



XPath und XQuery

Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
tolk@inf.fu-berlin.de



XML Path Language (XPath)

- Standard zum Zugreifen auf beliebige Teile eines XML-Dokuments
- keine XML-Anwendung
- wird von XSLT und XQuery benutzt
- Adressierungspfad eines Dateisystems ähnlich aber wesentlich mächtiger:
z.B. /order/item
- XPath 1.0 – W3C-Recommendation seit Nov. 1999
 - <http://www.w3.org/TR/xpath>
- XPath 2.0 – W3C-Recommendation seit Jan. 2007
 - <http://www.w3.org/TR/xpath20/>



- ähnliches Modell wie in DOM
 - XML-Dokument als Baum mit Elementen, Attributen und PCDATA als Knoten
- virtuelle Dokument-Wurzel (Wurzelknoten):
durch "/" repräsentiert (links von "/" steht nichts)
 - ⇒ Wurzel-Element immer Kind von "/":
z.B. /root

Knotentypen (I)

- **Wurzelknoten**

- oberster Knoten im Baum, dessen Kind der Elementknoten des Dokuments ist
- string-Wert: Verkettung der Zeichendaten aller Textknoten-Kinder in der Dokumentenreihenfolge

- **Elementknoten**

- Knoten für ein Element
- string-Wert: Verkettung der Zeichendaten aller Textknoten-Kinder des Elements

- **Attributknoten**

- Knoten für jedes Element zugeordnete Attribut
- string-Wert: Normalisierter Attributwert

- **Textknoten**

- Knoten der Zeichendaten enthält
- string-Wert: die Zeichendaten des Textknotens

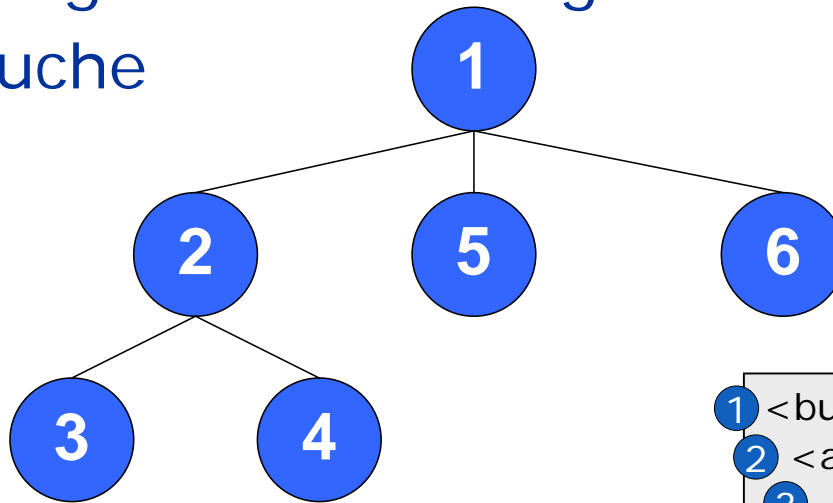
- **Namensraumknoten**

- der Namensraum ist jeweils einem Elementknoten als Elternknoten zugeordnet, ist aber nicht Kind dieses Elementknoten
- string-Wert: URI des Namensraum

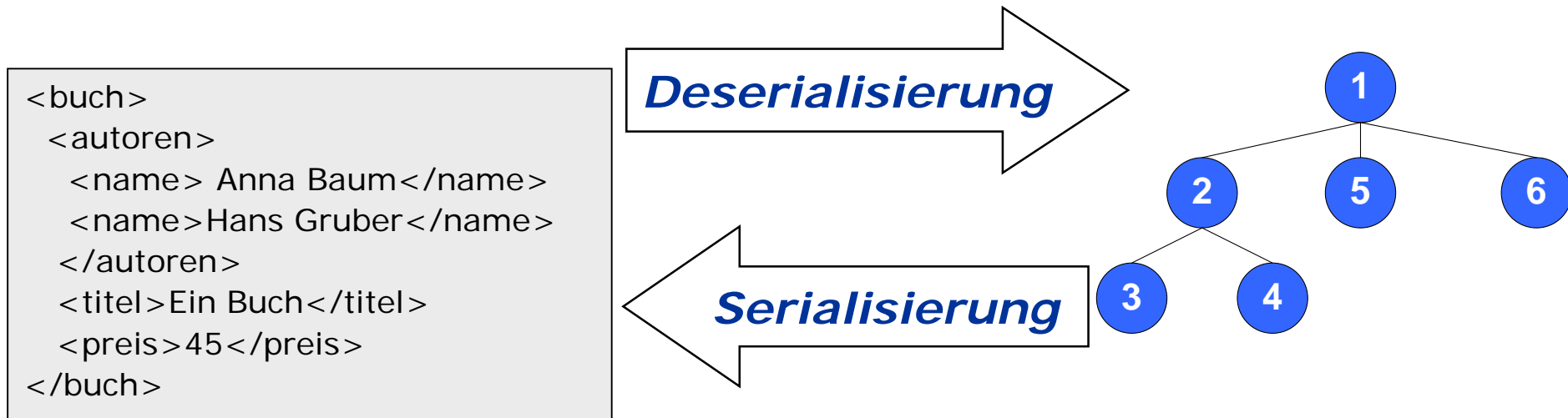
- **Kommentarknoten**

- Knoten für jeden einzelnen Kommentar
- string-Wert: Kommentarinhalt

- Baummodell als Basis
- feste Dokumentreihenfolge (document order) = Reihenfolge der Start-Tags im Dokument
- Tiefensuche



```
1 <buch>  
2 <autoren>  
3 <name> Anna Baum</name>  
4 <name> Hans Gruber</name>  
  </autoren>  
5 <titel> Ein Buch</titel>  
6 <preis> 45</preis>  
  </buch>
```



- **Deserialisierung** – Erzeugung eines Baums aus einem Dokument
- **Serialisierung** – Erzeugung eines Dokuments aus einem Baum

- Elemente werden einfach **über ihren Namen** identifiziert:
z.B. **order** oder **order/item**
- Attribute werden mit "**@name**" identifiziert:
z.B. **@id** oder **order/@id**

```
<?xml version="..." encoding="..."?>
<order id="O56">
  <item item-id="E16-2">
    <name>buch</name>
  </item>
</order>
```

- aktueller Knoten
- .. Eltern-Knoten
- * beliebiges Kind-Element
- @* beliebiges Attribut
- // überspringt ≥ 0 Hierarchie-Ebenen nach unten
- [] spezifiziert ein Element
- | Auswahl (Vereinigung)

Beispiel: * | @*

„Kind-Element oder Attribut des aktuellen Knotens“

Absolute und relative Pfade

- absolute Pfade

- beginnen mit /
z.B. /order/item

lesen: (➔) Folge dem Pfad von der Dokument-Wurzel zu einem Kind-Element order und von dort aus zu einem Kind-Elementen item!

- relative Pfade

- beginnen mit einem Element oder Attribut
z.B. order/item

lesen: (➚) item-Elemente, die Kind eines Elementes order sind

- Element order kann an beliebiger Stelle des XML-Dokumentes stehen

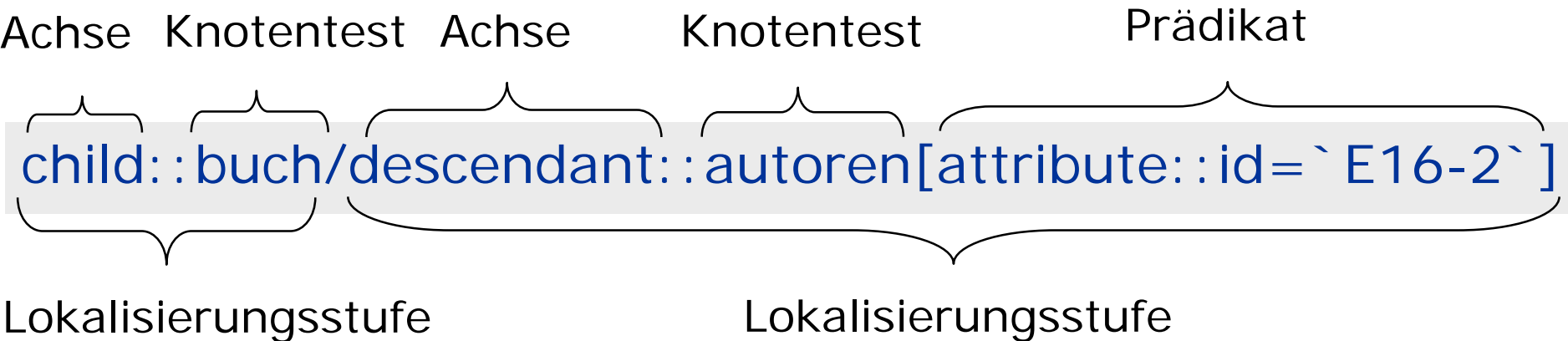
- XPath-Pfade werden in XSLT immer bzgl. eines bestimmten **Kontext-Knotens** ausgewertet:
Element-, Attribut- oder Text-Knoten

- Beispiel:

```
<xsl:template match="p" >
  <DIV>
    <xsl:value-of select="."/ >
  </DIV>
</xsl:template>
```

- Was bedeutet hier aktueller Knoten "." ?
- "." = Kontext-Knoten
- Kontext-Knoten = Knoten, auf den das Template angewandt wird (hier ein p-Element)

```
<?xml version="..." encoding="..."?>
<buch>
  <autoren item-id="E16-2">
    <name> Anna Baum</name>
    <name>Hans Gruber</name>
  </autoren>
  <titel>Ein Buch</titel>
  <preis>45</preis>
</buch>
```



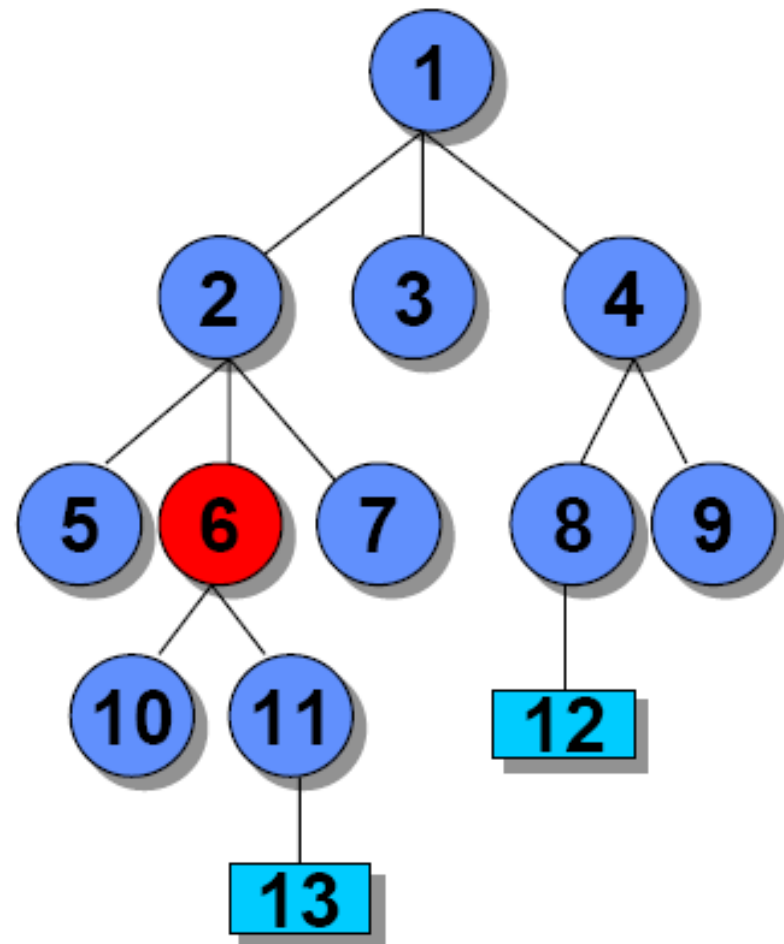
Lokalisierungsstufe



- besteht aus:
 - einem Achsenbezeichner
 - einem Knotentest
 - einem oder mehreren Prädikat (optional)
- ...in der Form:

Achsenbezeichner::Knotentest[Prädikat1][Prädikat2]

- :: Trennzeichen zwischen Achsenbezeichner und Knotentest



Quelle: <http://swt.cs.tu-berlin.de/informatik2000/skripte/xml-datenbank.pdf>

- self:: : 6
- child:: : 10, 11
- parent:: : 2
- descendant:: : 10, 11, 13
- descendant-or-self:: : 6, 10, 11, 13
- ancestor:: : 2, 1
- ancestor-or-self:: : 6, 2, 1
- preceding-sibling:: : 5
- preceding:: : 5, 2, 1
- following-sibling:: : 7
- following:: : 10, 11, 13, 7, 3, 4, 8, 12, 9

- attribute:: (Kürzel @)
- namespace::



- Filterung der Knotenmenge
- Filterungs-Kriterium:
 - Knotenname
z.B.: `child::buch`
 - Knotentyp
z.B.: `child::text()`
`child::node()`



- Verfeinerung der Filterung durch Prädikate
- Anzahl der Prädikate ≥ 0
- [] Bedingung
- Prädikatausdruck unterstützen
 - logische Operatoren: $<$, $>$, \leq , \geq , $=$, \neq
 $>$, $<$ müssen als Entity-Referenzen $\>$ und $\<$ benutzt werden
 - numerische Operatoren: $+$, $-$, $*$, div , mod

- `order/item[@item-id = 'E16-2']`
 - item-Elemente, die Kind von order sind und Attribut item-id mit Wert 'E16-2' haben

```
<order id="4711">  
  <item item-id="E16-2">  
    <name>buch</name>  
  </item>  
</order>
```

- Randbedingungen können an beliebiger Stelle in einem Pfad vorkommen:

- `order[@order-id = '4711']/item`

```
<orders>  
  <order id="4711">  
    <item item-id="E16-2">  
      <name>buch</name>  
    </item>  
  </order>  
  <order id="4711">  
    <item item-id="E16-3"/>  
  </order>  
</orders>
```



XPath Funktionen

Grundlegende Datentypen

- node-set – eine ungeordnete Knotenmenge
- string – Zeichenfolge
- boolean – true und false
- number – Fließkommazahl

- Funktionen:

- *number* last() eine Zahl, die die Größe der aktuellen Knotenmenge entspricht
- *number* position() Position eines Knotens
- *number* count(node-set) Anzahl der Knoten in der Knotenmenge

- Beispiele:

order/item[position() = 1]

order/item[position() = last()]

- **Funktionen:**
 - *string* string(object)
interpretiert ein übergebenes Argument als Zeichenkette
 - *string* string-length(string)
Länge vom String (Anzahl der Zeichen)
 - *boolean* starts-with(string, string)
true wenn die erste Zeichenkette mit der zweiten Zeichenkette anfängt

- Funktionen:

- *boolean* boolean(object)
Objekt ist +0,-0,NaN, {}, "" => **false** , sonst **true**

- *boolean* not(boolean)
Negation

- *boolean* true() – immer true
- *boolean* false() – immer false

- Beispiel:

order/item[not(position()=last())]

- **Funktionen:**
 - *number* number(object)
interpretiert Zeichenkette als Zahl
 - *number* sum(node-set)
Gesamtsumme der Zahlenwerte der Knotenmenge
(nach Umwandlung)
 - *number* round(number)
rundet den Wert zur nächsten Ganzzahl
- **Beispiel:**
number(3xy) → 3

Wähle das
Wurzelement AAA
aus:

```
<AAA>
  <BBB/>
  <CCC/>
  <BBB/>
  <BBB/>
  <DDD>
    <BBB/>
  </DDD>
  <CCC/>
</AAA>
```

/AAA

Wähle alle CCC Elemente
aus, die Kinder des
Elements AAA sind:

```
<AAA>
  <BBB/>
  <CCC/>
  <BBB/>
  <BBB/>
  <DDD>
    <BBB/>
  </DDD>
  <CCC/>
</AAA>
```

/AAA/CCC

//BBB

```
<AAA>  
  <BBB/>  
  <CCC/>  
  <BBB/>  
  <DDD>  
    <BBB/>  
  </DDD>  
  <CCC>  
    <DDD>  
      <BBB/>  
      <BBB/>  
    </DDD>  
  </CCC>  
</AAA>
```

//DDD/BBB

```
<AAA>  
  <BBB/>  
  <CCC/>  
  <BBB/>  
  <DDD>  
    <BBB/>  
  </DDD>  
  <CCC>  
    <DDD>  
      <BBB/>  
      <BBB/>  
    </DDD>  
  </CCC>  
</AAA>
```

Beispiele

`/* /* /* /BBB`

```
<AAA>
  <XXX>
    <DDD>
      <BBB/>
      <FFF/>
    </DDD>
  </XXX>
<CCC>
  <BBB>
    <BBB>
      <BBB/>
    </BBB>
  </BBB>
</CCC>
</AAA>
```

`//*`

```
<AAA>
  <XXX>
    <DDD>
      <BBB/>
      <FFF/>
    </DDD>
  </XXX>
<CCC>
  <BBB>
    <BBB>
      <BBB/>
    </BBB>
  </BBB>
</CCC>
</AAA>
```

`/AAA/BBB[last()]`

```
<AAA>  
  <BBB/>  
  <BBB/>  
  <BBB/>  
  <BBB/>  
</AAA>
```

`//@id`

```
<AAA>  
  <BBB id="b1"/>  
  <BBB id="b2"/>  
  <BBB  
    name="bbb"/>  
  <BBB/>  
</AAA>
```

//CCC | //BBB

```
<AAA>
  <BBB/>
  <CCC/>
  <DDD>
    <CCC/>
  </DDD>
  <EEE/>
</AAA>
```

//CCC/following-sibling::*

```
<AAA>
  <BBB>
    <CCC/>
    <DDD/>
  </BBB>
  <XXX>
    <EEE/>
    <CCC/>
    <FFF/>
    <FFF>
      <GGG/>
    </FFF>
  </XXX>
</AAA>
```

⇒ <http://ponderer.org/download/xpath/> und andere

- Januar 2007 – W3C Recommendation für neue Version von XPath
- zeitgleich mit XQuery 1.0 & XSLT 2.0
 - erweitertes Datenmodell
 - neue Konstrukte für Ausdrücke
 - neue Datentypen
 - neue Operatoren
 - erweiterte Funktionsbibliothek

- Berücksichtigung einzelner Werte (**atomic values**)
- Daten unterschiedlichen Typs:
 - Zeichenfolgen, Zahlen, logische Werte, Datums- und Zeitwerte
 - qualifizierte Namen & URIs
 - einfache Sequenzen & Listen
- Ergebnis eines XPath Ausdrucks: Auswahl von Knoten, Einzelwert oder Sequenz
- XPath 2.0 auf Knotenbaum
 - nur Daten auslesen
- XPath 2.0 auf Einzelwerten & Sequenzen
 - neue Werte/Sequenzen erzeugen

Konstrukte für Ausdrücke

- für Operationen mit Sequenzen Verwendung des **for**-Ausdrucks

- Beispiel

```
for $i in 1 to 3 return $i*$i
```

Variable
(Bereichsvariable)

binding sequenz

return-Ausdruck

Ergebnis: 1, 4, 9

- bedingter Ausdruck *if*

```
if @menge > 1000 then "gut" else "weniger gut"
```

- quantifizierende Ausdrücke *some* und *every*

```
some $a in $lager/artikel satisfies $lager/artikel/menge=0
```

- wahr, wenn die Menge mind. bei einem Artikel = 0

```
every $a in $lager/artikel satisfies $lager/artikel/menge>0
```

- wahr, wenn von allen Artikeln mind. einer vorhanden ist.

- Unterstützung der XML-Schema Datentypen
- XPath 1.0
 - number – Fließkommazahl
- XPath 2.0
 - integer
 - decimals
 - single precision
 - Datums-, Zeit- und Dauerwerte

- Knotenvergleiche

- **is** – prüft, ob zwei Ausdrücke den selben Knoten liefern
- **<<, >>** – prüfen, welcher von zwei Knoten in der Dokumentreihenfolge früher oder später erscheint

- Kombination von Knotensequenzen

- **union** – Vereinigung zwei Knotensequenzen zu einer Sequenz
- **intersect** – erzeugt aus zwei Sequenzen eine Sequenz, die Knoten enthält, die in beiden vorkommen
- **except** – erzeugt aus zwei Sequenzen eine Sequenz, die Knoten enthält, die in der ersten Sequenz aber nicht in der zweiten vorkommen

- Xpath, Xquery und XSLT Ausdrücke können Funktionen verwenden
- Vereinheitlicht in Standard
 - XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition)
 - W3C Recommendation 14 December 2010
 - <http://www.w3.org/TR/xpath-functions/>

- Zugriffe
- Numerische Funktionen
- Zeichenketten
- Zeit, Datum
- Knoten
- Listen
- Kardinalitäten
- Statistik
- Verarbeitungskontext

- http://www.w3schools.com/xpath/xpath_functions.asp
gute Darstellung



XQuery

Was ist XQuery?

XQuery ...

- ist **die Abfragesprache** für XML-Daten
 - XML-Dateien &
 - alles was in XML darstellbar ist (auch DBs)
- ist für XML das, was SQL für Datenbanken
- basiert auf **XPath-Ausdrücken**
- ist sein Januar 2007 eine **W3C Recommendation**
→ <http://www.w3.org/TR/xquery/>

XQuery kann benutzt werden um ...

- Informationen zu extrahieren, um sie in Web Services zu nutzen
- in Web-Dokumenten nach relevanten Informationen zu suchen
- XML in andere Sprachen zu transformieren
- Reports zu generieren

- XPath Ausdruck ist eine Anfrage an ein XML-Dokument

```
doc('test')//student
```

- Ergebnis der Anfrage: Alle <student>-Knoten

- Komplexe Anfrageausdrücke als FLWOR („Flower“)
 - For Auswahl von Elementen
 - Let Wertzuweisung an Variablen
 - Where Filtern von Elementen
 - Order by Ergebnisordnung
 - Return Ergebnisrückgabe

```
for $d in fn:doc("depts.xml")/depts/deptno  
let $e := fn:doc("emps.xml")/emps/emp[deptno=$d]  
where fn:count($e) >= 10  
order by fn:avg($e/salary) descending  
return
```

```
<big-dept>  
  { $d,  
    <headcount> { fn:count($e) } </headcount> ,  
    <avgsal> { fn:avg($e/salary) } </avgsal>    }  
</big-dept>
```

- For erzeugt einen Strom aus Objektupeln
- Dieser wird iterativ verarbeitet

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

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

- Strom aus zu betrachtenden Knoten mit XPath Ausdruck

```
for $s in doc('test')//student  
return $s/name/text()
```

- Ergibt

Joe AverageJack Doe

- doc(URI) betrachtet das Document bei URI

```
<students>  
  <student id="100026">  
    <name>Joe Average</name> <age>21</age>  
    <major>Biology</major>  
    <results>  
      <result course="Math 101" grade="C-"/>  
      <result course="Biology 101" grade="C+"/>  
      <result course="Statistics 101" grade="D"/>  
    </results>  
  </student>  
  <student id="100078">  
    <name>Jack Doe</name> <age>18</age>  
    <major>Physics</major>  
    <major>XML Science</major>  
    <results>  
      <result course="Math 101" grade="A"/>  
      <result course="XML 101" grade="A-"/>  
      <result course="Physics 101" grade="B+"/>  
      <result course="XML 102" grade="A"/>  
    </results>  
  </student>  
</students>
```

Beispiele nach *Anders Møller and Michael I. Schwartzbach*. An Introduction to XML and Web Technologies Addison-Wesley, 2006

- Let ist eine Zuweisung innerhalb einer Iteration über den Tupelstrom:

```
for $s in (<one/>, <two/>, <three/>)  
let $t := ($s, <four/>, <five/>)  
return <out>{$t} </out>
```

```
<out>  
  <one/>  
  <four/>  
  <five/>  
</out>
```

```
<out>  
  <two/>  
  <four/>  
  <five/>  
</out>
```

```
<out>  
  <three/>  
  <four/>  
  <five/>  
</out>
```

Am Beispiel

```
for $s in doc("test")//student
```

```
let $m := $s/major/text()
```

```
return <studies> {$s/name/text()}: {$m} </studies>
```

```
<studies>Joe Average: Biology</studies>
```

```
<studies>Jack Doe: PhysicsXML Science</studies>
```

- Filter der mit `allen` aus `for` und `let` erzeugten Tupeln geprüft wird
- Bei `true` wird `return` Klausel ausgeführt

```
for $s in fn:doc("students.xml")//student
let $m := $s/major
where fn:count($m) ge 2
return <double>{ $s/name/text() } </double>

<double>Jack Doe</double>
```


Verarbeitungsreihenfolge

- Bearbeitung und Ausgabe entsprechend dem Tupelstrom
- Änderbar durch ordering mode

```
unordered {  
for $p in fn:doc("parts.xml")/parts/part[color = "Red"],  
$s in fn:doc("suppliers.xml")/suppliers/supplier  
where $p/suppno = $s/suppno  
return <ps> { $p/partno, $s/suppno } </ps>  
}
```

- ordered: Entsprechend Standard, Dokumentenordnung
- unordered: Implementierungsabhängig, nichtdeterministisch

- order by Klausel ordnet den Tuplestrom entsprechend um und wendet dann die Return Klausel an

```
for $s in fn:doc("students.xml")//student
let $m := $s/major
where fn:count($m) ge 2
order by $s/@id
return <double> { $s/name/text() } </double>
```

- Sortieren nach mehreren Kriterien:

```
for $s in doc("students.xml")//student
order by fn:count($s/results/result[fn:contains(@grade,"A")])
        descending,
        fn:count($s/major) descending,
        xs:integer($s/age/text()) ascending
return $s/name/text()
```

```
<students>
<student id="100026">
  <name>Joe Average</name><age>21</age>
  <major>Biology</major>
  <results>
    <result course="Math 101" grade="C-"/>
    <result course="Biology 101" grade="C+"/>
    <result course="Statistics 101" grade="D"/>
  </results>
</student>
```

```
<student id="100078">
  <name>Jack Doe</name><age>18</age>
  <major>Physics</major>
  <major>XML Science</major>
  <results>
    <result course="Math 101" grade="A"/>
    <result course="XML 101" grade="A-"/>
    <result course="Physics 101" grade="B+"/>
    <result course="XML 102" grade="A"/>
  </results>
</student>
</students>
```

Beispiel aus Standard

```
<bib>
  <book>
    <title>TCP/IP Illustrated</title>
    <author>Stevens</author>
    <publisher>Addison-Wesley</publisher>
  </book>
  <book>
    <title>Advanced Programming
      in the Unix Environment</title>
    <author>Stevens</author>
    <publisher>Addison-Wesley</publisher>
  </book>
  <book>
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
  </book>
</bib>
```

```
<authlist>
{
  for $a in fn:distinct-values($bib/book/author)
  order by $a
  return
    <author>
      <name> {$a} </name>
      <books>
        {
          for $b in $bib/book[author = $a]
          order by $b/title
          return $b/title
        }
      </books>
    </author>
}
</authlist>
```

Xquery Dokumente
bestehen aus
Ausdrücken wie
Literalen, Variablen,
FLWORS etc.!

```
<authlist>
  <author>
    <name>Abiteboul</name>
    <books>
      <title>Data on the
Web</title>
    </books>
  </author>
  <author>
    <name>Buneman</name>
    <books>
      <title>Data on the
Web</title>
    </books>
  </author>
```

```
<author>
  <name>Stevens</name>
  <books>
    <title>Advanced Programming
in the Unix Environment</title>
    <title>TCP/IP Illustrated</title>
  </books>
</author>
<author>
  <name>Suciu</name>
  <books>
    <title>Data on the Web</title>
  </books>
</author>
</authlist>
```

- Return Klausel bei allen Tupeln angewandt die durch den where-Filter gekommen sind
- Return Ergebnisse zusammengehängt als Ergebnis der Anfrage

- Abstraktion und Fallunterscheidung

```
declare function local:grade($g) {  
  if ($g="A") then 4.0 else if ($g="A-") then 3.7  
  else if ($g="B+") then 3.3 else if ($g="B") then 3.0  
  else if ($g="B-") then 2.7 else if ($g="C+") then 2.3  
  else if ($g="C") then 2.0 else if ($g="C-") then 1.7  
  else if ($g="D+") then 1.3 else if ($g="D") then 1.0  
  else if ($g="D-") then 0.7 else 0  
};
```

Grade Point
Average

```
declare function local:gpa($s) {  
  fn:avg(for $g in $s/results/result/@grade return  
    local:grade($g))  
};
```


- Abstraktion und Fallunterscheidung

```
<gpas>  
  { for $s in fn:doc("students.xml")//student  
    return <gpa id="{ $s/@id}" gpa="{ local:gpa($s)}" />  
  }  
</gpas>
```

```
<gpas>  
  <gpa id="100026" gpa="1.66666666666666666667" />  
  <gpa id="100078" gpa="3.75" />  
</gpas>
```