

XII Seminarium PLOUG  
Warszawa  
Marzec 2006

# Standard SQL/XML w Oracle 10g release 2

Krzysztof Jankiewicz  
Krzysztof.Jankiewicz@cs.put.poznan.pl

*Politechnika Poznańska, Instytut Informatyki*

**Abstrakt.** SQL/XML jest standardem (ANSI i ISO) opisującym możliwości i sposoby przetwarzania obiektów XML w bazach danych SQL. SQL/XML pozwala na składowanie dokumentów XML w relacyjnych bazach danych, przeglądanie ich za pomocą języków XPath i XQuery, konwersję danych znajdujących się w tabelach do postaci dokumentów XML.

W pracach nad standardem SQL/XML uczestniczyły takie firmy jak Oracle, IBM, Hewlett-Packard, Microsoft.

Począwszy wersji 9i Release 2 standard SQL/XML jest częścią XML DB – bazy danych dokumentów XML (Native XML Database) będącej składnikiem Oracle Database.

Artykuł ma na celu przybliżenie czytelnikowi kluczowych elementów standardu SQL/XML oraz zakresu jego implementacji w bazie danych Oracle Database 10g Release 2.

## 1. Wstęp

Standard SQL/XML został zdefiniowany w postaci specyfikacji przygotowanej przez nieformalną grupę SQLX. W pracach nad nim brali udział przedstawiciele takich firm jak: DataDirect Technologies, Hewlett-Packard, IBM, Microsoft, Oracle, Sybase i innych. SQL/XML to ogólny standard dotyczący zastosowania SQL w ogólnie rozumianym przetwarzaniu dokumentów XML. Jego pierwsze implementacje zostały stworzone przez Oracle, IBM i DataDirect Technologies.

Pod koniec roku 2003 SQL/XML stał się częścią standardu języka SQL. Został on przedstawiony w rozdziale 14. dokumentu *ISO/IEC 9075:2003 Information technology – Database languages – SQL* opisującego standard SQL:2003.

## 2. Zakres standardu SQL/XML

Tak jak wspomniano we wstępie standard SQL/XML opisuje ogólnie rozumiane wykorzystanie standardu SQL do przetwarzania dokumentów XML. W SQL 2003 definiuje on następujące zagadnienia:

- Mapowanie pomiędzy SQL a XML. W szczególności dotyczy to mapowania pomiędzy:
  - zestawami znaków SQL a Unicode
  - tabelami, schematami i katalogami a dokumentami XML
  - identyfikatorami SQL a nazwami XML
  - typami danych SQL i XML Schema
  - wartościami SQL i XML
- Typ danych XML
- Funkcje

## 3. Mapowanie pomiędzy SQL a XML

Operacje mapowania mogą być odpowiednio parametryzowane, dając w efekcie różne wyniki. Parametryzacja może dotyczyć zakresu konwersji, sposobu reprezentowania wartości pustych itd. Podstawą i jednocześnie pierwszym krokiem do mapowania obiektów SQL na dokumenty XML jest konwersja zestawu znaków na zestaw znaków UNICODE, dopiero po tej fazie możliwe są kolejne etapy.

### Mapowanie tabel, schematów i katalogów

Mapowanie tabel, schematów oraz katalogów, rozumianych jako zbiór schematów, obejmuje konwersję ich zawartości na dokumenty XML, a także konwersje ich definicji ich na dokumenty będące schematami XML (XML Schema). Dla przykładu poniżej zostanie przedstawiony sposób konwersji tabeli do dokumentu XML

Podczas mapowania tabeli użytkownik powinien mieć możliwość decydowania o następujących jego właściwościach:

- sposobu reprezentowania wartości pustych (parametr NULLS)
- wyniku mapowania mającego postać albo lasu elementów lub też pojedynczego elementu XML (parametr TABLEFOREST)
- umieszczenia elementów wynikowych wewnątrz określonej przestrzeni nazw (parametr TARGETNS)
- utworzenia dokumentu XML będącego reprezentacją danych znajdujących się w tabeli (parametr DATA)
- utworzenia schematu XML będącego odwzorowaniem definicji tabeli (parametr METADATA)
- sposobu reprezentacji danych binarnych jako zakodowanych w postaci hexadecymalnej lub base64 (parametr ENCODING)

ID_ZESP	NAZWA	ADRES
10	ADMINISTRACJA	PIOTROWO 3A
20	SYSTEMY ROZPROSZONE	(null)
50	BADANIA OPERACYJNE	MIELZYNSKIEGO 30

**rys. 1 Zawartość tabeli ZESP**

Dla przykładu, jeżeli zawartość tabeli ZESP byłaby zgodna z rys. 1 to dokument XML będący efektem mapowania tabeli ZESP dla następujących parametrów: TABLEFOREST=False, TARGETNS='', DATA=True i NULLS='' byłby zgodny z dokumentem przedstawionym na rys. 2, który powstał w wyniku konwersji zawartości tabeli przy wykorzystaniu serwletów DBUri, składnika Oracle XML DB.

```

<?xml version="1.0" ?>
- <ZESP>
- <ROW>
  <ID_ZESP>10</ID_ZESP>
  <NAZWA>ADMINISTRACJA</NAZWA>
  <ADRES>PIOTROWO 3A</ADRES>
</ROW>
- <ROW>
  <ID_ZESP>20</ID_ZESP>
  <NAZWA>SYSTEMY ROZPROSZONE</NAZWA>
</ROW>
- <ROW>
  <ID_ZESP>50</ID_ZESP>
  <NAZWA>BADANIA OPERACYJNE</NAZWA>
  <ADRES>MIELZYNSKIEGO 30</ADRES>
</ROW>
</ZESP>

```

**rys. 2 Dokument uzyskany za pomocą Oracle XML DB DBUri Servlet**

Bardzo podobny sposób mapowania możemy uzyskać za pomocą pakietu DBMS\_XMLGEN, choć akurat w tym przypadku, konieczna jest zmiana domyślnie działającego mechanizmu. Dokument będący przykładem standardowej konwersji za pomocą pakietu DBMS\_XMLGEN przedstawiono na rys. 3, natomiast sposób jej zmiany został przedstawiony na rys. 5.

```

<?xml version="1.0" ?>
<ROWSET>
  <ROW>
    <ID_ZESP>10</ID_ZESP>
    <NAZWA>ADMINISTRACJA</NAZWA>
    <ADRES>PIOTROWO 3A</ADRES>
  </ROW>
</ROWSET>

```

**rys. 3 Domyślny sposób konwersji zapytania na dokument XML przez pakiet DBMS\_XMLGEN**

Dzięki mapowaniu schematów i katalogów użytkownik może umieścić w jednym dokumencie wynikowym dane pochodzące z wielu tabel, które znajdować się mogą w wielu różnych schematach. Standardowo mapowanie schematów umieszcza w zewnętrznym elemencie, pochodzącym od nazwy schematu mapowania tabel składowych. Mapowanie katalogów natomiast, umieszcza w zewnętrznym elemencie pochodzącym od nazwy katalogu mapowania schematów, które do mapowanego katalogu należą.

## Mapowanie identyfikatorów SQL na nazwy XML

W związku z tym, że nazwy katalogów, schematów, tabel i atrybutów w wyniku mapowania stają się w dokumencie wynikowym elementami XML, konieczne jest określenie sposobu konwersji identyfikatorów SQL na nazwy XML. Wiadomo, że nazwy XML nie mogą posiadać pewnych znaków specjalnych, znaków białych itp. Takie reguły nie dotyczą identyfikatorów SQL, które mogą przyjmować stosunkowo swobodną postać, przykładem niech będzie tabela PRAC utworzona za pomocą polecenia przedstawionego na rys. 4.

```
create table PRAC (
  "Nazwisko i:imię"    varchar2(20),
  "xmlpłaca<pod!"     number(8,2)
);
```

rys. 4 Polecenie tworzące tabelę PRAC

Mapowanie identyfikatorów na nazwy XML rozpoczyna się jak już wcześniej wspomniano od konwersji zestawu znaków wykorzystywanych do utworzenia identyfikatorów SQL na standard UNICODE. Następnie, w zależności od sposobu konwersji (parametr EV) może ona przybierać formę mapowania częściowego (ang. partially escaped) oraz mapowania pełnego (ang. fully escaped).

Podstawowe reguły obu sposobów mapowania zostały przedstawione poniżej.

Mapowanie częściowe (partially-escaped)

- Znaki, które są prawidłowe w nazwach XML nie są modyfikowane
- Znaki, które są nieprawidłowymi w nazwach XML zastępowane są przez ciąg znaków "\_xHHHH\_" lub "\_xHHHHHHHH" będący reprezentacją heksadecymalną niedozwolonego znaku.
- Znak "\_" jest zawsze zastępowany przez "\_X005F\_".
- Początkowy znak ":" jest zawsze zastępowany przez "\_x003A\_".

Mapowanie pełne (fully-escaped)

- Wszystkie reguły mapowania częściowego, a dodatkowo:
- Wszystkie znaki ":" są zastępowane przez "\_x003A\_".
- Identyfikatory SQL rozpoczynające się od ciągu "XML" (dowolna wielkość liter) są zawsze poprzedzane przez "\_xFFFF\_".

Poniżej na rys. 5 przedstawiono wykorzystanie pakietu DBMS\_XMLGEN do konwersji wyniku zapytania na dokument XML. Aby korzeń dokumentu miał nazwę zgodną z tabelą użyto procedury setRowSetTag. Z dokumentu będącego wynikiem konwersji przedstawionego na rys. 6 wynika, że konwersja stosowana przez ten pakiet to konwersja częściowa (partially-escaped)

```
DECLARE
  ctx  dbms_xmlgen.ctxHandle;
  xml  CLOB;
BEGIN
  ctx := dbms_xmlgen.newContext('SELECT * FROM PRAC where rownum=1');
  dbms_xmlgen.setRowSetTag(ctx, 'PRAC');
  xml := dbms_xmlgen.getXML(ctx);
  dbms_xmlgen.closeContext(ctx);
  dbms_output.put_line(xml);
END;
```

rys. 5 Zmiana standardowego sposobu konwersji

```
<?xml version="1.0"?>
<PRAC>
  <ROW>
    <Nazwisko_x0020_i:imię>KRZYSZTOF NOWAK</Nazwisko_x0020_i:imię>
    <xmłpłaca_x003C_pod_x0021_>1000</xmłpłaca_x003C_pod_x0021_>
  </ROW>
</PRAC>
```

### rys. 6 Dokument wynikowy – konwersja identyfikatorów SQL

Bezpośrednie mapowanie zawartości tabel na dokumenty XML dostępne jest również za pomocą funkcji `ora:view()`, z której można skorzystać w zapytaniach XQuery (możliwość użycia zapytań XQuery oraz związków XQuery ze standardem SQL/XML zostanie omówiony w późniejszych rozdziałach). W tym przypadku można zaobserwować pełne mapowanie identyfikatorów SQL. Dla przykładu zapytanie przedstawione na rys. 7 daje w wyniku dokument jak na rys. 8.

```
for $i in ora:view('PRAC')
return $i
```

### rys. 7 Zapytanie XQuery

```
<ROW>
  <Nazwisko_x0020_i_x003A_imię>KRZYSZTOF NOWAK</Nazwisko_x0020_i_x003A_imię>
  <_x0000_xmłpłaca_x003C_pod_x0021_>1000</_x0000_xmłpłaca_x003C_pod_x0021_>
</ROW>
```

### rys. 8 Dokument wynikowy

## Mapowanie typów danych

Mapowanie typów danych ma szczególne znaczenie przy tworzeniu schematów XML będących jednym z możliwych efektów mapowania tabel, schematów lub katalogów. Podstawowe zasady związane z mapowaniem typów danych podczas generacji schematów to:

- Każdy typ SQL mapowany jest na swój nazwany odpowiednik definiowany w schemacie XML jako element globalny
- Kolumny posiadające taki sam typ danych, współdzielą go w schemacie XML
- Nie ma znaczenia strona kodowa poszczególnych kolumn gdyż `xsd:string` zawiera jedynie znaki Unicode
- Dla każdej tabeli tworzone są także dwa dodatkowe typy:
  - typ wiersza – nazwa złożona z: „RowType”, nazwy katalogu, nazwy schematu i nazwy tabeli rozdzielonych znakiem „.”, oraz
  - typ tabeli – analogicznie jak dla wiersza z użyciem ciągu znaków „TableType”

Dla przykładu schemat powstały w wyniku mapowania tabeli PRAC dla mapowania pełnego powinien mieć postać przypominającą schemat przedstawiony na rys. 9 gdzie: typy `VARCHAR_20` i `DECIMAL_8_2` są odpowiednikami typów SQL wykorzystywanych przez atrybuty tabeli, typ `RowType.DEVEL.SCOTT.PRAC` jest zdefiniowany dla reprezentacji wiersza tabeli a typ `TableType.DEVEL.SCOTT.PRAC` jest zdefiniowany dla reprezentacji tabeli.

Oracle nie daje mechanizmów pozwalających na generowanie schematów XML będących odpowiednikami definicji tabel. Pewne kroki w tym kierunku można dostrzec dla przykładu w funkcji `getXML` pakietu `DBMS_XMLGEN`. Ostatni, opcjonalny parametr tej funkcji `dtDOrSchema` może w przyszłości pozwalać na wygenerowania razem z dokumentem XML będącym reprezentacją danych uzyskanych w wyniku zapytania, schematu XML lub DTD. Zadaniem tego schematu byłoby opisywanie struktury dokumentu wynikowego. Niestety wciąż jedyną dostępną wartością tego parametru jest wartość `NULL`.

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="VARCHAR_20">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="20"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="DECIMAL_8_2">
    <xsd:restriction base="xsd:decimal">
      <xsd:totalDigits value="8"/>
      <xsd:fractionDigits value="2"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="RowType.DEVEL.SCOTT.PRAC">
    <xsd:sequence>
      <xsd:element name="Nazwisko_x0020_i_x003A_imię"
        type="VARCHAR_20"/>
      <xsd:element name="_x0000_xmlpłaca_x003C_pod_x0021_"
        type="DECIMAL_8_2"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="TableType.DEVEL.SCOTT.PRAC">
    <xsd:sequence>
      <xsd:element name="row" type="RowType.DEVEL.SCOTT.PRAC"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

**rys. 9 Schemat XML będący mapowaniem definicji tabeli PRAC**

Oczywiście to, że nie istnieje funkcja generująca schemat XML będący odpowiednikiem definicji tabeli, nie oznacza, że w bazie danych Oracle schematy XML nie są wykorzystywane lub, że generacja jakichkolwiek schematów nie jest dostępna. Pełne omówienie roli schematów XML w bazie danych Oracle wykracza jednak poza zakres tego opracowania, wynika to z faktu, że zagadnienia te nie wynikają bezpośrednio ze standardu SQL/XML. Jedyne jako przykład generacji schematów przedstawiono poniżej na rys. 11 generację schematów XML na podstawie definicji typów obiektowych przy wykorzystaniu pakietu DBMS\_XMLSCHEMA.

Dokumenty uzyskiwane w ten sposób znacząco odbiegają od reguł zdefiniowanych w standardzie SQL/XML, przykładowo typy atrybutów nie są definiowane na poziomie globalnym schematu, oraz nie są one wystarczająco ograniczane.

```

create type PRAC2 as object (
  "Nazwisko i:imię" VARCHAR2(20),
  "xmlpłaca<pod!" NUMBER(8,2));

```

**rys. 10 Utworzenie typu obiektowego**

```
select dbms_xmlschema.GENERATESCHEMA('SCOTT','PRAC2','PRAC') schemat
from dual;
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" . . .>
  <xsd:element name="PRAC" type="PRAC2Type"
    xdb:SQLType="PRAC2" xdb:SQLSchema="SCOTT"/>
  <xsd:complexType name="PRAC2Type"
    xdb:SQLType="PRAC2" xdb:SQLSchema="SCOTT"
    xdb:maintainDOM="false">
    <xsd:sequence>
      <xsd:element name="Nazwisko_x0020_i:imię"
        xdb:SQLName="Nazwisko i:imię"
        xdb:SQLType="VARCHAR2">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:maxLength value="20"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="xmlpłaca_x003C_pod_x0021_"
        type="xsd:double"
        xdb:SQLName="xmlpłaca&lt;pod!"
        xdb:SQLType="NUMBER"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

rys. 11 Generacja schematu XML w oparciu o definicję typu obiektowego

## Mapowanie wartości

Tak jak we wcześniejszych typach mapowania również w przypadku mapowania wartości rozpoczyna się ono od konwersji zestawu znaków do standardu UNICODE. Ponadto format danych umieszczonych w dokumencie XML powinien być zgodny z prawidłowym formatem XML dla danego typu danych. Standard SQL/XML definiuje bardzo dokładnie sposób postępowania przy konwersji wartości danych SQL na XML. W większości przypadków mechanizmy bazy danych Oracle nie dokonują samodzielnie konwersji wartości na ich odpowiedniki zgodne z typami XML. Konwersje pozostawia się użytkownikowi, który może ją dokonać za pomocą odpowiednich zmiennych środowiskowych bądź funkcji konwersji. Dla przykładu dokument XML będący reprezentacją danych zawartych w tabeli utworzonej poleceniem przedstawionym na rys. 12 ma postać jak na rys. 13. Niestety wartości obu kolumn są niepoprawne. Gdyby była to konwersja zgodna z regułami określonymi przez standard SQL/XML to wartości kolumn DAY\_TO\_SEC oraz DAY powinny w dokumencie XML spełniać odpowiednio reguły narzucone przez wzorce:

```
<xsd:pattern value="-?P\p{Nd}{PLIT}DT\p{Nd}{2}H\p{Nd}{2}MSECS"/>
<xsd:pattern value="\p{Nd}{4}-\p{Nd}{2}-\p{Nd}{2}"/>
```

Dokument poprawnie mapujący wartość danych z tabeli TEST\_MV został przedstawiony na rys. 14

```
create table TEST_MV (
  DAY_TO_SEC INTERVAL DAY TO SECOND,
  DAY         DATE
);
```

rys. 12 Polecenie tworzące tabelę TEST\_MV do testu konwersji wartości SQL

```
<?xml version="1.0"?>
<TEST_MV>
  <ROW>
    <DAY_TO_SEC>+00 00:10:22.000000</DAY_TO_SEC>
    <DAY>11-MAR-06</DAY>
  </ROW>
</TEST_MV>
```

rys. 13 Wynik konwersji tabeli TEST\_MV na dokument XML (dbURI Servlet)

```
<?xml version="1.0"?>
<TEST_MV>
  <ROW>
    <DAY_TO_SEC>P00DT00H1020MSEC</DAY_TO_SEC>
    <DAY>2006-03-11</DAY>
  </ROW>
</TEST_MV>
```

rys. 14 Prawidłowa postać dokumentu XML będącego mapowaniem tabeli TEST\_MV

## 4. Typ danych XML

W większości systemów zarządzania bazą danych użytkownicy mogą przechowywać dokumenty XML za pomocą typów VARCHAR, CLOB lub też w postaci dekomponowanej na kolumny typu prostego znajdujące się w jednej lub wielu tabelach. Wszystkie te metody mają swoje wady, najważniejszą jest to, że SZBD „nie wie”, że przetwarza dokumenty XML. W związku z tym sposób przetwarzania nie uwzględnia specyfiki struktur oraz metod przetwarzania XML. Ma to istotny wpływ na wydajność.

SQL/XML wprowadza nowy typ danych mający na celu zwiększenie funkcjonalności i wydajności przetwarzania dokumentów XML. Typ ten nazwany został po prostu XML i wg standardu można wykorzystywać go do definiowania: kolumn tabel, zmiennych, oraz parametrów procedur i funkcji.

Prawidłowymi wartościami nowego typu XML są:

- dokumenty XML
- pojedyncze elementy XML
- lasy elementów XML
- węzły tekstowe
- węzły z zawartością mieszaną (podelementami i węzłami tekstowymi)

Nieprawidłowe wartości to: atrybuty, komentarze XML oraz instrukcje przetwarzania. Mogą one wystąpić jednakże jako składowe elementów będących wartością typu XML.

Odpowiedzią na wymóg istnienia typu XML jest w przypadku bazy danych Oracle typ obiektowy XMLType. Jego obiektowy charakter umożliwia wykorzystywane wartości typu XMLType nie tylko w przypadkach narzucanych przez standard SQL/XML. Dla przykładu może on być wykorzystany jako:

- typ kolumn tabel, jak na rys. 15
- typ tabel obiektowych, rys. 16
- typ wynikowy dla funkcji, rys. 17
- typ parametrów, rys. 18
- typ zmiennych, rys. 18

```
create table XML_KOL (
  XML XMLType);
```

rys. 15 XMLType jako typ kolumny tabeli

```
create table XML_TABLE
  of XMLType;
```

rys. 16 XMLType jako typ tabeli

```

create function XML_FUN
  return XMLType is
begin
  return XMLType(
    '<maly>dokument</maly>');
end;
```

rys. 17 XMLType jako wynik funkcji

```

create procedure XML_PROC( A XMLType ) is
  XML_VAR XMLType := XML_FUN;
begin
  insert into XML_KOL(XML) values (A);
  insert into XML_TABLE values (XML_VAR);
end;
```

rys. 18 XMLType jako parametr i zmienna

Wartości, jakie może przyjmować typ XMLType są zależne od sposobu jego wykorzystania, dla przykładu funkcje i metody dające w wyniku obiekty typu XMLType mogą produkować zarówno dokumenty XML, jak na rys. 19, jak i lasy elementów co zostało przestawione na rys. 20. Natomiast w kolumnach typu XMLType można przechowywać już tylko obiekty XML poprawne w sensie well-formed a zatem dokumenty XML lub pojedyncze elementy XML. To samo ograniczenie dotyczy obiektów tworzonych przez konstruktor, którego przykład użycia został przedstawiony na rys. 17.

```

select sys_xmlgen('DOKUMENT',
XMLFormat('MALY')) DOK
from dual;
DOK
-----
<?xml version="1.0"?>
<MALY>DOKUMENT</MALY>
```

rys. 19 Tworzenie dokumentu XML

```

select extract(
  DBMS_XMLGEN.GETXMLTYPE(
    'select * from ZESP'),
  '//NAZWA') LAS
from dual;
LAS
-----
<NAZWA>ADMINISTRACJA</NAZWA><NAZWA>SYSTEMY
ROZPROSZONE</NAZWA><NAZWA>BADANIA OPERACYJ-
NE</NAZWA>
```

rys. 20 Las elementów XML jako wynik działania funkcji

Pełne omówienie możliwości, jakie daje przed programistą wykorzystywanie typu XMLType wykracza niestety poza ramy tego opracowania.

## 5. Funkcje

Kolejną bardzo ważną dziedziną, w której SQL/XML ustala standardy wykorzystywania SQL przy przetwarzaniu dokumentów XML są wszelkiego rodzaju funkcje. Można podzielić na dwie podstawowe kategorie:

- funkcje tworzące obiekty XML jako wynik polecenia SELECT, oraz
- funkcje adresujące fragmenty dokumentów XML lub wykonujące zapytania na dokumentach XML i wykorzystywane w operacjach SQL.

### Funkcje tworzące obiekty XML jako wynik polecenia SELECT

Funkcje, które występują w bazie danych Oracle i które należą do tej pierwszej, najbardziej popularnej, kategorii to:

- XMLElement – tworzy element XML
- XMLAttributes – dodaje do elementu XML utworzonego za pomocą funkcji XMLElement atrybuty
- XMLForest – tworzy las elementów XML
- XMLConcat – łączy wiele fragmentów XML w jeden
- XMLAgg – funkcja grupowa łącząca wiele elementów XML występujących na poziomie wierszy w jeden fragment na poziomie grupy
- XMLPI – tworzy instrukcję przetwarzania (dodana w wersji 10g release 2)
- XMLComment – tworzy komentarz XML (dodana w wersji 10g release 2)
- XMLRoot – dodaje nagłówek dokumentu XML (dodana w wersji 10g release 2)

- XMLSerialize – konwertuje dokument XML lub jego fragment na ciąg znaków lub obiekt CLOB (dodana w wersji 10g release 2)
- XMLParse – służy do parsowania ciągu znaków traktowanego jako dokument XML lub jego fragment na obiekt typu XMLType (dodana w wersji 10g release 2)

Oprócz wyżej wymienionych funkcji wchodzących w skład standardu SQL/XML, Oracle proponuje również własne funkcje, do których należą przede wszystkim:

- XMLSequence – tworzy tablicę obiektów XMLType dla przykładu na podstawie danych pobieranych przez kursor
- XMLColAttVal – funkcja, która tworzy las elementów XML o nazwie COLUMN
- XMLCDATA – tworzy obiekt CDATA, dodana w wersji 10g release 2
- SYS\_XMLGen – tworzy dokument XML dla każdego wiersza zapytania.
- SYS\_XMLAgg – funkcja grupowa, tworzy dokument XML oparty na zawartości wierszy w grupie.

Poniżej zostaną przedstawione funkcje wchodzące w zakres standardu SQL/XML i występujące w bazie danych Oracle.

### Funkcja XMLElement

Składnia:

```
XMLElement ( [Nazwa] [, Atrybuty] [, Zawartość [,Zawartość]... ] )
```

Gdzie:

Nazwa – nazwa tworzonego elementu

Atrybuty – wywołanie funkcji XMLAttributes w celu dodania do elementu zbioru atrybutów

Zawartość – zawartość tworzonego elementu XML, może być wartością skalarną lub obiektem XMLType

### Funkcja XMLAttributes

Składnia:

```
XMLAttributes (Wartość_atrybutu [AS Alias][,Wartość_atrybutu [AS Alias]...)
```

Gdzie:

Wartość\_atrybutu – wartość atrybutu

alias – opcjonalny alias definiujący nazwę atrybutu

### Funkcja XMLForest

Składnia:

```
XMLAttributes (Wartość_atrybutu [AS Alias][,Wartość_atrybutu [AS Alias]...)
```

Gdzie:

Wartość\_atrybutu – wartość atrybutu

alias – opcjonalny alias definiujący nazwę atrybutu

Powyżej przedstawione funkcje zostały wykorzystane w zapytaniu przedstawionym na rys. 21, wynik zapytania został zamieszczony poniżej. Zwróćmy uwagę na prawidłową konwersję atrybutu ZATR tabeli PR, który jest typu DATE. Niestety prawidłowa konwersja nie jest regułą w przypadku innych bardziej złożonych typów takich jak np. INTERVAL DAY TO SEC itp.

```
select XMLElement("PRAC",
                  XMLAttributes(ZATR),
                  XMLForest(NAZWA, PLACA))
from PR;
```

rys. 21 Wykorzystanie funkcji XMLElement, XMLAttributes, XMLForest

```

<PRAC ZATR="2006-02-20">
  <NAZWA>KRZYSZTOF NOWAK</NAZWA>
  <PLACA>1000</PLACA>
</PRAC>
<PRAC ZATR="2006-02-10">
  <NAZWA>TOMASZ KOWALSKI</NAZWA>
  <PLACA>2000</PLACA>
</PRAC>

```

rys. 22 Wynik działania funkcji XMLElement, XMLAttributes, XMLForest

### Funkcja XMLConcat

Składnia:

```
XMLConcat ( XMLType [, XMLType]... )
```

Gdzie:

XMLType – to obiekt typu XMLType

### Funkcja XMLAgg

Składnia:

```
XMLAgg (XMLType [ ORDER BY sort_list ] )
```

Gdzie:

XMLType – obiekt typu XMLType, tworzony na poziomie wiersza, który następnie za pomocą funkcji grupowej XMLAgg zostanie skonkatenowany

sort\_list – lista atrybutów, które określą porządek konkatencji

### Funkcja XMLPI

Składnia:

```
XMLPI ( [NAME] Identyfikator [, Wartość] )
```

Gdzie:

Identyfikator – to identyfikator (adresat) instrukcji przetwarzania, nie może być ciągiem znaków 'xml'

Wartość – treść instrukcji przetwarzania

Zastosowanie powyższych trzech funkcji przedstawia poniższy przykład.

```

select XMLElement("PRACOWNICY",
  XMLAgg (
    XMLConcat (
      XMLForest (NAZWA, PLACA) ,
      XMLElement ("Data_Zatr", ZATR)
    ) ORDER BY NAZWA DESC ),
  XMLPI (NAME "xmlpdb", 'end_od_document')
)
from PR;

```

rys. 23 Wykorzystanie funkcji XMLConcat, XMLAgg, XMLPI

```

<PRACOWNICY>
  <NAZWA>TOMASZ KOWALSKI</NAZWA>
  <PLACA>2000</PLACA>
  <Data_Zatr>2006-02-10</Data_Zatr>
  <NAZWA>KRZYSZTOF NOWAK</NAZWA>
  <PLACA>1000</PLACA>
  <Data_Zatr>2006-02-20</Data_Zatr>
  <?xmlpdb end_od_document?>
</PRACOWNICY>

```

rys. 24 Wynik działania funkcji XMLConcat, XMLAgg, XMLPI

W przykładzie tym wyniki działania funkcji XMLForest i XMLElement zostają połączone w jeden obiekt typu XMLType za pomocą funkcji XMLConcat. Połączenie to zostanie wykonane dla każdego wiersza w grupie. Następnie, wyniki z poszczególnych wierszy zostaną skonkatelowane za pomocą funkcji grupowej XMLAgg, zanim jednak do tego dojdzie zostaną one uszeregowane wg wartości znajdujących się w kolumnie NAZWA. Na zakończenie do wyniku działania funkcji XMLAgg zostanie dołożona instrukcja przetwarzania xmldb a całość umieszczona we wnętrzu elementu PRACOWNICY za pomocą funkcji XMLElement.

### Funkcja XMLComment

Składnia:

XMLComment (Wartość)

Gdzie:

Wartość – treść komentarza

### Funkcja XMLRoot

Składnia:

XMLRoot ( Zawartość, VERSION Wersja [, STANDALONE WartośćStandalone )

Gdzie:

Zawartość – jest zawartością dokumentu, do którego zostanie dołożony nagłówek

Wersja – wartość pseudoatributu version w nagłówku dokumentu

WartośćStandalone – wartość opcjonalnego pseudoatributu standalone w nagłówku dokumentu

Poniższy przykład obrazuje sposób użycia funkcji XMLRoot oraz XMLComment. Ponadto pokazuje kompatybilność funkcji wchodzących w skład standardu SQL/XML i funkcji spoza tego standardu możliwych do wykorzystania w bazie danych Oracle.

```
select XMLRoot (
           XMLElement ("DOKUMENT",
                       sys_xmlgen('TRESA',XMLFormat('MALY')),
                       XMLComment('mały dokument')),
           VERSION '1.1' )
from dual;
```

rys. 25 Przykład użycia funkcji XMLRoot oraz XMLComment

```
<?xml version="1.1"?>
<DOKUMENT>
  <MALY>TRESA</MALY>
  <!--mały dokument-->
</DOKUMENT>
```

rys. 26 Wynik zapytania wykorzystującego funkcje XMLRoot i XMLComment

### Funkcja XMLSerialize

Składnia:

XMLSerialize ( DOCUMENT|CONTENT Wartość [, AS TypDanych)

Gdzie:

DOCUMENT|CONTENT – określa, co będzie zawartością obiektu wynikowego

Wartość – jest wartością typu XMLType, która ma zostać przekonwertowana

TypDanych – jest jednym z typów danych (VARCHAR, CLOB), do którego ma zostać dokonana konwersja

### Funkcja XMLParse

Składnia:

XMLParse ( DOCUMENT|CONTENT Wartość [WELLFORMED] )

**Gdzie:**

DOCUMENT | CONTENT – określa, co będzie zawartością obiektu wynikowego

Wartość – wartość, która będzie przekonwertowana na typ XMLType

WELLFORMED – deklaracja, dzięki której parser nie będzie weryfikował poprawności parametru

Wartość w sensie well-formed.

Przykłady użycia obu powyższych funkcji przedstawiono poniżej.

```
select LENGTH(
    XMLSerialize(DOCUMENT XMLType('<MALY>TRES</MALY>') AS VARCHAR(20)))
from dual;
```

**rys. 27 Przykład użycia funkcji XMLSerialize**

```
select XMLParse(CONTENT
    'Tu byłem <OSOBA>Mały</OSOBA>' WELLFORMED).isFragment()
from dual;
```

**rys. 28 Przykład użycia funkcji XMLParse**

## Funkcje wykonujące zapytania na dokumentach XML

Do drugiej kategorii funkcji pozwalających na przeszukiwanie dokumentów XML należą przede wszystkim funkcje XMLQuery oraz XMLTable. Stanowią one pewnego rodzaju pomost pomiędzy zapytaniami wyrażanymi w języku XQuery a zapytaniami SQL.

Funkcja XMLQuery konstruuje obiekt XMLType, który jest wynikiem zapytania XQuery. Zapytanie XQuery przekazywane jest funkcji XMLQuery jako jej pierwszy parametr. W zależności od budowy zapytania XQuery wynikiem funkcji może być zarówno pojedynczy element XML lub las elementów.

Funkcja XMLTable stanowi pewnego rodzaju rozwinięcie XMLQuery. Jej zadaniem jest stworzenie wirtualnej perspektywy, której wiersze będą pochodziły z dekompozycji wyniku uzyskanego z ewaluacji zapytania XQuery.

Standardowo źródłem danych dla zapytań XQuery są pojedyncze dokumenty XML lub ich kolekcje, czyli nazwane zbiory dokumentów XML.

W bazie danych Oracle dokumentem dla zapytań XQuery może być:

- dokument przechowywany w repozytorium XML
- dokument znajdujący się w tabeli lub kolumnie typu XMLType
- dokument zewnętrzny dostępny np. za pomocą protokołu http

Jako kolekcja dokumentów może natomiast być potraktowany:

- zbiór dokumentów znajdujący się w określonym katalogu repozytorium XML
- wiersze tabeli lub perspektywy, reprezentowane w postaci obiektów XML o strukturze zbliżonej do mapowania SQL/XML

Poniżej przedstawione zostaną pokrótce obie funkcje.

### Funkcja XMLQuery

Składnia:

```
XMLQuery ( XQuery [ PASSING { BY REF | BY VALUE } ListaParametrów ]
    RETURNING { CONTENT | SEQUENCE } { BY REF | BY VALUE } )
```

Gdzie:

XQuery – to ciąg znaków zawierający zapytanie XQuery

ListaParametrów – lista parametrów rozdzielana przecinkami, która umożliwi przypisanie zmiennym występującym w zapytaniu XQuery wartości SQL. Składnia każdego z parametrów jest następująca:

```
Wartość AS NazwaZmiennej [ BY REF | BY VALUE ]
```

Gdzie:

Wartość – wartość SQL, która zostanie przypisana do zmiennej

NazwaZmiennej – nazwa zmiennej, do której zostanie przyporządkowana wartość SQL. Sposób przekazywania wartości za pomocą referencji (BY REF) lub kopiowania wartości (BY VALUE) dotyczy tylko wartości typu XML. W przypadku typów danych innych niż XML sposób przekazywania wartości nie może być określony, zawsze wykonywane jest kopiowanie wartości.

Wśród przekazywanych za pomocą parametrów wartości, co jedna może nie mieć określonej nazwy zmiennej, jest to przypadek, w którym wartość ta jest traktowana jako kontekst zapytania. Kontekst zapytania to źródło danych, na rzecz, którego zapytanie jest wykonywane. Wynik zapytania może być sekwencją obiektów XML (RETURNING SEQUENCE) lub też pojedynczym obiektem XML nie koniecznie poprawnym w sensie well-formed (RETURNING CONTENT).

Funkcja XMLQuery, która dostępna jest w bazie danych Oracle jest nieco okrojona. Przekazywanie wartości realizowane jest tylko za pomocą kopiowania wartości, w związku z tym klauzula BY VALUE jako domyślna może zostać pominięta. Kolejnym ograniczeniem jest to, że wynikiem funkcji XQuery nie może być sekwencją obiektów XML.

Poniżej przedstawiono kilka przykładów zapytań wykorzystujących najbardziej charakterystyczne właściwości omawianej funkcji.

```
select XMLQuery('for $l in doc("/public/bib.xml")//last
                return $l' RETURNING CONTENT)
from dual;
```

**rys. 29 Zapytanie oparte na dokumencie znajdującym się w repozytorium XML**

```
select XMLQuery('for $t in collection("/public/shakespeare")/PLAY/TITLE
                return $t' RETURNING CONTENT)
from dual;
```

**rys. 30 Zapytanie oparte na kolekcji dokumentów znajdujących się w katalogu w repozytorium XML**

Powyższe przykłady wykonują zapytania na dokumentach XML umieszczonych w repozytorium XML, które od wersji bazy danych Oracle 10g release 2 zyskało bardzo na znaczeniu. Stało się to dzięki umożliwieniu wykonywania zapytań XQuery na dokumentach w nim zawartych, przez co repozytorium XML stało się jedną z podstawowych składnic dokumentów XML w bazie danych Oracle. Efektem tego jest także to, że bazę danych Oracle można traktować już nie tylko jako bazę relacyjną czy obiektową, ale również bazę danych dokumentów XML (native XML database).

```
select XMLQuery('for $n in ora:view("ZESP")/ROW/NAZWA
                return $n' RETURNING CONTENT)
from dual;
```

**rys. 31 Zapytanie oparte na XML-owej reprezentacji danych w tabeli ZESP**

```
select XMLQuery('for $e in $D/ROW/NAZWA
                return $e'
                PASSING BY VALUE XML as D
                RETURNING CONTENT)
from XML_KOL;
```

**rys. 32 Zapytanie oparte na dokumentach umieszczonych w kolumnie typu XMLType**

Źródłem dokumentów dla zapytań XQuery zatopionych we wnętrzu funkcji XMLQuery mogą być nie tylko dokumenty znajdujące się w repozytorium XML, ale także dane znajdujące się w tabelach. Przykładem może być tutaj rys. 31 gdzie użyta funkcja ora:view konwertuje każdy pojedynczy wiersz na dokument XML w efekcie umożliwiając użytkownikowi traktowanie tabeli lub perspektywy jako kolekcję dokumentów XML. Źródłem danych może być także dowolny dokument przesłany do funkcji XMLQuery jako parametr, tak jak ma to miejsce na rys. 32 gdzie dokumentami przetwarzanymi kolejno przez zapytanie XQuery są wartości kolumny XML typu XMLType. W tym przypadku zapytanie XQuery zostanie wykonane wielokrotnie, raz dla każdego wiersza tabeli XML\_KOL.

## Funkcja XMLTable

Funkcja `XMLQuery` daje możliwość wykonywania zaawansowanych zapytań `XQuery` na dokumentach XML dając w wyniku dokumenty tego samego typu. Zadaniem funkcji `XMLTable` jest wyekstrahowanie z wynikowego dokumentu XML informacji, które mogą zostać przedstawione w postaci relacyjnej. W zależności od budowy zapytania, a także typu źródła danych, wynikiem zapytania `XQuery` może być albo pojedynczy element XML lub las elementów XML. `XMLTable` konwertuje każdy z elementów XML na oddzielną krotkę, z której zawartości konstruuje wiersz wynikowy o ściśle określonej budowie.

Składnia:

```
XMLTABLE ( [ DeklaracjePrzestrzeniNazw , ] XQuery [ PASSING ListaParametrów ]
          COLUMNS DefinicjeKolumn )
```

Gdzie:

`XQuery` – to ciąg znaków zawierający zapytanie `XQuery`

`ListaParametrów` – lista parametrów o specyfice analogicznej jak w funkcji `XMLQuery`

`DeklaracjePrzestrzeniNazw` – zawiera deklaracje przestrzeni nazw, których prefiksy mogą zostać wykorzystane w poleceniu `XQuery`

`DefinicjeKolumn` – definicje kolumn jakie mają pojawić się w wyniku funkcji `XMLTable`.

Definicja każdej z kolumn ma następującą postać:

```
NazwaKolumny TypDanych [ BY REF | BY VALUE ] [ WartośćDomyślna ]
[ PATH WyrażenieXPath ]
```

Gdzie:

`NazwaKolumny` – nazwa kolumny wyniku zapytania

`TypDanych` – typ danych kolumny

`WartośćDomyślna` – klauzula zawierająca definicję wartości domyślnej

`WyrażenieXPath` – wyrażenie XPath wskazujące w każdym elemencie wynikowym zapytania `XQuery` fragment, który ma stać się wartością kolumny

Aby przedstawić pokrótce funkcjonalność funkcji `XMLTable` przyjmijmy, że zapytanie przedstawione na rys. 31 daje w wyniku dokument jak rys. 33. Wynikiem w tym przypadku jest jedna krotka.

```
<NAZWA>ADMINISTRACJA</NAZWA>
<NAZWA>SYSTEMY ROZPROSZONE</NAZWA>
<NAZWA>BADANIA OPERACYJNE</NAZWA>
```

**rys. 33 Wynik zapytania XMLQuery  
na tabeli ZESP**

```
select *
from XMLTable(
  'for $i in ora:view("ZESP")/ROW/*
  return $i'
COLUMNS B XMLType PATH '.' );
```

**rys. 34 Zapytanie z użyciem funkcji XMLTable**

Zapytanie na rys. 34 jest zainteresowane już nie tylko elementem nazwa, ale każdym elementem znajdującym się bezpośrednio w elemencie `ROW`. Na rys. 1 przedstawiono zawartość tabeli `ZESP`, składa się ona z trzech kolumn przy czym w jednym wierszu zawartość kolumny `ADRES` jest pusta. Zatem wynikiem zapytania `XQuery` będzie las ośmiu elementów XML, każdy będący reprezentantem kolejnej kolumny w kolejnych wierszach. Każdy z tych elementów zostanie wykorzystany przez funkcję `XMLTable` do utworzenia oddzielnego wiersza. Wynik zapytania z rys. 34 został przedstawiony na rys. 36. Gdyby klauzula definiująca kolumny będące wynikiem zapytania miała postać dla przykładu taki jak na rys. 35, to jego wynik byłby zgodny z rys. 37. W tym drugim przypadku zdefiniowano dwie kolumny wyniku, pierwsza zdefiniowana za pomocą klauzuli `FOR ORDINALITY` stała się kolumną numerującą wiersze.

```

select *
from XMLTable('for $i in ora:view("ZESP")/ROW/*
              return $i'
              COLUMNS
                LP          FOR ORDINALITY,
                WARTOSC VARCHAR2(20) PATH 'text()'
              );

```

rys. 35 Zapytanie z użyciem funkcji XMLTable

	LP	WARTOSC
<ID_ZESP>10</ID_ZESP>	1	10
<NAZWA>ADMINISTRACJA</NAZWA>	2	ADMINISTRACJA
<ADRES>PIOTROWO 3A</ADRES>	3	PIOTROWO 3A
<ID_ZESP>20</ID_ZESP>	4	20
<NAZWA>SYSTEMY ROZPROSZONE</NAZWA>	5	SYSTEMY ROZPROSZONE
<ID_ZESP>50</ID_ZESP>	6	50
<NAZWA>BADANIA OPERACYJNE</NAZWA>	7	BADANIA OPERACYJNE
<ADRES>MIELZYNSKIEGO 30</ADRES>	8	MIELZYNSKIEGO 30

rys. 36 Wynik zapytania

rys. 37 Wynik zapytania

Na zakończenie jeszcze jeden przykład zapytania z wykorzystaniem funkcji XMLTable. Dzięki temu, że źródłem danych dla zapytania XQuery może być dowolny dokument przesłany do funkcji XMLTable jako parametr, może nim być dla przykładu dokument zewnętrzny. Taka sytuacja ma miejsce w zapytaniu na rys. 38. Tym razem parametr przekazywany zapytaniu XQuery nie posiada nazwy zmiennej, pod którą będzie dostępna jego wartość, dzięki temu wartość ta stała się kontekstem zapytania. Wynik zapytania będzie składał się z trzech kolumn o różnych typach danych. Dwie z nich zostały wykorzystane do odfiltrowania wyniku zapytania, który ostatecznie został przedstawiony rys. 39.

```

select * from
XMLTable('for $e in /ETATY/ETAT
          return $e '
          passing xmlparse(document(
            httpuritype('http://www.../etaty.xml').getclob())
          COLUMNS NAZWA XMLType PATH 'NAZWA',
                   PLACA_MIN NUMBER(10) PATH 'PLACA_MIN',
                   PLACA_MAX NUMBER(10) PATH 'PLACA_MAX' )
where PLACA_MIN < PLACA_MAX ;

```

rys. 38 Zapytanie oparte na zewnętrznym dokumencie XML przekazany jako parametr

NAZWA	PLACA_MIN	PLACA_MAX
<NAZWA>PROFESOR</NAZWA>	800	1500
<NAZWA>ADIUNKT</NAZWA>	510	750
<NAZWA>ASYSTENT</NAZWA>	300	500
<NAZWA>STAZYSTA</NAZWA>	150	250
<NAZWA>SEKRETARKA</NAZWA>	270	450
<NAZWA>DYREKTOR</NAZWA>	1280	2100

rys. 39 Wynik zapytania

## 6. Podsumowanie

Standard SQL/XML bez wątplenia ustala drogę wymiany danych pomiędzy światem SQL a XML. Pozwala oprzeć się na standardach, na mechanizmach, które być może w przyszłości będą powszechnie stosowane. Wprowadzenie funkcji standardu SQL/XML do bazy danych Oracle powo-

duje, że możemy nie tylko konwertować dane relacyjne do dokumentów XML, ale także wykorzystywać mechanizmy integracji, jakie niesie za sobą wykorzystywane standardów powiązanych z XML. Możliwość użycia i połączenia w pojedynczym zapytaniu XQuery danych pochodzących z dokumentów XML, z tabel relacyjnych i plików XML znajdujących się w sieci daje wręcz nieograniczone możliwości zastosowań. O wersji 10g release 2 bazy danych Oracle można rzeczywiście powiedzieć, że jest już nie tylko obiektowo relacyjna, ale także w pełni XML-owa, a jej komponent XML DB to rzeczywiście baza danych dokumentów XML w pełnym tego słowa znaczeniu.

## 7. Bibliografia

1. Implementacja SQL/XML – DataDirect Technologies  
[http://www.datadirect.com/products/sql\\_xml/csxml\\_overview/index.ssp](http://www.datadirect.com/products/sql_xml/csxml_overview/index.ssp)
2. Implementacja SQL/XML – IBM  
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0311wong/index.html>
3. Implementacja SQL/XML – Oracle  
<http://www.oracle.com/technology/tech/xml/xquery/sqlxml/index.html>
4. Specyfikacja SQL/XML – *Information technology – Database languages – SQL – Part 14: XML-Related Specifications (SQL/XML)*  
<http://www.sqlx.org/SQL-XML-documents/5FCD-14-XML-2004-07.pdf>
5. *Oracle® XML DB Developer's Guide 10g Release 2 (10.2)*
6. Andrew Eisenberg, *Jim Melton Advancements in SQL/XML*  
<http://www.sigmod.org/sigmod/record/issues/0409/11.JimMelton.pdf>