes=6)
   2    1     INDEX (UNIQUE SCAN) OF 'IND_A_A1' (UNIQUE)
```

```
Plan wykonywania 3
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=1 Bytes=6)
   1    0   TABLE ACCESS (FULL) OF 'A' (Cost=2 Card=1 Bytes=6)
```

```
Plan wykonywania 4
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=1 Bytes=6)
   1    0   TABLE ACCESS (FULL) OF 'A' (Cost=2 Card=1 Bytes=6)
```

```
Plan wykonywania 5
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=97 Card=17280 Bytes=51840)
   1    0   MERGE JOIN (CARTESIAN) (Cost=97 Card=17280 Bytes=51840)
   2    1     INDEX (FULL SCAN) OF 'IND_A_A1' (UNIQUE) (Cost=1 Card=96 Bytes=288)
   3    1     BUFFER (SORT) (Cost=96 Card=180)
   4    3       INDEX (FULL SCAN) OF 'IND_B_B1' (UNIQUE) (Cost=1 Card=180)
```

```
Plan wykonywania 6
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=5 Card=180 Bytes=1620)
   1    0   HASH JOIN (Cost=5 Card=180 Bytes=1620)
   2    1     TABLE ACCESS (FULL) OF 'A' (Cost=2 Card=96 Bytes=576)
   3    1     TABLE ACCESS (FULL) OF 'B' (Cost=2 Card=180 Bytes=540)
```

Algorytm HASH JOIN – stosuje algorytm hashowania na kolumnie a1 tabeli a i algorytm hashowania na kolumnie b1 tabeli b i następnie .dokonuje łączenia.

```
Plan wykonywania 7
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=180 Bytes=1620)
   1    0   NESTED LOOPS (Cost=2 Card=180 Bytes=1620)
   2    1     TABLE ACCESS (FULL) OF 'B' (Cost=2 Card=180 Bytes=1080)
   3    1     INDEX (UNIQUE SCAN) OF 'IND_A_A1' (UNIQUE)
```

```
Plan wykonywania 8
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=1 Card=96 Bytes=576)
   1    0   NESTED LOOPS (Cost=1 Card=96 Bytes=576)
   2    1     INDEX (FULL SCAN) OF 'IND_A_A1' (UNIQUE) (Cost=1 Card=96 Bytes=288)
   3    1     INDEX (UNIQUE SCAN) OF 'IND_B_B1' (UNIQUE)
```

```
Plan wykonywania 9
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=5 Card=180 Bytes=1620)
   1    0   HASH JOIN (Cost=5 Card=180 Bytes=1620)
   2    1     TABLE ACCESS (FULL) OF 'A' (Cost=2 Card=96 Bytes=576)
   3    1     TABLE ACCESS (FULL) OF 'B' (Cost=2 Card=180 Bytes=540)
```

```
Plan wykonywania 10
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=9 Card=3198 Bytes=57564)
   1    0   HASH JOIN (Cost=9 Card=3198 Bytes=57564)
   2    1     HASH JOIN (Cost=5 Card=96 Bytes=1152)
   3    2       TABLE ACCESS (FULL) OF 'A' (Cost=2 Card=96 Bytes=576)
   4    2       TABLE ACCESS (FULL) OF 'B' (Cost=2 Card=180 Bytes=1080)
   5    1     TABLE ACCESS (FULL) OF 'C' (Cost=3 Card=5996 Bytes=35976)
```

```
Plan wykonywania 11
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=9 Card=5996 Bytes=107928)
   1    0   HASH JOIN (Cost=9 Card=5996 Bytes=107928)
   2    1     HASH JOIN (Cost=5 Card=180 Bytes=2160)
   3    2       TABLE ACCESS (FULL) OF 'A' (Cost=2 Card=96 Bytes=576)
   4    2       TABLE ACCESS (FULL) OF 'B' (Cost=2 Card=180 Bytes=1080)
   5    1     TABLE ACCESS (FULL) OF 'C' (Cost=3 Card=5996 Bytes=35976)
```

```
Plan wykonywania 12
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=9 Card=5996 Bytes=107928)
   1    0   HASH JOIN (Cost=9 Card=5996 Bytes=107928)
   2    1     TABLE ACCESS (FULL) OF 'B' (Cost=2 Card=180 Bytes=1080)
   3    1     HASH JOIN (Cost=6 Card=5996 Bytes=71952)
   4    3       TABLE ACCESS (FULL) OF 'A' (Cost=2 Card=96 Bytes=576)
   5    3       TABLE ACCESS (FULL) OF 'C' (Cost=3 Card=5996 Bytes=35976)
```

§ 17.5. Rola administratora

- Tworzenie roli `plustrace`.
- Nadawanie uprawnień do odczytywania planu wykonania poleceń SQL.
- Robienie okresowo potrzebnych statystyk.
- Zalecenie projektantom użycia odpowiednich wskazówek w poleceniach SQL występujących w ich aplikacjach.

DODATEK A – TABELE DO OPTYMALIZACJI ZAPYTAŃ

```
DROP TABLE c;
DROP TABLE b;
DROP TABLE a;
DROP SEQUENCE seq;

CREATE TABLE a (a1 INTEGER, a2 INTEGER);
CREATE TABLE b (b1 INTEGER, b2 INTEGER);
CREATE TABLE c (c1 INTEGER, c2 INTEGER);

CREATE UNIQUE INDEX ind_a_a1 ON a(a1);
CREATE UNIQUE INDEX ind_b_b1 ON b(b1);

ALTER TABLE a ADD PRIMARY KEY (a1) USING INDEX ind_a_a1;
ALTER TABLE b ADD PRIMARY KEY (b1) USING INDEX ind_b_b1;

ALTER TABLE b ADD FOREIGN KEY (b2) REFERENCES a;
ALTER TABLE c ADD FOREIGN KEY (c1) REFERENCES b;
ALTER TABLE c ADD FOREIGN KEY (c2) REFERENCES a;

CREATE SEQUENCE seq START WITH 54;

INSERT INTO a VALUES (19, 31);
INSERT INTO a VALUES (11, 42);
INSERT INTO a VALUES (14, 34);
INSERT INTO a VALUES (23, 45);
INSERT INTO a VALUES (27, 39);
INSERT INTO a VALUES (21, 47);

INSERT INTO a SELECT a2, a1 FROM a;
INSERT INTO a SELECT a1 * 10, a2 * 3 - 4 FROM a;
INSERT INTO a SELECT a1 * 100, a2 * 5 - 41 FROM a;
INSERT INTO a SELECT a1 + 1, a2 * 3 - 5 FROM a;
INSERT INTO b SELECT seq.NEXTVAL, a1 FROM a;
INSERT INTO b SELECT seq.NEXTVAL * 2, a1 FROM a WHERE MOD(a1, 2) = 0;
INSERT INTO b SELECT seq.NEXTVAL * 3, a1 FROM a WHERE MOD(a1, 3) = 0;
INSERT INTO c SELECT b1, a1 FROM a, b WHERE a1 < b1;
COMMIT;
```

18. WYBRANE ZMIENNE PLIKU PARAMETRÓW

§ 18.1. Przykładowy plik init<sid>.ora

```
db_cache_size=591396864
java_pool_size=4194304
large_pool_size=4194304
shared_pool_size=201326592
sga_target=768M
streams_pool_size=0
audit_file_dest='C:\oraclexe\app\oracle\admin\XE\adump'
user_dump_dest='C:\oraclexe\app\oracle\admin\XE\udump'
background_dump_dest='C:\oraclexe\app\oracle\admin\XE\bdump'
core_dump_dest='C:\oraclexe\app\oracle\admin\XE\cdump'
audit_trail='DB'
compatible='10.2.0.1.0'
control_files='C:\oraclexe\oradata\XE\control.dbf'
db_name='XE'
DB_RECOVERY_FILE_DEST_SIZE=10G
DB_RECOVERY_FILE_DEST='C:\oraclexe\app\oracle\flash_recovery_area'
dispatchers='(PROTOCOL=TCP) (SERVICE=XEXDB)'
job_queue_processes=4
open_cursors=300
os_authent_prefix=''
pga_aggregate_target=256M
remote_login_passwordfile='EXCLUSIVE'
sessions=20
shared_servers=4
sql_trace=TRUE
timed_statistics=TRUE
undo_management='AUTO'
undo_tablespace='UNDO'
```

§ 18.2. Opis podstawowych parametrów pliku init.ora

Należy zdawać sobie sprawę z tego, że **pewne parametry można określić tylko jeden raz** – w momencie kiedy baza danych jest tworzona.

Im więcej rzeczy da się przewidzieć na samym początku, tym mniej problemów czysto technicznych i organizacyjnych trzeba będzie rozwiązywać w czasie użytkowania bazy danych.

NAZWA PARAMETRU	OPIS
DB_NAME	**Nazwa bazy danych.**
INSTANCE_NAME	**Nazwa instancji.**
DB_FILES	**Maksymalna liczba plików jaką może otworzyć baza danych.**
CONTROL_FILES	**Specyfikuje nazwy i położenie plików kontrolnych.**
COMPATIBLE	Numer wersji serwera z jaką powinna być kompatybilna ta instancja.
OPEN_CUROSRS	**Maksymalna liczba kursorów, które może jednocześnie otworzyć sesja**
DB_BLOCK_BUFFERS	Liczba buforów w SGA przydzielona dla bufora danych.

NAZWA PARAMETRU	OPIS
SHARED_POOL_SIZE	**Rozmiar współdzielonej puli w bajtach.**
LARGE_POOL_SIZE	Rozmiar w bajtach obszaru w SGA przechowującego duże obiekty niezwiązane z poleceniami SQL.
JAVA_POOL	Rozmiar w bajtach struktury w SGA przeznaczonej do przechowywania kodu Java.
LOG_CHECKPOINT_INTERVAL	Opóźnienie punktu kontrolnego (wyrażone w blokach systemu operacyjnego) względem końca pliku dziennika powtórzeń.
LOG_CHECKPOINT_TIMEOUT	Maksymalny odstęp czasowy pomiędzy punktami kontrolnymi (wyrażony w sekundach).
PROCESSES	**Maksymalna liczba procesów systemu operacyjnego podłączonych do instancji.**
LOG_BUFFER	Liczba buforów w SGA przydzielona dla bufora dziennika powtórzeń.
AUDIT_TRAIL	**Włącza lub wyłącza audyt bazy danych.Dopuszczalne wartości:** **NONE(FALSE) - audyt wyłączony** **DB(TRUE) - audyt włączony, wyniki będą zapisywane w bazie danych**
MAX_DUMP_FILE_SIZE	**Maksymalny rozmiar plików śladu wyrażony w blokach systemu operacyjnego.**
GLAOBAL_NAMES	**Sprawdza czy link bazy danych ma taką samą nazwę jak baza danych.** **TRUE – sprawdzanie włączone,** **FALSE – sprawdzanie wyłączone.**
USER_DUMP_DEST	**Lokalizacja plików śladu tworzonych przez procesy użytkownika.**
BACKGROUND_DUMP_DEST	**Lokalizacja plików śladu procesów drugoplanowych oraz lokalizacja plików alertów.**
DB_BLOCK_SIZE	Rozmiar bloku bazy danych – wyrażony w bajtach
REMOTE_LOGIN_PASSWORDFILE	**Uruchamia autoryzację poprzez plik haseł.** **NONE – brak autoryzacji poprzez plik haseł** **EXCLUSIVE – włączona autoryzacja poprzez plik haseł**
SESSIONS	**Maksymalna liczba jednorazowych sesji systemowych i użytkowników.**
TRANSACTION	**Maksymalna liczba jednoczesnych transakcji.**
SQL_TRACE	**Podawany w pliku parametrów powoduje włączenie zapisu do plików śladu informacji na temat realizowanych poleceń SQL. Informacje o błędach zapisywane są niezależnie od wartości tego parametru. TRUE/FALSE**
TIMED_STATISTICS	**Podawany w pliku parametrów powoduje włączenie generowania statystyki dotyczącej ilości czasu potrzebnego do wykonania poleceń.TRUE/FALSE**
LOG_ARCHIVE_START	**Automatyczne archiwizowanie.**
LOG_ARCHIVE_FORMAT	**Format nazwy plików archiwum.**
LOG_ARCHIVE_DEST	**Miejsce archiwizowania plików rejestru.**
UNDO_MANAGEMENT	**Metoda zarządzania wycofywaniem transakcji.**
HASH_JOIN_ENABLED	**Dostęp do metody Hash Join w optymalizatorze zapytań**
RESOURCE_LIMIT	**Włączenie kontroli limitów zdefiniowanych przez profile.**
OPTIMIZAL_MODE	**Parametr ten może przyjąć następujące wartości: RULE, CHOOSE, FIRST_ROWS, ALL_ROWS .**

19. WSPÓŁBIEŻNOŚĆ W SYSTEMIE ORACLE

Z pracą bazy danych wiąże się pojęcie **transakcji** (*transaction*). Wszystkie operacje wykonywane w Oracle odbywają się w trybie transakcyjnym.

Transakcja może być realizowana w jednym z trzech trybów.

1. Tryb **`READ COMMITTED`**,
2. Tryb **`READ ONLY`**,
3. Tryb **`SERIALIZABLE`**.

Z transakcjami wiąże się ściśle zjawisko **blokowania danych**.

Blokowanie danych ma na celu zapewnienie synchronizacji zapisów.

§ 19.1. Transakcyjny tryb pracy w systemie ORACLE

Własności transakcji w Systemie Oracle:

1. ***Atomowość*** - wszystkie operacje wykonywane w ramach transakcji muszą zakończyć się pomyślnie, niepowodzenie jednej z nich powoduje wycofanie całej transakcji.
2. ***Spójność*** - w wyniku realizacji transakcji otrzymujemy spójny stan bazy danych, w którym żadne z ograniczeń integralnościowych nie jest naruszone.
3. ***Izolacja*** - zmiany wprowadzone przez transakcję są widoczne dla innych użytkowników dopiero w momencie jej zatwierdzenia.
4. ***Trwałość*** - po zatwierdzeniu transakcji zmiany są zapisywane na trwałe do bazy.

Zatwierdzenie lub wycofanie aktualnej transakcji jest początkiem następnej.

Polecenia z grupy DDL (np. **`Create, Alter, Drop Table`**) oraz polecenia z grupy DCL (**`Grant, Revoke`**) kończą się niejawnym zatwierdzeniem transakcji.

§ 19.2. Tryby w jakich może pracować transakcja

Transakcje mogą być prowadzone w jednym z trzech trybów: **`Read Committed, Read Only`** lub **`Serializable`**.

Do ustawiania trybu pracy transakcji służy polecenie:

```
SQL> SET TRANSACTION
     {
        READ ONLY
      | ISOLATION LEVEL {SERIALIZABLE | READ COMMITTED}
     }
      NAME nazwa;
```

19.2.1. Tryb READ COMMITTED

1. Wszystkie transakcje w Systemie Oracle wykonywane są domyślnie w tym trybie.
2. Transakcja T_1 widzi dane zmodyfikowane przez transakcję T_2 dopiero po jej zatwierdzeniu poleceniem `COMMIT`.
3. Polecenie umożliwiające przestawienie pojedynczej transakcji w tryb `Read Commited`:

```
SET TRANSACTION ISOLATION LEVEL Read Committed;
```

Uwaga. Polecenie należy wykonać jako pierwsze w ramach transakcji.

Polecenie umożliwiające ustawienie trybu Read Commited dla wszystkich transakcji realizowanych w ramach danej sesji:

```
SQL> ALTER SESSION SET ISOLATION_LEVEL=Read Committed;
```

Uwaga. Polecenie należy wykonać jako pierwsze w ramach sesji.

19.2.2. Tryb READ ONLY

1. Transakcja T_1 operuje na wersji danych z momentu jej rozpoczęcia.
2. Transakcja w tym trybie nie może modyfikować danych.
3. Nie widzi zmian wprowadzonych w między czasie przez inne, zatwierdzone transakcje.

Tryb **`Read Only`** stosowany jest w przypadku obliczeń analitycznych.

Polecenie umożliwiające przestawienie pojedynczej transakcji w tryb **`Read Only`**:

```
SQL> SET TRANSACTION Read Only;
```

Uwaga. Polecenie należy wykonać jako pierwsze w ramach transakcji.

19.2.3. Tryb SERIALIZABLE

1. Transakcja w trybie `Serializable`, podobnie jak transakcja w trybie `Read Only`, operuje na wersji danych z momentu jej rozpoczęcia.
2. Różnica polega na tym, że można modyfikować dane, które nie zostały zmienione przez inne transakcje w trakcie jej trwania.

Polecenie umożliwiające przestawienie pojedynczej transakcji w tryb `Serializable`:

```
SQL> SET TRANSACTION ISOLATION LEVEL Serializable;
```

Uwaga. Polecenie należy wykonać jako pierwsze w ramach transakcji.

Polecenie umożliwiające ustawienie trybu **`Serializable`** dla wszystkich transakcji realizowanych w ramach danej sesji:

```
SQL> ALTER SESSION SET ISOLATION_LEVEL= Serializable;
```

Uwaga. Polecenie należy wykonać jako pierwsze w ramach sesji.

Tryb **`Serializable`** może być włączony na stałe poprzez ustawienie w pliku parametrów **`Init<SID>.ora`** parametru **`SERIALIZABLE=TRUE`**.

Domyślnie jego wartość wynosi **`FALSE`** co oznacza, że wszystkie transakcje są realizowane w trybie **`Read Committed`**;

§ 19.3. Mechanizm blokowania danych

W systemie Oracle stosowana jest metoda blokowania.

Blokady zakładane są na czas trwania transakcji.

Dwie blokady są ze sobą zgodne jeżeli mogą być założone na tę samą daną przez wiele transakcji.

Blokowanie może dotyczyć:

- **tabeli** (*table lock*),
- **rekordu** (*row lock*).

Blokowanie całej tabeli zmniejsza stopień współbieżności transakcji, ułatwiając zarządzanie blokadami i szybsze wykrywanie związanych z nimi konfliktów.

Blokowanie całej tabeli powoduje, że blokada dotyczy wszystkich jej rekordów a co za tym idzie system nie musi blokować każdego z nich oddzielnie.

Sposób zakładania blokad zależy od ustawienia parametrów:

```
SERIALIZABLE = False
ROW_LOCKING = Always
```

Są to ustawienia domyślne, przy których blokady są zakładane tylko w przypadku wykonywania poleceń modyfikujących dane (**DELETE, UPDATE, INSERT**).

Operacja **SELECT** nie wymaga nakładania blokady na tabeli i rekordzie.

Blokowanie rekordów odbywa się zawsze w trybie **EXCLUSIVE** (**X**).

Dwie blokady **X** nie są ze sobą zgodne.

Blokowanie tabeli odbywa się w trybie RS, RX, S, SRX oraz X gdzie:

- **RS** ROW SHARE
- **RX** ROW EXCLUSIVE
- **S** SHARE
- **SRX** SHARE ROW EXCLUSIVE
- **X** EXCLUSIVE

Zgodność blokad tabeli:

	Brak	RS	RX	S	SRX	X
Brak	**TAK**	**TAK**	**TAK**	**TAK**	**TAK**	**TAK**
RS	**TAK**	**TAK**	**TAK**	**TAK**	**TAK**	*Nie*
RX	**TAK**	**TAK**	**TAK**	*Nie*	*Nie*	*Nie*
S	**TAK**	**TAK**	*Nie*	**TAK**	*Nie*	*Nie*
SRX	**TAK**	**TAK**	*Nie*	*Nie*	*Nie*	*Nie*
X	**TAK**	*Nie*	*Nie*	*Nie*	*Nie*	*Nie*

19.3.1. Jawne założenie blokady na tabeli

Blokady mogą być nałożone na rekordy lub tabele w następujący sposób:

- **niejawny** - w momencie wykonywania operacji modyfikujących dane,
- **jawny** – nałożenie blokady następuje po wydaniu polecenia **LOCK TABLE**.

Do jawnego założenia blokady na tabeli służy następujące polecenie:

```
SQL> LOCK TABLE nazwa_tabeli IN tryb MODE;
```

gdzie:

`tryb` – oznacza rodzaj blokady, podany pełną nazwą.

19.3.2. Właściwości blokad

WŁAŚCIWOŚCI BLOKADY RS (ROW SHARE)

Zakładana jest w intencji późniejszego zmodyfikowania rekordów, uniemożliwia zmianę zawartości rekordów przez inne transakcje.

Jej założenie następuje automatycznie przy realizacji polecenia:

```
SQL> SELECT lista_atrybutów  FROM nazwa_tabeli
     WHERE warunek_selekcji FOR UPDATE [NOWAIT];
```

Użycie **NOWAIT** powoduje, że polecenie zostanie automatycznie przerwane, jeżeli nie można założyć blokady **RS** ze względu na istnienie innej blokady z nią niezgodnej.

WŁAŚCIWOŚCI BLOKADY RX (ROW EXCLUSIVE)

Zakładana jest automatycznie w przypadku realizacji poleceń **DELETE, INSERT, UPDATE**.

Pojawienie się blokady tego typu oznacza, że niektóre lub wszystkie rekordy tabeli zostały zmodyfikowane.

Modyfikowane rekordy są zawsze blokowane w trybie EXCLUSIVE (X).

WŁAŚCIWOŚCI BLOKADY S (SHARE)

Zakładana jest gdy transakcja T_1 chce uniemożliwić zmianę danych w tabeli przez inne równolegle działające transakcje i jednocześnie sama nie będzie ich modyfikowała.

Transakcje niezmieniające zawartości tabeli mogą współpracować z transakcją T_1.

WŁAŚCIWOŚCI BLOKADY SRX (SHARE ROW EXCLUSIVE)

Zakładana jest wtedy gdy transakcja T_1 będzie modyfikować zawartość tabeli i jednocześnie chce uniemożliwić zmianę danych w tej tabeli przez inne równolegle działające transakcje.

Inne transakcje wymagające, aby w trakcie ich pracy zawartość tabeli pozostała niezmieniona lub chcące ją zmodyfikować nie mogą współpracować z transakcją T_1.

WŁAŚCIWOŚCI BLOKADY X (EXCLUSIVE)

Uniemożliwia modyfikowanie danych dopuszczając tylko ich przeglądanie. Założenie innej blokady nie jest możliwe.

Uwaga. Wszystkie blokady zezwalają na wykonywanie operacji SELECT ...

§ 19.4. Informacja o założonych blokadach

Można ją uzyskać z widoku systemowego **V$LOCK** w połączeniu z danymi pomocniczymi pochodzącymi z widoków **V$SESSION** oraz **SYS.OBJ$.**

Przykład 19.4.1. Przykład składni **SELECT** wyświetlającej informacje o założonych blokadach:

```
SELECT s.username, l.type, o.name,
DECODE(L.LMODE,0,'BRAK',2,'RS',3,'RX',4,'S',5,'SRX',6,'X',L.LMODE)
                                                    AS TYP_BLOKADY,
DECODE(l.request,0,'brak',2,'rs',3,'rx',4,'s',5,'srx',6,'x',
                                L.REQUEST)AS TYP_BLOKADY_OCZEKUJACEJ,
s.sid, s.status, s.osuser, s.machine, s.program
FROM v$session s, v$lock l, sys.obj$ o
WHERE s.sid=l.sid AND l.id1=o.obj#(+);
```

W wyniku realizacji powyższej składni w kolumnie **typ_blokady** uzyskamy typ założonej blokady a w kolumnie **typ_blokady_oczekujacej** typ blokady oczekującej na założenie.

W kolumnie **TYPE** wartość **TM** oznacza blokadę tabeli natomiast **TX** blokadę rekordu.

Pozostałe kolumny to:

USERNAME	- nazwa użytkownika bazy danych
NAME	- nazwa obiektu którego dotyczą blokady
SID	- numer sesji
STATUS	- status sesji użytkownika
OSUSER	- nazwa użytkownika w systemie operacyjnym
MACHINE	- nazwa komputera z którego użytkownik dołączył się do bazy danych
PROGRAM	- nazwa programu za pomocą którego użytkownik dołączył się do bazy

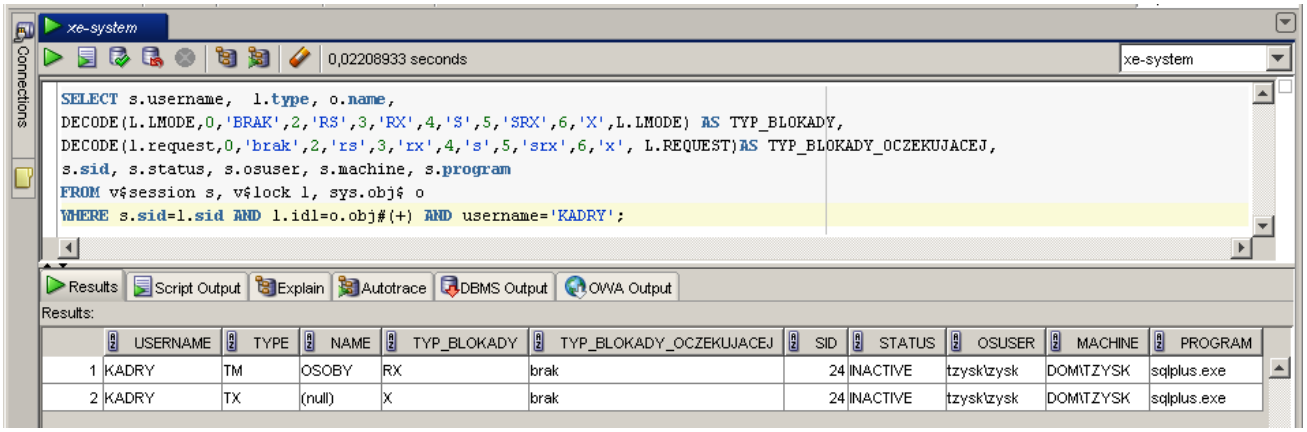
Przykład 19.4.2. Przykład o założonych blokadach.

Pierwsza sesja:

```
SQL> CONNECT kadry/kadry@xe
SQL> SET TRANSACTION NAME 't1';
SQL> SELECT * FROM osoby FOR UPDATE;
```

Sesja administratora (wystarczy `CONNECT system/test@xe`):

```
SQL> select z przykładu 19.4.1
```

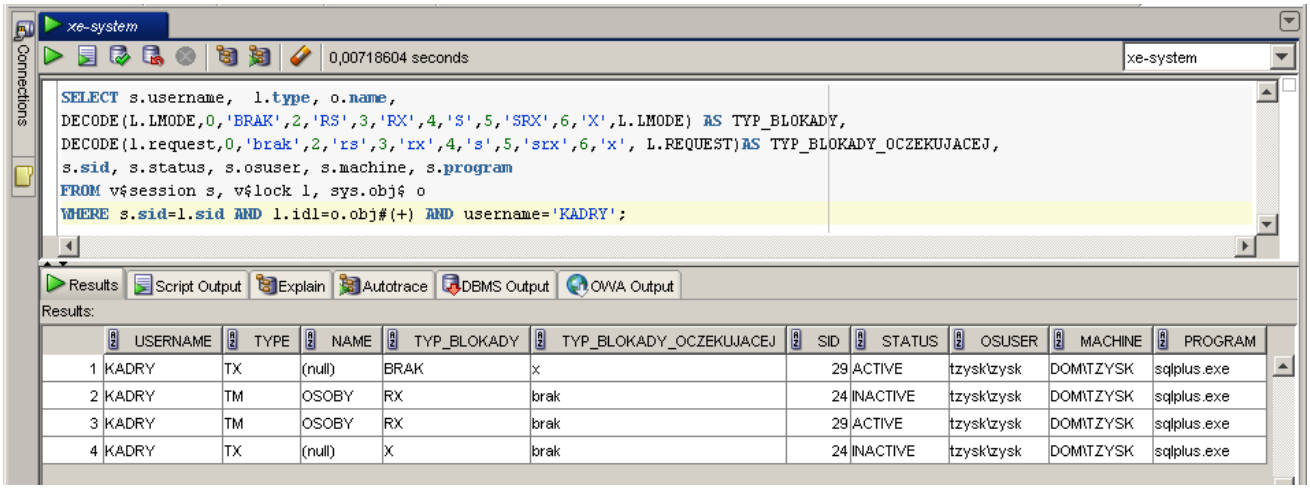


Druga sesja:

```
SQL> CONNECT kadry/kadry@xe
SQL> SET TRANSACTION NAME 't2';
SQL> SELECT * FROM osoby FOR UPDATE;
SQL> ...
```

Sesja administratora (wystarczy `CONNECT system/test@xe`):

```
SQL> select z przykładu 19.4.1
```



§ 19.5. Zakleszczenia (deadlocks)

Zaletą metody blokowania danych jest zapewnienie synchronizacji zapisu w przypadku wielu transakcji próbujących modyfikować te same dane.

Metoda ta posiada jednak dwie wady:

1. zmniejsza stopień współbieżności (transakcja, która próbuje założyć blokady niezgodne z blokadami już założonymi przez inną transakcję, musi czekać na zdjęcie blokad);
2. wprowadza możliwość wystąpienia **zakleszczenia** (*deadlock*), kiedy dwie transakcje blokują sobie wzajemnie zasoby. Wówczas żadna z transakcji nie może kontynuować pracy.

System Oracle wykrywa zakleszczenie i rozwiązuje je wykorzystując pewien algorytm wyboru tej transakcji, która zostanie przerwana, tj. jej ostatnie polecenie zostanie przerwane, wycofane.

Przykład 19.5.1. Przykład wystąpienia zakleszczenia.

	Pierwsza sesja:	**Druga sesja:**
1.	`CONNECT kadry/kadry@xe`	
2.	`SET TRANSACTION NAME 't1';`	
3.		`CONNECT kadry/kadry@xe`
4.		`SET TRANSACTION NAME 'T2';`
5.	`SELECT * FROM osoby FOR UPDATE;`	
6.		`SELECT * FROM wydzialy FOR UPDATE;`
7.	`SELECT * FROM wydzialy FOR UPDATE;`	
8.		`SELECT * FROM osoby FOR UPDATE;`

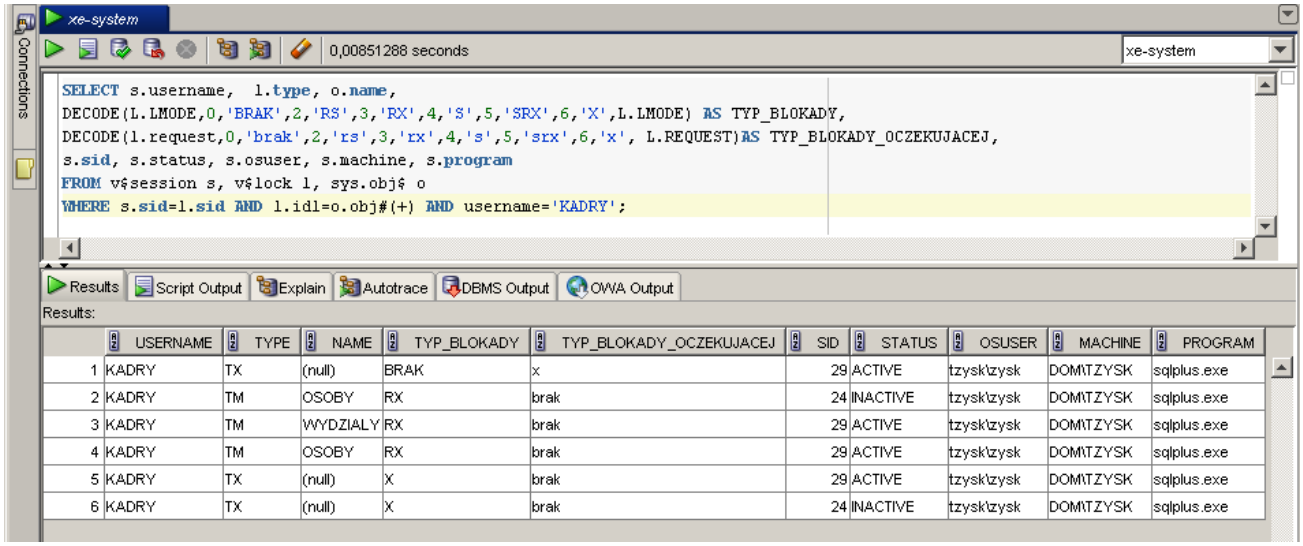
Właściciel transakcji, dla której nastąpiło zakleszczenie otrzymuje wówczas komunikat:

```
ORA-.....: deadlock detected while waiting for resource
```

Algorytm wyboru transakcji do przerwania nie został wyspecyfikowany w dokumentacji Oracle.

Sesja administratora:

```
SQL> CONNECT system/test@xe
SQL> select z przykładu 19.4.1
```



```
SELECT s.username,  l.type, o.name,
DECODE(L.LMODE,0,'BRAK',2,'RS',3,'RX',4,'S',5,'SRX',6,'X',L.LMODE) AS TYP_BLOKADY,
DECODE(l.request,0,'brak',2,'rs',3,'rx',4,'s',5,'srx',6,'x', L.REQUEST)AS TYP_BLOKADY_OCZEKUJACEJ,
s.sid, s.status, s.osuser, s.machine, s.program
FROM v$session s, v$lock l, sys.obj$ o
WHERE s.sid=l.sid AND l.idl=o.obj#(+) AND username='KADRY';
```

Results:

	USERNAME	TYPE	NAME	TYP_BLOKADY	TYP_BLOKADY_OCZEKUJACEJ	SID	STATUS	OSUSER	MACHINE	PROGRAM
1	KADRY	TX	(null)	BRAK	x	29	ACTIVE	tzysk\zysk	DOM\TZYSK	sqlplus.exe
2	KADRY	TM	OSOBY	RX	brak	24	INACTIVE	tzysk\zysk	DOM\TZYSK	sqlplus.exe
3	KADRY	TM	WYDZIALY	RX	brak	29	ACTIVE	tzysk\zysk	DOM\TZYSK	sqlplus.exe
4	KADRY	TM	OSOBY	RX	brak	29	ACTIVE	tzysk\zysk	DOM\TZYSK	sqlplus.exe
5	KADRY	TX	(null)	X	brak	29	ACTIVE	tzysk\zysk	DOM\TZYSK	sqlplus.exe
6	KADRY	TX	(null)	X	brak	24	INACTIVE	tzysk\zysk	DOM\TZYSK	sqlplus.exe

§ 19.6. Rola administratora

- Monitorowanie transakcji wszystkich użytkowników.
- Sugerowanie zmian w transakcjach powodujących spadek wydajności systemu.
- Tryb `Serializable` może być włączony na stałe poprzez ustawienie w pliku parametrów `Init<SID>.Ora` parametru **`SERIALIZABLE=TRUE`**.
- Domyślnie jego wartość wynosi **`FALSE`** co oznacza, że wszystkie transakcje są realizowane w trybie **`Read Committed`**.

20. TWORZENIE NOWEJ BAZY

Do tworzenia nowej instancji można użyć programu `Oradim`.

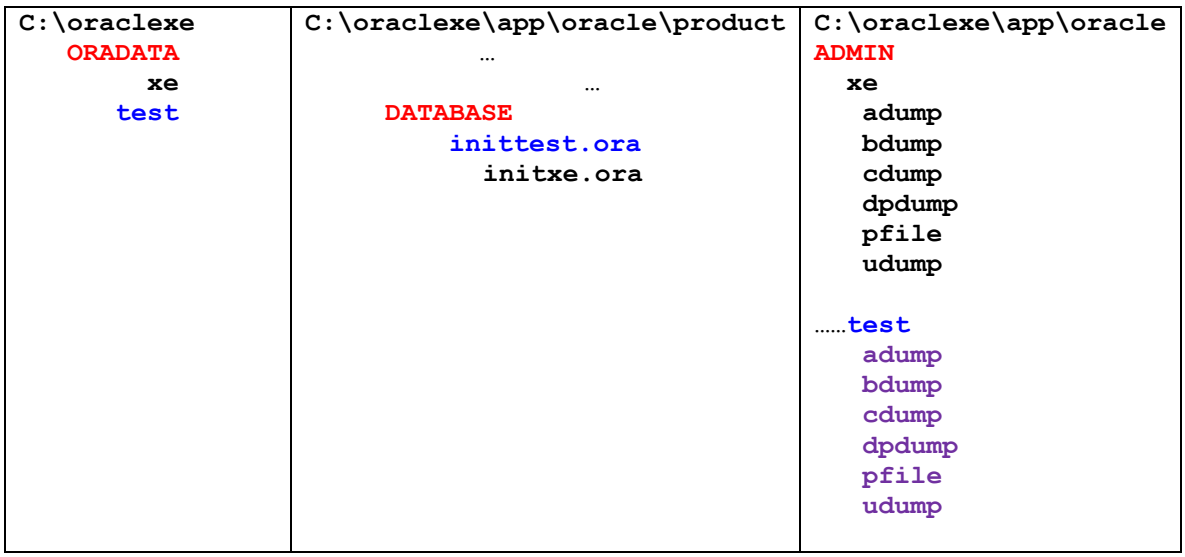
```
c:\oradim -NEW -SID sid | -SRVC service_name [-SYSPWD password]
  [-STARTMODE auto | manual][-PFILE filename]
```

gdzie

• `sid`	• - jest nazwą nowej instancji.
• `service_name`	• - jest nazwą serwisu nowej instancji (`OracleServiceSID`).
• `password`	• - jest hasłem dla użytkownika logującego się z uprawnieniami SYSDBA.
• `auto, manual`	• - oznacza sposób startu instancji. Domyślnie jest `manual`.
• `filename`	• - jest nazwą pliku z parametrami instancji.

§ 20.1. Przygotowanie katalogów dla nowej bazy

Należy przygotować następujące katalogi i pliki:



Rysunek 20.1.1. Struktua katalogów nowej bazy.

W pliku `listener.ora` dodać wpis:

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = C:\oraclexe\app\oracle\product\10.2.0\server)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (SID_NAME = CLRExtProc)
      (ORACLE_HOME = C:\oraclexe\app\oracle\product\10.2.0\server)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (GLOBAL_DBNAME = test)
      (ORACLE_HOME = C:\oraclexe\app\oracle\product\10.2.0\server)
      (SID_NAME = test)
     )


  )

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC_FOR_XE))
```

```
      (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    )

  )

DEFAULT_SERVICE_LISTENER = (XE)
```

Rysunek 20.1.2. Wpis w pliku `listener.ora` dla nowej bazy.

W pliku `tnsname.ora` dodać wpis:

```
test =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = test)
    )
  )
```

§ 20.2. Tworzenie serwisu dla nowej instancji

Utworzenie serwisu dla instancji o nazwie **`test`**.

```
c:\> oradim -NEW -SRVC oracleservicetest -SYSPWD test
  -STARTMODE auto -PFILE c:\oraclexe\admin\test\pfile\inittest.ora
```

§ 20.3. Tworzenie instancji dla nowej bazy

Utworzenie instancji o nazwie `test`.

```
c:\> oradim -NEW -SID test -SYSPWD test -STARTMODE auto
     -PFILE c:\oraclexe\admin\test\pfile\inittest.ora
```

gdzie **`inittest.ora`** jest np. postaci:

```
db_cache_size=167772160
java_pool_size=4194304
large_pool_size=25165824
shared_pool_size=83886080
streams_pool_size=0
control_files=('c:\oraclexe\oradata\test\control01.ctl',
               'c:\oraclexe\oradata\test\control02.ctl',
               'c:\oraclexe\oradata\test\control03.ctl')
audit_file_dest='c:\oraclexe\app\oracle\admin\test\adump'
user_dump_dest='c:\oraclexe\app\oracle\admin\test\udump'
background_dump_dest='c:\oraclexe\app\oracle\admin\test\bdump'
core_dump_dest='c:\oraclexe\app\oracle\admin\test\cdump'
audit_trail='TRUE'
compatible='10.2.0.1.0'
db_name='TEST'
DB_RECOVERY_FILE_DEST_SIZE=12G
DB_RECOVERY_FILE_DEST='c:\oraclexe\app\oracle\flash_recovery_area'
dispatchers='(PROTOCOL=TCP) (SERVICE=TESTXDB)'
job_queue_processes=4
open_cursors=300
os_authent_prefix=''
pga_aggregate_target=90M
remote_login_passwordfile='EXCLUSIVE'
sessions=50
sga_target=270M
shared_servers=4
undo_management='AUTO'
undo_tablespace='UNDO'
```

Rysunek 20.3.1. Plik `inittest.ora` nowej bazy.

§ 20.4. Tworzenie nowej bazy danych

```
SQL> CONNECT sys@test AS SYSDBA
```

```
SQL> STARTUP NOMOUNT
```

```
SQL> CREATE DATABASE test
     CONTROLFILE REUSE
     LOGFILE
        GROUP 1 ('c:\oraclexe\oradata\test\log1.log') SIZE 50M,
        GROUP 2 ('c:\oraclexe\oradata\test\log2.log') SIZE 50M
     MAXLOGFILES 5
     MAXLOGHISTORY 100
     MAXDATAFILES 10
     MAXINSTANCES 2
     NOARCHIVELOG
     CHARACTER SET EE8MSWIN1250
     NATIONAL CHARACTER SET AL16UTF16
     DATAFILE
       'c:\oraclexe\oradata\test\system.dbf'
        SIZE 50M AUTOEXTEND ON NEXT 100k MAXSIZE UNLIMITED
     DEFAULT TEMPORARY TABLESPACE temp
       TEMPFILE 'c:\oraclexe\oradata\test\temp.dbf'
       SIZE 100M REUSE AUTOEXTEND ON NEXT 512K MAXSIZE UNLIMITED
     UNDO TABLESPACE undo
       DATAFILE 'c:\oraclexe\oradata\test\undo.dbf'
       SIZE 100M REUSE AUTOEXTEND ON NEXT 512K MAXSIZE UNLIMITED
     SYSAUX
       DATAFILE 'c:\oraclexe\oradata\test\sysaux.dbf'
       SIZE 200M REUSE AUTOEXTEND ON NEXT 512K MAXSIZE UNLIMITED;
```

Można również stworzyć dodatkowe przestrzenie tabel.

```
SQL> CREATE TABLESPACE users
     DATAFILE 'c:\oraclexe\oradata\test\users01.dbf'
     SIZE 25M REUSE AUTOEXTEND ON NEXT  128K MAXSIZE 500M
```

```
SQL> CREATE TABLESPACE indx
     DATAFILE 'c:\oraclexe\oradata\test\indx01.dbf'
     SIZE 25M REUSE AUTOEXTEND ON NEXT 128K MAXSIZE UNLIMITED
```

§ 20.5. Tworzenie słownika danych

Uruchomić SQL*Plus w trybie **NOLOG**:

```
c:\sqlplus.exe /nolog
```

oraz

```
SQL> CONNECT sys@test AS SYSDBA
```

```
SQL> STARTUP
```

```
SQL>@c:\oraclexe\app\oracle\product\10.2.0\server\
                                        rdbms\admin\catalog.sql
```

```
SQL>@c:\oraclexe\app\oracle\product\10.2.0\server\
                                        rdbms\admin\catproc.sql
```

i ewentualnie (potrzebne do exportu i importu przy pomocy programów `Exp` i `Imp`)

```
SQL>@c:\oraclexe\app\oracle\product\10.2.0\server\
```

```
                                        rdbms\admin\catexp.sql
```

§ 20.6. Plik spfile<sid>.ora

Poleceniem

```
SQL> CREATE SPFILE='c:\oraclexe\app\oracle\product\10.2.0\
                                        server \dbs\spfiletest.ora'
     FROM PFILE='c:\oraclexe\app\oracle\product\10.2.0\
                                      server \database\inittest.ora';
```

można utworzyć nowy plik **spfiletest.ora** i restartować instancję.

W czasie pracy instancji zmiana parametrów systemu może być zmieniona w pliku **spfile<sid>.ora.**

21. WYZWALACZE SYSTEMOWE, INSTEAD OF I TRANSAKCJE AUTONOMICZNE

§ 21.1. Wyzwalacze `Instead Of`

Wyzwalacze `INSTEAD OF` (zamiast) umożliwiają np. modyfikowanie perspektyw (widoków), których zawartość nie można zmieniać używając poleceń DML (np. perspektyw utworzonych z klauzulą `GROUP BY`).

```
CREATE OR REPLACE TRIGGER nazwa_wyzwalacza
INSTEAD OF INSERT ON perspektywa
DECLARE
  ...
BEGIN
  ...
END;
```

Przykład 21.1.1. Przykład zastosowania wyzwalacza typu `INSTEAD OF`.

Mamy perspektywę `pracownicy` utworzoną poleceniem:

```
SQL> CREATE VIEW pracownicy
     AS
     SELECT o.id_os, o.nazwisko, o.imie1, z.od, z.id_w, z.id_s,
z.penja
     FROM osoby o LEFT JOIN zatrudnienia z ON o.id_os=z.id_os
     WHERE z.do IS NULL;
```

```
SQL> CREATE SEQUENCE z_licznik
     INCREMENT BY 1
     START WITH 60;
```

```
SQL> CREATE OR REPLACE TRIGGER nowa_pensja
      INSTEAD OF INSERT ON pracownicy
      BEGIN
        INSERT INTO zatrudnienia
        VALUES    (z_licznik.nextval,:new.id_os,    sysdate,    null,
:new.id_w,
                  :NEW.pensja, :NEW.id_s);
      END;
```

```
SQL>   INSERT   INTO   pracownicy   VALUES   (30,'Got','Danuta',
NULL,1,1,2000);
```

§ 21.2. Wyzwalacze systemowe

Wyzwalacze systemowe mogą być definiowane na poziomie **bazy danych** lub na poziomie **schematu użytkownika**.

Przykłady zdarzeń systemowych:

`STARTUP, AFTER LOGON, AFTER DROP, AFTER GRANT, AFTER CREATE,…`

Przykład 21.2.1. Przykład rejestracji logowania.

```
SQL> CREATE TABLE rejestracja_logowania
     (
```

```
       uzytkownik VARCHAR2(10) NOT NULL,
       data DATE NOT NULL
      );

SQL> CREATE OR REPLACE TRIGGER rejestracja_logowania
     AFTER LOGON
     ON SCHEMA
     BEGIN
       INSERT INTO rejestracja_logowania VALUES (user,SYSDATE);
     END;
```

Przykład 21.2.2. Przykład rejestracji startowania bazy (`startup`).

```
SQL> CREATE TABLE rejestracja_start (data DATE NOT NULL);


SQL> CREATE OR REPLACE TRIGGER rejestracja_start
     AFTER STARTUP
     ON DATABASE
     BEGIN
       INSERT INTO rejestracja_start VALUES (SYSDATE);
     END;
```

§ 21.3. Wyzwalacze a transakcje autonomiczne

W przypadku, gdy wyzwalacz jest częścią transakcji i chcemy, aby wewnątrz nastąpiło zatwierdzenie (`COMMIT`) dokonanych wewnątrz zmian bez względy na to czy cała transakcja będzie zatwierdzona czy też nie należy w części deklaracyjnej użyć

`PRAGMA AUTONOMOUS_TRANSACTION.`

Przykład 21.3.1. Przykład rejestracji prób zmian w tabeli `sprzedaz`.

```
SQL> CREATE TABLE rejestracja
     ( id_towaru NUMBER NOT NULL,
       ilosc NUMBER,
       data DATE NOT NULL);

SQL> CREATE OR REPLACE TRIGGER rejestracja
     BEFORE INSERT ON sprzedaz
     FOR EACH ROW
     DECLARE
       PRAGMA AUTONOMOUS_TRANSACTION;
     BEGIN
        INSERT INTO rejestracja
                VALUES (:NEW.id_towaru, :NEW.ilosc, SYSDATE);
        COMMIT;
    END;
```

22. TRYB ARCHIVELOG I TRYB NOARCHIVELOG

§ 22.1. Wstęp

Przykład polecenia tworzącego bazę `xxxx` (w trybie `NOMOUNT`):

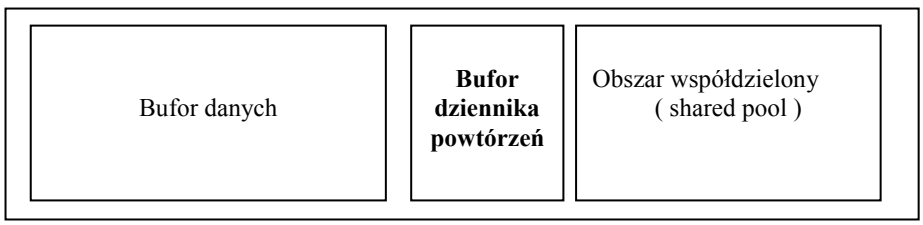
```
CREATE DATABASE xxxx
 CONTROLFILE REUSE
 LOGFILE
  GROUP 1 ('c:/oraclexe/oradata/xxxx/redo11.log',
           'c:/oraclexe/oradata/xxxx/redo12.log') SIZE 30M,
  GROUP 2 ('c:/oraclexe/oradata/xxxx/redo21.log',
           'c:/oraclexe/oradata/xxxx/redo22.log') SIZE 30M,
  GROUP 3 ('c:/oraclexe/oradata/xxxx/redo31.log',
           'c:/oraclexe/oradata/xxxx/redo32.log') SIZE 30M
 MAXLOGFILES 32
 MAXDATAFILES 15
 ARCHIVELOG
 CHARACTER SET EE8MSWIN1250
 NATIONAL CHARACTER SET AL16UTF16
 DATAFILE 'c:/oraclexe/oradata/xxxx/system01.dbf' SIZE 200M
                                                  AUTOEXTEND ON
 DEFAULT TEMPORARY TABLESPACE tempts1 TEMPFILE
             'c:/oraclexe/oradata/xxxx/temp01.dbf' SIZE 200M  REUSE
 UNDO TABLESPACE undo DATAFILE
  'c:/oraclexe/oradata/xxxx/undo01.dbf' SIZE 200M REUSE AUTOEXTEND
                                  ON NEXT 512K MAXSIZE UNLIMITED
 SYSAUX
      DATAFILE 'C:\baza\oradata\sysaux01.dbf'
      SIZE 200M REUSE AUTOEXTEND ON NEXT 5120K MAXSIZE UNLIMITED;;
```

Po utworzeniu bazy danych powstaną następujące pliki:

Pliki dziennika powtórzeń:	Pliki kontrolne:	Pliki danych:
Redo11.log	Controlfile01.ctl	System01.dbf
Redo12.log	Controlfile02.ctl	Undo01.dbf
Redo21.log	Controlfile03.ctl	Temp01.dbf
Redo22.log		Sysaux01.dbf
Redo31.log		…
Redo32.log		

Należy jeszcze uruchomić pliki `catalog.sql` i `catproc.sql` w trypie `OPEN`.

W momencie startu instancji tworzony jest w pamięci RAM obszar **SGA** (System Global Area).

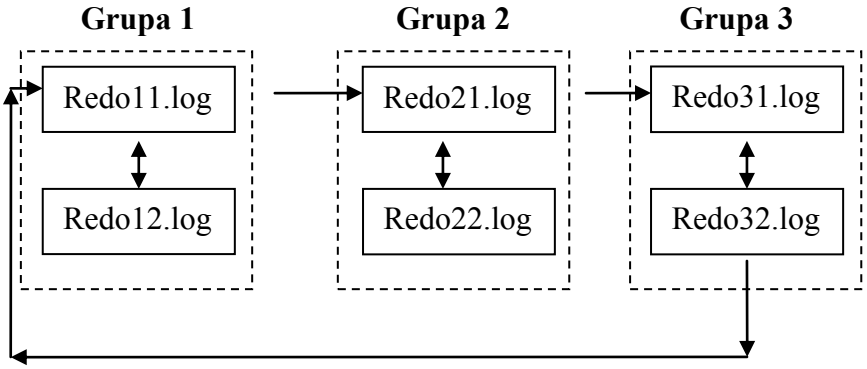


Rysunek 22.1.1. Obszar SGA.

Zapisywaniem wszystkich zatwierdzonych zmian na bazie danych do plików dziennika powtórzeń zajmuje się proces `LGWR` (`Log Writer`). Zapis jest cykliczny i można go zilustrować następująco:

1. Baza danych w trybie bez archiwizacji (`NOARCHIVELOG`)

Zatwierdzone zmiany w bazie zapisywane są cyklicznie do plików REDO.

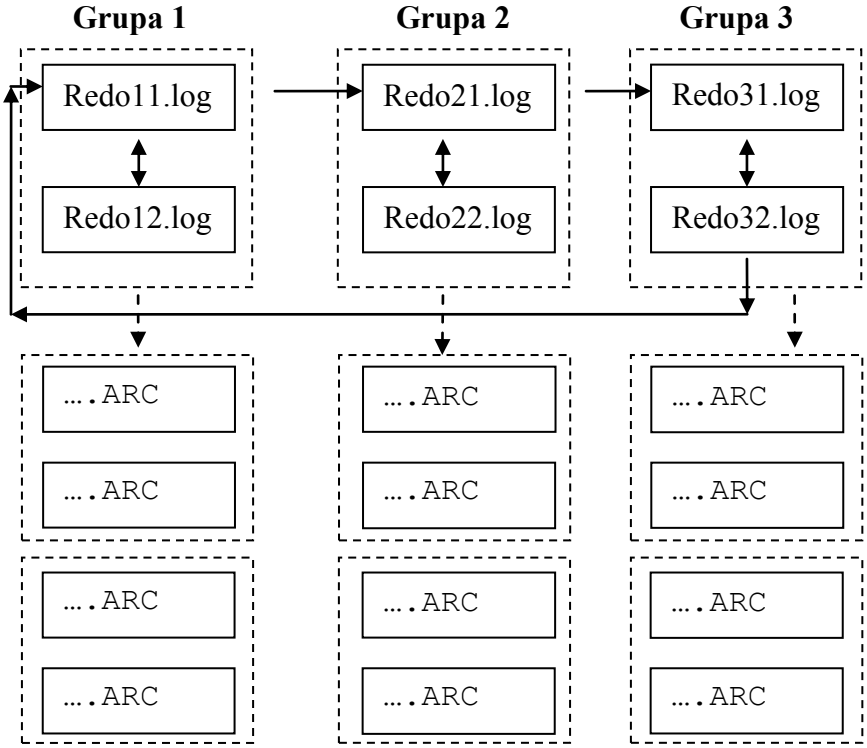


Rysunek 22.1.2. Pliki Redo w trybie NOARCHIVELOG.

W przypadku zapełnienia wszystkich plików następuje nadpisywanie i w ten sposób można ewentualnie powrócić do danych do określonego momentu (są ograniczenia wynikające z nadpisywania).

2. Baza danych w trybie z archiwizacją (ARCHIVELOG)

Zatwierdzone zmiany w bazie zapisywane są cyklicznie do plików REDO.



Rysunek 22.1.3. Pliki Redo w trybie ARCHIVELOG.

W przypadku zapełnienia wszystkich plików następuje kopiowanie plików i dopiero następuje nadpisywanie i w ten sposób można ewentualnie powrócić do danych do dowolnego momentu.

§ 22.2. Dodawanie plików dziennika powtórzeń w czasie pracy bazy

Dodawanie grupy:

```
SQL> ALTER DATABASE
     ADD LOGFILE GROUP numer_grupy
             ('nazwa_pliku', 'nazwa_pliku'…..) SIZE rozmiar;
```

Np.

```
SQL> ALTER DATABASE ADD LOGFILE GROUP 4
                ('c:/oraclexe/oradata/xxxx/redo41.log',
                 'c:/oraclexe/oradata/xxxx/redo42.log') SIZE 30M;
```

Wszystkie pliki w grupie mają ten sam rozmiar.

Można dodawać pliki do grupy poleceniem:

```
SQL> ALTER DATABASE
 ADD LOGFILE ('nazwa_pliku', 'nazwa_pliku'….) TO GROUP numer_grupy;
```

Np.

```
SQL> ALTER DATABASE
    ADD LOGFILE ('c:/oraclexe/oradata/xxxx/redo43.log') TO GROUP 4;
```

Dodany plik do istniejącej grupy ma rozmiar taki sam jak pliki już istniejące.

§ 22.3. Archiwizowanie plików dziennika powtórzeń

Informacje o trybie pracy bazy można uzyskać poleceniem

```
SQL> ARCHIVE LOG LIST
    Tryb dziennika bazy danych           Tryb archiwizacji
    Automatyczna archiwizacja                   Wyłączona
    Miejsca archiwizowania               c:\oraclexe\archiwum
    ...
```

lub w perspektywie systemowej `V$DATABASE`.

22.3.1. Archiwizacja automatyczna

W pliku `initxxxx.ora/spfilexxxx.ora` musi znajdować się wpis:

```
LOG_ARCHIVE_START=TRUE
```

lub w czasie, kiedy instancja jest otwarta:

```
SQL> CONNECT system@xxxx AS SYSDBA

SQL> ALTER SYSTEM ARCHIVE LOG START;
```

lub

```
SQL> ALTER SYSTEM ARCHIVE LOG START TO 'c:\oraclexe\archiwum';
```

Archiwizowane pliki REDO będą zapisywane do katalogu `c:\oraclexe\archiwum`.

22.3.2. Wyłączenie automatycznej archiwizacji

```
SQL> CONNECT system@xxxx AS SYSDBA
SQL> SHUTDOWN
SQL> STARTUP MOUNT
SQL> ALTER DATABASE NOARCHIVELOG;
SQL> SHUTDOWN
```

Zmiana w pliku `initxxxx.ora` na `LOG_ARCHIVE_START=FALSE`

```
SQL> STARTUP
```

22.3.3. Archiwizowanie dziennika powtórzeń przez administratora

Instancja musi działać w trybie ARCHIVELOG, co można uzyskać następującymi poleceniami:

```
SQL> CONNECT system@xxxx AS SYSDBA
SQL> SHUTDOWN
SQL> STARTUP MOUNT
SQL> ALTER DATABASE ARCHIVELOG;
SQL> ALTER DATABASE OPEN;
```

Postać polecenia:

```
SQL> ALTER SYSTEM ARCHIVE LOG {ALL | LOGFILE 'nazwa_pliku'
     | CURRENT | GROUP numer}
     [TO katalog];
```

23. ARCHIWIZACJA BAZY DANYCH

§ 23.1. Podstawowe reguły sporządzania kopii bezpieczeństwa

Przykładowe zalecenia:

- Archiwizowanie plików dziennika powtórzeń najpierw na dysk, a potem na inny nośnik.
- Zarchiwizowane pliki dziennika powtórzeń powinno się przechowywać na innym dysku niż aktywne (online) pliki dziennika.
- Wykonywanie kopii plików bazy danych najpierw na dysk, a dopiero potem na inny nośnik.
- Utrzymywanie wielu kopii pliku kontrolnego, z których każda powinna być umieszczona na odrębnym dysku.
- Po każdorazowej zmianie struktury bazy danych, tj. dodaniu, zmianie bądź usunięciu pliku danych lub dziennika powtórzeń należy sporządzić kopie archiwalną pliku kontrolnego.

§ 23.2. Rodzaje archiwizacji

- **Archiwizacja fizyczna (physical backup)** - polega na sporządzaniu kopii plików bazy danych za pomocą polecenie systemu operacyjnego.
- **Archiwizacja logiczna (logical backup)** - polega na skorzystaniu z programu **Exp** do sporządzenia kopii struktur logicznych bazy danych. Program ten zapisuje informacje o obiektach bazy danych i same dane w pliku binarnym. Plik ten może być przetwarzany przez program **Imp** .

Archiwizacje można przeprowadzić różnymi metodami w zależności od trybu pracy bazy:

- `ARCHIVELOG` /*z archiwizacją plików dziennika powtórzeń */
- `NOARCHIVELOG` /*bez archiwizacji plików dziennika powtórzeń */

§ 23.3. Archiwizacja fizyczna całej bazy danych w trybie `OFFLINE`

W celu wykonania archiwizacji fizycznej należy:

1. Określić nazwy i położenie wszystkich plików bazy dany. W tym celu należy wydać zapytania:

```
SQL> SELECT status, enabled, name FROM v$datafile;
SQL> SELECT * FROM V$LOGFILE;
SQL> SELECT * FROM V$CONTROLFILE;
```

2. Zatrzymać instancję.

```
SQL> SHUTDOWN
```

3. Korzystając z poleceń systemu operacyjnego sporządzić kopie wszystkich plików określonych w kroku 1. Sporządzić kopie pliku **`init<SID>.ora`** i pliku haseł **`pwd<sid>.ora`**.

4. Uruchomić instancję.

```
SQL> STARTUP
```

§ 23.4. Archiwizacja fizyczna całej bazy danych w trybie `ONLINE`

Archiwizacja ta jest możliwa tylko, gdy baza pracuje w trybie archiwizacji plików dziennika powtórzeń (`ARCHIVELOG`).

Przed sporządzeniem fizycznej kopii plików składającą się na wybraną przestrzeń tabel należy wydać polecenie

```
SQL> ALTER TABLESPACE nazwa_przestrzeni BEGIN BACKUP;
```

Po zrobieniu kopii fizycznych plików wchodzących w skład tej przestrzeni tabel należy wydać polecenie

```
SQL> ALTER TABLESPACE nazwa_przestrzeni END BACKUP;
```

Należy również zrobić kopie **pliku kontrolnego** i wszystkich **zarchiwizowanych plików dziennika powtórzeń**.

§ 23.5. Archiwizacja plików kontrolnych

Plik kontrolny zarchiwizować można w następujący sposób.

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO 'nazwa pliku';
```

Należy je wydać w programie SQL*Plus w czasie, gdy baza pracuje w trybie `MOUNT` lub `OPEN`.

Polega to na utworzeniu pliku binarnego będącego kopią pliku kontrolnego.

§ 23.6. Archiwizacja logiczna

Do wykonania archiwizacji logicznej służy program **`Exp`**.

Do pracy wymaga on dodatkowych obiektów systemowych. Obiekty te tworzone są za pomocą skryptu `catexp.sql` (wywoływanego za pomocą skryptu `catalog.sql`), muszą być własnością użytkownika `SYS`.

W systemie Oracle mogą być eksportowane:

- **Pojedyncze tabele.**
- **Wszystkie obiekty określonych użytkowników.**
- **Cała baza danych.**

W przypadku eksportowania pojedynczej tabeli oprócz definicji eksportowane są również: dane, prawa dostępu, indeksy, ograniczenia integralnościowe i wyzwalacze.

Eksport schematu użytkownika polega na sporządzeniu kopii wszystkich obiektów użytkownika.

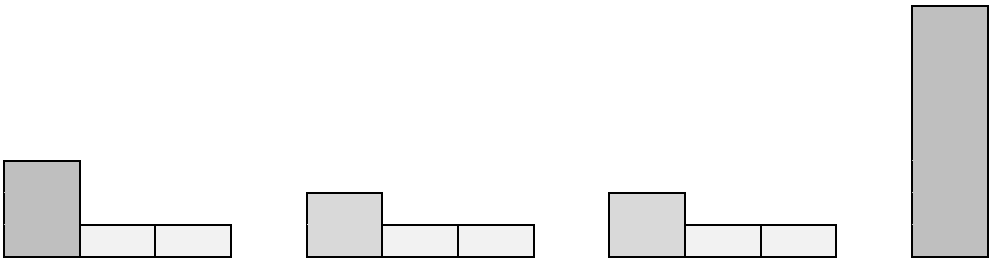
Eksport całej bazy danych jest równoważny:

- wyeksportowaniu wszystkich użytkowników (z wyjątkiem SYS),
- synonimów,
- praw dostępu,
- uprawnień systemowych,
- definicji przestrzeni tabel,
- definicji segmentów wycofania,
- opcji auditingu,
- profili użytkowników i wszystkich wyzwalaczy.

Tego rodzaju operacje może wykonać użytkownik, który posiada role `EXP_FULL_DATABASE`.

Może on być wykonany w trzech trybach:

- pełnym,
- inkrementalnym,
- kumulacyjnym.



Rysunek 23.6.1. Tryby operacji exportu (program `exp`).

Rodzaje eksportu:

- **Eksport kompletny** jest równoważny eksportowi całej bazy danych. Dodatkowo wykonanie tego eksportu powoduje usunięcie informacji systemowych dotyczących kolejnych eksportów inkrementalnych i kumulacyjnych.

- **Eksport kumulacyjny** umożliwia sporządzenie pełnych kopii tych tabel, których zawartość bądź definicja zostały zmienione od czasu wykonania ostatniego eksportu **kumulacyjnego** lub **pełnego**. Oznacza to, że kopie danych sporządzone za pomocą wcześniejszych eksportów inkrementalnych są już niepotrzebne.
- **Eksport inkrementalny** umożliwia sporządzenie pełnych kopii tych tabel, których zawartość bądź definicja zostały zmienione od czasu wykonania ostatniego eksportu **inkrementalnego**, **kumulacyjnego** lub **pełnego**.

Procedura sporządzania kopii zawartości całej bazy danych z wykorzystaniem eksportów inkrementalnych, kumulacyjnych i pełnych może być np. następująca:

- Wykonanie początkowego eksportu kompletnego bazy danych.
- Wykonanie eksportów inkrementalnych pod koniec każdego dnia.
- Wykonanie eksportów kumulacyjnych okresowo, np. co tydzień.
- Wykonanie kompletnego eksportu okresowo np. co miesiąc.

§ 23.7. Archiwizacja logiczna. Program Exp

23.7.1. Eksport konta użytkownika

Konto użytkownika może eksportować właściciel lub użytkownik z rolą `DBA`.

```
c:\exp system@xe OWNER=(kadry) FILE=c:\kadry1.dmp
                                              LOG=c:\kadry1.log
```

lub

```
c:\exp kadry@xe FILE=c:\kadry1.dmp LOG=c:\kadry1.log
```

Można używać pliku parametrów.

```
c:\exp PARFILE=c:\exp_kadry1.par
```

gdzie plik `exp_kadry1.par` jest postaci:

```
USERID=system@xe
FILE=c:\kadry1.dmp
LOG=c:\kadry1.log
OWNER=(kadry)
```

23.7.2. Eksport kont kilku użytkowników

```
c:\exp PARFILE=c:\exp_kadry2.par
```

gdzie plik `exp_kadry2.par` jest postaci:

```
USERID=system@xe
FILE=c:\kadry2.dmp
LOG=c:\kadry2.log
OWNER=(kadry,pbd)
ROWS=N
```

23.7.3. Eksport wybranych tabel

```
c:\exp PARFILE=c:\exp_kadry3.par
```

gdzie plik `exp_kadry3.par` jest postaci:

```
USERID=system@xe
FILE=c:\kadry3.dmp
LOG=c:\kadry3.log
TABLES=(kadry.osoby,kadry.wydzialy)
GRANTS=N                                    /* Bez przywilejów */
```

23.7.4. Eksport całej bazy - kompletny

Do eksportu i importu całej bazy trzeba posiadać role `EXP_FULL_DATABASE` i `IMP_FULL_DATABASE`.

```
c:\exp PARFILE=c:\exp_xe4_full.par
```

gdzie plik `exp_xe4.par` jest postaci:

```
USERID=system@xe
FILE=c:\exp_xe4_full.dmp
LOG=c:\exp_xe4_full.log
FULL=Y
INCTYPE=COMPLETE                          /* Eksport kompletny */
```

23.7.5. Eksport całej bazy – przyrostowy

```
c:\exp PARFILE=c:\exp_xe5_full.par
```

gdzie plik `exp_xe5_full.par` jest postaci:

```
USERID=system@xe
FILE=c:\exp_xe5_full.dmp
LOG=c:\exp_xe5_full.log
FULL=Y
INCTYPE=INCREMENTAL                       /* Eksport przyrostowy */
```

Najczęściej używane parametry programu **Exp**:

USERID - **Określa nazwę i hasło użytkownika, który dokonuje exportu.**

BUFFER - **Określa rozmiar bufora pamięci operacyjnej wykorzystywanego do przetwarzania eksportowanych danych. Rozmiar tego bufora (w bajtach) powinien mieć przynajmniej taką wartość, jak najdłuższy eksportowany rekord.**

FILE - **Wskazuje nazwę pliku, do którego będą eksportowane dane.**

FULL - **Wartość `Y` powoduje eksport całej bazy danych, natomiast `N` (domyślna) umożliwia eksportowanie obiektów wybranych użytkowników lub tylko określonych tabel.**

OWNER - **Określa nazwy użytkowników, których obiekty będą eksportowane. Przykładowo, parametr `OWNER=(u1,u2)` umożliwi wyeksportowanie wszystkich obiektów użytkowników `u1,u2`. Prawo do eksportowania innych użytkowników posiada administrator bazy danych.**

TABLES - **Określa zbiór tabel, które będą wyeksportowane. `TABLES=(kadry.osoby, kadry.wydzialy)`.**

ROWS - **Wartość `Y` (domyślna) powoduje wyeksportowanie definicji wraz z jej zawartością, `N` – tylko definicja.**

INDEXES - **Wartość `Y` (domyślna) powoduje wyeksportowanie indeksów związanych z eksportowanymi tabelami. `N` – indeksy nie są eksportowane.**

CONSTRAINTS - **Wartość `Y` (domyślna) powoduje wyeksportowanie ograniczeń integralnościowych. w przypadku `N` nie będą one eksportowane.**

GRANTS - **Wartość `Y` (domyślna) powoduje wyeksportowanie przywilejów związanych z eksportowanymi tabelami.**

INCTYPE - **Określa tryb eksportu: `COMPLETE`, `CUMULATIVE`, `INCREMENTAL`. Eksport w tych trybach można wykonywać tylko dla `FULL=Y`.**

LOG - **Wskazuje plik, do którego zostaną zapisane informacje o przebiegu eksportu.**

RECORD - **Wartość `Y` (domyślna) oznacza, że informacje o eksportach inkrementalnych i kumulacyjnych będą zapisywane w tabelach: `SYS.INCVID`, `SYS.INCFIL`, `SYS.INCEXP`.**

PARFILE - **Wskazuje plik z parametrami eksportu.**

HELP - Umożliwia wyświetlenie opisu parametrów.

§ 23.8. Odtwarzanie bazy danych w trybie NOARCHIVELOG

23.8.1. Wczytanie pełnej kopii archiwalnej

Procedura jest następująca:

- **Zamknięcie bazy danych poleceniem `SHUTDOWN` lub `SHUTDOWN ABORT`.**
- **Wgranie wszystkich plików bazy danych z ostatniej kopii na ich właściwe miejsca.**
- **Otwarcie bazy danych poleceniem `STARTUP`.**

23.8.2. Odtwarzanie bazy danych na podstawie pliku eksportu – program `Imp`

Program `Imp` umożliwia importowanie:

- Pojedynczych tabel.
- Wszystkich obiektów określonych użytkowników.
- Całej bazy danych.

Mechanizm importowanie ma również zastosowanie w przypadku przenoszenia danych z jednej bazy do drugiej.

§ 23.9. Importowanie danych jednego użytkownika do drugiego

Mamy użytkowników `kadry` i `pbd`.

```
c:\imp PARFILE=c:\imp_kadry1.par
```

gdzie plik `imp_kadry1.par` jest postaci:

```
USERID=system@xe
BUFFER=4096
FILE=c:\kadry1.dmp
```

```
LOG=c:\imp_kadry1.log
FROMUSER=(kadry)
TOUSER=(pbd)
ROWS=Y                    /* Z danymi */
GRANTS=N                  /* Bez przywilejów */
COMMIT=Y                  /* Zatwierdzanie po pełnym buforze */
```

lub

```
USERID=system@xe
BUFFER=4096
FILE=c:\kadry1.dmp
LOG=c:\imp_kadry1.log
FROMUSER=(kadry)
TOUSER=(pbd)
TABLES=(osoby,wydzialy)      /* Wybrane tabele */
ROWS=N                       /* Bez danych */
GRANTS=N                     /* Bez przywilejów */
INDEXES=N                 /* Bez indeksów */
```

§ 23.10. Odtwarzanie bazy danych na podstawie pełnego eksportu

Musi być stworzona instancja `test` i baza danych.

```
c:\imp PARFILE=c:\imp_test2.par
```

gdzie plik `imp_test2.par` jest postaci:

```
USERID=system@test
BUFFER=4096
FILE=c:\test4.dmp
LOG=c:\imp_test2.log
FULL=Y
```

Parametry programu `Imp`:

USERID - **Określa nazwę i hasło użytkownika, który importuje dane.**

BUFFER - Określa rozmiar bufora (w bajtach) pamięci operacyjnej wykorzystywanej do przetwarzania danych.

FILE - **Wskazuje plik eksportu.**

SHOW - Wartość `Y` oznacza, że dane nie zostaną zaimportowane do bazy, a zawartość zostanie tylko wyświetlona na ekranie. Natomiast `N` spowoduje wczytanie danych.

IGNORE - Określa sposób reagowania na błędy powstające podczas importu danych wynikające z istnienia obiektu w bazie danych. Jeżeli parametr przyjmie wartość `Y`, a importowane są definicje obiektów już istniejących, to program nie tworzy ponownie tych obiektów, lecz je pomija nie wyświetlając informacji o błędach. W tabelach tych mogą się pojawić zduplikowane rekordy, jeśli nie miały one założonych ograniczeń integralnościowych: `unique` i `primary key`. W przypadku wartości `N` próba wczytania obiektu już istniejącego zakończy się niepowodzeniem i wyświetleniem informacji o błędzie.

FULL - **Wartość `Y` powoduje import całej bazy danych, natomiast `N` (wartość domyślna) umożliwia importowanie obiektów wybranych użytkowników lub określonych tabel. Importu całej bazy może dokonać użytkownik z uprawnieniami `IMP_FULL_DATABASE`.**

FROMUSER - **Określa zbiór użytkowników, których obiekty zostaną wczytane.**

TOUSER - **Określa zbiór użytkowników, do których schematów dane zostaną wczytane. Opcję tę może wykorzystać użytkownik posiadający rolę `IMP_FULL_DATABASE`**

TABLES - **Określa zbiór tabel, które zostaną zaimportowane.**

ROWS - **Wartość `Y` (domyślna) powoduje wczytanie definicji tabel wraz z danymi.**

INDEXES - **Wartość `Y` powoduje wczytanie indeksów związanych z importowanymi tabelami.**

GRANTS - **Wartość Y powoduje wczytanie informacji o przywilejach związanych z importowanymi tabelami.**

COMMIT - **Wartość Y powoduje, że po wczytaniu do bazy danych bloku informacji o wielkości określonej BUFFER system zatwierdzi transakcję. Natomiast N powoduje, że dane zatwierdzane są dopiero po wczytaniu całej tabeli.**

DESTROY - Wartość Y powoduje nadpisanie istniejących przestrzeni tabel definicjami zawartymi w importowanym pliku.

INCTYPE - Określa typ importowania danych i może przyjmować wartości: SYSTEM lub RESTORE.

LOG - **Wskazuje plik, do którego zostaną zapisane informacje o przebiegu importu.**

PARFILE - **Wskazuje plik zawierający parametry importu.**

HELP - Umożliwia wyświetlenie parametrów importu.

24. ZABEZPIECZENIE BAZY DANYCH PRZED AWARIĄ

24.1.1. System Change Number (SCN)

Każda zatwierdzona transakcja otrzymuje unikatowy numer **System Change Number** (**SCN**).

Informacje o tych numerach można znaleźć w perspektywie systemowej `V_$LOG_HISTORY`.

24.1.2. Punkt kontrolny (Checkpoint)

Punkt kontrolny jest systemowym zdarzeniem bazy danych, dzięki któremu dane z buforów pamięci SGA są zapisywane na dyski (**do plików danych trafiają zmiany zatwierdzone i niezatwierdzone**). Zapisu dokonuje proces serwera `DBWR (Database Writer)`.

Występują dwa rodzaje punktu kontrolnego:

- **Bazy danych** (`database checkpoint`) – wykonywany dla wszystkich plików danych.
- **Pliku danych** (`datafile checkpoint`) – wykonywany dla określonych plików danych.

Punkt kontrolny bazy danych (`database checkpoint`) występuje, gdy:

- Następuje przełączenie grupy plików dziennika powtórzeń.
- Proces `LGWR` zapisał do pliku dziennika powtórzeń pewną liczbę bloków z bufora dziennika powtórzeń (z SGA).
- Wykonane zostanie polecenie:

```
SQL> ALTER SYSTEM CHECKPOINT;
```

- Wykonane zostanie polecenie:

```
SQL> SHUTDOWN NORMAL
```

lub

```
SQL> SHUTDOWN IMMEDIATE
```

Punkt kontrolny pliku danych (`datafile checkpoint`) występuje, gdy:

- Rozpoczyna się sporządzanie kopi bezpieczeństwa wybranych przestrzeni tabel

```
SQL> ALTER TABLESPACE nazwa_przestrzeni BEGIN BACKUP;
```

- Po

```
SQL> ALTER TABLESPACE nazwa_przestrzeni OFFLINE;
```

- …

§ 24.2. Pliki kontrolne

W pliku kontrolnym i w nagłówku każdego pliku danych są przechowywane informacje m.in.:

- Numer SCN w czasie wystąpienia punktu kontrolnego (`checkpoint at scn` w pliku danych, `stop scn` w pliku kontrolnym).
- Data i czas SCN.
- Licznik punktu kontrolnego (`checkpoint cnt`).

Plik kontrolny **(control file**) jest plikiem binarnym, który zawiera m.in.:

- Nazwę bazy danych.
- Datę utworzenia bazy.
- **Informacje o wszystkich grupach dziennika powtórzeń.**
- **Bieżącą grupę dziennika powtórzeń (aktualnie używaną).**
- **Nazwę, lokalizację i rozmiar każdego pliku dziennika powtórzeń.**
- **Informacje o plikach danych (włączony, wyłączony, …).**
- **Informacje dotyczące punktów kontrolnych.**

§ 24.3. Mechanizm odtwarzania bazy danych

Odtwarzanie bazy danych po awarii bazy przebiega w dwóch fazach:

- Z dziennika powtórzeń poprawiane są dane w plikach danych.
- Wycofanie niezatwierdzonych transakcji (korzystanie z segmentów wycofywania lub pliku automatycznego wycofywania).

NAGŁÓWEK PLIKU DANYCH

Nagłówek pliku danych przechowuje m.in.:

- Bieżący numer SCN punktu kontrolnego (`Checkpoint at scn`).
- Licznik punktu kontrolnego (`Chkpt cnt`).
- Numer sekwencyjny bieżącego pliku dziennika powtórzeń w momencie wystąpienia punktu kontrolnego.

Nagłówek pliku danych jest uaktualniany, gdy system wykonuje punkt kontrolny dla danego pliku.

NAGŁÓWEK PLIKU KONTROLNEGO

Nagłówek pliku kontrolnego przechowuje m.in.:

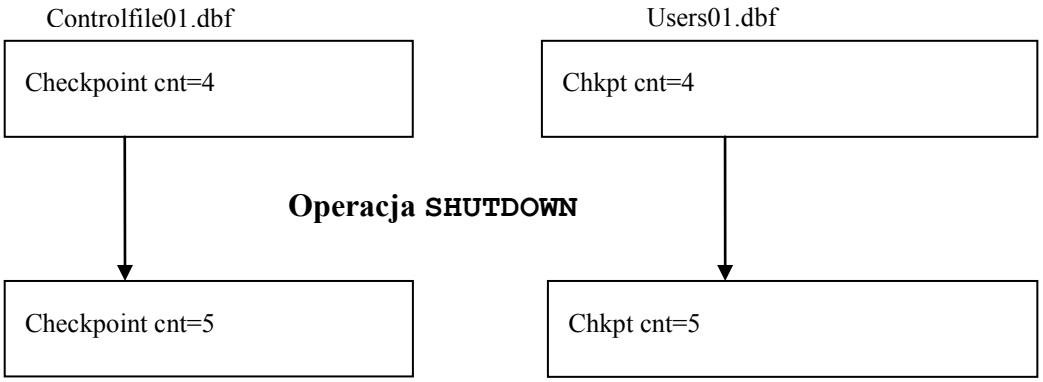
- Informacje o końcowym SCN (`Stop scn`). W czasie pracy instancji wartość ta jest równa `0.FFFF.FFFFFFFF`.
- Bieżący numer SCN (`scn`).
- Licznik punktów kontrolnych.

WYKORZYSTANIE INFORMACJI Z NAGŁÓWKÓW PLIKÓW DANYCH I KONTROLNYCH W CZASIE ODTWARZANIA BAZY DANYCH

a) W wypadku, gdy instancja zostanie zamknięta w trybie `NORMAL` lub `IMMEDIATE` to punkt kontrolny zostanie wykonany. Wtedy

```
Stop scn = Checkpointed at scn
```

dla wszystkich plików danych.



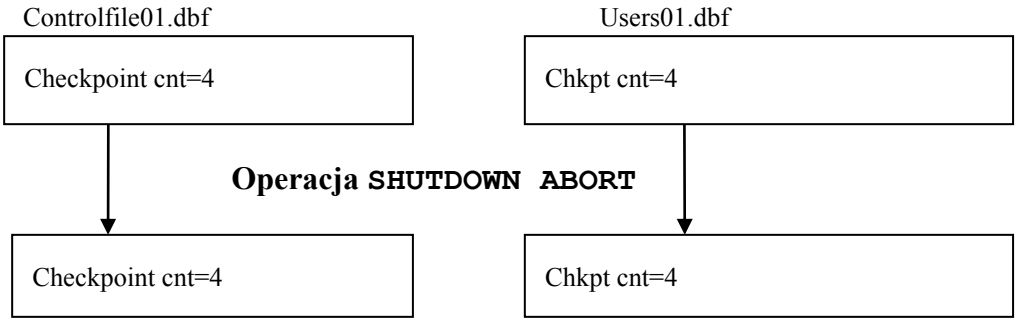
Rysunek 24.3.1. Punkty kontrolne

Oznacza to, że baza danych nie wymaga odtwarzania. Po otwarciu **Stop scn** (w pliku kontrolnym) przyjmuje znowu wartość **0xffff.ffffffff** dla wszystkich plików danych.

b) W wypadku, gdy baza danych zostanie zamknięta w trybie ABORT punkt kontrolny nie zostanie wykonany. Oznacza to, że

```
Stop scn <> Checkpointed at scn
```

dla wszystkich plików danych.



Rysunek 24.3.2. Punkty kontrolne

Oznacza to, że

- Jeżeli plik danych nie zostanie wczytany z kopi bezpieczeństwa to należy dokonać operacji odtwarzania zmian z plików dziennika powtórzeń.
- Jeżeli plik danych zostanie wczytany z kopi bezpieczeństwa to nie będzie się zgadzała wartość licznika punktów kontrolnych (chkpt cnt) w pliku danych z odpowiadającą wartością (checkpoint cnt) w pliku kontrolnym. Niezgodność ta będzie wykryta przy starcie bazy. Wówczas zgłoszona zostanie przez system konieczność odtworzenia bazy danych (z zarchiwizowanych plików dziennika powtórzeń).

25. ODTWARZANIE BAZY DANYCH PO AWARII

§ 25.1. Rodzaje odtwarzania

W trybie NOARCHIVELOG – umożliwia odtwarzanie z kopii archiwalnej do momentu sporządzenia kopii.

W trybie ARCHIVELOG:

- Odtwarzanie pełne (**full recovery**) – do stanu przed awarią.
- Odtwarzanie niepełne (**partial recovery**) – do określonego stanu przed awarią.

§ 25.2. Odtwarzanie bazy danych w trybie NOARCHIVELOG

```
SQL> SHUTDOWN lub SHUTDOWN ABORT
```

Wgranie wszystkich plików bazy danych z ostatniej kopii bezpieczeństwa na ich właściwe miejsce.

```
SQL> STARTUP
```

§ 25.3. Odtwarzanie pełne w trybie ARCHIVELOG

Trzy sposoby realizacji odtwarzania pełnego:

- Odtwarzanie całej bazy danych – w trybie MOUNT
- Odtwarzanie pojedynczej przestrzeni tabel – w trybie OPEN przy wyłączonej tej przestrzeni tabel.
- Odtwarzanie pojedynczych plików danych – w trybie OPEN lub zamkniętej bazie danych

25.3.1. Odtwarzanie pełne całej bazy danych

Należy wydać polecenie:

```
SQL> RECOVER [AUTOMATIC] [FROM 'scieżka_log'] DATABASE;
```

gdzie:

- AUTOMATIC - automatycznie uwzględnia zmiany zawarte w plikach dziennika powtórzeń.
- ścieżka_log - miejsce zarchiwizowanych plików dziennika powtórzeń (wartość parametru LOG_ARCHIVE_DEST w pliku init<SID>.ora).

25.3.2. Odtwarzanie pełne pojedynczej przestrzeni tabel

Należy wydać polecenie:

```
SQL> RECOVER [AUTOMATIC]
     [FROM 'scieżka_log'] TABLESPACE przestrzeń,…,przestrzeń;
```

25.3.3. Odtwarzanie pełne pojedyńczego pliku danych

Należy wydać polecenie:

```
SQL> RECOVER [AUTOMATIC] [FROM 'scieżka_log'] DATAFILE plik,…,plik;
```

25.3.4. Odtwarzanie bazy danych w przypadku utraty systemowej przestrzeni tabel

Kolejność postępowania:

```
SQL> CONNECT sys@xe AS SYSDBA
SQL> SHUTDOWN
```

Wgrać wszystkie pliki uszkodzonej przestrzeni z kopii archiwalnej

```
SQL> STARTUP MOUNT
SQL> RECOVER DATABASE;
SQL> ALTER DATABASE OPEN [RESETLOGS];
```

25.3.5. Odtwarzanie bazy danych do określonego momentu w czasie

Postać polecenia:

```
SQL> RECOVER [AUTOMATIC] [FROM 'scieżka_log'] DATABASE
     UNTIL TIME 'YYYY-MM-DD:HH24:MI:SS'
     [USING BACKUP CONTROLFILE];
```

Kolejność postępowania:

```
SQL> CONNECT sys@xe AS SYSDBA
SQL> SHUTDOWN
```

Wgranie plików danych z kopi archiwalnej.

```
SQL> STARTUP MOUNT
SQL> RECOVER DATABASE UNTIL TIME '2003-01-01:00:00:00';
SQL> ALTER DATABASE OPEN RESETLOGS;
                                    /*otwarcie bazy z zerowaniem numerów sekwencyjnych*/
```

25.3.6. Odtwarzanie bazy danych do przerwania

Postać polecenia:

```
SQL> RECOVER [AUTOMATIC] [FROM 'scieżka_log'] DATABASE
     UNTIL CANCEL
     [USING BACKUP CONTROLFILE];
```

Kolejność postępowania:

```
SQL> CONNECT sys@xe AS SYSDBA
SQL> SHUTDOWN
```

Wgranie plików danych z kopi archiwalnej.

```
SQL> STARTUP MOUNT
SQL> RECOVER DATABASE UNTIL CANCEL;
SQL> ALTER DATABASE OPEN RESETLOGS;
                                    /*otwarcie bazy z zerowaniem numerów sekwencyjnych*/
```

26. MECHANIZM EXPORTU I IMPORTU DATA PUMP

§ 26.1. Export

```
SQL> CREATE DIRECTORY dtpump AS 'c:\dtpump';

SQL> GRANT READ ON DIRECTORY dtpump TO kadry, system;

SQL> GRANT WRITE ON DIRECTORY dtpump TO kadry, system;
```

W przypadku pełnego exportu wymagane jest posiadanie uprawnienia systemowego

`EXP_FULL_DATABASE`

Utwórzmy plik z parametrami `dp1.par`:

```
DIRECTORY=dtpump
DUMPFILE=dp1.dmp
CONTENT= all                          /* metadata_only, … */
```

```
C:\expdp kadry/kadry PARFILE=c:\oraclexe\dp1.par
```

Utwórzmy plik z parametrami `dp2.par`:

```
DIRECTORY=dtpump
DUMPFILE=dp2.dmp
CONTENT= all
```

```
C:\expdp system/test PARFILE=c:\oraclexe\dp2.par
```

§ 26.2. Import

W przypadku pełnego importu wymagane jest posiadanie uprawnienia systemowego

`IMP_FULL_DATABASE`

Utwórzmy plik z parametrami `dp1imp.par`:

```
DIRECTORY=dtpump
DUMPFILE=dp1.dmp
CONTENT= all
REMAP_SCHEMA=kadry:pbd
```

```
C:\expdp pbd/pbd PARFILE=c:\oraclexe\dp1imp.par
```

27. RMAN

W trybie **offline** kopie robi się następująco (baza może działać w trybie ARCHIVELOG lub NOARCHIVELOG):

```
C:\rman

Recovery Manager: Release 10.2.0.1.0 - Production on So Maj 12 02:52:09 2007
Copyright (c) 1982, 2005, Oracle.  All rights reserved.

RMAN>

RMAN> connect target
  connected to target database: XE (DBID=2503361492)

RMAN> SHUTDOWN IMMEDIATE
  using target database control file instead of recovery catalog
  database closed
  database dismounted
  Oracle instance shut down

RMAN> STARTUP MOUNT
  connected to target database (not started)
  Oracle instance started
  database mounted
  Total System Global Area     285212672 bytes
  Fixed Size                     1287016 bytes
  Variable Size                117443736 bytes
  Database Buffers             163577856 bytes
  Redo Buffers                   2904064 bytes

RMAN> BACKUP DATABASE FORMAT 'c:\rmanbac\rman_%d_%t_%U.bus';

Starting backup at 07/05/12
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=35 devtype=DISK
channel ORA_DISK_1: starting full datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
input datafile fno=00001 name=C:\ORACLEXE\ORADATA\XE\SYSTEM.DBF
input datafile fno=00003 name=C:\ORACLEXE\ORADATA\XE\SYSAUX.DBF
input datafile fno=00002 name=C:\ORACLEXE\ORADATA\XE\UNDO.DBF
input datafile fno=00004 name=C:\ORACLEXE\ORADATA\XE\USERS.DBF
channel ORA_DISK_1: starting piece 1 at 07/05/12
channel ORA_DISK_1: finished piece 1 at 07/05/12
piece handle=C:\RMANBAC\RMAN_XE_622350054_01IHGJN6_1_1.BUS
                                        tag=TAG20070512T030053 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:01:05
channel ORA_DISK_1: starting full datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
including current control file in backupset
channel ORA_DISK_1: starting piece 1 at 07/05/12
channel ORA_DISK_1: finished piece 1 at 07/05/12
piece handle=C:\RMANBAC\RMAN_XE_622350120_02IHGJP8_1_1.BUS
                                        tag=TAG20070512T030053 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:02
Finished backup at 07/05/12
```

W trybie **online** kopie robi się takim samym poleceniem z tym tylko, że baza musi być w trybie ARCHIVELOG.

Kopie wykonuje się poleceniem:

```
RMAN> BACKUP DATABASE FORMAT 'c:\rmanbac\rman_%d_%t_%U.bus';
```

lub do katalogu domyślnego z domyślną nazwą plików poleceniem:

```
RMAN> BACKUP DATABASE;
```

§ 27.1. Tworzenie kopii zapasowej przestrzeni tabel

Możliwość podawania jedynie podzbioru przestrzeni tabel podczas procesu archiwizacji danych zwiększa zakres możliwych strategii tworzenia kopii zapasowych.

Kopie wykonuje się poleceniem:

```
RMAN> BACKUP TABLESPACE system, users FORMAT
                                    'c:\rmanbac\rman_%d_%t_%U.bus';
```

lub do katalogu domyślnego z domyślną nazwą plików poleceniem:

```
RMAN> BACKUP TABLESPACE system, users;
```

§ 27.2. Tworzenie kopii zapasowej plików danych

Pliki danych można archiwizować przez podanie ich numerów lub nazw.

Składnia polecenia jest następujaca:

```
RMAN> BACKUP DATAFILE 1,2;
RMAN> BACKUP DATAFILE 'c:\oraclexe\oradata\users.dbf';
```

Przy czym numery plików można uzyskać z perspektywy `V$DATAFILE` poleceniem:

```
SQL> SELECT file#, name FROM v$datafile;
```

§ 27.3. Tworzenie kopii zapasowej plików kontrolnych

Przydatną właściwością systemu Oracle jest możliwość automatycznej archiwizacji plików kontrolnych po każdorazowym wydaniu polecenia `backup`. Można to osiągnąć za pomocą polecenia `configure` w następujący sposób:

```
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;
```

Samodzielnie archiwizację pliku kontrolnego w systemie Oracle przeprowadza się za pomocą polecenia:

```
RMAN> BACKUP CURRENT CONTROLFILE;
```

§ 27.4. Tworzenie kopi zapasowej archiwalnych plików dziennika powtórzeń

W przypadku wersji Oracle składnia polecenia tworzącego kopie zapasową archiwalnych plików dziennika powtórzeń (archived redo log files) jest następująca:

```
RMAN> BACKUP ARCHIVELOG ALL;
```

Inna technika, możliwa do zastosowania w systemie Oracle, jest zastosowanie w poleceniu `backup` wyrażenia `plus archivelog`, dzięki czemu archiwalne pliki dziennika powtórzeń wchodzą w skład kopii zapasowej.

Wydanie poniższej instrukcji powoduje utworzenie co najmniej dwóch elementów kopii zapasowej - dla plików danych i dla archiwalnych plików dziennika powtórzen:

```
RMAN> BACKUP DATABASE FORMAT 'c:\rmanbac\rman_%d_%t_%U.bus'
                                                  plus archivelog;
```

§ 27.5. RESTORE (Odtwarzanie) i RECOVER (rekonstrukcja)

Uruchomienie programu RMAN i ustanowienie połączenia z docelową bazą danych.

```
RMAN> CONNECT target
```

Wywołanie odpowiedniego polecenia `RESTORE` programu RMAN w celu przywrócenia wymaganych plików.

Odtwarzane są pliki wymagane oraz odpowiednie archiwalne pliki dziennika powtórzeń.

Po odtworzeniu potrzebnych plików należy zrekonstruować (`recover`) bazę danych oraz ją otworzyć.

Rekonstrukcji bazy danych można dokonać z poziomu programu RMAN lub z narzędzia SQL*Plus.

Przykład 27.5.1. W poniższym przykładzie pokazano sposób odtwarzania i rekonstrukcji przestrzeni tabel `USERS`:

```
RMAN> RESTORE TABLESPACE users;
RMAN> RECOVER TABLESPACE users;
```

Oprócz przestrzeni tabel można odtworzyć i zrekonstruować konkretny plik danych:

```
RMAN> RESTORE DATAFILE 'c:\oraclexe\backupxe\users.dbf';
RMAN> RECOVER DATAFILE 'c:\oraclexe\backupxe\users.dbf';
```

Po wydaniu polecenia `RECOVER` z poziomu programu RMAN w pierwszej kolejności następuje sprawdzenie dostępności potrzebnych archiwalnych plików dziennika powtórzeń.

Jeśli te pliki są niedostępne, program RMAN przywraca je za pomocą wcześniej utworzonej kopii zapasowej archiwalnych plików dziennika powtórzeń.

W celu samodzielnego odtworzenia archiwalnych plików dziennika powtórzeń wydaje się polecenie

```
RMAN> RESTORE ARCHIVELOG
```

Przykład 27.5.1. W poniższym przykładzie pokazano sposób odtwarzania i rekonstrukcji przestrzeni tabel `USERS` z wybranej kopii.

Na wstępie należy uzyskać informacje o dostępnych backupach poleceniem:

```
RMAN> LIST BACKUP;

using target database control file instead of recovery catalog
List of Backup Sets
===================
BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
1       Full    460.55M    DISK        00:00:55     07/06/02
        BP Key: 1   Status: AVAILABLE  Compressed: NO  Tag: TAG20070602T005019
        Piece Name: C:\ORACLEXE\APP\ORACLE\FLASH_RECOVERY_AREA\XE\BACKUPSET\2007
_06_02\O1_MF_NNNDF_TAG20070602T005019_3618OD7T_.BKP
  List of Datafiles in backup set 1
  File LV Type Ckp SCN    Ckp Time Name
  ---- -- ---- ---------- -------- ----
  1       Full 256764     07/06/02 C:\ORACLEXE\ORADATA\XE\SYSTEM.DBF
  2       Full 256764     07/06/02 C:\ORACLEXE\ORADATA\XE\UNDO.DBF
```

```
  3       Full 256764     07/06/02 C:\ORACLEXE\ORADATA\XE\SYSAUX.DBF
  4       Full 256764     07/06/02 C:\ORACLEXE\ORADATA\XE\USERS.DBF

BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
2       Full    6.80M      DISK        00:00:02     07/06/02
        BP Key: 2   Status: AVAILABLE  Compressed: NO  Tag: TAG20070602T005115
        Piece Name: C:\ORACLEXE\APP\ORACLE\FLASH_RECOVERY_AREA\XE\AUTOBACKUP\200
7_06_02\O1_MF_S_624156235_3618Q4HV_.BKP
  Control File Included: Ckp SCN: 256764       Ckp time: 07/06/02
  SPFILE Included: Modification time: 07/06/02
```

a następnie wykonać polecenia:

```
RMAN> RESTORE TABLESPACE users FROM tag= TAG20070602T005019;
RMAN> RECOVER TABLESPACE users;
```

28. GŁÓWNA KONCEPCJA DZIAŁANIA ORACLE DATA GUARD (ODG)

Oprócz podstawowych narzędzi backupu zabezpieczających przed utratą danych Oracle wprowadziło do swojego produktu narzędzie pod nazwa Oracle Data Guard (ODG).

System oparty o takie rozwiązanie, składa się z bazy produkcyjnej[1] oraz z baz zapasowych[2], gotowych w każdej chwili przejąć kontrolę w przypadku awarii bazy głównej.

§ 28.1. Niezawodność według Oracle a Data Guard

- **Odtwarzalność** – jest to cecha pozwalająca na przywrócenie stanu systemu sprzed awarii tj. poczynając od drobnych pomyłek typu usunięcie wiersza z tabeli lub całej tabeli, a skończywszy na awarii platformy sprzętowej, gdzie utracie ulega większość danych.
- **Ciągłość pracy** – jest to dobrze zdefiniowany poziom dostępności, który najczęściej wyrażany jest jako procentowa wartość nieprzerwanego jego działania w ciągu całego roku.
- **Wykrywanie uszkodzeń** – potencjalne zagrożenia powinny być automatycznie wykrywane przez odpowiednie narzędzia systemu oraz przekazywane do informacji osoby administrującej nim.

Analiza możliwości Oracle Data Guard świadczy, że system opierający na nim swoją działalność posiada wszystkie wymienione wyżej cechy. Jest on odtwarzalny, a możliwość utraty danych w wypadku posiadania kilku baz zapasowych jest redukowana do minimum.

Dzięki mechanizmowi **fast-start failover** klienci korzystający z systemu w większości przypadków nie są nawet w stanie zauważyć, że zaszła awaria, ponieważ podczas usterki bazy produkcyjnej jedna z baz zastępczych przejmuje automatycznie jej rolę.

Zaawansowane monitorowanie i zarządzanie pracą całego sytemu opartego o Data Guard dostępne jest z poziomu **Enterprise Menager** (Oracle 10g, 11g), przez co w łatwy sposób można wykryć i zapobiec wielu problemom mogącym doprowadzić w późniejszym czasie do awarii.

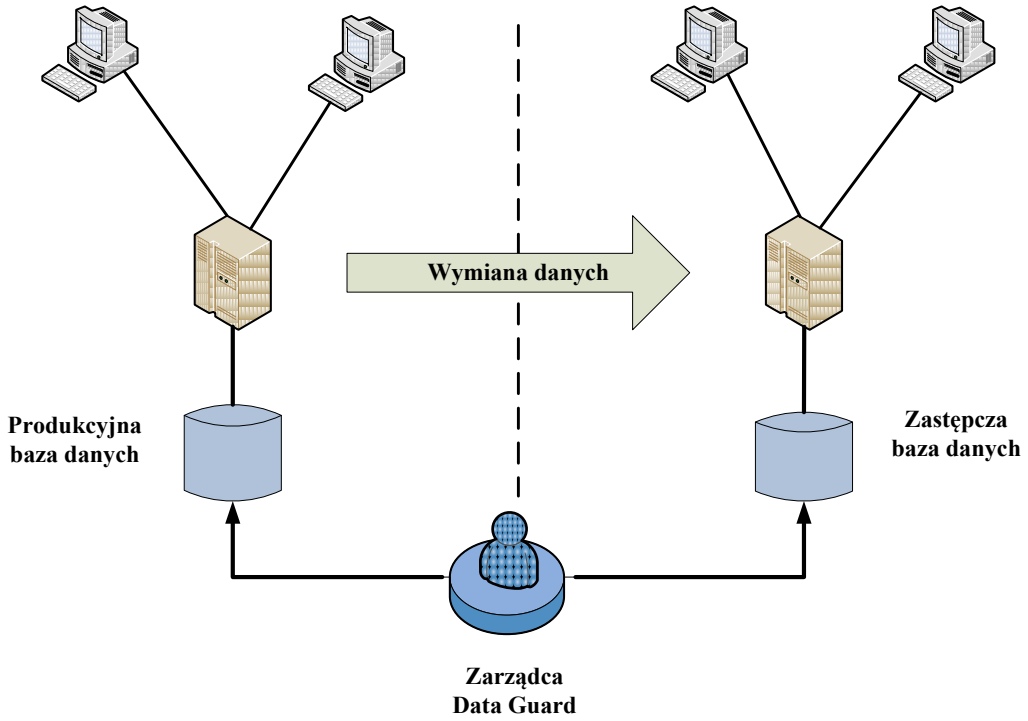
Fizyczne bazy zapasowe w Oracle Data Guard mogą być używane do generowania raportów, dzięki czemu znacznie zwiększona zostaje wydajność systemu.

Technologia Data Guard zapewnia nie tylko wysoką niezawodność systemu i dostępność, ale potrafi również znacznie zwiększyć jego wydajność.

[1] Ang. Primary database – czasami nazywana też bazą produkcyjną na której pracują aplikacje klienckie
[2] Ang. Standby database – baza gotowa do przejęcia roli bazy głównej w przypadku jej awarii

§ 28.2. Funkcjonowanie i ogólny zarys mechanizmu.

Poniższy rysunek przedstawia nam ogólną koncepcję funkcjonowania Oracle Data Guard.



Rysunek 27.2.1. Ogólna koncepcja funkcjonowania Oracle Data Guard

Do poprawnego działania Data Guard muszą być spełnione następujące wymagania:

- Główna baza danych musi działać w trybie `ARCHIVELOG` – tryb archiwizacji dziennika powtórzeń.
- Każda z baz musi posiadać swój własny plik kontrolny.

Konfiguracja Oracle Data Guard opiera się na jednej bazie produkcyjnej oraz jednej lub więcej bazach zastępczych.

ODG umożliwia skonfigurowanie do dziewięciu baz zastępczych(???) dla produkcyjnego systemu serwera bazy danych.

Bazy te są połączone ze sobą za pomocą sieci Oracle Net i mogą znajdować się w dowolnych lokalizacjach.

Ważne jest, aby miały zapewnione swobodną transmisję plików dziennika powtórzeń.

Przykładowo możemy posiadać jedną bazę zastępczą na maszynie, na której znajduje się baza produkcyjna oraz kilka baz zapasowych rozproszonych na zdalnych platformach sprzętowych.

Główna baza danych (produkcyjna) to baza, z którą łączy się większość aplikacji. Może ona być złożona z jednej instancji lub wielu instancji tej samej bazy danych.

Zapasowe bazy uaktualniane są poprzez aplikowanie do nich zmian, które zarejestrowane zostały w plikach dziennika powtórzeń głównej bazy produkcyjnej.

Transfer danych w kierunku z bazy głównej do zapasowej jest automatyczny i może odbywać się w jednym z dwóch trybów:

- **synchronicznym;**
- **asynchronicznym.**

W **trybie synchronicznym**, każda z wprowadzonych i zatwierdzonych zmian na bazie produkcyjnej nie zostanie utracona w przypadku jej nieoczekiwanej awarii. Dzieje się tak, ponieważ **transakcje głównej bazy danych zostają zatwierdzone dopiero po pomyślnej ich transmisji do co najmniej jednej bazy zapasowej**. Tryb ten jest zazwyczaj wykorzystywany przy działaniu ODG, gdzie skupiamy się głównie na maksymalnej ochronie danych i maksymalnej dostępności.

Jeżeli zaś podczas konfiguracji, najważniejsze będzie dla nas uzyskanie maksymalnej wydajności, to wtedy dane dziennika powtórzeń bazy produkcyjnej przesyłane będą w trybie **asynchronicznym** do pozostałych baz zapasowych.

Oznacza to, że transakcje na bazie głównej mogą być zatwierdzane przed ich poprawnym przetransmitowaniem na bazy zapasowe.

Wiąże się to z częściową rozbieżnością danych pomiędzy bazą produkcyjną, a zapasowymi co może prowadzić do utraty niektórych informacji podczas awarii.

§ 28.3. Logiczna i fizyczna zastępcza baza danych

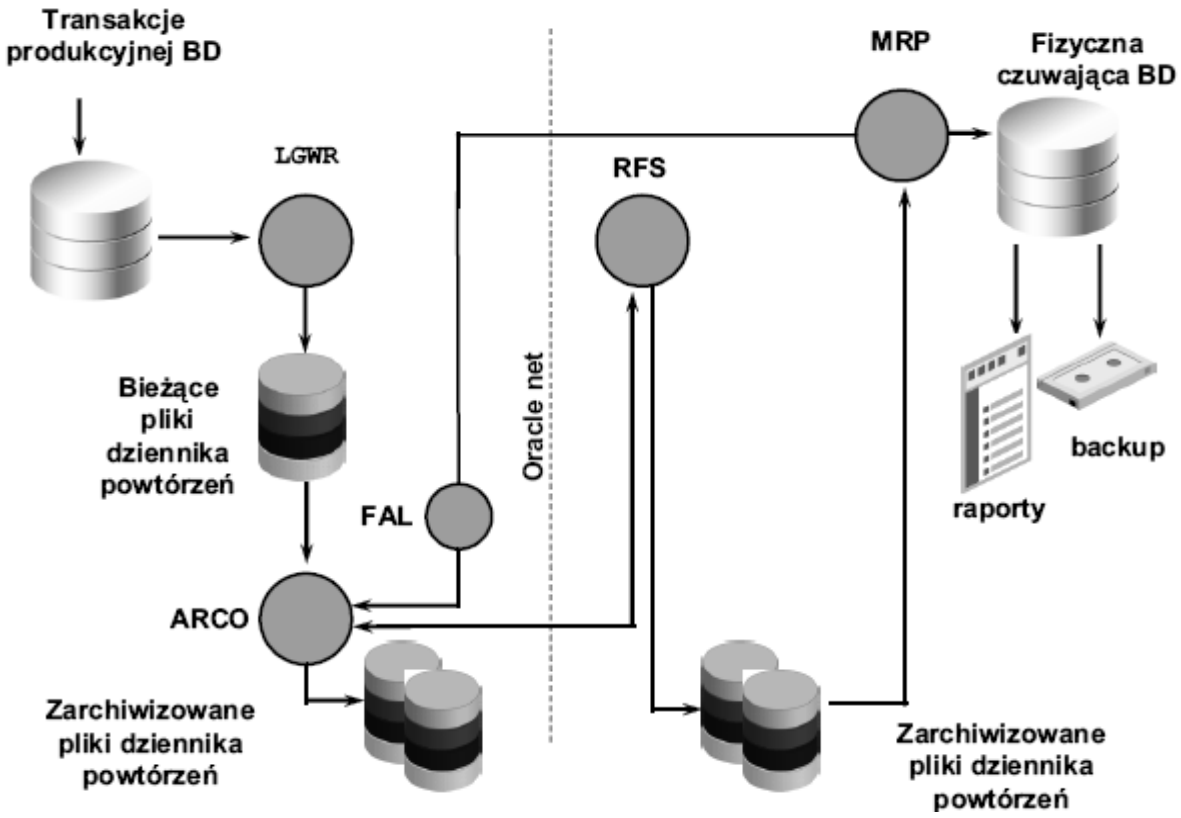
Zapasowe bazy danych mogą być fizycznymi albo logicznymi kopiami bazy produkcyjnej.

28.3.1. Fizyczna zapasowa baza danych

Fizyczna baza danych jest kopią bazy produkcyjnej, synchronizowaną poprzez aplikowanie zmian zarejestrowanych w dziennikach powtórzeń bazy produkcyjnej.

Może ona pracować w trybie tylko do odczytu co pozwala na znaczne zwiększenie mocy obliczeniowej całego systemu, niezbędnej między innymi do szybkiego generowania raportów.

Spójrzmy na przykład działania bazy danych z mechanizmem Oracle Data Guard opartym o fizyczną zapasową bazę danych i transportem danych poprzez proces `ARC`.



Rysunek 27.3.1.1. ODG z bazą fizyczną i przesyłaniem danych przez proces ARC0

W przypadku działania takiej konfiguracji bazy danych, wszystkie transakcje aplikowane na bazę produkcyjną trafiają do procesu LGWR[3], który to rejestruje w bieżących plikach dziennika powtórzeń zmiany wprowadzone na tą bazę.

Następnie pliki dziennika powtórzeń są archiwizowane na bazie głównej przez proces ARC[4] oraz przesyłane do baz zapasowych.

Proces RFS[5] na bazie zapasowej odbiera zarchiwizowane pliki z dziennika powtórzeń bazy produkcyjnej i zapisuje je na bazie zastępczej.

W dalszym kroku pliki te przejmuje proces MRP[6], który to aplikuje zmiany na fizyczną bazę danych.

W zależności od trybu ochrony bazy danych możliwe jest odtwarzanie plików dziennika powtórzeń na bazie zapasowej.

W przypadku, gdy część plików przesyłanych z bazy produkcyjnej nie dotarło do bazy czuwającej na skutek awarii bądź dużego obciążenia sieci, powinna nastąpić operacja odtworzenia transmisji tych plików. Odpowiedzialny jest za to proces FAL[7].

Fizyczne bazy danych wymagają do poprawnego działania identycznych platform sprzętowo-programowych na wszystkich współdziałających ze sobą bazach.

Tak restrykcyjnych wymagań nie posiada inny typ baz zapasowych, a mianowicie bazy logiczne.

[3] LGWR – proces odpowiedzialny za rejestrowanie w bieżących plikach dziennika powtórzeń wszystkich zmian wprowadzonych do bazy danych
[4] ARC – proces archiwizujący pliki dziennika powtórzeń
[5] RFS – proces odbierający pliki dziennika powtórzeń z bazy produkcyjnej
[6] MRP – proces aplikujący zmiany z plików dziennika powtórzeń do fizycznej bazy danych
[7] FAL – proces, który odtwarza powstałe na skutek awarii transmisji danych dziury w plikach dziennika powtórzeń przesłanych do czuwającej bazy danych

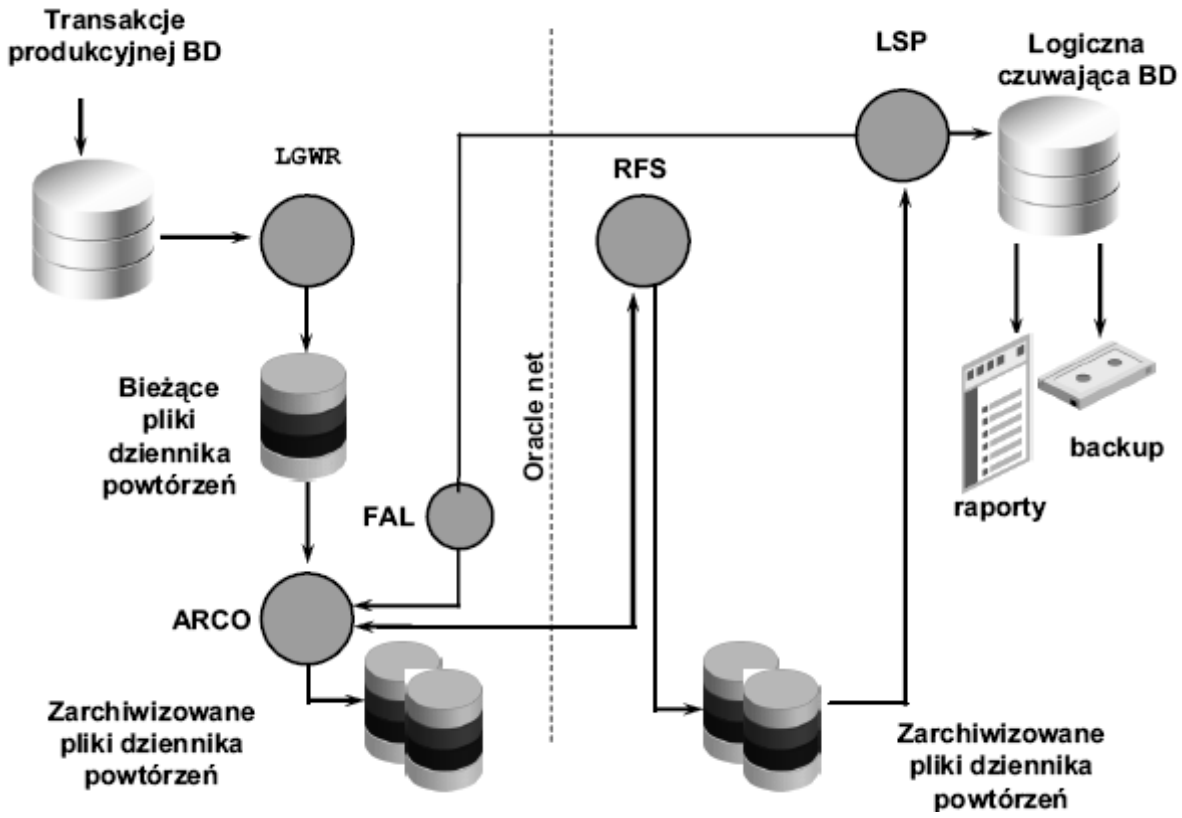
28.3.2. Logiczna zapasowa baza danych

Logiczna baza danych jest logiczną kopią obiektów (indeksów, widoków, tabel itp.) i danych z bazy głównej poprzez aplikowanie poleceń SQL.

Polecenia SQL wydobywane są za pomocą specjalistycznego narzędzia LogMiner z przesyłanych do baz zapasowych zarchiwizowanych plików dziennika powtórzeń bazy produkcyjnej.

Logiczne zapasowe bazy danych pracują w trybie odczyt-zapis i mogą być dodatkowo wykorzystywane jako hurtownie danych.

Poniższy rysunek przedstawia pracę systemu opartego o logiczną zapasową bazę danych.



Rysunek 27.3.2.1. ODG z bazą logiczną i przesyłem danych przez proces ARC0

Główną różnicą w stosunku do bazy fizycznej jest tu proces aplikujący zmiany na logiczną bazę zapasową, w tym przypadku jest to LSP[8].

Zarządzanie zarówno bazą produkcyjną jak i zapasowymi może odbywać się z linii poleceń SQL lub z poziomu tzw. zarządcy Data Guard, który posiada zarówno interfejs graficzny (zintegrowany w Oracle Enterprise Manager) jak i tekstowy (DGMGRL).

Wszystkie zadania związane z konfiguracją środowiska ODG można zautomatyzować w prosty sposób przez wykorzystanie Oracle Enterprise Manager.

[8] LSP - proces aplikujący polecenia SQL znajdujące się w plikach dziennika powtórzeń na bazę logiczną

§ 28.4. Podsumowanie

Zalety:

- **Administratorzy baz opartych o ODG nie muszą martwić się o codzienne ręczne wykonywanie kopii bezpieczeństwa, ponieważ ciągle aktualizowane bazy zapasowe wyręczają ich z tego obowiązku.**
- **Duża elastyczność pozwalająca na zapewnienie zwiększonej wydajności oraz wzrostu niezawodności i dostępności systemu bazodanowego przy stosunkowo małych nakładach potrzebnych na wzrost obu tych cech.**
- Odporność na nieplanowane awarie, jak i zaplanowane przestoje w pracy głównej bazy, co zapewnia dużą dostępność systemu.
- **Wysoka ochrona danych składowanych w bazie. ODG posiada mechanizmy, dzięki którym awaria fizyczna plików na produkcyjnej bazie danych nie zostanie rozpropagowana na bazy zapasowe.**
- Wzrost efektywności wykorzystywanych zasobów. Fizyczne zapasowe bazy danych będące aktualizowane zarchiwizowanymi plikami dziennika powtórzeń mogą być otwarte w trybie tylko do odczytu. Dzięki temu można w swobodny sposób użyć ich jako baz danych, na których wykonywane są różnego rodzaju analizy. Pozwala to w znacznej mierze odciążyć główny serwer bazy danych biorąc pod uwagę, że to właśnie zapytania przeważnie wykorzystują dużą moc obliczeniową procesora oraz ilość dostępnej pamięci operacyjnej.
- Wysoka ochrona danych składowanych w bazie. ODG posiada mechanizmy, dzięki którym awaria fizyczna plików na produkcyjnej bazie danych nie zostanie rozpropagowana na bazy zapasowe.

Jednak pomimo wielu plusów przemawiających na korzyść ODG podczas pracy z tym produktem jest także **kilka cech negatywnych**.

Jedną z nich jest występowanie małego opóźnienia pomiędzy zawartością danych z głównej bazy, a bazami zastępczymi. Może to niekiedy doprowadzić do utraty niewielkiej ilości danych.

Jednak ilość informacji, które mogą zostać zniszczone zależy tak naprawdę od administratora, ponieważ to on określa jak wielkie ma być to opóźnienie.

Poprzez analizę działania systemu powinien on w jak najlepszy sposób dostosować częstotliwość archiwizacji mając na względzie zarówno bezpieczeństwo danych i możliwości wydajnościowe platformy sprzętowej.

29. STROJENIE INSTANCJI

Przed przystąpieniem do omawiania poszczególnych kroków strojenia instancji należy najpierw wyjaśnić, czym jest strojenie.

Strojenie jest procesem dokonywania modyfikacji w sprzęcie lub oprogramowaniu, którego celem jest zmiana właściwości systemu. Należy podkreślić, że w powyższej definicji nie występuje słowo wydajność, ponieważ strojenie nie koniecznie ma służyć do polepszania wydajności.[9]

Konieczność strojenia może zachodzić m.in. z następujących powodów:

- **Strojenie przetwarzania**. Strojenie polegające na takim skonfigurowaniu systemu, aby możliwe było przetwarzanie największej ilości danych w najkrótszym czasie.
- **Strojenie czasu odpowiedzi**. Polega na takim skonfigurowaniu systemu, aby żądane dane zwracane były jak najszybciej.
- **Strojenie systemu dla dużej liczby użytkowników**. Strojenie polegające na skonfigurowaniu systemu, aby mógł on obsługiwać jednocześnie jak największą liczbę użytkowników.
- **Strojenie czasu ładowania**. Strojenie polegające na poprawie parametrów procesu ładowania.
- **Strojenie wydajności procesu tworzenia kopii zapasowych i odtwarzania**. Strojenie, którego celem jest zminimalizowanie czasu tworzenia kopii bezpieczeństwa i odtwarzania systemu po awarii.

Do głównych etapów strojenia instancji bazy danych Oracle należą:

- **strojenie przydziału pamięci (`SGA, PGA`),**
- **strojenie wykorzystania urządzeń dyskowych,**
- **strojenie segmentów wycofywania (`Rollback segments, undomanagement`),**
- **strojenie punktów kontrolnych (`Checkpoint`),**
- **strojenie sortowań,**
- **zmniejszenie rywalizacji o semafory bufora dziennika powtórzeń,**
- **minimalizacja rywalizacji o listę wolnych bloków,**
- **…**

§ 29.1. Strojenie pamięci

Instancja systemu Oracle przechowuje dane w dwóch miejscach: w pamięci oraz na dysku.

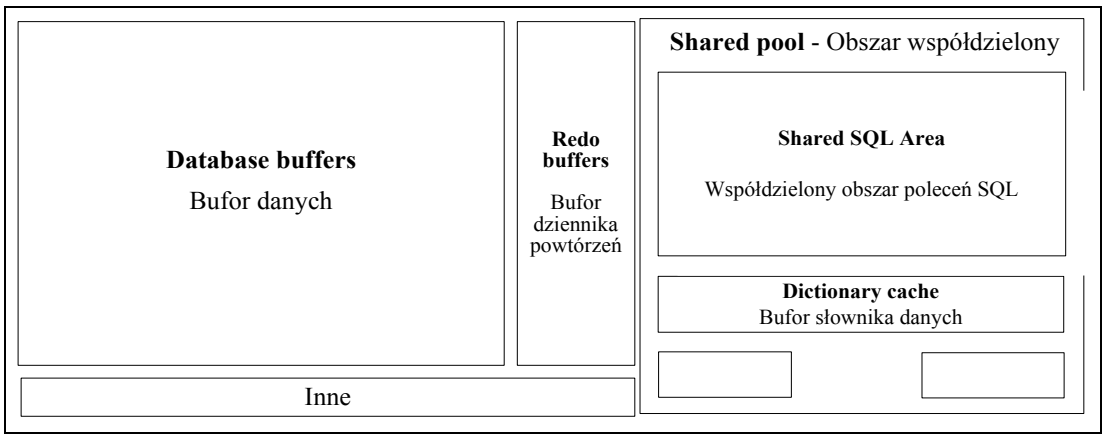
Przechowywanie danych w pamięci operacyjnej powoduje bardzo wysoką wydajność, ale jednocześnie wysoki koszt.

Z drugiej strony dysk może przechowywać ogromną ilość danych przy jednoczesnym relatywnie niskim koszcie w stosunku do pamięci.

[9] Whalen Ed.: Oracle – Optymalizacja wydajności. Helion, 2003 – str. 17

Biorąc pod uwagę wydajność osiąganą przy przechowywaniu danych w pamięci, należy przechowywać w niej dane zawsze kiedy to możliwe.

Uwzględniając ogromną ilość danych oraz liczbę użytkowników, którzy potrzebują tych danych, trzeba uwzględnić możliwość występowania konfliktów związanych z pamięcią. Żeby możliwie efektywnie wykorzystać użycie pamięci, należy osiągnąć równowagę między pamięcią używana przez bufor systemu Oracle oraz pamięcią wykorzystywaną przez użytkowników.



Rysunek 29.1.1. Globalny obszar systemowy SGA

Strojenie pamięci w instancji systemu Oracle obejmuje następujące obszary:

- **strojenie bufora dziennika powtórzeń (`Redo buffers`),**
- **strojenie przydziału prywatnego obszaru poleceń SQL (`Shared SQL Area`) ,**
- **strojenie pamięci współdzielonej (`Shared pool`),**
- **strojenie bufora danych (`Database buffers`),**
- **zmniejszanie zajętości pamięci operacyjnej.**

29.1.2. Strojenie bufora dziennika powtórzeń (`REDO BUFFERS`)

W przypadku systemów wyposażonych w szybki procesor i stosunkowo wolny dysk, pozostała część bufora może być w całości zapełniona zanim **LGWR** zdąży zapisać odpowiednia porcję informacji na dysk. Wówczas bufor ten może stać się ”wąskim gardłem” systemu. Z tego powodu większy rozmiar bufora dziennika powtórzeń powoduje mniejsze prawdopodobieństwo wystąpienia kolizji pomiędzy nowymi wpisami a wpisami starymi – będącymi w trakcie zapisu na dysk.

Ponieważ rozmiar bufora dziennika powtórzeń (zdefiniowany parametrem inicjalizacyjnym `LOG_BUFFER`) jest stosunkowo niewielki w porównaniu z całym obszarem SGA, nawet małe zwiększenie tego rozmiaru może znacząco poprawić wydajność.

Określenie rywalizacji o bufor dziennika powtórzeń można dokonać przeglądając dynamiczną perspektywę `V$SYSSTAT` szukając tam statystyki `REDO BUFFER ALLOCATION RETRIES`. Statystyki te można uzyskać za pomocą zapytania:

```
SQL> SELECT name, value FROM v$sysstat
     WHERE name = 'redo buffer allocation retries';
  NAME                                          VALUE
-------------------------------------------------------
  redo buffer allocation retries                    0
```

Wartość parametru REDO BUFFER ALLOCATION RETRIES powinna być **bliska zeru**.

Jeśli liczba ta permanentnie rośnie w trakcie działania systemu, wówczas należy zwiększyć wielkość buforu za pomocą parametru LOG_BUFFER.

```
SQL> ALTER SYSTEM SET log_buffer=7024640 SCOPE=spfile;
```

Wielkość tego parametru wyrażona jest w bajtach i powinna być wielokrotnością rozmiaru bloku systemu plików (parametr DB_BLOCK_SIZE).

Jeżeli powiększenie bufora nie przyniesie oczekiwanych rezultatów, wówczas należy rozważyć:

- **usprawnienie procesów wykonywania punktu kontrolnego (CKPT),**
- **poprawienie wydajności archiwizacji dziennika powtórzeń (być może poprzez przeniesienie plików dziennika na szybsze urządzenia zewnętrzne / dyski).**

29.1.3. Strojenie przydziału prywatnego obszaru poleceń SQL

Prywatny obszar SQL jest obszarem pamięci zawierającym między innymi bufory podręczne. Każda sesja, w której jest wykonywane zapytanie SQL, posiada taki prywatny obszar. Zmniejszenie tego obszaru może okazać się efektywne w środowisku, gdzie pracuje wielu użytkowników.[10]

Strojenie prywatnego obszaru SQL polega na zidentyfikowaniu zbędnych parsowań i następnie ograniczeniu ich liczby. Ponieważ operacja parsowania nie musi być wykonywana często (jest wykonywana w miarę udostępniania pamięci), toteż zmniejszenie liczby parsowań znacząco wpływa na wydajność systemu.

Aby określić liczbę niepotrzebnych parsowań można wykorzystać widok V$SQLAREA wykonując zapytanie:

```
SQL> SELECT sql_text, parse_calls, executions FROM v$sqlarea;
```

SQL_TEXT	PARSE_CALLS	EXECUTIONS
select * from osoby	32	32
select * from pensje where pensja > 2000	2	24

Kiedy wartość PARSE_CALLS jest bliska wartości EXECUTIONS, wówczas liczba parsowań jest zbyt duża (kursory nie są wielokrotnie wykorzystywane).

Aby aplikacja mogła wielokrotnie wykorzystywać te same kursory musi posiadać wystarczająco dużą pamięć podręczną.

Wielkość tę można zmieniać za pomocą parametru konfiguracyjnego OPEN_CURSORS. Parametr ten określa maksymalną liczbę otwartych kursorów przez aplikację użytkownika. Im większa jest wartość tego parametru, tym więcej pamięci prywatnej rezerwuje sesja użytkownika i tym większe są możliwości przechowywania informacji związanej z kursorami. Warunkiem koniecznym wielokrotnego wykorzystania kursorów jest także aby aplikacja nie zwalniała obszaru pamięci prywatnej przy zamknięciu kursora lub przy zakończeniu transakcji.

[10] Whalen Ed.: Oracle – Optymalizacja wydajności. Helion, 2003 – str.50

29.1.4. Strojenie pamięci współdzielonej (`Shared pool`)

Aby zoptymalizować pamięć współdzieloną, należy przeanalizować jej poszczególne części. Pamięć współdzielona składa się z bibliotecznej pamięci podręcznej i bufora słownika danych.

Dostęp do informacji w pamięci współdzielonej jest o wiele częstszy niż do bufora danych, zatem odpowiednie dostrojenie tego obszaru ma bardzo istotne znaczenie.

BIBLIOTECZNA PAMIĘĆ PODRĘCZNA

Biblioteczna pamięć podręczna zawiera współdzielone obszary SQL oraz PL/SQL. Przechowują one m.in. ostatnio wykonywane zapytania(na wypadek powtórzenia takiego samego zapytania przez któregoś z użytkowników), oraz kody wykonywanych procedur, funkcji i pakietów PL/SQL.

Strojenie bibliotecznej pamięci podręcznej polega głównie na zwiększeniu tzw. współczynnika trafień(*ang. hit ratio*) oraz zwiększeniu szybkości uzyskiwania dostępu do bufora biblioteki, uzyskiwane poprzez dłuższe przechowywanie rzadziej wykonywanych zapytań SQL.

Nietrafienie w bufor może zajść, gdy parsowane jest zapytanie a jego poprzednio sparsowana postać już nie istnieje buforze lub gdy próbujemy wykonać zapytanie, a obszar pamięci współużytkowanego SQL przechowujący sparsowaną postać tego zapytania został zwolniony.

Korzystanie z wcześniej sparsowanych zapytań SQL i PL/SQL jest możliwe jeśli spełnione będą poniższe kryteria:

ciąg znaków zapytania SQL musi być identyczny z ciągiem znaków zapytania przechowywanego w buforze (dotyczy to także: spacji, komentarzy, wielkości liter , itp.).

Poniższe zapytania nie mogą używać tego samego współdzielonego obszaru SQL:

```
SQL> SELECT * FROM Osoby;
SQL> SELECT * FROM osoby;
SQL> SELECT *  FROM osoby;
```

Referencje do obiektów schematu w zapytaniu SQL muszą wskazywać na te same obiekty w tym samym schemacie jak w zapytaniu przechowywanym w buforze. Przykładowo jeśli schematy użytkowników `Jan` oraz `Adam` posiadają tabelę `osoby`, i jeśli ci użytkownicy wykonają zapytanie:

```
SELECT * FROM osoby;
```

wówczas nie możliwe będzie współdzielenie tego samego obszaru SQL.

Możliwe to byłoby gdyby obaj wykonali np. zapytanie:

```
SQL> SELECT * FROM adam.osoby;
```

Zmienne wiązane w zapytaniach SQL muszą mieć taką samą nazwę i być tego samego typu.

Przykładowo poniższe zapytania nie będą używały tego samego obszaru SQL:

```
SQL> SELECT * FROM osoby WHERE wydzial=:wydzial_nr;
SQL> SELECT * FROM osoby WHERE wydzial=:w_nr;
```

Zapytania SQL muszą mieć plany wykonania wyznaczone w tym samym trybie pracy optymalizatora.

W przypadku użycia optymalizatora kosztowego identyczny musi być też cel optymalizacji.

- Statystyki opisujące efektywność bibliotecznej pamięci podręcznej są zawarte w perspektywie V$LIBRARYCACHE. Najistotniejszymi kolumnami w tej tabeli są kolumny: PINS oraz RELOADS. Pierwsza kolumna zawiera informacje o liczbie odwołań do obiektu znajdującego się w buforze danych, natomiast druga informuje o liczbie odwołań nietrafionych w bufor danych, powodujących ponowne załadowanie obiektu do bufora danych. Aby wyświetlić liczbę nietrafień do bufora można posłużyć się poniższym zapytaniem:

```
SQL> SELECT SUM(reloads) 'nietrafień',
            SUM(pins) 'odwołania do obiektu',
            100*(SUM(reloads)/SUM(pins)) 'wsp. nietrafień %'
     FROM v$librarycache;
```

nietrafień	odwołania do obiektu	WSP. nietrafień %
18	2821	0,638072

Z powyższego, przykładowego raportu wynika, że wszystkich odwołań do SQL, PL/SQL oraz innych obiektów było w sumie 2821, z czego 18 musiało zostać powtórnie załadowane. Z tego wynika, iż tylko 0,63% zapytań musiało zostać sparsowanych powtórnie.

Można również wygenerować podobny raport z podziałem na typy obiektów:

```
SQL> SELECT namespace "typ objektu", SUM(reloads) "nietrafień",
SUM(pins) "odwołania", 100*(SUM(reloads)/SUM(pins)) "wsp.nietrafień
%"
FROM v$librarycache
GROUP BY namespace;
```

typ	nietrafień	odwołania	wsp.nietrafień %
SQL AREA	**6**	**1426**	**0,420757**
TABLE/PROCEDURE	**8**	**402**	**1,99005**
BODY	0	0	0
TRIGGER	0	0	0
INDEX	0	24	0
CLUSTER	0	16	0
OBJECT	0	0	0

8 rows selected.

Współczynnik nietrafień powinien być mniejszy niż 1. Jeśli tak nie jest należy podjąć kroki zmierzające do obniżenia tej wartości. Można tego dokonać poprzez wykonywanie identycznych zapytań lub przez zwiększenie rozmiaru bibliotecznej pamięci podręcznej za pomocą parametru konfiguracyjnego

SHARED_POOL_SIZE.

Nie należy jednak przydzielać więcej pamięci niż jest fizycznie dostępne, gdyż stronicowanie pamięci może całkowicie zniweczyć zalety bufora biblioteki.[11]

STROJENIE BUFORA SŁOWNIKA DANYCH

Bufor słownika danych jest zbiorem tabel i perspektyw zawierających informacje o wszystkich obiektach bazy danych. Przechowywane są tu informacje, dotyczące między innymi: użytkowników i ich uprawnień, więzów integralności zdefiniowanych na tabelach bazy,

[11] Whalen Ed.: Oracle – Optymalizacja wydajności. Helion, 2003 – str.53

nazw i typów danych wszystkich kolumn i tabel, alokacji przestrzeni wykorzystywanej na potrzeby schematów obiektów.

Strojenie bufora słownika danych odgrywa znaczącą rolę w całym procesie strojenia instancji, gdyż ewentualne „wąskie gardła" istniejące w tym buforze są odczuwalne przez wszystkich użytkowników.

Aby sprawdzić wydajność bufora słownika danych należy posłużyć się statystykami zawartymi w dynamicznej perspektywie V$ROWCACHE.

Najważniejszymi kolumnami w tej perspektywie są kolumny: GETS, GETMISSES.

Kolumna GETS zawiera liczbę odwołań do obiektu, natomiast kolumna GETMISSES zawiera liczbę odwołań do obiektu zakończonych nietrafieniem w bufor.

Aby wyświetlić współczynnik nietrafień można posłużyć się poniższym zapytaniem:

```
SQL> SELECT SUM(getmisses) Ile_nietrafien,
       SUM(gets) Ile_odwołan,
     100*(SUM(getmisses)/SUM(gets)) Wspolczynnik_nietrafien
     FROM v$rowcache;

 Ile_nietrafien      Ile_odwołan    Wspolczynnik_nietrafien
 452                 2478           18,24052
```

Jeśli chcielibyśmy obejrzeć statystykę nietrafień dla poszczególnych typów obiektów słownika, wówczas należałoby wykonać następujące zapytanie:

```
SQL> SELECT parameter Typ,getmisses Nietrafien,
           gets Odwolania,
           100*(SUM(getmisses)/SUM(gets)) Wspolczynnik_nietrafien
      FROM v$rowcache;

  Typ                 Nietrafien  Odwolania  Wspolczynnik_nietrafien
dc_free_extents       0           0
dc_used_extents       0           0
dc_segments           184         518        35,52123
dc_tablespaces        3           21         14,28571
dc_tablespace_quotas  0           0
dc_files              0           0
dc_users              25          126        19,84126
dc_rollback_segments  21          235        8,93617
dc_objects            167         685        24,37956
dc_global_oids        6           16         37,5
dc_constraints        0           0
dc_objects_ids        164         687
dc_sequences          1           1          100
dc_usernames          10          192        5,20833
dc_database_links     0           0
dc_histogram_defs     24          24         100
dc_table_scns         0           0
...                   ...         ...        ...
```

„W przypadku intensywnie wykorzystywanego bufora słownika współczynnik nietrafień nie powinien być większy niż 10-15%.

Jeżeli współczynnik ten rośnie w trakcie pracy serwera, należy zwiększyć ilość pamięci dla tego bufora przez podniesienie wartości tego samego parametru, co dla bufora biblioteki – SHARED_POOL_SIZE.

Zazwyczaj podstrojenie współczynnika nietrafień we współdzielony obszar SQL gwarantuje poprawną wartość nietrafień w bufor słownika danych.

Jest to spowodowane sposobem gospodarowania pamięcią przez system Oracle, który preferuje dłuższe utrzymywanie w pamięci obiektów ze współdzielonego obszaru poleceń SQL niż obiektów z bufora słownika danych.

29.1.5. Strojenie bufora danych

Bufor danych jest najważniejszym buforem spośród wszystkich buforów występujących w systemie Oracle. Bufor ten zajmuje większość obszaru SGA i jest wykorzystywany podczas wykonywania każdej operacji odczytywania i modyfikowania danych z bazy.

Podczas wykonywania tych operacji proces serwera sprawdza najpierw, czy żądane dane znajdują się już w SGA. Jeśli tak, dane są bezpośrednio obsługiwane w SGA, natomiast jeśli nie dane zostaną obsłużone po uprzednim skopiowaniu ich z dysku do SGA.

Strojenie bufora danych polega na dobraniu takiego rozmiaru tego bufora, aby zagwarantować dobry współczynnik trafień. Przy dobieraniu rozmiaru należy pamiętać, że zależność między wielkością bufora a współczynnikiem trafień nie jest liniowa. Przy dużym współczynniku trafień dodatkowe zwiększenie bufora daje znikome efekty, może natomiast zwiększyć zużycie czasu procesora, który jest obciążony zarządzaniem zbyt dużą listą LRU.

Statystyka odwołań do bufora danych jest przechowywana w dynamicznej perspektywie *V$SYSSTAT*, której najważniejszymi kolumnami są:

DB BLOCK GETS	Zwykłe odwołania do bufora.
CONSISTENT GETS	Odwołania do bufora w trybie spójnym. Suma DB BLOCK GETS i CONSISTENT GETS daje liczbę wszystkich odwołań od bufora.
PHYSICAL READS	Ogólna liczba odwołań do bufora, które zakończyły się fizycznym odczytem danych z dysku, czyli nietrafieni.

Aby uzyskać informacje dotyczące poprawności działania bufora można posłużyć się poniższym poleceniem:

```
SQL> SELECT name, value FROM v$sysstat
     WHERE name in
               ('db block gets','consistent gets','physical reads');
```

NAME	VALUE
db block gets	155
consistent gets	5293
physical reads	334

Współczynnik trafień wylicza się na podstawie wzoru:

```
Cache Hit Ratio = 1-(PHYSICAL READS/(DB BLOCK GETS + CONSISTENT
GETS))
Cache Hit Ratio = 1-(334/(155+5293))=1-0,0613=0.938
```

Zatem w powyższym przykładzie współczynnik trafień jest równy 93,8%.

Jeśli współczynnik trafień jest mniejszy niż 70% - 80%, należy zwiększyć wielkość bufora danych w celu poprawienia wydajności. Można to zrobić poprzez zwiększenie wartości parametru inicjalizacyjnego

DB_CACHE_SIZE.

Firma Oracle nie zaleca obecnie używania do tego celu parametru DB_BLOCK_BUFFERS, który został pozostawiony w celu zachowania wstecznej kompatybilności.

29.1.6. Zmniejszenie zajętości pamięci operacyjnej

Zmniejszenie wielkości wykorzystywanej pamięci operacyjnej można uzyskać zarówno przez zmniejszenie liczby równoczesnych sesji użytkownika, jak i poprzez zdefiniowanie maksymalnego czasu bezczynności. Skorzystanie z pierwszego sposobu polega na zmniejszeniu wartości parametru inicjalizacyjnego

LICENSE_MAX_SESSIONS.

Aby zdefiniować maksymalny czas (w minutach) bezczynności użytkownika należy w profilu przypisanym temu użytkownikowi określić zasób IDLE_TIME.

§ 29.2. Strojenie wykorzystania urządzeń dyskowych

W procesie strojenia instancji kluczową rolę odgrywa optymalne wykorzystanie urządzeń dyskowych.

Do najważniejszych zadań administratora systemu Oracle, mających na celu strojenie urządzeń dyskowych należą:

- **diagnostyka obciążenia urządzeń dyskowych,**
- **równoważenie obciążenia urządzeń dyskowych,**
- **unikanie migracji i łańcuchowania bloków,**
- **unikanie dynamicznego zarządzania rozszerzeniami,**
- **strojenie procesu DBWR.**

29.2.1. Diagnostyka obciążenia urządzeń dyskowych

Bardzo ważnym etapem strojenia instancji bazy danych jest strojenie dostępu do dysków.

Najważniejsze informacje o operacjach wejścia-wyjścia dysku możemy znaleźć w dynamicznej tabeli V$FILESTAT, której najważniejszymi kolumnami są: PHYRDS, PHYWRTS.

Pierwsza kolumna zawiera liczbę fizycznych odczytów wykonywanych na plikach danych, natomiast druga zawiera liczbę fizycznych zapisów na tych plikach.

Informacja w V$FILESTAT stosuje wewnętrzne identyfikatory plików, zatem aby uzyskać pełne nazwy tych plików można skorzystać z perspektywy słownikowej V$DATAFILE. Jej najważniejszymi kolumnami są: NAME – zawiera nazwę pliku, STATUS – informuje o typie pliku i jego bieżącym statusie, oraz BYTES – mówiąca o wielkości danego pliku.

Aby określić liczbę fizycznych operacji na plikach można skorzystać ze wspomnianych wyżej obu tabel wykonując poniższe zapytanie:

```
SQL> SELECT substr(name,1,40) Plik, phyrds, phywrts, status, bytes
     FROM v$datafile df, v$filestat fs
     WHERE df.file# = fs.file#;
```

PLIK	PHYRDS	PHYWRTS	STATUS	BYTES
C:\ORACLE\ORADATA\TEST\SYSTEM01.DBF	1579	178	SYSTEM	209715200
C:\ORACLE\ORADATA\TEST\TEST_DANE.DBF	8	53	ONLINE	209715200
C:\ORACLE\ORADATA\TEST\TEST_RBS.DBF	7	96	ONLINE	62914560

Liczba operacji wejścia-wyjścia jest sumą odczytów i zapisów.

Należy pamiętać, aby nie przekraczać fizycznych ograniczeń dysku, gdyż problemy z wydajnością jednego dysku, w zależności od typu danych mogą spowolnić pracę całego systemu.

Redukcję przeciążenia dysków można osiągnąć przez:

- **zrównoważenie obciążenia,**
- **odpowiednie składowanie danych w blokach,**
- **unikanie dynamicznego zarządzania rozszerzeniami,**
- **unikanie dynamicznego zarządzania przestrzenią w segmentach wycofania.**

29.2.2. Równoważenie obciążenia

Równoważenie obciążenia urządzeń dyskowych polega na umieszczeniu równocześnie wykorzystywanych zasobów systemu Oracle na różnych fizycznych dyskach.

Rozmieszczenie plików danych oraz plików dziennika powtórzeń na różnych dyskach jest szczególnie zalecane przy dużej liczbie operacji modyfikowania, wstawiania i usuwania danych, które wymagają równoczesnego dostępu do danych i które generują intensywne zapisy do dziennika powtórzeń.

Biorąc pod uwagę sposób działania procesu LGWR (kończy zapis do grupy dzienników w momencie zakończenia zapisu do wszystkich plików grupy), należy pamiętać, aby podczas stosowania wielu kopii pliku dziennika powtórzeń w grupie, używane do tego celu dyski miały podobną wydajność. W przeciwnym razie jeden plik dziennika umieszczony na wolniejszym dysku będzie znacząco zmniejszał wydajność procesu LGWR.

Bardzo dobrym przykładem równoważenia obciążenia urządzeń dyskowych jest umieszczenie równocześnie wykorzystywanych tabel (np. często łączonych) w różnych przestrzeniach tabel, na różnych dyskach. Wówczas równoległy dostęp do tych tabel spowoduje przyśpieszenie operacji łączenia. Poniżej został przedstawiony inny przykład rozproszenia tabeli na różne dyski korzystający z mechanizmu alokacji rozszerzeń.

```
SQL> CREATE TABLESPACE rozproszona_przestrzeń_tabel
     DATAFILE 'C:\oradata\file1.dbf' size 10M,
              'D:\oradata\file2.dbf' size 10M,
              'E:\oradata\file3.dbf' size 10M,
              'F:\oradata\file4.dbf' size 10M;
```

```
SQL> CREATE TABLE rozproszona_tabela(…)
     TABLESPACE rozproszona_przestrzeń_tabel
     STORAGE (
               INITIAL 10000K
               NEXT 10000K
               MINEXTENTS 4
               PCTINCREASE 0
              );
```

Pierwsze polecenie tworzy przestrzeń tabel składającą się z czterech plików umieszczonych na różnych dyskach, natomiast drugie tworzy tabelę w utworzonej wcześniej przestrzeni.

Parametry składowania tabeli umożliwiają przydzielenie jej pamięci we wszystkich plikach utworzonej przestrzeni, z których każdy znajduje się na innym dysku.

Innym sposobem rozpraszania danych jest wykorzystanie mechanizmu partycjonowanych tabel.

Równoważąc obciążenie urządzeń dyskowych, nie ma potrzeby rozdzielania tabel i związanych z nimi indeksów, gdyż dostęp do nich nie jest równoległy lecz sekwencyjny. Najpierw są odczytywane indeksy, a następnie bloki tabel.

29.2.3. Migracja i łańcuchowanie bloków

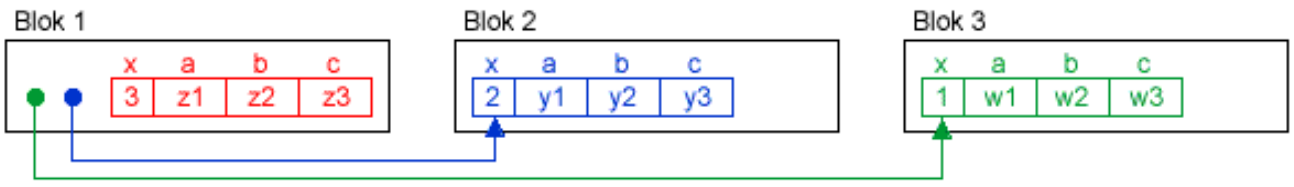
Bardzo ważną rolę w procesie strojenia wykorzystania urządzeń dyskowych odgrywa proces redukcji **operacji migracji** oraz **łańcuchowania rekordów**.

Operacje te występują, gdy zmodyfikowanie wiersza powoduje zwiększenie jego rozmiaru na tyle, iż nie mieści się on już w jednym bloku danych. W takich sytuacjach system Oracle szuka innego bloku, który jest w stanie pomieścić cały, zmodyfikowany rekord. Jeśli taki blok zostanie znaleziony, wówczas zmodyfikowany rekord jest do niego przenoszony i proces ten nazywany jest **migracją rekordu**.

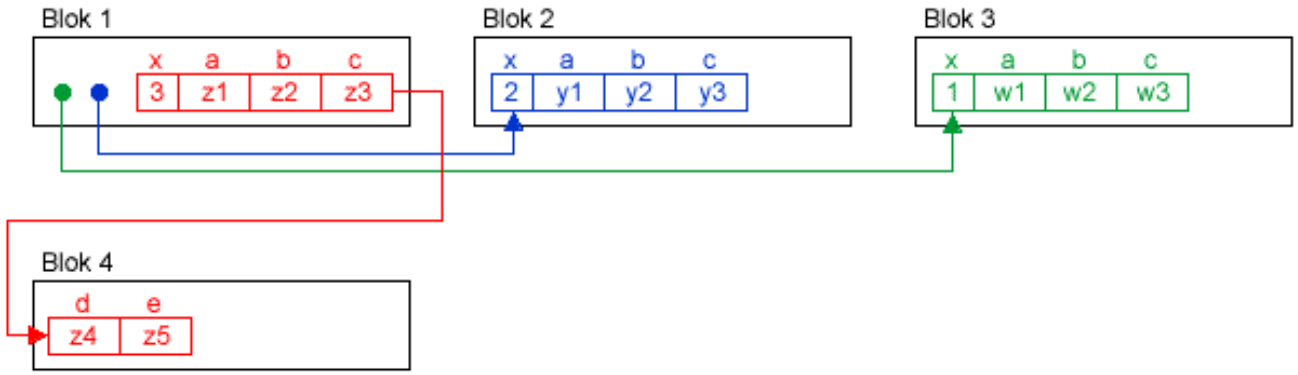
Migracja rekordu została zilustrowana rysunkiem 28.2.3.1.

W przypadku nie znalezienia takiego bloku, Oracle dzieli zmieniony rekord na kilka części i umieszcza je w różnych blokach tworząc tzw. **łańcuch bloków**. Sytuacja taka została zobrazowana rysunkiem 28.2.3.2.

Łańcuchowanie rekordu powoduje, iż późniejsze odczytanie tego rekordu zajmuje więcej niż jedną operację wejścia-wyjścia, gdyż wczytane do pamięci SGA będą musiały być wszystkie bloki danych zawierające omawiany rekord.



Rysunek 28.2.3.1. Migracja rekordu



Rysunek 28.2.3.2. Łańcuchowanie rekordu

Aby znaleźć w tabeli rekordy, które uległy migracji lub łańcuchowaniu, należy wykonać na niej polecenie ANALYZE z opcją LIST CHAINED ROWS.

Wyniki tego zapytania są gromadzone w tabeli o nazwie CHAINED_ROWS, którą można utworzyć uruchamiając skrypt UTLCHAIN.SQL. Tabela ta jest tworzona w schemacie użytkownika wykonującego ten skrypt.

Przykład 28.2.3.1. Przykład pokazujący kroki umożliwiające zredukowanie zjawiska łańcuchowania i migracji w tabeli OSOBY.

Znalezienie wierszy, które uległy łańcuchowaniu lub migracji, korzystając z polecenia ANALYZE TABLE.

```
SQL> ANALYZE TABLE osoby LIST CHAINED ROWS;
```

Wyświetlenie wyjściowej tabeli za pomocą polecenia:

```
SQL> SELECT * FROM CHAINED_ROWS WHERE TABLE_NAME = 'OSOBY';
```

OWNER_NAME	TABLE_NAME	…	HEAD_ROWID	TIMESTAMP
KADRY	OSOBY	…	AAAA1uAAHAAAAA1AAA	04-JAN-06
KADRY	OSOBY	…	AAAA1uAAHAAAAA1AAB	04-JAN-06
KADRY	OSOBY	…	AAAA1uAAHAAAAA1AAC	04-JAN-06

Tabela ta zawiera zarówno łańcuchowane jak i migrowane wiersze.

Jeśli tabela ta nie jest pusta wówczas należy podjąć kroki mające na celu eliminacje łańcuchowania i migrowania wyświetlonych rekordów postępując zgodnie z poniższymi punktami:

- Utworzenie tymczasowej tabeli z takimi samymi kolumnami jak w tabeli OSOBY, do której skopiujemy łańcuchowane i migrowane wiersze.

```
SQL> CREATE TABLE temp_osoby
     AS SELECT * FROM osoby
        WHERE ROWID IN (SELECT HEAD_ROWID FROM CHAINED_ROWS
                        WHERE TABLE_NAME = 'OSOBY');
```

4. Usunięcie z tabeli OSOBY rekordów łańcuchowanych i migrowanych.

```
SQL> DELETE FROM osoby
     WHERE ROWID IN (SELECT HEAD_ROWID FROM CHAINED_ROWS
                     WHERE TABLE_NAME = 'osoby');
```

5. Wstawienie wierszy z tabeli TEMP_OSOBY do tabeli OSOBY.

```
SQL> INSERT INTO osoby SELECT * FROM temp_osoby;
```

6. Usunięcie tabeli tymczasowej TEMP_OSOBY.

```
SQL> DROP TABLE temp_osoby;
```

7. Usunięcie informacji zebranych w kroku 1 (z tabeli CHAINED_ROWS).

```
SQL> DELETE FROM CHAINED_ROWS WHERE TABLE_NAME = 'OSOBY';
```

8. Ponowne wykonanie polecenia ANALYZE - wyświetlone zostaną wówczas rekordy łańcuchowane. Wyeliminowanie łańcuchowania tych rekordów jest możliwe tylko poprzez zwiększenie rozmiaru bloków.

W pewnych sytuacjach nie można uniknąć łańcuchowania tj. w tabelach z kolumnami typu LONG lub długimi kolumnami CHAR lub VARCHAR2. Informacja o występowaniu w naszym systemie tego typu kolumn musi być wzięta pod uwagę, podczas ustalania rozmiaru bloku w fazie projektowania bazy.

Wielkość bloku jest ustalana parametrem DB_BLOCK_SIZE. Wyznaczenie optymalnego rozmiaru bloku pozwala efektywnie wykorzystać te bloki, a także na zminimalizowanie ilości operacji wejścia-wyjścia.

29.2.4. Unikanie dynamicznego zarządzania rozszerzeniami

Wraz ze wzrostem ilości danych w tabeli tworzone są nowe rozszerzenia, w których te dane mogą być przechowywane. Przydzielanie tych rozszerzeń jest procesem bardzo kosztownym, dlatego przy strojeniu wykorzystania urządzeń dyskowych nie można zapomnieć o redukowaniu liczby tych operacji.

Aby zmniejszyć liczbę operacji częstego przydzielania nowych rozszerzeń powinno się określić parametry składowania obiektów: `initial, next, minextents, pctincrease`.

Korzyścią stosowania dużych rozszerzeń jest zmniejszenie liczby operacji przydziału kolejnego rozszerzenia oraz ciągłość zaalokowanego miejsca na dysku, co przyśpiesza sekwencyjny odczyt wszystkich rekordów tabeli. Minusem tego rozwiązania mogą być problemy z przydzieleniem wystarczająco dużej, ciągłej przestrzeni na dysku. Rozwiązanie tych problemów wymaga rozszerzenia lub dodania pliku danych do przestrzeni tabel.

Aby wyświetlić liczbę rozszerzeń należy posłużyć się perspektywą systemową DBA_EXTENTS. Zapytanie oraz jego wynik mogą wyglądać następująco:

```
SELECT owner, segment_name, segment_type,
count(*) liczba_rozszerzeń
FROM dba_extents
GROUP BY owner, segment_name, segment_type;
```

OWNER	SEGMENT_NAME	SEGMENT_TYPE	liczba_rozszerzeń
SCOTT	AB_DATA_ROZP	INDEX	9
SCOTT	AB_DATA_ZAK	INDEX	9
SCOTT	AB_ET_FK_I	INDEX	10

Jeśli po kilkukrotnym wykonaniu powyższego zapytania zauważymy, iż liczba rozszerzeń któregoś obiektu dynamicznie wzrasta, wówczas należy rozważyć zwiększenie wartości parametrów składowania `next` oraz `pctincrease` tego obiektu.

29.2.5. Strojenie procesu DBWR

Proces DBWR dokonuje zapisu zmodyfikowanych bloków danych z bufora danych na dysk w następujących przypadkach:

a) **proces użytkownika stwierdzi, że lista LRU przekroczyła określony rozmiar;**
b) **proces użytkownika nie znajdzie wolnego bloku w buforze danych, a bufor taki jest wymagany do odczytu nowych danych na dysku;**
c) **upłyną … sekundy od ostatniego zapisu;**
d) **zostanie zgłoszony punkt kontrolny (`Checkpoint`).**

W przypadku podpunktu d) proces `DBWR` zapisuje wszystkie zmodyfikowane bloki bufora danych, natomiast w pozostałych podpunktach, ilość każdorazowo zapisywanych bloków określona jest przez parametr `DB_BLOCK_CHECKPOINT_BATCH`. Ustawienie wartości tego parametru na odpowiednio niewielką wartość zapobiega dyskryminacji pozostałych zadań realizowanych przez `DBWR`, natomiast wyższa wartość tego parametru umożliwia szybsze wykonanie punktu kontrolnego.

29.2.6. Strojenie segmentów wycofywania

We wcześniejszych wersjach systemu Oracle, spójność odczytu danych obsługiwana była przez mechanizm segmentów wycofania. Od wersji Oracle9i wprowadzony został nowy, automatyczny mechanizm zapewniający spójność odczytów, wycofywania zmian i odtwarzania bazy po awarii - SMU (*ang. System Management Undo*). W przypadku serwerów od Oracle9i zaleca się ustawienie ich w tryb pracy SMU, co znacząco ułatwia zarządzanie bazą danych i usuwa konieczność ręcznego strojenia segmentów wycofania jak to ma miejsce w trybie pracy RBU (ang. Rollback Segment Undo). Praca w trybie RBU jest możliwa ze względu na konieczność zachowania zgodności z poprzednimi wersjami systemu Oracle.

29.2.7. Strojenie serwera w trybie SMU- System Management Undo

Przestrzeń wycofania musi być na tyle duża, by była w stanie pomieścić całą informację związaną z wycofywaniem transakcji.

Przybliżony rozmiar tej przestrzeni możemy wyliczyć korzystając z wartości parametru `UNDO_RETENTION`, określającego jak długo dane mają pozostawać na dysku. Znając tą wartość oraz ilość generowanych danych wycofania na sekundę, można wyliczyć wielkość przestrzeni wycofania za pomocą wzoru:

```
Wielkość przestrzeni=czas przechowywania * ilość informacji w jednostce czasu
```

Przykładowo, jeśli czas przechowywania wynosi 200s, a system generuje 100 bloków informacji wycofania na sekundę, to otrzymamy:

```
Liczba bloków undo = (200s)*(100 bloków/s) = 20000 bloków
```

czyli wielkość przestrzeni wycofania wynosi 20000 razy wielkość bloku.

Mając „chodzący” system w trybie SMU możemy przejść do poprawienia jego wydajności. Statystyki opisujące pracę w trybie automatycznym możemy znaleźć w dynamicznej perspektywie V$UNDOSTAT, która składa się m.in. z następujących kolumn:

BEGIN_TIME	- Początek badanego przedziału czasowego.
END_TIME	- Koniec badanego przedziału czasowego.
UNDOBLKS	- Liczba bloków wycofania zużyta na potrzeby przechowywania informacji wycofania w tym przedziale.
SSOLDERRCNT	- Liczba błędów *snapshot too old*, która wystąpiła w trakcie trwania tego przedziału czasu.
NOSPACEERRCNT	- Liczba błędów *snapshot too old*, która wystąpiła dla tej instancji.

Jeśli w kolumnach zawierających błędy występuje jakaś wartość większa do zera, wówczas można skrócić czas przechowywania lub zwiększyć wielkość przestrzeni wycofania.

Domyślna wartość czasu przechowywania informacji służących do wycofywania transakcji jest zazwyczaj wystarczającą wartością dla systemów, w których nie wykonuje się długotrwałych zapytań. Jeśli jednak wiemy, że w naszym systemie wykonywane będą długotrwałe transakcje (często wymagające spójnego obrazu bazy przez cały czas) warto zastanowić się nad zwiększeniem tego czasu. Jeśli w trakcie wykonywania takiej transakcji wystąpi błąd *snapshot too old*, wówczas czas przechowywania należy zwiększyć.

29.2.8. Strojenie serwera w trybie RBU

Ze względu na zalecenie Oracla ustawienia trybu pracy serwrów w tryb SMU pominiemy ten punkt.

§ 29.3. Strojenie punktów kontrolnych

Punkty kontrolne zapewniają, że wszystkie „brudne bloki” znajdujące się w SGA zostaną zapisane na dysk. Ponieważ proces DBWR bazuje na algorytmie LRU, jest zatem możliwość wystąpienia sytuacji, w której często modyfikowane bloki nie zostaną zapisane na dysku.

Rozwiązaniem tego problemu jest stosowanie punktów kontrolnych.

Punkty te są generowane w następujących przypadkach:

- następuje przełączenie pliku dziennika powtórzeń;
- przekroczony zostanie czas określony parametrem konfiguracyjnym

  LOG_CHECKPOINT_TIMEOUT

  od poprzedniego punktu kontrolnego;
- przekroczona zostanie liczba zapisów określona parametrem konfiguracyjnym

  LOG_CHECKPOINT_INTERVAL

  od poprzedniego punktu kontrolnego;
- następuje przełączenie przestrzeni tabel w tryb wykonywania kopii zapasowej;
- przy przełączaniu pliku danych w tryb offline;
- przy zamykaniu bazy za pomocą poleceń SHUTDOWN NORMAL i SHUTDOWN IMMEDIATE;

- administrator może jawnie wywołać punkt kontrolny za pomocą polecenia

```
ALTER SYSTEM CHECKPOINT.
```

- inne.

Częste wykonywanie punktu kontrolnego przyspiesza operację odtwarzania instancji po awarii, jednak jednocześnie wprowadza dodatkowy narzut w postaci częstych zapisów zmodyfikowanych bloków danych na dysk.

Punkty kontrolne mogą jednocześnie obciążać proces LGWR, odpowiedzialny za zapisanie informacji kontrolnych do wszystkich plików danych i plików kontrolnych.

§ 29.4. Strojenie sortowań

Podczas wykonywania operacji sortowania wykorzystywany jest obszar pamięci PGA, którego wielkość określona jest parametrem konfiguracyjnym SORT_AREA_SIZE.

Jeżeli wielkość tego obszaru jest zbyt mała, wówczas występuje tendencja do przeprowadzania operacji sortowania na dysku. W takim przypadku należy zwiększyć wielkość tego obszaru.

Aby sprawdzić wykorzystanie pamięci i dysku do operacji sortowania można posłużyć się dynamiczną perspektywą V$SYSSTAT. Interesującymi statystykami tej perspektywy są Sorts(memory), zawierająca liczbę sortowań, które zmieściły się całkowicie w pamięci oraz Sorts(disk), zawierająca liczbę sortowań, które wymagały zapisania wyników pośrednich na dysku.

Przykładowe zapytanie może wyglądać następująco:

```
SELECT name, value
FROM v$sysstat
WHERE name IN ('sorts(memory)', 'sorts(disk)');
```

NAME	VALUE
sorts(memory)	19
sorts(disk)	0

Jeżeli liczba operacji sortowania na dysku jest zbyt duża, wówczas można rozważyć zwiększenie wartości parametru

SORT_AREA_SIZE.

Zwiększenie tego parametru skutkuje szybszym wykonywaniem sortowania i zmniejszeniem liczby operacji wejścia-wyjścia.

Po zwiększeniu obszaru sortowania można zmniejszyć wielkość obszaru pozostawionego za pomocą parametru

SORT_AREA_RETAINED_SIZE.

Parametr ten określa minimalną wielkość pamięci zwalnianej po wykonaniu operacji sortowania.

Jeżeli operacja sortowania wymaga większej ilości pamięci niż rozmiar określony wartością parametru SORT_AREA_SIZE, wówczas dane dzielone są na mniejsze części i każda z nich sortowana jest indywidualnie.

W procesie sortowania, obok obszaru sortowania, udział bierze także przestrzeń tymczasowa, która przechowuje pośrednie wyniki podczas sortowania dużych ilości danych. Właściwe skonfigurowanie przechowywania danych tymczasowych przyczynia się do wzrostu wydajności systemu.

Najbardziej efektywną metodą strojenia przestrzeni tymczasowej jest umieszczenie jej na bardzo szybkim nośniku danych. Zastosowanie tego rozwiązania znacząco przyspiesza dostęp do tabel tymczasowych.

Innym sposobem na zwiększenie wydajności operacji sortowania jest ustawienie wartości rozszerzenia przestrzenie tabel równej wartości parametru `SORT_AREA_SIZE`, dzięki czemu kolejne rozszerzenia nie będą przydzielane zbyt często.

Należy także ustawić parametr `PCTINCREASE` na wartość 0, a `INITIAL` i `NEXT` na wartość będącą wielokrotnością wartości `SORT_AREA_SIZE`. Trzeba upewnić się, aby wielkości te były dostatecznie duże, co zmniejszy liczbę alokacji nowych rozszerzeń.

§ 29.5. Zmniejszenie rywalizacji o semafory bufora dziennika powtórzeń

Dostęp do buforów dziennika powtórzeń jest kontrolowany przez dwa typy semaforów: alokujące oraz kopiujące.

Aby zapisać dane w buforze dziennika powtórzeń, proces użytkownika musi najpierw uzyskać semafor alokujący, przydzielający fragment bufora dziennika powtórzeń, do którego następnie kopiowane są stosowne informacje.

Kiedy proces użytkownika kończy kopiowanie danych do bufora, zwalniany jest ten semafor, zezwalając w ten sposób użycie tego semafora innym procesom użytkownika.

Ponieważ istnieje tylko jeden semafor alokacyjny, zatem jednocześnie do bufora zapisywać może tylko jeden proces użytkownika. Maksymalna wielkość danych, jaka może być zapisana po pojedynczym uzyskaniu semafora alokującego określona jest parametrem konfiguracyjnym

`LOG_SMALL_ENTRY_MAX_SIZE.`

Jeśli wielkość zapisu do bufora przekracza wartość parametru `LOG_SMALL_ENTRY_SIZE`, wówczas aby skopiować informacje do bufora, proces użytkownika musi uzyskać semafor kopiujący.

W systemach wieloprocesorowych takich semaforów może być więcej niż jeden. Pozwala to na wykonywanie sekwencyjnego zapisu do bufora dziennika powtórzeń, co przyczynia się do wzrostu wydajności.

Liczba semaforów kopiujących określona jest parametrem konfiguracyjnym

`LOG_SIMULTANEOUS_COPIES.`

W systemach jednoprocesorowych nie występują semafory kopiujące i na dostęp do bufora zezwala semafor alokujący.

Dostęp do semaforów może być wykonywany w jednym z dwóch trybów: w trybie z oczekiwaniem i w trybie natychmiastowym. Jeśli proces nie uzyskuje wyłącznego dostępu do semafora w trybie z

oczekiwaniem, to rozpoczyna oczekiwanie na jego zwolnienie. Jeżeli natomiast proces żądający wyłącznego dostępu do semafora w trybie natychmiastowym nie otrzyma dostępu, to kontynuuje swoje przetwarzanie.

Aby uzyskać informacje o stopniu rywalizacji o semafory bufora dziennika powtórzeń, należy posłużyć się dynamiczna perspektywą V$LATCH, która zawiera m.in. poniższe kolumny:

GETS	- ogólna liczba dostępu w trybie z oczekiwaniem
MISSES	- liczba nieudanych dostępów w trybie z oczekiwaniem
IMMEDIATE_GETS	- ogólna liczba dostępu w trybie natychmiastowym
IMMEDIATE_MISSES	- liczba nieudanych dostępów w trybie natychmiastowym

Przykładowe zapytanie może wyglądać następująco:

```
SELECT name, gets, misses, immediate_gets imm_gets,
                                    immediate_misses imm_misses
FROM v$latch
WHERE name IN ('redo allocation', 'redo copy');
```

NAME	GETS	MISSES	IMM_GETS	IMM_MISSES
redo allocation	563571	523	0	0
redo copy	2	2	23452	3

Stopień rywalizacji w każdym z tych przypadków określa się jako stosunek nieudanych dostępów do ogólnej liczby dostępów i w obu trybach nie powinien on przekraczać jednego procenta.

Aby zmniejszyć rywalizację o semafor alokujący, należy zmniejszyć wartość parametru

LOG_SMALL_ENTRY_MAX_SIZE,

dzięki czemu mniej procesów będzie blokowało ten semafor w czasie kopiowania.

Aby zmniejszyć rywalizację o semafor kopiujący, należy zwiększyć wartość parametru

LOG_SIMULTANEOUS_COPIES,

co spowoduje możliwość obsłużenia większej liczby operacji zapisu do bufora dziennika powtórzeń.

§ 29.6. Minimalizacja rywalizacji o listę wolnych bloków

Kolejnym miejscem, gdzie może wystąpić rywalizacja jest lista wolnych bloków. Listy te przechowują informacje związane z wolnymi blokami w buforze danych, w których jest miejsce na wstawienie danych.

Aby sprawdzić stopień rywalizacji o listy wolnych bloków można posłużyć się poniższym zapytaniem do dynamicznej perspektywy V$WAITSTAT.

```
SELECT class, count
FROM v$waitstat
WHERE class = 'free list';
```

CLASS	COUNT
free list	0

Jeśli liczba oczekiwań na listę przekracza 1% wszystkich żądań, wówczas należy zwiększyć liczbę takich list poprzez ponowne utworzenie tabeli ze zwiększoną wartością parametru FREELISTS. Liczbę wszystkich odwołań do listy możemy uzyskać sumując ilość odczytów bloków z perspektywy V$SYSSTAT.

```
SELECT SUM(value) Liczba_zadan
FROM v$sysstat
WHERE name in ('db block gets', 'consistent gets');
```

```
LICZBA_ZADAN
------------
2332
```

Rywalizacja o listy występuje głównie przy współbieżnym wykonywaniu przez wiele sesji operacji wstawiania `insert`, zatem optymalna liczba list wolnych bloków powinna być równa liczbie sesji równocześnie wstawiających dane do bazy.

SPIS ILUSTRACJI