

Porównanie wydajności bibliotek XML-owych w Oracle 9i

Bartłomiej Jabłoński

Uniwersytet Łódzki

Oracle 9i udostępnia wiele narzędzi do obsługi danych XML. Dostęp ten możliwy jest z poziomu różnych interfejsów: SQL, PL/SQL, Javy uruchamianej w wirtualnej maszynie Oracle lub poza środowiskiem serwera. Rosnąca popularność standardu XML powoduje, że coraz więcej firm przesyła dane w tym formacie. Nierzadko są to dane o dużej objętości. Proces ładowania tych dokumentów lub tworzenie obszernych raportów w formacie XML to najczęściej spotykane sytuacje, gdzie wydajność ma kluczowe znaczenie. W takich przypadkach istotne wydaje się wybranie takiego rozwiązania, które spełnia określone wymagania wydajnościowe.

Artykuł przedstawia różne metody dostępu do danych XML przechowywanych na serwerze Oracle 9i. W artykule znajduje się porównanie wydajności procesów ładowania i raportowania dużych ilości danych w formacie XML. Wyniki eksperymentu mogą następnie posłużyć projektantom do właściwego wyboru określonej techniki w zastosowaniu standardu XML w swojej firmie.

Informacja o autorze:

Bartłomiej Jabłoński od 1999 roku pracuje na stanowisku asystenta na Wydziale Matematyki Uniwersytetu Łódzkiego, gdzie zajmuje się głównie bazami danych Oracle oraz standardem XML ze szczególnym uwzględnieniem jego zastosowań w bazach danych. Od 1994 roku wielokrotnie uczestniczył w projektach informatycznych związanych z bazami danych Oracle jako konsultant. Od 1995 prowadzi szkolenia w Centrum Edukacyjnym Oracle Polska. Jest autorem trzech i tłumaczem czterech książek z zakresu informatyki.

1. Wstęp

Początek wieku XXI jawi się nam jako czas, w którym informacja jest głównym „towarem”. Każda firma gromadzi mnóstwo danych, które pozwalają na szybkie podejmowanie decyzji. W epoce szybkich łączy i łatwej dostępności Internetu bardzo popularnym stała się wzajemna wymiana danych. Coraz więcej przedsiębiorstw posiada dane zarówno z jej działalności operacyjnej, ale również te, które otrzymuje od podmiotów, z którymi kooperuje. Ilość i częstość przesyłanych danych wymusiła na producentach standaryzację tego procesu.

Standard XML dzięki swojej otwartości i rozszerzalności powoduje, że może być on stosowany jako format wymiany danych w najróżniejszych dziedzinach gospodarki, niezależnie od stosowanych rozwiązań sprzętowych i programowych. Szczególną jego zaletą jest to, że w oparciu o ten standard powstało wiele narzędzi. Ich stosowanie nie wymaga od programisty szczególnej wiedzy – większość zadań związanych z wczytywaniem pliku XML do pamięci, jego analizy i/lub przekształcania wykonuje się wywołując określone standardem procedury.

Serwer Oracle 9i posiada możliwości przechowywania dokumentów XML lub ich fragmentów, wyszukiwania, konwersji danych XML na dane relacyjne i odwrotnie. Obok serwera istnieje również możliwość dysponowania pakietem narzędziowym XDK, który dostarcza bibliotek do obsługi danych XML wewnątrz własnych programów.

Celem tego artykułu jest przedstawienie różnych metod, które pozwalają na eksport relacyjnych danych z bazy danych Oracle do pliku XML oraz na proces odwrotny – import danych XML do tabel bazy danych. W toku dokonywania implementacji poszczególnych metod okazało się, że każda z nich charakteryzuje się inną wydajnością, której analiza została zamieszczona w dalszej części.

2. Standardowe biblioteki do obsługi danych XML w Oracle 9i

W Oracle 9i można znaleźć trzy biblioteki pozwalające na konwersję danych relacyjnych na dane w formacie XML i/lub odwrotnie. Biblioteki te można wywoływać z poziomu procedur PL/SQL przechowywanych po stronie serwera, jak i własnoręcznie napisanych (np. w języku Java) klas łączących się z serwerem przy pomocy JDBC. W opisanym tu eksperymencie postarano się wykorzystać wszystkie możliwe kombinacje uzyskując w sumie 6 sposobów importowania danych XML oraz 7 sposobów eksportowania tych danych. Dla porównania w tabelach umieszczono także wyniki importu analogicznych danych z pliku tekstowego przy pomocy narzędzia SQL*Loader i generowanie dokumentu XML przy pomocy „czystego” SQL-a.

Pakiet XSU

Biblioteka *XML SQL Utility for Java* składa się z dwóch klas: *OracleXMLQuery* i *OracleXMLSave*. Pierwsza z nich tworzy dokument XML (jako plik tekstowy lub struktura DOM), na podstawie podanego jako parametr, zapytania `SELECT`, druga – pozwala na podstawie danych w formacie XML przeprowadzać operacje `INSERT`, `UPDATE` i `DELETE`.

Pakiet ten jest dostępny w postaci biblioteki języka Java oraz w postaci pakietów *DBMS_XMLQuery* i *DBMS_XMLSave*, które dostarczają PL/SQL-owych interfejsów do w/w klas. W Oracle 9iR2 doszedł jeszcze jeden pakiet *DBMS_XMLGen* zastępujący funkcjonalnością *DBMS_XMLQuery*. Jest on zintegrowany z jądrem serwera i może być wywoływany tylko wewnątrz bazy.

Importowane i generowane dane XML posiadają znaczniki, których nazwy odpowiadają nazwom kolumn. Nazwy znaczników rekordów oraz zbioru rekordów są konfigurowalne.

Pakiet TransX

Biblioteka *TransX Utility for Java* to narzędzie specjalnie przeznaczone do ładowania lub generowania danych w formacie XML. Dane te jednak muszą być poprzedzone definicją struktury danych (opis podobny jest do XML Schema), zaś poszczególne pola muszą być opisane specjalnymi znacznikami. Podczas tych operacji może być dokonywana transformacja XSL, walidacja pól, sprawdzanie typów danych, itp. Można powiedzieć, że jest to funkcjonalny odpowiednik SQL*Loadera (z możliwością eksportu).

Pakiet SQLX

Pakiet *SQLX* składa się z kilku funkcji i operatorów, które mogą być wykorzystywane bezpośrednio w zapytaniach SQL. Funkcje te mogą być zarówno stosowane do danych XML przechowywanych w bazie, jak i do danych typów prostych (VARCHAR2, NUMBER, itp). Dzięki tym funkcjom można: wyszukiwać, modyfikować i tworzyć dokumenty XML (lub jego fragmenty).

W eksperymencie wykorzystano tylko jedną z w/w wspomnianych funkcji: *XMLForest*. Funkcja ta pobiera całą kolekcję wartości i tworzy z niej las elementów.

3. Zastosowane metody importu danych w formacie XML

SQL*Loader (sqlldr)

SQL*Loader jest najczęściej stosowanym narzędziem do wczytywania danych tekstowych. Pozwala na obsługę pliku, gdzie poszczególne pola oddzielone są specjalnymi znakami (np. przecinkami) lub tam, gdzie pola mają określoną długość. Niestety w wersji Oracle 9iR2, SQL*Loader nie posiada możliwość analizowania danych XML. Jedyną możliwością to wczytanie przez to narzędzie danych XML jako danych CLOB i późniejsza ich interpretacja przy pomocy np. PL/SQL-a.

W niniejszym eksperymencie wyniki działania tego rozwiązania zostały zaprezentowane jako punkt odniesienia do porównania, czy wczytywanie danych XML jest wolniejsze od wczytywania danych tekstowych.

Pakiet XSU obsługiwany w Java (*xsu_java*)

Biblioteka XSU jest dostępna z poziomu wywołań języka Java. Aby skorzystać z tej możliwości należy napisać własną klasę. Przy inicjalizacji klasy *OracleXMLSave* podaje się wskaźnik na połączenie JDBC z bazą oraz nazwę modyfikowanej tabeli. Kolejnymi wywołaniami metod można podać niestandardowe nazwy znaczników, format daty, itp. Metoda *insertXML* wczytuje plik – dokument XML – i jego zawartość umieszcza w postaci rekordów tabeli.

```
public class xsu_import {
    public static void main(String args[]) throws SQLException {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn =
            DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ora",
                                      "zakop", "zakop");
        OracleXMLSave sav = new OracleXMLSave(conn, "KSIAZKA");
        sav.setDateFormat("dd-mm-yyyy");
        sav.setRowTag("KSIAZKA");
        URL url = sav.createURL(args[0]);
        int rowCount = sav.insertXML(url);
        conn.close();
    }
}
```

Pakiet XSU obsługiwany z linii polecenia (*xsu_cmd*)

Wywołanie bibliotek XSU jest możliwe również bez konieczności pisania i kompilowania własnych klas. Klasa OracleXML jest wyposażona w metodę *main*, a zatem można ją wywołać bezpośrednio z linii polecenia:

```
java OracleXML putXML -user "zakop/zakop" -conn "jdbc:oracle:oci8:@ora" -  
rowTag "KSIAZKA" -dateFormat dd-mm-yyyy -fileName "fname.xml" KSIAZKI
```

Pakiet XSU wywoływany z PL/SQL (*xsu_plsql*)

Biblioteka XSU może być wywołana z wnętrza procedury PL/SQL. Dane przetwarzane przez tę bibliotekę muszą jednak znajdować się na serwerze, biblioteka jako taka nie ma możliwości odczytywania plików. W zamieszczonej poniżej procedurze następuje wczytanie dokumentu XML do zmiennej tymczasowej CLOB. Następnie jest ona analizowana i wpisywana do odpowiedniej tabeli w bazie.

```
procedure imp_xml(filename in varchar2) is  
  src_loc      bfile ;  
  dst_loc      clob  ;  
  amt          number := dbms_lob.lobmaxsize;  
  src_offset   number := 1 ;  
  dst_offset   number := 1 ;  
  lang_ctx    number := dbms_lob.default_lang_ctx;  
  warning     number;  
  c           dbms_xmlsave.ctxType;  
BEGIN  
  src_loc := bfilename('XML_DIR',filename) ;  
  dbms_lob.createtemporary(dst_loc, TRUE);  
  dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);  
  dbms_lob.LOADCLOBFROMFILE(dst_loc,src_loc, amt, dst_offset, src_offset,  
    dbms_lob.default_csid, lang_ctx,warning) ;  
  c := dbms_xmlsave.newContext('KSIAZKI');  
  dbms_xmlsave.setRowTag(c, 'KSIAZKA');  
  dbms_xmlsave.setDateFormat(c, 'dd-mm-yyyy');  
  amt := dbms_xmlsave.insertXML(c, dst_loc);  
  DBMS_XMLSave.closeContext(c);  
  dbms_lob.freetemporary(dst_loc);  
  dbms_lob.filecloseall() ;  
END ;
```

Parser SAX (*sax*)

W pakiecie XDK dostępna jest implementacja parsera SAX firmy Oracle. Obsługa parsera wymaga jednak napisania klasy z metodami wybierającymi kolejne pola i składającymi je w polecenie INSERT. Powoduje to pewną niedogodność dostosowywania programu do niemal najmniejszych zmian w strukturze danych zawartych w importowanym pliku.

Zastosowane w tym eksperymencie dane były płaskie – nie występowały zagnieżdżenia, więc i napisanie własnego parsera było względnie proste (ok. 100 linii kodu). Należy jednak pamiętać, że dla bardziej skomplikowanych struktur, listing programu będzie rósł nieproporcjonalnie, co znacznie utrudni dokonywanie ewentualnych późniejszych zmian.

Pakiet TransX obsługiwany z poziomu języka Java (*transx_java*)

Klasa *TransX* jest interfejsem biblioteki z poziomu języka Java. Jako parametry otwarcia sesji ładowania danych podaje się te same parametry jak w połączeniu JDBC, lecz sama inicjalizacja drivera JDBC zachodzi już wewnątrz tej klasy (w przeciwieństwie do biblioteki XSU).

```
public class imp_transx {
    public static void main(String args[]) throws SQLException {
        try {
            TransX transx = loader.getLoader();
            transx.open("jdbc:oracle:thin:@localhost:1521:ora", "zakop", "zakop");
            transx.setValidationMode( false );
            transx.load( args[0] );
            transx.close();
        } catch (Exception e) {
            e.printStackTrace(System.out);
        }
    }
}
```

Pakiet TransX obsługiwany z linii polecenia (*transx_cmd*)

W eksperymencie wykorzystano również możliwość skorzystania z tej biblioteki za pośrednictwem wywołania z linii polecenia. Klasa *loader* zajmuje się analizą parametrów wywołania i wywołaniem właściwych procedur biblioteki *TransX*.

```
java oracle.xml.transx.loader -u -o "jdbc:oracle:oci8:@ora" zakop zakop filename.xml
```

4. Zastosowane metody eksportu danych w formacie XML

Pakiet XSU obsługiwany w Java (*xsu_java*)

Zaprezentowana poniżej klasa jest prostym przykładem wykorzystania biblioteki *XSU* z poziomu języka Java. Jako parametr podaje się zapytanie SQL, które następnie jest konwertowane do postaci dokumentu XML, który w dalszej kolejności jest wysyłany do pliku.

```
public class xsu_eksport {
    public static void main(String args[]) throws SQLException {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn =
            DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ora",
                                      "zakop", "zakop");
        OracleXMLQuery que = new OracleXMLQuery(conn, "select * from KSIAZKI");
        que.setDateFormat("dd-mm-yyyy");
        que.setRowTag("KSIAZKA");
        String xmlString = que.getXMLString();
        try {
            File outputFile = new File(args[0]);
            FileWriter out = new FileWriter(outputFile);
            out.write(xmlString);
            out.close();
        } catch (Exception e) {
            System.out.println("Byk!");
        }
        conn.close();
    }
}
```

Pakiet XSU obsługiwany z linii polecenia (*xsu_cmd*)

Podobnie jak w przypadku importu, tutaj również można skorzystać z biblioteki *XSU* poprzez wywołanie klasy *OracleXML* z odpowiednimi parametrami:

```
java OracleXML getXML -user "zakop/zakop" -conn "jdbc:oracle:oci8:@ora" -
rowTag "KSIAZKA" -dateFormat dd-mm-yyyy "select * from ksiazki" > fname.xml
```

W tym przypadku jednakże dokument XML jest wyświetlany na standardowe wyjście, więc należy zadbać o jego przekierowanie do pliku.

Pakiet XSU wywoływany z PL/SQL (*xsu_plsql*, *xsu_plsql_gen*)

Zaprezentowany poniżej listing przedstawia fragment procedury PL/SQL, która korzysta z biblioteki *XSU*. Sama biblioteka napisana jest w języku Java, a zatem wywołanie jej procedur wymaga zastosowania specjalnych procedur interfejsowych. Procedury te instalowane są standardowo w postaci pakietu *DBMS_XMLQuery*.

```
procedure exp_xml(filename in varchar2) is
  dst_loc      clob;
  c            dbms_xmlquery.ctxType;
BEGIN
  dbms_lob.createtemporary(dst_loc, TRUE);
  c := dbms_xmlquery.newContext('select * from KSIAZKI');
  dbms_xmlquery.setRowTag(c, 'KSIAZKA');
  dbms_xmlquery.setRowsetTag(c, 'KSIAZKI');
  dbms_xmlquery.setDateFormat(c, 'dd-mm-yyyy');
  dst_loc := dbms_xmlquery.getXML(c);
  DBMS_XMLquery.closeContext(c);

  ....
  /* zapisanie LOBa do pliku */
  ....
END;
```

Zoptymalizowaną wersją pakietu *DBMS_XMLQuery* jest pakiet *DBMS_XMLGen*, który napisany w języku C, jest kompilowany do kodu zintegrowanego z jądrem serwera bazy danych. Wadą tego rozwiązania jest to, że nie można ustawić własnego formatu daty.

```
procedure exp_xml_gen(filename in varchar2) is
  dst_loc      clob;
  c            dbms_xmlquery.ctxType;
BEGIN
  dbms_lob.createtemporary(dst_loc, TRUE);
  c := dbms_xmlgen.newContext('select * from KSIAZKI');
  dbms_xmlgen.setRowTag(c, 'KSIAZKA');
  dbms_xmlgen.setRowsetTag(c, 'KSIAZKI');
  dst_loc := dbms_xmlgen.getXML(c);
  DBMS_XMLgen.closeContext(c);

  ....
  /* zapisanie LOBa do pliku */
  ....
END;
```

Pakiet TransX obsługiwany z poziomu języka Java (*transx_java*)

Wykorzystanie biblioteki *TransX* z poziomu języka Java jest zaprezentowane na poniższym listingu:

```
public class exp_transx {
    public static void main(String args[]) throws SQLException {
        try {
            String s[] = {"*"};
            FileWriter out = new FileWriter(new File(args[0]));
            TransX transx = loader.getLoader();
            transx.open("jdbc:oracle:thin:@localhost:1521:ora", "zakop", "zakop");
            transx.setValidationMode( false );
            out.write("<?xml version='1.0'?>\n");
            transx.unload( "KSIAZKI", s, out);
            transx.close();
            out.close();
        } catch (Exception e) {
            e.printStackTrace(System.out);
        }
    }
}
```

Pakiet TransX obsługiwany z linii polecenia (*transx_cmd*)

Przykładowe wywołanie biblioteki *TransX* z poziomu linii polecenia zaprezentowano poniżej:

```
java oracle.xml.transx.loader -s "jdbc:oracle:oci8:@ora" zakop zakop fname.xml
KSIAZKI
```

Biblioteka SQLX (forest)

Pakiet SQLX to możliwość wywołania funkcji bezpośrednio w zapytaniach SQL. Ponieważ w eksperymencie chodzi tylko o wyeksportowanie danych jednej tabeli, wystarczy skorzystać z funkcji, która poszczególne wartości wzięte z kolumn otoczy znacznikami. Nazwy tych znaczników będą się pokrywać z nazwami kolumn/aliasów.

```
SELECT '<KSIAZKA>'||XMLForest(tytul,
                                autor, to_char(data_wyd, 'dd-mm-yyyy') as
dt_wyd,
                                nr_wyd)||'</KSIAZKA>'
FROM ksiazki;
```

Powyższe zapytanie uruchamiane może być np. w narzędziu SQL*Plus, a rezultat przekierowany do pliku instrukcją *spool*. Aby wygenerowany plik był poprawny, w sensie XML, należy jeszcze uzupełnić go o prolog oraz znacznik główny.

Standardowy SQL (sql)

Otoczanie znacznikami poszczególnych wartości, można zrobić również „ręcznie” wykorzystując konkatencję ciągów znakowych.

```
select '<KSIAZKA>'||'<TYTUL>'||tytul||'</TYTUL>'
        ||'<AUTOR>'||autor||'</AUTOR>'
        ||'<DATA_WYD>'||to_char(data_wyd, 'dd-mm-
yyyy')||'</DATA_WYD>'
        ||'<NR_WYD>'||nr_wyd||'</NR_WYD>'
        ||'</KSIAZKA>'
from ksiazki;
```

W powyższym przykładzie nie wykorzystuje się żadnych bibliotek XML-owych. Jest to możliwe tylko dlatego, że generowane dane nie zawierają zagnieżdżeń.

5. Warunki eksperymentu

Ekspertyment wykonywano na serwerze Oracle 9iR2 w środowisku Windows 2000. Zarówno serwer jak i dane znajdowały się na tym samym komputerze, choć pliki danych serwera i pliki wejściowe lub generowane znajdowały się na odrębnych dyskach.

Programy bazujące na Javie realizowane po stronie serwera były wykonywane w zintegrowaną z serwerem wirtualną maszyną Javy firmy Oracle. Program realizowane po stronie klienta (choć na tym samym komputerze) wykonywane były w wirtualnej maszynie Javy formy Sun Microsystems. Oba środowiska zgodne są ze specyfikacją wersji 1.3.1_01.

Ekspertyment wykonywano na plikach o różnych wielkościach. Za każdym razem struktura danych i same dane były takie same:

```
<?xml version="1.0"?>
<KSIEGOZBIOR>
  <KSIAZKA>
    <TYTUL>Aplikacje w Delphi. Przyklady</TYTUL>
    <AUTOR>Teresa Pamula</AUTOR>
    <DATA_WYD>01-08-2003</DATA_WYD>
    <NR_WYD>1</NR_WYD>
  </KSIAZKA>
  .....
</KSIEGOZBIOR>
```

Wyjątek stanowiło tutaj zastosowanie biblioteki *TransX*, gdzie w każdym znaczniku musiała być nazwa kolumny. Na potrzeby tego pakietu przygotowano oddzielne dokumenty XML z takimi samymi danymi:

```
<?xml version="1.0"?>
<table name="KSIAZKI">
  <lookup-key/>
  <columns>
    <column name="NR_WYD" type="number"/>
    <column name="TYTUL" type="string"/>
    <column name="AUTOR" type="string"/>
    <column name="DATA_WYD" type="string"/>
  </columns>
  <dataset>
  <row>
  <col name="TYTUL">UBKRJBBGRRJUAPABMGBMGBAMVWUAEKWLXRAQHB FOLWXRWXIXW</col>
  <col name="AUTOR">RLGNCSVITNPJOJEQXOSOTUMSUKP</col>
  <col name="DATA_WYD">1976-10-18T12:12:00</col>
  <col name="NR_WYD">7</col>
  </row>
  ...
  </dataset>
</table>
```

Dokumenty XML ładowane były bez przeprowadzania wstępnej transformacji XSLT. Transformacja taka mogłaby bowiem zaburzyć wyniki czasowe samego procesu ładowania lub generowania danych.

Dokumenty były ładowany do tablicy relacyjnej o następującej strukturze:

```
create table ksiazki(
  tytul    varchar2(100),
  autor    varchar2(50),
  data_wyd date,
  nr_wyd   number(3)
)
```

Celem eksperymentu było porównanie wydajności różnych metod ładowania danych XML do struktur relacyjnych i generowanie dokumentów XML (do pliku) ze struktur relacyjnych, przy wykorzystaniu standardowych narzędzi, których konfiguracja nie wymaga pisania własnych procedur dostosowywanych każdorazowo przy zmianie struktury pliku. Wyjątkowo, dla porównania, zamieszczono wyniki dla importu narzędziem SQL*Loader oraz importu przy pomocy własnoręcznie napisanego parsera SAX, zaś dla eksportu pokazano wyniki realizacji „zwykłego” zapytania SELECT.

Eksperyment wykonywany był wielokrotnie, dla różnych wielkości plików (od 100 rekordów do 500 tys. rekordów). Pomiar czasu rejestrowany był automatycznie - błąd pomiaru wyniósł ok. 30 ms. W niniejszej pracy przedstawiono czasy będące średnią arytmetyczną pięciu pomiarów. Serwer bazy danych był restartowany przed każdym pomiarem, aby wyeliminować wpływ buforowania danych pomiędzy różnymi doświadczeniami.

Należy zwrócić tutaj uwagę, że mimo iż niektóre procedury wywoływane są w PL/SQL, to jednak faktycznie wykonywany jest kod Javy na wirtualnej maszynie Javy zintegrowanej z serwerem Oracle. Ponieważ procedury wymagają najczęściej dużej ilości pamięci, wymaga to odpowiedniego skonfigurowania serwera i ustawienia parametru `JAVA_POOL_SIZE`. Dla programów uruchamianych po stronie klienta również należy zadbać o prawidłową konfigurację wirtualnej maszyny Javy. Ustawiono limi dostępnej pamięci dla wirtualnej maszyny Javy na 300MB. Jak się okazało, nie dla wszystkich metod była to wystarczająca ilość pamięci.

6. Wyniki eksperymentu i ich interpretacja

Import danych

Wyniki eksperymentu przedstawione są w Tabeli 1. Czas podany jest w sekundach. Jeżeli pole jest niewypełnione, oznacza to, że dla danej wielkości pliku metoda wymagała więcej niż 300MB dostępnej pamięci.

Tabela 1. Czas importu danych XML [s]

Metody importu	Liczba rekordów [tys.]							
	0,1	1	10	100	200	300	400	500
met. tradycyjne:								
sqlldr	9,52	9,51	9,96	15,54	25,36	35,00	54,89	93,10
XSU:								
plsql	14,39	9,22	14,44	84,37	-	-	-	-
xsu_java	10,10	9,92	12,14	34,09	58,77	87,01	-	-
xsu_cmd	9,37	9,81	12,59	39,32	96,03	151,69	-	-

TransX:								
transx_java	9,63	10,54	35,17	2175,55	-	-	-	-
transx_cmd	9,53	10,40	35,96	2177,50	-	-	-	-
Inne:								
sax	10,36	14,21	47,66	431,99	802,32	1304,68	1711,41	2080,84

Eksport danych

Wyniki eksperymentu przedstawione są w Tabeli 2. Czas podany jest w sekundach. Jeżeli pole jest niewypełnione, oznacza to, że dla danej liczby rekordów w tabeli metoda wymagała więcej niż 300MB dostępnej pamięci.

Tabela 2. Czas eksportu danych XML [s]

Metody eksportu	Liczba rekordów [tys.]							
	0,1	1	10	100	200	300	400	500
Met. tradycyjne:								
sql	8,55	8,59	9,73	19,85	29,45	40,02	53,83	79,01
XSU:								
plsql	13,03	10,30	41,99	936,64	2110,07	3009,45	4136,56	5145,54
plsql_gen	9,01	10,52	37,21	657,98	1375,35	1902,86	2596,06	3356,62
xsu_java	9,92	9,52	11,08	27,04	49,17	-	-	-
xsu_cmd	9,30	9,53	10,86	25,12	40,94	-	-	-
TransX:								
transx_java	23,74	42,19	28,15	83,64	-	-	-	-
transx_cmd	13,28	13,11	15,72	70,14	-	-	-	-
Inne:								
forest	9,38	8,92	11,33	58,56	167,64	337,25	592,03	923,34

Wnioski

1.

Dla niewielkiej liczby rekordów (do ok. 10 tys rekordów) czasy są porównywalne. Niezależnie od liczby rekordów czy wybranej metody czas ładowania lub eksportu danych mieści się w granicach od 10 do 40 s.

Przy tak niewielkich operacjach programista ma zatem możliwość wyboru dowolnego narzędzia.

2.

Mniej więcej począwszy od 200 tys. rekordów niektóre metody ładowania i generowania danych okazały się zawodne. W przypadku importu, część metod zgłaszając wyjątek wycofywała transakcję (*TransX*), jednak niektóre (*XSU*) po zgłoszeniu wyjątku tego nie robiły i w tabeli zostawała część importowanych danych. Podczas wdrażania należy zatem: albo importować dane

najpierw do tabeli pośredniej, albo dokładnie przetestować zastosowaną metodę używając maksymalnej wielkości spodziewanego zbioru danych.

3.

Zdecydowanie najkorzystniej wypadają metody tradycyjne. Generowanie XML przy pomocy języka SQL i „zwykłego” operatora konkatenacji oraz ładowanie danych SQL*Loader’em. Należy jednak pamiętać, że użyty w eksperymencie zbiór danych jest płaski – w elemencie głównym umieszczono kolejne rekordy, które posiadają wartości typów prostych bez żadnych zagnieżdżeń.

Dla zbiorów danych, gdzie występują zagnieżdżenia i wzajemne odwołania między rekordami wykorzystanie tych narzędzi wiąże się z koniecznością starannego ich konfigurowania. Każda najmniejsza zmiana w strukturze danych to kosztowny i długotrwały proces dostosowywania narzędzi.

4.

Biblioteki *XSU* i *TransX* mogą być wywoływane z linii polecenia, z wnętrza własnej klasy Javy, a w przypadku *XSU* również z poziomu języka PL/SQL. Dla biblioteki *TransX* nie stwierdzono znacznych różnic czasowych.

Dla biblioteki *XSU* występują różnice czasowe (sięgające nawet 60%). Dla różnych metod wywołania różny był też limit rekordów możliwych do przetworzenia w ramach dostępnych 300MB pamięci RAM.

Może to częściowo wynikać z różnicy wydajności zastosowanych maszyn wirtualnych Javy.

7. Podsumowanie

Analiza wyników dowiodła, że narzędzia związane z technologią XML wciąż są mało wydajne w stosunku do tradycyjnych formatów tekstowych dla dużych ilości danych. Co więcej, nierzadko w ogóle odmawiają współpracy, a ich uruchomienie związane jest z rezerwacją znacznej ilości pamięci RAM.

W pokazanym eksperymencie stosunkowo łatwo o wzajemną konwersję plików tekstowych na dane XML i na odwrót tak, aby skorzystać z tradycyjnych, relacyjnych narzędzi. Jednak należy pamiętać, że istnieją zastosowania, gdzie format XML może okazać się lepszy. Są to sytuacje, gdy dane obejmują więcej niż jedną „tabelę”, gdy pomiędzy danymi występują wzajemne zależności, gdy dane otrzymujemy z wielu źródeł, a każde z nich formatuje je inaczej.

Bibliografia

- [Lent03] Lentner M.: Oracle 9i kompletny podręcznik użytkownika. Wydawnictwo PJWSTK, 2003, ISBN 83-89244-05-5
- [CRZ03] Chaudhri A., Rashid A., Zicari R.: XML Data Management, Addison-Wesley, 2003, ISBN 0-201-84452-4
- [TWZ03] Traczyk T. Wojciechowski M, Zakrzewicz Z.: Referaty II Szkoła PLOUG, Poznań 2003, ISSN 1641-2117
- [CSK01] Chang B., Scardina M., Kiritzov S.: Oracle 9i XML Handbook, Osborne/McGraw-Hill, 2001, ISBN 0-07-213495-X
- [ORACLE1] Oracle 9i, API Reference XDK and Oracle XML DB, part no A96616-01
- [ORACLE2] Oracle 9i, Developer’s Guide - Oracle XML DB, part no A96620-01
- [ORACLE3] Oracle 9i, XML Developer’s Kit Guide - XDK, part no A96621-01