

Typ danych XMLType

Marek Wojciechowski

marek@cs.put.poznan.pl

<http://www.cs.put.poznan.pl/~marek/>

Podejścia do składowania danych XML w bazie danych

- Rozkład dokumentu XML poza bazą danych, a następnie składowanie danych w postaci wierszy jednej lub wielu tabel
 - SZBD nie wie, że przetwarza dane XML
- Składowanie dokumentów XML w kolumnie typu CLOB lub VARCHAR
 - SZBD nie wie, że przetwarza dane XML
 - Istnieje możliwość przetwarzania XML programowo z użyciem XDK
- Składowanie dokumentów XML w postaci wystąpień typu XMLType
 - SZBD jest świadomy, że przetwarza dane XML, co umożliwia mu efektywne ich przetwarzanie

XMLType

- Nowy typ danych dodany w Oracle9iR1 oferujący wewnętrzne mechanizmy przetwarzania danych XML w bazie danych
- Wystąpienia typu XMLType reprezentują dokumenty XML w SQL
- XMLType może być również używany w PL/SQL
- Interfejs typu XMLType zawiera metody do manipulacji zawartością XML
- Funkcjonalność typu XMLType jest dostępna z języków PL/SQL i Java poprzez API
- XMLType może być składowany na dwa sposoby:
 - Jako duży obiekt: CLOB (domyślnie)
 - W postaci strukturalnej: zdekomponowany do tabel

Kiedy używać XMLType ?

- Gdy istnieje konieczność kierowania zapytań do dokumentów XML lub ich części
- W celu zagwarantowania, że przetwarzane dane tekstowe rzeczywiście mają postać XML (kontrola typu)
- Dla zwiększenia wydajności ewaluacji wyrażeń XPath
- Aby umożliwić zakładanie indeksów na wyrażeniach XPath
- Aby odseparować aplikacje od szczegółów dotyczących składowania dokumentów XML w bazie danych

Tabele i kolumny typu XMLType

- Dokumenty XML mogą być składowane w postaci wystąpień typu XMLType:
 - W kolumnach typu XMLType
 - W tabelach typu XMLType

```
CREATE TABLE produkty_xml (  
    kod NUMBER(11),  
    opis XMLType  
)  
/
```

```
CREATE TABLE ksiazki_xml OF XMLType  
/
```

XMLType w poleceniach SQL (1)

- Obiekt typu XMLType można utworzyć z danych typu VARCHAR lub CLOB konstruktorem XMLType() lub statyczną metodą XMLType.createXML()

```
INSERT INTO produkty_xml
VALUES (67653829370, XMLType('<produkt>
                                <nazwa>Antena dachowa</nazwa>
                                <symbol>1709765</symbol>
                                <cena>85</cena>
                                </produkt>' ));
```

odczyt
jako
VARCHAR

odczyt
jako
CLOB

```
SQL> select opis from produkty_xml;
```

OPIS

```
-----
<produkt>
  <nazwa>Antena dachowa</nazwa>
  <symbol>1709765</symbol>
  <cena>85</cena>
</produkt>
```

```
SELECT p.opis.getStringVal()
FROM produkty_xml p
WHERE kod = 67653829370;
```

```
SELECT p.opis.getClobVal()
FROM produkty_xml p
WHERE kod = 67653829370;
```

XMLType w poleceniach SQL (2)

- Operacje na **całych** obiektach typu XMLType realizowane są tradycyjnymi poleceniami SQL

```
UPDATE produkty_xml
SET opis = XMLType('<produkt>
                    <nazwa>Antena dachowa</nazwa>
                    <symbol>1709765</symbol>
                    <cena>100</cena>
                    </produkt>')
WHERE kod = 67653829370;
```

```
DELETE FROM produkty_xml
WHERE kod = 67653829370;
```

Tworzenie wartości XMLType funkcjami SQLX

- Dokumenty XML, które mają być umieszczone w kolumnach/tabelach typu XMLType można tworzyć funkcjami SQLX
- Wartością XMLType może stać się element, a nie las elementów (!)

```
INSERT INTO produkty_xml
SELECT kod, XMLElement(NAME "produkt",
                        XMLForest(nazwa "nazwa",
                                   symbol "symbol",
                                   cena "cena" )
                        )
FROM produkty_sql;
```

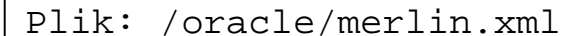
Ładowanie pliku XML do kolumny XMLType

```
CREATE TABLE myxml (doc_id NUMBER(6), doc XMLType);

CREATE DIRECTORY xmldir AS '/oracle'; ←
-- wymagane uprawnienie "CREATE ANY DIRECTORY"

DECLARE
  xmldoc CLOB;
  file BFILE;
BEGIN
  dbms_lob.createTemporary(xmldoc,true);
  file := BFileName('XMLDIR','merlin.xml');
  dbms_lob.fileOpen(file);
  dbms_lob.loadFromFile(xmldoc, file, dbms_lob.getLength(file));
  INSERT INTO myxml VALUES (1, XMLType.createxml(xmldoc));
  dbms_lob.fileClose(file);
END;
```

Plik: /oracle/merlin.xml



Funkcje SQL operujące na danych XMLType

- existsNode()
- extract()
- extractValue()
- updateXML()
- XMLSequence()
- XMLTransform()

Uwaga: Wiele funkcji operujących na XMLType jest dostępnych jako funkcje SQL i jako metody typu XMLType

Funkcja existsNode() (1)

- Sprawdza czy wartościowanie danego wyrażenia XPath względem wartości XMLType zwraca choć jeden element lub węzeł tekstowy
- Zwraca wartość liczbową 1 lub 0

```
SELECT p.kod
FROM produkty_xml p
WHERE p.opis.existsNode('/produkt[cena=85]') = 1;
```

```
SELECT kod
FROM produkty_xml
WHERE existsNode(opis, '/produkt[cena=85]') = 1;
```

```
DELETE FROM produkty_xml p
WHERE p.opis.existsNode('/produkt[cena=85]') = 1;
```

Funkcja existsNode() (2)

Odczyt wartości XMLType
z tabeli typu XMLType

```
SQL> SELECT value(p)
  2 FROM ksiazki_xml p
  3 WHERE p.existsNode('//cena[text()="109"]') = 1;
```

VALUE(P)

```
-----
<ksiazka isbn="83-7197-275-X">
  <tytul>XML. Księga eksperta</tytul>
  <cena>109</cena>
  <autorzy>
    <autor>Elliotte Rusty Harold</autor>
  </autorzy>
  <wydawnictwo>Helion</wydawnictwo>
  <rok>2000</rok>
</ksiazka>
```

Funkcja existsNode() (3)

Odczyt wartości XMLType
jako VARCHAR

```
SQL> SELECT p.getStringVal()  
 2 FROM ksiazki_xml p  
 3 WHERE p.existsNode('/ksiazka[cena=109]') = 1;
```

```
P.GETSTRINGVAL()  
-----
```

```
<ksiazka isbn="83-7197-275-X">  
<tytul>XML. Księga eksperta</tytul>  
<cena>109</cena>  
<autorzy>  
<autor>Elliotte Rusty Harold</autor>  
</autorzy>  
<wydawnictwo>Helion</wydawnictwo>  
<rok>2000</rok>  
</ksiazka>
```

Funkcja extract()

- Zwraca fragment wskazany wyrażeniem XPath jako:
 - Pojedynczy węzeł
 - Zbiór węzłów
 - Wartość tekstową
- Gdy XPath zwraca zbiór pusty, wynikiem extract() jest NULL

```
SQL> SELECT p.extract('//autor')  
      2 FROM ksiazki_xml p;
```

```
P.EXTRACT(' //AUTOR ' )
```

```
-----  
<autor>Stephen Forte</autor>  
<autor>Thomas Howe</autor>  
<autor>Kurt Wall</autor>
```

```
<autor>Paul Cassel</autor>  
<autor>Craig Eddy</autor>  
<autor>Jon Price</autor>
```

```
...
```

← Autorzy pierwszej
książki

← Autorzy drugiej
książki

Funkcja extractValue()

- Zwraca wartość skalarną odpowiadającą wynikowi wartościowania wyrażenia XPath
- Może zastąpić konstrukcję `extract().getStringVal()/getNumberVal()`

```
SQL> SELECT extractValue(value(p), '//cena/text()')
      2 FROM ksiazki_xml p;
```

```
EXTRACTVALUE(VALUE(P), '//CENA/TEXT()')
```

```
-----
```

```
79
```

```
65
```

```
59
```

```
...
```

```
SQL> SELECT extract(value(p), '//cena/text()').getNumberVal()
      2 FROM ksiazki_xml p;
```

```
EXTRACT(VALUE(P), '//CENA/TEXT()').GETNUMBERVAL()
```

```
-----
```

```
79
```

```
65
```

```
59
```

```
...
```

Funkcja updateXML()

- Modyfikuje wybrane węzły w obiekcie XMLType
- Działa na instancjach XMLType w pamięci
 - Np. na tymczasowych obiektach tworzonych w pamięci w ramach przetwarzania zapytań
- Modyfikacje w bazie danych dokonywane poleceniem UPDATE
 - Podając wartości w klauzuli SET można wykorzystać funkcję updateXML()

Operacja na tymczasowym obiekcie

```
UPDATE ksiazki_xml p
SET VALUE(p) = updateXML(VALUE(p), '//cena/text()', '56',
                        '//rok/text()', 2003)
WHERE extractValue(VALUE(p), '/ksiazka/@isbn')='83-7279-149-X'
```

Obiekt w bazie danych jest "podmieniany" w całości

updateXML() - Wielokrotna modyfikacja

- Jako argumenty updateXML() można podać kilka par: wyrażenie XPath, wartość
- Umożliwia to modyfikację kilku węzłów w jednej operacji
- Dopuszczalne jest również zmodyfikowanie tego samego węzła kilkakrotnie:
 - Wyrażenia XPath wartościowane od lewej do prawej
 - Każde kolejne wyrażenie XPath działa na wynikach poprzedniego

```
UPDATE ksiazki_xml p
SET VALUE(p) = updateXML(VALUE(p), '//cena/text()', '56',
                        '//ksiazka/rok/text()', 2004)
WHERE extractValue(VALUE(p), '/ksiazka/@isbn')='83-7279-149-X';
```

```
UPDATE ksiazki_xml p
SET VALUE(p) = updateXML(VALUE(p), '//cena/text()', '56',
                        '//ksiazka/rok', XMLType.createxml('<rok>2054</rok>'))
WHERE extractValue(VALUE(p), '/ksiazka/@isbn')='83-7279-149-X'
```

updateXML() w zapytaniach i perspektywach

```
SQL> SELECT updateXML(VALUE(p), '//cena/text()', 0)
2 FROM ksiazki_xml p
3 WHERE extractValue(VALUE(p), '/ksiazka/@isbn')='83-7279-149-X';
```

```
UPDATEXML(VALUE(P),'//CENA/TEXT()',0)
```

```
-----
<ksiazka isbn="83-7279-149-X">
  <tytul>Dane w sieci WWW</tytul>
  <cena>0</cena>
```

```
...
```

```
</ksiazka>
```

```
SQL> SELECT updateXML(VALUE(p), '//cena/text()', NULL)
2 FROM ksiazki_xml p
3 WHERE extractValue(VALUE(p), '/ksiazka/@isbn')='83-7279-149-X';
```

```
UPDATEXML(VALUE(P),'//CENA/TEXT()',NULL)
```

```
-----
<ksiazka isbn="83-7279-149-X">
  <tytul>Dane w sieci WWW</tytul>
  <cena/>
```

```
...
```

```
</ksiazka>
```

```
CREATE VIEW ksiazki_za_darmo OF XMLType AS SELECT ...
```

Indeksy funkcyjne na XMLType

- W celu skrócenia czasu odpowiedzi dla zapytań dotyczących wartości XMLType można założyć indeksy:
 - Funkcyjne: w oparciu o funkcje `extract()`, `existsNode()`, ...
 - Indeksy Oracle Text

Utworzenie indeksu funkcyjnego



```
CREATE INDEX autor_idx ON ksiazki_xml p (p.extract('//autor').getStringVal())  
/
```

Utworzenie bitmapowego indeksu funkcyjnego



```
CREATE BITMAP INDEX czy_autorzy_idx ON ksiazki_xml p (p.existsNode('//autor'))  
/
```

XMLType w języku PL/SQL

```
DECLARE
  ks_xml XMLType;
  cena NUMBER;

BEGIN

  SELECT VALUE(p) INTO ks_xml
  FROM ksiazki_xml p
  WHERE extractValue(VALUE(p), '/ksiazka/@isbn')='83-7279-149-X';

  cena := ks_xml.extract('//cena/text()').getNumberVal();

  dbms_output.put_line('Cena: ' || cena);

END;
/
```

Funkcja XMLSequence() (1)

- Rozbija pojedynczą wartość typu XMLType na kolekcję (VARRAY) wartości XMLType

```
SQL> SELECT XMLSequence(extract(VALUE(p),
2           '/ksiazka/autorzy/autor'))
3 FROM ksiazki_xml p
4 WHERE extractValue(VALUE(p), '@isbn')='83-7197-669-0'
5 /

XMLSEQUENCE(EXTRACT(VALUE(P), '/KSIAZKA/AUTORZY/AUTOR'))
-----
XMLSEQUENCETYPE(XMLTYPE(<autor>Stephen Forte</autor>
), XMLTYPE(<autor>Thomas Howe</autor>
), XMLTYPE(<autor>Kurt Wall</autor>
))
```

Wynikiem zapytania jest jeden rekord zawierający kolekcję VARRAY typu XMLSequenceType

Funkcja XMLSequence() (2)

- Dostęp do poszczególnych elementów kolekcji zwróconej przez XMLSequence() można uzyskać za pomocą funkcji TABLE()

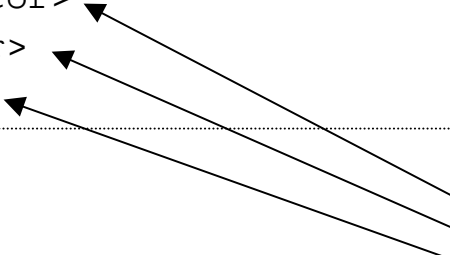
```
SQL> SELECT VALUE(t)
2  FROM TABLE(SELECT XMLSequence(extract(VALUE(p),
3                                '/ksiazka/autorzy/autor'))
4                                FROM ksiazki_xml p
5                                WHERE extractValue(VALUE(p), '@isbn')='83-7197-669-0') t
6  /
```

VALUE(T)

<autor>Stephen Forte</autor>

<autor>Thomas Howe</autor>

<autor>Kurt Wall</autor>



Poszczególne elementy kolekcji
stanowią oddzielne rekordy
wyniku zapytania

Funkcja XMLTransform() (1)

- Dokonuje transformacji dokumentu XML w oparciu o podany arkusz stylów XSL
- Zarówno dokument źródłowy jak i arkusz stylów przekazane są jako argumenty typu XMLType
- Wynikiem jest przekształcony dokument typu XMLType

```
SELECT XMLTransform(z.cv,  
                    (SELECT arkusz_stylow FROM arkusze_xsl  
                     WHERE arkusz_id = 1)  
).getStringVal()  
FROM zyciorisy z;
```

Funkcja XMLTransform() (2)

```

CREATE OR REPLACE PROCEDURE cennik_html AS
  html_doc VARCHAR2(2000);
BEGIN
  SELECT
    XMLTransform(XMLType('
      <?xml version="1.0" encoding="windows-1250" ?>
      <?xml-stylesheet type="text/xsl" href="produkty.xsl"?>
      <cennik>
        <produkt kod="67653829370"><nazwa>Antena dachowa</nazwa>
        <symbol>1709765</symbol><cena>85</cena></produkt>
        <produkt kod="56486294304"><nazwa>Radioodtworacz CAR 2001</nazwa>
        <symbol>3209765</symbol><cena>525</cena></produkt>
      </cennik>'),
    XMLType('<?xml version="1.0" encoding="windows-1250"?>
      <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
      <xsl:template match="cennik">
      <HTML><BODY><H1>Cennik akcesoriów</H1>
      <xsl:for-each select="produkt">
        <B><xsl:value-of select="nazwa"/></B> -
        <I><xsl:value-of select="cena"/></I> PLN
        <BR/>
      </xsl:for-each>
      </BODY></HTML>
      </xsl:template>
      </xsl:stylesheet>')) .getStringVal() INTO html_doc FROM dual;
  http.print(html_doc);
END;
/

```

Funkcję XMLTransform() można wykorzystać w procedurach PL/SQL, wywoływanych przez serwer WWW (za pośrednictwem mod_plsql)

| Adres | http://miner/xmlcourse/cennik_html |
|-----------------------------------|------------------------------------|
| Cennik akcesoriów | |
| Antena dachowa - 85 PLN | |
| Radioodtworacz CAR 2001 - 525 PLN | |

Sposoby składowania danych XMLType

- W dużych obiektach (LOB)
 - W Oracle9iR2 tylko CLOB (duży obiekt znakowy)
 - Dokumenty składowane w całości (jak w plikach)
 - Składowanie z zachowaniem "białych znaków"
 - Domyślny sposób składowania przy tworzeniu tabel/kolumn typu XMLType bez podania schematu XML Schema
- W postaci strukturalnej (w tabelach)
 - Dokumenty pamiętane w postaci zdekomponowanej
 - Mechanizm oparty o typy obiektowe, uzupełniony o składowanie informacji potrzebnych do zachowania DOM

Porównanie sposobów składowania

| | LOB | Strukturalne |
|---|-----------------|-----------------------|
| Elastyczność przy zmianach schematu | (+) Duża | (-) Ograniczona |
| Zachowanie źródłowej postaci | (+) Pełne | (-) Na poziomie DOM |
| Wydajność DML | (-) Średnia | (+) Duża |
| Stopień dostępności technik SQL (indeksy, ograniczenia, ...) | (-) Ograniczony | (+) Duży |
| Wymagana przestrzeń | (-) Duża | (+) Może być mniejsza |

- Jako CLOB odpowiednie gdy:
 - Dane o słabej strukturze
 - Operowanie całymi dokumentami
 - Konieczne zachowanie postaci źródłowej "bajt po bajcie"
- Strukturalne odpowiednie gdy:
 - Dane o silnej strukturze
 - Operowanie na fragmentach dokumentów
 - Wymagana walidacja dokumentów

Wsparcie dla XML Schema w Oracle XML DB

- Rejestrowanie w bazie danych schematów zgodnych z rekomendacją W3C
 - Z podziałem na lokalne (użytkownika) i globalne
 - Z możliwością korzystania ze schematów innego użytkownika
- Walidacja dokumentów w oparciu o schematy XML
- Automatyczna generacja sposobu mapowania schematu XML w schemat obiektowo-relacyjny
- Tworzenie tabel/kolumn typu XMLType opartych na schematach
- Efektywne operacje DML i zapytania na danych XMLType opartych o schemat
- Obsługa dokumentów opartych o schematy w Oracle XML DB Repository (dostęp FTP, HTTP/WebDav)

Rejestracja schematu XML w bazie danych

- Przed wykorzystaniem schematu w bazie danych należy go zarejestrować
- Do tego celu służy procedura `DBMS_XMLSCHEMA.registerSchema()`

Nazwa/URL schematu służąca do jego identyfikacji

Treść schematu jako tekst, XMLType lub URI schematu

```
DBMS_XMLSCHEMA.registerSchema('ksiazka.xsd',  
'<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <xs:element name="ksiazka">  
    <xs:complexType>  
      <xs:sequence>  
        <xs:element name="tytul" type="xs:string"/>  
        <xs:element name="cena" type="xs:float"/>  
        <xs:element name="autorzy" minOccurs="0" maxOccurs="1">  
          <xs:complexType>  
            <xs:sequence>  
              <xs:element name="autor" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>  
            </xs:sequence>  
          </xs:complexType>  
        </xs:element>  
        <xs:element name="wydawnictwo" type="xs:string"/>  
        <xs:element name="rok" type="xs:integer"/>  
      </xs:sequence>  
      <xs:attribute name="isbn" type="xs:string"/>  
    </xs:complexType>  
  </xs:element>  
</xs:schema>  
' );
```

Schematy lokalne i globalne

- Schemat XML może być zarejestrowany jako:
 - Schemat lokalny
 - Domyślnie widziany tylko przez właściciela
 - Referencja do dokumentu zawierającego schemat umieszczana w Oracle XML DB Repository w katalogu /sys/schemas/<username>
np. /sys/schemas/SCOTT/**www.myserv.com/myschema.xsd**
(dla schematu o URL "http://www.myserv.com/myschema.xsd")
 - Schemat globalny
 - Widziany przez wszystkich użytkowników
 - Tworzony przez uprzywilejowanego użytkownika
 - Referencja do dokumentu zawierającego schemat umieszczana w Oracle XML DB Repository w katalogu /sys/schemas/PUBLIC
np. /sys/schemas/PUBLIC/**www.myserv.com/myschema.xsd**
- Sposób rejestracji schematu zależy od trzeciego parametru procedury `DBMS_XMLSCHEMA.registerSchema()`
`procedure registerSchema(schemaURL IN varchar2,schemaDoc IN VARCHAR2,local IN BOOLEAN := TRUE,...)`

Rejestracja schematu XML: Infrastruktura

- Proces rejestracji schematu XML w bazie danych obejmuje operacje wykonywane niejawnie przez Oracle, mające na celu przygotowanie infrastruktury do składowania dokumentów XML zgodnych ze schematem oraz ich przetwarzania:
 - Utworzenie **obiektywnych typów SQL** umożliwiających składowanie dokumentów XML w postaci strukturalnej (obiekto-relacyjnej)
 - Utworzenie **domyślnych tabel** dla głównych elementów schematu

Typy obiektowe tworzone przy składowaniu obiektowo-relacyjnym

- Przy rejestrowaniu schematów Oracle domyślnie tworzy typy obiektowe automatycznie generując nazwy typów obiektowych, atrybutów i dobierając typy proste zgodnie z domyślnym mapowaniem typów danych XML Schema w typy SQL
- Rejestrując schemat można jawnie podać:
 - Nazwę tworzonego typu obiektowego
 - Nazwy atrybutów typu obiektowego odpowiadające elementom XML
 - Typ danych SQL dla typów prostych
np. dla float – domyślnie NUMBER, inne zgodne: FLOAT, DOUBLE
- Domyślne mapowanie "nadpisuje się" za pomocą specjalnych, specyficznych dla Oracle, znaczników w schemacie XML Schema

Specyfikacja mapowania do SQL w schemacie XML - Przykład

```

DBMS_XMLSCHEMA.registerSchema('ksiazka.xsd',
'<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xdb="http://xmlns.oracle.com/xdb">
  <xs:element name="ksiazka" xdb:SQLType="KSIAZKA_T">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="tytul" type="xs:string"/>
        <xs:element name="cena" type="xs:float" xdb:SQLName="CENA" xdb:SQLType="FLOAT"/>
        <xs:element name="autorzy" minOccurs="0" maxOccurs="1" xdb:SQLType="CLOB">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="autor" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="wydawnictwo" type="xs:string"/>
        <xs:element name="rok" type="xs:integer"/>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
');

```

```

SQL> desc KSIAZKA_T
KSIAZKA_T is NOT FINAL
Nazwa                               Wartość NULL? Typ
-----
SYS_XDBPD$                          XDB.XDB$RAW_LIST_T
isbn                                  VARCHAR2(4000)
tytul                                 VARCHAR2(4000)
CENA                                FLOAT(126)
autorzy                               CLOB
wydawnictwo                          VARCHAR2(4000)
rok                                   NUMBER(38)

```

Usunięcie schematu XML z bazy danych

- Do usunięcia zarejestrowanego schematu służy procedura `DBMS_XMLSCHEMA.deleteSchema()`
- Możliwe tryby usuwania schematu:
 - `DELETE_RESTRICT` - usunięcie schematu nie powiedzie się gdy istnieją obiekty do niego się odwołujące
 - `DELETE_INVALIDATE` – usuwa schemat i zmienia status obiektów zależnych na `INVALID`
 - `DELETE_CASCADE` – usuwa schemat i jego domyślne typy i tabele, ale nie usunie schematu gdy istnieją powiązane z nim instancje `XMLType`
 - `DELETE_CASCADE_FORCE` – jak wyżej, ale nie sprawdza czy istnieją instancje `XMLType` powiązane ze schematem, ignoruje wszelkie błędy

```
DBMS_XMLSCHEMA.deleteSchema('schemat1.xsd')
```

```
DBMS_XMLSCHEMA.deleteSchema('schemat2.xsd', DBMS_XMLSCHEMA.DELETE_CASCADE_FORCE)
```

Automatyczna generacja schematu z typu obiektowego

- Dla typu obiektowego można automatycznie wygenerować schemat w oparciu o domyślne reguły mapowania

```
CREATE TYPE Employee AS OBJECT
(
  empno NUMBER(10),
  ename VARCHAR2(200),
  sal NUMBER(8,2)
);
```

```
SQL> SELECT DBMS_XMLSCHEMA.generateSchema('SCOTT', 'EMPLOYEE') FROM dual;
```

```
DBMS_XMLSCHEMA.GENERATESCHEMA('SCOTT', 'EMPLOYEE')
```

```
-----
<xsd:schema targetNamespace="http://ns.oracle.com/xdb/SCOTT"
xmlns="http://ns.oracle.com/xdb/SCOTT" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xdb="http://xmlns.oracle.com/xdb" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/xdb http://xmlns.oracle.com/xdb/XDBSchema.xsd">
  <xsd:element name="EMPLOYEE" type="EMPLOYEEType" xdb:SQLType="EMPLOYEE"
xdb:SQLSchema="SCOTT"/>
  <xsd:complexType name="EMPLOYEEType">
    <xsd:sequence>
      <xsd:element name="EMPNO" type="xsd:double" xdb:SQLName="EMPNO" xdb:SQLType="NUMBER"/>
      <xsd:element name="ENAME" type="xsd:string" xdb:SQLName="ENAME" xdb:SQLType="VARCHAR2"/>
      <xsd:element name="SAL" type="xsd:double" xdb:SQLName="SAL" xdb:SQLType="NUMBER"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Wykorzystanie schematu w XMLType (1)

- W oparciu o zarejestrowany schemat można tworzyć tabele, perspektywy i kolumny typu XMLType
- Wskazując żądany schemat należy podać:
 - URL schematu, pod którym został zarejestrowany
 - Wybrany element główny (ang. root element)

```
CREATE TABLE ksiazki_xsd OF XMLType
XMLSCHEMA "ksiazka.xsd" ELEMENT "ksiazka";
```

```
CREATE TABLE ksiazki_xsd OF XMLType
ELEMENT "ksiazka.xsd#ksiazka";
```

```
CREATE TABLE ksiazki_col_xsd
(id NUMBER,
 pozycja XMLType
)
XMLTYPE COLUMN pozycja
XMLSCHEMA "ksiazka.xsd" ELEMENT "ksiazka";
```

↑
Skrócony zapis w notacji XPointer

Wykorzystanie schematu w XMLType (2)

- Przy wstawianiu danych XMLType do tabel opartych o schemat następuje walidacja dokumentu

Metoda statyczna, działa podobnie do konstruktora XMLType(), który również mógłby tu wystąpić

```
INSERT INTO ksiazki_xsd
VALUES(XMLType.createXML(' <ksiazka xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="ksiazka.xsd"
      isbn="83-7197-669-0">
<tytul>Access 2002. Projektowanie baz danych. Księga eksperta</tytul>
<cena>79</cena>
<autorzy>
<autor>Stephen Forte</autor>
<autor>Thomas Howe</autor>
<autor>Kurt Wall</autor>
</autorzy>
<wydawnictwo>Helion</wydawnictwo>
<rok>2002</rok>
</ksiazka>' ));
```

Błąd przy walidacji
w oparciu o schemat →

```
SQL> INSERT INTO ksiazki_xsd
      2 VALUES(XMLType.createXML('<a>well-formed document</a>'));
INSERT INTO ksiazki_xsd
      *
BŁĄD w linii 1:
ORA-19007: Schema and element do not match
```

Składowanie danych XMLType opartych o schemat

- Domyślnie dane XMLType oparte o schemat są składowane w kolumnach typów obiektowych
- Przy tworzeniu tabeli można jawnie wyspecyfikować sposób składowania w klauzuli XMLTYPE STORE AS
 - OBJECT RELATIONAL (tylko gdy w połączeniu ze schematem!)
 - CLOB

```
CREATE TABLE ksiazki_clob_xsd OF XMLType
XMLTYPE STORE AS CLOB
XMLSCHEMA "ksiazka.xsd" ELEMENT "ksiazka";
```

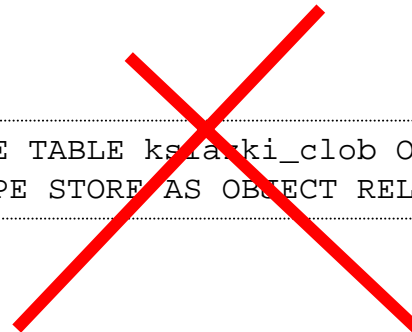
```
CREATE TABLE ksiazki_col_xsd
(id NUMBER,
 pozycja XMLType
)
XMLTYPE COLUMN pozycja STORE AS CLOB;
```

```
CREATE TABLE ksiazki_or OF XMLType
XMLTYPE STORE AS OBJECT RELATIONAL
XMLSCHEMA "ksiazka.xsd" ELEMENT "ksiazka";
```

Domyślnie

```
CREATE TABLE ksiazki_clob OF XMLType
XMLTYPE STORE AS CLOB;
```

```
CREATE TABLE ksiazki_clob OF XMLType
XMLTYPE STORE AS OBJECT RELATIONAL;
```



Dostęp do kolumn obiektowych przy składowaniu obiektowo-relacyjnym

- Bezpośredni dostęp do kolumn obiektowych, w których składowane są dane instancji XMLType jest możliwy poprzez pseudoatrybut XMLDATA

```
SQL> SELECT p.XMLDATA."tytul" FROM ksiazki_xsd p;
```

```
XMLDATA.tytul
```

```
-----
Access 2002. Projektowanie baz danych. Księga eksperta
Access 2002/XP PL dla każdego
ASP.NET. Vademecum profesjonalisty
```

← Notacja obiektowa

Notacja XML →

```
SQL> SELECT extractValue(VALUE(p), '/ksiazka/tytul') FROM ksiazki_xsd p;
```

```
EXTRACTVALUE(VALUE(P), '/KSIAZKA/TYTUL')
```

```
-----
Access 2002. Projektowanie baz danych. Księga eksperta
Access 2002/XP PL dla każdego
ASP.NET. Vademecum profesjonalisty
```

Indeksy na kolumnach obiektowych przy składowaniu obiektowo-relacyjnym

- Zakładając indeks na kolumnie obiektowej można stosować notację obiektową lub notację XML przy wyborze indeksowanych kolumn
- Niezależnie od notacji założony zostanie indeks na kolumnie obiektowej (a nie indeks funkcyjny)

```
CREATE INDEX index_1 ON ksiazki_xsd(XMLDATA."rok");
```

Indeksy na tej samej kolumnie (nie mogą istnieć jednocześnie)



```
CREATE INDEX index_2 ON ksiazki_xsd p (extractValue(value(p), '/ksiazka/rok'));
```

Ograniczenia integralnościowe przy składowaniu obiektowo-relacyjnym

- Dostęp do kolumn obiektowych przy składowaniu obiektowo-relacyjnym umożliwia zakładanie tradycyjnych ograniczeń integralnościowych (PRIMARY KEY, UNIQUE, ...) na tych kolumnach
- Ograniczenia na poziomie kolumn obiektowych, w których składowane są po dekompozycji dane XMLType, skutkują w ograniczeniach na poziomie dokumentów XML

```
ALTER TABLE ksiazki_xsd  
ADD CONSTRAINT ksiazki_pk PRIMARY KEY (XMLDATA."isbn");
```

Przepisywanie zapytań

- Gdy obiekty XMLType są przechowywane w postaci strukturalnej, wtedy niektóre zapytania wykorzystujące wyrażenia XPath w funkcjach existsNode(), extract(), extractValue() i updateXML() są przepisywane na zapytania operujące na tabelach i kolumnach źródłowych
- Przepisywanie zapytań odbywa się automatycznie
- Przepisywanie zapytań umożliwia wykorzystanie klasycznych technik indeksowania danych (np. B+ drzewo)

Przepisywanie zapytań - przykład

Zapytanie:

```
SELECT VALUE(k) FROM katalog k
WHERE extractValue(value(k), '/katalog/ksiazka/rok') =
'2002';
```

zostanie automatycznie przepisane do postaci:

```
SELECT VALUE(k) FROM katalog k
WHERE k.xmldata.ksiazka.rok = '2002';
```

Warunki przepisывania zapytań

- Przepisywaniu podlegają zapytania wykorzystujące następujące rodzaje wyrażeń XPath:
 - ścieżki prowadzące przez elementy potomne i atrybuty, np.
/katalog/ksiazka/@isbn
 - ścieżki zawierające predykaty, np.
/katalog/ksiazka[rok='2002']
 - ścieżki zawierające indeksy tablic, np.
/katalog/ksiazka[1]
- Przepisywaniu nie podlegają m.in. funkcje XPath, wyrażenia zawierające specyfikatory współrzędnych inne niż child i attribute, ścieżki zawierające *

Przepisywanie predykatów - przykład

Wyrażenie:

```
/katalog/ksiazka[rok=2002 and wydawnictwo='Mikom']
```

zostanie przepisane do postaci:

```
(XMLData."rok" = 2002 and XMLData."wydawnictwo" = "Mikom")
```

Wyrażenie:

```
/katalog/ksiazka[autorzy/autor = 'Serge Abiteboul']
```

zostanie przepisane do postaci:

```
EXISTS ( SELECT null  
FROM TABLE (XMLDATA."autorzy") x  
WHERE x."autor" = 'Serge Abiteboul' )
```

Przepisywanie ścieżek grupowych - przykład

Zapytanie:

```
SELECT extract(value(m), '/katalog/ksiazka[cena>50]/tytul')  
FROM myxml m;
```

zostanie przepisane do postaci:

```
SELECT (SELECT XMLAgg( XMLForest(x."tytul" AS "tytul"))  
FROM TABLE (XMLData."ksiazka") x  
WHERE x."cena" > 50 )  
FROM katalog_tab;
```

Walidacja obiektów XMLType

- Podczas umieszczania obiektu XMLType w tabeli, następuje automatyczne sprawdzenie, czy dokument XML jest dobrze uformowany
- Jeżeli tabela jest oparta na schemacie XML, to następuje uproszczone sprawdzenie składni wprowadzanego dokumentu XML
- Pełna walidacja obiektów XMLType jest możliwa przy wykorzystaniu dodatkowych funkcji SQL i XMLType API

Funkcje walidacji dla XMLType

- Operator **XMLIsValid()** sprawdza, czy podany dokument spełnia wskazany schemat XML,
- Metoda **schemaValidate()** sprawdza, czy podany dokument spełnia schemat XML, który jest związany z tabelą, w której znajduje się ten dokument; po pomyślnej walidacji, status dokumentu zmienia się na **VALIDATED**
- Metoda **isSchemaValidated()** zwraca informację, czy podany dokument XML był już walidowany
- Metoda **setSchemaValidated()** ustawia dokumentowi status **VALIDATED**, jednak nie dokonuje faktycznej walidacji
- Metoda **isSchemaValid()** sprawdza, czy podany dokument spełnia wskazany schemat XML; nie zmienia statusu dokumentu

Przykład wykorzystania funkcji walidacji

```
CREATE TRIGGER mtrig BEFORE INSERT OR UPDATE ON myxml FOR EACH ROW
DECLARE
    newxml xmltype;
BEGIN
    newxml := :new.sys_nc_rowinfo$;
    xmltype.schemavalidate(newxml);
END;
```

Próba umieszczenia w tabeli myxml dokumentu, który nie spełnia schematu XML związanego z tą tabelą, zakończy się zgłoszeniem błędu.

Perspektywy XMLType

- Perspektywy XMLType są perspektywami budowanymi ponad tabelami obiektowo-relacyjnymi, a zwracającymi obiekty XMLType
- Perspektywy XMLType umożliwiają przezroczystą prezentację istniejących danych relacyjnych w formacie XML
- Perspektywy XML mogą być definiowane przy użyciu schematów XML

Perspektywy XML - Przykład 1

```
CREATE VIEW emp_xml_view (doc) AS
SELECT XMLElement(NAME "EMPLOYEE", XMLForest(ename, job, sal))
FROM emp;
```

```
SELECT doc FROM emp_xml_view v
WHERE v.doc.existsNode('/EMPLOYEE[SAL>2900]')=1
```

DOC

```
<EMPLOYEE>
  <ENAME>JONES</ENAME>
  <JOB>MANAGER</JOB>
  <SAL>2975</SAL>
</EMPLOYEE>
```

```
<EMPLOYEE>
  <ENAME>SCOTT</ENAME>
  <JOB>ANALYST</JOB>
  <SAL>3000</SAL>
</EMPLOYEE>
```

...

Perspektywy XML - Przykład 2

```
CREATE VIEW emp_xml_view2 OF XMLType
WITH OBJECT ID (extract(sys_nc_rowinfo$, '/EMPLOYEE/@EMPID').getNumberVal())
AS
SELECT XMLElement(NAME "EMPLOYEE", XMLAttributes("EMPNO"),
                 XMLForest(ename, job, sal))
FROM emp;
```

```
SELECT VALUE(p)
FROM emp_xml_view2 p
WHERE p.existsNode('/EMPLOYEE[ENAME="JAMES"]') = 1

VALUE(P)
-----
<EMPLOYEE EMPNO="7900">
  <ENAME>JAMES</ENAME>
  <JOB>CLERK</JOB>
  <SAL>950</SAL>
</EMPLOYEE>
```

Tworzenie perspektyw XML opartych o schematy XML

1. Utwórz i zarejestruj schemat XML
2. Korzystając z funkcji SQL/XML, zbuduj perspektywę XMLType opartą o ten schemat

Perspektywa oparta o schemat XML (1)

Definicja i rejestracja schematu XML

```
exec dbms_xmlschema.registerSchema('empview2schema.xsd','<?xml version = "1.0"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="EMPLOYEE"><xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ENAME" type="xsd:string" minOccurs="0"/>
        <xsd:element name="JOB" type="xsd:string" minOccurs="0"/>
        <xsd:element name="SAL" type="xsd:float" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="EMPNO" type="xsd:integer"/>
    </xsd:complexType></xsd:element>
  </xsd:schema>' )
```

Perspektywa oparta o schemat XML (2)

Utworzenie perspektywy opartej na zarejestrowanym schemacie XML

```
CREATE OR REPLACE VIEW emp2_xml_view OF XMLType
XMLSCHEMA "empview2schema.xsd" ELEMENT "EMPLOYEE"
WITH OBJECT ID (extract(sys_nc_rowinfo$, '/EMPLOYEE/@EMPID').getNumberVal())
AS
SELECT XMLElement("EMPLOYEE",
    XMLAttributes(EMPNO, 'http://www.w3.org/2001/XMLSchema-instance'
    AS "xmlns:xsi",
    'empviewschema.xsd' as "xsi:noNamespaceSchemaLocation"),
    XMLForest(ename, job, sal))
FROM emp
```