XML-QL – język zapytań

(prezentacja by Marcin Stopyra)

Właściwości języka XML-QL

- kompletny zestaw narzędzi do obsługi danych w postaci relacyjnej
- wystarczająco prosty aby można było zastosować znane techniki bazodanowe do optymalizacji zapytań
- umożliwia dostęp do danych w dokumentach XML oraz tworzenie nowych dokumentów w formacie XML
- wspiera zarówno uporządkowane jak i nieuporządkowane dokumenty XML (obecna implementacja tylko nieuporządkowane)

Konstrukcja XML-QL

- składnia *where/construct*
- wzorce
- opisy ścieżek

Przykładowe DTD:

```
<!ELEMENT book (author+, title, publisher)>
<!ATTLIST book year CDATA>
<!ELEMENT publisher (name, address)>
<!ELEMENT author (firstname?, lastname)>
```

Zapytanie :

```
WHERE <bib><book>
        <publisher><name>"Addison-Wesley"</name></publisher>
        <title>  $t </title>
        <author> $a </author>
      </book></bib> IN "bib.xml"
CONSTRUCT $a
```

```
WHERE <bib><book>
        <publisher> <name>"Addison-Wesley" </> </>
        <title>  $t </>
        <author> $a </>
      </></> IN "bib.xml"
CONSTRUCT <result>
            <author> $a </>
            <title>  $t </>
          </>
```

Przykładowy dokument XML:

```
<bib>
  <book year="1995">
    <!-- A good introductory text -->
    <title>An Introduction to Database Systems </title>
    <author><lastname>Date </lastname></author>
    <publisher><name>Addison-Wesley </name >  </publisher>
  </book>
  <book year="1998">
    <title>Foundation for Object/Relational Databases</title>
    <author><lastname>Date </lastname></author>
    <author><lastname>Darwen </lastname></author>
    <publisher><name>Addison-Wesley </name >  </publisher>
  </book>
</bib>
```

Odpowiedź:

```
<XML>
 <result>
   <author><lastname>Date </lastname></author>
   <title>An Introduction to Database Systems </title>
 </result>
 <result>
   <author><lastname>Date </lastname></author>
   <title>Foundation for Object/Relational Databases</title>
 </result>
 <result>
   <author><lastname>Darwen </lastname></author>
   <title>Foundation for Object/Relational Databases</title>
 </result>
 </XML>
```

Dodanie własnego korzenia:

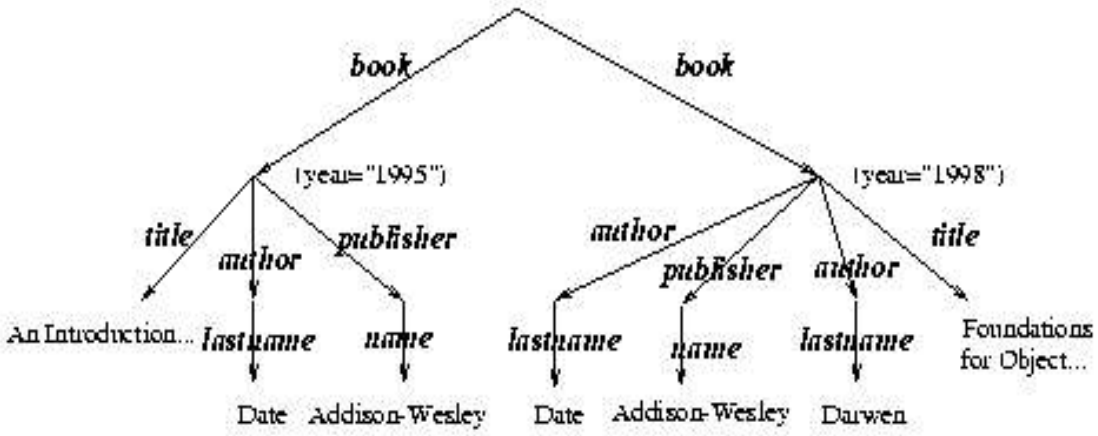
```
CONSTRUCT <answer> {
  WHERE <bib><book>
        <publisher> <name>"Addison-Wesley" </> </>
        <title>  $t </>
        <author> $a </>
       </></> IN "bib.xml"
  CONSTRUCT <result>
            <author> $a </>
            <title>  $t </>
          </>
} </aswer>
```

Model danych

- obecnie tylko nieuporządkowane dane
- ignoruje komentarze

Graf XML

- każdy węzeł reprezentowany jest przez unikatowy string zwany OID
- krawędzie etykietowane są przez znaczniki (tagi)
- węzły etykietowane są przez pary *atrybut-wartość*
- liście grafu etykietowane są przez pojedynczą wartość string
- graf ma wyróżniony jeden element zwany korzeniem



Pomiędzy dowolnymi dwoma wierzchołkami może istnieć co najwyżej jedna krawędź z daną etykietą.

Żaden węzeł nie może posiadać dwóch liści o tej samej etykiecie i tym samym stringu.

<author> <lastname> Dates </lastname> <lastname> Dates </lastname> </author>

będzie zamienione na:

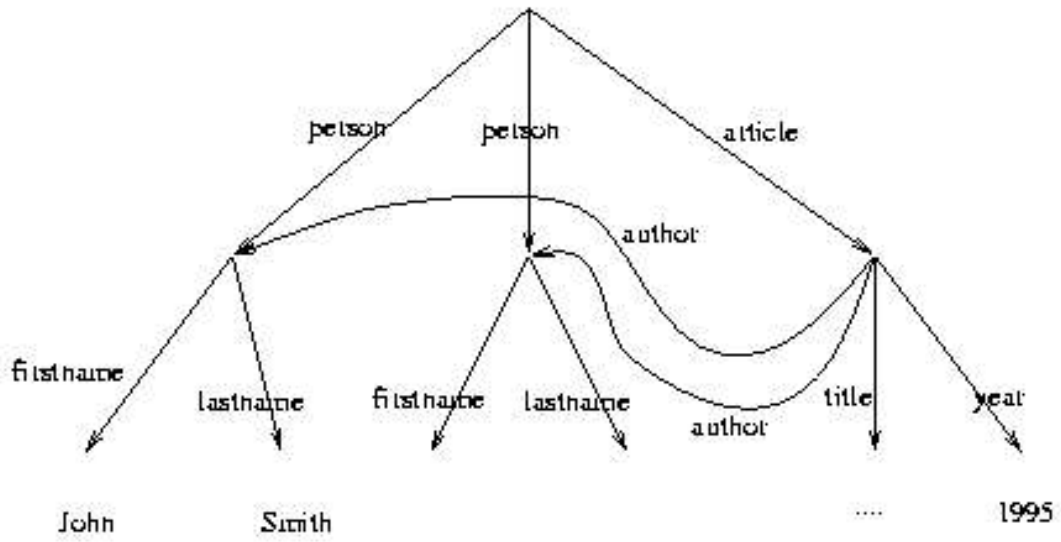
<author> <lastname> Dates </lastname> </author>

ID i IDREF

- każdy węzeł ma unikalny OID, który jest wartością atrybutu ID lub generowany automatycznie jeśli takiego nie ma
- elementy XML mogą odwoływać się do siebie za pomocą IDREF lub IDREFS
- atrybuty IDREF reprezentowane są za pomocą krawędzi w grafie z etykietą będącą nazwą atrybutu IDREF (nie wartością)

```
<!ATTLIST person  ID ID #REQUIRED>
<!ATTLIST article author IDREFS #IMPLIED>

<person ID="o123">
   <firstname>John</firstname>
   <lastname>Smith<lastname>
</person>
<person ID="o234">
   . . .
</person>
<article author="o123 o234">
   <title> ... </title>
   <year> 1995 </year>
</article>
```



Wygodne poruszanie się po grafie:

WHERE <article><author><lastname> $n</></></> IN "abc.xml"

WHERE <article author=$i>
<title> </> ELEMENT_AS $t
</>,
<person ID=$i>
<lastname> </> ELEMENT_AS $l
</>
CONSTRUCT <result> $t $l</>

Uwaga 1:
Zapis *<title></> ELEMENT_AS $t* wiąże zmienna t z dowolnym elementem <title> również z *<title/>*

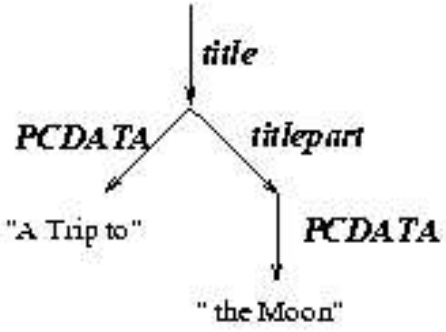
Uwaga 2:
Tylko liście mogą posiadać wartość (string) i tylko jedną

Poniższy fragment

<title>A Trip to <titlepart> the Moon </titlepart></title>

zostanie reprezentowany przez

<title><PCDATA> A Trip to </PCDATA>
<titlepart><PCDATA> the Moon</PCDATA></titlepart>
</title>



Złączenie elementów według wartości

```
WHERE <bib.article>
        <author>
          <firstname.PCDATA> $f </>  // firstname $f
          <lastname.PCDATA>  $l </>  // lastname $l
        </>
      </> CONTENT_AS $a IN "bib.xml",

      <book year=$y>
        <author>
          <firstname.PCDATA>$f</> // join on same firstname $f
          <lastname.PCDATA>$l</> // join on same lastname $l
        </>
      </> IN "bib.xml",
      y > 1995
CONSTRUCT <article> $a </>
```

```
WHERE <bib.article>
        <author>
          <firstname>$f1</>  // firstname $f
          <lastname> $l1</>  // lastname $l
        </>
      </> CONTENT_AS $a IN "bib.xml",

      <book year=$y>
        <author>
          <firstname>$f2n</> // join on same firstname $f
          <lastname>$l2</>  // join on same lastname $l
        </>
      </> IN "bib.xml",
      y > 1995,
      text($f1) = text($f2),
      text($l1) = text($l2)
CONSTRUCT <article> $a </>
```

Elementy opcjonalne

- podstawowa różnica między XML a danymi modelu relacyjnego

Zakładamy, że element 'price' jest opcjonalny.

Przykład złego zapytania:

```
WHERE <book><title> $t </title>
                  <price> $p </price>
            </book> IN "bib.xml"
CONSTRUCT <result> <booktitle> $t </booktitle>
                              <bookprice> $p </bookprice>
                  </result>
```

Tak można zapisać poprawnie:

```
WHERE <book> $b </book> IN "bib.xml",
            <title> $t </title> IN $b
CONSTRUCT <result> <booktitle> $t </booktitle>
                              {WHERE <price> $p </price> IN $b
                              CONSTRUCT <bookprice> $p </bookprice>}
                  </result>
```

Grupowanie i zagnieżdżone zapytania

```
WHERE <*.book > $p </> IN "bib.xml",
      <*><title > $t </>
          <publisher> <name> "Addison-Wesley" </> </> </> IN $p
CONSTRUCT <result>
                    <title> $t </>
                   { WHERE  <author> $a </> IN $p
                    CONSTRUCT <author> $a </> }
                  </>
```

Zmienne i znaczniki

```
WHERE <$p>
          <title> $t </title>
          <year> 1995 </>
          <$e> Smith </>
          </> IN "bib.xml",
          $e IN {author, editor}
CONSTRUCT <$p>
                  <title> $t </title>
                  <$e> Smith </>
                  </>
```

Wyrażenia regularne

- dozwolone wszędzie tam gdzie występują elementy (znaczniki)
- dopuszczalne symbole: '|', '.', '*', '+', '?'

Przykładowe DTD:

<!ELEMENT part (name brand part)>*
<!ELEMENT name (#PCDATA)>
<!ELEMENT brand (#PCDATA)>

Przykładowe zapytanie:

WHERE <part> <name> $r </> <brand> Ford </> </> IN "bib.xml"*
CONSTRUCT <result> $r </>

Symol '$' zastępuje dowolny znacznik.

Zapis '$*' można skrócić do '*'

Np. *<*.brand> Ford </>*

Inny przykład:

WHERE <part+.(subpart|component.piece)>$r</> IN "parts.xml"
CONSTRUCT <result> $r</>

Funkcje Skolema

- generowanie nowego identyfikatora
- grupowanie przy pomocy funkcji Skolema

```
<!ELEMENT person (lastname, firstname, address?, publicationtitle*)>
```

```
WHERE <$> <author> <firstname> $fn </>
                          <lastname> $ln </>
                   </>
              <title> $t </>
         </> IN "bib.xml",
CONSTRUCT <person ID=PersonID($fn, $ln)>
                    <firstname> $fn </>
                    <lastname> $ln </>
                   <publicationtitle> $t </>
                </>
```

Zmienne indeksowane

- odpowiadają porządkowi w grafie (kolejność krawędzi wychodzących z danego węzla)

```
WHERE <person> $p </> IN "people.xml",
      <firstname [$i]> $x </> IN $p,
      <lastname [$j]> $y </> IN $p,
      $j < $i
CONSTRUCT <person> $p </>
```

Opcjonalna klauzula 'order-by'

- określa porządek elementów

```
WHERE <pub> &p </> in "www.a.b.c/bib.xml",
      <title> $t </> in $p,
      <year> $y </> in $p
      <month> $z </> in $p
ORDER-BY $y,$z
CONSTRUCT $t
```

Definicje funkcji a DTD

```
FUNCTION findDeclaredIncomes($Taxpayers, $Employees) {
  WHERE <taxpayer> <ssn> $s </>
                  <income> $x </> </> IN $Taxpayers,
              <employee> <ssn> $s </>
                  <name> $n </> </> IN $Employees
  CONSTRUCT <result> <name> $n </>
                        <income> $x </> </>
}
```

Wywołanie:

findDelcaredIncomes("taxpayers.xml", "employees.xml")

Użycie DTD do funkcji:

```
FUNCTION findDeclaredIncomes($Taxpayers:"tp.dtd",
                        $Employees:"employees.dtd") : "myresult.dtd" {
  WHERE ...
  CONSTRUCT ...
}
```

Na razie wsparcie tylko do sprawdzania w czasie wykonania, czy wejsciowe dane XML pasują do DTD.

Złączenie zewnętrzne

```
{ WHERE <person>
                <name> </> ELEMENT_AS $n
                <ssn> $ssn </>
              </> IN "www.a.b.c/data.xml"
  CONSTRUCT <result ID=SSNID($ssn)> <ssn> $ssn </> $n </>}
{ WHERE <taxpayer>
                <ssn> $ssn </>
                <income> </> ELEMENT_AS $i
             </> IN "www.irs.gov/taxpayers.xml"
  CONSTRUCT <result ID=SSNID($ssn)> ssn> $ssn </> $i </>}
```

Źródło

http://www.research.att.com/sw/tools/xmlql/