UNIVERSITÄT SIEGEN

Vorlesung Datenbanksysteme II

XQuery

Inhalt

- Einordnung von XQuery
- Grundlegende Merkmale von XQuery
- XQuery-Ausdrücke
- FLWOR-Ausdrücke
- Gruppierung
- Rekursion
- Diskussion

UNIVERSITÄT SIEGEN

Einordnung von XQuery

Entwicklung von XML-Abfragesprachen

Frühe (prototypische) Abfragesprachen wie Quilt, XPath, XQL, XML-QL, … aus sehr unterschiedlichen Anwendungskontexten:

- Link-Generierung für dynamische WWW-Inhalte.
- Suche in schwach/stark strukturierten Datenbeständen.
- Ad-hoc Suche und Information Retrival.
- Komplexe, integrierte Informationssysteme,

Ziele und Gestaltungskriterien von XQuery

Anwendbar auf XML-Daten aus unterschiedlichen Quellen:

- Semi-strukturierte Dokumente.
- Relationale / tabellarische Daten, die z.B. über eine Middleware nach XML konvertiert wurden.
- Daten aus objektorientierten Datenbanken.
- Beliebigen Mischformen der vorigen Punkte.

Ziele und Gestaltungskriterien von XQuery

Syntaktische Schreibvarianten von Abfragen / Transformationen:

- Als XML-Element.
- In lesbarer Form, angelehnt an SQL.

Ziele und Gestaltungskriterien von XQuery

Integration mit anderen XML-Technologien:

- Ausgaben wieder als reine XML-Dateien.

Details: siehe XML Query Requirements. W3C Working Draft, Jan. 31, 2000; http://www.w3.org/TR/xmlquery-req

Versionen von XQuery

- XPath 1.0 war (mit leichten Modifikationen) Teilmenge von XQuery.
- Diverse Erweiterungen in XPath 2.0 waren von XQuery inspiriert.
- XPath 2.0 ist echte (mit minimalen Abweichungen) Teilmenge von XQuery, d.h. jeder XPath-2.0-Ausdruck ist auch eine korrekte XQuery-Abfrage.

Entwicklungsstand von XQuery

- XPath 2.0 und XQuery 1.0 haben einheitliches Datenbankmodell (Syntaxbaum).
- XQuery umfasst „halbe“ Programmiersprache mit benutzerdefinierten Datentypen (XML Schema), Funktionen usw.
- XQuery mit umfangreicher Spezifikationen, u.a. Semantikspezifikation durch XML Query Algebra.

UNIVERSITÄT SIEGEN

Grundlegende Merkmale von XQuery

Ausführungsmodell von XQuery

Eingabe:

- Menge “abstrakter” (Eingabe-)Syntaxbäume (XML-Dokumente bzw. Dokumentteile).
- XQuery-Ausdruck.

Ausdruck:

- Menge “abstrakter” (Ausgabe-)Syntaxbäume (XML-Dokumente bzw. Dokumentteile).

Ausführungsmodell von XQuery

Bearbeitungsphasen eines Ausdrucks:

1. Analyse des Ausdrucks: Bearbeitung von Namensräumen, Schemadefinitionen, Variablendefinitionen, Funktionen u.a.
2. Auswertung des Ausdrucks auf dem Eingabebaum und Erzeugung des Ausgabebaums.

Funktionale Sprache von XQuery

Funktionaler Kern, analog zur relationalen Algebra:

- Schachtelbare Ausdrücke.
- (Fast) keine Seiteneffekte.
- Streng typisiert.
- **Keine** “nicht-imperativen” Konstrukte wie etwa die Transformationsregeln von XSLT.

Funktionale Sprache von XQuery

Insgesamt über 10 verschiedene Arten von Ausdrücken (wir betrachten nur die Wichtigsten):

- Path expressions.
- Element constructors.
- FLWOR expressions.
- ....

UNIVERSITÄT SIEGEN

XQuery-Ausdrücke

Primary Expressions

PrimaryExpr ::= Literal | VarRef |
ParenthesizedExpr |
ContextItemExpr |
FunctionCall | Constructor |
OrderedExpr | UnorderedExpr

Primary Expressions

- „Literal“: Konstante Zahlen, Zeichenketten usw. (sehr viele Details und Varianten).
- „VarRef“: Variablenreferenzierungen (siehe später).
- „ParenthesizedExpr“: Klammerung wie üblich.
- „ContextItemExpr“: entspricht ’.’
- „Constructor“: Generierung von XML-Elementen (siehe später).

Kommentare

Comment ::= "(:" (CommentContent | Comment)* ":)"

CommentContent ::= (Char+ - (Char* ('(:' | ':)') Char*))

Beispiel:

```
(: dies ist ein XQuery-Kommentar,
der nicht ausgegeben wird :)
<!-- dies erzeugt einen Kommentar-
Knoten im XML-Ausgabebaum -->
```

Sequenzen

- Sequenzen sind geordnete Mengen von atomaren Werten oder Knoten.
- Keine Schachtelung (d.h. innere Sequenz als Element einer äußeren Sequenz) möglich.
- Kein Unterschied zwischen Sequenz mit einem Element und dem enthaltenen Element.

Entstehung von Sequenzen

Sequenzen entstehen:

- “Von Hand” durch Aufbau mittels Literalen. Syntax wie in funktionalen Sprachen üblich:
  - außen „(...)“
  - enthaltene Elemente durch „,“ getrennt auflisten.
- Als Ergebnis der Auswertung von Pfaden.
- Durch die übliche Mengenoperationen.

Eingabefunktionen

Eingabefunktionen für die Verbindung zu Eingabedaten (einzelne Dokumente, Kollektionen).

Beispiel: Funktion „doc“

„fn:doc(’kunden.xml’)“

„fn:doc(’http://xyz.org/xmldb’)“

Eingabefunktionen

Erläuterungen zur Funktion „doc“:

- Namensraumbezeichner: „fn“.
- Argument: URI.
- Bedeutung: Liefert als Kontextknoten **den Wurzelknoten** des Syntaxbaums desjenigen Dokuments, das unter der angegebenen Quelle zu finden ist.

Eingabefunktionen

Beispiel: Funktion „collection“

„fn:collection(’http://xyz.org/xmldb’)/kunde“

- Argument: URI.
- Bedeutung: liefert als Kontextknoten **alle Knoten** des Syntaxbaums desjenigen Dokuments, das unter der angegebenen Quelle zu finden ist.

Variablen

VarRef ::= ’$’ VarName

VarName ::= QName

- Variablennamen beginnen mit „$“.
- Variablenwert kann eine **Sequenz** von Knoten sein.

Wertzuweisungen und Ausgabe von Variablen

- Wertzuweisung erfolgt durch LET-Klauseln.
- Ausgabe der Knotensequenz einer Variablen durch: { $variablenname }

(Details zu {...} siehe später)

Wertzuweisungen und Ausgabe von Variablen

Beispiel:

```
let $s := (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

Ergebnis ist ein Element „out“ mit der textuellen Darstellung:

```
<out><one/><two/><three/></out>
```

Pfadausdrücke

Fast alle Konstrukte von XPath (2.0):

- Relative und absolute Pfadausdrücke.
- Beliebig viele Schritte.
- Pro Schritt: Navigationsrichtung (axis), Knotentest und weitere Selektionen.

Pfadausdrücke

Ausnahmen:

- Richtung „namespace“ nicht unterstützt.
- Richtungen „ancestor“, „ancestor-or-self“, „following“, „following-sibling“, „preceding“, „preceding-sibling“ nur optional unterstützt.

(Hintergrund: Bei diesen Navigationsrichtungen muss der Eingabebaum komplett gepuffert werden (dadurch hoher Speicherplatzbedarf).)

Konstruktoren

XQuery definiert mehrere mögliche Arten von Konstruktoren, mit denen letztlich alle Arten von Knoten des XML-Ausgabebaums erzeugt werden können.

Direkte Konstruktoren

- “Direkte” Angabe der auszugebenden Elemente, Textknoten, CDATA-Sektionen, XML-Processing Instructions, XML-Kommentare, …
- Sehen aus wie das jeweilige XML-Element.
- Sind eine Anweisung, diesen “Text” zu erzeugen und die inneren Ausdrücke auszuwerten (wie bei XSLT).

Direkte Konstruktoren

Einfaches Beispiel ohne innere Ausdrücke:

```
<Adresse>
   <Name>Meier</Name>
   <Vorname>Hans</Vorname>
   <Strasse>Hauptstr.</Strasse>
   <Hausnummer>5</Hausnummer>
   <PLZ>57076</PLZ>
   <Ort>Siegen</Ort>
</Adresse>
```

Direkte Konstruktoren mit inneren Ausdrücken

Innere Ausdrücke werden durch geschweifte Klammern markiert.

Direkte Konstruktoren mit inneren Ausdrücken

Beispiel:

```
let $n = 'Meier'
let $v = 'Hans' ...
return ...
   <Adresse>
      <Name>{$n}</Name> <Vorname>{$v}</Vorname>
      <Strasse>Hauptstr.</Strasse>
      <Hausnummer>5</Hausnummer>
      <PLZ>57076</PLZ> <Ort>Siegen</Ort>
   </Adresse>
```

UNIVERSITÄT SIEGEN

Berechnete Konstruktoren

Aufbau von berechneten Konstruktoren:

1. Schlüsselwort, das den Typ des zu erzeugenden Knotens angibt (alle 7 XML-Knotentypen): „document“, „element“, „attribute“, „text“, „comment“, „namespace“, „processing-instruction“.
2. Element- / Attributname (optional).
3. Inhalt des Knotens in „{..}“.

(„namespace“-Knoten zuerst, danach Attribute)

Berechnete Konstruktoren

Beispiel:

```
element Adresse {
    attribute ADrNr {'134'}
    element Name {'Meier'}
    element Vorname {'Hans'}
    element Strasse {'Hauptstr.'}
    element Hausnummer {'5'}
    element PLZ {'57076'}
    element Ort {'Siegen'}
}
```

UNIVERSITÄT SIEGEN

FLWOR-Ausdrücke

Syntax von FLWOR-Ausdrücken

FLWORExpr ::= (ForClause | LetClause)+
WhereClause?
OrderByClause?
'return' ExprSingle

Benannt nach den darin auftretenden Schlüsselwörtern/Klauseln:
„**f**or“, „**l**et“, „**w**here”, “**o**rder by”, “**r**eturn”

FLWOR-Ausdrücke

Beispiel (ohne „order by“):

```
for     $adr in fn:doc("adressen.xml") //Adresse
let     $n = $adr/Name
where   $adr/Ort = 'Siegen'
return
   <Adresse>
      <Name>{$n}</Name>
      <Vorname>{$adr/Vorname}</Vorname>
   </Adresse>
```

Bedeutung von FLWOR-Ausdrücken

for-Klausel (Schleife):

- Iteriert durch die Sequenz, die sich aus dem Ausdruck hinter dem „in“ ergibt.
- Rumpf der for-Schleife ist Rest der Abfrage (mehrere for-Schleifen sind geschachtelt und entsprechen Kreuzprodukt).
- Produziert “Tupel”-Sequenz, in der jeder Tupel je einen Wert von jeder Sequenz erhält, über die iteriert wird.

Bedeutung von FLWOR-Ausdrücken

let-Klausel:

- Führt genau eine (evtl. mengenwertige) Wertzuweisung pro Durchlauf der äußeren Schleife(n) durch.

Bedeutung von FLWOR-Ausdrücken

where-Klausel:

- Enthält Boole‘schen Ausdruck, der Variablen enthalten kann.
- Falls Auswertung negativ, wird aktuelles Tupel nicht weiter bearbeitet.

Bedeutung von FLWOR-Ausdrücken

order-by-Klausel:

- Gibt Sortierung an (optional).
- Weitere Optionen:
  - „stable“: bei gleichen Sortierwerten bleibt Eingabereihenfolge erhalten.
  - „descending“: fallende Sortierung.

Bedeutung von FLWOR-Ausdrücken

Beispiel:

```
for      $adr in fn:doc("adressen.xml") //Adresse
order by $adr/PLZ
return   <Adresse>
           <Name>{$adr/Name}</Name>
           <Vorname>{$adr/Vorname}</Vorname>
         </Adresse>
```

Bedeutung von FLWOR-Ausdrücken

return-Klausel:

- Legt die Ausgaben fest.
- Kann beliebig komplexer Ausdruck sein, der ein Ausgabeelement generiert.
- Insbesondere sind auch innere FLWOR-Ausdrücke erlaubt.

FLWOR-Ausdrücke

Beispiel: LR-Ausdruck.

```
let $s := (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

Ausgabe:

```
<out><one/><two/><three/></out>
```

FLWOR-Ausdrücke

Beispiel: FR-Ausdruck.

```
for $s in (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

Ausgabe:

```
<out><one/></out>
<out><two/></out>
<out><three/></out>
```

FLWOR-Ausdrücke

Beispiel: 2*F-Ausdruck.

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

Oder:

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

FLWOR-Ausdrücke

Ausgabe:

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

Verbunde

- Keine kompakte Notation wie NATURAL JOIN (wo die Verbundattribute automatisch bestimmt werden) vorhanden.
- Verbunde können durch geschachtelte „for“-Schleifen und entsprechende „where“-Bedingungen nachgebildet werden.

Verbunde

Beispielaufgabe:

- Gegeben: Die Dateien „kunden.xml“ und „lieferungen.xml“ mit den üblichen Attributen.
- Gesucht: Name des Kunden und Datum der Lieferung für alle Lieferungen von Kunden aus Bonn.

Verbunde

Lösung:

```
for     $l in fn:doc('lieferungen.xml')//lieferung
for     $k in fn:doc('kunden.xml')//kunde
where   $l/@KdRef = $k/@KdId
   and  $k/Ort = 'Bonn'
return <Lfg> attribute KdName { $k/Name }
             attribute LDatum { $l/Datum }
       </Lfg>
```

Gruppierung

Gruppierung

- Auch hier keine kompakte Notation wie „GROUP BY“ vorhanden.
- Gruppierungen können ebenfalls durch passend geschachtelte Schleifen nachgebildet werden.

Gruppierung

Beispielaufgabe: Gib die Adressliste nach Städten gruppiert aus und pro Stadt die Namen der dort wohnenden Personen.

Gewünschtes Ergebnis:

```
<Staedteliste>
   <Ortsliste>
      <Ortsname>Bonn</Ortsname>
      <Adresse><Name>...</Name>...</Adresse>
      <Adresse><Name>...</Name>...</Adresse> ...
   </Ortsliste>
   <Ortsliste> ... </Ortsliste>
</Staedteliste>
```

Gruppierung

Lösung:

```
<Staedteliste>
   {for $o in fn:distinct-values
                        ( fn:doc("adressen.xml") // Ort )
   order by $o
   return
      <Ortsliste>... s.u. ...
      </Ortsliste>
   }
</Staedteliste>
```

Gruppierung

… Inhalt der „return“-Klausel:

```
<Ortsliste>
   <Ortsname>{$o/text()}</Ortsname>
   { for $adr in
            fn:doc("adressen.xml")//adresse[Ort=$o]
     order by $adr/Name
     return <Adresse>
        <Name>{$adr/Name}</Name>
        <Vorname>{$adr/Vorname}</Vorname>
     </Adresse>
   }
</Ortsliste>
```

Gruppierung

Allgemeines Vorgehen:

- Äußerste Funktion: ein direkter Konstruktor, erzeugt Wurzelelement der Ausgabeliste, passenden Wurzelelementtyp wählen.
- Inhalt des Wurzelelements: durch äußere Schleife generieren, diese iteriert über die Gruppen.
  Hierzu Bildung einer Liste unterschiedlicher Werte mit der Funktion „fn:distinct-values()“.

Gruppierung

- Bei mehreren Gruppierungsfeldern: pro Feld eine Schleife.
- Im Rumpf der äußeren Schleife (also innerhalb von deren „return“-Klausel): eingeschachtelter FLWOR-Ausdruck bestimmt pro Gruppierungswert die Mitglieder der dieser Gruppe.

  (Beispiel: siehe nächste Folie.)

Gruppierung

- Beispiel:
  „fn:doc("adressen.xml")//adresse[Ort = $o]“
  Suche nach allen Adressen, bei denen der Ort den gerade betrachteten Ortsnamen hat.
- Diese Sequenz kann einer Variablen zugewiesen werden oder direkt in einer inneren for-Schleife berechnet werden.

Gruppierung

- Für jeden Gruppierungswert wird der komplette Datenbestand erneut nach zugehörigen Elementen durchsucht.
- Sehr flexibel, aber ineffizient (lineare Suche).

Aggregation

Beispielaufgabe: Bestimme Orte und Zahl der zugehörigen Adressen für Orte mit mindestens 10 zugehörigen Adressen.

Aggregation

Lösung:

```
<Staedteliste> {
    for $o in fn:distinct-values
            ( fn:doc("adressen.xml") // Ort )
    let $nl = fn:doc("adressen.xml") //
            adresse [ Ort=$o ] / Name
    where count($nl) >= 10
    order by $o
    return
    <Ortsliste>
        <Ort>{$o/text()}</Ort>
        <Adressenzahl>{count($nl)}</Adressenzahl>
    </Ortsliste>
} </Staedteliste>
```

Aggregation

- Die Funktionen „count“, „avg“ usw. arbeiten auf Sequenzen.
- „let $nl = ...“ weist „$nl“Liste von Namen zu.
- Achtung: Die Bedingung „count($nl) >= 10“ bezieht sich auf einen Aggregationswert.
- Unterschied zu SQL: dort stehen solche Bedingungen in der „having“-Klausel, nicht in der „where“–Klausel.

Aggregation mit gemischten Bedingungen

- Bei Aggregationen mit gemischten Bedingungen stehen in SQL Bedingungen in der „having“ **und** in der „where“-Klausel.
- Können in XQuery meist nachgebildet werden, indem die Bedingung an die Basisknoten **bei der Bestimmung der Elemente einer Gruppe** hinzugenommen wird (evtl. durch Schachteln von FLWOR-Ausrücken).

Aggregation mit gemischten Bedingungen

Beispielaufgabe: Wie zuvor, aber nur für Adressen, bei denen Hausnummer = 5 ist.

Aggregation mit gemischten Bedingungen

Lösung:

```
<Staedteliste> {
    for $o in fn:distinct-values
            ( fn:doc("adressen.xml") // Ort )
    let $nl = fn:doc("adressen.xml") //
            adresse[ Ort=$o and Hausnummer=5 ] / Name
    where .....
} </Staedteliste>
```

UNIVERSITÄT SIEGEN

Rekursion

Motivation

Beobachtungen zu den bisherige Ausgabemöglichkeiten und Breite / Tiefe des erzeugten Ausgabebaums:

- „for“-Schleifen können den “Abfrage”-Baum nur breiter machen.
- Tiefe des erzeugten Ausgabebaums entspricht Schachtelungstiefe der Abfrage (d.h. statisch festgelegt).

Motivation

Konsequenz: mit bisherigen Mitteln kann zum Beispiel **keine identische** (d.h. die Eingabe reproduzierende) **Transformation** beschrieben werden.

Rekursion

- Lösung: Benutzerdefinierte Funktionen, die sich rekursiv aufrufen.
- Nachteil: Sehr umständlich und fehlerträchtig.
- Erzeugung beliebig verschachtelter Ausgabebäume abhängig vom Eingabebaum (z.B. bei “Projektionen”) ist eine prinzipielle Schwäche von XQuery.
- XSLT ist hierzu viel besser geeignet.

UNIVERSITÄT SIEGEN

Diskussion

Perspektiven von XQuery

XML-DBMS/XQuery als primäres DVS statt RDBMS?

- Bietet in bestimmten Anwendungsfällen große Vorteile, aber weniger bei klassischen betrieblichen Anwendungen.
- Umfangreicheres Datenbankmodell führt zu komplizieren und komplexeren Operationen.
- Unklar, ob Implementierungen auf Großanwendungen skaliert werden können.
- *Tendenz:* Bleibt vermutlich „Speziallösung“.

Perspektiven von XQuery

XML-DBMS/XQuery als Middleware-Lösung?

- Einsatz bei der “losen Integration” existierender Altanwendungen, die in Schnittstellen auf Basis von XML eingekapselt worden sind.
- *Tendenz*: Beschränkung auf Integrations-Szenarien mit transienten Daten und kleinen Datenvolumina.

Perspektiven von XQuery

XQuery oder XSLT/XPath?

- XQuery wurde ursprünglich als Ablösung von XSLT 1.0 / XPath 1.0 angesehen.
- Aber: XSLT / XPath haben in Version 2.0 sehr stark aufgeholt, sodass Technologien inzwischen mehr Gemeinsamkeiten als früher haben.
- *Tendenz*: kein klarer „Sieger“, werden vermutlich noch länger koexistieren.

Prüfungsstoff

- Ausgewählte Grundkonzepte von XQuery erklären.
- XQuery auf einfache Beispiele anwenden können (Verbunde, Aggregationen, Sortierungen).
- XQuery vergleichen können mit XPath und SQL.