

# Standard SQL/XML

## Wprowadzenie do XQuery

Marek Wojciechowski

[marek@cs.put.poznan.pl](mailto:marek@cs.put.poznan.pl)

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

# Języki zapytań dla XML

- Wraz z pojawieniem się standardu XML pojawiały się również propozycje języków zapytań dla danych XML:
  - Lorel, XML-QL, XQL
- Obecnie finalizowane są prace związane ze standaryzacją dwóch języków zapytań dla XML:
  - **SQL/XML** - zdefiniowany w postaci specyfikacji przygotowanych przez grupę SQLX  
(ogólny standard dotyczący zastosowań SQL w kontekście XML)
  - **XQuery** - opracowany przez grupę roboczą W3C XML Query Working Group
- Oracle bierze aktywny udział w pracach nad obydwoma standardami

# Język SQL/XML

- SQL/XML stanie się częścią standardu SQL (ANSI i ISO), opisującą sposoby wykorzystywania SQL w połączeniu z XML
  - Information technology - Database languages - SQL - Part 14: XML-Related Specifications (SQL/XML)
- SQL/XML rozszerza SQL o funkcje i operatory służące do przetwarzania danych XML w bazach danych
- Stan prac nad standardem:
  - przedłożony INCITS H2 (ciału zajmującemu się standardami w USA)
- Oracle XML DB jest pierwszą implementacją SQL/XML

# Język XQuery

- Język zapytań dedykowany dla XML
  - W połączeniu z XPath określany również jako XML Query
- Pozwala na przetwarzanie danych XML pochodzących z różnych źródeł (bazy danych, pliki)
- Stan prac nad standardem:
  - Working Draft; standard ciągle ewoluuje
  - Kompletna rekomendacja planowana na rok 2003
- Oracle udostępnia prototypową implementację XQuery w języku Java

# Obszary zastosowań SQL/XML i XQuery

- Kiedy używać SQL/XML?
  - Gdy część danych ma charakter strukturalny, a część semi-strukturalny
  - Gdy planujemy używanie narzędzi bazujących na SQL
  - Gdy chcemy używać jedynie ustabilizowanych standardów (SQL and SQL/XML)
- Kiedy używać XQuery?
  - Gdy wszystkie dane są w postaci dokumentów XML
  - Gdy nie opieramy się na narzędziach bazujących na SQL
  - Gdy niepełne ustalenie standardu nie stanowi przeszkody

# Zakres standardu SQL/XML

- Mapowanie między zestawami znaków SQL a Unicode
- Mapowanie identyfikatorów SQL w nazwy XML (i odwrotnie)
  - Problem znaków: spacja, `_`, `:`, prefiksu *xml/* w nazwach
- Mapowanie między typami danych SQL i XML Schema
- Mapowanie wartości SQL i XML
- Mapowanie tabel do dokumentów XML
  - Dane + XML Schema
  - Problem wartości NULL
    - Rozwiązanie 1: pominięcie znacznika
    - Rozwiązanie 2: atrybut *xsi:nil="true"*
- Typ danych *XML*
  - Dopuszczalne wartości: dokumenty, elementy, lasy elementów, węzły tekstowe, zawartość mieszana
  - Funkcje (operatory) produkujące wartości *XML*:  
XMLElement, XMLForest, XMLGen, XMLConcat, XMLAgg

# Funkcje produkujące dane *XML*

- XMLElement – tworzy element XML z danych nie-XML
- XMLForest – tworzy las elementów XML z danych nie-XML
- XMLGen – tworzy element XML z wykorzystaniem szablonu
- XMLConcat – tworzy las elementów przez konkatencję elementów XML wywiedzionych z jednego wiersza relacji
- XMLAgg – tworzy las elementów z kolekcji elementów XML wywiedzionych z różnych wierszy relacji

# SQL/XML w Oracle9iR2

- Funkcje rozszerzające SQL zgodne z SQL/XML (określane jako funkcje SQLX):
  - XMLElement, XMLForest, XMLConcat, XMLAgg (brak XMLGen !)
- Rozszerzenia standardu SQL/XML w Oracle9iR2
  - Dodatkowa funkcja SQLX: XMLColAttVal (podobna do XMLForest)
- Alternatywne mechanizmy generacji XML w Oracle9iR2
  - Funkcje SQL: SYS\_XMLGEN, SYS\_XMLAGG, XMLSequence
  - Pakiet DBMS\_XMLGEN (mapowanie danych relacyjnych w XML !)
  - Generacja zawartości XML poprzez API typu XMLType
  - XSQL Pages
  - XML SQL Utility (XSU)

# Generacja XML z bazy danych funkcjami SQLX

## - Przykładowe tabele źródłowe

```
SQL> describe emp
Nazwa      NULL?      Typ
-----
EMPNO      NOT NULL   NUMBER(4)
ENAME                               VARCHAR2(10)
JOB                               VARCHAR2(9)
MGR                               NUMBER(4)
HIREDATE    DATE
SAL                               NUMBER(7,2)
COMM                               NUMBER(7,2)
DEPTNO      NUMBER(2)
```

```
SQL> describe dept
Nazwa      NULL?      Typ
-----
DEPTNO     NUMBER(2)
DNAME       VARCHAR2(14)
LOC         VARCHAR2(13)
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	80/12/17	800		20
7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30
7521	WARD	SALESMAN	7698	81/02/22	1250	500	30
7566	JONES	MANAGER	7839	81/04/02	2975		20
7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30
7698	BLAKE	MANAGER	7839	81/05/01	2850		30
7782	CLARK	MANAGER	7839	81/06/09	2450		10
7788	SCOTT	ANALYST	7566	82/12/09	3000		20
7839	KING	PRESIDENT		81/11/17	5000		10
7844	TURNER	SALESMAN	7698	81/09/08	1500	0	30
7876	ADAMS	CLERK	7788	83/01/12	1100		20
7900	JAMES	CLERK	7698	81/12/03	950		30
7902	FORD	ANALYST	7566	81/12/03	3000		20
7934	MILLER	CLERK	7782	82/01/23	1300		10

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

# Funkcja XMLElement (1)

- Tworzy element XML z danych nie będących XML

```
SELECT e.empno,  
       XMLElement ( NAME "emp", e.ename) AS "result"  
FROM emp e;
```

zapytanie

wynik

```
EMPNO result  
-----  
7369 <emp>SMITH</emp>  
7499 <emp>ALLEN</emp>  
7521 <emp>WARD</emp>  
7566 <emp>JONES</emp>  
7654 <emp>MARTIN</emp>  
7698 <emp>BLAKE</emp>  
7782 <emp>CLARK</emp>  
7788 <emp>SCOTT</emp>  
7839 <emp>KING</emp>  
7844 <emp>TURNER</emp>  
7876 <emp>ADAMS</emp>  
7900 <emp>JAMES</emp>  
7902 <emp>FORD</emp>  
7934 <emp>MILLER</emp>
```

# Funkcja XMLElement (2)

- Atrybuty tworzonych elementów definiowane funkcją XMLAttributes

```
SELECT XMLElement (NAME "emp",  
                  XMLAttributes (e.empno, e.ename)  
                  ) AS "result"  
  
FROM emp e  
  
WHERE e.empno < 7600;
```

zapytanie

wynik

```
result  
-----  
<emp EMPNO="7369" ENAME="SMITH"/>  
<emp EMPNO="7499" ENAME="ALLEN"/>  
<emp EMPNO="7521" ENAME="WARD"/>  
<emp EMPNO="7566" ENAME="JONES"/>
```

XMLAttributes zawsze  
jako drugi argument  
XMLElement

# Funkcja XMLElement (3)

- Wartość NULL argumentu XMLAttributes skutkuje brakiem danego atrybutu w tworzonym elemencie

```
SELECT XMLElement (NAME "emp",  
                  XMLAttributes (e.empno, e.ename, e.comm)  
                  ) AS "result"  
  
FROM emp e  
  
WHERE e.empno < 7600;
```

zapytanie

wynik

```
result  
-----  
<emp EMPNO="7369" ENAME="SMITH"/>  
<emp EMPNO="7499" ENAME="ALLEN" COMM="300"/>  
<emp EMPNO="7521" ENAME="WARD" COMM="500"/>  
<emp EMPNO="7566" ENAME="JONES"/>
```

# Funkcja XMLElement (4)

- Zmienić domyślną nazwę atrybutu można podając alias dla kolumny tabeli
- Elementy-dzieci tworzone są przez zagnieżdżone wywołania XMLElement

```
SELECT XMLElement (NAME "emp",  
                 XMLAttributes (e.empno as "id"),  
                               XMLElement (NAME "name", e.ename),  
                               XMLElement (NAME "salary", e.sal)  
                 ) AS "result"  
  
FROM emp e  
  
WHERE e.empno < 7600;
```

zapytanie

wynik

```
result  
-----  
<emp id="7369">  
  <name>SMITH</name>  
  <salary>800</salary>  
</emp>  
<emp id="7499">  
  <name>ALLEN</name>  
  <salary>1600</salary>  
</emp>  
<emp id="7521">  
  <name>WARD</name>  
  <salary>1250</salary>  
</emp>  
<emp id="7566">  
  <name>JONES</name>  
  <salary>2975</salary>  
</emp>
```

# Funkcja XMLElement (5)

- Wartość NULL argumentu XMLElement powoduje utworzenie pustego elementu

```
SELECT XMLElement (NAME "emp",
                  XMLAttributes (e.empno as "id"),
                  XMLElement (NAME "name", e.ename),
                  XMLElement (NAME "salary", e.sal),
                  XMLElement (NAME "commission", e.comm)
                  ) AS "result"
FROM emp e
WHERE e.empno < 7600;
```

zapytanie

```
result
-----
<emp id="7369">
  <name>SMITH</name>
  <salary>800</salary>
  <commission/>
</emp>
<emp id="7499">
  <name>ALLEN</name>
  <salary>1600</salary>
  <commission>300</commission>
</emp>
<emp id="7521">
  <name>WARD</name>
  <salary>1250</salary>
  <commission>500</commission>
</emp>
<emp id="7566">
  <name>JONES</name>
  <salary>2975</salary>
  <commission/>
</emp>
```

wynik

# Funkcja XMLElement (6)

- XMLElement może generować elementy o mieszanej zawartości (elementy-dzieci i tekst)

```
SELECT XMLElement (NAME "emp",  
                  XMLElement (NAME "name", e.ename ),  
                  ' earns ',  
                  XMLElement (NAME "salary", e.sal )  
                ) AS "result"  
FROM emp e  
WHERE e.empno < 7600 ;
```

zapytanie

wynik

```
result  
-----  
<emp>  
  <name>SMITH</name>  
  earns   <salary>800</salary>  
</emp>  
<emp>  
  <name>ALLEN</name>  
  earns   <salary>1600</salary>  
</emp>  
<emp>  
  <name>WARD</name>  
  earns   <salary>1250</salary>  
</emp>  
<emp>  
  <name>JONES</name>  
  earns   <salary>2975</salary>  
</emp>
```

# Funkcja XMLForest (1)

- Tworzy las elementów XML z danych nie będących XML

zapytanie

```
SELECT e.empno, XMLForest (e.ename AS "name",  
                           e.sal AS "salary"  
    ) AS "result"  
FROM emp e  
WHERE e.empno < 7600 ;
```

wynik

```
EMPNO result  
-----  
7369 <name>SMITH</name>  
      <salary>800</salary>  
7499 <name>ALLEN</name>  
      <salary>1600</salary>  
7521 <name>WARD</name>  
      <salary>1250</salary>  
7566 <name>JONES</name>  
      <salary>2975</salary>
```

# Funkcja XMLForest (2)

- Może być wykorzystana do utworzenia listy elementów-dzieci w elemencie tworzonym funkcją XMLElement

zapytanie

```
SELECT XMLElement (NAME "emp",  
                  XMLAttributes (e.empno as "id"),  
                  XMLForest (e.ename AS "name",  
                              e.sal AS "salary"  
                              )  
        ) AS "result"  
FROM emp e  
WHERE e.empno < 7600 ;
```

wynik

```
result  
-----  
<emp id="7369">  
  <name>SMITH</name>  
  <salary>800</salary>  
</emp>  
<emp id="7499">  
  <name>ALLEN</name>  
  <salary>1600</salary>  
</emp>  
<emp id="7521">  
  <name>WARD</name>  
  <salary>1250</salary>  
</emp>  
<emp id="7566">  
  <name>JONES</name>  
  <salary>2975</salary>  
</emp>
```

# Funkcja XMLForest (3)

- Wartość NULL powoduje pominięcie danego elementu

zapytanie

```
SELECT XMLElement (NAME "emp",  
                XMLAttributes (e.empno as "id"),  
                XMLForest (e.ename AS "name",  
                           e.sal AS "salary",  
                           e.comm AS "commission"  
                )  
      ) AS "result"  
FROM emp e  
WHERE e.empno < 7600 ;
```

wynik

```
result  
-----  
<emp id="7369">  
  <name>SMITH</name>  
  <salary>800</salary>  
</emp>  
  
<emp id="7499">  
  <name>ALLEN</name>  
  <salary>1600</salary>  
  <commission>300</commission>  
</emp>  
  
<emp id="7521">  
  <name>WARD</name>  
  <salary>1250</salary>  
  <commission>500</commission>  
</emp>  
  
<emp id="7566">  
  <name>JONES</name>  
  <salary>2975</salary>  
</emp>
```

# Funkcja XMLGen

- Tworzy element z wykorzystaniem szablonu
- Funkcja niedostępna w Oracle9iR2 !

zapytanie

```
SELECT XMLGen ( '<emp id="{ $EMPNO }">
                <name>{ $NAME }</name>
                <salary>{ $SAL }</salary>
            </emp>',
            e.empno,
            e.ename AS name,
            e.sal
        ) AS "result"
FROM emp e
WHERE e.empno < 7600 ;
```

Kolejne wartości wstawiane w miejsce zmiennych szablonu

wynik

```
result
-----
<emp id="7369">
  <name>SMITH</name>
  <salary>800</salary>
</emp>
<emp id="7499">
  <name>ALLEN</name>
  <salary>1600</salary>
</emp>
<emp id="7521">
  <name>WARD</name>
  <salary>1250</salary>
</emp>
<emp id="7566">
  <name>JONES</name>
  <salary>2975</salary>
</emp>
```

# Funkcja XMLConcat

- Tworzy las elementów przez konkatenację elementów XML wywiedzionych z jednego wiersza relacji

zapytanie

```
SELECT e.empno, XMLConcat (XMLElement ( NAME "name", e.ename),
                           XMLElement ( NAME "salary", e.sal)
) AS "result"
FROM emp e
WHERE e.empno < 7600 ;
```

wynik

```
EMPNO result
-----
7369 <name>SMITH</name>
     <salary>800</salary>
7499 <name>ALLEN</name>
     <salary>1600</salary>
7521 <name>WARD</name>
     <salary>1250</salary>
7566 <name>JONES</name>
     <salary>2975</salary>
```

# Funkcja XMLAgg (1)

- Tworzy las elementów z kolekcji elementów XML wywiedzionych z różnych wierszy relacji

wynik

```
SELECT XMLElement ( NAME "department",
                  XMLAttributes( e.deptno AS "number" ),
                  XMLAgg( XMLElement ( NAME "emp", e.ename )
                          )
                ) AS "result"
FROM emp e
GROUP BY e.deptno ;
```

zapytanie

```
result
-----
<department number="10">
  <emp>CLARK</emp>
  <emp>KING</emp>
  <emp>MILLER</emp>
</department>
<department number="20">
  <emp>SMITH</emp>
  <emp>FORD</emp>
  <emp>ADAMS</emp>
  <emp>SCOTT</emp>
  <emp>JONES</emp>
</department>
<department number="30">
  <emp>ALLEN</emp>
  <emp>BLAKE</emp>
  <emp>MARTIN</emp>
  <emp>TURNER</emp>
  <emp>JAMES</emp>
  <emp>WARD</emp>
</department>
```

# Funkcja XMLAgg (2)

- Przed utworzeniem lasu elementów dane źródłowe można posortować (nie działa w Oracle9iR2 !)

wynik

```
SELECT XMLElement ( NAME "department",
                  XMLAttributes( e.deptno AS "number" ),
                  XMLAgg( XMLElement ( NAME "emp", e.ename )
                        ORDER BY e.ename
                  )
) AS "result"
FROM emp e
GROUP BY e.deptno ;
```

zapytanie

```
result
-----
<department number="10">
  <emp>CLARK</emp>
  <emp>KING</emp>
  <emp>MILLER</emp>
</department>
<department number="20">
  <emp>ADAMS</emp>
  <emp>FORD</emp>
  <emp>JONES</emp>
  <emp>SCOTT</emp>
  <emp>SMITH</emp>
</department>
<department number="30">
  <emp>ALLEN</emp>
  <emp>BLAKE</emp>
  <emp>JAMES</emp>
  <emp>MARTIN</emp>
  <emp>TURNER</emp>
  <emp>WARD</emp>
</department>
```

# Funkcja XMLAgg (3)

- Hierarchiczne struktury XML można budować pobierając dane z kilku relacji

```
SELECT XMLElement ( NAME "department",
                  XMLAttributes( dname AS "name" ),
                  XMLAgg( XMLElement ( NAME "emp", ename )
                        )
                ) AS "result"
FROM emp NATURAL JOIN dept
GROUP BY dname ;
```

zapytanie

```
result
-----
<department name="ACCOUNTING">
  <emp>CLARK</emp>
  <emp>KING</emp>
  <emp>MILLER</emp>
</department>
<department name="RESEARCH">
  <emp>SMITH</emp>
  <emp>FORD</emp>
  <emp>ADAMS</emp>
  <emp>SCOTT</emp>
  <emp>JONES</emp>
</department>
<department name="SALES">
  <emp>ALLEN</emp>
  <emp>BLAKE</emp>
  <emp>MARTIN</emp>
  <emp>TURNER</emp>
  <emp>JAMES</emp>
  <emp>WARD</emp>
</department>
```

wynik

# Funkcja XMLColAttVal

- Nie ma jej w obecnej postaci standardu SQL/XML (jest w Oracle9iR2)
- Generuje las elementów <column> - podobna do XMLForest
- Nazwy kolumn z SQL stają się wartościami atrybutu *name* w <column>
- Cel: uniknięcie konwersji nazw kolumn w SQL do nazw elementów XML

```
SELECT XMLElement (NAME "emp",  
                XMLAttributes (e.empno as "id"),  
                XMLColAttVal (e.ename AS "name",  
                             e.sal AS "salary"  
                )  
        ) AS "result"  
FROM emp e  
WHERE e.empno < 7600 ;
```

zapytanie

```
result  
-----  
<emp id="7369">  
  <column name="name">SMITH</column>  
  <column name="salary">800</column>  
</emp>  
  
<emp id="7499">  
  <column name="name">ALLEN</column>  
  <column name="salary">1600</column>  
</emp>  
  
<emp id="7521">  
  <column name="name">WARD</column>  
  <column name="salary">1250</column>  
</emp>  
  
<emp id="7566">  
  <column name="name">JONES</column>  
  <column name="salary">2975</column>  
</emp>
```

wynik

# Charakterystyka języka XQuery

- XQuery Version 1.0 jest rozszerzeniem XPath Version 2.0
  - Każde wyrażenie które jest poprawne w obu tych standardach, zwróci w obu ten sam wynik
- XQuery operuje na modelu danych wspólnym dla XQuery 1.0 i XPath 2.0
  - Model reprezentuje informacje w formie węzłów i wartości atomowych
- XQuery jest językiem funkcyjnym
  - Bazuje na wyrażeniach (wyrażenia można zagnieżdżać)
  - Wartością wyrażenia jest zawsze sekwencja (uporządkowana kolekcja węzłów i wartości atomowych; w szczególności sekwencja pusta)
- Podobnie jak w XML, w XQuery wielkość liter ma znaczenie
  - Wszystkie słowa kluczowe pisane małymi literami (!)
- XQuery jest językiem "silnie typowanym" (typy XML Schema)
- Składnia XQuery nie jest składnią XML
  - Istnieje reprezentacja XQuery w XML - XQueryX

# Zapytania w XQuery

- Zapytanie w XQuery ma postać wyrażenia:
  - Odczytującego sekwencję węzłów lub wartości atomowych
  - Zwracającego sekwencję węzłów lub wartości atomowych
- Podstawowe formy wyrażeń w XQuery
  - Wyrażenia pierwotne (literały, zmienne, wywołania funkcji)
  - Wyrażenia ścieżkowe (selekcja węzłów)
  - Konstruktory elementów
  - Wyrażenia FLWOR ("flower") (FOR-LET-WHERE-ORDER BY-RETURN)
  - Wyrażenia listowe (manipulacja na listach)
  - Wyrażenia warunkowe
  - Kwantyfikatory ("istnieje" i "dla każdego")
  - Wyrażenia testujące/modyfikujące typy danych
- Funkcje odczytujące dane:
  - `document()` - zwraca węzeł dokumentu np. `document("a.xml")/cennik`
  - `collection()` - zwraca węzły z kolekcji np. `collection("http://a.org")//kod`
  - `input()` - zwraca sekwencję wejściową np. `input()//symbol`

# Wyrażenia ścieżkowe

- Najprostsze zapytania w XQuery to wyrażenia XPath

```
document ("produkty.xml") //produkt [cena>500]
```

zapytanie

```
<?xml version="1.0" encoding="windows-1250"?>
<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>
<produkt kod="56486294306">
  <nazwa>Radioodtworacz CAR 2002</nazwa>
  <symbol>3209766</symbol>
  <cena>625</cena>
</produkt>
<produkt kod="56486294308">
  <nazwa>Radioodtworacz CAR 2003</nazwa>
  <symbol>3209767</symbol>
  <cena>675</cena>
</produkt>
...
</cennik>
```

produkty.xml

wynik

```
<produkt kod="56486294304">
  <nazwa>Radioodtworacz CAR 2001</nazwa>
  <symbol>3209765</symbol>
  <cena>525</cena>
</produkt>
<produkt kod="56486294306">
  <nazwa>Radioodtworacz CAR 2002</nazwa>
  <symbol>3209766</symbol>
  <cena>625</cena>
</produkt>
<produkt kod="56486294308">
  <nazwa>Radioodtworacz CAR 2003</nazwa>
  <symbol>3209767</symbol>
  <cena>675</cena>
</produkt>
```

# FOR-LET-WHERE-RETURN (1)

- Odpowiednik SELECT-FROM-WHERE z SQL

zapytanie

```
for $d in document("produkty.xml")//produkt[cena > 500]
return $d
```

wynik

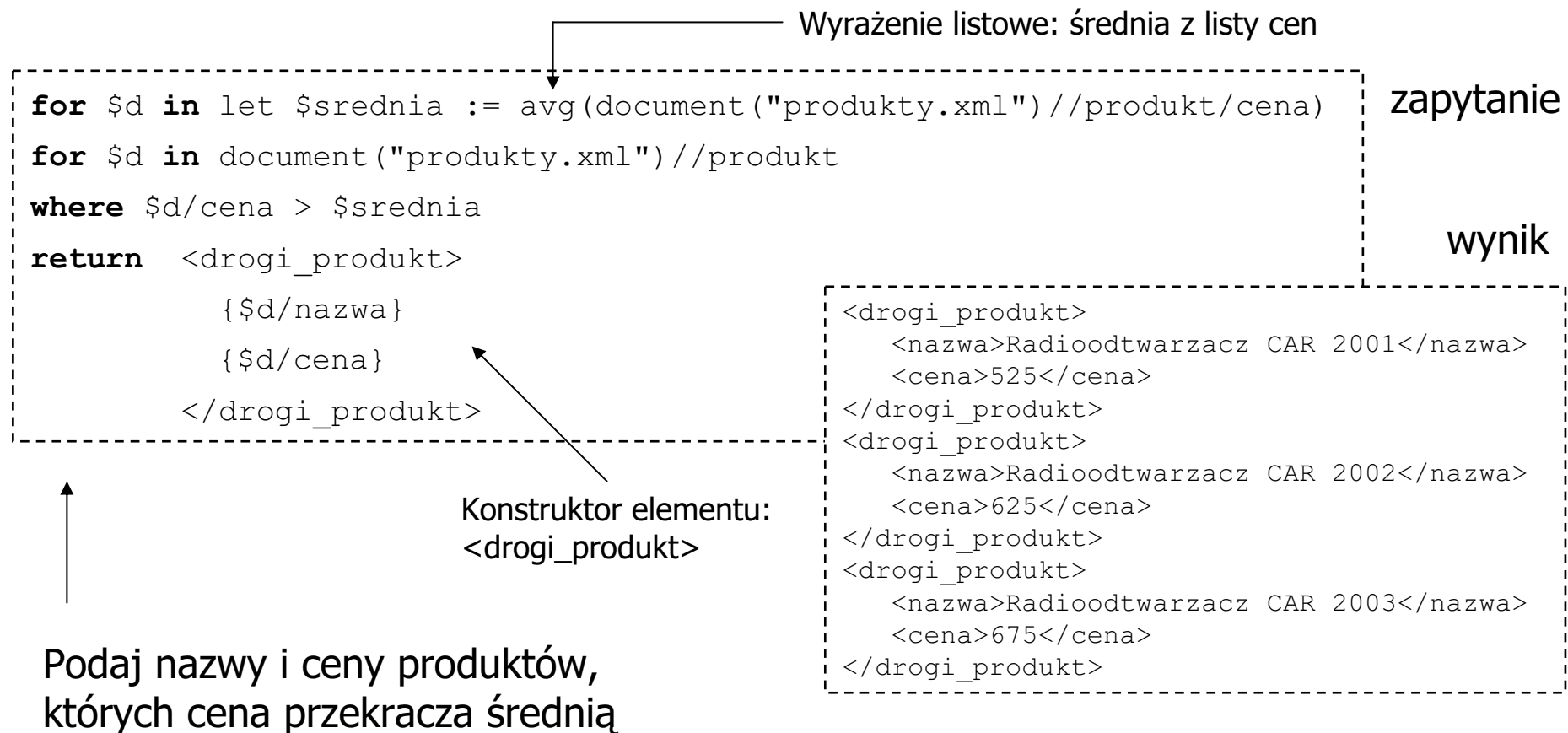
```
<produkt kod="56486294304">
  <nazwa>Radioodtworacz CAR 2001</nazwa>
  <symbol>3209765</symbol>
  <cena>525</cena>
</produkt>
<produkt kod="56486294306">
  <nazwa>Radioodtworacz CAR 2002</nazwa>
  <symbol>3209766</symbol>
  <cena>625</cena>
</produkt>
<produkt kod="56486294308">
  <nazwa>Radioodtworacz CAR 2003</nazwa>
  <symbol>3209767</symbol>
  <cena>675</cena>
</produkt>
```

W tym wypadku  
wystarczyłoby  
wyrażenie ścieżkowe



# FOR-LET-WHERE-RETURN (2)

- Klauzule FOR i LET może być wiele, mogą się dowolnie przeplatać, musi wystąpić co najmniej jedna z nich



# FOR-LET-WHERE-RETURN (3)

- Wyrażenia FLWR można zagnieżdżać

```
let $srednia := avg(document("produkty.xml")//produkt/cena)
return <drogie_produkty próg={$srednia}>
  {
    for $d in document("produkty.xml")//produkt
    where $d/cena > $srednia
    return $d
  }
</drogie_produkty>
```

zapytanie

wynik

```
<drogie_produkty próg="402.5">
  <produkt kod="56486294304">
    <nazwa>Radioodtworacz CAR 2001</nazwa>
    <symbol>3209765</symbol>
    <cena>525</cena>
  </produkt>
  <produkt kod="56486294306">
    <nazwa>Radioodtworacz CAR 2002</nazwa>
    <symbol>3209766</symbol>
    <cena>625</cena>
  </produkt>
  <produkt kod="56486294308">
    <nazwa>Radioodtworacz CAR 2003</nazwa>
    <symbol>3209767</symbol>
    <cena>675</cena>
  </produkt>
</drogie_produkty>
```

↑  
Wygeneruj katalog produktów,  
których cena przekracza średnią  
(podaj średnią cenę)

# FOR-LET-WHERE-RETURN (4)

- Konstrukcję FOR można wykorzystać do operacji połączenia (join)

zapytanie

```
for $d in document("produkty.xml")//produkt,  
    $e in document("produkty.xml")//produkt  
where $d/cena = $e/cena and $d/@kod < $e/@kod  
return <para_produktyw>  
    <nazwa_1>{$d/nazwa/text()}</nazwa_1>  
    <nazwa_2>{$e/nazwa/text()}</nazwa_2>  
    {$d/cena}  
</para_produktyw>
```

wynik

Podaj pary produktów o tej samej cenie (bez duplikatów)

```
<para_produktyw>  
  <nazwa_1>Zestaw głośnomówiący LOUD 1</nazwa_1>  
  <nazwa_2>Ładowarka samochodowa ATUT</nazwa_2>  
  <cena>320</cena>  
</para_produktyw>
```

# FLWOR: Klauzula ORDER-BY

- Rozszerza wyrażenia FLWR o możliwość sortowania wyników
- Zastępuje wcześniej proponowaną klauzulę SORTBY w FOR (!)

```
for $d in document("produkty.xml")//produkt[cena > 500]
order by $d/cena descending
return $d
```

zapytanie



Uporządkuj produkty wg cen,  
od najdroższych do najtańszych

```
<produkt kod="56486294308">
  <nazwa>Radioodtwarzacz CAR 2003</nazwa>
  <symbol>3209767</symbol>
  <cena>675</cena>
</produkt>
<produkt kod="56486294306">
  <nazwa>Radioodtwarzacz CAR 2002</nazwa>
  <symbol>3209766</symbol>
  <cena>625</cena>
</produkt>
<produkt kod="56486294304">
  <nazwa>Radioodtwarzacz CAR 2001</nazwa>
  <symbol>3209765</symbol>
  <cena>525</cena>
</produkt>
```

wynik

# Wyrażenia warunkowe

zapytanie

```
for $d in document("produkty.xml")//produkt
return <produkt klasa={ if ($d/cena > 400) then "drogi" else "tani" }>
      {$d/nazwa, $d/cena}
</produkt>
```



wynik

```
<produkt klasa="tani">
  <nazwa>Antena dachowa</nazwa>
  <cena>85</cena>
</produkt>
<produkt klasa="drogi">
  <nazwa>Radioodtworacz CAR 2001</nazwa>
  <cena>525</cena>
</produkt>
<produkt klasa="drogi">
  <nazwa>Radioodtworacz CAR 2002</nazwa>
  <cena>625</cena>
</produkt>
...
```

Przypisz produktom atrybut  
*klasa*: drogi/tani

- Uwagi:
  - Warunek musi być podany w nawiasach
  - Klauzula ELSE jest obowiązkowa
  - Wyrażenia warunkowe można zagnieżdżać

# Kwantyfikikator: SOME-IN-SATISFIES

```
for $d in document("produkty.xml")//cennik
where some $p in $d/produkt satisfies ($p/cena < 100)
return $d
```

zapytanie

wynik



Zwróć cenniki zawierające  
co najmniej jeden produkt  
w cenie poniżej 100

```
<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>
<produkt kod="56486294306">
  <nazwa>Radioodtworacz CAR 2002</nazwa>
  <symbol>3209766</symbol>
  <cena>625</cena>
</produkt>
<produkt kod="56486294308">
  <nazwa>Radioodtworacz CAR 2003</nazwa>
  <symbol>3209767</symbol>
  <cena>675</cena>
</produkt>
...
</cennik>
```

# Kwantyfikikator: EVERY-IN-SATISFIES

```
for $d in document("produkty.xml")//cennik
where every $p in $d/produkt satisfies ($p/cena < 100)
return $d
```

zapytanie

wynik



Zwróć cenniki zawierające  
tylko produkty  
w cenie poniżej 100

# XQueryX - Reprezentacja XQuery w XML

- Umożliwia zapytania na zapytaniach, generację zapytań, zagnieżdżanie zapytań w XML, ...

XQuery

```
FOR $b IN document("bib.xml")//book
WHERE $b/publisher = "Morgan Kaufmann" AND $b/year = "1998"
RETURN $b/title
```

XQueryX

```
<q:query xmlns:q="http://www.w3.org/2001/06/xqueryx">
  <q:flwr>
    <q:forAssignment variable="$b">
      <q:step axis="SLASHSLASH">
        <q:function name="document">
          <q:constant datatype="CHARSTRING">bib.xml</q:constant>
        </q:function>
        <q:identifier>book</q:identifier>
      </q:step>
    </q:forAssignment>
    <q:where>
      ...
    </q:where>
    <q:return>
      ...
    </q:return>
  </q:flwr>
</q:query>
```

# Podsumowanie

- W najbliższym czasie należy spodziewać się pełnych, zatwierdzonych specyfikacji dwóch języków zapytań dla XML
  - SQL/XML - standard ANSI/ISO
    - Dla mieszanych danych strukturalnych i semi-strukturalnych w bazach danych
  - XQuery - rekomendacja W3C
    - Dla "czystych" danych XML w bazach danych, plikach, etc.
- Oracle bierze udział w pracach nad obydwojoma standardami i już oferuje ich implementacje
  - SQL/XML poprzez funkcje SQLX w ramach XML DB
  - XQuery w postaci prototypu w języku Java