Semistrukturierte Daten und XML

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Überblick

- Semistrukturierte Daten
  - Motivation, Grundkonzepte
- XML-Einführung
  - Dokumentorientiert vs. Datenorientierte Sicht
  - Grundkonzepte (Syntax)
    - wohlgeformtes (*well-formed*) XML
- Schemadefinition für XML Dokumente
  - Dokumenttypedefinitionen
  - XML Schema
- Anfrageverarbeitung mit XML
  - Pfadausdrücke (XPath)
  - Anfragen mit XQuery

Konventionelle Datenmodelle

- Unterstützung für strukturierte Daten
  - Trennung von Schema (Strukturinformation) und Daten
- DB-Schema
  - Vollständige Strukturbeschreibung (strukturelle Meta-Daten)
  - Wird vor der Speicherung von Datenobjekten spezifiziert
  - Grundlage zur Interpretation, Manipulation von Daten
- Daten
  - Sind immer Instanzen des Schemas
    - Struktur festgelegt, keine Abweichungen möglich
  - Tragen selbst keine Strukturinformation
    - Müssen mit Hilfe des Schemas interpretiert, manipuliert werden

Person		
Name	Addresse	Alter

⟷

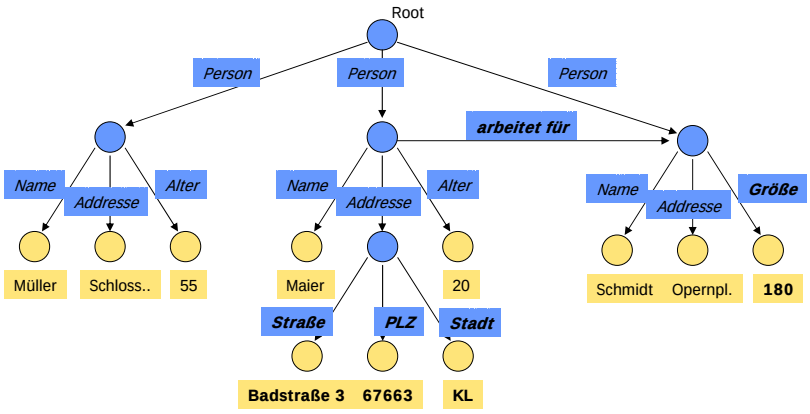
Müller	Schlossallee 1, ...	55
Maier	Badstraße 3, ...	20
Schmidt	Opernplatz 5, ...	35

Semistrukturierte Daten

- Probleme mit konventionellen DM
  - Keine Flexibilität bzgl. Strukturvorgaben
  - Schlechte Eignung für Daten- und Informationsintegration
    - Heterogenität muss immer auf Schemaebene aufgelöst werden
  - Datenaustausch
- Semi-strukturierte DM
  - Daten sind selbstbeschreibend
    - Daten und Strukturbeschreibung sind integriert
  - Keine Schemadefinition a priori notwendig
    - Breites Spektrum bzgl. Typisierung
    - Schema als “nachträgliche” Beschreibung von Struktur zur Optimierung bzw. Unterstützung der Datenmanipulation
      - Schemaextraktion, Schemainferenz
  - Flexiblere, mächtigere Anfrage- und Verarbeitungsmodelle

⇨ Nutzung von XML

Semistrukturierte Daten – Beispiel*



- Blattknoten: Daten
- innere Knoten, Kanten: Struktur/Schemainformation
- Kantenbeschriftung entspricht Attributname, Beziehungsname

* Darstellung als Object Exchange Model (OEM) graph
S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener: The lorel query language for semistructured data, 1996.

XML Ursprünge – Strukturierte Dokumente

- Zentrales Problem: Dokumentformate sind format- bzw. darstellungsorientiert, deshalb Probleme bei
  - Austausch von Dokumenten
  - Wiederverwendung von Inhalten für untersch. Darstellungsformen
- SGML (Standard Generalized Markup Language)
  - Int. Standard zur Dokumentrepräsentation (1986)
  - Auszeichnungssprache
    - Definition von beliebigen Tags zur Auszeichnung von (mglw. geschachtelten) Dokument-Elementen
      - Meta-Sprache: erlaubt Definition beliebiger Sprachen (z.B. HTML)
      - Tags haben keine vordefinierte Semantik
    - Trennung von Form und Struktur/Inhalt
    - Dokumente sind selbstbeschreibend
- XML (Extensible Markup Language) ist eine vereinfachte Form von SGML

XML - Beispiel

- HTML
  - Vermischung von Struktur und Darstellung

```
<h1>Personen</h1>
<p><i>Müller</i>
   <br>Schlossalle 1, ...
   <br>55
<p><i>Maier</i>
   <br>Badstraße 3, ...
   <br>20
<p><i>Schmidt</i>
   <br>Opernplatz 5, ...
   <br>35
```

  - Menge von Tags mit vorgeg. Bedeutung

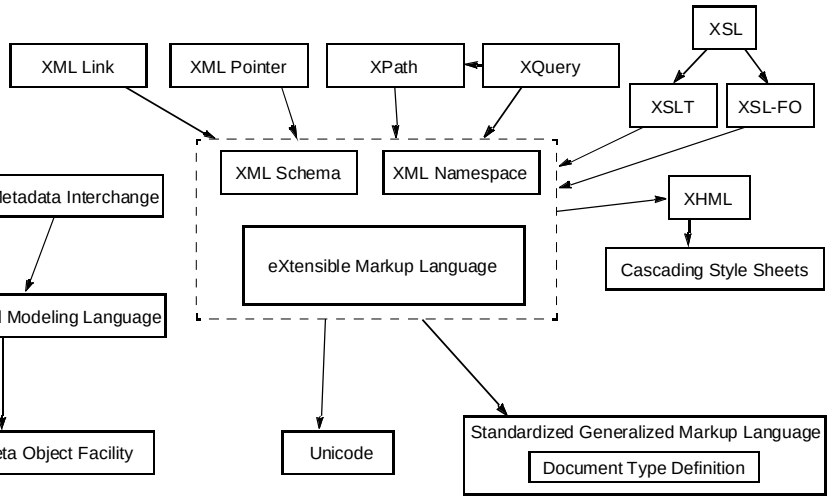
- XML
  - Kann den Inhalt beschreiben

```
<Personen>
  <Person>
    <Name>Müller</Name>
    <Addresse>
          Schlossalle 1, ...
    </Addresse>
    <Alter>55</Alter>
  </Person>
  <Person>
    <Name>Maier</Name>
    ...
  </Person>
  ...
</Personen>
```


Nutzung von XML

- Dokumentorientierte Sicht
  - Dokumentverarbeitung
    - Nutzung eines Dokuments in verschiedenen, sich verändernden Systemen
    - Aspekte: Struktur, Inhalt, Darstellung
- Datenorientierte Sicht
  - Datenaustausch
    - Daten oft strukturiert, getypt, schemabehaftet
  - Semi-strukturierte Daten und Informationsintegration
    - Schema mglw. unbekannt, dynamisch

XML Sprachspezifikationen (W3C)



XML Dokumente

- Sind Text (Unicode)
  - Markup (beginnt immer mit '<' oder '&')
    - (Start-/Ende-) Tags (z.B. <Person>, </Person>)
    - Referenzen (&lt, &amp, ...)
    - Deklarationen, Kommentare, Verarbeitungsanweisungen, ...
  - Daten (character data)
    - Zeichen '<' oder '&' müssen im Text durch Referenzen (z.B. &lt) oder direkte Verwendung des Zeichencodes angegeben werden
    - Alternative: Syntax <![CDATA[Formel: (a<b)&(c<d)]]>
- Folgen syntaktischen Regeln (**wohlgeformt – *well formed***)
  - Logische Struktur
    - (optionaler) Prolog (XML-Version, ...)
    - (optionales) Schema (dazu später mehr)
    - (Wurzel-) Element (Schachtelung möglich)
    - Kommentare, ...
  - Korrekte Folge von Start-/Ende-tags (Schachtelung!)
  - Eindeutigkeit von Attributnamen
  - ...
- Werden von "XML-Prozessoren" verarbeitet (Parser, etc.)

Elemente und Attribute

- Element
  - beginnt mit <*tagname*> , endet mit </*tagname*>
    - Ausnahme: leeres Element <*tagname*/>
  - kann Textdaten, andere Elemente oder beides beinhalten (element content)
    - *Mixed content* ist insbesondere für dokumentorienterte Anwendungen gedacht
  - Schachtelung: Start-tag und zugeordnetes Ende-tag haben den gleichen Namen und befinden sich im gleichen (umgebenden) Element
  - Elemente des gleichen Typs (d.h., mit gleichem Tag-Namen) können mehrfach vorkommen
- Attribut
  - Name/Wert-Paar im Kontext eines Elements
    - Syntax: *attname=value* innerhalb des Start-tags
    - Attributname ist eindeutig innerhalb eines Elements

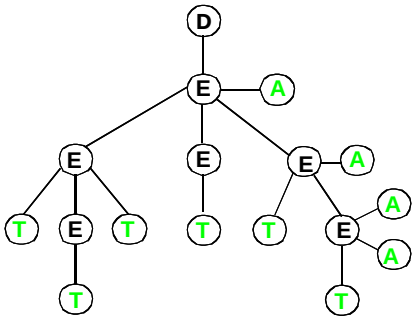
⇨ Welches Konzept soll man wann nutzen?

Beispiel

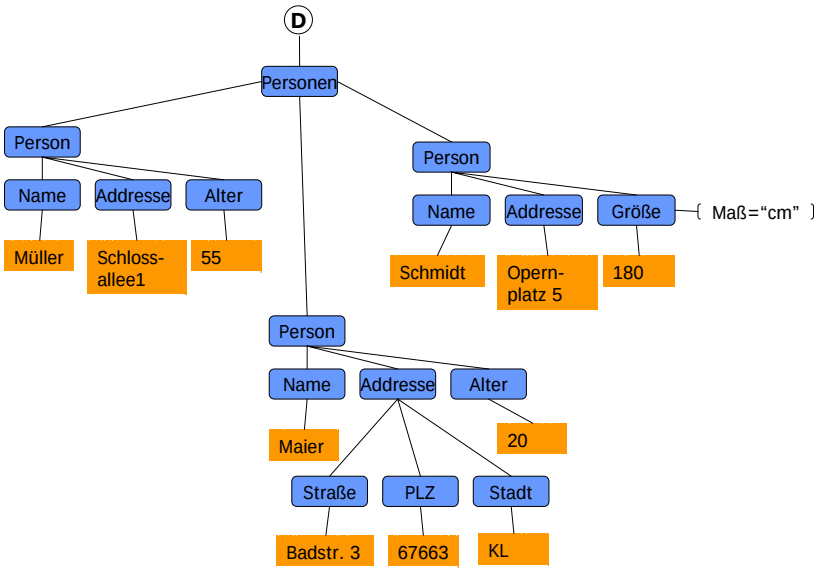
```
<?xml version="1.0"?>
<Personen>
   <Person>
     <Name>Müller</Name>
     <Addresse>Schlossalle 1, ... </Addresse>
     <Alter>55</Alter>
   </Person>
   <Person>
     <Name>Maier</Name>
     <Addresse>
              <Straße>Badstraße 3, ... </Straße>
              <PLZ>67663</PLZ>
              <Ort>KL</Ort>
     </Addresse>
     <Alter>20</Alter>
   </Person>
   <Person>
     <Name>Schmidt</Name>
     <Addresse>Opernplatz 5, ... </Addresse>
     <Größe Maß="cm">180</Größe>
   </Person>
</Personen>
```

XML Datenmodell

- Es existiert kein einheitliches Datenmodell für XML
  - Verschiedene Ansätze mit unterschiedlichem Ziel
    - XML Information Set, DOM Structure Model, XPath Datenmodell, **XQuery Datenmodell**, ...
  - Gemeinsame Sicht: XML-Dokument als Baumstruktur mit unterschiedlichen Knotentypen
    - Document, Element, Attribute, Text, Comment, ...



Beispiel



Schemadefinition für XML - Dokumente

- XML-Dokument kann (optional) Schema besitzen
  - Standardisierter Datenaustausch, ...
- Schema schränkt die für das Dokument erlaubten Strukturen und Datentypen ein
  - Dokument heisst **gültig (*valid*)**, falls es die Anforderungen des Schemas erfüllt
  - Rekursive Schemadefinitionen erlaubt
    - Bauteil besteht aus weiteren Bauteilen ...
- Zwei wichtige Ansätze
  - Document Type Definition (DTD)
    - im Dokument enthalten, oder
    - in einer separaten Datei gespeichert, im Dokument referenziert
  - XML Schema

XML Document Type Definition (DTD)

- Definition von Elementtypen
  - Welche Elemente dürfen vorkommen
  - Welche Attribute darf bzw. muss ein Element haben
  - Welche Unterelemente dürfen bzw. müssen in einem Element auftreten, und wie oft
- DTD bietet keine Unterstützung für Datentypen
  - Daten sind, wie gehabt, nur Zeichenketten ohne weitere Einschränkungen
- DTD-Syntax
  - <!ELEMENT element (subelements-specification) >
  - <!ATTLIST element (attributes) >

Elementspezifikation

- Mögliche Unterelemente
  - Name des Elements
  - #PCDATA (parsed character data), d.h., bel. Zeichenkette
  - EMPTY (keine Unterelemente) or ANY (beliebige Unterelemente)
- Strukturierung mit Hilfe von regulären Ausdrücken
  - Sequenz (*subel* , *subel* , ...), Alternative (*subel* | *subel* | ...)
  - Wie oft darf *subel* vorkommen?
    - "?" - 0 oder 1
    - "+" - mindestens 1
    - "*" - beliebig oft
- Beispiel:

```
<!DOCTYPE  Personen [
    <!ELEMENT Personen (Person*)
    <!ELEMENT Person (Name, Addresse+, (Alter | Größe) )
    <!ELEMENT Name (#PCDATA)
    <!ELEMENT Addresse (#PCDATA | (Straße, PLZ, Stadt) )
    ...
```


Attributspezifikation

- Für jedes Attribut
  - Name
  - Attributtyp
    - Zeichenkette (CDATA) oder Name Token
    - Aufzählungstyp
    - ...
  - Vorkommen
    - Attribut(wert) muss vorhanden sein, oder
    - Attribut is optional, oder
    - Defaultwert
- Beispiel:
  - <!ATTLIST Größe Maß (cm | inches) #REQUIRED>

Elementidentität und Referenzen

- Attributtyp ID, IDREF
  - Element kann max. ein Attribut vom Typ ID besitzen
    - Wert muss eindeutig sein im ganzen Dokument (-> Objekt-ID)
- Attribut vom Typ IDREF muss den ID-Wert eines (beliebigen) Elements im gleichen Dokument enthalten
  - Typ IDREFS: ein oder mehrere REFs
- Beispiel:
  - <!ATTLIST Person oid ID #REQUIRED
    arbeitetFür IDREF #IMPLIED>
- Erlaubt mit großen Einschränkungen Repräsentation von Beziehungen
  - ID, IDREFs sind ungetypt, d.h., die Menge der referenzierbaren Elemente ist nicht einmal auf Typebene definierbar
    - Bsp.: arbeitetFür könnte auch Addresselemente referenzieren, wenn diese eine ID hätten ...

Schemabeschreibung mit XML Schema

- Unterstützung von Datenmodellierungskonzepten, u.a.
  - Datentypen
    - integer, string, ...
  - Constraints
    - min/max-Werte für mgl. Anzahl von Elementwiederholungen
    - Werteindeutigkeit (unique), Fremdschlüssel
  - Getypte Referenzen
  - Namensräume in XML (name spaces)
  - Benutzerdefinierte Typen
    - Vererbung
- Modularisierung und Wiederverwendung von Schemadefinitionen
- Schemadefinition ist wiederum ein XML-Dokument
- Deutlich komplizierter als DTDs

Namensräume (Namespaces)

- Ziel: Vermeidung von Namenkollisionen bei Nutzung von verschiedenen Vokabularen (Schemata) im gleichen Dokument
  - Beispiel
    - Ein Element Titel kommt sowohl im Kontext von Büchern (Buchtitel) als auch im Kontext von Personen (z.B. akademischer Titel) vor
    - Ein Dokument das Information über Bücher und Autoren enthält soll Bestandteile aus beiden Schemate verwenden können
- Namensraum
  - "Enthält" eine Menge von Namen
  - Durch URI "weltweit" eindeutig identifiziert
  - Kann in einem XML Dokument oder Element genutzt werden
    - Deklaration als Default-Namespace
    - Definition und Angabe von "Kürzeln" (prefix)
      - <lit:titel>, <pers:titel>, ...

XML Schema – Definition und Nutzung

- XML-Schema Dokument
  - Deklariert Elemente/Attribute bzw. definiert (einfache oder zusammengesetzte) Datentypen
  - Ordnet (optional) die deklarierten/definierten Konzepte einem Namensraum zu (target namespace)
- XML-Dokument
  - Nutzt ein Schema durch Deklaration/Referenz des entspr. Namensraums
    - Nutzung mehrerer Schemata im gleichen Dokument
    - Namensraumdefinition für Wurzelelement und/oder beliebige Unterelemente

XML Schema - Beispiel

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.personen.org"
        xmlns ="http://www.personen.org" >
<xsd:element name=“Person”>
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name=“Name” type=“xsd:string”/>
         <xsd:element name=“Addresse” type=“AddresseTyp”
              minOccurs=“1” maxOccurs=“3”/>
        <xsd:choice>
         <xsd:element name=“Alter” type=“xsd:string” />
         <xsd:element name=“Größe” type=“xsd:integer” />
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name=“oid” type=“xsd:ID” />
      <xsd:attribute name=“ arbeitetFür” type=“xsd:IDREF” />
   </xsd:complexType>
...
```


Abbildung ER-Modell -> XML Schema

- Entities
  - 1:1-Abbildung auf XML Elemente
  - <key> in XML-Schema, um Schlüsselattribute darzustellen
- Relationships
  - 1:1, 1:N
    - Schachtelung von Elementen
      - Probleme: Existenzabhängigkeit, Eindeutigkeit von Schlüsselattributen in gesch. Elementen, ...
    - “Flache” Repräsentation
      - Nutzung von Schlüsseln, Fremdschlüsseln
  - N:M
    - Flache Repräsentation mit Hilfselementen

Anfragesprachen

- Anfragen auf
  - (großen) XML-Dokumenten
  - XML-Daten in nativen XML-Datenbanken
  - logischen XML-Sichten auf beliebigen Datenquellen
- Semi-strukturierter Ansatz erfordert Umgang mit
  - schemalosen Daten/Dokumenten
  - heterogenen Dokumentstrukturen
- Funktionalität
  - Pfadausdrücke zur Lokalisierung von Knoten im XML-Baum
  - Komplexe Anfragen
- XML-Anfragesprache XQuery *
  - beinhaltet u.a. wesentliche Funktionen von XPath
  - wird z.Zt. noch standardisiert (W3C)

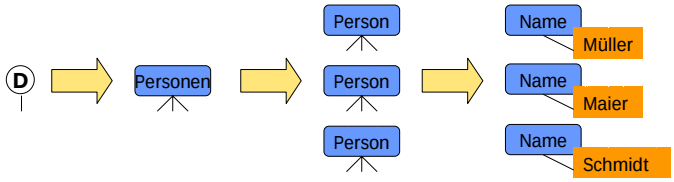
* W.Lehner, H.Schöning: *XQuery – Grundlagen und fortgeschrittene Methoden*, dpunkt.verlag, 2004.

Datenmodell für XML Anfragen

- Ziel: Abgeschlossenheit bzgl. der Operationen der Anfrageverarbeitung
  - Baummodell nicht ausreichend
- Datenmodell in XPath, XQuery
  - Objekte des Datenmodells sind Sequenzen
    - Sequenz hat 0, 1 oder mehrere Einträge (*items*)
      - Entweder Knoten (*nodes*) oder atomare Werte (*atomic values*)
    - Einträge sind geordnet
  - Knoten als Sequenzeinträge
    - Repräsentieren Baumstruktur eine Dokuments bzw. Fragments
      - 7 Knotentypen: Element, Dokument, Attribut, Namensraum, Verarbeitungsanweisung, Kommentar, Text
    - Knoten haben eine Identität (nicht verwechseln mit ID, IDREF)
    - Ordnung im Baum entspricht Dokumentordnung
      - Eltern < Kinder
      - Namenräume < andere Attribute < Kinder
      - Geschwister in Dokumentreihenfolge
- Anfrageausdrücke operieren auf einer oder mehreren Sequenzen und liefern wieder eine Sequenz

Pfadausdrücke in XPath, XQuery

- Pfadausdruck addressiert (selektiert) ein Sequenz von Knoten in einem Dokument
  - Besteht aus Schritten, durch "/" voneinander getrennt
    - Bsp.: Namen aller Personen
      /child::Personen/child::Person/child::Name
  - Wird sukzessive, von links nach rechts ausgewertet
    - Pfadanfang: Dokumentwurzel oder von aussen vorgegebener Kontext
    - Jeder Schritt geht von Knotensequenz aus
      - Sucht für jeden Knoten in der Sequenz weitere Knoten auf
      - Duplikateliminierung aufgrund der impliziten Knotenidentität
      - Mglw. Sortierung der Knoten in Dokumentreihenfolge
    - Leere Resultats führt nicht zu Fehler

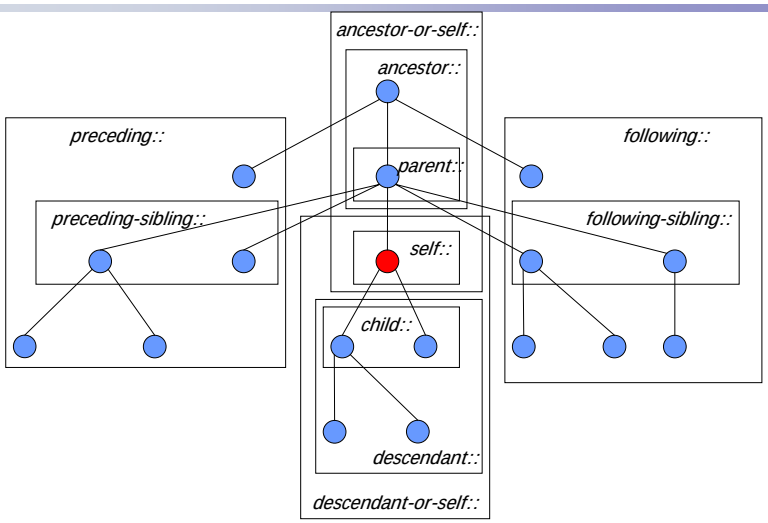


Pfadausdrücke - Schritte

- Initialer "/" bezeichnet Dokumentknoten (Wurzel)
- Ein Achsenschritt hat i.A. drei Bestandteile
  - Achse – beschreibt Navigationsrichtung vom Kontextknoten ausgehend
  - Knotentest – Auswahl von Knoten aufgrund des Namens oder Typs
  - Optionale Prädikat(e) – Weitere Selektion von sich qualifizierenden Knoten
  - Beispiel: Alle Personen älter als 30
    child::Person[child::Alter > 30]

    Syntax: Achse::Knotentest [Prädikat] ...
- Alternativ: Filterschritt
  - Anstellt Achse, Knotentest kann ein Ausdruck stehen, der für Knoten aus dem Kontext weitere Knoten lokalisiert

Mögliche Achsen



- Attribute und Namensräume hier nicht beinhaltet
  - Weitere (künstliche) Achse: *attribute::*

Knotentests

- Namenstest
  - Element- oder Attributename
    - child::*name* – <*name*> Elementknoten
    - child::* - alle Elementknoten
    - attribute::*name*, attribute::* - analog für Attributnamen
  - *namespace*:*name*, *namespace*:* – Beschränkung auf angegebenen Namensraum
- Knotentyptest
  - Wählen nur Knoten des entspr. Typs aus
    - comment()
    - text()
    - processing-instruction()
    - node() – beliebiger Knoten
    - element()
      - element(*name*)
      - element(*name, type*)

Pfadausdrücke – Beispiele

- /descendant::Person/child::Name
  - <Name>Müller</Name>
    <Name>Maier</Name>
    <Name>Schmidt</Name>
- /descendant::Person/child::Name/child::text()
  - Müller
    Maier
    Schmidt
- /descendant::Person/child::Größe/attribute::Maß
  - Maß="cm"
- /descendant::Addresse/child::*
  - Schloßallee 1, ...
    <Straße>Badstraße 3</Straße>
    <PLZ>67663</PLZ>
    <Stadt>KL</Stadt>
    Opernplatz 5, ...

Prädikate

- Wertevergleiche (und viele Funktionen) betrachten nicht Knoten, sondern deren (getypte) Werte
  - "Atomisierung"
- Boolsche Ausdrücke:
  - /descendant::Person[child::Name = "Maier"]
    - logische Konnektoren unterstützt
- Numerische Ausdrücke:
  - /descendant::Person[2]
    - *> liefert das zweite Personenelement*
- Existenztests:
  - /descendant::Person[child::Größe]
    - *> Personenelemente*
  - /descendant::*[attribute::Maß]
    - *> Elemente mit Maß-Attribut*
- Nutzung von Funktionen:
  - /child::Personen/child::Person[fn:position() = fn:last()]
    - *> letztes Personenelement unter <Personen>*

Nutzung von Funktionen

- Nutzung von Funktionen:
  - /descendant::Person[child::Name="Maier"]
    /fn:string(child::Addresse)
    - string() liefert textuellen Wert eines Knotens
    - *> Badstr. 367663KL*
  - /descendant::Person
    [fn:id(attribute::arbeitetFür)/child::Alter < 30]/child::Name
    - Funktion id() liefert Elemente mit angeg. ID-Wert
    - *> Namen der Personen, die für Personen jünger als 30 arbeiten*
  - /descendant::Person[child::Alter<30]
    /fn:id(attribute::arbeitetFür)/child::Name
    - Verwendung von id() in Filterschritt
    - *> Name von Personen, für die Personen jünger als 30 arbeiten*
- Zugriff auf externe Dokumente:
  - fn:doc("Personen.xml")/descendant::*
    - *> alle Knoten im angegebenen Dokument*
      - *auf beliebiger Schachtelungsebene*

Verkürzte XPath-Syntax

- Verkürzte Schreibweisen für häufig genutzte Konstrukte
  - Punkt (".") - aktueller Referenzknoten
  - Doppelter Punkt ("..") - parent::node()
  - "//" - /descendant-or-self::node()/
  - "@" - attribute::
  - Achse fehlt - child::
    (bzw. attribute:: bei Attributtest)
- Folgende Ausdrücke sind bspw. äquivalent
  - /descendant-or-self::Person
    [fn:id(attribute::arbeitetFür)/child::Alter < 30]/child::Name
  - //Person[fn:id(@arbeitetFür)/Alter < 30]/Name

XQuery – Überblick

- Pfadausdrücke sind wesentlicher Bestandteil einer XML-Anfragesprache
- Weitere Sprachkonstrukte werden benötigt
  - Konstruktion neuer XML-Objekte als Anfrageresultat
    - Bsp.: Schachtelung der durch einen Pfadausdruck lokalisierten Elemente in ein neues Element zur Ausgabe
  - Binden und Nutzen von Variablen zur Iteration über mehreren Sequenzen von Knoten
  - Verbundoperationen
  - Sortierung
  - Aggregation
  - ...
- Wesentliche zusätzl. Konzepte in XQuery
  - Konstruktoren
  - FLWOR-Ausdrücke (gespr.: *flower*)

Konstruktoren

- Nutzung der XML-Dokumentsyntax um Dokumentfragmente zu konstruieren

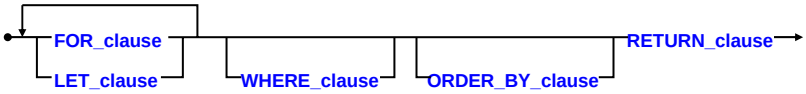
```
<buch isbn = "12345">
  <titel>Huckleberry Finn</titel>
</buch>
```

- Geschweifte Klammern, um Resultate eines Anfrageausdrucks als Inhalt zu berücksichtigen

```
<buch isbn = "{$x}">
  {$b/title }
</buch>
```

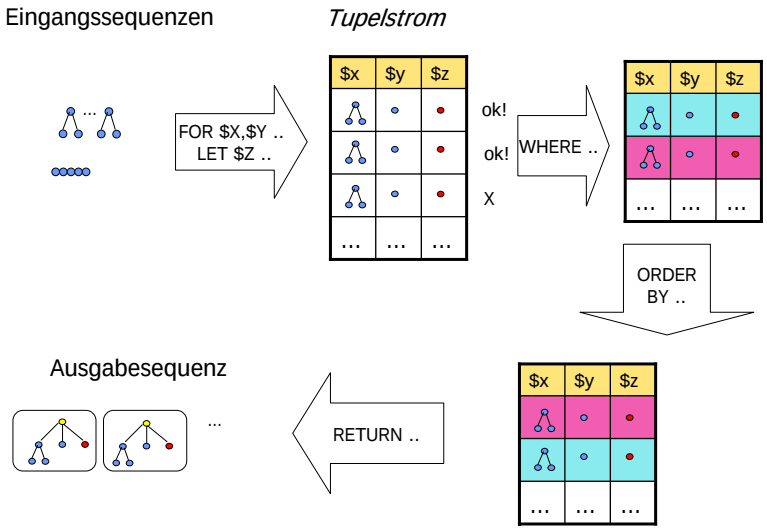
  - Variablenbelegungen sind hier durch umgebenden Ausdruck vorgegeben
  - Dynamische Berechnung von Namen und Inhalten möglich.
    - element {*name-expr*} {*content-expr*}
    - attribute {*name-expr*} {*content-expr*}

FLWOR - Ausdrücke



- FOR, LET binden Variablen durch
  - Iteration über eine Sequenz von Knoten (FOR)
  - Auswertung eines Ausdrucks (LET)
- WHERE ermöglicht Selektion von Variablenbelegungen aufgrund von Prädikaten
- ORDER BY erlaubt Sortieren von Inhalten
- RETURN generiert Resultat des Ausdrucks anhand der Variablenbelegungen
- Vergleichbar mit folgenden Konzepten in SQL:
  for ⇔ SQL from
  where ⇔ SQL where
  order by ⇔ SQL order by
  return ⇔ SQL select
  let hat keine Entsprechung

Auswertung von FLWOR-Ausdrücken



FLWOR - Beispiel

- Einfacher FLWOR-Ausdruck
  - Suche alle Personen (Name, Alter) jünger als 25
    - for $p in //Person
      let $n := $p/Name, $a := $p/Alter
      where $a < 25
      order by $n
      return <JungePerson> {$n, $a } </JungePerson>
  - Äquivalente Anfrage ohne LET, WHERE:
    - for $p in //Person[Alter < 25]
      order by $p/Name
      return <JungePerson> {$p/Name, $p/Alter} </JungePerson>
- Variablen beginnen mit “$”-Präfix

Verbundoperationen

- Verfolgung von Referenzen
  - for $p in //Person[@arbeitetFür]
    let $chef := fn:id($p/@arbeitetFür)
    return
      <arbeitetFür>
        <Person> {$p/Name} </Person>
        <Chef> {$chef/Name} </Chef>
      </arbeitetFür>
- Symmetrischer Verbund
  - for $p in //Person, $chef in //Person
    where $p/@arbeitetFür = $p/@oid
    return
      <arbeitetFür>
        <Person> {$p/Name} </Person>
        <Chef> {$chef/Name} </Chef>
      </arbeitetFür>

Left Outer Join

- Beispiel: Liste alle Personen mit Namen und Chef (falls vorhanden)
  - for $p in //Person
    return
      <Person>
        { $p/Name,
          for $chef in fn:id($p/@arbeitetFür)
          return <Chef> {$chef/Name} </Chef> }
      </Person>

Geschachtelte FLWOR-Ausdrücke

- Beispiel: Invertierung einer Hierarchie
  - Eliminiert (wertbasiert) Duplikate mit fn:distinct-values()

```
<buch>
   <titel>
   <autor>
   <autor>
</buch>
<buch>
   <titel>
   <autor>
   <autor>
</buch>
```



```
for $a in fn:distinct-values(//autor)
order by $a/name
return
    <autor>
        <name> { $a/text() } </name>
        { for $b in //buch[autor = $a]
          return $b/title }
    </autor>
```

```
<autor>
   <name>
   <titel>
   <titel>
</autor>
<autor>
   <name>
   <titel>
   <titel>
</autor>
```

Sortierung und Schachtelung

- Suche alle Herausgeber, sortiere nach Name, und liste alle Titel und Preise ihrer Bücher, absteigend nach Preis sortiert
  - <verlag_liste>
    {for $v in distinct-values(doc("bib.xml")//verlag)
     order by $v/name
     return <verlag>
              <name> {$v/text()} </name>
             {for $b in doc("bib.xml")//buch[verlag = $v]
              order by $b/preis descending
              return <buch>
                       {$b/titel}
                       {$b/preis}
                     </buch>
              }
             </verlag>
    }
    </ verlag_liste >

Gruppierung und Aggregation

- Aggregatfunktion werden auf Sequenz angewendet
- Gruppierung
  - entlang einer XML-Hierarchie (-> Unterelemente)
    - for $v in //verlag
      let $p := $v//buch/preis *<- Sequenz von Preisen!*
      return <verlag> {$v/name}
                <d-buchpreis> { **fn:avg**($p) } </d-buchpreis>
             </verlag>
  - aufgrund von Werten
    - for $jahr in fn:distinct-values(//buch/erscheinungsjahr)
      let $preise := //buch[erscheinungsjahr=$jahr]/preis
      return <d-preis> { $jahr }
                <betrag> { **fn:avg**($preise) } </betrag>

Zusätzliche Funktionalität

- Ausdrücke
  - arithmetische und konditionale Ausdrücke
  - Mengenoperationen
  - quantifizierte Ausdrücke
  - Typumwandlung
- Funktionen
  - Definition und Nutzung
- Anfrageverarbeitungsmodell
  - Statische Analyse und dynamische Evaluierung
  - Nutzung von Typinformation
- ...

Zusammenfassung

- Semistrukturierte Daten
  - selbst-beschreibend (Integration von Schema und Daten)
- XML
  - Meta-Sprache zur sem. Beschreibung von Dokumenten
    - Struktur und Inhalt
    - wohlgeformtes (*well-formed*) XML
  - Dokumentorientiert vs. Datenorientierte Sicht
- Schemadefinition für XML Dokumente
  - DTDs
    - rudimentäre, struktur-orientierte Schemata
  - XML Schema
    - unterstützte effektivere Datenmodellierung mit XML
- Anfrageverarbeitung mit XML
  - Pfadausdrücke (XPath)
    - wichtiges Konzept zur flexiblen Lokalisierung von Knoten in XML-Bäumen
  - XQuery
    - komplexe Anfragen und Transformationen auf XML-Daten/Dokumenten