Universität Kaiserslautern, AG Datenbanken und Informationssysteme

Seminar XML und Datenbanken

Thema: XQuery 1.0 und XPath 2.0

Jan Schäfer
j_schaef@informatik.uni-kl.de

31. Januar 2003

Übersicht

- Einleitung
- XQuery 1.0 Grundlagen
- XPath 2.0
- FLWOR
- Zusammenfassung
- Ausblick

Einleitung

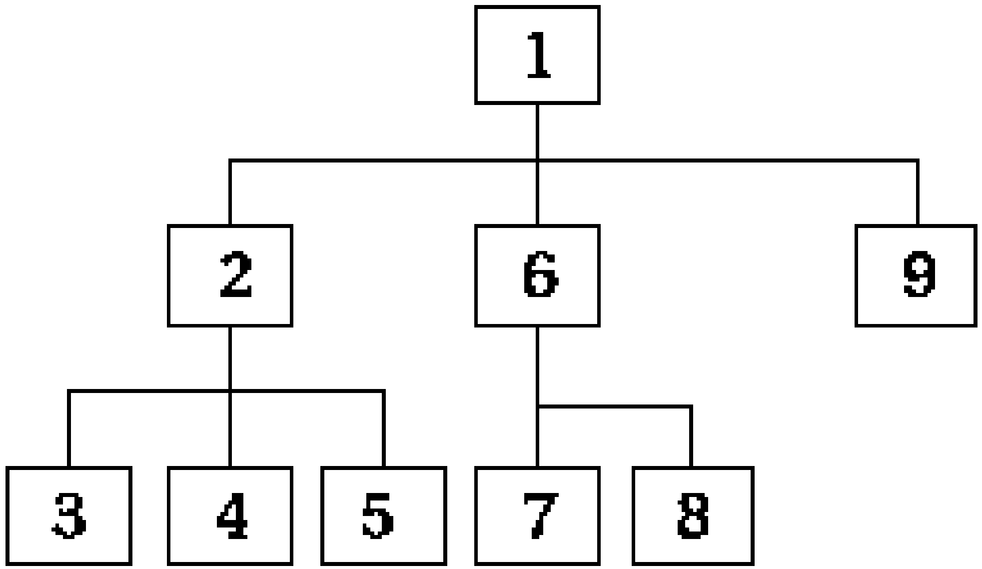
- XQuery 1.0 - XML-Anfragesprache
- XPath 2.0 - Sprache zur Adressierung von XML-Knoten
- Beide Spezifikationen zur Zeit noch working draft des W3C
- Datentypen in XML-Schema definiert
- Datenmodell separat spezifiziert
- Bibliothek von Operatoren und Funktionen

XQuery 1.0 Grundlagen

- Anfragesrache für XML-Dokumente
- Vollständige funktionale Programmiersprache
- Stark getypt
- Nicht XML-konforme Syntax
- Ergebnis ist eine Liste und nicht unbedingt korrektes XML-Dokument !

Datenmodell

- Datenmodell ist ein knotenbeschrifteter Baum
- Jeder Knoten hat eine eindeutige Identität
- Dokumentordnung ist eindeutige totale Ordnung auf dem Datenmodell



Beispiel 1

Berechne die Summe aller Zahlen von 1 bis 100 !

Anfrage :

```
let $i := (1 to 100)
return sum($i)
```

Ergebnis :

```
5050
```

- Kein XML-Knoten vorhanden !

Beispiel 2

Was ist (4+6)*2 ?

Anfrage :

```
(4+6)*2
```

Ergebnis :

```
20
```

- Völlig korrekter XQuery-Ausdruck !

Datentypen

- Atomare Typen, definiert in XML-Schema
  - `string`, `integer`, `boolean` usw.
  - benutzerdefinierte Typen
- Knotentypen
  - Dokumentknoten
  - Elementknoten
  - Attributknoten
  - Namensraumknoten
  - Processing-Instruction-Knoten `<? ?>`
  - Kommentarknoten `<!-- -->`
  - Textknoten

Ausdrücke

- Grundbaustein von XQuery sind Ausdrücke
- Beliebig schachtelbar
- Ergebnis ist immer eine **Liste**
- Primäre Ausdrücke
  - Literale : `1, "a", 5.65`
  - Variablen : `$foto`
  - Funktionsaufrufe : `contains($farbe, "rot")`
  - geklammerte Ausdrücke : `(5+4)*3`
- Arithmetische Ausdrücke : `+, -, *, mod, idiv, div`

Ausdrücke (2)

- Vergleichsausdrücke :
  - Wertevergleich : eq, ne, lt, le, gt, ge
  - Allgemeiner Vergleich : =, !=, <, <=, >, >=
  - Knotenvergleich : is, isnot
  - Ordnungsvergleich : <<, >>
- Logische Ausdrücke : and, or, not
- Bedingte Ausdrücke : if ($a eq 5) then 1 else 2
- Quantifizierte Ausdrücke :
  some | every <Variable> in <Liste> satisfies <Ausdruck>

Der Listentyp

- Geordnete Liste aus atomaren Werten oder XML-Knoten
- Ergebnis jedes XQuery-Ausdrucks
- Kann mehrere, ein oder kein Element enhalten
- Duplikate sind möglich
- Elemente werden durch Kommas getrennt und i. d. R. geklammert
- *Beispiele :*
  - `1`
  - `(3,3,"rot",3+4)`
  - `(<eins/>,<zwei/>,<drei/>)`

Konstruktoren

- Dient zum Erzeugen von XML-Knoten

- Elementkonstruktoren :

```
<foto datum="{$date}">
  <url>berghuette.jpeg</url>
</foto>
```

- Berechnete Konstruktoren :

```
element foto {
  attribute datum { $date },
  element url { "berghuette.jpeg" }
}
```

XPath 2.0

- Sprache zur Adressierung von Elementen eines XML-Dokuments
- Steht nicht alleine sondern braucht Wirtssprache
- Wird in XQuery 1.0 und XSLT 2.0 verwendet
- Alle XPath 2.0 Ausdrücke sind prinzipiell auch korrekte XQuery 1.0 Ausdrücke
- Ausnahmen sind z. B. einige Rückwärtsachsen
- Hauptunterschied zu XPath 1.0 : Listen statt Mengensemantik

Der Pfadausdruck

- Kernausdruck von XPath
- Relativ zu Kontextknoten oder absolut (beginnt mit „/”)
- Besteht aus durch „/” getrennten Lokalisierungsschritten
- Ein Lokalisierungsschritt besteht aus :
  - einem primärern Ausdruck oder Achsenname::Knotenabfrage
  - einem optionalen Prädikat
- *Beispiel :*

```
document("fotoalbum.xml")/descendent::gruppe[@name="sommerurlaub"]
```

Achsen

- Beschreibt Liste von Knoten, ausgehend vom Kontextknoten
- Es gibt 6 verschiedene Achsen in XQuery :

Achsenname	*Beschreibung*
child::	Die direkten Kindknoten des Kontextknotens
descendent::	Alle Kind- und Kindeskindknoten des Kontextknotens bis zu den Blättern
descendant-or-self::	Das selbe wie descendent, nur ist der Kontextknoten selbst auch enthalten
attribute::	Die Attribute des Kontextknotens
self::	Der Kontextknoten selbst
parent::	Der direkte Elternknoten

- *Beispiel :* `/`**`desendent::`**`node()/`**`attribute::`**`name`

Knotenabfrage

- Bedingung die für jeden Knoten eines Schrittes wahr sein muss
- Hängt entweder vom Knotentyp oder vom Namen ab
- Es gibt folgende Knotenabfragen :

Abfragename	*Beschreibung*
text()	Selektiert alle Textknoten
comment()	Selektiert alle Kommentarknoten
processing-instruction()	Selektiert alle processing-instruction-Knoten
node()	Selektiert alle Knoten, egal welchen Typs
name	Selektiert alle Knoten mit dem Namen *name*
*	Trifft auf jeden Namen zu

- *Beispiel :* `child::ort/child::text()`

Prädikate

- Optionale Angabe eines Ausdruck in eckigen Klammern [ ]
- Dient zum Filtern der Trefferliste
- Wahrheitswert wird folgendermaßen bestimmt :
  1. Ist des Prädikat eine Zahl, ist der Ausdruck genau dann `true`, wenn die Kontextposition des Knotens dieser Zahl entspricht.
  2. Ansonsten entspricht er dem boolschen Wert des Prädikats.
- *Beispiele :*
  - `child::kapitel[2]`
  - `descendant::spielzeug[attribute::farbe = "rot"]`

Abkürzende Schreibweise

- Häufig benutzte Pfadausdrücke können abgekürzt werden
- Es gibt folgende Abkürzungsmöglichkeiten :
  - `child::` kann ganz weggelassen werden, d. h. `child::` ist die Standardachse.
  - `attribute::` kann mit `@` abgekürzt werden.
  - `//` ist die Kurzform für `/descendant-or-self::node()/`
  - `.` steht für den Kontextknoten selbst.
  - `..` ist kurz für `parent::node()`.
- *Beispiel :*
  - Normal : `child::foto[attribute::datum="2002-20-08"]`
  - Abkürzung : `foto[@datum="2002-20-08"]`

Der FLWOR-Ausdruck

- FLWOR (sprich: flower) steht für `for`, `let`, `where`, `order by`, `return`
- Allgemeiner Aufbau :

```
for <Variable> in <Liste>
let <Variable> := <Liste>
where <Ausdruck>
order by <Ordnungsliste>
return <Audruck>
```

- Beschreibung :
  - `for, let` - Erzeugen eines Tupelstroms und Binden von Variablen an Tupel/Zeilen
  - `where` - Filtern des Tupelstroms
  - `order by` - Sortierung des Tupelstroms
  - `return` - Erzeugen eines Ergebniselements pro Tupel

Beispiel

Gib aus der Datei fotoalbum.xml alle `url`-Elemente der Fotos aus, die im Beschreibungstext das Wort ,,Sand" enthalten. Die `url`-Elemente sollen absteigend sortiert sein.

Anfrage :

```
for $foto in document("fotoalbum.xml")//foto
let $url := $foto/url
where contains($foto/beschr/text(),"Sand")
order by $url descending
return {$url}
```

Ergebnis :

```
<url>wueste.jpeg</url>
<url>aegypten.jpeg</url>
```

Beispiel Erklärungen

- `for $foto in document("fotoalbum.xml")//foto`
  - Erzeugung eines Tupelstroms aller foto-Elemente
  - Binden der Variable `$foto` an das jeweils aktuelle foto-Element

- `let $url := $foto/url`
  - Binden der Variable `$url` an das url-Element des aktuellen foto-Elements

- `where contains($foto/beschr/text(),"Sand")`
  - Verwerfen der Tupel, die das Wort ,,Sand” nicht im Beschreibungstext enthalten.

- `order by $url descending`
  - Sortieren der restlichen Tupel absteigend nach dem url-Element.

- `return {$url}`
  - Zurückliefern eines `url`-Elementes pro Tupel.

Vergleich mit SQL

XQuery :

```
for $foto in document("fotoalbum.xml")//foto
let $url := $foto/url
where contains($foto/beschr/text(),"Sand")
order by $url descending
return {$url}
```

SQL :

```
SELECT url
FROM fotoalbum
WHERE beschr LIKE '%Sand%'
ORDER BY url DESC
```

```
FROM fotoalbum
WHERE beschr LIKE '%Sand%'
ORDER BY url DESC
SELECT url
```

Beispiel : Joins

Beispiel : Gib die Orte (ohne Duplikate) aller Fotos samt ihrer Beschreibung aus dem Lexikon aus:

```
let $fotos := document("fotoalbum.xml")//foto
for $ort in distinct-values($fotos/ort) ,
    $begriff in document("lexikon.xml")//begriff
where $ort/text() eq $begriff/@name
return
<ortsbeschreibung>
{$ort}
{$begriff/beschr}
</ortsbeschreibung>
```

Ergebnis :

```
<ortsbeschreibung>
  <ort>Kairo</ort>
  <beschr>Hauptstadt von Ägypten</beschr>
</orstbeschreibung>
<ortsbeschreibung>
  <ort>Ägypten</ort>
  <beschr>Land in Nordafrika</beschr>
</ortsbeschreibung>
[...]
```

Gruppierung

Beispiel: Gib alle Orte aus, von denen es mindestens ein Foto gibt, das vor dem 22.8.2002 oder nach dem 26.12.2002 gemacht wurde:

```
let $doc := document("fotoalbum.xml")
for $ort in distinct-values($doc//ort)
let $fotos := $doc//fotos[ort/text() eq $ort/text()]
where some $datum in $fotos/@datum satisfies
      (xs:date($datum) lt xs:date("2002-08-22") or
       xs:date($datum) gt xs:date("2002-12-26")    )
return $ort
```

Ergebnis :

```
<ort>Alpen</ort>
<ort>Kairo</ort>
```

Schachtelungen

Beispiel: Gib für jede Gruppe das Foto mit der maximalen Größe aus

```
for $gruppe in document("fotoalbum.xml")/gruppe
return
  <gruppe name="{$gruppe/@name}"> {
      for $foto in $gruppe/foto
      where $foto/kbytes eq max($gruppe/foto/kbytes)
      return $foto  }
  </gruppe>
```

Ergebnis:

```
<gruppe name="sommerurlaub">
  <foto datum="24.8.2002">
    <ort>Sahara</ort>
    <beschr>Eine Karavane zieht über eine Sanddüne</beschr>
    <url>sahara.jpeg</url>
    <kbytes>800</kbytes>
  </foto>
</gruppe>
<gruppe name="skiurlaub">
  <foto datum="27.12.2001">
    <ort>Alpen</ort>
    <beschr>Es ist die Zugspitze zu erkennen</beschr>
    <url>zugspitze.jpeg</url>
    <kbytes>400</kbytes>
  </foto>
</gruppe>
```

Zusammenfassung

- XQuery 1.0 Grundlagen
  - Anfragesprache für XML
  - Selbst nicht XML-konform
  - Beliebig schachtelbar
- XPath 2.0
  - Adressierung von XML-Knoten
  - In XQuery 1.0 (z.T.) enthalten
- FLWOR
  - Ähnlich zu SQL
  - Mächtige Audrücke möglich
  - Leicht erlernbar

Ausblick

- Recommendation Ende 2003 erwartet
- Update-Fähigkeiten werden danach in Angriff genommen
- XQueryX - XML-Syntax für XQuery

Fragen ?