

# **XML-Anfragesprachen (Schwerpunkt XQuery)**



# Übersicht

---

- Anforderungen an XML-Anfragesprachen
- Stand der Entwicklung
- XPath
- Anfragesprache XQuery
- XML und Änderungsoperationen
- XML und SQL
- Ausblick

# Allgemeine Anforderungen

- **Ad-hoc-Formulierung**
  - Anfragen direkt, ohne Erstellung kompletter Programme
- **Deskriptivität**
  - Umschreiben des gewünschten Ergebnisses, nicht Konstruktionsweg
- **Kompaktheit**
  - kompakte Notation, verwendbar in URLs
- **Mengenorientiertheit**
  - Anfragen auf Mengen von Objekten und Dokumenten
  - Nicht nur auf einzelnen Objekten und XML-Dokumenten oder –fragmenten operieren

# Allgemeine Anforderungen (2)

- **Adäquatheit**
  - alle Konstruktoren des zugrundeliegenden Modells unterstützt
- **Orthogonalität**
  - Sprachkonstrukte uneingeschränkt miteinander kombinieren
- **Abgeschlossenheit**
  - Resultat (XML) kann wieder als Eingabe (XML) einer nächsten Anfrage dienen
- **Vollständigkeit**
  - gemäß Datenmodell gespeicherte Informationen lassen sich verlustfrei durch Query wiedergewinnen

# Allgemeine Anforderungen (3)

- **Optimierbarkeit**
  - kleine Menge an Grundoperationen mit Optimierungsregeln
- **Effizienz**
  - jede Operation wird durch Algorithmus effizient umgesetzt
- **Sicherheit**
  - syntaktisch korrekte Query liefert endliche Menge,
  - Anfrage terminiert
- **Eingeschränktheit**
  - keine vollständige Programmiersprache
  - Garantiert Optimierbarkeit, Effizienz und Sicherheit

# XML-Anforderungen (1)

- **Einbettung**

- Anfragen in XML einbetten oder als XML formulieren
- XML-Fragmente in Anfrage → fester Teil in Ergebnis

- **Server-Verarbeitung**

- Geeignet für Server-side Processing → erfordert Abgeschlossenheit, Kontextfreiheit

- **Ordnungserhaltung (Order Preserving)**

- Abfolge von Elementen, Schachtelungsreihenfolge
- Vor allem bei mixed Content
- Nicht für alle Anwendungen benötigt (mögliches Abschalten aus Effizienzgründen)
- Probleme bei relationaler Speicherung

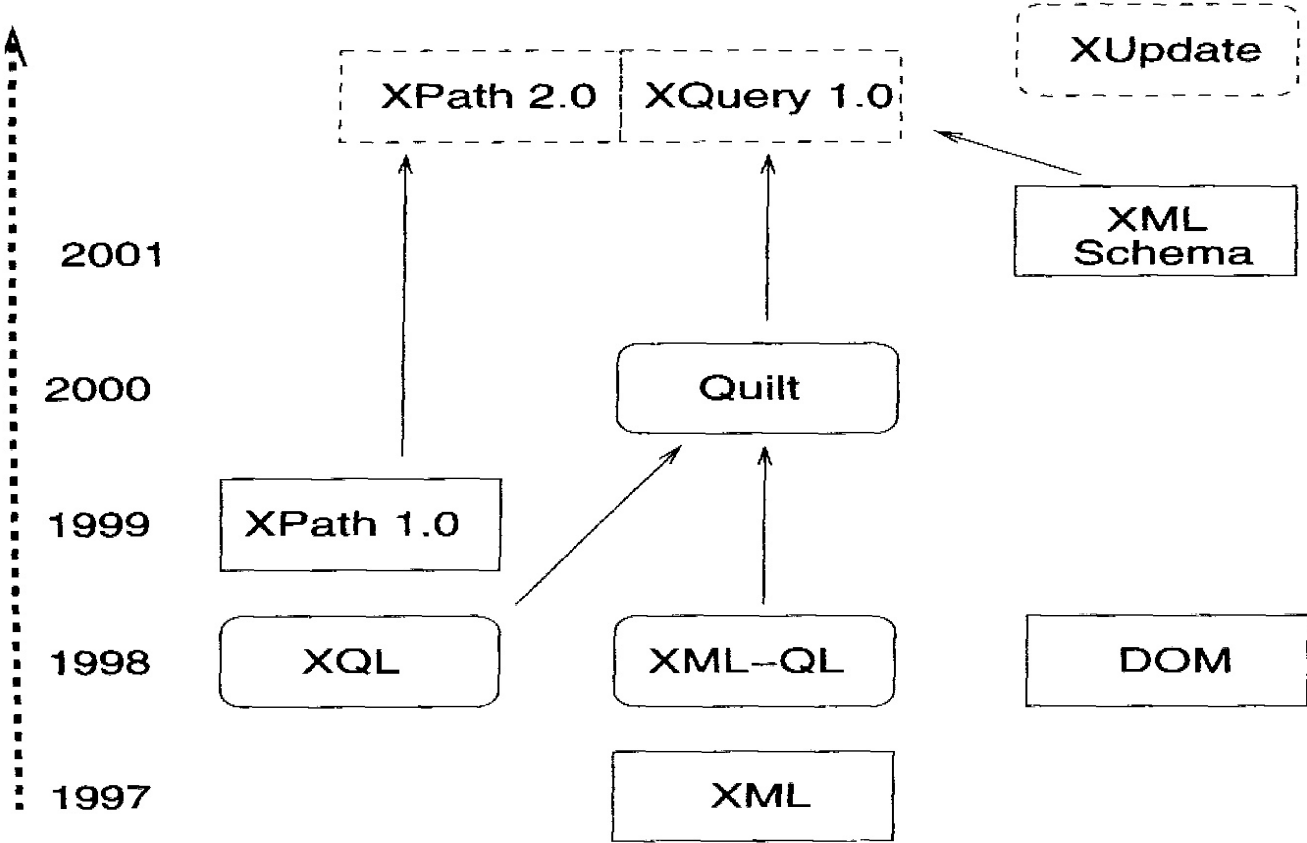
# XML-Anforderungen (2)


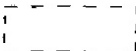

- **Hyperlinks**
  - Ausnutzen und Verfolgen von Links in ( ID/IDREF ) und zwischen ( XLink, XPointer ) Dokumenten
- **Flexible Typen**
  - Verarbeiten von unbekanntem oder wechselnden Typen
  - Mechanismen zur expliziten Typkonvertierung
  - Unterstützung eines erweiterbaren und robusten (fehlertoleranten) Typkonzepts
- Unterstützung des Zugriffs auf Metadaten und Schemainformationen (gleiche Mittel)
- Berücksichtigung von Namensräumen
- Protokoll-Unabhängigkeit

# Grundoperationen

- Selektion (anhand von Inhalt, Struktur, Attributwert)
- Extraktion und Reduktion (vergleichbar Projektion)
- Kombination (Join)
- Restrukturierung von Elementen
- Aggregation (z.B. arithmetische Funktionen)
- Gruppierung (Schachtelung bzw. Entschachtelung von Elementstrukturen)
- Volltextoperationen (literal bzw. Phrasen-, Stammformsuche, etc.)
- Datenmanipulation
  - über APIs wie DOM, SAX
  - Auch Änderungsoperationen über XML-Anfragesprache

# Stand der Entwicklung



-  W3C Empfehlungen
-  noch in der Entwicklung
-  andere Vorschläge

# XPath 1.0 - Einführung

- W3C Recommendation vom Nov 1999
  - siehe: <http://www.w3.org/TR/xpath>
- Grundlage für XQuery, XSLT, weitere Standards
- geht von abstrakter Baumstruktur des XML-Dokuments aus
- dient zur Adressierung von Teilen eines Dokumentes
- kompakte Nicht-XML-Syntax, keine vollständige QL
- zur Selektion und Extraktion von Knotenmengen

# XPath 1.0 - Einführung

- **Knotenarten**

- 7 Knotenarten auf Basis des XML Information Set – ähnlich zu DOM
- Wichtigste: Wurzel, Element, Attribut, Text

- **Datentypen**

- atomare Werte: boolean, number, string
- Knotenmengen (node-set)

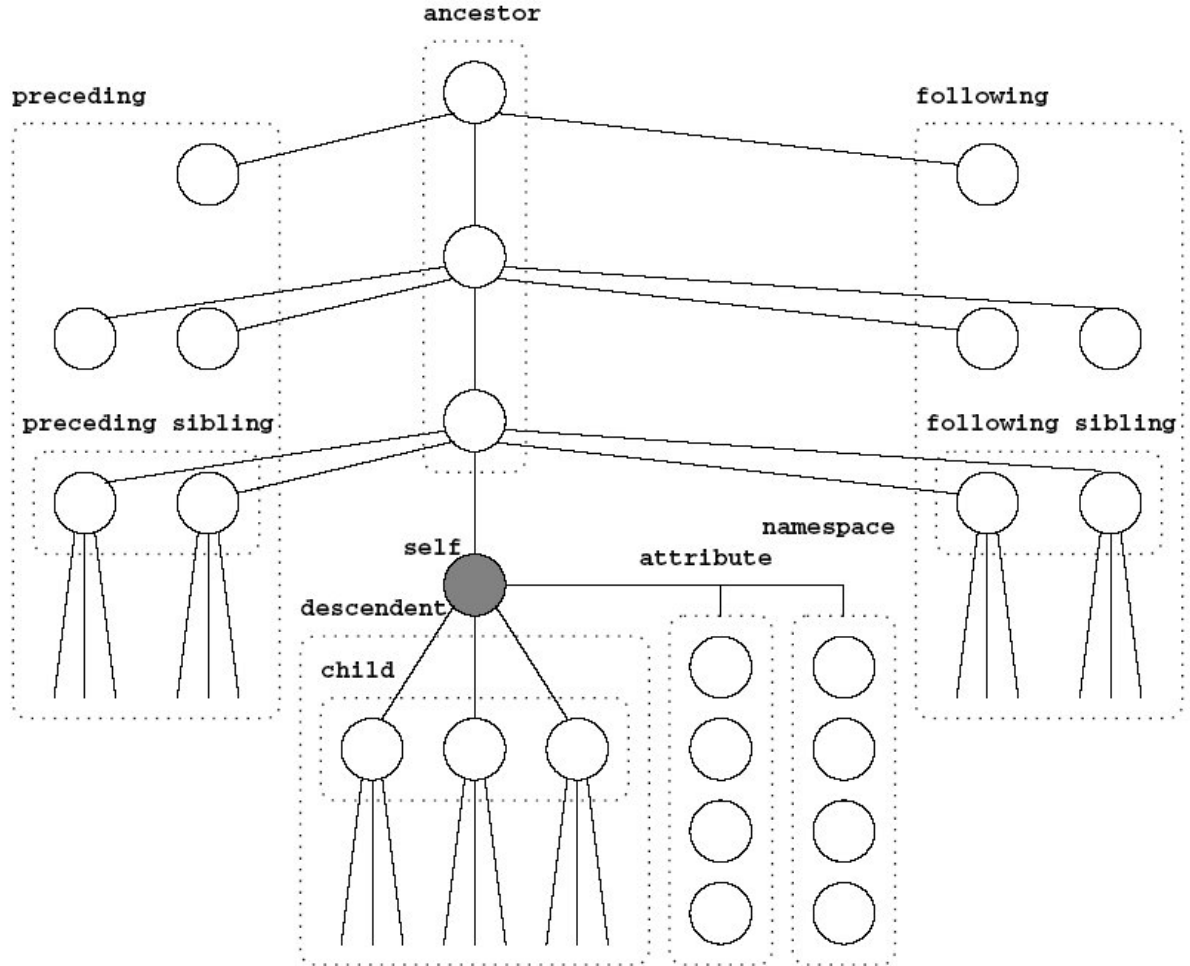
- **Grundlegendes Konstrukt sind XPath-Ausdrücke**

- Pfadausdrücke (location paths)
- logische und mathematische Verknüpfungen
- Funktionsaufrufe
- Relativ zu einem Kontext ausgewertet
- Keine Variablenbindung

# XPath 1.0 - Pfadausdrücke

- XPath-Ausdrücke relativ zu einem Kontext ausgewertet
- Extraktion der interessierenden Bestandteile
- Selektion von Knotenmengen aufgrund ihrer Struktur und den in ihnen enthaltenen Werten
- Formulierung von Bedingungen an diese Knotenmenge
- mehrere Steps, Kopplung mit “/“
- Schritt (Location Step) *axis::node-test[predicate]*
  - axis: Beziehung Kontextknoten u. zu selektierende Knoten
  - node-test: Knotentyp und Namen des zu sel. Knoten
  - predicate: Einschränkung best. Elemente durch Prädikate

# Navigationsachsen in XPath



Klettke/Meyer "XML & Datenbanken"

# Knotentest in XPath

## Einschränken des Knotentyps

- *node()*: alle Knoten
- *text()*: alle Textknoten
- \*: alle Elementknoten
- *comment()*: Kommentarknoten
- Angabe eines Knotennamens

## Beispiele

*descendant::\** alle untergeordneten Elemente des Kontextknotens

*child::href* alle Elemente vom Typ href

*attribute::id* Attribut id des Kontextknotens

Kürzel	Langform	Bedeutung
	<i>child::</i>	<i>child</i> ist die Standardachse
.	<i>self::node()</i>	bezeichnet den aktuellen Knoten, den Kontextknoten
..	<i>parent::node()</i>	bezeichnet den Vaterknoten
/		bezeichnet den Wurzelknoten oder trennt Pfadkomponenten
//	<i>/descendant-or-self::node()/</i>	bezeichnet Nachkommen des Kontextknotens
@	<i>attribute::</i>	bezeichnet Attribute des Kontextknotens
*		bezeichnet einen Knoten mit einem beliebigen Namen
@*		bezeichnet alle Attribute des Kontextknotens
[ <i>expr</i> ]		bezeichnet ein Prädikat für den aktuellen Teilpfad, <i>expr</i> ist ein boolescher Ausdruck
[ <i>n</i> ]		wenn das Prädikat aus einer natürlichen Zahl <i>n</i> besteht, bezeichnet es das <i>n</i> -te Element aus der Liste von Knoten, für die der aktuelle Teilpfad gilt
	Quelle ebd.	

# Prädikate in XPath

## Selektionsprädikate

- logische Operatoren (*and*, *or*)
- Vergleichsoperatoren ( "<" , "<=" , ">" , ">=" , "=" , "!=" )
- Operationen auf numerischen Werten ( "+" , "-" , "\*" , *div*, *mod* )
- Vereinigung von Knotenmenge ( "|" )
- Ändern der Auswertreihenfolge durch Klammerung von Teilausdrücke mit "(" und ")" )

## Beispiele

- `/descendant-or-self::node()/album/song[2]`
- `//album/song[last()-1]/title`
- `//hotel[zimmertyp/attribute::typ = 'Doppelzimmer']`
- `/hotel/adresse/ort[@Name="Leipzig"]`

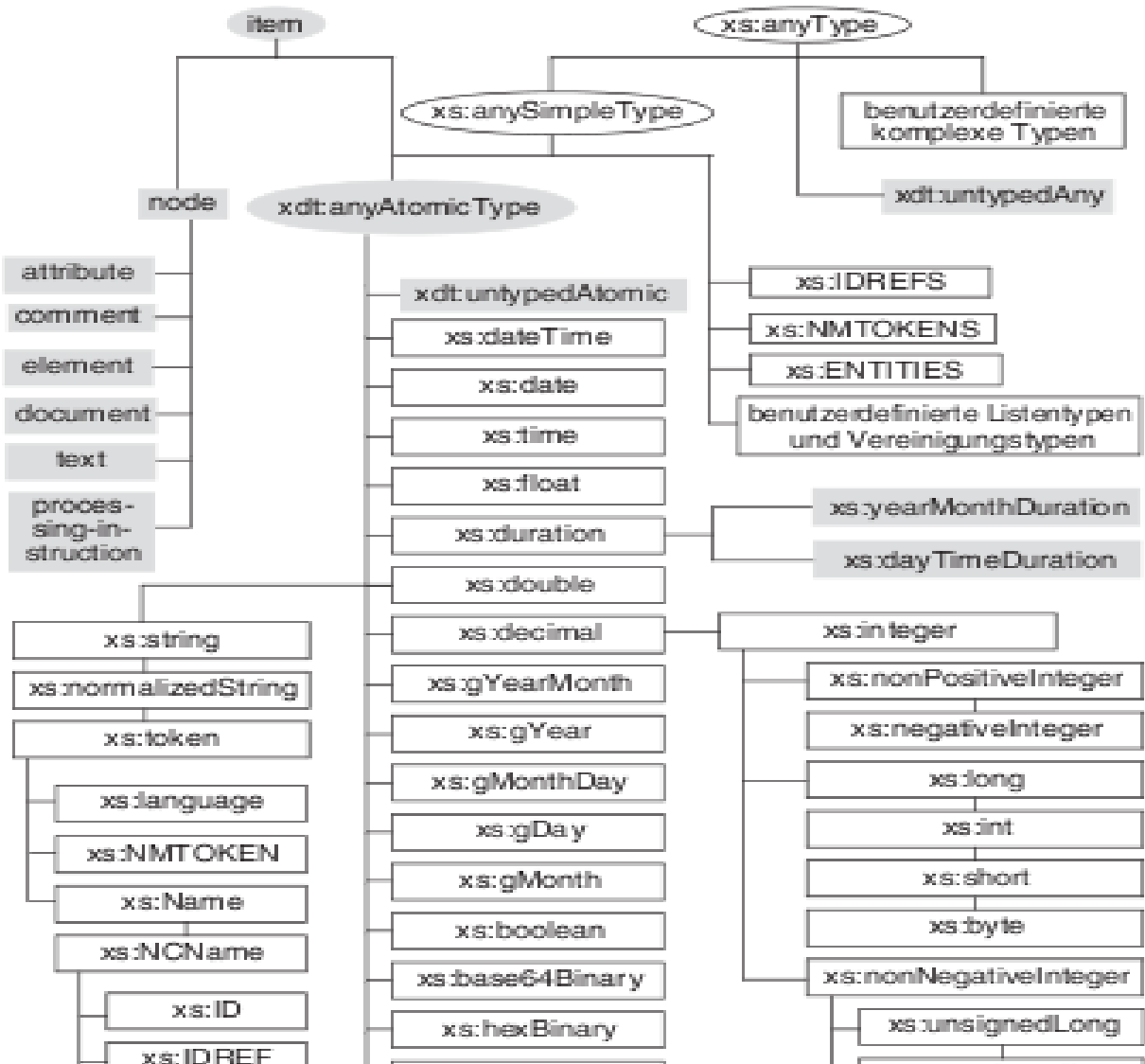
# XQuery

- Überblick und Datenmodell
- Einfache Ausdrücke
- FLWOR-Ausdrücke
- Erweiterte XQuery-Ausdrücke:
  - Verbund
  - Gruppierung
- Benutzerdefinierte Funktionen
- Erweiterte Konzepte
  - Modulkonzept
  - XQuery Prolog
- Verarbeitungskonzept
- Implementierungen

# Sprachüberblick

- W3C Recommendation vom Januar 2007 - (siehe: <http://www.w3.org/TR/xquery>)
- geht aus Quilt hervor (inoffizieller Vorschlag des W3C), basierend auf XPath, beeinflusst durch SQL, XML-QL
- Ausdrücke beliebig ineinander schachtelbar
  - Elementkonstruktoren
  - Pfadausdrücke zur Selektion (wie XPath)
  - FLWOR-Ausdrücke (ähnlich SQL - SFW)
  - datenspezifische Operatoren, standard- und selbstdef. Funktionen
  - bedingte Anweisungen, Test von Datentypen, Typumwandlung
  - Verwendung von Quantoren: every und some

# Typhierarchie



# Grundlagen

- Konstanten oder Literale
  - Werte für einfache numerische und Zeichenkettenwerte (in Hochkommas)
  - Beispiele:
 

<code>'Hotel Neptun', "12.34"</code>	<code>{-- xs:string --}</code>
<code>123, -24, 0, +7</code>	<code>{-- xs:integer --}</code>
<code>-24.0, 123.45, +.23</code>	<code>{-- xs:decimal --}</code>
<code>-123.5e3, 200e6</code>	<code>{-- xs:double --}</code>
- Variablen:
  - Beispiele:
 

```
$name := "Wolfgang Lehner"
let $hotel := //hotel[typ='Pension']
return $hotel
```
- Kommentare
  - An beliebiger Stelle, ohne Bedeutung
  - Syntax: geschachtelt durch ein Klammerpaar (: und :)
  - Beispiel: `(: XQuery (: Kommentar :) !! :)`
  - Orthogonal zu einem Kommentar im XML-Dokument
 

```
<!-- - Kommentar im XML-Dokument - -!>
```

# Ausdrücke in XQuery

- **Arithmetische Ausdrücke**
  - Funktionen und Operatoren
- **Vergleichsausdrücke**
  - Wertevergleich
  - Allgemeiner Vergleich
  - Knotenvergleich (Knotenidentität, Vergleich der relativen Positionierung)
- **Logische Ausdrücke**
  - fn:true(), fn:false(), fn:not()
- **Pfadausdrücke**
  - Basieren auf XPath 2.0 (Erweiterungen gegenüber 1.0)
- **FLWOR-Ausdrücke**
- **Konditionale Ausdrücke**
- **Quantifizierende Ausdrücke**
  - existenzielle Quantifizierung
  - universelle Quantifizierung

# Einfache Ausdrücke in XQuery

- **Arithmetische Ausdrücke**
  - Mit arithmetischen Operationen
  - Auswertereihfolge durch Klammerung beeinflussen
  - Beispiel: **(`$preis-10`) div 100**
- **Vergleichsausdrücke (3 Vergleichsoperatoren)**
  - allgemein mit Sequenzen: **`=, !=, <, <=, >, >=`**
  - einfache Wertvergleiche zwischen elementaren Werten:  
**`eq, ne, lt, le, gt, ge`**
  - Vergleich der Knotenidentität:  
**`is`** (identisch), **`is not`** (nicht identisch),
- **logische Ausdrücke:**
  - **`and, or, not`**
  - Beispiel: **`not($hotel/name eq $restaurant/name)`**
- **Pfadausdrücke**
  - Basieren auf XPath 2.0 (Erweiterungen gegenüber 1.0)

# Identität vs. Gleichheit (Beispiel)

- Zwei Knoten sind identisch, wenn folgendes gilt:  
\$knoten1 is \$knoten ---> true
- **Beispiel:**

```
<x>
  <titel>Harry Potter und der Stein der Weisen</titel>
  <titel>Harry Potter und der Stein der Weisen</titel>
</x>
```

XQuery:

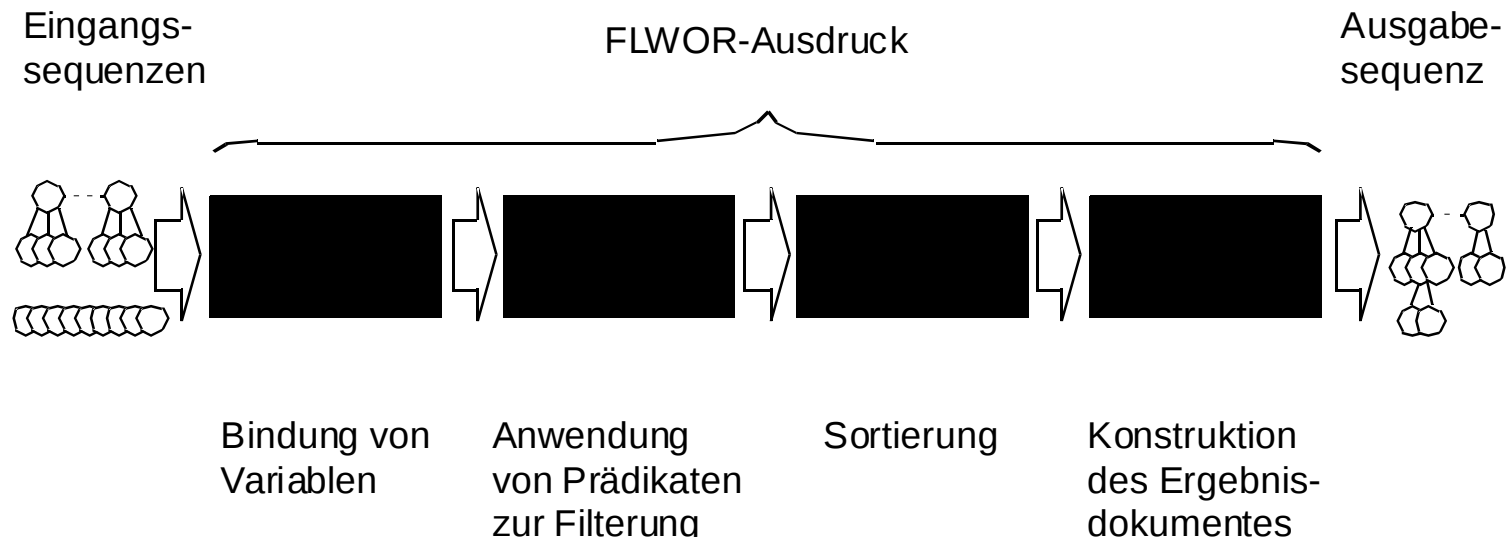
```
let $titel := doc("1.xml")//titel
return
<a>
  <x>{ $titel[1] = $titel[1] }</x>
  <x>{ $titel[1] is $titel[1] }</x>
  <x>{ $titel[1] = $titel[2] }</x>
  <x>{ $titel[1] is $titel[2] }</x>
</a>
```

Ergebnis:

```
<a>
  <x>true</x>
  <x>true</x>
  <x>true</x>
  <x>>false</x>
</a>
```

# FLWOR-Ausdrücke

- FLWOR-Ausdruck
  - gesprochen wie engl. “Flower“
  - Basis für Anfragen an XML-Datenbanken
  - analog zu SELECT-FROM-WHERE in SQL
  - steht als Abkürzung von for-let-where-order by-return
- Klauseln eines FLWOR-Ausdrucks



# Struktur eines FLWOR-Ausdrucks

FLWORExpr ::=  
 (*ForClause* | *LetClause*)<sup>+</sup> *WhereClause*? *OrderByClause*? **return**  
*ExprSingle*

*ForClause* ::=  
**for** *\$VarName* *TypeDeclaration*? *PositionalVar*? **in** *ExprSingle* (,  
*\$VarName* *TypeDeclaration*? *PositionalVar*? **in** *ExprSingle*)<sup>\*</sup>

*LetClause* ::=  
**let** *\$VarName* *TypeDeclaration*? := *ExprSingle*  
 (, *\$VarName* *TypeDeclaration*? := *ExprSingle*)<sup>\*</sup>

*TypeDeclaration* ::= **as** *SequenceType*

*PositionalVar* ::= **at** *\$VarName*

*WhereClause* ::= **where** *Expr*

*OrderByClause* ::=  
 (**order by** | **stable order by**) *OrderSpecList*

*OrderSpecList* ::= *OrderSpec* (, *OrderSpec*)<sup>\*</sup>

*OrderSpec* ::= *ExprSingle* *OrderModifier*

*OrderModifier* ::= (**ascending** | **descending**)? ((**empty greatest**) |  
**empty least**)? (**collation** *StringLiteral*)?

# Variablen

- Nach der erfolgten Bindung sind die Werte nicht mehr änderbar
- Variablenbindung nur innerhalb des aktuellen und aller eingeschlossenen Anfrageausdrücke sichtbar
- Wird Ausdruck verlassen, ist Variable ungebunden
- Zugriff auf ungebundenen Variablen → Ausnahme
- Wird hingegen Variable mehrfach gebunden, ist immer unmittelbar zuvor erfolgte Bindung sichtbar
- Typ einer Variablen ergibt sich aus Bindung
- Variablenbindung erfolgt in for und let-Ausdrücken

# let-Klausel

- Bindet Menge von Werten, die bei Auswertung eines Ausdrucks *expr* entstehen geschlossen an eine Variable *\$var*
- Im einfachsten Fall folgt nach der let-Klausel ein return-Ausdruck
- Liefert den Variableninhalt als Ergebnis
- Beispiel:  
**let \$z := //zimmer typ**  
**return \$z**
- Abarbeitung in 3 Schritten:
- Werte den XPath-Ausdruck **//zimmer typ** aus
- Weise die resultierende Knotenmenge an die Variable **\$z** zu (Variablenbindung)
- Gebe die gesamte Sequenz von Knoten zurück

# for-Klausel

- Für jedes Element der Ergebnismenge erfolgt eine Bindung an `$var`
- Wie bei `let` können mehrere Variablen gebunden werden
- Variablen für alle untergeordneten Ausdrücke sichtbar
- Beispiel:  

```
for $z in //zimmertyp  
return $z
```
- Abarbeitung anders als bei `let`
  - `$z` wird jeweils an Elemente der Sequenz (Auswertung von `//zimmertyp`) gebunden
  - Bindung erfolgt mehrfach, für jeden Zimmertyp genau einmal
  - Nachfolgende Klauseln werden für jede Iteration und Bindung einmal ausgewertet
  - `return` wird für jeden Schritt ausgewertet, Resultat zu einem Zwischenergebnis hinzugefügt
  - Aufsummiertes Gesamtergebnis von `return` wird am Schleifenende zurückgegeben

# Beispiel

```
<hotel name="Hotel Neptun">
  <zimmertyp typ="EZ" preis="180" währung="EUR"/>
  <foto href="neptun01.jpeg"/>
</hotel>

<hotel name="Hotel Hübner">
  <zimmertyp typ="EZ" preis="75" währung = "EUR"/>
  <zimmertyp typ = "DZ" preis = "90" währung = "EUR"/>
</hotel>

<hotel name="Pension Dräger">
  <foto href="bild-pd01.jpeg"/>
  <foto href="bild-pd02.jpeg"/>
</hotel>
```

# Beispiel (Forts.)

---

XQuery-Anfrage

```
for $hotel in //hotel  
return $hotel/foto
```

Ergebnis:

```
<foto href="neptun01.jpeg"/>  
<foto href="bild-pd01.jpeg"/>  
<foto href="bild-pd02.jpeg"/>
```

# Vergleich LET- und FOR-Klauseln

```
for $x in (<Arzt/>, <Pfleger/>)  
let $y := (<Operationssaal/>, <Station/>)  
return  
  (<Berufsgruppe>{ $x }</Berufsgruppe>,  
   <Arbeitsort>{ $y }</Arbeitsort>)
```

liefert als Ergebnis

```
<Berufsgruppe><Arzt/></Berufsgruppe>  
<Arbeitsort><Operationssaal/><Station/>  
</Arbeitsort>  
<Berufsgruppe><Pfleger/></Berufsgruppe>  
<Arbeitsort><Operationssaal/><Station/>  
</Arbeitsort>
```

# Geschachtelte FOR-Klauseln

```
<billighotels> {  
  for $h in //hotel  
    for $z in $h/zimmertyp  
      where $z/@preis <= 100  
        return <hotel>  
          <name>{ data($h/@name) }</name>  
          <preis>{ data($z/@preis) }</preis>  
        </hotel> } </billighotels>
```

Ergebnis:

```
<billighotels>  
  <hotel><name>...</name><preis>...</preis></hotel>  
</billighotels>
```

# Geschachtelte FOR-Klauseln (2)

```
<billighotels> {  
  for $h in //hotel  
  return <hotel name={ $h/@name }> {  
    for $z in $h/zimmertyp  
    where $z/@preis <= 100  
    return <preis>{ data($z/@preis) }</preis>  
  } </hotel> } </billighotels>
```

Ergebnis:

```
<billighotels>  
  <hotel name="Hotel Hübner">  
    <preis>...</preis>  
  </hotel>...  
</billighotels>
```

# where-Klausel

where-Klausel zur Angabe eines Selektionsprädikats mit weitergehenden Filtermöglichkeiten in Bezug auf Gruppen (ähnlich zu HAVING)

Beispiel:

- Suche Pflegepersonal das nicht nach 1974 geboren ist
- Betrachte nur jeden 5. Pfleger
- Suche Pfleger, die 3 oder mehr Zertifikate (zum Nachweis bestimmter Fähigkeiten besitzen)

```
for $p at $i in fn:doc("Klinik.xml")//Pfleger
where
  fn:not($p/Geburtsdatum > xs:date("1974-01-01"))
  and $i mod 5 = 0
  and fn:count($p//Zertifikat) > 2
return
  <Personaleintrag LfdNR={$i idiv 5}>
    {$p/@Station, $p/Name,}
  </Personaleintrag>
```

# order by-Klausel

- Explizite Angabe einer Sortierung für die Elemente einer Sequenz
- Spezifische Sortierordnungen
  - Global im Prolog einer XQuery
  - Lokal zusätzlich in der order by-Klausel
- 2 wichtige Eigenschaften der order by-Klausel
  - Wenn keine eindeutige Reihenfolge, so ist Anordnung der Duplikate implementierungsabhängig, Angabe von **stable** erzwingt die Einhaltung der Dokumentreihenfolge
  - Wenn Eigenschaften, nach denen sortiert wird, nicht existieren: Einordnung dieser Einträge explizit steuern mit Sortiermodifikator **empty greatest** bzw. **empty least**
- Beispiele:

Zusätzlich zum Beispiel für where-Klausel:

```
order by $p//Wohnort empty least,  
fn:get-year-from-date($p/Geburtsdatum) descending
```

Weitere Beispiele bei Verbundoperationen

# Elementkonstruktoren (return)

- Literales XML wird in das Ergebnis übernommen, ohne modifiziert zu werden

```
<zimmer typ="DZ">  
  <ausstattung>TV</ausstattung>  
</zimmer>
```

- XML mit geschachtelten Ausdrücken erlaubt es, Element- und Attributinhalt durch XQuery-Ausdrücke berechnen zu lassen

```
<zimmer typ="{ $z/@typ }">  
</zimmer>
```

- XML mit berechneten Element- und Attributnamen bietet die Möglichkeit, die Bezeichner von XML-Elementen und Attributen durch XQuery-Ausdrücke zu berechnen

```
element { $z } {  
  attribute { $t } { "EZ" },  
  element { $a } { "Minibar" }  
}
```

# Verbund – Verfolgung von Referenzen

Beispiel:

```
for $s in fn:doc("Klinik.xml")//Station
let $p := fn:id($s/@Leitung)
return
  <Station>
    {$s/Name}
    <Leitung>{ $p/Name }</Leitung>
  </Station>
```

Ergebnis:

- Zuordnung der leitenden Person (Name) an jede Station im Krankenhaus

# Gruppierung

Beispiel:

Berechnung des Durchschnittsalters pro Berufsgruppe (Arzt oder Pfleger)

## Variante 1: Gruppierung entlang der XML-Hierarchie

```
<MedizinischesPersonal>
{ for $p in fn:doc("...")//MedizinischesPersonal/*
  let $x := $p//Alter
  return
    element { fn:node-name($p) }
      { <Alter>{ fn:avg($x) }</Alter> }
</MedizinischesPersonal>
```

## Variante 2: Gruppierung nach Wertegleichheit

```
<MedizinischesPersonal>
{ for $b in fn:distinct-values(fn:doc("...")//Beruf)
  let $x := fn:doc("...")//Alter[../Beruf = $b]
  return
    element {$b}
      {<Alter>{ fn:avg($x) }</Alter> }
</MedizinischesPersonal>
```

# Aggregationsfunktionen

- **fn:count()**  
liefert die Anzahl der Elemente der übergebenen Sequenz zurück
- **fn:avg()**  
liefert den durchschnittlichen Wert aller Elemente der übergebenen Sequenz  
**sum(\$arg) div count(\$arg)**
- **fn:max()**  
liefert den wertemäßig größten Wert optional bezüglich einer Sortierordnung zurück
- **fn:min()**  
liefert den wertemäßig kleinsten Wert optional bezüglich einer Sortierordnung zurück
- **fn:sum()**  
liefert den summarischen Wert aller in der Sequenz enthaltenen Elementwerte zurück; wird der zweite Parameter nicht angegeben, so wird der Wert 0.0E0 bei einer leeren Sequenz zurückgeliefert; andernfalls der Wert des zweiten Parameters

# Konditionale Ausdrücke

- Syntax

*if (expr) then expr\_1 else expr\_2*

- Beispiel Hotel-Datenbank:

Gewähre Nachlass von 10%, wenn ein Hotelzimmer länger als 5 Tage belegt wird

```
let $stage := ($r/abreise - $r/anreise),  
    $zimmertyp := fn:id(zimmertyp)
```

```
return
```

```
    if ($stage > 5)
```

```
    then $stage*$zimmertyp/@preis*0.9
```

```
    else $stage*$zimmertyp/@preis
```

- Hinweis: **fn:id** funktioniert nicht überall

# Quantifizierende Ausdrücke

- Syntax

*[some | every ] var in expr\_1 satisfies expr\_2*

- Beispiel 1

Namen der Hotels, die unter anderem auch Einzelzimmer anbieten

```
for $hotel in fn:doc("hotels.xml")//hotel
```

```
where some $z in $hotel/zimmertyp
```

```
    satisfies $z/@typ = "EZ"
```

```
return $hotel/name
```

- Beispiel 2

Suche Appartmenthotels, d.h. Hotels, die ausschließlich Appartements anbieten

```
for $hotel in fn:doc("hotels.xml")//hotel
```

```
where every $z in $hotel/zimmertyp
```

```
    satisfies $z/@typ = "Appartement"
```

```
return $hotel/name
```

# XQuery-Implementierungen

- ❑ »Kweelt« von der University of Pennsylvania, Arnaud Sahuguet, [db.cis.upenn.edu/kweelt/](http://db.cis.upenn.edu/kweelt/)
- ❑ Fatdog Software, Howard Katz, [www.fatdog.com](http://www.fatdog.com)
- ❑ »XML Query Demo«, Microsoft, Michael Rys, [msdn.microsoft.com/xml/](http://msdn.microsoft.com/xml/)
- ❑ Software AG, QuiP, Jonathan Robie, [www.softwareag.com/developer/downloads/default.htm](http://www.softwareag.com/developer/downloads/default.htm)
- ❑ Lucent Bell Labs, Jérôme Siméon, XMLDevConSpring2001
- ❑ GMD Darmstadt, Peter Fankhauser, XMLDevConSpring2001
- ❑ Universität Rostock, Guido Rost, [www.xml-and-databases.com](http://www.xml-and-databases.com)

Aktuelle Liste: <http://www.w3.org/XML/Query#implementations>

# XML-Update

- XML-Update Sprache vom Sep 2000 - ( siehe: [7.1] )
- basiert auf XPath
- beschreibt welche Änderungen in XML-File gemacht werden
- diese Änderungen werden als XML formuliert
- ist kein W3C- oder ISO-Standard, sondern XML:DB Initiative
- praktisch, nicht so gut spezifiziert
- Popularität in einigen Implementierungen gefunden - [7.2]

# Literatur

- [1] “XML&Datenbanken” M.Klettke, H.Meyer, dpunkt.verlag, 2002
- [2] XML-QL: <http://www.w3.org/TR/NOTE-xml-ql>
- [3] XQL: <http://www.w3.org/TandS/QL/QL98/pp/xql.html>
- [4] XPath: <http://www.w3.org/TR/xpath>
- [5] XQuery: <http://www.w3.org/XML/Query>  
<http://www.w3.org/TR/xpath-datamodel/>
- [6] XML/SQL: <http://www.sqlx.org/>  
<http://www.oracle.com/technology/oramag/oracle/03-may/o33xml.html>
- [7] XUpdate: <http://xmldb-org.sourceforge.net/xupdate/index.html>  
<http://uche.ogbuji.net/tech/akara/nodes/2004-09-30/xupdate>
- [8] “XML – Von Anfang an”, rororo, 2003
- [9] “Essential XML”, D.Box, A.Skonnard, Addison-Wesley, 2001
- [10] “XSL und XPath”, M.Bach, Addison-Wesley, 2000
- [11] “XQuery – Einführung und fortgeschrittene Methoden“ W.Lehner, H. Schöning, dpunkt Verlag, 2003.
- [12] “XQuery – ein Überblick“ W.Lehner, H.Schöning, in Datenbank-Spektrum 11/2004.