XQuery

Jirka Kosek

Visual FoxPro DevCon
21.–23. června 2005

Praha

Agenda

- úvod do XQuery
- základy XPath 2.0
- FLWOR výrazy
- typový systém
- implementace XQuery

Úvod

Proč potřebujeme XQuery?

- XML se nepoužívá jen pro přenos dat, ale i jako úložiště strukturovaných dat
- data je potřeba prohledávat, vybírat z nich dílčí údaje, počítat statistiky, …
- datový model XML je strom
- SQL nejde využít, protože je navržené pro relační model dat
- XPath je příliš jednoduchý a mnoho věcí neumí
- je potřeba nový dotazovací jazyk pro XML = XQuery

Kořeny XQuery

- **funkcionalita inspirovaná několika jazyky:**
  - **XPath – navigace po stromové struktuře dokumentu**
  - **SQL – agregace dat, filtrování dat**
  - **XSLT – možnost definování struktury výsledného dokumentu a vytváření nových elementů**
  - **Quilt, XML-QL, OQL – prototypy jazyků pro dotazování nad XML**
- **vyvíjeno od roku 1999 v rámci W3C**

Standardizace XQuery

- v současné době je ve fázi poměrně stabilního pracovního návrhu
- finální verze se očekává na začátku roku 2006
- XQuery je definováno několika dílčími normami:
  - XQuery 1.0 and XPath 2.0 Data Model[1]
  - XSLT 2.0 and XQuery 1.0 Serialization[2]
  - XQuery 1.0 and XPath 2.0 Formal Semantics[3]
  - XQuery 1.0: An XML Query Language[4]
  - XML Path Language (XPath) 2.0[5]
  - XQuery 1.0 and XPath 2.0 Functions and Operators[6]
- v současné době je XQuery jen dotazovací jazyk
- v budoucnu bude XQuery rozšířeno o další funkcionalitu – např. možnost aktualizace dat, fulltextové prohledávání

[1] http://www.w3.org/TR/xpath-datamodel/
[2] http://www.w3.org/TR/xslt-xquery-serialization/
[3] http://www.w3.org/TR/xquery-semantics/
[4] http://www.w3.org/TR/xquery/
[5] http://www.w3.org/TR/xpath20/
[6] http://www.w3.org/TR/xpath-functions/

XPath × XQuery × XSLT

- XPath 1.0
  - standard od roku 1999
  - využívá jej mnoho dalších jazyků – XSLT, XML Schema, XML Encryption, XML Signature, Schematron, …
- XSLT 1.0
  - jazyk pro transformaci dokumentů XML
  - jako dotazovací jazyk používá XPath 1.0
- XSLT 2.0
  - nová verze transformačního jazyka
  - vyžaduje silnější dotazovací jazyk než XPath 1.0
- XQuery 1.0
  - dotazovací jazyk, který je bohatší než XPath
- XPath 2.0
  - zabraňuje rozštěpení dotazovacích jazyků pro XML
  - jedná se o podstatně rozšířený XPath 1.0
  - s XPath 1.0 je skoro zpětně kompatibilní
  - využívá se jako dotazovací jazyk v XSLT 2.0 a je základem XQuery 1.0
- XQuery 1.0 je nadmnožina XPath 2.0
- každý XPath 2.0 je tedy XQuery 1.0 dotaz

Základní rysy jazyka

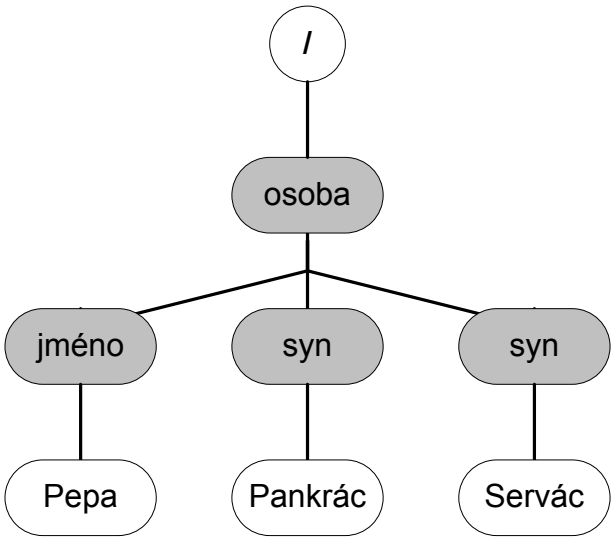
- funkcionální jazyk
  - výrazy lze libovolně kombinovat dohromady
  - výsledek jednoho výrazu může sloužit jako parametr dalšího výrazu
  - funkce nemají vedlejší efekt
- uzavřenost nad datovým modelem
  - výsledkem XQuery dotazu je vždy instance datového modelu
- úzké provázání s W3C XML Schema
  - datové typy jsou definovány na základě schématu
- silná typová kontrola
  - větší bezpečnost
  - někdy více psaní
- statické typování
  - mnoho kontrol správnosti dotazu lze provést ještě před spuštěním dotazu při jeho analýze

XPath 2.0

XPath 2.0

- **jednoduchý dotazovací jazyk nad dokumenty XML**
- **operuje nad stromovou strukturou dokumentu XML**

```
<osoba>
  <jméno>Pepa</jméno>
  <syn>Pankrác</syn>
  <syn>Servác</syn>
</osoba>
```



- **výsledkem dotazu je posloupnost složená z uzlů stromu dokumentu nebo hodnot**
- **velmi úsporná syntaxe zaměřená především na navigaci a výběr hodnot ze stromu dokumentu**

Zdroje dat

- **data lze číst přímo ze souboru pomocí funkce `doc()`**

  ```
  doc('katalog.xml')/katalog/polozka
  ```

- **v praxi bude XML uloženo v databázi podporující XML**
- **funkce `collection()` umožňuje definovat implementačně závislé zdroje dat**

  ```
  collection('obchod/katalogy')
  ```

  funkce může vracet několik dokumentů najednou
- **v některých případech je zdroj dat určen mimo XPath – například když se XPath používá uvnitř XSLT**

Cesty v dokumentu

- **nejčastější typ dotazu prochází strom dokumentu po jednotlivých úrovních a vybírá uzly výsledku**

```
(: Výběr všech názvů položek v katalogu :)

doc('katalog.xml')/katalog/polozka/nazev
```

- **syntaxe je analogická zápisu cesty k souboru na disku**
- **lze použít i známé zkratky `.` (aktuální uzel) a `..` (rodič)**
- **výsledné uzly jsou vždy vráceny v pořadí, v jakém se vyskytují v dokumentu**
- **ze seznamu uzlů výsledku jsou odstraněny duplicitní uzly**

Divoké znaky

- neznáme-li přesně hloubku elementu, můžeme použít `//`

```
(: Výběr všech elementů název :)

doc('katalog.xml')/katalog//nazev
```

- výběr elementu s jakýmkoliv názvem (`*`)

```
(: Výběr všech elementů uvnitř položky :)

doc('katalog.xml')/katalog/polozka/*
```

Predikáty

- **umožňují snadné filtrování uzlů**
- `(: Výběr první položky :)`

  ```
  doc('katalog.xml')/katalog/polozka[1]
  ```

- `(: Výběr poslední položky :)`

  ```
  doc('katalog.xml')/katalog/polozka[last()]
  ```

- `(: Výběr názvů položek dražších než 10000 :)`

  ```
  doc('katalog.xml')/katalog/polozka[cena > ►
  10000]/nazev
  ```

Funkce

- **lze používat v predikátech**

```
(: Výběr názvů položek, které obsahují text ►
"MiniDisc" :)

doc('katalog.xml')//polozka/nazev[contains(., ►
'MiniDisc')]
```

- **nebo samostatně pro výpočet hodnoty**

```
(: Spočítání počtu položek v katalogu :)

count(doc('katalog.xml')//polozka)

(: Zjištění průměrné ceny položky v ►
katalogu :)

avg(doc('katalog.xml')//polozka/cena)
```

Přehled funkcí

- práce s řetězci
- regulární výrazy
- základní matematické operace
- práce s datem, časem a intervaly
- agregační funkce (count, sum, avg, min, max)
- práce s posloupnostmi
- práce se stromem dokumentu XML a jeho uzly

Osy

- lomítko v XPathu výrazu odděluje jednotlivé části cesty a hledá další uzly mezi dětmi
- XPath umožňuje prohledávat i jiné uzly než jen děti
- ```
  (: Výběr položek, za kterými následuje ► položka s reproduktorem :)

  doc('katalog.xml')//polozka[following-sibling::polozka[1][kategorie ► = 'Reproduktory']]
  ```

- podporované osy:

ancestor	following
ancestor-or-self	following-sibling
attribute	parent
child	preceding
descendant	preceding-sibling
descendant-or-self	self
  ```

Posloupnosti

- **mohou vzniknout výběrem uzlů se stromu dokumentu nebo uměle**
- **vytvoření posloupnosti obsahujících 100 čísel od jedné do sta je velmi jednoduché:**

```
1 to 100
```

- **posloupnosti lze dále zpracovávat pomocí příkazu `for`**

```
(: Vygenerování poslupnosti čísel od 1 do ►
10 a jejich mocnin :)

for $i in (1 to 10) return ($i, $i * $i)
```

Další možnosti

- **podmíněné výrazy (`if … then … else …`)**
- **kvantifikátory**
  - **existenční kvantifikátor (∃ → `some $x in` *`poslupnost`* `satisfies` *`podmínka`***
  - **všeobecný kvantifikátor (∀ → `every $x in` *`posloupnost`* `satisfies` *`podmínka`***
- `(: Výpis prvočísel mezi 1 a 100 :)`

```
for $i in (2 to 100) return
  if (every $j in (2 to $i - 1) satisfies ►
$i mod $j ne 0)
    then $i
    else ()
```

FLWOR výrazy

FLWOR výrazy

- **XPath 2.0 je celkem silný dotazovací jazyk, ale**
  - **zápis složitějších podmínek je někdy nepřehledný**
  - **výsledek dotazu nejde seřadit**
  - **na výstupu nejde generovat nové elementy**
- **FLWOR [flaur] výrazy tyto nevýhody XPathu odstraňují**
- **XQuery = XPath 2.0 + FLWOR výrazy + výrazy konstruující nové elementy + uživatelsky definované funkce + několik dalších direktiv**

Struktura FLWOR výrazu

- **FOR – výběr posloupnosti uzlů k dalšímu zpracování**
- **LET – přiřazení proměnných pro každý prvek posloupnosti**
- **WHERE – filtrování uzlů v posloupnosti**
- **ORDER BY – seřazení vybraných a odfiltrovaných uzlů**
- **RETURN – specifikace výstupu pro každý vybraný a odfiltrovaný uzel**

Ukázkový dotaz

- ```
  (: Vypsání názvů všech minidisků do ►
  elementu minidisk :)

  for $p in doc('katalog.xml')//polozka
  where $p/kategorie = 'MiniDisc'
  order by $p/cena
  return
    <minidisk cena="{ $p/cena }">
      { string($p/nazev) }
    </minidisk>
  ```
  ```

Seskupování dat

- **přímá podpora není, ale pomocí funkce `distinct-value` lze jednoduše simulovat**
-

```
(: Vypsání kategorií, průměrné ceny a počtu ► 
výrobků v každé kategorii :)

for $k in ► 
distinct-values(doc('katalog.xml')//polozka/kategorie)
let $p := ► 
doc('katalog.xml')//polozka[kategorie = $k]
return
  <kategorie>
    <nazev>{ $k }</nazev>
    <prcena>{ avg($p/cena) }</prcena>
    <pocetv>{ count($p) }</pocetv>
  </kategorie>
```

Zanořování dotazů

- **XQuery je funkcionální jazyk a proto je možné jednotlivé výrazy navzájem kombinovat a zanořovat**
-
```
(: Vypsání všech kategorií a výrobků, které ► do ní patří :)

for $k in ► distinct-values(doc('katalog.xml')//polozka/kategorie)
return
  <kategorie>
    <nazev>{ $k }</nazev>
    {
      for $p in ► doc('katalog.xml')//polozka[kategorie = $k]
      return
        <vyrobek>{ string($p/nazev) ► }</vyrobek>
    }
  </kategorie>
```

Spojení dat

- jeden dotaz může sahat do libovolného množství dokumentů XML
- spojení dat se provede ručně, pomocí podmínky
- s využitím predikátu a proměnné funguje podobně jako OUTER JOIN

```
(: Spojení dvou XML dokumentů pomocí ► predikátu :)

for $p in doc('katalog.xml')//polozka
let $v := doc('vyrobci.xml')//vyrobce[nazev ► = $p/vyrobce]
return
  <polozka>
    { $p/nazev, $p/vyrobce, $v/web }
  </polozka>
```

- s využitím `where` funguje jako INNER JOIN

```
(: Spojení dvou XML dokumentů pomocí where ► :)

for $p in doc('katalog.xml')//polozka
for $v in doc('vyrobci.xml')//vyrobce
where $v/nazev = $p/vyrobce
return
  <polozka>
    { $p/nazev, $p/vyrobce, $v/web }
  </polozka>
```

Konstruování nových elementů

- dotaz může přímo obsahovat fragmenty kódu XML
- uvnitř fragmentů lze používat XQuery podvýrazy uzavřené do `{ … }`
- ```
  <pozdrav>Ahoj</pozdrav>,
    1 to 3,
    ►
  <osoba><jméno>Pepa</jméno><věk>30</věk></osoba>,
    <prcena>{ ►
  avg(doc('katalog.xml')//polozka/cena) ►
  }</prcena>
  ```
  ```

Generování HTML kódu

- **konstruování nových fragmentů lze výhodně využít například pro generování kódu HTML**
-
```
<html>
  <head>
    <title>Přehled výrobků</title>
  </head>
  <body>
    <table>
      <tr>
        <td>Název</td>
        <td>Cena</td>
      </tr>
      {
          for $p in ►
doc('katalog.xml')//polozka
          order by $p/nazev
          return
            <tr>
              <td> { string($p/nazev) } ►
</td>
              <td> { string($p/cena) } </td>
            </tr>
      }
    </table>
  </body>
</html>
```

Typový systém

Typový systém

- dokumenty XML mohou datové typy využívat různě
- XQuery proto musí nabízet velmi flexibilní práci s typovými informacemi
  - datové typy vůbec nepoužíváme
  - využíváme datové typy až při provádění dotazu
  - datové typy využíváme tak, že typovou kontrolu lze provést ještě před spuštěním dotazu
  - kombinujeme předchozí přístupy
- k dispozici jsou typy XML schémat a uživatelsky definované typy v importovaných schématech

Nepoužíváme typy

- pro dokumenty XML nemáme schéma
- v dotazech neprovádíme přetypování
- všechny hodnoty se pak chápou jako textové řetězce a pouze u funkcí, které očekávají jako parametr jiný typ (třeba číslo), se provede pokus o automatické přetypování
- špatně se řadí a porovnávají například čísla nebo datumy
-
```
(: Seřazení položek podle jejich ceny.
   Bez informace o typu se provede ► 
lexikografické sařezení. :)

for $p in doc('katalog.xml')//polozka
order by $p/cena
return
  <vyrobek>
    { $p/nazev, $p/cena }
  </vyrobek>
```

Ruční přetypování

- **v případě potřeby můžeme hodnotu ručně přetypovat na nějaký jiný datový typ**
- **(: Pomocí ručního přetypování dosáhneme ► korektního seřazení :)**

```
for $p in doc('katalog.xml')//polozka
order by xs:decimal($p/cena)
return
  <vyrobek>
    { $p/nazev, $p/cena }
  </vyrobek>
```

Automatické přiřazení typů

- pro dokument máme schéma, kde jsou definovány typy

```
<xs:element name="cena" type="xs:decimal"/>
```

- toto schéma naimportujeme do dotazu

```
import schema "urn:x-pokus:schemas:katalog" ►
at "katalog.xsd"
```

- jakékoliv dotazy na uzel odpovídající elementu `cena` nyní budou vracet číselný typ, a ne textový řetězec
-

```
(: Naimportováním schématu zajistíme ►
validaci dokumentu
   a přiřazení datových typů jednotlivým ►
uzlům.
   Seřazení podle elementu cena tak bude ►
fungovat správně. :)

import schema "urn:x-kosek:schemas:katalog" ►
at "katalog.xsd";
declare namespace k = ►
"urn:x-kosek:schemas:katalog";

for $p in ►
doc('katalog-s-typy.xml')//k:polozka
order by $p/k:cena
return
  <vyrobek>
```

Automatické přiřazení typů (Pokračování)

```
</vyrobek>
```

- **jedná se nepovinnou funkcionalitu XQuery implementací**

Využití typů a schémat

- validace a otypování vstupních dat
- validace generovaných dat
- signatury funkcí
- určení typů proměnných

```
(: Nenalezne-li se výrobce ohlásí dotaz ► chybu,
   protože proměnná $v nebude obsahovat ► právě jeden element :)

for $p in doc('katalog.xml')//polozka
let $v as element() := ► doc('vyrobci.xml')//vyrobce[nazev = ► $p/vyrobce]
return
  <polozka>
    { $p/nazev, $p/vyrobce, $v/web }
  </polozka>
```

- některé implementace nabízejí statickou typovou kontrolu

Další informace

Implementace XQuery

- **standard XQuery ještě není finalizovaný**
- **většina výrobců databází již ve stávajících nebo připravovaných produktech podporuje návrhy XQuery**
- **většinou je možné v jednom dotazu míchat SQL a XQuery**
- **existují i samostatné XQuery enginy nebo middlewarové produkty podporující XQuery**
- **přehled implementací**

  http://www.w3.org/XML/Query.html#products

Další zdroje informací

- **stránky W3C o XQuery**[7]
- **pěkný tutoriál základů XQuery**[8]
- **PDF verze těchto slidů**[9]
- **zdrojové texty příkladů**[10]

[7] http://www.w3.org/XML/Query
[8] http://www.datadirect.com/developer/xquery/xquerybook/index.ssp
[9] http://www.kosek.cz/xml/2005devcon/xquery.pdf
[10] http://www.kosek.cz/xml/2005devcon/priklady.zip