

# 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*
- *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
  - XML-Verarbeitung oft noch 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 und XPointer
  - Navigation und Selektion von Teildokumenten
  - keine vollständige Anfragesprache
- „richtige“ Anfragesprache: W3C XQuery



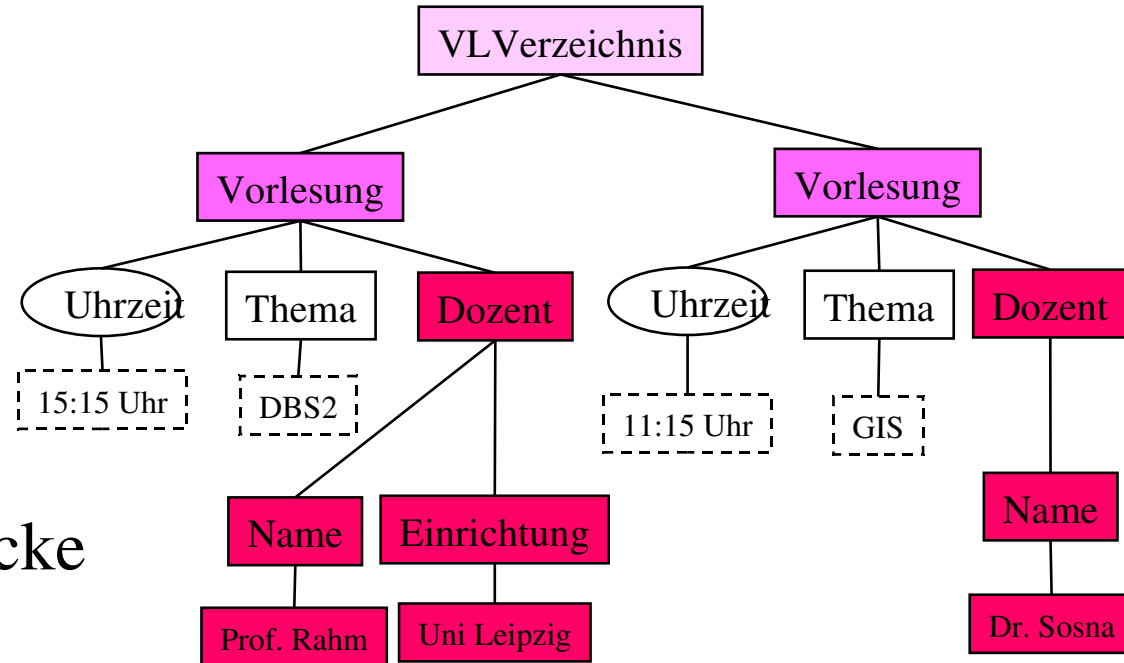
# XPath 1.0\*

- 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

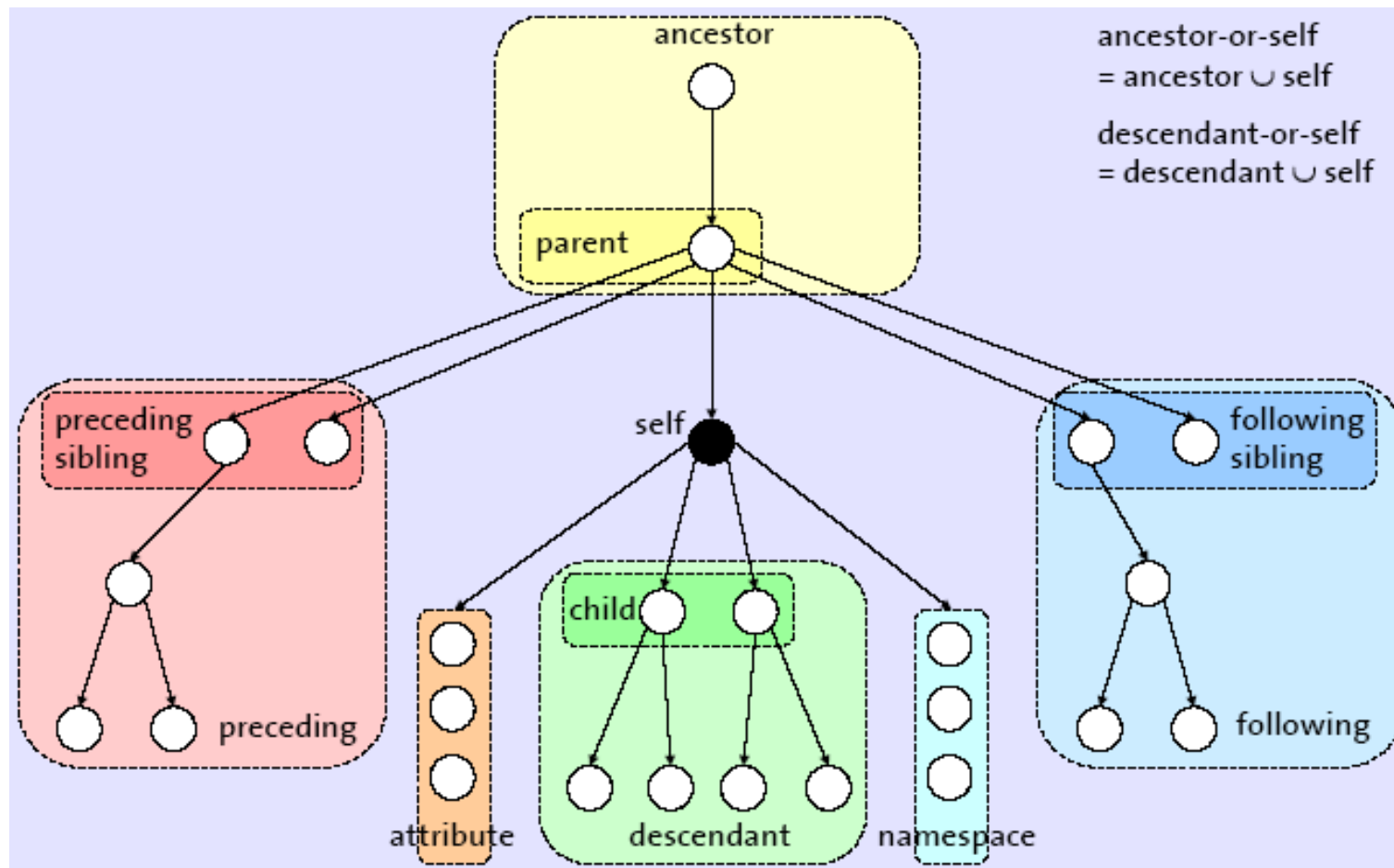
- Syntax eines Schrittes: **Achse::Knotentest [Prädikat]**

\* <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*()





# XQuery-Trainer

Hilfe Tutorial vorhandene Sprachen: | de | en |

## Anzahl der Resultate

30

## Ausgabe

- Resultat in TextArea
- Resultat als HTML (Transformation mit XSLT)
- Resultat als HTML formatiert (SAX mit Whitespaces)
- Resultat als HTML(SAX mit Whitespaces (pretty print))

## vorhandene Dokumente

SigmodRecord.xml  
europe.xml  
VLVerzeichnis.xml  
europe.dtd

```
doc("VLVerzeichnis.xml")//Vorlesung[Dozent/Name="Prof. Rahm"]
```

XQuery abschicken

```
<Vorlesung Uhrzeit="15:15 Uhr">  
  <Thema> DBS2 </Thema>  
  <Dozent>  
    <Name> Prof. Rahm </Name>  
    <Einrichtung> Uni Leipzig </Einrichtung>  
  </Dozent>  
</Vorlesung>  
<Vorlesung Uhrzeit="13:15 Uhr">  
  <Thema> IDBS </Thema>  
  <Dozent>  
    <Name> Prof. Rahm </Name>  
    <Einrichtung> Uni Leipzig </Einrichtung>  
  </Dozent>  
</Vorlesung>
```

2 Resultate vorhanden

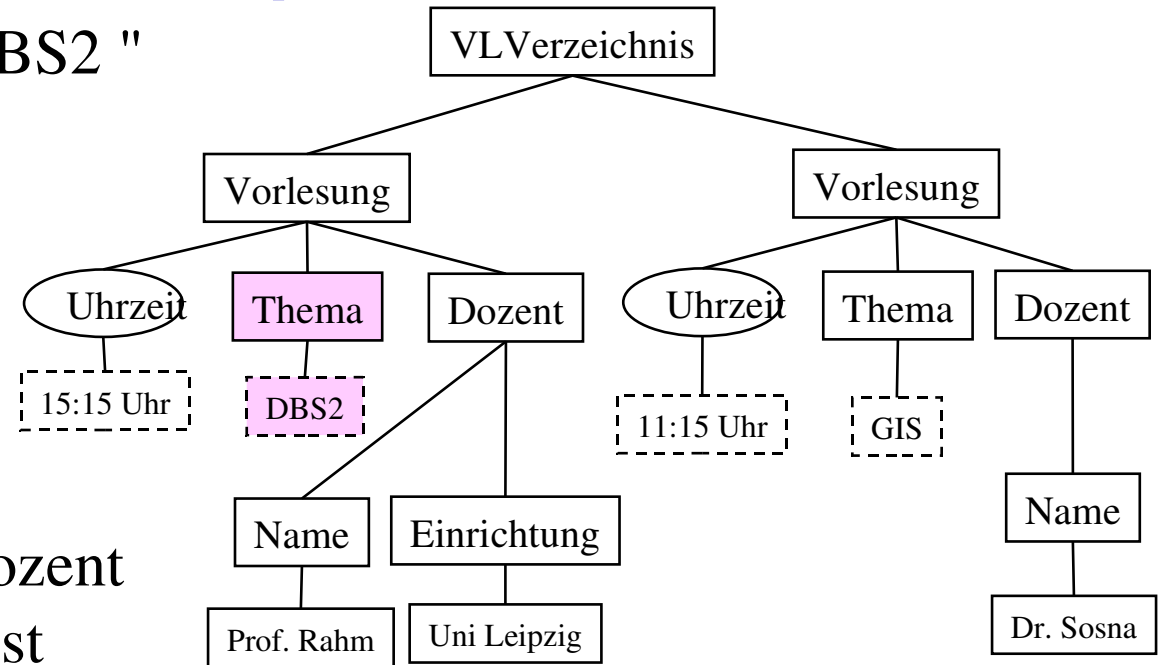




# XPath: Beispiele

- Finde Uhrzeit der Vorlesung "DBS2 "

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



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

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



## XPath: Beispiele (2)

- Wie viele Vorlesungen sind im Verzeichnis enthalten?
- Liste alle Einrichtungen, die den Text "Leipzig" enthalten
- Welche Mitarbeiter sind nach 1975 geboren?
- Liste Name und Telefonnummer aller Mitarbeiter

```
<!ELEMENT Mitarbeiterverzeichnis (Mitarbeiter* )>  
<!ELEMENT Mitarbeiter (Name, Lebenslauf, TelNr?)>  
<!ATTLIST Mitarbeiter Geburtsjahr CDATA #REQUIRED >  
<!ELEMENT Name (#PCDATA)>  
<!ELEMENT Lebenslauf (#PCDATA)>  
<!ELEMENT TelNr (#PCDATA)>
```



# Beispiel XPath/XQuery-Trainer

```
doc("Mitarbeiterverzeichnis.xml")//Mitarbeiter[@Geburtsjahr > 1975]
```

XQuery abschicken

```
<Mitarbeiter Geburtsjahr="1981">  
  <Name> Silke Neumann </Name>  
  <Lebenslauf> Lebenslauf von Silke Neumann. </Lebenslauf>  
  <TelNr> 23458 </TelNr>  
</Mitarbeiter>  
<Mitarbeiter Geburtsjahr="1982">  
  <Name> Jens Richter </Name>  
  <Lebenslauf> Lebenslauf von Jens Richter. </Lebenslauf>  
  <TelNr> 34577 </TelNr>  
</Mitarbeiter>  
<Mitarbeiter Geburtsjahr="1982">  
  <Name> Christiane Schröder </Name>  
  <Lebenslauf> Lebenslauf von Christiane Schröder. </Lebenslauf>  
</Mitarbeiter>  
<Mitarbeiter Geburtsjahr="1984">  
  <Name> Torsten Schiller </Name>  
  <Lebenslauf> Lebenslauf von Torsten Schiller. </Lebenslauf>  
</Mitarbeiter>
```

4 Resultate vorhanden



# 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\*

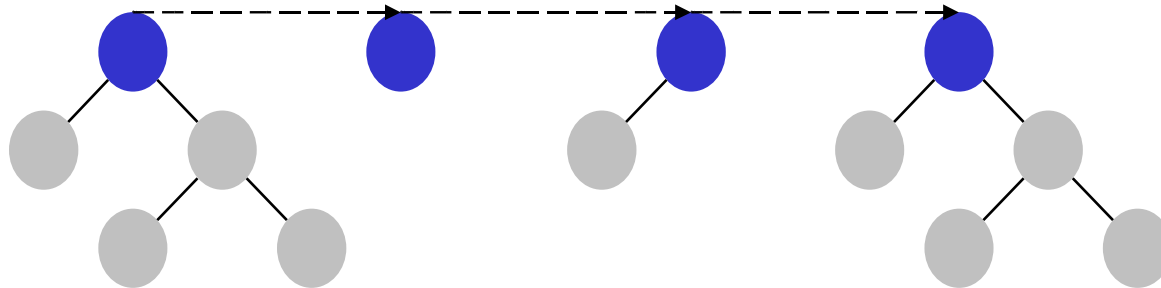
- XQuery: W3C-Standardisierung für einheitliche XML-Anfragesprache
- abgeleitet von vorangegangenen proprietären XML-Anfragesprachen (Quilt, XPath, XQL, XML-QL, ...) sowie SQL und OQL
- **FLWOR** („flower“) –**Syntax**:  
For ... **L**et ... **W**here ... **O**rders By ... **R**eturn)
- Weitere Eigenschaften
  - 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

\* <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.: <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) OrderSpecList  
OrderSpecList ::= OrderSpec (, OrderSpec)*  
OrderSpec   ::= Expr OrderModifier  
OrderModifier ::= (ascending | descending)? (empty greatest | empty least)? (collation StringLiteral)?
```

- For/Let: Variablenbindung an Datenquellen / Sequenzen
- Where: Auswahlbedingung
  - wird für jeden Wert der Variablenbindungen ausgewertet
  - bei mehreren Variablen erfolgt Anwendung auf jeden Wert des kartesischen Produkts
- 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)
- 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. 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
- Auswertung des Ausdrucks liefert 'falsch' bei:
  - einer leeren Sequenz
  - dem booleschen Wert 'false'
  - einem 0 Zeichen langen Wert des Typs xs:string oder xdt:untypedAtomic
  - einem numerischen Wert gleich 0
  - dem Wert NaN (Not a Number) der Typen xs:double oder xs:float

```
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

```
ORDERBY-expr ::= (order by | stable order by) OrderSpecList
OrderSpecList ::= OrderSpec (, OrderSpec)*
OrderSpec ::= Expr OrderModifier
OrderModifier ::= (ascending | descending)? (empty greatest | empty least)? (collation StringLiteral)?
```

- 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
```

- *empty greatest / empty least* spezifiziert, wie nicht vorhandene Werte bzw. leere Sequenzen einsortiert werden
- *collation*: URI-Argument spezifiziert Zeichenordnung  
anderenfalls Sortierung entsprechend Unicode-Wert

```
for $i in ("a", "ä", "A", "Ä", "s", "S", "B")
order by $i
return $i
```



```
A S a s Ä B ä
```



# 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

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



```
<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>
```

- **berechnete Konstruktoren** (computed constructor)

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

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

```
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

## ■ Kommentare

- werden durch (: und :) begrenzt und können geschachtelt werden  
z. B. (: Kommentar 1 (: eingeschachtelter Kommentar :) ... :)





# 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, \$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: 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
- können in Module ausgelagert werden
- Funktion muss Namensraum zugeordnet werden
  - Funktionsaufruf ohne Namensraumangabe verwendet Namensraum der XQuery-Funktionen

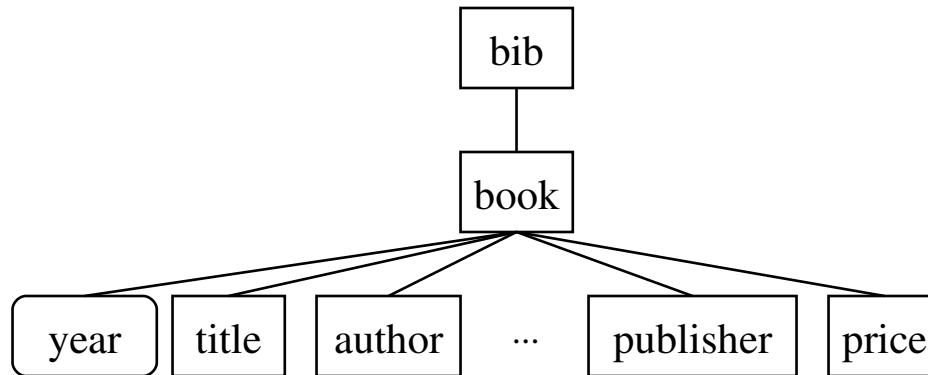
```
declare namespace dbs = "http://dbs.uni-leipzig.de/ns/dbs";  
declare function dbs:authorCount ( $p as element() ) as xs:integer {  
    count($p//author)  
};  
for $e in doc("bib.xml")//book  
where dbs:authorCount($e) > 2  
return $e/title
```



# XQuery: Beispielschema

- folgendes Schema liegt den folgenden XQuery-Beispielen zugrunde (aus XQuery Use cases)
- das Dokument dazu 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>

```



# Klassifikation von XML-Dokumenten

## ■ breites Anwendungsgebiet für XML

- große Varianz in Dokumenteigenschaften (strukturiert, semi-strukturiert, Textdokumente)
- Auswirkung auf Verarbeitung der Daten

## ■ Klassifikation von XML-Dokumenten (semi-strukturiert)

	datenorientiert	dokumentorientiert	gemischt strukturiert
Struktur	feingranular strukturiert	grobgranular und unstrukturiert	enthält sowohl datenorientierte als auch dokumentorientierte Eigenschaften
Schema	explizit vorgegeben	meist kein explizites Schema	
Typ-Information	getypt	ungetypt	
Elementordnung	nicht signifikant	signifikant	
Elementinhalt	element content	mixed content	
Operationen	Zugriff/Änderung einzelner Dokumentbereiche	Zugriff auf komplette Dokumente (originalgetreue Wiederherstellung notwendig)	
Beispiele	Bestellungen, Rechnungen, XML Darstellung relationaler Daten	wissenschaftliche Artikel, Geschäftsberichte	Online-Buchladen (datenorientiert: Titel, Verlag, usw; dokumentorientiert: Inhaltsangabe, Rezensionen), Produktkataloge



# DB-Speicherungsverfahren für XML-Dokumente

	Speicherung als Ganzes	Dekomposition		Hybride Speicherung
		generisch	schemabasiert	
Speicherung	komplett (nicht zerlegt) in Datei oder DBS (Attributtyp: CLOB, BLOB, XML-Type)	zerlegt bzgl. eines generischen Modells (z.B. Graphmodell, DOM)	anwendungsbezogene strukturierte Speicherung in DB (automatisches/manuelles Mapping)	meist anwendungsbezogene Kombination verschiedener Speicherungsverfahren
Zugriff	- Indizes - Parsen des Dokumentes	- Indizes - Pfadnavigation	auf spezifische DB-Struktur	
Vorteile	- originalgetreue Dokumentrekonstruktion - sehr schneller Zugriff auf komplettes Dokument - sehr schnelles Einbringen der Daten	- nur ein Parsingdurchlauf beim Laden		- Anpassung auf unterschiedliche Eigenschaften innerhalb eines Dokumentes
		- einfache Änderung von Dokumentteilen - schemaunabhängig	- performanterer Zugriff/Änderung einzelner Daten	
Nachteile	- hoher Zeitaufwand zum Parsen - hoher Änderungsaufwand	- hoher Aufwand zur Rekonstruktion von Dokumentfragmenten - oft keine originalgetreue Rekonstruktion		- höherer Verwaltungsaufwand
			- pro Schema muss Mapping erstellt werden	
geeignet für XML Dok.	dokumentenorientierte		datenorientierte	
	gemischt strukturierte			



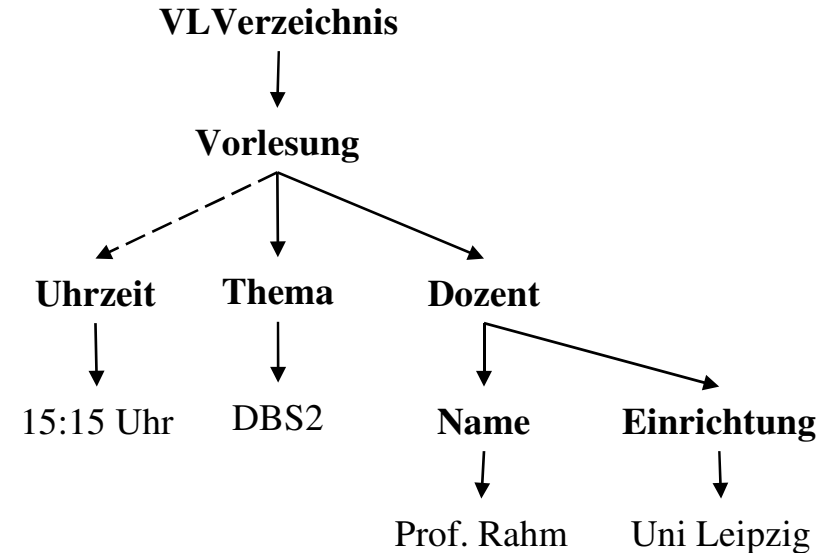
# Speicherung in ORDBS

- Speicherung als Ganzes: in BLOB/CLOB bzw. UDT

- Anlegen von Indizes für ausgewählte Elemente/Attribute
- Zugriff über UDFs

- generische Dekomposition

- Abbildung Hierarchie (XML-Instanz) bzw. Graph (XML Schema) auf Relationen mittels Mapping
- modellbasiertes Mapping: Überführung in generisches Datenmodell, welches auf ein Relationenschema abgebildet wird



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

docID	attributname	elementID	wert
1	Uhrzeit	2	15:15

Attribute

Elemente



# Speicherung in ORDBS (2)

## ■ schemabasierte Dekomposition

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

VLVerzeichnis	Vorlesung						
VLVerzeichnigID	Vorlesungl	VorlesungPo	VlVerzeichnisl	Uhrzeit	UhrzeitPo	Them	ThemaPos
1	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	Vorlesung				
VLVerzeichnigID	VorlesungsID	Uhrzeit	Thema	Dozent	
1	1	15:15	DBS2	Name	Einrichtung
				Prof. Rahm	Uni Leipzig

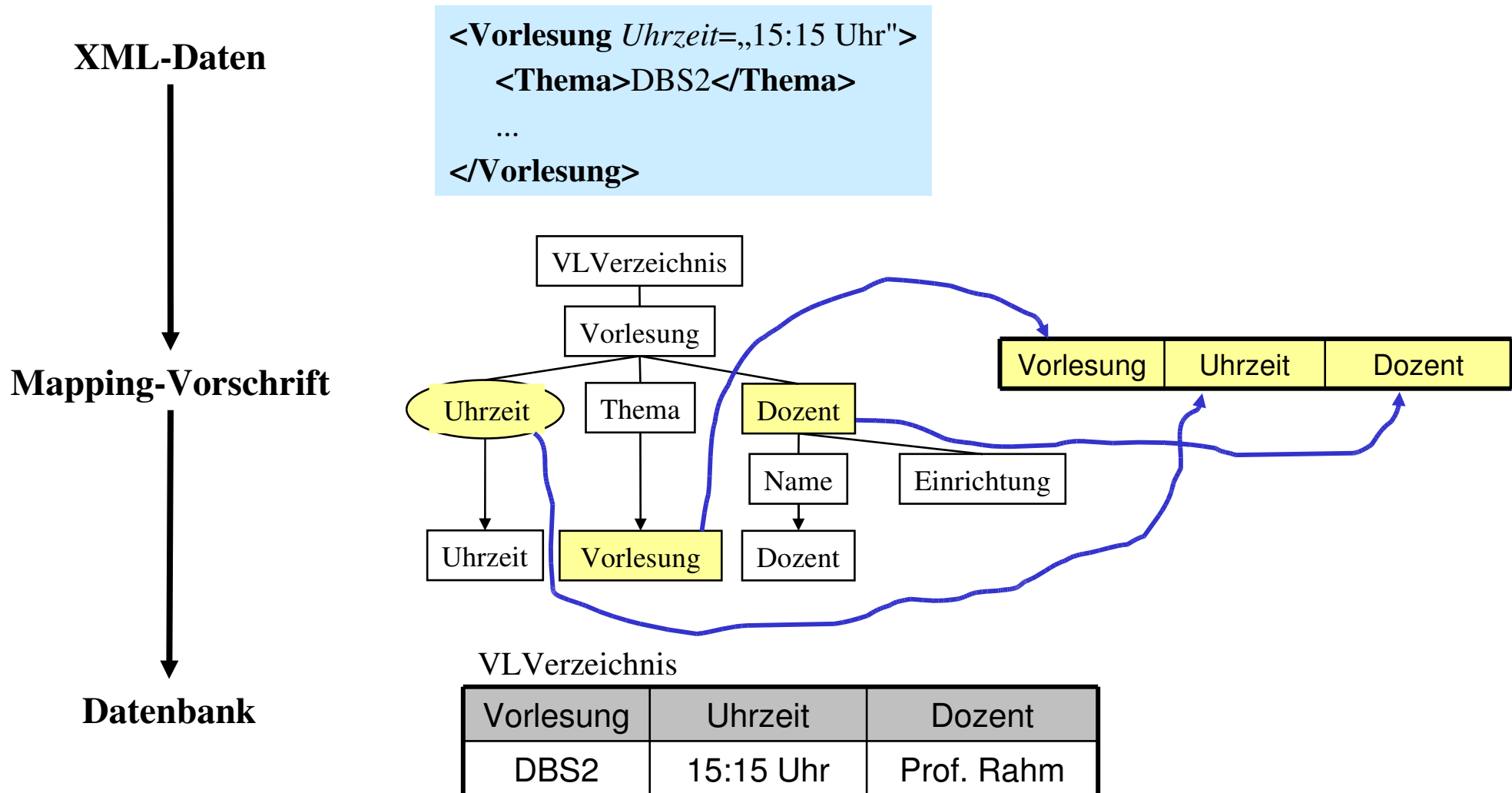
\* <http://www.rpbouret.com/xmldbms/>



# Speicherung in ORDBS (3)

## manuelles Mapping

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



# XML-Unterstützung in kommerziellen DBS

## ■ 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 ...

## ■ DBS mit XML-Erweiterung

- DBS mit XML-unabhängigen Schnittstellen und Datenmodell (z. B. (objekt)relational, objektorientiert)
- Datenzugriff vorrangig über Anfragesprache des DBS
- enthält Erweiterungen zur Transformation zwischen XML-Dokumenten und internen Datenstrukturen (z. B. relationale DBS mit UDTs und UDFs zum Speichern und Abfragen von XML-Dokumenten)
- vorrangig für datenorientierte XML-Dokumente optimiert
- Bsp.: Oracle, IBM DB2, Microsoft SQL-Server



# 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 stellt hohe Anforderungen an Datenbanksysteme
  - unterschiedliche Arten von XML-Dokumenten (datenorientiert, dokumentorientiert, gemischt strukturiert)
  - XML-Datenbanken unterstützen unterschiedliche Speicherformen für verschiedene Dokumenttypen: Speicherung als Ganzes, Dekomposition, hybride Speicherung
  - Generisches vs. anwendungsspezifisches Mapping von XML-Strukturen in Tabellen (Dekomposition / Shredding)

