Oracle 8i R2 w praktyce

Wiesław Grabowski
Oracle Polska Sp. Z o.o.
e–mail: Wieslaw.Grabowski@oracle.com

Abstrakt. Oracle Server 8i R2 jest najnowszą wersją serwera bazy danych Oracle. Wersja ta zawiera bardzo wiele różnych nowych własności. Wiele z nich poznaliśmy jedynie z literatury. Wykład ten jest próbą podsumowania doświadczeń zebranych przez grupę CORE & CDM Działu Usług Konsultacyjnych Oracle Polska w czasie różnych przedsięwzięć z wykorzystaniem serwera Oracle 8i R2. Wśród poruszonych zagadnień będą omówione zagadnienia „prywatnej bazy danych „ , partycjonowania, administrowania bazą danych, wykorzystania indeksów opartych na funkcjach, itp.. Celem całego referatu jest przedstawienie rzeczywistych zastosowań przedstawionych wyżej nowych cech serwera bazy danych Oracle w wersji 8i.

1. Nowe cechy serwera bazy danych Oracle 8i wydanie 2

Wydanie 2 serwera bazy danych Oracle 8i wprowadza wiele nowych cech, z których wiele znamy tylko od strony prezentacji marketingowych lub ze stron dokumentacji. Poznanie wszystkich nowych cech jest bardzo trudne zwłaszcza biorąc pod uwagę tempo pojawiania się nowych, „lepszych" wersji oprogramownia. Wszyscy użytkownicy tej wersji serwera bazy danych mają jakieś doświadczenia z wykorzystania różnych cech tego oprogramowania. Wymiana uzyskanych w czasie takich prób doświadczeń może nam pozwolić na szybsze przyswojenie sobie kolejnych wersji różnych programów a w tym i serwera bazy danych Oracle. Spośród wykorzystywanych przez pracowników grupy CORE & CDMi cech za interesujące uznałem wybrane elementy wymienionych poniżej grup własności:

1. Autonomiczne transakcje
2. Java
3. Fine grain security i kontekst aplikacyjny
4. Wyzwalacze systemowe
5. Kontekst aplikacyjny
6. Partycjonowanie

2. Autonomiczne transakcje

Niekiedy mamy potrzebę aby zapisać lub zrezygnować z zapisu części zmian w sposób niezależny od tego w jaki sposób zakończy się główna transakcja. Typowym przykładem może być zrealizowanie własnej rejestracji informacji o błędach zapewniającej ich zapis do bazy nawet wtedy gdy oryginalna transakcja zostanie wycofana.

2.1. Podstawowe informacje o transakcja autonomicznych

Autonomiczna transakcja jest to niezależna transakcja rozpoczęta przez inną tzw. Główną transakcję. Umożliwia ona „zawieszenie" transakcji głównej, zatwierdzenie lub wycofanie wchodzących w skład transakcji autonomicznej zmian a następnie wznowienie przetwarzania transakcji głównej.

Transakcja taka ma swój własny zakres obowiązywania. Jest nim zakres podprogramu oznaczonego pragmą AUTONOMOUS_TRANSACTION.

Za podprogram w tym kontekście uważa się:

1. Najwyższy (nie zagnieżdżony), anonimowy blok PL/SQL
2. Lokalne, samodzielne i pakietowe funkcje i procedury

3. Metody typów obiektowych
4. Wyzwalacze bazodanowe

Autonomiczne transakcje charakteryzują się tym że:

- Zmiany autonomicznych transakcji nie zależą od stanu ani ostatecznego zakończenia transakcji głównej. Na przykład:
  - Transakcja autonomiczna nie widzi żadnych zmian wprowadzonych przez transakcję główną.
  - Zatwierdzenie lub wycofanie zmian w transakcji autonomicznej nie wpływa w żaden sposób na wynik transakcji głównej.
- Wyniki zmian wprowadzonych przez transakcję autonomiczną są widoczne dla innych transakcji natychmiast po ich zatwierdzeniu. Oznacza to, że inne transakcje mogą skorzystać ze zmodyfikowanych w ten sposób informacji nie czekając na zakończenie transakcji głównej.
- Transakcja autonomiczna może rozpoczynać inne transakcje autonomiczne.
- Transakcja autonomiczna to całkowicie niezależna transakcja a nie transakcja zagnieżdżona.

2.2. Wykorzystanie

W naszej praktyce zaistniała potrzeba zaimplementowania możliwości śledzenia wykonywanych przez użytkowników poleceń select z dokładnością do przeglądanych wierszy. W tym celu wykorzystaliśmy połączenie fine grained security z autonomicznymi transakcjami. W ramach mechanizmów fine grained security do każdego wykonywanego polecenia select doklejany jest niewidoczny predykat zawierający wywołanie funkcji zapisującej informacje o przeglądanych wierszach do odpowiedniej tabeli śladu. Wywoływana funkcja działa jako autonomiczna transakcja.

3. Java

Poczynając od wersji 8i mamy możliwość korzystania z tworzenia procedur i funkcji składowanych w Javie. Pełne przedstawienie wszelkich nowych możliwości wynikających z tego faktu wykracza poza ramy tego referatu. Tutaj przedstawimy dość nietypowe zastosowanie związane z próbą implementacji tzw. fine grained security zależnego od treści wykonywanego polecenia SQL.

3.1. Podstawowe informacje o funkcjach składowanych w Javie

Procedury składowane w Javie to publikowane do SQL metody klas Javy. Publikowanie klasy polega na stworzeniu specyfikacji wywołania która dokonuje mapowania nazw metod Javy, typów parametrów oraz typów zwracanych wartości do ich odpowiedników w SQL.

Utworzenie procedury składowanej w Javie wymaga wykonania następujących kroków:

1. Tworzymy kod klasy w wybranym narzędziu (np.Jdeveloper)

```
public class Oscar {
  // return a quotation from Oscar Wilde
  public static String quote() {
    return "I can resist everything except temptation.";
  }
}
```

2. Załadować i rozwiązać nazwy klas Javy

```
loadjava -user scott/tiger Oscar.class
```

3. Opublikować klasę Javy

```
SQL> connect scott/tiger

SQL> CREATE FUNCTION oscar_quote RETURN VARCHAR2
  2  AS LANGUAGE JAVA
  3   NAME 'Oscar.quote() return java.lang.String';

4. Możemy już wywoływać stworzoną funkcję

 SQL> VARIABLE theQuote VARCHAR2(50);

SQL> CALL oscar_quote() INTO :theQuote;

SQL> PRINT theQuote;

THEQUOTE
--------------------------------------------------
I can resist everything except temptation.
```

Tworzona procedura może potrzebować dostępu do zawartych w bazie danych informacji. Jest to możliwe z wykorzystaniem JDBC. W celu dostępu do bazy danych musimy utworzyć obiekt należący do klasy Connection przy czym mamy dwie możliwe sytuacje :

1. wykonujemy tzw. call out czyli podłączamy się do tej samej bazy danych.

W tym przypadku możemy wykorzystać tzw. internal driver czyli

```
Connection conn = new oracle.jdbc.driver.OracleDriver().defaultConnection();
```

2. podłączamy się do innej bazy danych lub chcemy otworzyć inną sesję

```
Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@host:1521:SID","system","manager")
;
```

Następnie musimy utworzyć obiekty klasy np. PreparedStatement, gdy korzystamy ze zmiennych wiązania.

```
PreparedStatement stmt = conn.prepareStatement(sqlStmt);
```

lub CallableStatement ,gdy chcemy wywołać blok PL/SQL lub procedurę składowaną.

```
CallableStatement stmt = conn.prepareCall(" begin zapisz_slad(?); end ;");
```

W kolejnym kroku musimy nadać wartości funkcjom wiązania

```
stmt.setString(1,sqlStmt);
```

potem wykonać polecenie tworząc ResultSet

```
ResultSet result = stmt.executeQuery();
```

I w końcu przetwarzać wiersze wchodzące w ResultSet

```
while (rset.next()) {
           sqlStmt = sqlStmt.append(rset.getString(1));
        }
```

3.2. Wykorzystanie

Próbując opracować procedury fine grained security reagujące różnie w zależności od treści wykonywanego polecenie pojawił się problem jak w czasie wykonywania polecenia znaleźć jego treść. Okazało się, że w PL/SQLu w funkcji dołączonej przez mechanizm fine grained security do predykatów klauzuli where nie daje mamy takiej możliwości. Bieżące zapytanie dla danej sessji może być znalezione korzystając z kolumn sql_address, sql_hash_value perspektywy V$SESSION i wybierając odpowiednie wiersze z perspektywy v$sqltext . Ponieważ wszystkie kolejne zapytania odbywają się w ramach tej samej sesji dlatego każde kolejne zapytanie zmienia wartości kolumn sql_address, sql_hash_value pojawiające się w perspektywie V$SESSION i nie jesteśmy w stanie znaleźć interesującego nas zapytania. Gdybyśmy wykonali odpowiednie zapytania z odrębnej sesji znalezienie potrzebnego nam zdania jest bardzo proste. Aby tego dokonać musimy skorzystać z procedury zewnętrznej (external procedure). Do wyboru mamy programowanie w C lub Javie. W tym przypadku wybrano Javę. Aby znaleźć aktualnie wykonywane polecenie musimy otworzyć połączenie zwrotne do bazy danych nie korzystające z domyślnego, wewnętrznego sterownika (internal driver).

Poniżej znajduje się kod programu napisanego w celu sprawdzenia możliwości realizowalności (proof of concept)takiego zadania. Poniższy kod nie był jeszcze optymalizowany pod kątem wydajności i należy go traktować jedynie jako przykładowy.

```
// Copyright (c) 2000 Oracle Polska SP. z o.o.
package sql;
import java.lang.*;
import java.sql.*;
import java.util.*;
import oracle.jdbc.driver.*;
/*import oracle.sql.*;
import sqlj.runtime.ref.DefaultContext;*/

//import oracle.sqlj.runtime.Oracle;


/**
 * A Class class.
 * <P>
 * @author Wiesław Grabowski
 */
   class Zdanie extends Object {
      long  sqlStmtId ;
      boolean sledzic ;

      public Zdanie(long id, boolean sledz){
         sqlStmtId = id;
         sledzic = sledz;
      }
   }

public class Daj_zdanie extends Object {
 static Vector zdania = new Vector();
```

```
    /**
     * Constructor
     */
    static int  juzBylo (long sqlStmtId) {

        for ( int i = 0; i< zdania.size();i++) {
            if (((Zdanie) zdania.elementAt(i)).sqlStmtId==sqlStmtId) return i;
        }
        return -1;
    }
    static public String daj_zdanie(int numerSesji, long sqlStmtId )
    throws SQLException {
    System.out.println("dajzdanie");
    Zdanie zdanie;
    int nr;
    boolean sledzic = false;
    StringBuffer sqlStmt = new StringBuffer();
    if (  (nr=juzBylo(sqlStmtId))<0 ) {
   // otwieramy osobną sesję
       Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@host:1521:SID","system","manager")
;

       try {
  // polecenie znajdujące aktualne polecenie dla sesji o zadanym numrze AUDSID
          PreparedStatement pstmt
           = conn.prepareStatement ("SELECT sql_text FROM v$sqltext where
(address,hash_value) = (select sql_address,sql_hash_value from v$session where
audsid = (?)) order by piece");
          pstmt.setInt(1, numerSesji);
          ResultSet rset = pstmt.executeQuery ();
          while (rset.next()) {
             sqlStmt = sqlStmt.append(rset.getString(1));
          }
          rset.close();
       } catch (Exception e) {
          return e.getMessage();
       }
  // zdanie wystepuje po raz pierwszy
       zdanie = new
Zdanie(sqlStmtId,czySledzic(sqlStmt.toString().toLowerCase(),conn));
       conn.close();
    } else {
  // zdanie już było sprawdzane
       zdanie = ((Zdanie)zdania.elementAt(nr));
    }
    if(zdanie.sledzic)  sledz(sqlStmt.toString());
    return  (zdanie.sledzic?"true":"false");

  }
     static boolean czySledzic(String stmt, Connection conn)
      throws SQLException {
  // do zanalizowania polecenia korzystamy z metody parseStatement
  // w wyniku uzyskujemy ResultSetMetaData opisujący list
  ResultSetMetaData metaData = parseStatement(stmt,(OracleConnection)conn);
       for(int i = 1 ; i<= metaData.getColumnCount(); i++) {
```

```
            if (
kolumnaSledzona(metaData.getColumnName(i),metaData.getTableName(i), conn)){
                return true;
            }
         }
         return false;
      }

      static boolean kolumnaSledzona( String kolumna,String tabela, Connection
conn)
      throws SQLException  {
   // sprawdzenie czy kolumna znajduje się zestawie kolumn które nalezy obserwowac
         PreparedStatement stmt = conn.prepareStatement("select count(*) from bla
where k1 = ?");
         stmt.setString(1,kolumna);
         ResultSet result = stmt.executeQuery();
         result.next();
         if (result.getInt(1)>0 ) return true;
         return false;
      }

      static ResultSetMetaData parseStatement(String sqlStmt, OracleConnection
conn)
      throws SQLException {

         PreparedStatement stmt = conn.prepareStatement(sqlStmt);
         System.out.println("parseStatement 2"+sqlStmt+" ");
         ResultSetMetaData metaData = stmt.executeQuery().getMetaData() ;
   //     ResultSetMetaData metaData = null ;
         return metaData;
      }

      static void sledz (String sqlStmt) throws SQLException{
          Connection conn = new
oracle.jdbc.driver.OracleDriver().defaultConnection();
          CallableStatement stmt = conn.prepareCall(" begin zapisz_slad(?); end
;");
          stmt.setString(1,sqlStmt);
          stmt.executeUpdate();
      }
   }
```

Po załadowaniu klasy do serwera przy pomocy loadjava

Należy jeszcze opublikować żądane metody:

```
CREATE FUNCTION my_sql (p_ses_audid number, p_stmt_id number) RETURN VARCHAR2
  2  AS LANGUAGE JAVA
  4   NAME 'Daj_zdanie.daj_zdanie(int , long ) return java.lang.String';
```

Opisany wyżej mechanizm w parze z autonomiczymi transakcjami mógłby być wykorzystany np. do utworzenia tabeli zapamiętującej wybrane przez polecenie wiersze. Mogłoby to być użyteczne jeżeli po przeanalizowaniu wyniku polecenia select chcielibyśmy w kolejnych

zapytaniach ograniczyć się do wybranego już zbioru wierszy. Sytuacja taka występuje czasami w narzędziu Discoverer.

4. Fine grained security , kontekst aplikacyjny oraz wyzwalacze systemowe

Fine grained security umożliwia wprowadzenie nowego poziomu bezpieczeństwa poprzez dynamiczne ograniczanie widocznych dla użytkowników wierszy.

4.1. Podstawowe informacje

Implementacja fine grained security wykorzystuje nastepujące elementy: pakiet DBMS_RLS (dostepny tylko w wersji Enterprise Edition), definiowanie kontekstu aplikacyjnego oraz wyzwalacza systemowego ON LOGON.

Definiowanie procedur bezpieczeństwa na poziomie tabel lub perspektyw, zamiast wbudowywania ich w kod aplikacji, zapewnia wyższy poziom bezpieczenstwa, prostotę implementacji i elastyczność..

Bezpieczeństwo

Implementacja procedur bezpieczeństwa na poziomie tabeli bądź perspektywy powoduje, że nieistotny staje się sposób podłączenia do bazy, poprzez aplikacje czy też np. SQLPLUS. Wymuszane reguły bezpieczeństwa obowiązują zawsze

Prostota

Implementacja taka upraszcza także zarządzanie ponieważ modyfikujemy tylko pojedynczy obiekt a skutki będą widoczne dla wszystkich metod dostępu i aplikacji.

Elastyczność

Dla jednego obiektu, jak tabela czy perspektywa można mieć wiele procedur bezpieczeństwa np. jedna dla poleceń SELECT, inną dla INSERT, a jeszcze inna dla UPDATE i DELETE. Dla przykładu możemy chcieć umożliwić pracownikowi kadr wykonywanie poleceń SELECT dla wszystkich pracowników w obsługiwanej przez niego komórce organizacyjnej lecz ograniczyć możliwość modyfikowania przez niego płac tylko do pracowników, których nazwiska zaczynają się na litery od „A” do „F”. Możemy też definiować wiele procedur bezpieczeństwa dla danej tabeli lub perspektywy i wtedy będą one dołączane z udziałem operatora AND.

Wdrożenie własnej polityki bezpieczeństwa wymaga wykonania następujących kroków:

1. tworzymy funkcję zwracającą wymagany przez nas dodatkowy predykat dołączany do każdego polecenia
2. rejestrujemy utworzona funkcję przy pomocy pakietu

```
DBMS_RLS.ADD_POLICY (
   object_schema   IN VARCHAR2 := NULL,
   object_name     IN VARCHAR2,
   policy_name     IN VARCHAR2,
   function_schema IN VARCHAR2 := NULL,
   policy_function IN VARCHAR2,
   statement_types IN VARCHAR2 := NULL,
   update_check    IN BOOLEAN  := FALSE,
   enable          IN BOOLEAN  := TRUE);
```

Podane wyżej dwa niezbędne kroki to przypadek najprostszy. Najczęściej przydatne jest wykorzystanie dodatkowych informacji zapisywanych w kontekście aplikacji. Umożliwia to bardziej elastyczne reagowanie przez procedurę bezpieczeństwa na wykonywane polecenia.Do wykorzystania mamy 2 poziomy kontekstu :systemowy i definiowany przez użytkownika .

Pierwszy jest to grupa informacji dostępna w grupie USERENV. Informacje takie możemy tylko czytać nie można natomiast zmieniać ich wartości. W bardziej zaawansowanych przypadkach interesująca będzie możliwość tworzenia własnych grup. W tym celu należy :

1. Utworzyć procedurę służącą do ustawiania kontekstu np.

```
CREATE OR REPLACE PACKAGE empsec AS
   PROCEDURE setempctx;
END empsec;
/
CREATE OR REPLACE PACKAGE BODY empsec AS
PROCEDURE setempctx AS
   the_role VARCHAR2(10);
   me       VARCHAR2(10);
   myename  VARCHAR2(30);
   myempno  NUMBER;
   myboss   NUMBER;
   subs     NUMBER;
BEGIN
   dbms_session.set_context('empctx','role','EMP');
-- By default I am an EMP
   me:= SYS_CONTEXT('userenv','session_user');
--
-- get my information from emp
--
   SELECT empno, mgr , ename
   INTO   myempno, myboss, myename
   FROM   emp
   WHERE  ename=me;
--
-- If I have any subordinates I am an MGR
--
   SELECT COUNT(*)
   INTO   subs
   FROM   emp
   WHERE  mgr=myempno;
--
-- @ do i really still need this ?
--
    IF NVL(subs,0) > 0 THEN
      dbms_session.set_context('empctx','role','MGR');
    END IF;
--
-- If I do NOT have a manager then I am CEO
--
   IF myboss IS NULL THEN
      dbms_session.set_context('empctx','role','CEO');
   END IF;
EXCEPTION
WHEN OTHERS THEN NULL;
END setempctx;
END empsec;
/
```

2. Utworzyć własny kontekst i powiązać go z procedurą ustawiającą kontekst

```
CREATE CONTEXT emp_sec USING empsec;
```

3. Ustawić potrzebne zmienne wchodzące w skład kontekstu w wyzwalaczu ON LOGON

```
create or replace trigger ajh_security_trg
after logon on database
begin
if (user in ('SYS','SYSTEM','ORDSYS','DBSNMP','TRACESVR','CTXSYS',
'ORDPLUGINS','MDSYS','OUTLN','HST','RAD','OAS_PUBLIC','WEBDB') )
then
  return;
else
  scott.empsec.setempctx;
end if;
end;
/
```

4. Wykorzystać ustawione zmienne w pakiecie bezpieczeństwa. W poniższym pakiecie wykorzystano tez autonomiczne transakcje do zapisania w tabeli slad pewnych informacji o stanie zmiennych pakietowych i kontekstowych.

```
--
CREATE OR REPLACE PACKAGE hr_access AS
   FUNCTION secure_access( obj_schema VARCHAR2, obj_name VARCHAR2 )
      RETURN VARCHAR2;
END hr_access;
/
CREATE OR REPLACE PACKAGE BODY hr_access IS
FUNCTION secure_access( obj_schema VARCHAR2, obj_name VARCHAR2 )
      RETURN VARCHAR2 IS
pragma autonomous_transaction;
   l_predicate VARCHAR2(2000);
   myename     VARCHAR2(30);
   myempno     NUMBER;
   v_time      number :=0;
   v_hash      number;
   v_addr      raw(4);
   v_ses_audid number;
   CURSOR c1 IS SELECT empno FROM val_emp WHERE ename = myename;
--
BEGIN
--
-- get the ename from the context
--
   myename := SYS_CONTEXT('userenv','session_user');
   v_ses_audid := SYS_CONTEXT('userenv','sessionid')
;
   insert into slad values (v_ses_audid);
   commit;
   begin
      select sql_hash_value, sql_address
      into   v_hash, v_addr
      from v$session
      where audsid=v_ses_audid;
      insert into slad values (myename);
      insert into slad values(v_hash);
      insert into slad values(rawtohex(v_addr));
   exception
     when others then
       null;
```

```
   end ;

--
   OPEN C1;
     FETCH C1 INTO myempno;
   CLOSE C1;
   insert into slad values (myempno);
   IF SYS_CONTEXT('empctx','role') = 'CEO' THEN
      RETURN '';
      -- I AM GOD! - no extra qualification to the query
   ELSIF  SYS_CONTEXT('empctx','role') = 'MGR' THEN
      l_predicate :=
          'empno='||myempno||
          ' OR EMPNO IN ( SELECT empno FROM val_emp  WHERE mgr= '||
          myempno|| ')';
          -- Make this a tree walk later
   ELSIF  SYS_CONTEXT('empctx','role') = 'EMP' THEN
      -- I can only see my own data
      l_predicate := 'empno = '||myempno;
   END IF;
// liczba 1/100 s od uruchomienia instancji jest wykorzystywana jako unikalny
identyfikator zapytań uruchamianych w czasie sesji.
   select hsecs
   into v_time
   from v$timer;
if length(trim(l_predicate))>0 then
      l_predicate := l_predicate ||' and';
   end if;
//  my_sql to funkcja składowana w Javie
   l_predicate := l_predicate ||' my_sql('||
v_ses_audid||’,''||to_char(v_time)||''') is not null';
   insert into slad values (l_predicate);
   commit;
   RETURN l_predicate;
END secure_access;
END hr_access;
/
REM create the policy
BEGIN
   DBMS_RLS.DROP_POLICY( 'SCOTT','EMP','EMP_POLICY');
END;
/
BEGIN
   DBMS_RLS.ADD_POLICY(
                        'SCOTT',
                        'EMP',
                        'EMP_POLICY',
                        'SCOTT',
                        'HR_ACCESS.SECURE_ACCESS',
                        'SELECT', true, true
                       );
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

Dodatkowy krok w celu utworzenia użytkowników testowych
```

```
REM RUN ONLY ONCE !!!!!
REM THIS ISN'T REALLY PART OF THE DEMO
REM However lets ensure the is a database user for
REM all the people in EMP
REM ************************************************
REM BEWARE BEWARE BEWARE
REM This will drop any User with the same name as an emp
REM ename dynamically. If you REALLY have someone called
REM e.g. ADAMS THEY WILL BE dropped and recreated
REM The other consideration is that your EMP shouldn't
REM Have 2 million rows or this will take a long time
REM
SET serveroutput on
DECLARE
   l_statement varchar2(2000);
   l_statement2 varchar2(2000);
   l_statement3 varchar2(2000);
   l_dropuser varchar2(20);
   l_creuser varchar2(20);
   -- Everyone who is in emp ( except me ) who has an existing
   -- Oracle login
   CURSOR c1 IS SELECT a.ename
                FROM scott.emp a, dba_users b
                WHERE a.ename = b.username
                AND a.ename !=user;
   -- All EMP's
   CURSOR c2 IS SELECT ename
                FROM scott.emp
                WHERE  ename != user;
BEGIN
   FOR c1rec in c1 LOOP
      dbms_output.put_line('Dropping user '||c1rec.ename);
      l_statement:='DROP USER '||c1rec.ename||' cascade';
      EXECUTE IMMEDIATE l_statement;
   END LOOP;
   FOR c2rec in c2 LOOP
      dbms_output.put_line('Creating user '||c2rec.ename);
      l_statement2 := 'CREATE USER '||c2rec.ename||
                      ' IDENTIFIED BY '||c2rec.ename;
      l_statement3 := 'GRANT CONNECT TO '||c2rec.ename;
      EXECUTE IMMEDIATE l_statement2;
      EXECUTE IMMEDIATE l_statement3;
   END LOOP;
END;
```

4.2. Wykorzystanie

Opisany wyżej mechanizm bezpieczeństwa wykorzystujący fine grained security , kontekst aplikacyjny oraz wyzwalacz on logon został z powodzeniem zaimplementowany u jednego z klientów. Zaimplementowane rozwiązanie różni się od przedstawionego powyżej tym, że posłużyło ono do zrealizowania śledzenia wierszy oglądanych przez użytkowników. Oznacza to, że w czasie wykonywania polecenia SELECT w tle odbywa się autonomiczna transakcja rejestrująca w tabelach śladu obejrzane wiersze. Pomimo dodatkowego przetwarzania wydajność jest zadawalająca. Jedyny problem to zapytania wykonujące pełne sekwencyjne przeglądanie dużych tabel.

5. Partycjonowanie

Wiele już słów padło o zaletach partycjonowania. Doskonale tez są znane podstawowe własności partycjonowania w implementacji Oracle. W tej części chciałbym wspomnieć o : metodzie zmiany właściciela tabeli z wykorzystaniem partycjonowania oraz o wykorzystaniu partycjonowania w procesach ładowania danych.

5.1. Zmiana właściciela tabeli

Wszyscy z pewnością znamy standardowa metodę zmiany właściciela tabel tj. EXPORT oraz IMPORT z opcjami FROMUSER /TOUSER i wiemy że jej bolesność jest wprost proporcjonalna do rozmiarów tabeli.

Do wykonania tego samego zadania możemy wykorzystać partycjonowanie tabel. Mogą tu wystąpić dwa przypadki: tabela jest partycjonowana lub nie.

W pierwszym przypadku tworzymy tabele tymczasową poleceniem:

```
CREATE TABLE xxx
AS
 SELECT *
 FROM <NASZA TABELA>
WHERE 1 = 0;
```

Warunek 1=0 klauzuli WHERE zapewnia, że do tworzonej tabeli nie zostaną zapisane żadne wiersze.

Tworzymy też tabelę docelową o identycznej strukturze partycji jak źródłowa w schemacie nowego właściciela. Nieistotne są przy tym parametry przechowywania tabeli (STORAGE) ponieważ zapełniony segment danych już istnieje i tylko zmieni właściciela. Dane nie będą fizycznie przepisywane.

Do utworzenia takiej tabeli możemy wykorzystać np natępujący ciąg poleceń:

```
CREATE TABLE <NOWY WŁAŚCICIEL>.<NASZA TABELA>
PARTITION BY RANGE ( <KOLUMNA PARTYCJONOWANIA>) {
PARTITION <NAZWA 1  PARTYCJI> VALUES LESS THAN (<ZAKRES 1 PARTYCJI>)
)
AS
 SELECT *
 FROM <NASZA TABELA>
WHERE 1 = 0;
```

Nastepnie piszemy skrypt SQLPLUS do odtworzenia pozostałych partycji.

```
SQL>set pages 0
SQL>set feed off
SQL>spool tworzpart.sql
SQL>SELECT 'ALTER TABLE <NOWY WŁAŚCICIEL>.<NASZA TABELA> ADD PARTITION '||
2>PARTITION_NAME ||' VALUES LESS THAN (',HIGH_VALUE,');'
3>FROM DBA_TAB_PARTITIONS
4>WHERE TABLE_NAME = '<NASZA TABELA>'
5>AND OWNER ='<STARY WŁAŚCICIEL>'
6>AND PARTITION_POSITION >1
7>ORDER BY PARTITION_POSITION
```

```
/
@tworzpart
```

Następnie dla każdej partycji „przenoszonej" tabeli wykonujemy:

```
ALTER TABLE <NASZ TABELA>
EXCHANGE PARTITION <NAZWA PARTYCJI>
WITH TABLE xxx;
```

a następnie :

```
ALTER TABLE  <NOWY WŁAŚCICIEL>.<NASZ TABELA>
EXCHANGE PARTITION WITH TABLE xxx;
```

W drugim przypadku jako tabelę tymczasową tworzymy tabelę partycjonowaną z jedną partycją.

```
CREATE TABLE xxx
PARTITION BY <DOWOLNA KOLUMNA>
( PARTITION P1 VALUES LESS THAN ( MAXVALUE))
AS
 SELECT *
 FROM <NASZA TABELA>
WHERE 1 = 0;
```

W docelowym schemacie powielamy strukturę tabeli.

A w następnej kolejności

```
ALTER TABLE xxx
EXCHANGE PARTITION <NAZWA PARTYCJI>
WITH TABLE <NASZ TABELA>;
```

i

```
ALTER TABLE xxx
EXCHANGE PARTITION <NAZWA PARTYCJI>
WITH TABLE <NOWY WŁAŚCICIEL>.<NASZ TABELA>;
```

5.2. Ładowanie danych

Dla potrzeb jednego z klientów opracowano środowisko do tworzenia agregatów. Zarówno tabela źródłowa z danymi szczegółowymi jak i tabela agregatów są partycjonowane. Ponieważ obecnie w tabeli źródłowej istnieje ponad 400 partycji przyjęto, że agregaty będą tworzone partycja po partycji, kolejność agregowania partycji może być różna. Dodatkowo przyjęto, w celu minimalizacji wpływu na już istniejące zagregowane partycje, że proces tworzenia agregatów będzie wykorzystywał osobne tabele, które po zbudowaniu zostaną wpięte do odpowiednich partycji tabeli agregatów przy pomocy polecenia ALTER TABLE EXCHANGE PARTITION. Kolejnym założeniem była możliwość uruchomienia wielu równoległych procesów agregujących dane z których każdy może mieć swój własny stopień zrównoleglenia. Wdrożone ostatecznie rozwiązanie wykorzystuje mechanizm harmonogramowania zadań w bazie danych DBMS_JOB. Uruchamiając proces podajemy nazwę tabeli dla której dokonywana jest agregacja, nazwę agregatu który tworzymy, partycję od której chcemy rozpocząć agregację , liczbę partycji którą będziemy agregować oraz liczbę równoległych procesów które będą jednocześnie agregowały dane. Ponieważ żądana liczba procesów może być mniejsza od ilości partycji, w momencie startu tworzona jest liczba procesów równa ilości partycji z tym, że każdy z nich przydzielany jest do jednej z kolejek. Liczba kolejek jest równa ilości żądanych równoległych procesów. W danej chwili czasu aktywny jest tylko jeden proces z danej kolejki.

6. Indeksy oparte na funkcjach

Innym być może drobnym, lecz bardzo użytecznym usprawnieniem servera bazy danych Oracle8i jest możliwość tworzenia indeksów opartych funkcjach. W jednym z prowadzonych projektów mieliśmy następująca sytuację:

Tabela T jest bardzo duża.

Tabela T posiada pole opcjonalne P.

Indeks na polu P jest rozsądnie selektywny.

W polu P jest niewiele wartości nullowych.

Następuje częste wyszukiwania po polu P, przy czym zdarza się, że interesują nas również te wiersze które nie mają tam wartości. Aby to zaimplementować piszemy:

```
select (...) from T
where nvl(P,<wartośćniemożliwa>) = nvl(:zmienna,<wartośćniemożliwa>)
```

Taka konstrukcja w poprzednich wersjach implikuje wykonanie pełnego przeszukania tabeli. Obecnie mamy jednak możliwość utworzenia indeksu bazującego na funkcji.

Po utworzeniu indeksu podane wyżej zapytanie może zostać wykonane z wykorzystaniem indeksu.

Implementacja indeksu opartego na funkcji wymaga ustawienia parametrów QUERY_REWRITE_ENABLE = true oraz QUERY_REWITE_INTEGRITY = enforced w pliku init.ora. Dodatkowo sama funkcja na której opieramy indeks musi być deterministyczna czyli zawsze dla tych samych parametrów wejściowych zwracać tę samą wartość.

Bibliografia

1. Oracle8i Application Developer's Guide
2. Oracle8i PL/SQL Guide
3. Oracle8i Java Developer's Guide
4. Oracle8i Java Stored Procedure Developer's Guide
5. Oracle8i JDBC Developer's Guide and Reference
6. Oracle8i Supplied PL/SQL Packages Reference