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

22. Januar 2003

Inhaltsverzeichnis

1 Einführung

XML wird immer häufiger zum Speichern von Daten verwendet. Es gibt mittlerweile XML-Datenbanken, die auf das Speichern von XML Dokumenten spezialisiert sind. Umso wichtiger wird es deshalb, eine standardisierte Sprache ähnlich zu SQL zu entwickeln, die es ermöglicht Anfragen auf XML Dokumenten zu stellen.

XQuery 1.0 liegt zur Zeit als working draft [1] [3] bei dem W3C Konsortium vor und soll in Zukunft die Standard-Sprache für XML-Anfragen werden.

Man kann XQuery nicht getrennt betrachten, sondern muss es im Kontext von anderen Spezifikationen des W3C sehen. XQuery ist abhängig von den folgenden Spezifikationen :

- Dem XQuery Datenmodell, das die Informationen eines XML-Dokumentes beschreibt auf die XQuery zugreifen kann. Das Datenmodell ist in [5] definiert.
- Dem XQuery Typensystem, das auf XML-Schema [2] basiert.
- Der Bibliothek von Funktionen und Operatoren, die von XQuery unterstützt werden. Sie sind in [6] definiert.

Des Weiteren ist die Spezifikation von XPath 2.0 [1] zu einem grossen Teil in der von XQuery 1.0 enhalten. Nicht alle Ausdrücke von XPath 2.0 sind in XQuery 1.0 erlaubt. Es gilt jedoch, dass ein Ausdruck, der in beiden Sprachen syntaktisch korrekt ist, auch in beiden Sprachen das selbe Ergebnis liefert.

XPath 2.0 ist eine Sprache, um Elemente von XML-Dokumenten adressieren zu können.

2 Grundlagen

2.1 Datenmodell

Da XML-Dokumente baumförmig sind, ist auch das Datenmodell entsprechend definiert. Das Datenmodell von XQuery ist ein knotenbeschrifteter Baum, bei dem jeder Knoten eine eindeutige Identität hat. Die Identität eines Knotens wird bei dessen Erzeugung festgelegt und kann danach nicht mehr verändert werden. Knotenbeschriftet bedeutet, dass jeder Knoten einen Namen haben muss, die Kanten jedoch nicht beschriftet sind.

[1] Ein working draft ist die erste Stufe des Standardisierungsprozesses des W3C, d. h. es kann sich noch einiges bis zum endgültigen Standard ändern.

2.2 Dokumentordnung

Auf allen Knoten eines Dokuments ist eine totale Ordnung, die Dokumentordnung definiert. Sie entsteht durch eine in-order, depth-first, left-to-right Traversierung des Datenmodells. Der Dokumentknoten ist der erste Knoten, Elternknoten sind vor Kindknoten und Nachbarknoten sind von links nach rechts geordnet. In Abbildung 1 wird die Reihenfolge verdeutlicht. Die Zahlen in den Knoten sind dabei die Position in der Ordnung.

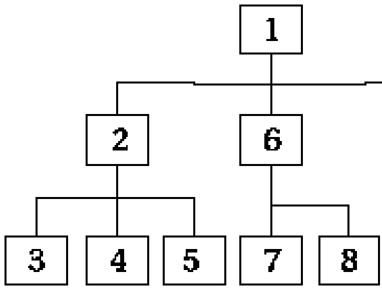

Abbildung 1: Dokument-Ordnung

2.3 Datentypen

XQuery ist eine stark-getypte funktionale Sprache. Operanden von Ausdrücken, Funktionen und Operatoren müssen den bestimmten Typen entsprechen. Die in XQuery vordefinierten Typen sind einmal die in XML-Schema [2] definierten *atomaren* Typen und die *Knotentypen*. Des Weiteren gibt es noch den *Listentyp*, der es erlaubt, Listen von Elementen zu erstellen.

Atomare Typen sind die aus Programmiersprachen bekannten Typen wie `integer`, `string` oder `boolean`, um nur einige zu nennen.

Knotentypen können einer der folgenden Typen sein :

- Dokumentknoten
- Elementknoten
- Attributknoten
- Namensraumknoten
- Processing-Instructionknoten

- Kommentarknoten
- Textknoten

2.4 Ausdrücke

Alles in XQuery ist ein **Ausdruck**.

Es gibt folgende Arten von Ausdrücken :

- Primäre Ausdrücke
- Arithmetische Ausdrücke
- Vergleichsausdrücke
- Logische Ausdrücke
- Listenausdrücke
- Ungeordnete Ausdrücke
- Bedingte Ausdrücke
- Quantifizierte Ausdrücke
- Konstruktoren
- Pfadausdrücke
- FLWOR-Ausdrücke
- und weitere

Auf alle Ausdrücke im Detail einzugehen, würde den Rahmen dieses Seminars sprengen. Ich werde hier deshalb nur kurz auf die verschiedenen Ausdrücke eingehen und mich dann auf die Pfad- und FLWOR-Ausdrücke konzentrieren.

2.4.1 Primäre Ausdrücke

Zu den primären Ausdrücken zählen Literale, Variablen, Funktionsaufrufe und geklammerte Ausdrücke. Literale stellen atomare Werte da und sind entweder Zahlen oder

Zeichenketten. Variablen beginnen mit einem „$” und werden zu dem Wert ausgewertet, an den sie gebunden wurden. Funktionsaufrufe haben die Syntax : *funktionsname(Parameter1,...)*. Geklammerte Ausdrücke sind Ausdrücke, die durch „(” und „)” eingeschlossen sind.

Beispiele für primäre Ausdrücke :

```
1 , 4.5 , $variable , contains(s,"a"), (5)
```

2.4.2 Arithmetische Ausdrücke

Arithmetische Ausdrücke bestehen aus zwei Operanden und den Operatoren : `+`, `-`, `*`, `div`, `idiv` oder `mod`. `Div` ist dabei die Gleitkommadivision `idiv` die ganzzahlige Division. Es hängt grundsätzlich von den Typen der Operanden ab, wie sich die einzelnen Operatoren verhalten. Wenn man z. B. zwei Operanden vom Typ Datum hat, ist das Ergebnis einer Subtraktion eine Zeitspanne. + und - können auch als einstellige Operatoren verwendet werden.

Beispiele für Arithmetische Ausdrücke :

```
-2*(3+5) , $foto1/@datum - $foto2/@datum , 3 div 2
```

2.4.3 Vergleichsausdrücke

Um Werte zu vergleichen gibt es die Vergleichsausdrücke. XQuery bietet vier Vergleichsarten an

- Wertevergleich
- Allgemeiner Vergleich
- Knotenvergleich
- Ordnungsvergleich

Der Wertevergleich dient zum Vergleichen von einfachen Werten. Das Ergebnis eines Vergleichs ist entweder `true` oder `false`.

Jeder Wertevergleichsoperator hat einen korrespondierenden allgemeinen Vergleichsoperator (Tabelle 1). Die Operanden des allgemeinen Vergleichs können jedoch Listen

Wertevergleich	*Allgemeiner Vergleich*
eq	=
ne	!=
lt	<
le	<=
gt	>
ge	>=

Tabelle 1: Vergleichsoperatoren

beliebiger Länge sein. Das Ergebnis eines allgemeinen Vergleichs ist ebenfalls immer `true` oder `false`.

Ein allgemeiner Vergleich von zwei Listen ist genau dann `true`, wenn der entsprechende Wertevergleich für mindestens einen Wert aus den beiden Listen `true` ist.

Beispiel : `(1,2,3) > (10,6,2)` ist `true`, da 3 `gt` 2.

Der *Knotenvergleich* vergleicht die Identität von zwei Knoten. Es gibt die beiden Operatoren `is` und `isnot`.

Beispiel : `$ort1/.. is $ort2/..`

testet, ob der Elternknoten der beiden Orts-Knoten identisch ist.

Um die Reihenfolge zweier Knoten in der Dokumentordnung zu vergleichen, gibt es den *Ordnungsvergleich*. Die Operatoren sind dafür << und >>. << ist `true`, wenn der erste Knoten vor dem zweiten Knoten liegt, >> genau umgekehrt.

2.4.4 Logische Ausdrücke

Logische Ausdrücke dienen zum Verknüpfen von boolschen Werten. Operatoren sind `and`, `or` und `not`.

Beispiele : `1 eq 1 and 2 eq 2` , `1 eq 1 or 2 gt 3`

2.4.5 Konstruktoren

Konstruktoren dienen zum Erzeugen von XML-Strukturen. Der einfachste und wichtigste Konstruktor ist der *Elementkonstruktor*. Er basiert auf der Standard-XML Schreibweise. Zum Beispiel erzeugt der folgende Ausdruck ein `buch`-Element mit Attribut, Unterelementen und Text :

```
 <buch isbn="0-261-10235-4">
  <titel>The fellowship of the ring</titel>
  <autor name="J.R.R. Tolkien"/>
</buch>
```

Text in geschweiften Klammern { } innerhalb eines Elementkonstruktors wird als XQueryAusdruck interpretiert und durch den Wert des Ausdrucks ersetzt.

Beispiel: `<addition>{1+2}</addition>` wird zu `<addition>3</addition>` ausgewertet.

Eine weitere Möglichkeit Elemente zu erzeugen, ist das Verwenden eines *berechneten* Konstruktors. Das Beispiel von eben würde damit folgendermaßen aussehen :

```
element buch
{
  attribute isbn { "0-261-10235-4" },
  element titel { "The fellowship of the ring" },
  element autor
  {
    attribute name { "J.R.R. Tolkien" }
  }
}
```

Berechnete Konstruktoren haben unter anderem den Vorteil, dass der Name eines Elementes berechnet werden kann.

2.4.6 Bedingte Ausdrücke

In XQuery kann man eine bedingte Abarbeitung durch die Schlüsselwörter `if`, `then`, `else` erreichen.

Beispiel :

```
if ($foto1/kbytes lt $foto2/kbytes)
then $foto1
else $foto2
```

2.4.7 Quantifizierte Ausdrücke

Es gibt existenz- und allquantifizierte Ausdrücke. Beide liefern entweder `true` oder `false` zurück und haben den allgemeinen Aufbau :

```
some|every <Variable> in <Liste> satisfies <Ausdruck>
```

Ein `some`-Ausdruck ist dann `true`, wenn der gegebene Ausdruck für mindestens ein Element der Liste `true` ist. Ein `every`-Ausdruck ist nur `true`, wenn der gegebene Ausdruck für alle Elemente der Liste `true` ist.

Beispiele :

- `some $i in (1,2,3) satisfies $i+1 = 4`
  wird zu `true` ausgewertet.
- `every $foto in //foto satisfies $foto/@datum`
  wird zu `true` ausgewertet, wenn alle `foto`-Elemente ein `datum`-Attribut haben, unabhängig von dessen Wert.

2.5 Listen

2.5.1 Grundlagen

Da XQuery eine Sprache ist, um Anfragen auf XML-Dokumenten zu stellen, könnte man meinen, dass jeder XQuery-Ausdruck auch immer ein XML-Dokument als Ausgabe hat. Dem ist allerdings nicht so. Alle XQuery-Ausdrücke haben gemeinsam, dass das Ergebnis eines XQuery-Ausdrucks immer eine *Liste* ist. Auf jeder Liste ist eine eindeutige Ordnung gemäß der Dokumentordnung definiert. Elemente einer Liste können entweder atomare Werte sein oder XML-Knoten.

Beispiele für Listen :

- `1` - eine Liste, die nur aus dem atomaren Wert 1 besteht
- `(1,2,2,3)` - Liste aus mehreren atomaren Werten
- `(<b>fett</b>,<i>kursiv</i>,"normal")` - Liste aus XML-Knoten und atomarem Wert

Listen können nicht geschachtelt werden. Eine geschachtelte Liste wird zu einer flachen Liste umgewandelt, d. h., `1,2,(2,3),4` würde zu `1,2,2,3,4` ausgewertet. Eine Liste muss im Allgemeinen geklammert sein, die Klammern können jedoch weggelassen werden, wenn die Liste an Top-Level eines XPath Ausdrucks steht.

2.5.2 Der to-Operator

Man kann eine Liste von Integern mit dem `to`-Operator erzeugen.
Der Ausdruck `(1,5 to 8)` z. B. wird zu `(1,5,6,7,8)` ausgewertet.

2.5.3 Listenoperatoren

Es gibt verschiedene Operatoren auf Listen :

- union bzw. | - Vereinigung zweier Listen
- intersect - Schnitt zweier Listen
- except - Differenz zweier Listen

Alle Operatoren führen eine Duplikat-Eliminierung durch.

Beispiele :

- `(1,2,3) union (3,4)` wird zu `(1,2,3,4)`
- `(1,2,3) intersect (3,4)` wird zu `(3)`
- `(1,2,3) except (3,4)` wird zu `(1,2)`

2.5.4 Listenfunktionen

Es gibt eine ganze Reihe von vordefinierten Funktionen auf Listen, die in einer von XQuery separaten Spezifikation definiert sind [6]. Wichtige Funktionen sind z. B. :

- `avg` - Durchschnitt aller Werte der Liste
- `sum` - Summe aller Werte der Liste
- `max` - Maximaler Wert der Liste

- `count` - Anzahl der Elemente der Liste
- `exists` - Ob ein Element in der Liste enhalten ist.
- `distinct-values` - Wertbasierte Duplikat-Eleminierung
- `distinct-nodes` - Knotenidentitätsbasierte Duplikat-Eleminierung

3 XPath 2.0

3.1 Grundlagen

XPath 2.0 ist eine Sprache, um Elemente eines XML-Dokuments adressieren zu können. XPath ist selbst nicht XML konform, sondern orientiert sich eher an gängigen Mechanismen zum Bilden von Pfaden, wie sie auch aus Dateisystemen bekannt sind. Daher kommt auch der Name XPath. Wie jeder XQuery-Ausdruck liefert auch jeder XPath-Ausdruck immer eine Liste zurück.

Zum Beispiel : Gib alle Beschreibungen der Fotos aus der Datei `fotoalbum.xml` aus (siehe A.1), die in den Alpen gemacht wurden :

```
document("fotoalbum.xml")//foto[ort = "Alpen"]/beschr
```

Ergebnis :

```
<beschr>Foto einer langweiligen Berghütte</beschr>
<beschr>Es ist die Zugspitze zu erkennen</beschr>
```

Ein Pfadausdruck ist entweder relativ zu einem aktuellen Kontextknoten oder ein absoluter Pfad. In letzterem Fall fängt der Ausdruck mit „/” an, und der Kontextknoten ist der Wurzelknoten des Dokuments. Von dem Kontextknoten aus wird der Pfad dann mit Lokalisierungsschritten weitergebildet. Ein Lokalisierungsschritt besteht aus einem Achsennamen, einer Knotenabfrage und einem Prädikat. Zwischen zwei Lokalisierungsschritten muss immer ein „/” stehen.

3.2 Achsen

Eine Achse beschreibt eine Liste von Knoten, ausgehend von dem Kontextknoten. In XQuery kann man 6 verschiedene Achsen benutzen, die in Tabelle 2 aufgelistet sind.

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

Tabelle 2: Achsen

3.3 Knotenabfrage

Eine Knotenabfrage ist eine Bedingung, die für jeden Knoten eines Schrittes wahr sein muss. Die Bedingung kann entweder von dem Knotentyp (element, attribute, text, usw.) oder von dem Namen des Knotens abhängen. In Tabelle 3 sind alle möglichen Abfragen aufgelistet.

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

Tabelle 3: Knotenabfragen

Der Unterschied zwischen `node()` und `*` ist derjenige, dass z. B. `child::*` nicht die Attribute des Kontextknotens auswählen würde, sondern nur Kinder, die Elementknoten sind; `child::node()` würde auch die Attribute auswählen. Um alle Attribute zu erhalten, könnte man z. B. `attribute::*` benutzen.

3.4 Prädikate

Ein Prädikat besteht aus einem Prädikatausdruck und wird in eckigen Klammern [ ] hinter die Knotenabfrage geschrieben. Prädikate dienen zum Filtern einer Trefferliste. Der Wahrheitswert des Prädikats wird folgendermaßen bestimmt :

1. Wenn das Prädikat eine einfache Zahl ist, dann ist das Prädikat genau dann `true`, wenn der Wert der Zahl gleich der Kontextposition des Knotens ist.

2. Ansonsten ist der Wahrheitswert der Wert des boolschen Wertes des Prädikats.

Beispiele :

- `child::kapitel[2]` - wählt das zweite `kapitel`-Element, das ein Kind des Kontextknotens ist.
- `descendant::spielzeug[attribute::farbe = "rot"]`
  - wählt alle Nachfolgerknoten des Kontextknotens, deren Name „spielzeug” ist und deren `farbe`-Attribut den Wert „rot” hat.

Man kann Prädikate auch auf Listen definieren. Zum Beispiel : Gib eine Liste aller Zahlen zwischen 1 und 100 aus, die durch 5 teilbar sind :

```
(1 to 100)[. mod 5 eq 0]
```

3.5 Abkürzende Schreibweise

In vielen Fällen kann man Pfadausdrücke abkürzen, indem man eine abkürzende Schreibweise verwendet. Es gibt folgende Abkürzungsmöglichkeiten :

- `child::` kann ganz weggelassen werden, d. h. `child::` ist die Standardachse. Die Kurzform für `child::foto/child::ort` ist `foto/ort` .
- `attribute::` kann mit `@` abgekürzt werden.
  Der Pfadausdruck `foto[@datum="2002-20-08"]` ist die Abkürzung für `child::foto[attribute::datum="2002-20-08"]`
- `//` ist die Kurzform für `/descendant-or-self::node()/` Die Abkürzung für `/descendant-or-self::node()/child::foto` ist z. B. `//foto` . Es wählt also alle Knoten des Dokuments aus, die ein `foto`-Element sind.
- `.` steht für den Kontextknoten selbst.
  Beispielsweise gibt `//ort[./text()="Alpen"]` alle Orte zurück, deren Text gleich „Alpen” ist.
- `..` ist kurz für `parent::node()` . Beispielsweise gibt `//ort[../@datum > "2002-8-22"]` alle Orte der Fotos zurück, die nach dem 22.8.2002 gemacht wurden.

4 FLWOR

4.1 Grundlagen

Der FLWOR-Ausdruck ist der Teil von XQuery, der SQL am ähnlichsten ist. FLWOR (sprich: flower) steht für `for`, `let`, `where`, `order by`, `return` und beschreibt den typischen Aufbau eines XQuery-Skriptes. Mit FLWOR-Ausdrücken kann man sehr mächtige Anfragen auf XML-Dokumenten formulieren.

Zum Beispiel : Gib aus der Datei `fotoalbum.xml` alle `url`-Elemente der Fotos aus, die nach dem 20.8.2002 gemacht worden sind und die im Beschreibungstext das Wort „Sand" enthalten. Die `url`-Elemente sollen in einem `sandfoto`-Element enthalten und absteigend sortiert sein [2].

```
for $foto in document("fotoalbum.xml")//foto
let $url := $foto/url
where contains($foto/text(),"Sand")
and   xs:date($foto/@datum) gt xs:date("2002-08-20")
order by $url descending
return
<sandfoto>
 {
   $url
 }
</sandfoto>
```

Geht man der Reihe nach durch das Skript, könnte man es folgendermaßen übersetzen: Für jedes `foto`-Element in dem Dokument setze die Variable $url auf das `url`-Element des `foto`-Elementes. Wenn der Text des Fotos das Wort „Sand" enthält und das Foto nach dem 20.8.2002 gemacht wurde, gib ein `sandfoto`-Element zurück, das das `url`-Element enthält und ordne die Ausgabe absteigend nach dem `url`-Element.

Die Abarbeitung eines FLWOR-Ausdruck kann man sich wie folgt vorstellen:

- `for` und `let` erzeugen Variablenbindungen. Man kann sich diese Variablenbindungen als eine Art Relation (oder auch Tupelstrom) vorstellen. `For` erzeugt dabei die Zeilen und `let` hängt weitere Spalten an diese Relation an.

[2] `xs:date()` ist eine, in XQuery eingebaute Funktion, die ein Element vom Typ `xs:date` erzeugt. Der `gt`-Operator vergleicht, dem entsprechend, zwei Datumswerte miteinander. Diese und weitere Konstruktorfunktionen sind in der separaten Spezifikation XQuery functions and operators [6] definiert.

- `where` dient zum Filtern des Tupelstroms, d. h., durch sie kann man Tupel aus dem Strom entfernen.
- `order by` sortiert den Tupelstrom.
- `return` erzeugt für jedes Tupel ein Ergebniselement.

An dem eben gezeigten Beispiel würde die Abarbeitung folgendermaßen aussehen[3]:

1. `for $foto in document("fotoalbum.xml")//foto`

`$foto`
$foto_1$
$foto_2$
$foto_3$
$foto_4$
$foto_5$

2. `let $url := $foto/url`

`$foto`	`$url`
$foto_1$	kairo.jpeg
$foto_2$	aegypten.jpeg
$foto_3$	wueste.jpeg
$foto_4$	berghuette.jpeg
$foto_5$	zugspitze.jpeg

3. `where contains($foto/text(),"Sand")`

`$foto`	`$url`
$foto_2$	aegypten.jpeg
$foto_3$	wueste.jpeg

4. `order by $url descending`

`$foto`	`$url`
$foto_3$	wueste.jpeg
$foto_2$	aegypten.jpeg

5. `return`

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

[3] $foto_i$ steht dabei für das i-te foto-Element. Die umschließenden <url> </url> Tags der $url Variablen wurden der Lesbarkeit halber ebenfalls weggelassen.

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

Um den Unterschied zwischen `for` und `let` noch mehr zu verdeutlichen, hier noch zwei Beispiele :

```
for $s in (<eins/>, <zwei/>, <drei/>)
return <ausgabe>{$s}</ausgabe>
```

gibt als Ergebnis zurück :

```
<ausgabe>
  <eins/>
</ausgabe>
<ausgabe>
  <zwei/>
</ausgabe>
<ausgabe>
  <drei/>
</ausgabe>
```

Das entsprechende Beispiel mit einer `let` Klausel würde folgendermaßen aussehen :

```
let $s := (<eins/>, <zwei/>, <drei/>)
return <ausgabe>{$s}</ausgabe>
```

Das Ergebnis sieht allerdings anders aus :

```
<ausgabe>
  <eins/>
  <zwei/>
  <drei/>
</ausgabe>
```

Da in diesem Beispiel keine `for` Klausel vorkommt, erzeugt die `let` Klausel nur ein Tupel, das die gesamte Liste enthält, an die die Variable $s gebunden ist.

Eine `for` Klausel kann auch mehrere Variablen enthalten, wobei sich jede auf einen entsprechenden Ausdruck bezieht. In diesem Fall ist das Ergebnis ein Tupelstrom, der ein Tupel für jede Wertekombination enthält, die durch das kartesische Produkt der verschiedenen Ausdruckslisten entsteht.

Zum Beispiel erzeugt der folgende Ausdruck :

```
for $s in (1,2), $j in (3,4)
```

den Tupel-Strom :

```
($i = 1, $j = 3)
($i = 1, $j = 4)
($i = 2, $j = 3)
($i = 2, $j = 4)
```

4.2 Joins

Mit Hilfe des FLWOR-Ausdrucks lassen sich auch sehr leicht Joins zwischen verschiedenen Dokumenten erstellen.

Zum 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)
let $begriff := 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>
[...]
```

4.3 Gruppierung

Es gibt in XQuery keine `group by`-Klausel, wie man sie aus SQL kennt. Diese ist auch nicht nötig, da FLWOR-Ausdrücke mächtig genug sind, um Gruppierungsanfragen zu stellen.

Beispiel : Gib alle Fotogruppen mit der Anzahl und Durchschnittsgröße der Fotos in Bytes aus :

```
<fotogruppen> {
  let $doc := document("fotoalbum.xml")
  for $gruppe in $doc/gruppe
  let $fotos := $gruppe/foto
  let $durchschnitt := avg($fotos/kbytes)*1024
  return
  <gruppe name="{$gruppe/text()}">
   <anzahl>{count($fotos)}</anzahl>
   <durchschnitt>{$durchschnitt}</durchschnitt>
  </gruppe>
} </fotogruppen>
```

Ergebnis :

```
<fotogruppen>
  <gruppe name="sommerurlaub">
    <anzahl>3</anzahl>
    <durchschnitt>669013</durchschnitt>
  </gruppe>
  <gruppe name="skiurlaub">
    <anzahl>2</anzahl>
    <durchschnitt>368640</durchschnitt>
  </gruppe>
</fotogruppen>
```

Eine `having`-Klausel ist ebenso wenig vorhanden wie notwendig.

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 eq $ort]
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>
```

4.4 Schachtelungen

Man kann FLWOR-Ausdrücke auch verschachteln um damit beliebige XML-Dokumente aufbauen zu können.

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>
```

5 Zusammenfassung, Bewertung und Ausblick

In diesem Seminar wurde XQuery 1.0 und XPath 2.0 vorgestellt. Es wurde auf die Grundlagen von XQuery eingegangen und die verschiedenen XQuery-Ausdrücke beschrieben. Des Weiteren wurden die in XQuery definierten Datentypen und dort vorallem der Listentyp genauer dargestellt. Ein wichtiger Bestandteil von XQuery ist XPath 2.0, das zum Adressieren von Elementen eines XML-Dokuments benötigt wird und dessen Aufbau genauer erläutert wurde. Abschließend wurde auf die FLWOR-Ausdrücke eingegangen und anhand von mehreren Beispielen erklärt.

Um den Rahmen des Seminars einhalten zu können, wurden in dieser Arbeit einige Aspekte der Sprache weglassen, bzw. nur kurz erwähnt. Vorallem die, in XQuery functions and operators [6] definierten Methoden, bieten viele Möglichkeiten an, um z. B. Strings, Zahlen oder Datumswerte zu manipulieren. Es ist in XQuery auch möglich eigene Datentypen und Funktionen zu definieren. Dadurch ist zum Beispiel auch Rekursion möglich. Um Anfragen leichter parsen zu können, gibt es auch eine XML-Syntax für XQuery, XQueryX [7].

XQuery sieht zur Zeit nur lesende Zugriffe auf XML-Dokumente vor. Ändernde Zugriffe sollen nach dem Abschluss der XQuery 1.0 Spezifikation in Angriff genommen werden. Deshalb wird es in näherer Zukunft weiterhin proprietäre Verfahren geben müssen, um mit XML-Datenbanken arbeiten zu können.

XQuery ist keine Sprache, um Ähnlichkeits- oder wage Anfragen auf XML-Dokumenten zu stellen. Es gibt nur boolsches Retrieval, jedoch keine Gewichtungen oder Rangordnungen.

Da XQuery von vielen anderen Spezifikationen abhängt, die selbst zum Teil noch nicht fertig sind, vollzieht sich der Standardisierungsprozess relativ langsam. Das W3C ist leider noch nicht mit der endgültigen Spezifikation fertig, was das Erscheinen von entsprechenden Werkzeugen und Büchern verzögert. Die endgültige Recommendation soll bis Ende 2003 fertig sein.

XQuery ist jedoch auf dem besten Weg, die Standardsprache für XML-Anfragen zu werden. Sie ist relativ leicht zu erlernen und selbst schwierige Anfragen sind ohne viel Aufwand umsetzbar.

A XML-Dateien

A.1 fotoalbum.xml

```
<?xml version="1.0"?>
<fotoalbum>
  <gruppe name="sommerurlaub">
    <foto datum="20.8.2002">
      <ort>Kairo</ort>
      <beschr>Kairo ist vom Flugzeug aus fotografiert</beschr>
      <url>kairo.jpeg</url>
      <kbytes>500</kbytes>
    </foto>
    <foto datum="22.8.2002">
      <ort>Ägypten</ort>
      <beschr>Man sieht einen weißen Sandstrand</beschr>
      <url>aegypten.jpeg</url>
      <kbytes>640</kbytes>
    </foto>
    <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="26.12.2001">
      <ort>Alpen</ort>
      <beschr>Foto einer langweiligen Berghütte</beschr>
      <url>berghuette.jpeg</url>
      <kbytes>320</kbytes>
    </foto>
    <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>
</fotoalbum>
```

A.2 lexikon.xml

```
<?xml version="1.0"?>
<lexikon>
  <begriff name="Alpen">
    <beschr>Gebirge in Mittel-Europa</beschr>
  </begriff>
  <begriff name="Ägypten">
    <beschr>Land in Nordafrika</beschr>
  </begriff>
  <begriff name="Kairo">
    <beschr>Hauptstadt von Ägypten</beschr>
  </begriff>
  <begriff name="Sahara">
    <beschr>Wüste in Nordafrika</beschr>
  </begriff>
</lexikon>
```

Literatur

[1] BERGLUND, A., S. BOAG, D. CHAMBERLIN, M. F. FERNANDEZ, M. KAY, J. ROBIE und J. SIMEON: *XML Path Language (XPath) 2.0, W3C Working Draft*, November 2002. http://www.w3.org/TR/2002/WD-xpath20-20021115.

[2] BIRON, P. V. und A. MALHOTRA: *XML Schema Part 2: Datatypes, W3C Recommendation*, Mai 2001.
http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

[3] BOAG, S., D. CHAMBERLIN, M. FERNANDEZ, D. FLORESCU, J. ROBIE und J. SIMÉON: *XQuery 1.0: An XML Query Language, W3C Working Draft*, November 2002. http://www.w3.org/TR/2002/WD-xquery-20021115/.

[4] CHASE, NICHOLAS: *Process XML using XML Query*. ibm.com/developerWorks, 2002.

[5] FERNANDEZ, M., A. MALHOTRA, J. MARSH, M. NAGY und N. WALSH: *XQuery 1.0 and XPath 2.0 Data Model, W3C Working Draft*, November 2002.
http://www.w3.org/TR/2002/WD-query-datamodel-20021115/.

[6] MALHOTRA, A., J. MELTON, J. ROBIE und N. WALSH: *XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft*, November 2002.
http://www.w3.org/TR/2002/WD-xquery-operators-20021115/.

[7] MALHOTRA, A., J. ROBIE und M. RYS: *XML Syntax for XQuery 1.0 (XQueryX), W3C Working Draft*, Juni 2001. http://www.w3.org/TR/2001/WD-xqueryx-20010607.