

6. XML-Datenbanken: Anfragesprachen

- XPath*
 - Schritte und Achsen
 - Beispiele
- XQuery*
 - Grundlagen
 - FLWOR-Ausdrücke
 - Beispiele
- Speicherung von XML-Dokumenten
 - Speicherungsverfahren
 - XML Mapping in ORDBS

Literatur zu XQuery

- H. Katz (Hrsg.): *XQuery from the Experts*, Addison-Wesley, 2003
<http://www.datadirect.com/developer/xquery/xquerybook/index.ssp>
- W. Lehner, H. Schöning: *XQuery*, dpunkt.verlag, 2004

* praktische Übungsmöglichkeit **XQuery-Trainer**: <http://lots.uni-leipzig.de/xqtrain/index.jsp>



Einleitung

- Deskriptive Anfragesprache für XML erforderlich
 - neben XML-Verarbeitung über XSLT oder Programmier-APIs (DOM)
- Anfragesprachen wie SQL für XML unzureichend
 - unzureichende Unterstützung für Pfadnavigation und Reihenfolgeabhängigkeiten
 - keine Unterstützung von Wildcards in Pfaden
 - unzureichende Anfragemöglichkeit zu Metadaten
 - Anfragen auf schemalosen Daten
 - keine Unterstützung zur Neustrukturierung der Ergebnismenge
- Basisunterstützung durch XPath
 - ist Bestandteil von XQuery, XSLT ...
 - Navigation und Selektion von Teildokumenten
- Vollständige Anfragesprache: W3C XQuery
 - Recommendation (V 1.0) seit Jan. 2007



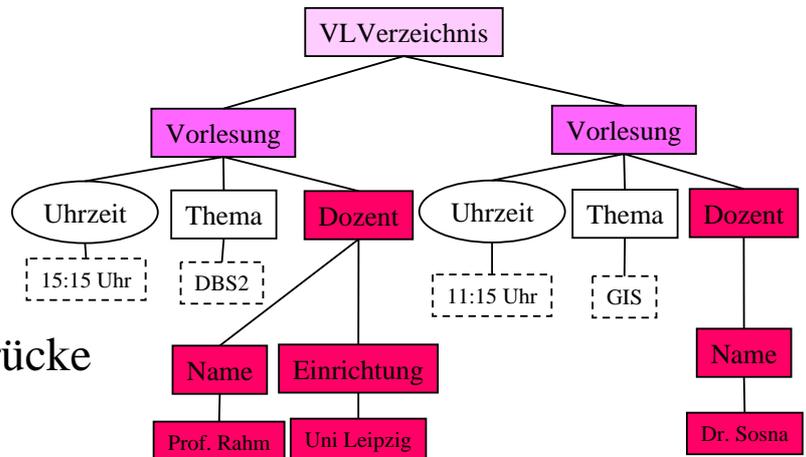
XPath 1.0*

- 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]**

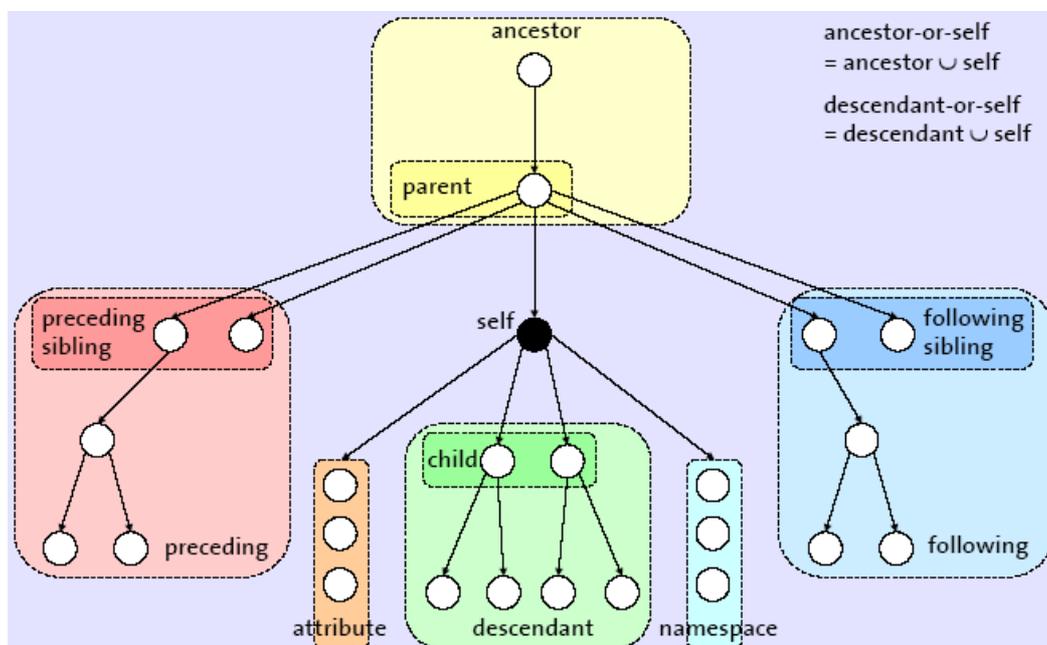
* <http://www.w3c.org/TR/xpath>



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::Name

alternativ: */VLVerzeichnis/Vorlesung [@Uhrzeit=„15:15“]/Name*

- 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

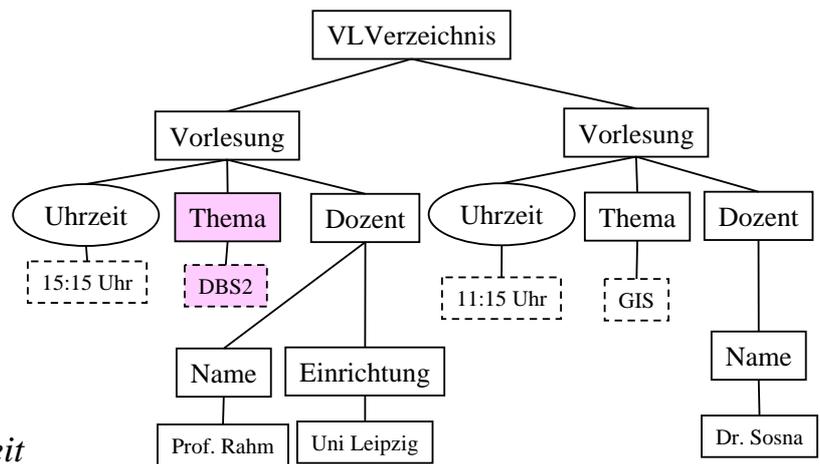
Langform	Abkürzung	Bemerkung
child::Knotentest	Knotentest	ohne Achsenangabe wird child-Achse verwendet
self::node()	.	aktueller Knoten
parent::node()	..	Elternknoten
descendant-or-self::node()	//	alle Nachkommen des aktuellen Knotens
attribute::Name	@Name	
[position()=X]	[X]	Prädikat zur Selektion von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

- Liste alle Vorlesungen, deren Dozent "Prof. Rahm" oder "Dr. Sosna" ist



XPath: Beispiele (2)

- Liste das Thema aller Vorlesungen, die „15:15“ beginnen und deren Dozent zur "Uni Leipzig" gehört

- Wie viele Vorlesungen sind im Verzeichnis enthalten?

- Liste alle Einrichtungen, die den Text "Leipzig" enthalten



XPath 2.0*

- W3C-Standardisierung im Rahmen der XQuery-Entwicklung
- Änderungen gegenüber XPath 1.0:
 - Verwendet Typsystem von XML Schema (XPath 1.0 nur Knotenmenge, boolesche, numerische und Zeichenkettenwerte)
 - basiert auf Sequenzen (XPath 1.0 basiert auf Mengen); Unterschied bei Duplikaten
 - unterstützt Referenzen
 - kann mit Dokumentkollektionen arbeiten
 - kennt Variable
 - Funktionen bzw. Variable können Pfadausdrücken vorangestellt werden
 - Unterscheidung Wertegleichheit und Knotenidentität
 - wesentlich erweiterte Funktionsbibliothek
 - Bereichsausdrücke in Prädikaten

* <http://www.w3.org/TR/xpath20/>



XQuery*

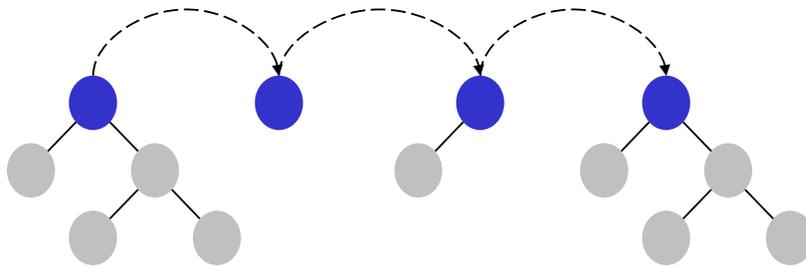
- XQuery: W3C-Standardisierung für einheitliche XML-Anfragesprache
- abgeleitet von vorangegangenen proprietären XML-Anfragesprachen (Quilt, XPath, XQL, XML-QL, ...) sowie SQL und OQL
- **FLWOR** („flower“) –**Syntax:**
For ... Let ... Where ... Order By ... Return)
- Weitere Eigenschaften
 - funktionale 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
- Noch fehlend in V1.0
 - Update-Operationen
 - Volltextsuche

* <http://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: $a \equiv (a)$
- Sequenzen sind flach: Sequenz aus a, (b, c), $() \equiv (a, b, c)$



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.: ,
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 Wurzelement der Ergebnissequenz erfolgt Bindung an Variable (Iteration über die Sequenz)

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

- Beispiel:

- \$d wird jeweils an Elemente der Sequenz von Dozentennamen gebunden (für jeden Namen genau einmal)
- RETURN wird für jede Bindung einmal ausgeführt

```
<Ergebnis>
  <Name>Prof. Rahm</Name>
</Ergebnis>
<Ergebnis>
  <Name>Dr. Sosna</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
 - einem numerischen Wert gleich 0

```
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 übriggeblieben, 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="2003">
  <title>XML und DB</title>
</book>
```

```
<book year="{1900 + $year}">...
  <title>{ data($b/name) }</title>
</book>
```

- **berechnete Konstruktoren** (computed constructor)

- Knotendefinition mit KNOTENTYP NAME { INHALT }
- Name kann dynamisch berechnet werden

```
element book {
  attribute year { "2003" },
  element title { "XML und DB" } }
```

```
element { node-name($b) } {
  attribute year { 1900 + $year },
  element title { data($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: Vergleichsoperatoren

- Wertevergleich: eq, ne, lt, le, gt, ge
 - vergleicht einzelne Werte (atomare Werte oder einelementige Sequenzen)
- Allgemeiner Vergleich =, !=, <, <=, >, >=
 - kann Sequenzen miteinander vergleichen
 - existentiell quantifiziert
 - ist nicht transitiv

let \$a := (1, 2, 3), \$b := (3, 4, 5), \$c := (5, 6, 7)

\$a = \$b liefert true

\$b = \$c liefert true

\$a = \$c liefert false

■ Knotenvergleich

- is: Identität von Knoten
- <<, >> Test auf Reihenfolge von Knoten
 - z. B. \$t/author << \$t/editor testet, ob author-Element vor editor-Element auftritt



XQuery: Funktionen

- XQuery (und XPath 2.0) enthalten ca. 100 Funktionen
 - Standardnamensraum ist *http://www.w3.org/2004/10/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:compare(\$arg, \$arg, (collation)?)*, *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, \$regex)*, *fn:replace(\$arg, \$regex, \$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: Benutzerdefinierte Funktionen

- Benutzerdefinierte Funktionen können im Prolog einer Anfrage deklariert werden
 - Funktionsdefinition kann direkt mit Sprachelementen von XQuery erfolgen
 - externe Funktionsdefinitionen müssen über Laufzeitumgebung bereitgestellt werden
- Rekursion und Überladen werden unterstützt
- Funktion muss Namensraum zugeordnet werden
 - Default: Namensraum der XQuery-Funktionen

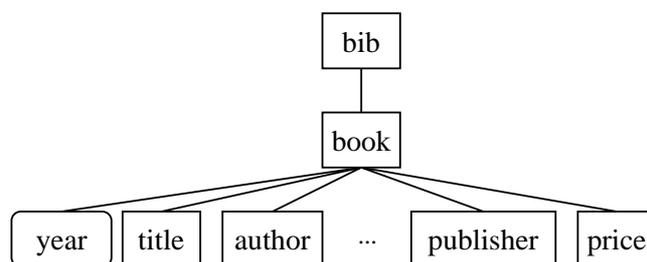
```
declare namespace dbs = "http://dbs.uni-leipzig.de/ns/dbs";  
declare function dbs:authorCount ( $p as element() ) as xs:integer {  
    count($p//author)  
};  
for $e in doc("bib.xml")//book  
where dbs:authorCount($e) > 2  
return $e/title
```



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 dpunkt 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

- Gebe alle Verlage und den Durchschnittspreis ihrer Bücher aus.

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

return <publisher>

</publisher>
```

- 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>
```



```
for $a in distinct-values(doc("bib.xml")//author)
return <Ergebnis> { $a }
  { for $b in doc("bib.xml")/bib/biblioentry [authorgroup/author = $a ]
    return $b/title }
  </Ergebnis>
```

XQuery abschicken

```
<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>
```



XML-Unterstützung in kommerziellen DBS

- (objekt-)relationale DBS mit XML-Erweiterung
 - Datenzugriff vorrangig über SQL
 - Erweiterungen zur Transformation zwischen XML-Dokumenten und relationalen Datenstrukturen (z. B. UDFs zum Speichern und Abfragen von XML-Dokumenten)
 - vorrangig für datenorientierte XML-Dokumente optimiert
 - Bsp.: Oracle, IBM DB2, Microsoft SQL-Server
- native XML-DBS
 - Datenzugriff erfolgt vorwiegend über XML-orientierte Schnittstellen (z. B. XPath, XSLT, DOM, XQuery)
 - DBS ist in erster Linie zur Speicherung und Manipulation von XML-Daten bestimmt
 - für alle XML-Dokumentarten geeignet (besonders für dokumentorientierte und gemischt strukturierte)
 - Beispiele: Tamino, eXtensible Information Server (XIS), Infonyte DB ...



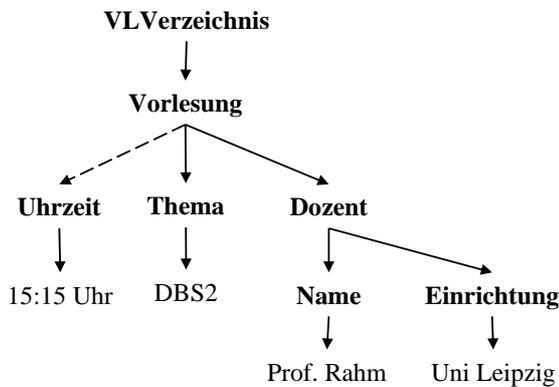
DB-Speicherverfahren für XML-Dokumente

- Ganzheitliche Speicherung von Dokumenten (BLOB, CLOB, UDT XML)
 - Erlaubt schnelle, originalgetreue Dokumentrekonstruktion
 - schnelles Einbringen der Daten
 - hoher Aufwand für Queries und Änderungen
 - weniger geeignet für strukturierte Daten
- Dekomposition: Zerlegung der XML-Daten
 - Generisch: Graphmodell, DOM
 - Allgemein; aufwändigere Queryverarbeitung
 - Günstig für Dokumente / Textanteile
 - Schemabasiert: aufgrund manuellem / automatischem Tabellen-Mapping
- Kombinationen



Generische Dekomposition

- Generische(anwendungsneutrale) Abbildung von Hierarchien (XML-Instanzen) bzw. Graphen (XML Schema) auf Relationen



docID	elementname	ID	vorgänger	pos	wert
1	VLVerzeichnis	1	Null	1	null
1	Vorlesung	2	1	1	Null
1	Thema	3	2	1	DBS2
1	Dozent	4	2	2	Null
1	Name	5	4	1	Prof. Rahm
1	Einrichtung	6	4	2	Uni Leipzig

Elemente

docID	attributname	elementID	wert
1	Uhrzeit	2	15:15

Attribute



Schemabasierte Dekomposition

- Relationenschema wird in Abhängigkeit vom XML-Schema erzeugt
- automatisches Mapping nach Bourret* (komplexe Elemente erhalten eigene Relation)

VLVerzeichnis

VLVerzeichnigID
1

Vorlesung

VorlesungID	VorlesungPos	VIVerzeichnisID	Uhrzeit	UhrzeitPos	Thema	ThemaPos
1	1	1	15:15	1	DBS2	2

Dozent

Dozent	DozentPos	VorlesungID	Name	NamePos	Einrichtung	EinrichtungPos
1	3	1	Prof. Rahm	1	Uni Leipzig	2

- mit ROW-Typ, ohne Elementposition

VLVerzeichnis

VLVerzeichnigID
1

Vorlesung

VorlesungsID	Uhrzeit	Thema	Dozent	
			Name	Einrichtung
1	15:15	DBS2	Prof. Rahm	Uni Leipzig

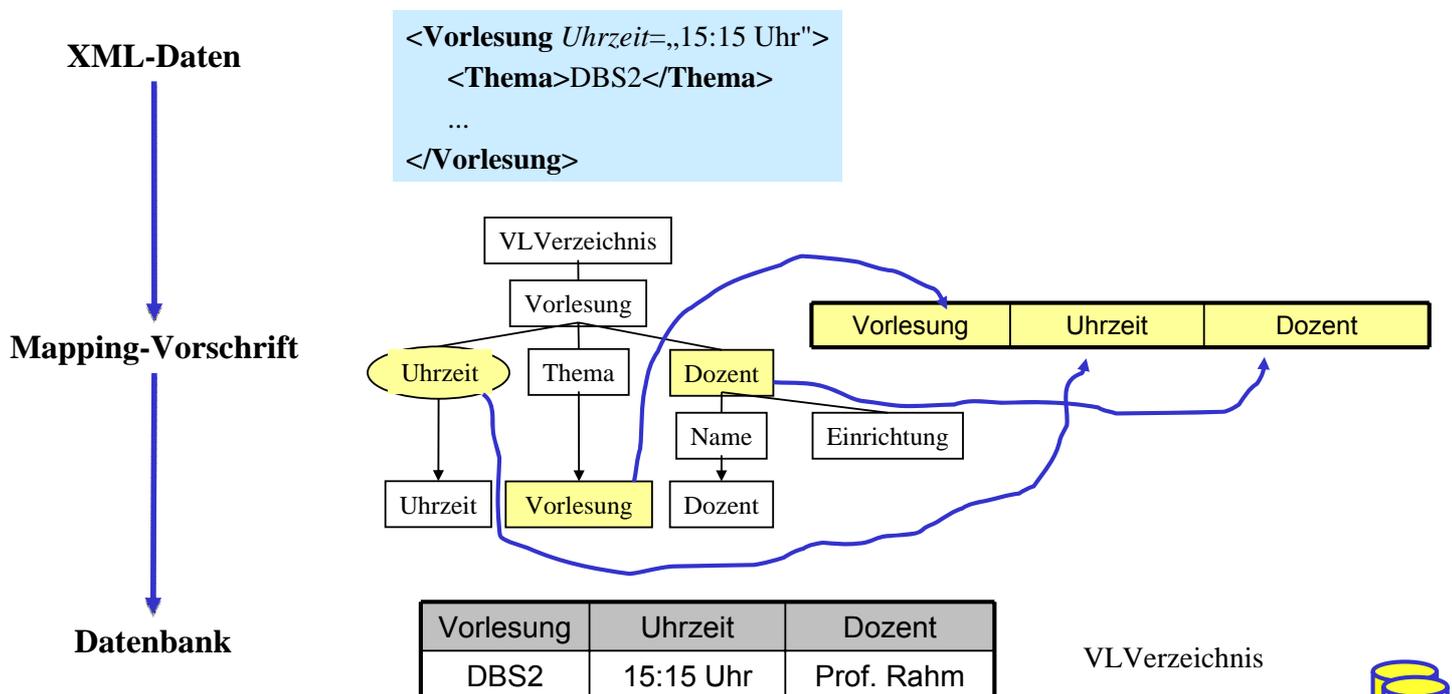
* <http://www.rpbouret.com/xmldbms/>



Schemabasierte Dekomposition (2)

■ manuelles Mapping

- Mapping zwischen XML-Dokument und Datenbank wird manuell festgelegt
- Mapping muss nicht vollständig sein



SQL-Datentyp: XML

- SQL:2003-Standard Teil 14: SQL/XML
- XML-Dokument wird als Ganzes gespeichert
 - Beibehaltung der Originaldokumente/-daten (ähnlich LOB)
- DBMS erlaubt XPath (und XQuery)-Anfragen auf XML-Dokumenten
- Beispiel: Tabelle Vorlesungsverzeichnis

```
- CREATE TABLE Vorlesungsverzeichnis (  
  Universitaet VARCHAR (255),  
  Semester VARCHAR (255),  
  Vorlesungen XML  
)  
  
- INSERT INTO Vorlesungsverzeichnis  
(Universitaet, Semester, Vorlesungen)  
VALUES  
(`Uni Leipzig`, `SoSe 2008`, `Uhrzeit="15:15 Uhr"><Thema> DBS2 </Thema><Dozent>  
<Name> Rahm </Name> <Einrichtung> IfI </Einrichtung> ...`)
```

SQL/XML-Operatoren

- Erweiterung der SQL-Anfragesprache zur XML-Verarbeitung
- Operatoren zur Erzeugung von XML-Daten
 - **XMLELEMENT**, **XMLATTRIBUTES** : Konstruktion von XML-Elementen bestehend aus Name, Attributen und Inhalt (Text oder andere XML-Elemente)
 - **XMLAGG**: Konstruktion gruppierter XML-Elemente
 - **XMLFOREST**, **XMLCONCAT**, **XMLNAMESPACES**, **XMLDOCUMENT**, ...
- Operatoren für Anfragen auf XML-Dokumenten
 - **XMLQUERY**: Einfache XPath-Anfragen auf XML-Dokumente
 - **XML EXISTS**: Selektionsbedingung für XML-Dokumente (in SQL-WHERE-Klausel)
 - **XMLTABLE**: Konvertierung eines XML-Anfrageergebnisses in Relation



XMLELEMENT, XMLATTRIBUTES

- Erstellung von XML-Elementen bzw. –Attributen aus relationalen Tabellen
- Beispiel: Vorlesungen jeweils als XML-Fragment

```
SELECT Id, XMLELEMENT (Name "Vorlesung",  
                        XMLATTRIBUTES (Uhrzeit AS "Uhrzeit"),  
                        XMLELEMENT (Name "Thema", Thema),  
                        XMLELEMENT (Name "Dozent",  
                                    XMLELEMENT (Name "Name", DName),  
                                    XMLELEMENT (Name "Einrichtung", DEinr))) AS X  
FROM Vorlesung
```

Vorlesung				
Id	Thema	Uhrzeit	DName	DEinr
1	DBS2	15:15 Uhr	Rahm	IfI
2	DatenInt	11:15 Uhr	Thor	IfI

Id	X (Typ:XML)
1	<Vorlesung Uhrzeit="15:15 Uhr"><Thema>DBS2</Thema> <Dozent><Name>Rahm</Name><Einrichtung>IfI</Einrichtung> </Dozent></Vorlesung>
2	<Vorlesung Uhrzeit="11:15 Uhr"><Thema>DatenInt</Thema> <Dozent><Name>Thor</Name><Einrichtung>IfI</Einrichtung> </Dozent></Vorlesung>



XMLAGG

- Zusammenfassung von XML-Elementen basierend auf
 - SQL-Gruppierung (GROUP BY)
 - Sub-Selects
- Beispiel: Alle Dozenten gruppiert nach ihrer Einrichtung

```
SELECT DEinr, XMLELEMENT (Name "Einrichtung",
                        XMLATTRIBUTES (DEinr AS "Name"),
                        XMLAGG(
                            XMLELEMENT (Name "Dozent", DName))) AS X
FROM Vorlesung
GROUP BY DEinr
```

Vorlesung				
Id	Thema	Uhrzeit	DName	DEinr
1	DBS2	15:15 Uhr	Rahm	Ifl
2	DatenInt	11:15 Uhr	Thor	Ifl
3	FinanzInf	11:15 Uhr	Alt	WiInf

DEinr	X (Typ:XML)
Ifl	<Einrichtung Name="Ifl"> <Dozent>Rahm</Dozent><Dozent>Thor</Dozent> </Einrichtung>
WiInf	<Einrichtung Name="WiInf"> <Dozent>Alt</Dozent> </Einrichtung>



XMLQUERY

- Unterstützung einfacher XPath-Anfragen
 - Ergebnis ist Knotenmenge (serialisiert als konkatenierte XML-Elemente)
 - PASSING-Klausel gibt an, in welcher Tabellen-Spalte XML-Dokumente ausgewertet werden sollen
- Beispiel: Alle Themen von 15:15-Uhr-Vorlesungen

```
SELECT Universitaet, XMLQUERY(
    '//Vorlesung/Thema[../@Uhrzeit="15:15 Uhr"]'
    PASSING Vorlesungen) AS X
FROM Vorlesungsverzeichnis
WHERE Semester = 'SoSe 2008'
```

Vorlesungsverzeichnis		
Univ.	Semester	Vorlesungen
Uni LE	SoSe 2008	<VLVerzeichnis ...
Uni LE	WiSe 07/08	<VLVerzeichnis ...
Uni DD	SoSe 2008	<VLVerzeichnis ...

Univ.	X (kein well-formed XML)
Uni LE	<Thema>DBS2</Thema><Thema>eBusiness </Thema>
Uni DD	<Thema>AlgoDat</Thema><Thema>MRDBS</Thema>



XMLEXISTS

- Prädikat zur Bedingungsprüfung auf XML-Dokumenten
 - SQL-WHERE-Klausel kann Eigenschaften der XML-Dokumente verwenden
 - PASSING-Klausel analog zu XMLQUERY
- Beispiel: Welche Universitäten bieten eine DBS2-Vorlesung an?

```
SELECT DISTINCT Universitaet
FROM Vorlesungsverzeichnis
WHERE XMLEXISTS ( ' //Vorlesung[Thema="DBS2"] '
PASSING Vorlesungen )
```



Vorlesungsverzeichnis		
Univ.	Semester	Vorlesungen
Uni LE	SoSe 2008	<VLVerzeichnis ...
Uni LE	WiSe 07/08	<VLVerzeichnis ...
Uni DD	SoSe 2008	<VLVerzeichnis ...

Univ.
Uni LE



XMLTABLE

- Konvertierung eines XML-Anfrageergebnisses in Relation
 - Verwendung mehrerer XPath-Ausdrücke
- Aufbau
 - XPath-Ausdruck zur Selektion einer Menge von Basiselementen
 - je ein Datensatz pro Basiselement in resultierender Relation
 - Für jedes Attribut der Relation jeweils ein XPath-Ausdruck
 - Auswertung des XPath-Ausdrucks relativ vom Basiselement
- Einsatzbereiche
 - Definition (relationaler) Sichten auf XML-Dokumente
 - Nutzerdefinierte Speicherung von XML-Daten in relationales Modell
 - Kombination von XML-Daten mit relationalen Daten (z.B. JOIN)



XMLTABLE (2)

■ Beispiel: Relationale Darstellung des Vorlesungsverzeichnis

Vorlesungsverzeichnis		
Uni	Sem	Vorlesungen
..	..	<VLVerzeichnis> <Vorlesung Uhrzeit="15:15 Uhr"> <Thema>DBS2</Thema> <Dozent> <Name>Rahm</Name> <Einrichtung>IfI</Einrichtung> </Dozent> </Vorlesung> <Vorlesung Uhrzeit="11:15 Uhr"> <Thema>DatenInt</Thema> <Dozent> <Name>Thor</Name> <Einrichtung>IfI</Einrichtung> </Dozent> </Vorlesung> </VLVerzeichnis>

```

SELECT X.*
FROM Vorlesungsverzeichnis,
XMLTABLE (
  '//Vorlesung' PASSING Vorlesungen
  COLUMNS
    "THEMA" VARCHAR(255) PATH 'Thema',
    "ZEIT" VARCHAR(255) PATH '@Uhrzeit',
    "DNAME" VARCHAR(255) PATH 'Dozent/Name'
) AS X
    
```



Thema	Zeit	DName
DBS2	15:15 Uhr	Rahm
DatenInt	11:15 Uhr	Thor



DB2: Relationale Speicherung von XML

■ Anwendungsbezogene Dekomposition von XML

- Nutzer/Administrator definiert „Aufteilung“ der XML-Daten in relationale Struktur
- Realisierbar mittels XMLTABLE oder Schema-Annotationen

■ XMLTABLE

- Dekomposition durch INSERT ... SELECT ... FROM XMLTABLE-Anfragen
- Flexibler Ansatz zur schrittweisen Speicherung von XML
- Pro Relation eine Anfrage: mehrfacher Durchlauf der XML-Daten erforderlich

■ Schema-Annotationen

- XML-Schema wird um DB-spezifische Mapping-Regeln ergänzt (vgl. Hibernate)
- „Befüllung“ aller Relationen in einem Durchlauf
- Annotation ist u.U. aufwändig und unübersichtlich



Zusammenfassung

- XML-Anfragesprachen: XPath, XQuery
- XPath: einfache Auswahl über Pfadausdrücke
 - keine vollwertige Anfragesprache (Verbund, Sortierung, Neustrukturierung, ...)
- XQuery: komplexe Anfragesprache
 - basierend auf Sequenzen, unterstützt Typisierung
 - großer Funktionsumfang (FLWOR-Ausdrücke)
- XML-Datenbanken
 - unterschiedliche Speicherformen: Speicherung als Ganzes, Dekomposition, hybride Speicherung
 - Generisches vs. anwendungsspezifisches Mapping von XML-Strukturen in Tabellen (Dekomposition / Shredding)
 - SQL:2003 unterstützt UDT XML mit Funktionen zum Speichern und Abfragen von XML



Vorschau WS2008/09

- DBS1
- IDBS1: Implementierung von Datenbanksystemen
 - u.a. Datenpufferung, Indexstrukturen, Join-Verarbeitung
- Bio-Datenbanken
- Geoinformationssysteme 2
- Data Warehouse Praktikum (MS SQL-Server 2005)
 - Entwurf Data-Warehouse-Schema
 - Datentransformation (ETL) und –cleaning
 - Datenanalyse (OLAP)
 - Data Mining

