"Billing" dla bazy danych Oracle

Jarosław Łagowski
iRM Polska
e–mail: jarek.lagowski@irm.at

Abstrakt.

Brama zakładu pracy. Setki pracowników przechodzą tędy codziennie w obydwie strony. Jak sprawdzić, czy Kowalski spędza w pracy 8 godzin dziennie? Proszę - oto lista wejść i wyjść, wykaz czasu pracy dla dowolnego pracownika. Każdy z nich ma obowiązek zarejestrować swoje przejście przez bramę poprzez proste urządzenie pomiaru czasu.

Faktura za telefon. Wysokość kwoty znacznie przekracza nasze najśmielsze oczekiwania. Jak sprawdzić kto zawinił, żona czy mąż, dzwoniąc do znajomych czy "surfując po internecie"? Proszę - oto wykaz połączeń, każda rozmowa z naszego aparatu jest rejestrowana w centrali i może być sprawdzona.

Administrator bazy danych może stanąć przed koniecznością odpowiedzi na podobne pytania. Ile czasu dany użytkownik pracował z bazą danych? W jakich godzinach? Co robił? Czy, i w jaki sposób modyfikował istotne dane? Odpowiedzi są tym bardziej ważne jeżeli mamy do czynienia z dostępem przez Internet lub coraz popularniejszym "outsourcing'iem" aplikacji.

Referat pokaże jak przy wykorzystaniu standardowych (darmowych!) mechanizmów przygotować "billing" dla bazy danych Oracle. Wykorzystane będą tradycyjne środki: polecenie AUDIT oraz wyzwalacze działające na poleceniach DML jak również właściwości Oracle 8*i*: „Log Miner" i wyzwalacze działające na zdarzeniach bazodanowych.

1. Rejestracja czasu pracy

Zacznijmy od rzeczy stosunkowo najprostszej: rejestru podłączeń do naszej bazy danych. Chcielibyśmy wiedzieć kto, kiedy i jak długo pracował w naszym systemie. Bieżące sesje można śledzić korzystając z tablic V$SESSION (ta tablica przyda się nam zresztą trochę później). Ale jak zapamiętać całą historię połączeń? Jak uchwycić moment zakończenia sesji?

1.1. Polecenie AUDIT

Tradycyjnym sposobem rejestracji czasu pracy w bazie danych Oracle, działającym dobrze już w wersji 6, jest polecenie AUDIT. Dzięki niemu możemy rejestrować a następnie raportować różne rodzaje aktywności użytkowników bazy danych.

1.1.1. Zakresy obserwacji wykonywanej poleceniem AUDIT

- Obserwacja polecenia (*statement auditing*) – zbiera informacje o wybranych typach poleceń (np. modyfikujących dane), niezależnie od obiektu na którym polecenie działało, obserwację można prowadzić dla wszystkich bądź dla wybranych użytkowników bazy danych.
- Obserwacja przywileju (*privilege auditing*) – zbiera informacje o wykorzystaniu wybranych przywilejów (np. przywileju do tworzenia nowych tablic), niezależnie od obiektu na którym polecenie działało, obserwację można prowadzić dla wszystkich bądź dla wybranych użytkowników bazy danych

- Obserwacja obiektu (*schema object auditing*) – zbiera informacje o wykonaniu konkretnego polecenia na konkretnym obiekcie (np. zapytania na wskazanej tablicy), obserwacja jest prowadzona dla wszystkich użytkowników bazy danych.

Rejestracja połączeń z bazą danych mieści się zarówno w zakresie obserwacji polecenia jak i przywileju.

1.1.2. Poziom obserwacji wykonywanej poleceniem AUDIT

Obserwacja bazy danych może gromadzić informacje mniej lub bardziej szczegółowo, biorąc pod uwagę:

- polecenia wykonane tylko prawidłowo, bądź tylko nieprawidłowo, bądź wszystkie,
- tylko pierwsze wystąpienie polecenia, bądź tylko pierwsze wystąpienie polecenia w sesji użytkownika, bądź wszystkie wystąpienia polecenia
- polecenia wykonane tylko przez wybranych użytkowników, bądź wszystkie wykonane polecenia

Rejestracja połączeń powinna obejmować wszystkie próby (również te nieudane), wykonywane przez wszystkich użytkowników.

1.1.3. Aktywacja obserwacji wykonywanej poleceniem AUDIT

Aktywacja obserwacji wykonywanej poleceniem AUDIT wymaga:

- ustawienia odpowiedniego parametru pliku startowego bazy danych,
- ustawienia zakresu i poziomu obserwacji poleceniem AUDIT

Parametr pliku startowego, który musimy ustawić to AUDIT_TRAIL. Dopuszczalne wartości parametru to:

- NONE – obserwacja wyłączona (wartość domyślna),
- TRUE – zapis obserwacji do bazy danych,
- OS – zapis obserwacji do pliku (tylko na wybranych platformach)

Parametr nie może być zmieniany dynamicznie, to znaczy, że nowa wartość parametru zadziała dopiero po kolejnym stracie bazy danych. Ustawienie zapisu obserwacji do bazy danych jest najwygodniejsze ze względu na możliwość późniejszego raportowania za pomocą polecenia SELECT. Zatem, wybrana linijka z pliku initX.ora powinna wyglądać następująco:

Przykład 1. Aktywacja obserwacji

```
audit_trail = true
```

Dezaktywacja obserwacji wymaga ustawienia parametru na wartość FALSE i ponownego startu bazy danych. Po restarcie bazy danych z parametrem ustawionym jak wyżej, pozostało nam włączyć żądaną opcję obserwacji. Polecenie AUDIT w postaci potrzebnej do rejestracji wszystkich podłączeń jest naprawdę proste:

Przykład 2. Ustawienie opcji obserwacji

```
SQL> audit session;

Obserwowanie zostało włączone.
```

I już – od tego momentu każde przejście (również nieudane) przez naszą „bramę do bazy danych” będzie rejestrowane. Wyłączenie obserwacji odbywa się za pomocą polecenia NOAUDIT.

1.1.4. Raportowanie

Opcje obserwacji pamiętane są w tablicy SYS.AUDIT$. Wyniki obserwacji gromadzone są w tablicy SYS.AUD$ (problemy związane z pielęgnacją tej tablicy będą omówione w punkcie 1.1.5). Ci, którzy lubią „teksty źródłowe" mogą czuć się uszczęśliwieni możliwością wykonywania zapytań bezpośrednio na tych tablicach. Mniej dociekliwi mogą zadowolić się zestawem perspektyw DBA_... które prezentują dane z ww. tablic w sposób bardziej przystępny.

Ustawione opcje obserwacji można odczytać w tablicach słownikowych:

- DBA_ STMT _AUDIT_OPTS – opcje obserwacji polecenia
- DBA_PRIV_AUDIT_OPTS – opcje obserwacji przywileju
- DBA_OBJ_AUDIT_OPTS – opcje obserwacji obiektu

Sprawdźmy efekt polecenia z przykładu 2.:

Przykład 3. Sprawdzenie opcji obserwacji

```
SQL> select *
  2    from DBA_STMT_AUDIT_OPTS;
USER_NAME                      PROXY_NAME                     AUDIT_OPTION                            SUCCESS    FAILURE
------------------------------ ------------------------------ --------------------------------------- ---------- ----------
                                                              CREATE SESSION                          BY ACCESS  BY ACCESS
```

Komentarz do przykładu: Kolumny USER_NAME i PROXY_NAME świadczą o tym, że można zawęzić obserwację połączeń. Fakt, że w kolumnie AUDIT_OPTION widzimy tylko „CREATE SESSION" nie przeszkadza w rejestracji zarówno połączeń jak i odłączeń. Wartości „BY ACCESS" w następnych dwóch kolumnach mówią o tym, że rejestrowane będzie każde zdarzenie. Dla rejestracji podłączeń (AUDIT SESSION) jest to jedyna możliwość. Dla innych opcji obserwacji można wybrać rejestrację „BY SESSION" – tylko pierwsze zdarzenie w sesji będzie notowane.

Wyniki obserwacji można odnaleźć w następujących tablicach słownikowych:

- DBA_AUDIT_TRAIL - pełna informacja, bez podziału na opcje obserwacji
- DBA_AUDIT_SESSION – rejestr podłączeń,
- DBA_AUDIT_STATEMENT – rejestr użycia poleceń GRANT, REVOKE, AUDIT, NOAUDIT oraz ALTER SYSTEM
- DBA_AUDIT_OBJECT – rejestr użycia obiektów i przywilejów zgodnie z ustawionymi opcjami obserwacji

I oto, za pomocą polecenia SELECT z DBA_AUDIT_SESSION uzyskujemy rejestr czasu pracy w naszej bazie danych.

Przykład 4. Wynik obserwacji podłączeń

```
SQL> select sessionid,
  2         to_char(timestamp,'YYYY.MM.DD HH24:MI:SS')   logon_time,
  3         to_char(logoff_time,'YYYY.MM.DD HH24:MI:SS') logoff_time,
  4         os_username,
  5         username,
  6         returncode
  7    from dba_audit_session
  8    order by timestamp;
  9
 SESSIONID LOGON_TIME          LOGOFF_TIME         OS_USERNAME          USERNAME             RETURNCODE
---------- ------------------- ------------------- -------------------- -------------------- ----------
      2134 2000.07.25 21:01:04 2000.07.25 21:01:15 JALA                 IOPT                          0
      2135 2000.07.25 21:01:15 2000.07.25 21:03:35 JALA                 ALFA                          0
      2136 2000.07.25 21:03:35 2000.07.25 21:19:51 JALA                 ALFA                          0
      2137 2000.07.26 09:57:00 2000.07.26 14:48:51 JALA                 IOPT                          0
      2138 2000.07.26 10:21:31 2000.07.26 15:30:17 JALA                 IOPT                          0
      2139 2000.07.26 10:21:47 2000.07.26 11:24:54 JALA                 IOPT                          0
      2162 2000.07.26 15:49:09                     JALA                 IOPT                          0
      2163 2000.07.26 16:07:37 2000.07.26 16:07:41 JALA                 IOPT                          0
      2164 2000.07.26 16:07:41                     JALA                 BETA                       1017
      2165 2000.07.26 16:07:46 2000.07.26 16:23:56 JALA                 BETA                          0
      2166 2000.07.26 16:23:56                     JALA                 GAMMA                         0
```

Komentarz do przykładu: SESSIONID jest unikalnym identyfikatorem sesji. Bieżąca sesja może sprawdzić swój identyfikator za pomocą funkcji standardowej USERENV('SESSIONID'). Dzięki temu identyfikatorowi będziemy mogli następnie łączyć prosty rejestr podłączeń z raportem szczegółowym o tym co dana sesja robiła. Identyfikator pozostaje unikalny w ramach jednej bazy danych. Jeżeli kolumna LOGOFF_TIME jest pusta to znaczy, że sesja jeszcze działa lub podłączenie było nieudane. Sesje przerwane przez administratora lub z powodu zamknięcia bazy danych uzyskują w LOGOFF_TIME czas zdarzenia, które wymusiło przerwanie sesji. Pomyślne podłączenie i brak problemów w trakcie działania sesji sygnalizuje wartość 0 w kolumnie RETURNCODE. Inna wartość to kod błędu ORA- sygnalizujący problem. W przykładzie, dla sesji 2164 wystąpił dosyć popularny błąd: „ORA-01017: nieprawidłowa nazwa użytkownika/hasło; odmowa rejestracji". Kolumny użyte w przykładzie nie są wszystkimi, dostępnymi w DBA_AUDIT_SESSION.

Warto wiedzieć, że istnieje pewna, bardzo ciekawa kolumna w tablicy DBA_AUDIT_TRAIL zamaskowana pod nieznaczącą nazwą: COMMENT_TEXT. Otóż zawarte w niej informacje nie są dostępne w DBA_AUDIT_SESSION a mogą okazać się pożyteczne. Dotyczą one sposobu i adresu z którego łączono się do bazy danych. Sprawdźmy klika sesji z przykładu 4.

Przykład 5. Dodatkowe informacje o sesjach

```
SQL> select sessionid,
  2         comment_text
  3    from dba_audit_trail
  4   where sessionid in ( 2138, 2164, 2134);

 SESSIONID COMMENT_TEXT
---------- --------------------------------------------------------------------------------------------------
      2134 Authenticated by: DATABASE
      2138 Authenticated by: DATABASE; Client address: (ADDRESS=(PROTOCOL=tcp)(HOST=192.168.4.81)(PORT=2615))
      2164 Authenticated by: DATABASE
```

Komentarz do przykładu: Sesja 2134 była pomyślnie zakończoną sesją lokalną. Sesja 2138 łączyła się wykorzystując protokół TCP/IP (Net8 lub SQL*Net 2). Sesja 2164 była sesją lokalną, która nie doszła do skutku z powodu błędu ORA-01017.

1.1.6. Pielęgnacja tablicy SYS.AUD$

Tablica AUD$ tworzona jest podobnie jak tablice słownikowe w przestrzeni tablic SYSTEM w schemacie SYS. Tym samym, znajduje się w grupie tablic, których „dotykanie" inaczej jak tylko poprzez SELECT jest zabronione. Na szczęście, stanowi ona jednak wyjątek – można (i należy) usuwać z niej rekordy. Administrator powinien zapewnić:

- Kontrolę rozmiaru tablicy AUD$
- Backup tablicy AUD$

Ponieważ AUD$ znajduje się w przestrzeni tablic SYSTEM (a dokumentacja stanowczo odradza przenoszenie tej tablicy w inne miejsce) nie wolno dopuścić aby w sposób niekontrolowany rosła zapychając (czy też rozpychając) tę najważniejszą przestrzeń w bazie danych Oracle. Zatem, należy okresowo czyścić tę tablicę ze starych rekordów. Ale, co zrobić, jeżeli będziemy chcieli sięgnąć do historii naszego dziennika „wejść i wyjść"? Otóż, jeżeli wcześniej nie zapewnimy mechanizmu archiwizacji tablicy AUD$ - nie będziemy mieli dostępu do usuniętych z niej rekordów, nawet jeżeli dysponujemy eksportem bazy danych sprzed momentu usunięcia starych rekordów. Eksport bazy danych nie obejmuje żadnych tablic użytkownika SYS. Z założenia, tenże użytkownik posiada tylko tablice słownikowe a te z kolei, automatycznie zapełniają się w przypadku importu pozostałych obiektów bazy danych.

Skutecznym, prostym i najczęściej stosowanym rozwiązaniem jest sporządzenie kopii tablicy AUD$ pod innym użytkownikiem, w innej przestrzeni tablic. Odświeżanie takiej kopii może odbywać się np. raz na dobę za pomocą prostej pętli:

Przykład 6. Pielęgnacja tablicy AUD$

```
begin
 for i in (select sessionid
             from sys.aud$
            where timestamp# < trunc(sysdate)
              and sessionid not in
                  ( select audsid
                      from v$session ) )
 loop
  insert into AUDIT_ARCH
  select *
    from sys.aud$
   where sessionid = i.sessionid;
  delete from sys.aud$
   where sessionid = i.sessionid;
  commit;
 end loop;
end;
/
```

Komentarz do przykładu: Sięgnięcie do tablicy V$SESSION pozwala nam uniknąć usunięcia rekordu dla działającej sesji. Zaproponowany warunek wybiera rekordy wczorajsze i starsze. Administrator powinien określić jak długą historię może przechowywać w tablicy oryginalnej. Pustą kopię tablicy AUD$ (w tym przypadku nazwaną AUDIT_ARCH) można przygotować za pomocą „create table AUDIT_ARCH *opcje_składowania* as select * from SYS.AUD$ where 1=2". Potwierdzanie transakcji po każdym przeniesionym wierszu pozwala nam uniknąć problemów z segmentami wycofania.

Tak przygotowana kopia tablicy AUD$ (na koncie użytkownika innego niż SYS) jest chroniona przez eksport bazy danych i tym samym może być czyszczona z możliwością powrotu do wersji zachowanej w czasie eksportu. Przeglądanie informacji zawartej w tablicy archiwalnej jest możliwe na dwa sposoby:

- czytanie wprost z tej tablicy (może być wspomagane perspektywami przygotowanymi na wzór DBA%AUDIT%)
- czytanie z SYS.AUD$ uprzednio załadowanej danymi z tablicy – kopii

1.2. Zdarzenia bazodanowe

Innym sposobem na rejestrację podłączeń do bazy danych jest przygotowanie wyzwalaczy działających na zdarzeniach bazodanowych. Jest to nowa możliwość, dostępna począwszy od wersji Oracle Server 8.1. Zdarzenia na poziomie bazy danych, które mogą być powodem do uruchomienia wyzwalacza są następujące:

- SERVERERROR – wystąpił błąd rejestrowany na poziomie bazy danych (niektóre błędy nie powodują uruchomienia wyzwalacza, np. „ORA-01034: Oracle not available" ☺)
- LOGON, LOGOFF – połączenie i rozłączenie z bazą danych
- STARTUP, SHUTDOWN – start i zamknięcie bazy danych

Oczywiście, aby rejestrować czas pracy należy wykorzystać zdarzenia LOGON i LOGOFF.

1.2.1. Porównanie z poleceniem AUDIT

Podstawową zaletą zastosowania wyzwalaczy w porównaniu z poleceniem AUDIT jest oczywiście możliwość oprogramowania zdarzenia. Dzięki temu można aktywnie kontrolować naszą „bramę zakładową" a nie tylko biernie rejestrować wejścia i wyjścia. Oto przykłady możliwości wyzwalaczy, niedostępne dla polecenia AUDIT:

- zebranie dodatkowej informacji z tablic słownikowych Oracle'a,

- rejestracja podłączeń według dowolnych warunków logicznych (np. tylko w określone dni tygodnia),
- odrzucanie podłączeń nie spełniających określonych warunków (np. połączenie z maszyny nie będącej na liście autoryzowanych)
- przesyłanie informacji (ostrzeżeń) za pomocą strumieni (DBMS_PIPE) lub kolejek (Advanced Queueing Option)

Podstawową wadą zastosowania wyzwalaczy w porównaniu z poleceniem AUDIT jest oczywiście konieczność oprogramowania zdarzenia. Administrator czy programista przygotowujący wyzwalacz działający na zdarzeniach bazodanowych musi rozwiązać podstawowe problemy:

- jak napisać program aby nie wywołać skutków ubocznych,
- gdzie przesyłać informację o zdarzeniu,
- jak nie dopuścić do zafałszowania informacji

Dodatkową wadą rejestracji czasu pracy za pomocą wyzwalaczy jest działanie wyzwalacza tylko „AFTER LOGON". Tym samym, nie można wychwycić prób nieudanych podłączeń.

1.2.2. Przygotowanie do zbierania informacji

Wyzwalacze działające na zdarzeniach bazodanowych mogą korzystać ze specjalnych funkcji zwracających atrybuty zdarzenia. I tak, zdarzenie AFTER LOGON udostępnia następujące atrybuty:

- ora_sysevent - dokładnie 'LOGON',
- ora_login_user – nazwa użytkownika bazy danych,
- ora_instance_num – numer instancji przez którą użytkownik łączy się z bazą danych,
- ora_database_name – nazwa bazy danych,
- ora_client_ip_address – adres z którego mam miejsce połączenie jeżeli wykorzystywany jest protokół TCP/IP

Zdarzenie BEFORE LOGOFF oferuje o jeden atrybut mniej – nie udostępnia „ora_client_ip_address". Jak widać informacja dostarczana przez atrybuty zdarzenia nie jest zbyt obfita. Musimy zatem zadbać aby wyzwalacz mógł zarejestrować nieco więcej danych.

Bardzo bogatym źródłem wiedzy o właściwościach sesji (połączenia) jest tablica V$SESSION. Cóż, kiedy jest ona dostępna tylko dla administratorów. A przecież pamiętać należy, że wyzwalacz wykonywany jest z prawami sesji powodującej zdarzenie. Dodatkowa trudność, nawet dla administratora może polegać na ustaleniu, która z sesji widzianych w V$SESSION jest moją sesją? Poniższy przykład jest jedną z możliwości rozwiązania tego problemu:

Przykład 7. Udostępnienie informacji o własnej sesji

```
SQL> connect sys
Podaj hasło:
Połączony.
SQL> create or replace view session_info
  2   as
  3  select *
  4    from v$session
  5   where sid = (select distinct
  6                       sid
  7                  from sys.v$mystat);
Perspektywa została utworzona.
SQL> create public synonym session_info for session_info;
Synonim został utworzony.
SQL> grant select on session_info to public;
Przyznanie uprawnień zakończone powodzeniem.
```

Komentarz do przykładu: Perspektywa V$mystat pokazuje statystyki dla bieżącej sesji. Niestety, standardowo dostępna jest tylko dla użytkownika SYS. Dlatego też, powyższe polecenia wykonywane są jako SYS. Korzystając z cudzego obiektu składowanego (np. perspektywa lub pakiet) użytkownik potrzebuje uprawnienia tylko na tenże obiekt. Wszelkie wewnętrzne odwołania realizowane są domyślnie z prawami właściciela

obiektu składowanego (w wersji 8.1 pojawiła się możliwość uruchamiania obiektów składowanych z prawami wołającego). Dlatego też wystarczy nadać uprawnienia na perspektywę końcową – SESSION_INFO.

Korzystając z perspektywy SESSION_INFO przygotowanej w przykładzie 7, każda sesja może „zobaczyć" swoje i tylko swoje parametry tak, jak są pokazane w V$SESSION. Kolejnym krokiem będzie przygotowanie miejsca składowania informacji.

Przykład 8. Przygotowanie tablicy gromadzącej historię połączeń

```
SQL> connect system
Podaj hasło:
Połączony.
SQL> create table session_hist
  2  ( sid           number,
  3    serial#       number,
  4    audsid        number,
  5    username      varchar2(30),
  6    osuser        varchar2(30),
  7    machine       varchar2(64),
  8    terminal      varchar2(16),
  9    program       varchar2(64),
 10    logon_time    date,
 11    logoff_time   date,
 12    ip_address    varchar2(64) )
 13  tablespace tools;
Tabela została utworzona.
SQL> create public synonym session_hist for session_hist;
Synonim został utworzony.
SQL> grant select,insert,update(logoff_time) on session_hist to public;
Przyznanie uprawnień zakończone powodzeniem.
```

Komentarz do przykładu: Tworząc tablice nie powinniśmy korzystać z konta SYS (porównaj punkt 1.1.6). Tworzenie tablicy na koncie SYSTEM też nie jest wskazane ale przykład jest realizowany na testowej bazie danych ☺. Jak zabezpieczyć SESSION_HIST przed fałszowaniem? Zostawmy to jako pracę domową dla bardziej dociekliwych.

A oto i zestaw dwóch, prostych wyzwalaczy rejestrujących podłączenia i rozłączenia z bazą danych:

Przykład 9. Wyzwalacze rejestrujące „wejścia i wyjścia"

```
SQL> create or replace trigger after_logon
  2  after logon on database
  3   declare
  4    ip_  varchar2(30);
  5   begin
  6    ip_ := ORA_CLIENT_IP_ADDRESS;
  7    insert into session_hist
  8    ( SID,        SERIAL#,
  9      AUDSID,     USERNAME,    OSUSER,
 10      MACHINE,    TERMINAL,    PROGRAM,
 11      LOGON_TIME, LOGOFF_TIME, IP_ADDRESS )
 12    select SID,
 13           SERIAL#,
 14           AUDSID,
 15           USERNAME,
 16           OSUSER,
 17           MACHINE,
 18           TERMINAL,
 19           PROGRAM,
 20           LOGON_TIME,
 21           NULL,
 22           ip_
 23      from session_info;
 24    if user = 'ALFA'
 25     then
 26      raise_application_error( -20001, 'Ciebie nie lubimy!');
 27    end if;
 28   end;
 29  /
Wyzwalacz został utworzony.
SQL>
SQL> create or replace trigger before_logoff
  2  before logoff on database
  3   begin
  4    update session_hist
  5       set logoff_time = sysdate
  6     where        (sid, serial#, audsid) =
  7           (select sid, serial#, audsid
  8              from session_info );
  9   end;
 10  /
Wyzwalacz został utworzony.
```

Komentarz do przykładu: Atrybut ORA_CLIENT_IP_ADDRESS użyty bezpośrednio w poleceniu INSERT zwraca NULL. Wyzwalacz AFTER_LOGON „nie wpuści" do bazy danych użytkownika ALFA. Sesja otworzona z „AS SYSDBA" (lub „INTERNAL") otrzymuje AUDSID = 0 (i nie jest rejestrowana przez AUDIT).

Jednoznaczna identyfikacja sesji jest zatem możliwa dopiero po użyciu trzech kolumn: SID, SERIAL# i AUDSID.

Wygenerujmy teraz kilka wpisów do tablicy SESSION_HIST

Przykład 10. Połączenia śledzone przez wyzwalacze

```
SQL> connect iopt
Podaj hasło:
Połączony.
SQL> connect beta/beta@iopt
Połączony.
SQL> connect sys as sysdba
Podaj hasło:
Połączony.
SQL> connect alfa/alfa
ERROR:
ORA-00604: wystąpił błąd na poziomie 1 rekurencyjnego SQL
ORA-20001: Ciebie nie lubimy!
ORA-06512: przy linia 24
Ostrzeżenie: Nie ma już połączenia z ORACLE.
SQL> connect gamma/gamma@iopt
Połączony.
```

Sprawdźmy zapisy w tablicy rejestrującej:

Przykład 11. Wynik rejestracji prowadzonej przez wyzwalacze

```
SQL> select to_char(LOGON_TIME,'YYYY.MM.DD HH24:MI:SS') LOGON_TIME,
  2         to_char(LOGOFF_TIME,'YYYY.MM.DD HH24:MI:SS') LOGOFF_TIME,
  3         USERNAME,
  4         OSUSER,
  5         MACHINE,
  6         IP_ADDRESS,
  7         PROGRAM
  8    from session_hist
  9   order by logon_time;

LOGON_TIME          LOGOFF_TIME         USERNAME OSUSER MACHINE     IP_ADDRESS   PROGRAM
------------------- ------------------- -------- ------ ----------- ------------ -----------
2000.07.28 16:37:28 2000.07.28 16:37:37 IOPT     JALA   IRM\JAREK1               SQLPLUS.EXE
2000.07.28 16:37:38 2000.07.28 16:37:51 BETA     JALA   IRM\JAREK1  192.168.4.81 SQLPLUS.EXE
2000.07.28 16:37:51 2000.07.28 16:38:01 SYS      JALA   IRM\JAREK1               SQLPLUS.EXE
2000.07.28 16:38:12                     GAMMA    JALA   IRM\JAREK1  192.168.4.81 SQLPLUS.EXE
```

Komentarz do przykładu: Odrzucona sesja użytkownika ALFA nie pozostawiła po sobie śladu. Sesje korzystające z połączenia Net8 zapisały swój adres. Sesja użytkownika GAMMA jeszcze trwa, lub ... sesja została przerwana w sposób gwałtowny (np. zamknięcie bazy danych z opcją IMMEDIATE lub ABORT).

1.3. Podsumowanie

Dla celów rejestracji i tylko rejestracji podłączeń do naszej bazy danych zdecydowanie polecam polecenie AUDIT. Jest prostsze w użyciu i dostarcza pełnej informacji, również o połączeniach nieudanych. Wyzwalacze działające na zdarzeniach bazodanowych należy traktować raczej jako element wspomagający, np. do filtrowania podłączeń. Niezależnie od zastosowanego sposobu, pamiętać trzeba o obsłudze bieżącej i backupie tablic z historią połączeń. Jest to przecież „billing" naszej bazy danych. Nie bez znaczenia pozostaje również fakt przygotowania odpowiednich indeksów dla naszych tablic historycznych. Bez nich, nie będzie możliwe efektywne raportowanie z tablic, które potencjalnie mogą przechowywać wiele tysięcy rekordów (ale to już temat na inny referat).

2. Rejestracja zmian danych

Rejestracja zmian w bazie danych Oracle jest zawsze włączona. Każda, potwierdzona zmiana musi znaleźć się w plikach dziennika powtórzeń (wyjątkiem są operacje z opcją NOLOGGING – kto używa tej opcji jest sam sobie winien w przypadku awarii). Aż do wersji 8.1, Oracle nie udostępniał żadnego narzędzia do oglądania zawartości dzienników w formie zrozumiałej dla człowieka. Począwszy od tej wersji administrator ma do dyspozycji narzędzie o nazwie Log Miner. Za pomocą tegoż narzędzia, zawartość dzienników powtórzeń (zarówno „on-line" jak i archiwalnych) jest dostępna do raportowania za pomocą SELECT. Zatem, czy potrzeba czegoś jeszcze? Tak. Potrzeba selektywnej rejestracji zmian. Selektywna rejestracja zmian pozwala

następnie łatwiej i szybciej raportować. Zdarza się, że w aplikację wbudowane są mechanizmy wypełniające przygotowane tablice zmian. Jak natomiast zorganizować selektywną rejestrację zmian w sposób zautomatyzowany, niezależny od aplikacji? Tradycyjnym sposobem (dostępnym od wersji 7.0) jest użycie wyzwalaczy działających na poleceniach DML na wybranych tablicach. Wymaga to trochę pracy ale, jak będę starał się pokazać, nie jest takie trudne.

2.1. Polecenie AUDIT

W rejestracji zmian może nam pomóc znane już polecenie AUDIT. Pozwala on ustawić śledzenie poleceń na wybranym obiekcie. Zatem, można sobie zażyczyć rejestracji każdego INSERT, UPDATE lub DELETE na wybranych tablicach. Niestety, informacja zebrana w ten sposób ogranicza się do stwierdzenia, że owszem, dany użytkownik na danej tablicy dokonał zmian danych. Nie wiadomo natomiast jakie dane zostały wstawione, usunięte lub zmienione. Jeżeli ten poziom informacji jest wystarczający to zachęcam do zgłębienia składni polecenia AUDIT i wybrania odpowiednich opcji. Aktywacja, start i raportowanie śledzenia pozostaje takie samo jak w przypadku śledzenia podłączeń. Wszyscy zainteresowani rejestracją wartości zmienianych danych, muszą użyć innych sposobów.

2.2. Wyzwalacze na poleceniach DML

Natura wyzwalacza została przedstawiona przy okazji śledzenia podłączeń do bazy danych. Jeżeli chcemy rejestrować wartość zmienianych danych możemy użyć wyzwalaczy działających na poleceniach DML (*Data Manipulation Language*). W treści, programista może użyć specyfikacji :NEW.*nazwa_kolumny* i :OLD.*nazwa_kolumny*. I to właśnie czyni użycie wyzwalaczy DML niezastąpionymi w rejestracji zmian danych. Co musimy wyspecyfikować:

- tablicę, którą chcemy śledzić,
- operację, którą chcemy śledzić – INSERT, UPDATE, DELETE
- kiedy chcemy uruchamiać wyzwalacz – BEFORE lub AFTER
- czy wyzwalacz ma działać ogólnie, na poziomie operacji, czy też dla każdego, zmienianego wiersza z osobna (FOR EACH ROW)

Dodatkowo oczywiście, należy przygotować tablicę rejestrującą zmiany. Tak naprawdę to powinny być to przynajmniej dwie tablice. Pierwsza, rejestrująca ogólne własności sesji modyfikującej dane i druga, pamiętająca wartości zmienianych kolumn. Poniżej znajdziemy przykład takich tablic:

Przykład 12. Tablice do rejestracji zmian danych

```
SQL> create table data_hist
  2  ( entry_id      number,
  3    timestamp     date,
  4    sid           number,
  5    serial#       number,
  6    audsid        number,
  7    username      varchar2(30),
  8    osuser        varchar2(30),
  9    machine       varchar2(64),
 10    terminal      varchar2(16),
 11    program       varchar2(64),
 12    table_owner   varchar2(30),
 13    table_name    varchar2(30),
 14    action        varchar2(1) )
 15  tablespace tools;
Tabela została utworzona.
SQL> create table data_hist_det
  2  ( entry_id      number,
  3    column_name   varchar2(30),
  4    old_value     varchar2(2000),
  5    new_value     varchar2(2000) )
  6  tablespace tools;
Tabela została utworzona.
SQL> create sequence data_hist_seq;
Sekwencja została utworzona.
```

Komentarz do przykładu: Zestaw początkowych kolumn tablicy DATA_HIST jest taki sam jak w tablicy SESSION_HIST. Udostępni to pełną informację o sesji modyfikującej dane a także pozwoli, w razie potrzeby,

połączyć tablice rejestrujące sesje i zmiany. Kolumna ACTION będzie przechowywać kod akcji: I – INSERT, U – UPDATE, D – DELETE. Sekwencja DATA_HIST_SEQ będzie generować unikalny klucz dla tablic rejestrujących zmiany. Oczywiście należy pamiętać o nadaniu odpowiednich uprawnień, przygotowaniu synonimów i zabezpieczeniu przed fałszowaniem.

Stworzenie wyzwalacza śledzącego zmiany może być w znacznym stopniu zautomatyzowane. Może odbywać się to na poziomie aplikacji, która wygeneruje odpowiedni skrypt do tworzenia wyzwalacza. Dzięki nowej właściwości Oracle 8.1 – wbudowanego, dynamicznego SQL (polecenie EXECUTE IMMEDIATE) jest to nawet prostsze. Poniżej, prezentowany jest przykład rejestrowania polecenia UPDATE dla dobrze znanej tablicy użytkownika SCOTT: DEPT.

Przykład 13. Wyzwalacz rejestrujący UPDATE na tablicy SCOTT.DEPT

```
SQL> create or replace trigger SCOTT_DEPT_U
  2    before UPDATE on SCOTT.DEPT for each row
  3  declare
  4   info_      session_info%rowtype;
  5   entry_id_  number;
  6  begin
  7   select *
  8     into info_
  9     from session_info;
 10   select data_hist_seq.nextval
 11     into entry_id_
 12     from dual;
 13
 14   insert into data_hist
 15   ( ENTRY_ID,   TIMESTAMP,
 16     SID,        SERIAL#,     AUDSID,
 17     USERNAME,   OSUSER,
 18     MACHINE,    TERMINAL,    PROGRAM,
 19     TABLE_NAME, TABLE_OWNER, ACTION )
 20   values
 21   ( entry_id_,      sysdate,
 22     info_.sid,      info_.serial#,  info_.audsid,
 23     info_.username, info_.osuser,
 24     info_.machine,  info_.terminal, info_.program,
 25     'DEPT',     'SCOTT',     'U');
 26
 27   insert into data_hist_det
 28   ( ENTRY_ID,  COLUMN_NAME,
 29     OLD_VALUE, NEW_VALUE )
 30   values
 31   ( entry_id_,   'DEPTNO',
 32     substr(to_char(:old.DEPTNO),1,2000),
 33     substr(to_char(:new.DEPTNO),1,2000) );
 34
 35   insert into data_hist_det
 36   ( ENTRY_ID,  COLUMN_NAME,
 37     OLD_VALUE, NEW_VALUE )
 38   values
 39   ( entry_id_,  'DNAME',
 40     substr(:old.DNAME,1,2000),
 41     substr(:new.DNAME,1,2000) );
 42
 43   insert into data_hist_det
 44   ( ENTRY_ID,  COLUMN_NAME,
 45     OLD_VALUE, NEW_VALUE )
 46   values
 47   ( entry_id_, 'LOC',
 48     substr(:old.LOC,1,2000),
 49     substr(:new.LOC,1,2000) );
 50
 51  end;
 52  /

Wyzwalacz został utworzony.
```

Komentarz do przykładu: Pozostałe zdarzenia mogą być rejestrowane przez analogiczne wyzwalacze BEFORE INSERT i BEFORE DELETE. Wykorzystywana jest perspektywa SESSION_INFO przygotowana w przykładzie 7. Wstawiając nowe i stare wartości kolumn do tablicy rejestrującej należy pamiętać o konwersji danych na tekst oraz ograniczeniu na długość tego tekstu.

Teraz, pozostało sprawdzić działanie rejestratora.

Przykład 14. Zmiany danych

```
SQL> connect gamma/gamma@iopt
Połączony.
SQL> update dept
  2     set loc = 'WARSZAWA'
  3   where deptno = 10;
1 wiersz został zmodyfikowany.
SQL> commit;
Zatwierdzanie zostało ukończone.
SQL> connect beta/beta
Połączony.
```

```
SQL> insert into dept
  2  values ( 50, 'NEW', 'ZAKOPANE');
1 wiersz został utworzony.
SQL> commit;
Zatwierdzanie zostało ukończone.
SQL> connect scott/tiger@iopt
Połączony.
SQL> delete from dept
  2   where deptno = 50;
1 wiersz został usunięty.
SQL> commit;
Zatwierdzanie zostało ukończone.
```

Przykład 15. Raport ze zmian danych

```
SQL> select to_char(A.TIMESTAMP,'YYYY.MM.DD HH24:MI:DD') TIMESTAMP,
  2         A.USERNAME,
  3         decode(A.ACTION,'I','INSERT','D','DELETE','U','UPDATE') ACTION,
  4         A.TABLE_OWNER||'.'||A.TABLE_NAME  CHANGED_TABLE,
  5         B.COLUMN_NAME,
  6         B.OLD_VALUE,
  7         B.NEW_VALUE
  8    from data_hist     A,
  9         data_hist_det B
 10   where A.entry_id = B.entry_id
 11   order by
 12         A.timestamp;

TIMESTAMP           USERNAME   ACTION CHANGED_TABLE COLUMN_NAME          OLD_VALUE            NEW_VALUE
------------------- ---------- ------ ------------- -------------------- -------------------- ------------
2000.07.28 19:23:28 GAMMA      UPDATE SCOTT.DEPT    DEPTNO               10                   10
                                                    DNAME                ACCOUNTING           ACCOUNTING
                                                    LOC                  NEW YORK             WARSZAWA
2000.07.28 19:24:28 BETA       INSERT SCOTT.DEPT    DEPTNO                                    50
                                                    DNAME                                     NEW
                                                    LOC                                       ZAKOPANE
2000.07.28 19:25:28 SCOTT      DELETE SCOTT.DEPT    DEPTNO               50
                                                    DNAME                NEW
                                                    LOC                  ZAKOPANE
```

2.2.1 Kilka uwag

Niezmiernie ważną sprawą jest ograniczenie rejestrowanych zmian. Gromadzenie zbyt dużej ilości informacji powoduje odczuwalne obciążenie systemu i dalsze problemy z raportowaniem. Oto kilka wskazówek jak ograniczyć ilość danych w tablicach rejestrujących zmiany.

- Gromadzenie informacji o zmianach danych powinno być ograniczone tylko do najważniejszych tablic.
- W przypadku rejestracji polecenia UPDATE nie warto pamiętać wartości kolumn nie zmienionych.
- W przypadku rejestracji poleceń INSERT i DELETE nie warto pamiętać kolumn z wartością NULL.
- Wyzwalacz na poleceniu UPDATE może być uruchamiany tylko przy zmianach wyznaczonych kolumn (patrz składania CREATE TRIGGER)
- Każdy wyzwalacz może mieć zdefiniowany warunek logiczny, którego spełnienie warunkuje uruchomienie wyzwalacza (patrz składania CREATE TRIGGER)

Zdanie powtórzone z punktu 1.3: Nie bez znaczenia pozostaje również fakt przygotowania odpowiednich indeksów dla naszych tablic historycznych. Bez nich, nie będzie możliwe efektywne raportowanie z tablic, które potencjalnie mogą przechowywać wiele tysięcy rekordów (ale to już temat na inny referat).

2.3. Log Miner

Jak już to było powiedziane, Log Miner jest to stosunkowo nowe narzędzie, pozwalające oglądać zawartość plików dziennika powtórzeń za pomocą polecenia SELECT. Dotyczy to zarówno dzienników bieżących („on-line") jak i archiwalnych, jeżeli baza danych działa w trybie archiwizacji. I tu dochodzimy do bardzo ważnego spostrzeżenia. Log Miner jako narzędzie do śledzenia zmian danych jest bezużyteczny jeżeli baza danych nie działa w trybie archiwizacji. A to dlatego, że bieżące dzienniki zapisywane są w cyklu, ciągle na nowo. Zatem, opierając się tylko na

dziennikach bieżących nie jesteśmy w stanie, w ogólnym przypadku, zapewnić raportowania zmian które miały miejsce nawet nie dalej jak np. godzinę temu.

Z kolei, dysponując dziennikami archiwalnymi, możemy potrzebować dużo czasu aby dotrzeć do informacji o interesujących nas zmianach. Dzieje się tak dlatego, że mimo użycia polecenia SELECT, tak naprawdę działamy na plikach sekwencyjnych. Ponieważ dzienniki gromadzą wszystkie zmiany w bazie danych, może to oznaczać potrzebę „przekopania" kilkuset megabajtów aby odnaleźć jedną transakcję. Z reguły nie wiemy kiedy dokładnie transakcja miała miejsce, a np. podejrzewamy, że wczoraj – Log Miner musi przejrzeć wszystkie, wczorajsze pliki dziennika.

Należy traktować zatem Log Miner'a jako narzędzie wspomagające a nie podstawowe w zadaniu raportowania zmian danych. Log Miner jest z kolei niezastąpiony, jeżeli chcemy uzyskać informacje o zmianach, które nie były śledzone wyzwalaczami. W poprzednich wersjach Oracle, administrator był „bez szans" wobec takiego zadania. Teraz, korzystając z wersji 8.1, może powiedzieć: „To potrwa, ale da się zrobić". Log Miner nie jest narzędziem typu SQL*Plus, które uruchamiamy i „jest". Tutaj, mamy do czynienia z zestawem pakietów i perspektyw, które odpowiednio użyte spełnią nasze oczekiwania. Nie będę przedstawiał specyfikacji tychże pakietów i perspektyw - ciekawych odsyłam do dokumentacji. Chciałbym natomiast pokazać w kolejnych punktach krok po kroku: przygotowanie i użycie Log Minera.

2.3.1 Parametr pliku startowego

Aby w pełni korzystać z usług Log Minera powinniśmy przygotować specjalny plik słownikowy. Co to takiego – o tym w następnym punkcie. Plik ten powstaje w katalogu wskazywanym parametrem **utl_file_dir** pliku init.ora. Jeżeli ten paramter nie jest ustawiony dla naszej bazy danych, należy czym prędzej poprawić to niedopatrzenie i ponownie wystartować bazę danych:

Przykład 16. Parametr init.ora

```
utl_file_dir                    = d:\OraDir\iOPT
```

2.3.2 Przygotowanie słownika

Specjalny plik – słownik powinien być przygotowany przed właściwym skorzystaniem z usług Log Minera. Słownik ten, budowany jest na podstawie aktualnego Słownika Danych w bazie danych Oracle. Bez tego wspomagającego pliku, Log Miner będzie nam prezentował np. nazwy kolumn w postaci heksadecymalnej a nie jest to sposób szczególnie miły dla oka. Słownik powinien być odświeżany po każdej zmianie w strukturze danych. Dla produkcyjnych baz danych, może to oznaczać jednokrotną generację pliku słownika.

Budowa słownika jest realizowana przez procedurę BUILD pakietu DBMS_LOGMNR_D (patrz przykład 17).

2.3.3 Przygotowanie listy plików dziennika do przeglądania

Dokąd nie przygotujemy listy plików dziennika, które chcemy przeglądać, wyniki sięgania do tablicy Log Minera będą puste. Pierwszy element listy jest jednocześnie początkiem nowej listy i jest dodawany procedurą ADD_LOGFILE z pakietu DBMS_LOGMNR z parametrem NEW. Kolejne pliki dodajemy wykorzystując tę samą procedurę, tym razem z parametrem ADDFILE (patrz przykład 17).

2.3.4 Aktywacja

Teraz jesteśmy gotowi do aktywacji sesji Log Minera. Odbywa się to za pomocą procedury START_LOGMNR z pakietu DBMS_LOGMNR. Pożytecznym (ale nieobowiązkowym) parametrem procedury jest pełna nazwa (łącznie z katalogiem) pliku słownikowego. Z innych parametrów warto wspomnieć o „startTime" i „endTime". Ich wartości domyślne są przygotowane przez uroczych Amerykanów jak następuje:

startTime IN DATE default TO_DATE('01-jan-1988','DD-MON-YYYY'),

endTime IN DATE default TO_DATE('01-jan-2988','DD-MON-YYYY'),

Zatem albo należy wyspecyfikować je dokładnie, albo przed wywołaniem procedury zmienić język sesji na AMERICAN.

Poniżej, zaprezentowany jest kompletny przykład przygotowujący Log Minera do prześledzenia bieżących dzienników:

Przykład 17. Przygotowanie Log Miner'a do przeglądania dzienników bieżących

```
SQL> declare
  2
  3    logmnr_dir_    varchar2(200);
  4    logmnr_file_   varchar2(30);
  5    redo_log_         varchar2(200);
  6    start_date_    date;
  7    stop_date_     date;
  8    new_list_         boolean := TRUE;
  9
 10  begin
 11
 12  -- Prepare new LogMiner Dictionary file
 13    select value
 14      into logmnr_dir_
 15      from v$parameter
 16     where name = 'utl_file_dir';
 17
 18    logmnr_file_ := 'lm'||to_char(sysdate,'MMDD')||'.dic';
 19
 20    sys.dbms_logmnr_d.build(logmnr_file_, logmnr_dir_);
 21
 22  -- Prepare on line redo log list
 23    for i in (select group#
 24                from v$log  )
 25    loop
 26
 27     select member
 28       into redo_log_
 29       from v$logfile
 30      where nvl(status,'OK') != 'INVALID'
 31        and group# = i.group#
 32        and rownum = 1;
 33
 34     if new_list_
 35     then
 36      sys.dbms_logmnr.add_logfile( redo_log_, sys.dbms_logmnr.new);
 37      new_list_ := FALSE;
 38     else
 39      sys.dbms_logmnr.add_logfile( redo_log_, sys.dbms_logmnr.addfile);
 40     end if;
 41
 42   end loop;
 43
 44  -- Start LogMiner
 45
 46   sys.dbms_logmnr.start_logmnr(DictFileName =>
 47                                logmnr_dir_||'\'||logmnr_file_);
 48
 49  end;
 50
 51  /
PL/SQL procedure successfully completed.
```

Komentarz do przykładu: Przygotowanie słownika zajmuje stosunkowo najwięcej czasu, zatem, jeżeli struktura danych nie zmieniła się – nie warto generować nowego słownika za każdym razem. Dla chętnych, proponuję przekształcenie powyższego przykładu tak, aby przygotować Log Minera do śledzenia wczorajszych dzienników archiwalnych. Podpowiedź : należy skorzystać z tablicy V$ARCHIVED_LOG.

2.3.4 Raportowanie

Tablica (perspektywa), która po wyżej zaprezentowanych krokach jest gotowa do użycia w celu raportowania zawartości dzienników nazywa się V$LOGMNR_CONTENTS. Pełną specyfikację, można odnaleźć w dokumentacji, ja zachęcam do prześledzenia kolejnego przykładu:

Przykład 18 Raportowanie za pomocą Log Miner'a

```
SQL> select to_char(timestamp,'YYYY.MM.DD HH24:MI:SS') TimeStamp,
  2         seg_owner,
```

```
  3         seg_name,
  4         seg_type,
  5         table_space,
  6         operation,
  7         username,
  8         session_info,
  9         sql_redo,
 10         sql_undo
 11    from v$logmnr_contents
 12   where operation != 'INTERNAL'
 13     and seg_owner  = 'SCOTT'
 14     and seg_name   = 'DEPT'
 15   order by
timestamp;


TIMESTAMP           SEG_OWNER                      SEG_NAME                       SEG_TYPE TABLE_SPACE
------------------- ------------------------------ ------------------------------ ---------- ----------
OPERATION                      USERNAME
------------------------------ ----------
SESSION_INFO
--------------------------------------------------------------------------------------------------------------------

SQL_REDO
--------------------------------------------------------------------------------------------------------------------

SQL_UNDO
--------------------------------------------------------------------------------------------------------------------

2000.07.28 18:26:15 SCOTT                          DEPT                                    2 USERS
INSERT                         SCOTT
LoginUserName = SCOTT, ClientInfo = , OsUserName = JALA, MachineName = IRM\JAREK1
insert into SCOTT.DEPT(DEPTNO,DNAME,LOC) values (10,'ACCOUNTING','NEW YORK');
delete from SCOTT.DEPT where DEPTNO = 10 and DNAME = 'ACCOUNTING' and LOC = 'NEW YORK' and ROWID =
'AAADnoAACAAAAAIAAD';

2000.07.28 18:26:21 SCOTT                          DEPT                                    2 USERS
INSERT                         SCOTT
LoginUserName = SCOTT, ClientInfo = , OsUserName = JALA, MachineName = IRM\JAREK1
insert into SCOTT.DEPT(DEPTNO,DNAME,LOC) values (20,'RESEARCH','DALLAS');
delete from SCOTT.DEPT where DEPTNO = 20 and DNAME = 'RESEARCH' and LOC = 'DALLAS' and ROWID = 'AAADnoAACAAAAAIAAA';

2000.07.28 18:26:27 SCOTT                          DEPT                                    2 USERS
INSERT                         SCOTT
LoginUserName = SCOTT, ClientInfo = , OsUserName = JALA, MachineName = IRM\JAREK1
insert into SCOTT.DEPT(DEPTNO,DNAME,LOC) values (30,'SALES','CHICAGO');
delete from SCOTT.DEPT where DEPTNO = 30 and DNAME = 'SALES' and LOC = 'CHICAGO' and ROWID = 'AAADnoAACAAAAAIAAB';

2000.07.28 18:26:32 SCOTT                          DEPT                                    2 USERS
INSERT                         SCOTT
LoginUserName = SCOTT, ClientInfo = , OsUserName = JALA, MachineName = IRM\JAREK1
insert into SCOTT.DEPT(DEPTNO,DNAME,LOC) values (40,'OPERATIONS','BOSTON');
delete from SCOTT.DEPT where DEPTNO = 40 and DNAME = 'OPERATIONS' and LOC = 'BOSTON' and ROWID = 'AAADnoAACAAAAAIAAC';

2000.07.28 19:23:25 SCOTT                          DEPT                                    2 USERS
UPDATE                         GAMMA
LoginUserName = GAMMA, ClientInfo = , OsUserName = JALA, MachineName = IRM\JAREK1
update SCOTT.DEPT set LOC = 'WARSZAWA' where ROWID = 'AAADnoAACAAAAAIAAD';
update SCOTT.DEPT set LOC = 'NEW YORK' where ROWID = 'AAADnoAACAAAAAIAAD';

2000.07.28 19:24:36 SCOTT                          DEPT                                    2 USERS
INSERT                         SYSTEM
LoginUserName = BETA, ClientInfo = , OsUserName = JALA, MachineName = IRM\JAREK1
insert into SCOTT.DEPT(DEPTNO,DNAME,LOC) values (50,'NEW','ZAKOPANE');
delete from SCOTT.DEPT where DEPTNO = 50 and DNAME = 'NEW' and LOC = 'ZAKOPANE' and ROWID = 'AAADnoAACAAAAAIAAE';

2000.07.28 19:25:03 SCOTT                          DEPT                                    2 USERS
DELETE                         SYSTEM
LoginUserName = SCOTT, ClientInfo = , OsUserName = JALA, MachineName = IRM\JAREK1
delete from SCOTT.DEPT where DEPTNO = 50 and DNAME = 'NEW' and LOC = 'ZAKOPANE' and ROWID = 'AAADnoAACAAAAAIAAE';
insert into SCOTT.DEPT(DEPTNO,DNAME,LOC) values (50,'NEW','ZAKOPANE');
```

Komentarz do przykładu: Wynikiem są wszystkie operacje na tablicy DEPT zapamiętane w dziennikach bieżących. Również cztery INSERT'y wykonane przed włączeniem śledzenia za pomocą wyzwalaczy. Pozostałe operacje można porównać z wynikiem w przykładzie 15. Czas realizacji polecenia wyniósł nieco ponad dwie minuty przy pięciu dziennikach o rozmiarze 10Mb każdy.

2.4. Podsumowanie

Okazuje się, że podobnie jak przy rejestracji połączeń, tradycyjne metody zbierania informacji o zmianach danych nadal są warte użycia. Nowinka w postaci Log Miner'a stanowi bardzo istotną pomoc, ale nie może być traktowana jako łatwe rozwiązanie problemu śledzenia zmian.

Bibliografia

1. Oracle8*i* Administrator's Guide, December 1999, Part No. A76956-01
2. Oracle8*i* Reference, December 1999, Part No. A76961-01
3. Oracle8*i* SQL Reference, December 1999, Part No. A76989-01
4. Oracle8*i* Application Developer's Guide - Fundamentals, December 1999, Part No. A76939-01
5. Oracle8*i* Supplied PL/SQL Packages Reference, December 1999, Part No. A76936-01
6. Oracle8*i* Concepts, December 1999, Part No. A76965-01