

Semistrukturierte Daten

XQuery

Stefan Woltran

Institut für Informationssysteme
Technische Universität Wien

Sommersemester 2011

Inhalt

- 1 Überblick
- 2 Diverses
- 3 FLWOR
- 4 Konstruktoren
- 5 Weitere Ausdrücke (Bedingte Ausdrücke, Quantoren, Joins, Groupings)
- 6 Funktionen
- 7 Module
- 8 Zusammenfassung und Links

Überblick

- XQuery steht für **XML Query Language**
 - aktuelle Version: XQuery 1.0 (W3C Recommendation seit 2007)
 - XQuery Version 1.0 ist eine Erweiterung von XPath Version 2.0
 - basiert auf dem XQuery/XPath Data Model (XDM)
 - Typsystem basiert auf XML Schema
 - bietet vordefinierte Funktionen wie schon aus XPath bekannt
 - syntaktisch korrekte und ausführbare Ausdrücke liefern in beiden Sprachen dasselbe
- **enge Verknüpfung**

Überblick

- XQuery ist die **Abfragesprache für XML-Dateien**
 - vergleichbar mit SQL für Datenbanken
- Hauptaufgaben:
 - Durchsuchen von Web-Dokumenten nach relevanten Daten
 - XML-Dateien zu XHTML-Dateien transformieren
 - gleichzeitiges Durchsuchen mehrerer Dokumente
 - ...
- Unterschiede gegenüber XPath und XSLT:
 - XPath dient zum Navigieren und Identifizieren von Knotenmengen
 - es können aber keine neuen Knoten/Bäume erzeugt werden
 - Selektion und Transformation werden von XQuery und XSLT unterstützt
 - XSLT wurde entwickelt, um XML-Dateien in "von Menschen lesbare" Dokumente zu transformieren, XQuery dient speziell zur Datenextraktion

Diverses

- Kommentare werden durch (: und :) begrenzt
- String-Literale:
 - Können entweder mit "" oder mit ' ' begrenzt werden
 - das jeweils andere Zeichen kann im String normal verwendet werden
 - zwei aufeinander folgende "" oder ' ' werden als ein Apostroph interpretiert

Beispiele

- XQuery-Kommentare:

```
(: Das ist ein XQuery-Kommentar :)
```

- Im Gegensatz zu der in XML-Dateien verwendeten Syntax:

```
<!-- Das ist ein XML-Kommentar -->
```

- String-Literale:

```
"ein 'String' "  
"noch ein ""String"""
```

Input Functions

- geben an, welche(s) XML-Dokument(e) verwendet werden
- `doc()` liefert den Dokumentenknoten einer einzigen durch die URI spezifizierten XML-Datei

Beispiele

```
doc('example.xml')  
doc('http://www.example.com/example.xml')
```

Input Functions

- `collection()` liefert eine Sequenz von Dokumentknoten, welche in der durch die URI spezifizierten XML-Datei angegeben sind

Beispiel

- XML-Datei:

```
<!-- Datei col.xml -->
<collection>
  <doc href="Buecher.xml"/>
  <doc href="Bestand.xml"/>
  <doc href="Autoren.xml"/>
</collection>
```

- XQuery-Ausdruck:

```
collection('col.xml')
```

liefert eine Sequenz, welche die drei in der XML-Datei angegebenen Dokumente enthält

Beispiel (XML-Dokument für folgende XQueries)

```
<?xml version="1.0"?>

<!-- Dateiname: Buchbestand.xml -->

<BUCHBESTAND>
  <BUCH Einband="Taschenbuch" Lagernd="ja">
    <TITEL>The Adventures of Huckleberry Finn</TITEL>
    <AUTOR>Mark Twain</AUTOR>
    <SEITEN>336</SEITEN>
    <PREIS>12.75</PREIS>
  </BUCH>
  <BUCH Einband="Taschenbuch" Lagernd="ja">
    <TITEL>In der Strafkolonie</TITEL>
    <AUTOR>Franz Kafka</AUTOR>
    <SEITEN>125</SEITEN>
    <PREIS>9.90</PREIS>
  </BUCH>
  ...
</BUCHBESTAND>
```

Beispiele

- einfache XQuery mittels Path Expression:

```
doc('Buchbestand.xml')/BUCHBESTAND/BUCH
```

liefert alle BUCH-Elemente, die vom document node aus durch /BUCHBESTAND/BUCH erreichbar sind (absoluter Pfad)

- Einschränken der Ergebnismenge durch Prädikate (Boolesche Bedingungen)

```
doc('Buchbestand.xml')//BUCH[AUTOR='Franz Kafka']
```

liefert alle BUCH-Elemente, deren Autor Franz Kafka ist (gesamtes Dokument)

- für Details: siehe Folien zu XPath

FLWOR

- ausgesprochen: "flower expressions"
 - steht für: **FOR-LET-WHERE-ORDER BY-RETURN**
 - ist ähnlich zu den SELECT-FROM-WHERE Ausdrücken in SQL
 - allerdings:
 - keine Tabellen, Zeilen und Spalten
 - bindet Variablen in FOR und LET Klauseln
 - liefert als Ergebnis Tupel
- jeder FLWOR-Ausdruck muss mindestens eine FOR- oder LET-Klausel enthalten

Beispiel

```
for $b in doc('Buchbestand.xml')//BUCH
where $b/PREIS < 10
return $b/TITEL
```

FOR-Klausel

- iteriert über alle Elemente in der angegebenen Sequenz
- bindet die Variable zu **jedem einzelnen Element**
- Reihenfolge der Elemente bleibt erhalten

Beispiel

```
for $i in (1, 2, 3)
return
<zahl>{$i}</zahl>
```

liefert

```
<zahl>1</zahl>
<zahl>2</zahl>
<zahl>3</zahl>
```

LET-Klausel

- bindet die Variable zum Ergebnis des **gesamten Ausdrucks**
- iteriert daher nicht über die einzelnen Elemente der Sequenz
- wird verwendet, um den Schreibaufwand zu reduzieren

Beispiel

Folgender Ausdruck erzeugt nur ein Tupel, \$i ist zur kompletten Sequenz gebunden:

```
let $i := (1, 2, 3)
return
<zahl>{$i}</zahl>
```

liefert

```
<zahl>1 2 3</zahl>
```

Beispiel (Verwendung von FOR und LET)

```
for $b in doc("buchbestand.xml")//BUCH
let $c := $b/AUTOR
return
  <BUCH>{$b/TITEL}<COUNT>{count($c)}</COUNT></BUCH>
```

liefert alle Buchtitel zusammen mit der Anzahl der AutorenInnen:

```
<BUCH>
  <TITEL>The Adventures of Huckleberry Finn</TITEL>
  <COUNT>1</COUNT>
</BUCH>
...
```

Mehrere FOR-Klauseln

- erzeugt **Kartesisches Produkt**
- Reihenfolge der Elemente im Ergebnis entsprechend den gebundenen Sequenzen von links nach rechts
- Verwendung für Joins (siehe später)

Beispiel

```
for $i in (1, 2),  
    $j in (3, 4)  
return  
<zahl><i>{$i}</i><j>{$j}</j></zahl>
```

liefert

```
<zahl><i>1</i><j>3</j></zahl>  
<zahl><i>1</i><j>4</j></zahl>  
<zahl><i>2</i><j>3</j></zahl>  
<zahl><i>2</i><j>4</j></zahl>
```

WHERE-Klausel

- filtert von FOR bzw. LET erzeugte Variablenbindungen
- Bedingung wird für jedes Tupel einmal überprüft

Beispiel

```
for $b in doc("buchbestand.xml")//BUCH
where $b/PREIS < 10.00
return $b/TITEL
```

liefert

```
<TITEL>In der Strafkolonie</TITEL>
<TITEL>The Legend of Sleepy Hollow</TITEL>
<TITEL>The Scarlet Letter</TITEL>
```

vergleiche mit äquivalentem XPath-Ausdruck

```
doc("Buchbestand.xml")//BUCH[PREIS < 10.00]/TITEL
```

ORDER BY-Klausel

- sortiert, bevor RETURN ausgeführt wird
- es kann nach einem oder mehreren Kriterien sortiert werden
- aufsteigend oder absteigend mittels `ascending` oder `descending`
- bei gleichen Werten kann die ursprüngliche Elementordnung mittels `stable` erhalten bleiben
- leere Sequenzen:
 - Reihenfolge ist nicht definiert (analog zu `null`-Werten in SQL)
 - Implementierungsunabhängigkeit kann mit `empty greatest` oder `empty least` garantiert werden

ORDER BY-Klausel

Beispiele

```
for $b in doc("buchbestand.xml")//BUCH
order by $b/AUTOR ascending, $b/TITEL descending
return $b/TITEL
```

```
for $b in doc("buchbestand.xml")//BUCH
stable order by $b/AUTOR empty greatest
return <buch>{$b/TITEL, $b/AUTOR}</buch>
```

RETURN-Klausel

- RETURN-Klausel wird für jedes Tupel aus dem Tupel-Stream einmal ausgewertet
- Ergebnisse werden aneinandergehängt
- ohne ORDER BY wird die Reihenfolge von den FOR- und LET-Klauseln bestimmt
- jeder beliebige XQuery-Ausdruck kann im RETURN vorkommen
- meistens werden Elementkonstruktoren verwendet

Konstruktoren

- es gibt Konstruktoren für element, attribute, document, text, comment und processing instruction nodes
- **direct element constructors:**
 - erzeugen element nodes
 - basieren auf Standard XML-Notation
 - können enclosed expressions beinhalten: durch geschwungene Klammern begrenzter Inhalt wird evaluiert und durch entsprechenden Wert ersetzt

Beispiel

```
<BUCH Einband="{ $b/@Einband }">  
    { $b/TITEL }  
</BUCH>
```

■ computed constructors:

- beginnt mit dem Schlüsselwort, welches den Typ des Knotens definiert
- bei Typen, welche einen Namen haben, folgt auf das Schlüsselwort der Name des Knotens
- in geschwungenen Klammern folgt der Inhalt des Knotens, die context expression

Beispiel

```
element BUCH {  
  attribute Einband {"Taschenbuch"},  
  element TITEL {"In der Strafkolonie"}  
}
```

Bedingte Ausdrücke

- XQuery unterstützt IF-THEN-ELSE-Ausdrücke

Beispiel

```
if ($b1/PREIS < $b2/PREIS)
  then $b1
  else $b2
```

Quantoren

- Existenz- und All-Quantoren
- besteht aus dem Schlüsselwort `every` oder `some`, gefolgt von einer oder mehreren `in`-Klauseln, dem Schlüsselwort `satisfies` und dem zu überprüfenden Ausdruck

Beispiele

```
every $b in doc('Buchbestand.xml')//BUCH  
satisfies $b/@Lagernd
```

liefert `true`, wenn alle BUCH-Elemente das Attribut `Lagernd` besitzen (unabhängig vom Attributwert)

```
some $b in doc('Buchbestand.xml')//BUCH  
satisfies ($b/PREIS < 10)
```

liefert `true`, wenn mindestens ein Buch einen Preis kleiner 10 hat

Joins

Beispiel (Inner Join)

```
<BUCHBESTAND>
{
  for $b in doc("Buecher.xml")//BUCH,
      $a in doc("Autoren.xml")//AUTOR[@id = $b/@Autorid]
  return
    <BUCH Einband="{ $b/@Einband}">
    {
      $b/TITEL,
      <AUTOR>{string($a/VORNAME), string($a/NACHNAME)}
    }
  </BUCH>
}
</BUCHBESTAND>
```

mehrere Quellen in einem Ergebnis kombinieren
liefert nur Ergebnisse für Bücher, deren AutorIn angegeben ist (und umgekehrt)

Joins

Beispiel (Left Outer Join)

```
for $a in doc("Autoren.xml")//AUTOR
return
  <AUTOR>
  {
    $a/VORNAME,
    $a/NACHNAME,
    for $b in doc("Buecher.xml")//BUCH[@Autorid = $a/@id]
    return
      $b/TITEL
  }
  </AUTOR>
```

listet alle AutorenInnen

zusätzlich werden zu diesen die verfassten Bücher angegeben (falls vorhanden)

Joins

Beispiel (Full Outer Join)

```
<GESAMT>
{
  for $a in doc("Autoren.xml")//AUTOR
  return
    <AUTOR>
      {$a/VORNAME, $a/NACHNAME,
        for $b in doc("Buecher.xml")//BUCH[@Autorid=$a/@id]
        return
          $b/TITEL }
      </AUTOR>,
    <KEINAUTOR>
      {for $b in doc("Buecher.xml")//BUCH
       where empty(doc("Autoren.xml")//AUTOR[@id=$b/@Autorid])
       return
         $b/TITEL }
      </KEINAUTOR>
}
</GESAMT>
```

listet alle AutorenInnen

zusätzlich werden diesn die verfassten Bücher angegeben (falls vorhanden)

fügt eine Liste mit allen Büchern hinzu, zu denen keinE AutorIn angegeben ist

Groupings

- ermöglicht Gruppieren von Daten
- Verwendung von Aggregatfunktionen (z.B. `fn:count` oder `fn:avg`)

Beispiel

```
for $a in distinct-values(doc("Buchbestand.xml")//AUTOR)
let $b := doc("Buchbestand.xml")//BUCH[AUTOR = $a]
where count($b) >= 2
return
  <AUTOR Anzahl="{count($b)}">
    {$a}
  </AUTOR>
```

`fn:distinct-values` eliminiert alle Duplikate von AutorInnen in `Buchbestand.xml`

`$a` repräsentiert jeweils eineN AutorIn, `$b` eine Sequence von Büchern zu der entsprechenden Person

auf diesem Set kann die Aggregatfunktion `fn:count` aufgerufen werden

Funktionen

- neben Built-in Functions (siehe XPath-Folien) können auch **eigene Funktionen** geschrieben werden
- ermöglicht das Aufteilen komplexer Queries in kleinere, übersichtliche Teile
- Funktionsdeklarationen bestehen aus den Schlüsselwörtern `declare function`, gefolgt von einem Funktionsnamen, den Datentypen und Namen der Parameter und dem Datentyp des Rückgabewertes
- jede Funktion muss in einem Namespace liegen
- die Datentypen entsprechen den aus XML-Schema bekannten Datentypen
- **rekursive Funktionsdeklarationen** sind erlaubt

Beispiel (Funktionen)

■ Funktion:

```
declare function local:mul($a as xs:decimal,  
$b as xs:decimal) as xs:decimal  
{  
  let $i := ($a * $b)  
  return $i  
};
```

definiert die Funktion `local:mul`, welche als Parameter zwei Dezimalzahlen übernimmt, diese multipliziert und das Ergebnis zurückliefert

■ kann zum Beispiel folgendermaßen aufgerufen werden:

```
<zahl>{local:mul(2,7)}</zahl>
```

Beispiel (Funktionen)

■ Funktion:

```
declare function local:format-authors
($auths as element(AUTOR)*)
as element(f-author)*
{
  for $a in $auths
  return
    <f-author>
      {string($a/NACHNAME)}, {string($a/VORNAME)}
    </f-author>
};
```

erwartet als Eingabe eine beliebig lange Sequenz von AUTOR-Elementen
formatiert die AutorInnen und liefert neue f-author-Elemente zurück

■ kann zum Beispiel folgendermaßen aufgerufen werden:

```
<f>{local:format-authors(doc("Autoren.xml")//AUTOR)}</f>
```

Module

- jede Query kann aus mehreren Fragmenten, sog. Modulen, zusammengestellt werden
- Module werden durch `module namespace`, gefolgt von Präfix und URI deklariert
- Module können von anderen Modulen oder Queries importiert werden
- durch `at` kann eine lokale Modul-Datei angegeben werden

Beispiele

```
module namespace format = "http://example.com/format";  
(: hier folgen Funktions- oder Variablendefinitionen :)
```

```
import module namespace  
f = "http://example.com/format" at "file:format.xq";
```

Zusammenfassung

- XQuery für XML-Dateien ist vergleichbar mit SQL für Datenbanken
- FLWOR (for - let - where - order by - return)-Expressions sind SELECT-Statements in SQL ähnlich
- XQuery unterstützt die Verwendung von Existenz- und All-Quantoren
- Joins und Groupings sind möglich
- in XQuery können die von XPath bekannten Built-in Functions verwendet werden
- es können auch eigene Funktionen geschrieben werden

Links

- W3C Recommendation: <http://www.w3.org/TR/xquery/>
- für die Ausführung von Queries kann der XSLT und XQuery-Processor SAXON verwendet werden: <http://saxon.sourceforge.net/>
- `saxonX.jar` muss in dem Java-Classpath hinzugefügt werden
- Aufruf von Queries mittels `java net.sf.saxon.Query xquery.xq`