6. XML-Anfragesprachen

- XPath
  - Schritte und Achsen
  - Beispiele

- XQuery
  - Grundlagen
  - FLWOR-Ausdrücke
  - Beispiele

Praktische Übungsmöglichkeit XQuery-Trainer: *http://lots.uni-leipzig.de/xqtrain/index.jsp*
  - XML//XQuery-Tutorial in LOTS (mit Übungsfragen)

Einleitung

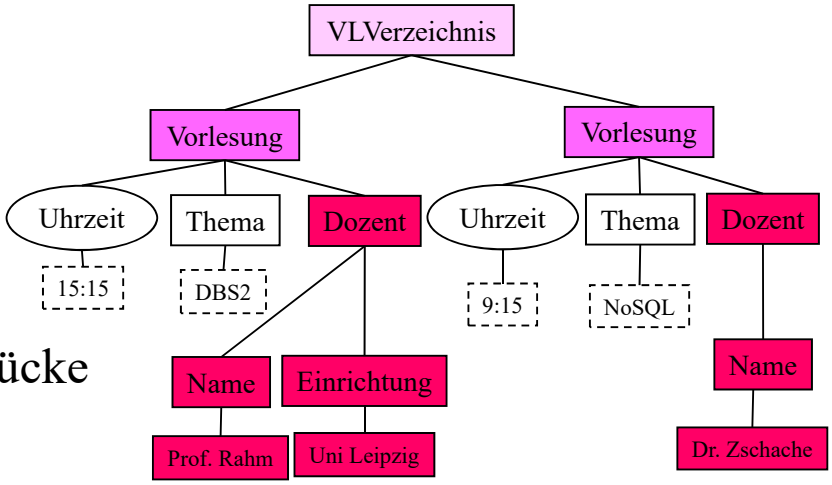
- deskriptive Anfragesprache für XML erforderlich
- Anfragesprachen wie SQL für XML unzureichend
  - Suche nach Inhalten auf beliebiger Hierarchieebene
  - Unterstützung für Pfadnavigation und Reihenfolgeabhängigkeiten
  - Unterstützung von Wildcards in Pfaden
  - Anfragemöglichkeit zu Metadaten und Daten
  - Anfragen auf schemalosen Daten
  - Unterstützung zur Neustrukturierung der Ergebnismenge
- Basisunterstützung durch XPath
  - ist Bestandteil von XQuery, XSLT …
  - Navigation und Selektion von Teildokumenten
  - 1999: V1.0, 2007: V2.0
- vollständige Anfragesprache: W3C XQuery
  - Recommendation (V1) seit Jan. 2007 , V3 seit 2014

XPath 1.0*

- Sprache zur Selektion von XML-Teildokumenten (Dokumentfragmente, Elemente, Attribute, Kommentare, Text, ...)

- Selektion durch schrittweise Navigation im Dokumentbaum

  Beispiel:
  /VLVerzeichnis/Vorlesung/Dozent

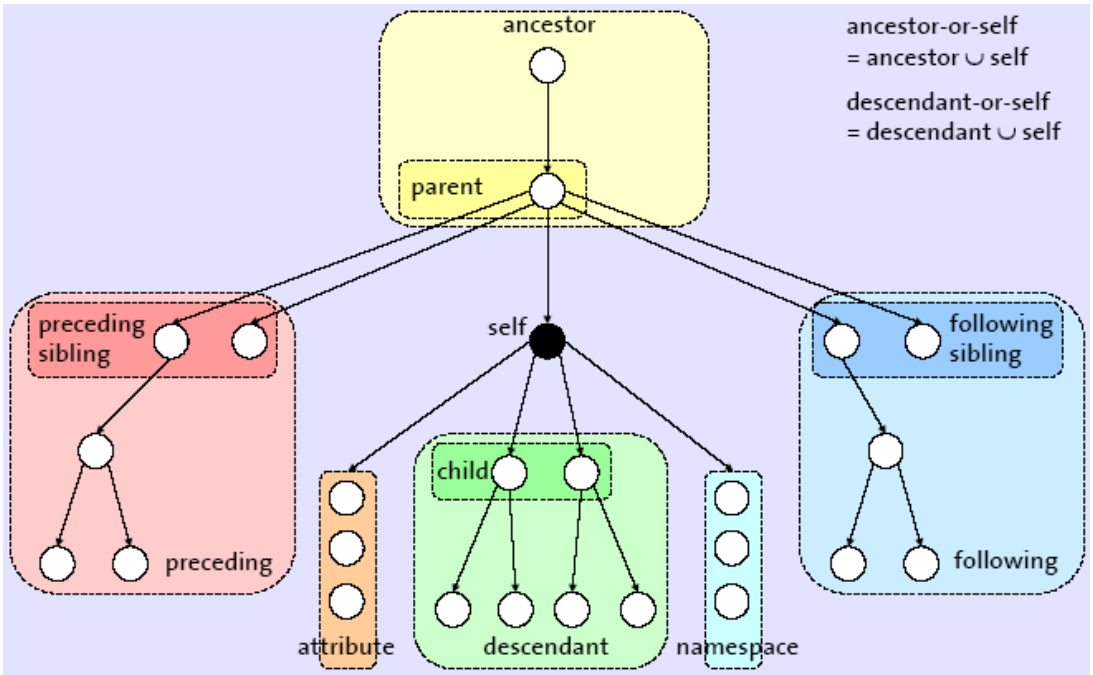


- absolute und relative Pfadausdrücke
  - absolut: /schritt1/schritt2
  - relativ: schritt1/schritt2
  - Abarbeitung von Pfadausdruck von links nach rechts
  - jeder Schritt liefert Knotenmenge oder Werte
- Syntax eines Schrittes: **Achse::Knotentest [Prädikat]**

* http://www.w3c.org/TR/xpath

XPath-Schritte: Achsen

- Achse: Richtung vom aktuellen Kontextknoten aus, in der die Knoten selektiert werden sollen
  - **13 Achsen**: ancestor, ancestor-or-self, child, descendant, descendant-or-self, following, following-sibling, parent, preceding, preceding-sibling, self, attribute, namespace)



XPath-Schritte: Knotentest

- **Knotentest**: selektiert Knoten aus der durch Achse vorgegebenen Menge (Selektion eines Knotens, wenn Test „wahr“ liefert)

Knotentest	selektierte Menge (aus Knotenmenge der Achse
Name	Elemente bzw. Attribute bzw. Namensraum mit diesem Namen
node()	alle Knoten
*	alle Elementknoten
text()	alle Textknoten
comment()	alle Kommentarknoten
processing-instruction()	alle Processing instructions

XPath-Schritte: Prädikate

- **Prädikat**: Filterausdruck, der nach dem Knotentest diejenigen Knoten selektiert, für die das Prädikat „wahr“ liefert
  - kann Vergleichsoperatoren (’=’, ’<’, ’<=’, ’!=’, ...)
  - logische Operatoren (and, or) und
  - Vereinigung von Knotenmengen (’|’) enthalten

  /child::VLVerzeichnis/child::Vorlesung[attribute::Uhrzeit=„15:15“]/child::Thema

  alternativ: */VLVerzeichnis/Vorlesung [@Uhrzeit=„15:15“]/Thema*
- Prädikat kann XPath-Ausdrücke enthalten: testet Existenz bestimmter Elemente/Attribute/Attributwerte

  /child::VLVerzeichnis/child::Vorlesung/child::Dozent[child::Einrichtung]/child::Name

  alternativ: */VLVerzeichnis/Vorlesung/Dozent [Einrichtung]/Name*
- bei Angabe von Zahlen werden Knoten der entsprechenden Kontextpositionen selektiert

  /child::*Produktliste/child::Produkt [position()=1]*

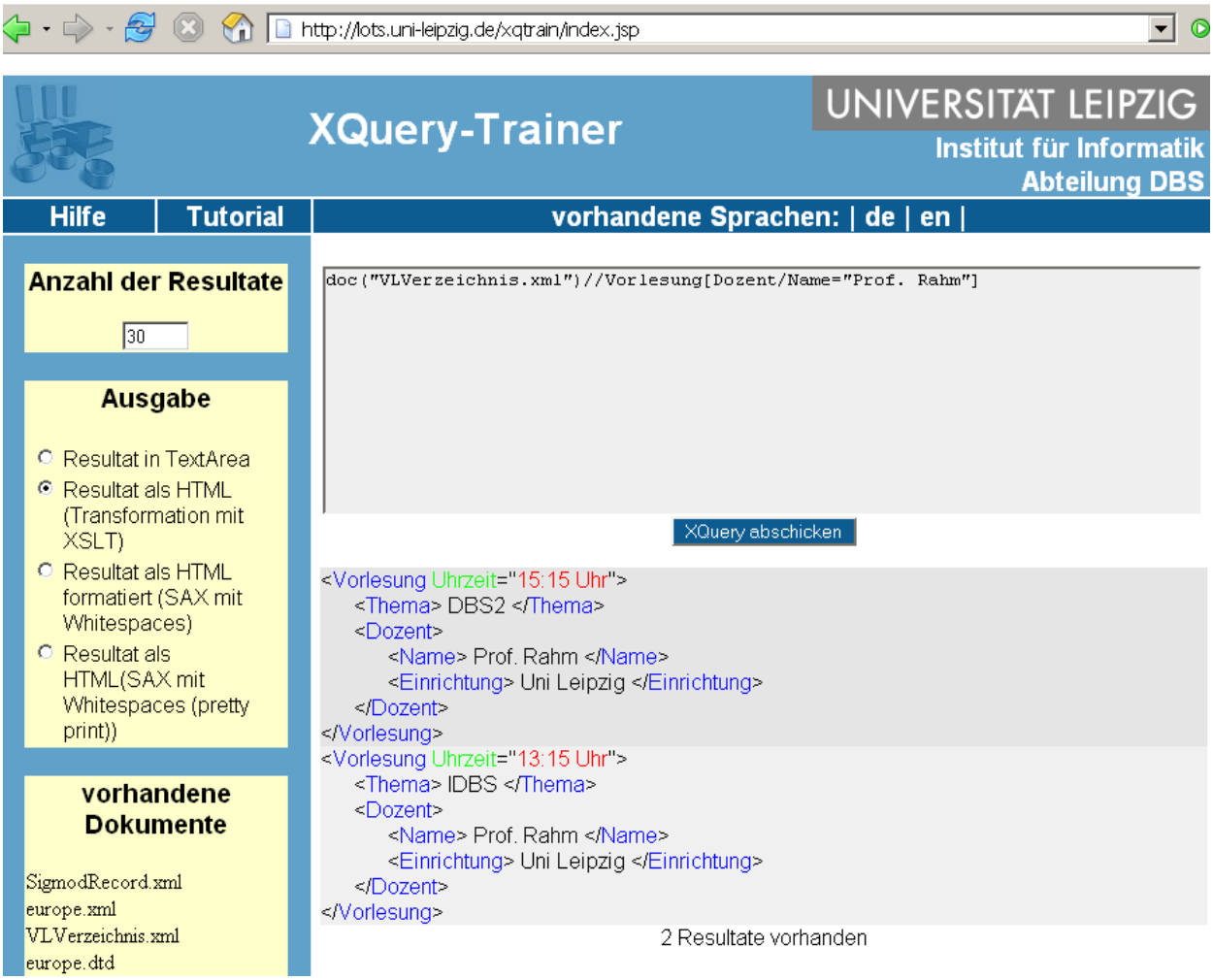
  alternativ: */Produktliste/Produkt [1]*

XPath: Abgekürzte Syntax/Funktionen

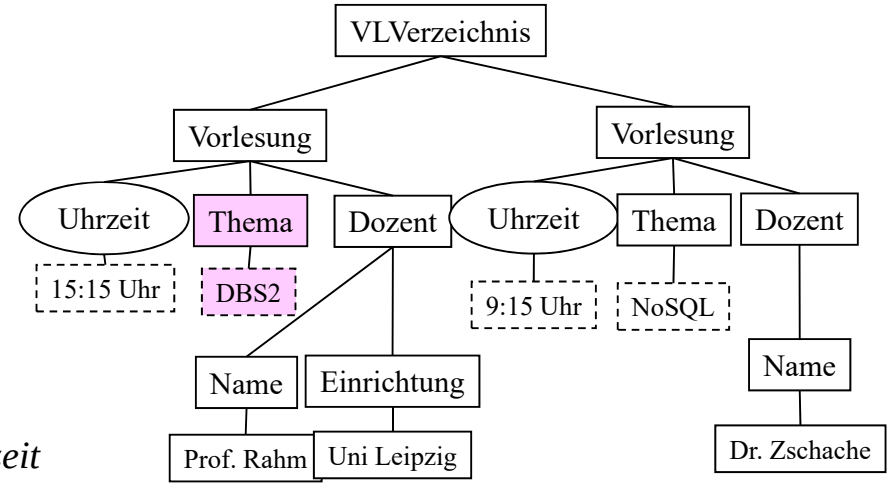
- XPath-Ausdrücke können durch abgekürzte Syntax vereinfacht werden

Langform	Abkürzung	Bemerkung
child::Knotentest	Knotentest	ohne Achsenangabe wird child-Achse verwendet
self::node()	.	aktueller Knoten
parent::node()	..	Elternknoten
descendant-or-self::node()	//	alle Nachkommen des aktuellen Knotens
attribute::Name	@Name	
[position()=X]	[X]	Prädikat zur Selektion von Knoten an Knotenposition X

- folgende Funktionen sind in XPath verfügbar (Auswahl):
  - für Knotenmenge: *name*(Knotenmenge), *last*(), *position*(), *count*(Knotenmenge), *id*(Name)
  - für Strings: *string*(Objekt), *concat*(String1, String2, ...), *starts-with*(String, Pattern), *contains*(String, Pattern), *substring*(String, Start[, Länge]), *string-length*()
  - für Zahlen: *number*(Objekt), *sum*(Knotenmenge), *round*(Zahl)
  - Boolesche Funktionen: *not* (Boolean), *true*(), *false*()



XPath: Beispiele



- Finde Uhrzeit der Vorlesung "DBS2"

//Thema[text() ="DBS2"]/../@Uhrzeit

- Vorlesungen, deren Dozent "Prof. Rahm" oder "Dr. Zschache" ist ?

//Vorlesung [Dozent/Name=„Prof. Rahm“ OR Dozent/Name=„Dr. Zschache“]

XPath: Beispiele (2)

- Themen aller Vorlesungen, die „15:15" beginnen und deren Dozent zur "Uni Leipzig" gehört ?

//Vorlesung [@Uhrzeit=„15:15“ AND

- Wie viele Vorlesungen sind im Verzeichnis enthalten?

- Welche Einrichtungen, enthalten den Text "Leipzig" ?

XQuery*

- XQuery: W3C-Standardisierung für einheitliche XML-Anfragesprache
- abgeleitet von vorangegangenen proprietären XML-Anfragesprachen (Quilt, XPath, XQL, XML-QL, ...) sowie SQL und OQL

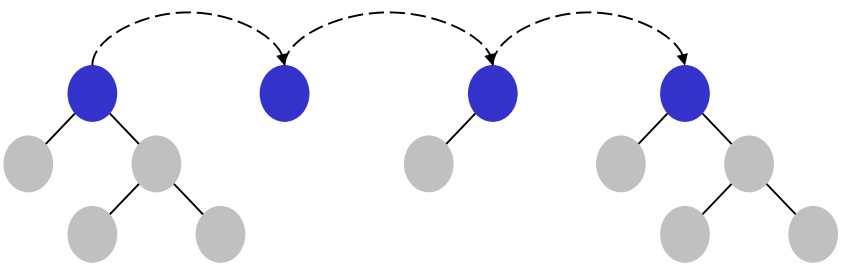
- FLWOR („flower“) –Syntax:
  For ... Let ... Where ... Order By ... Return)

- weitere Eigenschaften
  - funktionale Anfragesprache (Ausdrücke sind wieder als Parameter verwendbar)
  - komplexe Pfadausdrücke (basierend auf XPath 2.0)
  - Funktionen
  - konditionale und quantifizierte Ausdrücke
  - Ausdrücke zum Testen/Modifizieren von Datentypen
  - Elementkonstruktoren
  - Dokument-Änderungen mit „XQuery Update Facility“ (W3C Recommendation seit 2011)

* www.w3c.org/XML/Query

XQuery: Grundlagen

- jeder XQuery-Ausdruck liefert **Sequenz**
- *Sequenz*: geordnete Kollektion von **Items**
  - Syntax: (a, b, c)
- *Item*
  - entweder atomarer Wert (entsprechend den Simple Types von XML Schema)
    oder ein Knoten (Dokument-, Element-, Attribut-, Textknoten u. a.)
  - einzelnes Item entspricht Sequenz mit einem Item: a ≡ (a)
- Sequenzen sind flach: Sequenz aus a, (b, c), ( ) ≡ (a, b, c)



Syntax einer Query (unvollständig)

```
Query          ::= (Prolog)* QueryBody
Prolog         ::= NamespaceDecl | SchemaImport | VarDecl | FunctionDecl | ... ;
  NamespaceDecl ::= declare namespace NAME = URI
  SchemaImport  ::= import schema (namespace NAME =)? URI (at URI)?
  VarDecl       ::= declare variable $NAME (as TYPE)? ( := Expr | external)
  FunctionDecl  ::= declare function
                      NAME ( ($VAR (, $VAR)* )? ) (as TYPE)? ( {Expr} | external)
QueryBody      ::= Expr (, Expr)*
  Expr         ::= FLWOR-exp | QuantifiedExpr | TypeswitchExpr | IfExpr | OrExpr
```

XQuery-Ausdruck: Beispiele

- Literale, z. B. 1, 0.5, "a string"
- arithmetische Ausdrücke: 1+1
- Funktionen, z. B. true(), concat("1","3"), count (...), avg (...)
- Konstruktoren, z. B.: <a x="1"/>,
  element a { attribute x { 1 } }
- XPath 2.0-Ausdruck: doc("VLV.xml")//Dozent/Name
- FLWOR-Ausdruck:

```
for      $i in doc("VLV.xml")//Vorlesung
where    $i/@Uhrzeit=„15:15"
order by $i/Thema
return   $i/Thema
```

Teilschritte
- Dokumentzugriff mit *doc*(URI) oder *collection*(URI)
- Auswahl von Dokumentfragmenten mit XPath 2.0
- Variablenbindungen
- Operationen (Selektion, Verbund, ...) auf den gebundenen Daten
- Erzeugung neuer Knoten

FLWOR-Ausdrücke

```
FLWOR-expr    ::= (FOR-expr | LET-expr)+
                  WHERE-expr?
                  ORDERBY-expr?
                  return Expr
FOR-expr      ::= for $var in Expr (, $var in Expr)*
LET-expr      ::= let $var := Expr (, $var := Expr)*
WHERE-expr    ::= where Expr
ORDERBY-expr  ::= (order by | stable order by) OrderSpec (, OrderSpec)*
OrderSpec     ::= Expr OrderModifier
OrderModifier ::= (ascending | descending)? (empty greatest | empty least)? (collation StringLiteral)?
```

- For/Let: Variablenbindung an Datenquellen / Sequenzen
- Where: Auswahlbedingung
- Order by: Sortieranweisung (ansonsten Sortierung in Dokumentreihenfolge)
- Return: Festlegung, wie Ergebnis aussehen soll

FLWOR – For/Let (1)

- For/Let: Ergebnis eines XQuery-Ausdrucks wird an Variable gebunden
- For-Ausdruck
  - für jedes Wurzelelement der Ergebnissequenz erfolgt Bindung an Variable (Iteration über die Sequenz)
  - Beispiel:
    - $d wird jeweils an Elemente der Sequenz von Dozentennamen gebunden (für jeden Namen genau einmal)
    - RETURN wird für jede Bindung einmal ausgeführt

```
for $d in doc("VLV.xml")//Dozent/Name
return <Ergebnis> { $d } </Ergebnis>
```

```
<Ergebnis>
   <Name>Prof. Rahm</Name>
</Ergebnis>
<Ergebnis>
   <Name>Dr. Zschache</Name>
</Ergebnis>
```

- Let-Ausdruck
  - Ergebnis des XQuery-Ausdrucks wird geschlossen an Variable gebunden
  - Beispiel: RETURN wird einmal ausgeführt

```
let $d := doc("VLV.xml")//Dozent/Name
return <Ergebnis> { $d } </Ergebnis>
```

FLWOR – For/Let (2)

- Kombination von let/for-Ausdrücken

```
let $j := ("A", "B", "C")
for $i in $j
return
   <i>{ $i }</i>
```

→

```
<i>A</i>
<i>B</i>
<i>C</i>
```

```
for $i in (1, 2, 3)
let $j := ("A", "B", "C")
return
   <tuple>
        <i>{ $i }</i> <j>{ $j }</j>
   </tuple>
```

→

```
<tuple> <i>1</i> <j>A B C</j> </tuple>
<tuple> <i>2</i> <j>A B C</j> </tuple>
<tuple> <i>3</i> <j>A B C</j> </tuple>
```

```
for    $i in (1, 2, 3),
       $j in ("A", "B", "C")
return
   <tuple>
        <i>{ $i }</i> <j>{ $j }</j>
   </tuple>
```

→

```
<tuple> <i>1</i> <j>A</j> </tuple>
<tuple> <i>1</i> <j>B</j> </tuple>
<tuple> <i>1</i> <j>C</j> </tuple>
<tuple> <i>2</i> <j>A</j> </tuple>
<tuple> <i>2</i> <j>B</j> </tuple>
<tuple> <i>2</i> <j>C</j> </tuple>
<tuple> <i>3</i> <j>A</j> </tuple>
<tuple> <i>3</i> <j>B</j> </tuple>
<tuple> <i>3</i> <j>C</j> </tuple>
```

FLWOR - Where

- **Where**: eliminiert alle Wertekombinationen der durch let/for gebundenen Variablen, bei denen der nachfolgende Ausdruck nicht zu 'wahr' ausgewertet wird
  - wird für jeden Wert der Variablenbindungen ausgewertet
  - bei mehreren Variablen erfolgt Anwendung auf jeden Wert des kartesischen Produkts
- Auswertung des Ausdrucks liefert auch 'falsch' bei:
  - einer leeren Sequenz
  - einem 0 Zeichen langen String-Wert

```
for    $i in (1, 2, 3),
       $j in (4, 5, 6)
where ($i + $j) < 7
return
   <tuple>
        <i>{ $i }</i> <j>{ $j }</j>
   </tuple>
```

→

```
<tuple> <i>1</i> <j>4</j> </tuple>
<tuple> <i>1</i> <j>5</j> </tuple>
<tuple> <i>2</i> <j>4</j> </tuple>
```

```
for $d in doc("VLV.xml")//Dozent
where $d/Einrichtung
return $d/Name
```

FLWOR – Order by

- sortiert Wertekombinationen der durch for/let gebundenen Variablen, die nach *where*-Filter übrigblieben, entsprechend des Ausdruck-Ergebnisses
- jeder Ausdruck muss einen *atomaren Wert* liefern

falsch

```
for $b in doc("bib.xml")//book
order by $b/authors/author
return $b/title
```

richtig

```
for $b in doc("bib.xml")//book
order by $b/authors/author[1]
return $b/title
```

FLWOR - Return

- zu jeder Wertekombination der durch for/let gebundenen Variablen, die nach dem *where*-Filter übriggeblieben ist, wird der return-Ausdruck ausgewertet und als *Item* (oder Folge von *Items*) in die Ergebnissequenz aufgenommen
- ohne *order by* wird die Reihenfolge durch die *for/let*-Klauseln bestimmt

```
for $i in (1, 2, 1),
    $d in doc("Mitarbeiterverzeichnis.xml")
          //Mitarbeiter[position()=$i]//Name
return $d
```

→

```
<Name>Erich Schmidt</Name>
<Name>Silke Neumann</Name>
<Name>Erich Schmidt</Name>
```

- Neustrukturierung des Ergebnisses mit Hilfe von Konstruktoren

XQuery: Konstruktoren

- Konstruktoren erlauben die Erzeugung von XML-Strukturen
- direkte Konstruktoren (direct constructor)
  - Strukturen werden in XML-ähnlicher Syntax erzeugt
  - Ausdrücke in {} im Elementinhalt oder Attributwert werden ausgewertet/berechnet

```
<book year="2015">
   <title>Verteilte DBS</title>
</book>
```

```
<book year="{2000 + $year}">...
   <title>{ data($b/name) }</title>
</book>
```

- berechnete Konstruktoren (computed constructor)
  - Knotendefinition mit KNOTENTYP NAME { INHALT }
  - Name kann dynamisch berechnet werden

```
element book {
   attribute year { "2015" },
   element title { „Verteilte DBS" } }
```

```
element {node-name($b) } {
   attribute year { 2000 + $year },
   element title { data($b/name) } }
```

- können an Variable gebunden oder in Return-Klausel verwendet werden

```
let $b:=<books><title>A</title>
                <title>B</title></books>
return count($b/title)
```

```
<books>
  { for $b in (<title>A</title>, <title>B</title>)
    return <name>{ data($b) }</name>
  }
</books>
```

XQuery: Weitere Ausdrücke

- **quantifizierte Ausdrücke**

QuantifiedExpr ::= (**some** | **every**) $var **in** Expr (, $var **in** Expr)* **satisfies** Expr

  - Existenz- und Allquantifizierung werden unterstützt
  - Testausdruck wird über Kartesischem Produkt der gebundenen Sequenzen ausgewertet
  - Ergebnis ist ein Boolean-Wert

```
some $book in $b/book satisfies ($book/price < 10)
```

```
some $x in (2, 3, 4), $y in (3, 4, 5)
      satisfies $x + $y = 6
```

```
every $book in doc("bib.xml")//book satisfies $book/author[2]
```

- **konditionale Ausdrücke**

IfExpr ::= **if (** Expr (, Expr)* **) then** Expr **else** Expr

```
if ($book1/price < $book2/price)
   then $book1/title
   else $book2/title
```

XQuery: Funktionen

- XQuery (und XPath 2.0) enthalten ca. 100 Funktionen
  - Standardnamensraum ist *www.w3.org/TR/xpath-functions* mit Präfix *fn:*
- Funktionen auf numerischen Daten
  - fn:*abs*($arg), fn:*ceiling*($arg), fn:*floor*($arg), fn:*round*($arg)
- Funktionen auf Zeichenketten-Daten
  - fn:*concat*($arg, $arg, ...), fn:*substring*($arg, start (, length)?), fn:*string-length*($arg), fn:*contains*($arg1, $arg2), fn:*starts-with*($arg1, $arg2), fn:*matches*($arg, $regExpr), fn:*replace*($arg, $regExpr, $replacement) ...
- 20+ Funktionen für Datum und Zeit, z.B. fn:*day-from-date*
- Funktionen auf Sequenzen
  - fn:*empty*($seq), fn:*exists*($seq)
  - fn:*distinct-values*($arg): Atomisierung der Elemente und eliminiert danach Duplikate

Achtung:

```
let $a := (<A>aaa</A>, <B>bbb</B>, <C>aaa</C>)
return distinct-values($a)
```

liefert:

aaa
bbb

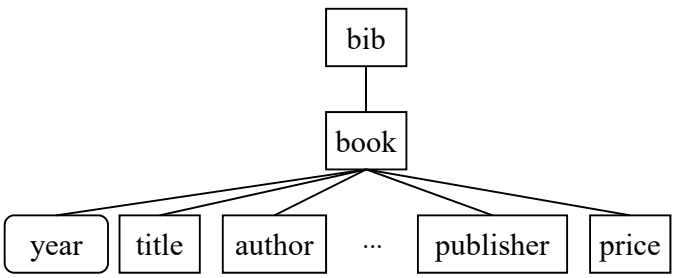
aber nicht:

<A>aaa</A>
<B>bbb</B>

XQuery: Beispielschema

- folgendes Schema (DTD) liegt den folgenden XQuery-Beispielen zugrunde (aus XQuery Use cases)
- zugehöriges Dokument sei durch *bib.xml* referenzierbar

```
<!ELEMENT bib  (book* )>
<!ELEMENT book  (title, author+, publisher, price )>
<!ATTLIST book  year CDATA  #REQUIRED >
<!ELEMENT author  (#PCDATA)>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT publisher  (#PCDATA )>
<!ELEMENT price  (#PCDATA )>
```



XQuery: Reihenfolge/Sortierung

- Liste alle Buchtitel, in denen "Erhard Rahm" Erstautor ist

```
doc("bib.xml")//title[../author[1]="Erhard Rahm"]
```

- Liste die ersten 10 Bücher

```
doc("bib.xml")//book[position() = (1 to 10)]
```

- Liste alle Titel von Büchern, die bei Springer nach 2000 publiziert wurden, in alphabetischer Reihenfolge

```
for $b in doc("http://dbs.uni-leipzig.de/bib.xml")/bib/book
where

order by
return
```

XQuery: Verbundoperationen

- Liste alle Dozenten, die auch Buchautoren sind (innerer Verbund)
  (Annahme: Dozent hat Vorname/Name)

```
for $d in distinct-values(doc("VLVerzeichnis.xml")//Dozent/Name)
for $a in doc("bib.xml")//author[text() = $d]
return
  <DozentUndAutor>
    { $a/text() }
  </DozentUndAutor>
```

- Liste alle Dozenten und markiere Autoren mit einem Attribut ’author’ (äußerer Verbund)

```
for $d in distinct-values(doc("VLVerzeichnis.xml")//Dozent/Name)
return
  <Dozent>
    {    for $a in doc("bib.xml")//author [text() = $d]
         return attribute author { "yes" },
      $d
    }
  </Dozent>
```

```
for $d in distinct-values(doc("VLVerzeichnis.xml")//Dozent/Name)
return
 <Dozent>
   {
    for $a in doc("bib.xml")//author[surname = substring-after(string($d)," ")]
    return
      attribute author { "yes" },
      $d
   }
 </Dozent>
```

XQuery abschicken

```
<Dozent author="yes"> Prof. Rahm </Dozent>
<Dozent> Dr. Sosna </Dozent>
<Dozent> Prof. Heyer </Dozent>
<Dozent> Prof. Gräbe </Dozent>
<Dozent> Prof. Fähnrich </Dozent>
```

5 Resultate vorhanden

XQuery: Gruppierung/Aggregation

- Ermittle alle Verlage und den Durchschnittspreis ihrer Bücher

```
<Verlage>
  for $p in distinct-values(doc("bib.xml")//publisher)

  return

</Verlage>
```

- Lösung mit XQuery V3 (Group-By-Klausel)

```
<Verlage>
 for $b in doc("bib.xml")//books
  group by $b/publisher
  return   <Name> data($b/publisher)  </Name>
           <Dpreis> avg ($b/price)  </Dpreis>
</Verlage>
```

XQuery: Gruppierung/Join

- Liste für jeden Autor seinen Namen und die Titel all seiner Bücher, gruppiert in einem Ergebnis-Element

```
for $a in distinct-values(doc("bib.xml")//author)
 return <Ergebnis> { $a }
            { for $b in doc("bib.xml")/bib/book
                where some $ba in $b/author satisfies $ba = $a
                return $b/title      }
             </Ergebnis>
```

```
<Ergebnis> Rade Lennart
    <title> Springers Mathematische Formeln </title>
</Ergebnis>
<Ergebnis> Westergren Bertil
    <title> Springers Mathematische Formeln </title>
</Ergebnis>
<Ergebnis> Bernstein Philip A.
    <title> Data Warehouse Scenarios for Model Management. </title>
    <title> Generic Schema Matching with Cupid. </title>
    <title> Panel: Is Generic Metadata Management Feasible? </title>
    <title> Rondo: A Programming Platform for Generic Model Management. </title>
</Ergebnis>
<Ergebnis> Rahm Erhard
    <title> Data Warehouse Scenarios for Model Management. </title>
    <title> Concurrency Control in DB-Sharing Systems. </title>
    <title> Comparative Evaluation of Microarray-based Gene Expression Databases. </title>
    <title> WebFlow: Ein System zur flexiblen Ausführung webbasierter, kooperativer Workflows.
    </title>
    <title> On Parallel Join Processing in Object-Relational Database Systems. </title>
    <title> XMach-1: A Benchmark for XML Data Management. </title>
```


Zusammenfassung

- standardisierte XML-Anfragesprachen: XPath, XQuery
- XPath: einfache Auswahl über Pfadausdrücke
  - keine vollwertige Anfragesprache (Verbund, Sortierung, Neustrukturierung, ...)
- XQuery: komplexe Anfragesprache
  - basierend auf Sequenzen
  - unterstützt Typisierung
  - vielfältige Anfrageausdrücke, u.a. FLWOR-Ausdrücke
  - nutzerdefinierte Funktionen
  - großer Funktionsumfang