

# Dane relacyjne a XML – metody konwersji

Bartłomiej Jabłoński  
e-mail: bartek@math.uni.lodz.pl

*Uniwersytet Łódzki*

**Abstrakt.** Pomimo, iż od momentu opublikowania standardu XML upłynęło osiem lat wprost fantastycznego rozwoju i upowszechnienia tej technologii, to model danych relacyjnych wcale się nie przeżył. Zdecydowana większość współczesnych aplikacji bazodanowych ciągle wykorzystuje krotki rekordów (tabele) z więzami klucza obcego jako podstawę zapisu danych i nie wydaje się, żeby stan ten zmienił się w najbliższym czasie.

Jednak tam, gdzie nacisk położony jest na wymianę danych i na integrację środowisk heterogenicznych, tam XML wypiera inne (relacyjne i nierelacyjne) sposoby kodowania informacji. Ze względu na duży formalizm oraz elastyczność w tworzeniu skomplikowanych struktur danych język ten zdobył sobie uznanie, stając się tym uznanym standardem i formatem, do którego wszyscy producenci udostępniają narzędzia konwersji.

Baza danych Oracle nie jest wyjątkiem. Po pierwsze, umożliwia przechowywanie danych w postaci zarówno relacyjnej jak i XML-owej. Po drugie, umożliwia przeglądanie/zapis danych, niezależnie od postaci tych danych, w razie potrzeby zapewniając konwersję „w locie”. Referat przedstawia różne narzędzia klientki oraz biblioteki i funkcje „zaszyte” w serwerze Oracle 10g, które służą do przekształcania danych relacyjnych do formatu XML i na odwrót oraz do przekształcania danych XML w inny format XML. Poruszane są aspekty praktycznego zastosowania tych funkcji w aplikacjach, procesach ładowania i generowania dużych wolumenów danych, jak i kwestie związane z wydajnością.

**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.

## Wstęp

W praktyce programistycznej można spotkać się z problemem wymiany informacji pomiędzy różnymi systemami informatycznymi. Typowym problemem jest wtedy ustalenie wspólnego dla systemu źródłowego i docelowego formatu danych. Ostatnie lata, to przełom w tej dziedzinie – standard XML został ogólnie przyjęty i większość producentów oprogramowania w taki czy inny sposób rozszerzyło swoje produkty o możliwość komunikacji w tym języku.

Firma Oracle wprowadziła pierwsze rozwiązania oparte na XML już w wersji 8. W kolejnych wersjach narzędzia te były rozbudowywane i coraz bardziej integrowane z relacyjnym motorem danych. Niniejszy referat to próba przedstawienia możliwości Oracle w dziedzinie wymiany danych XML.

Dokument XML jest jednocześnie dokumentem tekstowym i jako taki może być przechowywany w polach typu VARCHAR2 lub CLOB. Jednak dopiero potraktowanie tego tekstu odpowiednim parserem nadaje sens poszczególnym danym, które mogą być analizowane i integrowane z innymi danymi relacyjnymi. W Oracle „znacznikiem”, że tekst zawiera treść w formacie XML jest obiektowy typ danych XMLType. W przypadku, gdy dane pochodzą ze źródeł zewnętrznych należy zaimportować je odpowiednimi narzędziami. Inne narzędzia służą do generowania danych XML pochodzących z tabel relacyjnych.

## Systematyka

Narzędzi i mechanizmów do konwersji danych XML na relacyjne i odwrotnie jest wiele. Część z nich została specjalnie przeznaczona do konwersji, w innych zaś funkcja konwersji jest tylko jedną z wielu, którą można samodzielnie zaimplementować. W zależności od sposobu wykorzystania tych narzędzi można je podzielić na następujące grupy:

- funkcje wykorzystywane w zapytaniach SQL oraz wewnętrzne mechanizmy konwersji (np. funkcje SQL/XML, opcja STORE AS w poleceniu tworzącym tabele, XPath query rewrite),
- biblioteki procedur, funkcji i klas, które mogą być wykorzystywane w różnych językach programowania (XSU, TransX, Servlet OracleXSQL, Advanced Queue),
- gotowe narzędzia wykorzystywane głównie z poziomu linii poleceń (SQL\*Loader, SQL\*Plus).

## Co nowego w 10g?

W porównaniu z poprzednią wersją (9i) Oracle 10gR2 oferuje nowe możliwości w zakresie pracy z dokumentami XML:

- uaktualniono narzędzia w celu osiągnięcia zgodności z nowymi wersjami standardów (w niektórych bibliotekach C, C++ poszczególne zgodności mogą nie być zachowane)
  - XPath 2.0 Working Draft,
  - XSLT 2.0 Working Draft,
  - DOM 3.0,
  - SAX 2.0,
- poprawiono wewnętrzną implementację pakietów PL/SQL, porządkując funkcje, ujednoliciając nazewnictwo oraz polepszając wydajność,
- dodano możliwości do serwleta XSQL (m.in. generator FOP),
- pojawiła się nowa obsługa XML Pipeline Definition Language,
- dodano wirtualną maszynę przekształceń XSL,
- obsługa XQuery została zintegrowana z modułem zapytań SQL,
- dodano nowe funkcje do modyfikacji danych XML za pomocą SQL-a,
- umożliwiono wykonywanie hierarchicznych zapytań w pakiecie DBMS\_XMLGEN,
- dołożono serwlet do AQ, integrujący kolejki ze standardem SOAP.

## XML i Oracle a sprawa polska

Standard XML powstawał od samego początku z założeniem obsługiwanie znaków narodowych kodowanych zarówno w Unicode jak i w innych kodach zebranych w normach ISO/ANSI. I chociaż parser Oracle obsługuje jednobajtowe kody ISO-8859-2 oraz CP1250, to podczas prób z wkładaniem dokumentów XML poprzez serwlet AQ XML do kolejek, sukces osiągnięto dopiero gdy baza utworzona została z kodowaniem UTF-8. Uwagę rekomendującą ten właśnie standard można również znaleźć w dokumentacji Oracle.

Użytkowników Oracle, którzy chcieliby przenieść dane z bazy korzystającej z ISO-8859 lub WIN1250 do UTF-8 należy jednak przestrzec przed kłopotami wynikającymi z przejściem do nowego standardu kodowania liter. Tekstowe dane relacyjne przechowywane są zazwyczaj w polach CHAR lub VARCHAR2, które z reguły podlegają ostrym restrykcjom dotyczącym maksymalnej długości przechowywanych napisów. Ograniczenie to wyrażane jest jednak w bajtach(!) (kolumna varchar2(10) może przechować napis o długości 10 bajtów), co w przypadku strony kodowej UTF-8 i języka polskiego może oznaczać w szczególnym przypadku, że da się umieścić napis o połowę krótszy (w kolumnie varchar2(10) może zmieścić się co najwyżej 5 znaków charakterystycznych dla polszczyzny).

## Biblioteki związane z SQL

Język SQL został rozszerzony o funkcje, które na podstawie danych relacyjnych mogą tworzyć znaczniki lub nawet całe dokumenty XML. Rozszerzenie to jest zgodne ze specyfikacją SQL/XML (SQL:2005 Standard Part 14). Poniższa tabela przedstawia jedynie kilka wybranych funkcji z tego pakietu oraz proste przykłady ich wykorzystania.

Tabela 1. Wybrane funkcje SQL/XML w Oracle 10g

Funkcja SQL/XML	Przykład:	Rezultat:
<b>XMLElement</b> – tworzenie elementów	<pre>SELECT XMLElement („Date”, hire_date) FROM hr.employees WHERE employee_id = 203;</pre>	<pre>XMLELEMENT („DATE”, HIRE_DATE) ----- &lt;Date&gt;1994-06-07&lt;/Date&gt;</pre>
<b>XMLAttribute</b> – tworzenie atrybutu	<pre>SELECT XMLElement („Emp”, XMLAttributes(e.employee_id AS „ID”, e.first_name  ' '   e.last_name AS „name”)) AS „RESULT” FROM hr.employees e WHERE employee_id &gt; 200;</pre>	<pre>RESULT ----- &lt;Emp ID=„201” name=„Michael Hartstein”&gt;&lt;/Emp&gt; &lt;Emp ID=„202” name=„Pat Fay”&gt;&lt;/Emp&gt; &lt;Emp ID=„203” name=„Susan Mavris”&gt;&lt;/Emp&gt; &lt;Emp ID=„204” name=„Hermann Baer”&gt;&lt;/Emp&gt; &lt;Emp ID=„205” name=„Shelley Higgins”&gt;&lt;/Emp&gt; &lt;Emp ID=„206” name=„William Gietz”&gt;&lt;/Emp&gt;</pre>
<b>XMLForest</b> – tworzy las elementów	<pre>SELECT XMLElement („Emp”, XMLAttributes(e.first_name  ' '   e.last_name AS „name” ), XMLForest (e.hire_date, e.department AS „department”)) AS „RESULT” FROM employees e WHERE e.department_id = 20;</pre>	<pre>RESULT ----- &lt;Emp name=„Michael Hartstein”&gt; &lt;HIRE_DATE&gt;1996-02-17&lt;/HIRE_DATE&gt; &lt;department&gt;20&lt;/department&gt; &lt;/Emp&gt; &lt;Emp name=„Pat Fay”&gt; &lt;HIRE_DATE&gt;1997-08-17&lt;/HIRE_DATE&gt; &lt;department&gt;20&lt;/department&gt; &lt;/Emp&gt;</pre>
<b>XMLSequence</b> – wybiera z dokumentu XML elementy i tworzy z nich kolekcję	<pre>SELECT value(T).getStringval() AS Attribute_Value FROM table (XMLSequence (extract ( XMLType ('&lt;A&gt;&lt;B&gt;V1&lt;/B&gt;&lt;B&gt;V2&lt;/B&gt; &lt;B&gt;V3&lt;/B&gt;&lt;/A&gt;', '/A/B'))) T;</pre>	<pre>ATTRIBUTE_VALUE ----- &lt;B&gt;V1&lt;/B&gt; &lt;B&gt;V2&lt;/B&gt; &lt;B&gt;V3&lt;/B&gt;</pre>
<b>XMLConcat</b> – łączenie znaczników	<pre>SELECT XMLConcat (XMLElement („first”, e.first_name), XMLElement („last”, e.last_name)) AS „RESULT” FROM employees e;</pre>	<pre>RESULT ----- &lt;first&gt;Den&lt;/first&gt;&lt;last&gt;Raphaely&lt;/last&gt; &lt;first&gt;Alexander&lt;/first&gt;&lt;last&gt;Khoo&lt;/last&gt; &lt;first&gt;Shelli&lt;/first&gt;&lt;last&gt;Baida&lt;/last&gt; &lt;first&gt;Sigal&lt;/first&gt;&lt;last&gt;Tobias&lt;/last&gt; &lt;first&gt;Guy&lt;/first&gt;&lt;last&gt;Himuro&lt;/last&gt; &lt;first&gt;Karen&lt;/first&gt;&lt;last&gt;Colmenares&lt;/last&gt;</pre>

Funkcje te mogą posłużyć do eksportu danych relacyjnych w postaci XML. Jeżeli chcesz wyeksportować dane z 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 (tytuł,
                                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.

Identyczny efekt można uzyskać również poprzez otaczanie znacznikami poszczególnych wartości „ręcznie” wykorzystując konkatencję ciągów znakowych:

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

Należy jednak podkreślić, że takie konkatencjonowanie własnych znaczników w postaci napisów obarczone jest większym prawdopodobieństwem popełnienia pomyłki (na przykład poprzez niedomknięcie znacznika) oraz znacznym skomplikowaniem postaci zapytania, zwłaszcza dla niepłaskiej struktury danych.

## Typ XMLType

XMLType jest typem obiektowym służącym do obsługi XML w bazie danych. Przy jego pomocy można określać typ danych zarówno dla pojedynczej kolumny jak i całej tablicy (tablicy obiektowej). Typ XMLType posiada własne metody do obsługi kilku wybranych funkcji. Najważniejszą jednak jego cechą jest to, że typ ten dla funkcji SQL/XML i wielu innych odróżnia XML od innych tekstów nieparsowanych (CDATA). Fizycznie dane tego typu przechowywane są w postaci CLOB.

Poniżej pokazano dwa sposoby deklarowania danych XML w relacyjnych strukturach tabel.

```
CREATE TABLE mytable1(
  key VARCHAR2(10) PRIMARY KEY,
  xml XMLType);
```

```
CREATE TABLE mytable2 OF XMLType;
```

Dla tak zdefiniowanych danych można wykonywać zapytania XQuery, wyszukujące i generujące dane XML. Przykładowo:

```
XQUERY
FOR $i IN ora:view(„MYTABLE2”)
RETURN $i
/
```

Result Sequence

```
-----
<?xml version=„1.0”?>
<book>
  <title>Tytuł drugi</title>
</book>
```

```
<?xml version=„1.0”?>
<book>
```

```

<title>Tytuł pierwszy</title>
</book>

```

Jeśli dla pierwszego przykładu kolumnę *xml* zdefiniować jako CLOB otrzymamy następujący wynik:

```

xquery
for $i in ora:view(„MYTABLE3“)
return $i
/

```

Result Sequence

```

-----
-
<ROW><KEY>wart1</KEY><_XXXXX_XML>&lt;?xml version=&quot;1.0&quot;?&gt;
&lt;book&gt;
  &lt;title&gt;Tytuł pierwszy&lt;/title&gt;
&lt;/book&gt;</_XXXXX_XML></ROW>

<ROW><KEY>wart2</KEY><_XXXXX_XML>&lt;?xml version=&quot;1.0&quot;?&gt;
&lt;book&gt;
  &lt;title&gt;Tytuł drugi&lt;/title&gt;
&lt;/book&gt;</_XXXXX_XML></ROW>

```

## Przechowywanie danych XML

Dokumenty XML mogą być przechowywane w polach typu XMLType (czyli de facto w CLOBach). Optymalizacja wyszukiwania musi więc polegać na mało wydajnych (w porównaniu z relacyjnymi) indeksach tekstowych. Pewnym wyjściem jest wybranie regularnie powtarzających się elementów dokumentu i zapisanie ich w postaci wartości prostych w relacyjnych tabelach. Dane z jednego dokumentu mogą być zatem przechowane jako kombinacja danych relacyjnych i XML. W skrajnych przypadkach dokument może być w całości polem CLOB lub w całości może zostać rozparcelowany na pojedyncze pola relacyjne. Mówimy wtedy o strukturze grubo- i drobnoziarnistej.

Prosty przykład struktury gruboziarnistej został pokazany w poprzednim podrozdziale. Poniżej pokazano prosty przykład, który dokumenty XML zawierające dane pracownicze przechowuje w strukturze relacyjnej. Zaletą tego sposobu przechowywania danych jest możliwość deklarowania więzów relacyjnych (tutaj pokazano to dla zapewnienia unikalności numerów telefonów w ramach tego samego pracownika) oraz indeksów (unikalność telefonów).

Po pierwsze, należy zarejestrować schemat dla przechowywanych dokumentów:

```

BEGIN DBMS_XMLSCHEMA.registerSchema('emp.xsd',
  '<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xdb="http://xmlns.oracle.com/xdb">
    <xs:element name="Employee">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="EmployeeId" type="xs:positiveInteger"/>
          <xs:element name="PhoneNumber" maxOccurs="10">
            <xs:complexType>
              <xs:attribute name="No" type="xs:integer"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>',
TRUE,

```

```

TRUE,
FALSE,
FALSE);
END;
/

```

Następnie tworzymy tabelę oraz indeks unikalny:

```

CREATE TABLE emp_tab OF XMLType
  XMLSCHEMA „emp.xsd” ELEMENT „Employee”
  VARRAY xmldata.“PhoneNumber” STORE AS TABLE phone_tab;

```

```

ALTER TABLE phone_tab ADD UNIQUE (NESTED_TABLE_ID, „No”);

```

W celu przetestowania próbujemy wstawić dwa dokumenty XML. Drugi dokument zawiera powtarzające się wartości(!):

```

INSERT INTO emp_tab
  VALUES (XMLType('<Employee>
                  <EmployeeId>1234</EmployeeId>
                  <PhoneNumber No=„1234”/>
                  <PhoneNumber No=„2345”/>
                </Employee>').createSchemaBasedXML('emp.xsd'));

```

```

INSERT INTO emp_tab
  VALUES (XMLType('<Employee>
                  <EmployeeId>3456</EmployeeId>
                  <PhoneNumber No=„4444”/>
                  <PhoneNumber No=„4444”/>
                </Employee>').createSchemaBasedXML('emp.xsd'));

```

BŁĄD w linii 1:

```

ORA-00001: naruszono więzy unikatowe (PLOUG.SYS_C008390)

```

Wykorzystanie indeksu przy zapytaniach wykorzystujących XPath (mechanizm *query rewrite*) pokazano poniżej:

```

SELECT extractValue(OBJECT_VALUE,
                   '/Employee/EmployeeId',
                   'xmlns:x=„http://www.oracle.com/emp.xsd”')
FROM emp_tab
WHERE existsNode(OBJECT_VALUE,
                '/Employee/PhoneNumber[@No=„1234”]',
                'xmlns:x=„http://www.oracle.com/emp.xsd”') = 1
/

```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	141	4 (25)	00:00:01
1	NESTED LOOPS		1	141	4 (25)	00:00:01
2	SORT UNIQUE		1	50	2 (0)	00:00:01
* 3	INDEX SKIP SCAN	SYS_C008390	1		1 (0)	00:00:01
4	TABLE ACCESS BY INDEX ROWID	EMP_TAB	1	91	1 (0)	00:00:01
* 5	INDEX UNIQUE SCAN	SYS_C008388	1		0 (0)	00:00:01

Predicate Information (identified by operation id):

```

-----
3 - access („PHONE_TAB"."No"=1234)
    filter („PHONE_TAB"."No"=1234)
5 - access („NESTED_TABLE_ID"=„EMP_TAB"."SYS_NC0000900010$")

```

## Perspektywy XML

Automatyzację procesu konwersji danych relacyjnych na XML i odwrotnie w SQL-u można również osiągnąć konstruując odpowiednie perspektywy. Przypuśćmy, że dla tabeli *Departments* ma zostać utworzony interfejs generujący dane XML. Cel ten spełnia np. perspektywa *vDept*.

```
CREATE OR REPLACE VIEW vDept
  OF XMLType WITH OBJECT ID
  (extract(sys_nc_rowinfo$, 'Employee/@empno').getNumberVal()) AS
  SELECT XMLElement („Employee“ ,
    XMLAttributes(e.employee_id AS „Empno“),
    XMLForest(e.first_name, e.last_name, e.job_id)) AS result
  FROM Employees e;
```

Ponieważ zastosowano tutaj identyfikator, więc praca z poszczególnymi elementami dokumentu XML będzie oznaczała w rzeczywistości pracę na rekordach źródłowych tabeli. Kojarząc dodatkowo perspektywę z zarejestrowanym schematem uzyskuje się dodatkowe korzyści związane z mechanizmem *query rewrite*. Co więcej można utworzyć wiele takich perspektyw prezentujących te same dane w postaci różnych hierarchii.

```
CREATE OR REPLACE VIEW vDept
  OF XMLType XMLSCHEMA „emp.xsd“ ELEMENT „Employee“
  WITH OBJECT ID (extract(sys_nc_rowinfo$, 'Employee/@empno').getNumberVal())
  AS
  SELECT XMLElement („Employee“ ,
    XMLAttributes(e.employee_id AS „Empno“),
    XMLForest(e.first_name, e.last_name, e.job_id)) AS result
  FROM Employees e;
```

## Biblioteki języków programowania

### 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 już od wersji 9i *DBMS\_XMLGen* i *DBMS\_XMLSave*, które dostarczają PL/SQL-owych interfejsów do w/w klas. Ich mankamentem jest kosztowna wydajność zwłaszcza dla dużych dokumentów XML wynikająca z konieczności zbudowania drzewa DOM. W wersji 10g pojawił się nowy pakiet *DBMS\_XMLSTORE* napisany wewnątrz w C, który korzysta z parsera SAX.

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

Biblioteka XSU jest dostępna z poziomu wywołań języków Java, PL/SQL oraz w linii poleceń w postaci przygotowanej specjalnej klasy Java. Pozwala ona na

- pobranie danych XML z pliku i umieszczenie ich w tabelach relacyjnych (*import*) oraz
- pobranie danych relacyjnych z tabeli (lub zapytania) i wygenerowanie dokumentu XML (*export*).

Pliki XML, zarówno importowane jak i eksportowane, muszą strukturą odpowiadać płaskiej (nie hierarchicznej) strukturze tabeli, zaś znaczniki muszą nazywać się tak, jak kolumny. Jeśli plik XML nie posiada takiej struktury musi być wcześniej przekształcony, np za pomocą XSLT. Poniżej przedstawiona jest przykładowa struktura użyta w kolejnych przykładach:

```
<?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>
```

### Przykładowy import w języku Java

Aby skorzystać z tej możliwości należy napisać własną klasę. Przy inicjalizacji klasy *OracleXMLSave* podaje się wskaźnik na połączenie JDBC z bazą oraz nazwę tabeli, do której będą wstawiane rekordy. 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:@host:1521:ora",
                                      "user", "pass");
        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();
    }
}

```

### Przykładowy import z linii polecenia

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 "user/pass" -conn "jdbc:oracle:oci8:@db"
    -rowTag "KSIAZKA" -dateFormat dd-mm-yyyy -fileName "fname.xml" KSIAZKI

```

### Przykładowy import w języku PL/SQL

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;

```

### Przykładowy eksport w języku Java

Zaprezentowana poniżej klasa jest prostym przykładem wykorzystania biblioteki *XSU* do eksportowania danych 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:@host:1521:ora”,
                „user”, „pass”);
        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();
    }
}

```

### Przykładowy eksport z linii polecenia

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

```

java OracleXML getXML -user „user/pass” -conn „jdbc:oracle:oci8:@db”
    -rowTag „KSIAZKA” -dateFormat dd-mm-yyyy
    „select * from ksiazki” > fname.xml

```

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

### Przykładowy eksport w języku PL/SQL

Zaprezentowany poniżej listing przedstawia fragment procedury PL/SQL, która korzysta z biblioteki *XSU*.

```

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 DBMS\_XMLSTORE (przykład importu)

Pakiet DBMS\_XMLSTORE miał stać się następcą wspomnianych wcześniej pakietu DBMS\_XMLSAVE. Został napisany w C i zlinkowany z jądrem Oracle. Pozwala to osiągnąć większą wydajność widoczną zwłaszcza dla dużych dokumentów (>100KB). Niestety posiada również swoje wady: nie pozwala na wyspecyfikowanie własnego formatu daty. Aby to zmienić należy ustawić parametr NLS\_DATE\_FORMAT przynajmniej na poziomie sesji, ale ponieważ jest to polecenie, w który, niejawnie zawarty jest COMMIT, to nie we wszystkich przypadkach to jest możliwe.

```

create or replace procedure imp_xmlstore(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_XMLSTORE.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_xmlstore.newContext('KSIAZKI');
  dbms_xmlstore.setRowTag(c, 'KSIAZKA');
  amt := dbms_xmlstore.insertXML(c, dst_loc);
  DBMS_XMLstore.closeContext(c);
  dbms_lob.freetemporary(dst_loc);
  dbms_lob.filecloseall() ;

```

```
end;
/
```

Wyniki testów czasowych dla pakietów DBMS\_XMLSAVE i DBMS\_XMLSTORE niestety wypadają na niekorzyść tego ostatniego. Dla dokumentów XML do 50MB czasy zaimportowania rekordów przy pomocy DBMS\_XMLSTORE są dwukrotnie dłuższe.

## 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 – przykład znajduje się poniżej), 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). Biblioteka ta dostępna jest wyłącznie dla języka Java.

Przykładowy plik do importu dla narzędzia TransX:

```
<?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">UBKRJBBGRRJUAPABMGBMBAMVWUAEKWLXRAQHBFOLWXRWXIXW</col>
      <col name="AUTOR">RLGNSCSVITNPJOJEQXOSOTUMSUKP</col>
      <col name="DATA_WYD">1976-10-18T12:12:00</col>
      <col name="NR_WYD">7</col>
    </row>
    ...
  </dataset>
</table>
```

## Przykład importu w języku 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:@host:1521:ora", "user", "pass");
            transx.setValidationMode( false );
            transx.load( args[0] );
            transx.close();
        } catch (Exception e) {
            e.printStackTrace(System.out);
        }
    }
}
```

## Przykład importu z linii polecenia

Ta sama klasa co poprzednio tym razem wywołana za pośrednictwem klasy loader, która pełni rolę parsera parametrów:

```
java oracle.xml.transx.loader -u -o „jdbc:oracle:oci8:@ora” user pass
fname.xml
```

### Przykład eksportu w języku Java

Wykorzystanie biblioteki *TransX* w eksporcie 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:@host:1521:ora”, „user”, „pass”);
            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);
        }
    }
}
```

### Przykład eksportu z linii polecenia

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

```
java oracle.xml.transx.loader -s „jdbc:oracle:oci8:@ora” user pass
fname.xml KSIAZKI
```

### Wykorzystanie funkcji tabelowych

Funkcje tabelowe (*pipelined functions*) można użyć do zczytywania danych z plików zewnętrznych. Ponieważ przetwarzanie jest potokowe, więc do odczytywania danych z plików XML najlepiej nadaje się parser SAX. Niestety w języku PL/SQL jest on niedostępny. Można natomiast skorzystać z implementacji obiektowej takiej funkcji, gdzie metody zostaną napisane w języku Java. Wykorzystanie takiej funkcji przedstawiono poniżej (ze względu na znaczny rozmiar, listingu samej funkcji nie pokazano):

```
SELECT *
FROM table(getXML('testXML'));
```

## Narzędzia zewnętrzne

### SQL\*Loader

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 lub XMLType. Nie jest za to potrzebna specyfikacja wewnętrznej implementacji tego typu (grubo- czy drobnoziarnista). Dane XML mogą być ładowane zarówno ścieżką pośrednią jak i bezpośrednio.

Poniżej znajduje się przykładowy plik kontrolny wraz z odnośnikami do plików XML:

```
LOAD DATA
INFILE *
INTO TABLE customer
APPEND
XMLType(XMLDATA) (
lobfn FILLER CHAR TERMINATED BY ', ',
```

```

XMLDATA LOBFILE(lobfn) TERMINATED BY EOF
)
BEGINDATA
xml/cust01.xml
xml/cust02.xml

```

## Inne narzędzia

### Mechanizm Advanced Queuing

Mechanizm AQ to implementacja kolejek. Kolejki AQ to mechanizm pozwalający na przesyłanie komunikatów pomiędzy różnymi użytkownikami systemu z zaawansowanymi funkcjami buforującymi, ustalającymi priorytet, czas rozgłaszania i ważności, wyszukiwania, niezależności od systemu transakcyjnego. Jednym z rodzajów komunikatów może być dokument XML, który może być następnie automatycznie przetwarzany i rozgłaszany do odbiorców kolejki.

Protokół SOAP został wymyślony jako następca protokołów CORBA, COM, RMI/IIOP i RPC. Główny nacisk położono w nim na możliwość komunikowania się oprogramowania zainstalowanego na różnych platformach sprzętowych, pracujących pod kontrolą różnych systemów operacyjnych i napisanych w różnych językach programowania i w różnych środowiskach. Zaletą SOAP-a jest łatwe dostosowanie się do obecnej infrastruktury sieciowej. SOAP korzysta najczęściej ze sprawdzonego protokołu HTTP, co nie wprowadza zamieszania u administratorów zapór ogniowych. @@Odpowiednik tej zapory znajduje się na serwerze SOAP, który może, na podstawie zawartości nagłówek, podpisów cyfrowych lub innych danych, klasyfikować żądania.

Komunikaty w kolejkach AQ mogą pochodzić z serwera SOAP. Oracle udostępnia serwlet, który odbiera żądania HTTP będące wiadomościami SOAP. Polecenia ładujące lub pobierające(!) dane z kolejki tworzone są w formacie IDAP (*ang. Internet Data Access Protocol*), który został zanurzony w SOAPie. Przykład takiego dokumentu, zaprezentowano poniżej:

```

<?xml version="1.0"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <AQXmlSend xmlns="http://ns.oracle.com/AQ/schemas/access">
      <producer_options>
        <destination>QS.NEW_ORDERS_QUE</destination>
      </producer_options>
      <message_set>
        <message_count>1</message_count>
        <message>
          <message_number>1</message_number>
          <message_header>
            <correlation>ORDER1</correlation>
            <sender_id>
              <agent_name>scott</agent_name>
            </sender_id>
          </message_header>
          <message_payload>
            <ORDER_TYP>
              <ORDERNO>100</ORDERNO>
              <STATUS>NEW</STATUS>
              <ORDERTYPE>URGENT</ORDERTYPE>
              <ORDERREGION>EAST</ORDERREGION>
              <CUSTOMER>
                .... (customer description)
              </CUSTOMER>
              <PAYMENTMETHOD>CREDIT</PAYMENTMETHOD>
            </ORDER_TYP>
          </message_payload>
        </message>
      </message_set>
    </AQXmlSend>
  </Body>
</Envelope>

```

```

<ITEMS>
  <ITEMS_ITEM>
    <QUANTITY>10</QUANTITY>
    <ITEM>
      .... (item description)
    </ITEM>
    <SUBTOTAL>190</SUBTOTAL>
  </ITEMS_ITEM>
</ITEMS>
<CCNUMBER>NUMBER01</CCNUMBER>
<ORDER_DATE>2000-08-23 0:0:0</ORDER_DATE>
</ORDER_TYP>
</message_payload>
</message>
</message_set>
</AQXmlSend>
</Body>
</Envelope>

```

## Potokowe przetwarzanie danych XML

Proces pobierania, walidacji, przekształceń, drukowania, ładowania, itp danych XML może zostać zautomatyzowany. Narzędzie, które w prosty, deklaratywny sposób zarządza tymi procesami to Oracle XDK 10g XML Pipeline Processor. Deklaracja zachodzących procesów znajduje się w pliku XML zgodnym ze specyfikacją W3C XML Pipeline Definition Language v1.0 (<http://www.w3.org/TR/2002/NOTE-xml-pipeline-20020228/>), która jest jeszcze w fazie notatki (NOTE).

Poniżej znajduje się przykładowy dokument sterujący walidacją i drukowaniem:

```

<?xml version="1.0" ?>
<pipeline xmlns="http://www.w3.org/2002/02/xml-pipeline">
  <param name="target" select="result.xml"/>
  <processdef name="saxparser.p"
definition="oracle.xml.pipeline.processes.SAXParserProcess"/>
  <processdef name="saxprint.p" definition="SAXPrinter"/>

  <process id="p1" type="saxparser.p" ignore-errors="true">
    <input name="pxmlsource" label="$source"/>
    <output name="sax" label="saxevents"/>
  </process>
  <process id="p2" type="saxprint.p" ignore-errors="true">
    <input name="pxmlsource" label="saxevents"/>
    <param name="isPrint" select="true"/>
    <output name="printxml" label="$result"/>
  </process>
</pipeline>

```

Sam proces można uruchomić następująco:

```

java oracle.xml.pipeline.controller.orapipe -sequential -force
  -attr source dane.xml -attr result rezultat.xml pipeline.xml

```

W definicji procesów użyto dwóch klas: SAXParserProcess to standardowa klasa, SAXPrinter to klasa napisana samodzielnie w oparciu o parser SAX wysyłająca do zdefiniowanego strumienia dane z dokumentu (ze względu na objętość listingu klasy nie zamieszczono). Rezultat działania procesora można zobaczyć poniżej:

```

<Employee>
  <EmployeeId>1234</EmployeeId>
  <PhoneNumber No="1234"/>
  <PhoneNumber No="2345"/>
</Employee>

```

## Podsumowanie

Podsumowując, Oracle 10g dostarcza wielu rozwiązań pozwalających na ładowanie danych XML do bazy relacyjnej. Wiele z nich jest deklaratywna – obsługiwana później przez wewnętrzne mechanizmy serwera – inne zaś pozwalają na tworzenie własnych aplikacji przy wykorzystaniu dostępnego API. Istotną cechą wszystkich tych narzędzi jest zgodność z powszechnie uznanymi standardami (np. XML Pipeline, SOAP, XSLT) co znacznie upraszcza proces integracji bazy danych z innymi aplikacjami biznesowymi.

## Bibliografia

1. [GrSi02] Graham S., Simeonow S., i in.: Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI. 2002 SAMS Publishing.
2. [Ja03] Jabłoński B.: Porównanie wydajności bibliotek XML-owych w Oracle 9i. Konferencja PLOUG Zakopane 2003
3. [Ja05] Jabłoński B., Włodarski M.: Usługi sieciowe WebServices. Opis wdrożenia aplikacji Rezerwacja sal., Bazy danych: modele, technologie, narzędzia. Wydawnictwo Komunikacji i Łączności 2005
4. [Ja05] Jabłoński B.: Otwarty interfejs programistyczny w Oracle 10g – tworzenie własnych funkcji agregujących, tabelowych oraz indeksów. Konferencja PLOUG, Zakopane 2005
5. [OraAQ] Oracle Streams Advanced Queuing User's Guide and Reference, B14257-01
6. [OraDB] Oracle XML DB Developer's Guide, B14259-02
7. [OraJava] Oracle XML Database XML Java API Reference B14293-01
8. [OraMet] <http://metalink.oracle.com>
9. [OraOtn] <http://otn.oracle.com>
10. [OraXDK] Oracle XML Developer's Kit Programmer's Guide, B14252-01
11. [Pa02] Palarczyk E.: Advanced Queuing w biznesie. PLOUGtki nr 21
12. [SCW04] Scardina M., Chang B., Wang J.: Oracle Database 10g, XML & SQL: Design, Build & Manage XML Applications in Java, C, C++ & PL/SQL. McGraw-Hill/Osborne 2004