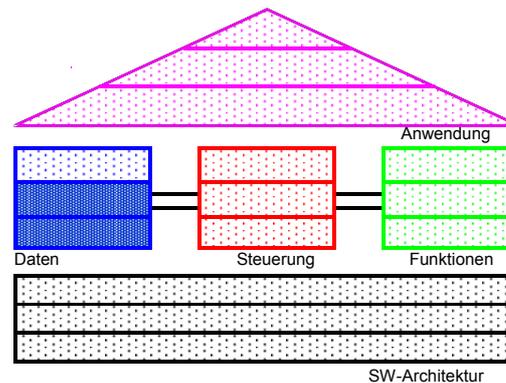


# 7. Semistrukturierte Daten und XML

Theo Härder  
[www.haerder.de](http://www.haerder.de)

GBIS-Rahmen: Einordnung



## Überblick

Semistrukturierte  
Daten

XML-Einführung

Vergleich  
XML <-> RM

Schemadefinition

XPath-  
Prädikate

XQuery-  
Anfragen

- **Semistrukturierte Daten**
  - Motivation, Grundkonzepte
- **XML-Einführung**
  - dokumentorientierte vs. datenorientierte Sicht
  - Grundkonzepte (Syntax)
    - wohlgeformtes (*well-formed*) XML
  - Vergleich XML ↔ Relationenmodell
  - Welches Modell gewinnt?
- **Schemadefinition für XML-Dokumente**
  - Dokumenttypdefinitionen
  - XML-Schema
- **Anfrageverarbeitung mit XML**
  - Pfadausdrücke (XPath)
  - Anfragen mit XQuery
  - Auswertungsbeispiele

# Konventionelle Datenmodelle

Semistrukturierte Daten

XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen

## ■ Unterstützung für strukturierte Daten

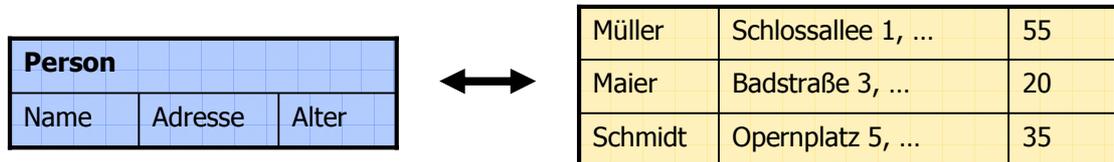
- Trennung von Schema (Strukturinformation) und Daten

## ■ 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



# Semistrukturierte Daten

Semistrukturierte Daten

XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen

## ■ Probleme mit konventionellen DM

- Keine Flexibilität bzgl. Strukturvorgaben
- Schlechte Eignung für Daten- und Informationsintegration
  - Heterogenität muss immer auf Schemaebene aufgelöst werden
- Datenaustausch

## ■ Semistrukturierte 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\*

Semistrukturierte Daten

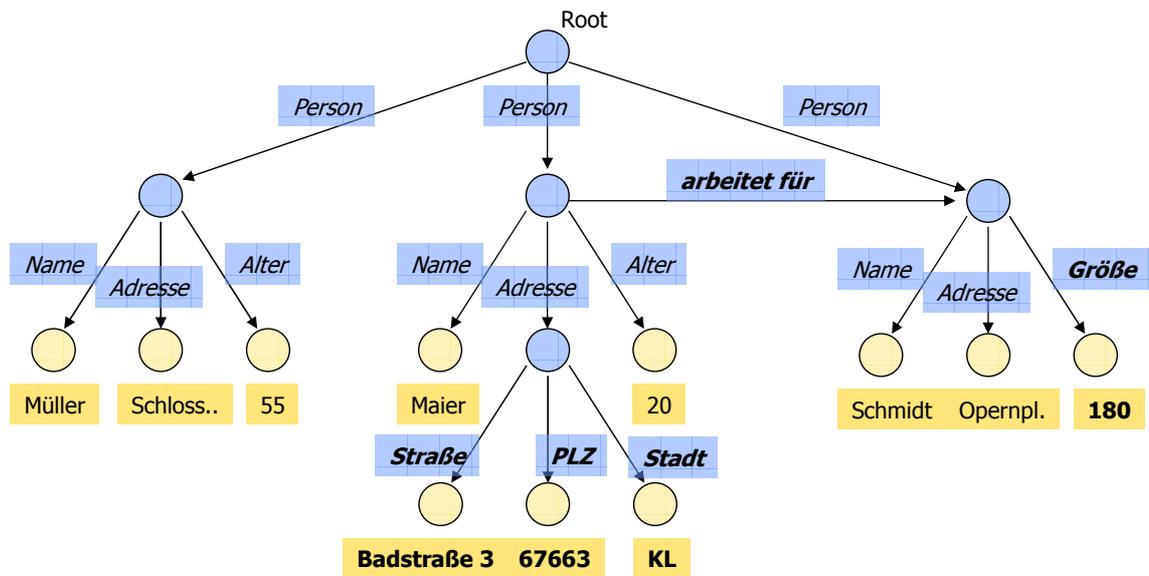
XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen



- 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

Semistrukturierte Daten

XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen

- **Zentrales Problem:** Dokumentformate sind format- bzw. darstellungsorientiert, deshalb Probleme bei
  - Austausch von Dokumenten
  - Wiederverwendung von Inhalten für unterschiedliche 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 vereinfachte Form von SGML



## 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 Formatierungsanweisungen (Tags) mit vorgegebener Bedeutung
- zur Darstellung für den menschlichen Benutzer geeignet

## XML

- kann den Inhalt (Struktur und Daten) beschreiben

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

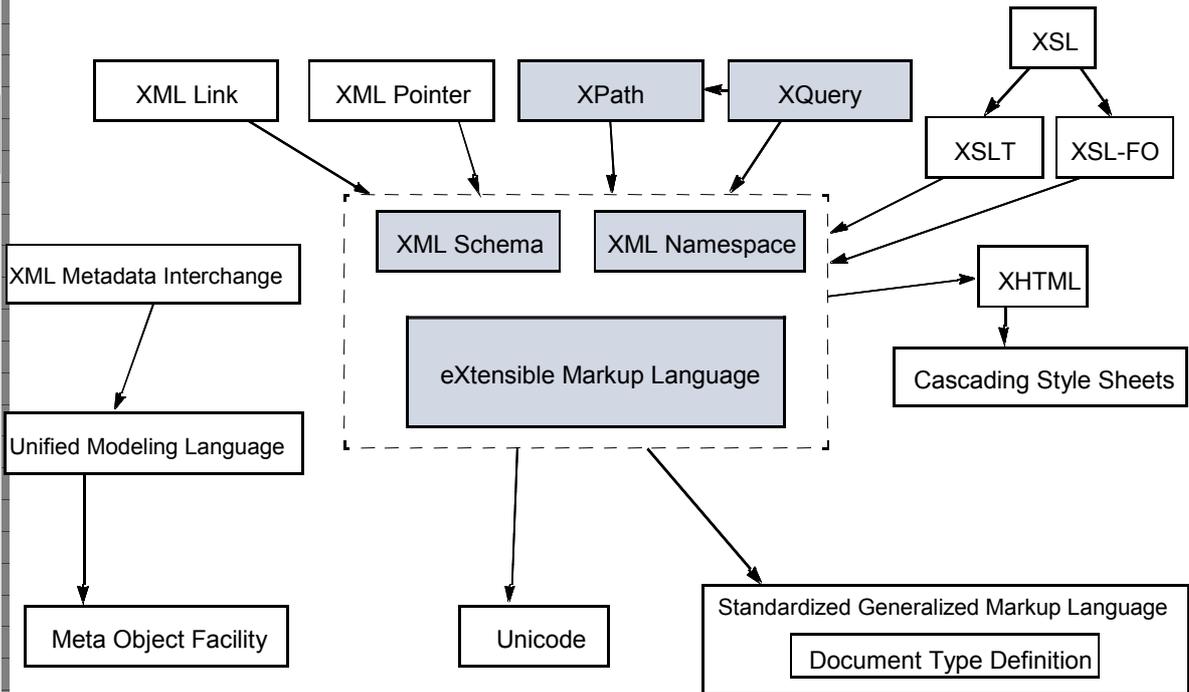
- erlaubt maschinelle Verarbeitung

## Dokumentorientierte Sicht

- Dokumentverarbeitung
  - Nutzung eines Dokuments in verschiedenen, sich verändernden Systemen
  - Aspekte: Struktur, Inhalt, Darstellung

## Datenorientierte Sicht

- Datenaustausch
  - Daten oft strukturiert, getypt, schemabehaftet
- semistrukturierte Daten und Informationsintegration
  - Schema möglicherweise unbekannt, dynamisch



<sup>1)</sup> <http://www.w3.org/XML/>

- 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.)

## ■ Element

- beginnt mit `<tagname>` , endet mit `</tagname>`
  - Ausnahme: leeres Element `<tagname/>`
- kann Textdaten, andere Elemente oder beides beinhalten (element content)
  - *Mixed content* ist insb. für dokumentorientierte Anwendungen gedacht
- Schachtelung: Start-Tag und zugeordnetes Ende-Tag haben 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
  - beschränkt auf atomare Werte
  - Syntax: `attname="value"` innerhalb des Start-Tags
  - Attributname ist eindeutig innerhalb eines Elements
  - es ist keine Ordnung für Attribute eines Elementes definiert

⇒ Welches Konzept soll man wann nutzen?

```
<?xml version="1.0"?>
<Personen>
  <Person>
    <Name>Müller</Name>
    <Adresse>Schlossalle 1, ... </Adresse>
    <Alter>55</Alter>
  </Person>
  <Person>
    <Name>Maier</Name>
    <Adresse>
      <Straße>Badstraße 3, ... </Straße>
      <PLZ>67663</PLZ>
      <Ort>KL</Ort>
    </Adresse>
    <Alter>20</Alter>
  </Person>
  <Person>
    <Name>Schmidt</Name>
    <Adresse>Opernplatz 5, ... </Adresse>
    <Größe Maß="cm" Gemessen_am="01.07.2006">180</Größe>
  </Person>
</Personen>
```



# Beispiel: XML ↔ Relationenmodell

Semistrukturierte Daten

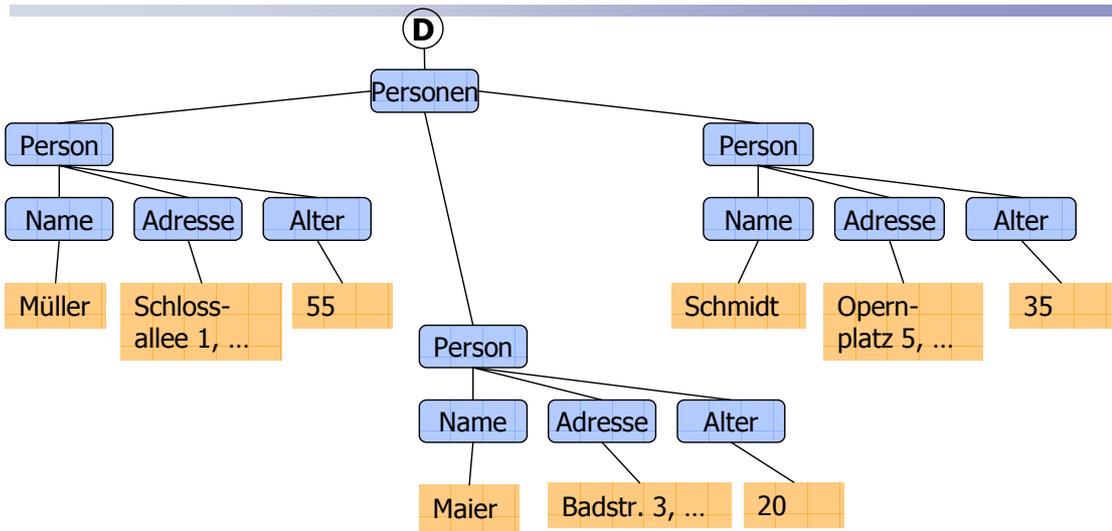
XML-Einführung

Vergleich XML ↔ RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen



Person		
Name	Adresse	Alter
Müller	Schlossallee 1, ...	55
Maier	Badstraße 3, ...	20
Schmidt	Opernplatz 5, ...	35

Perfekte Abbildung im RM nur, wenn

- gleichförmige Objekte (Entities)
- genau dreistufige Beschreibung O/A/W
- atomare Werte
- keine Aspekte (Attribute von XML-Elementen)
- Ordnung ohne Bedeutung ist.

# Erweitertes Beispiel: XML ↔ Relationenmodell

Semistrukturierte Daten

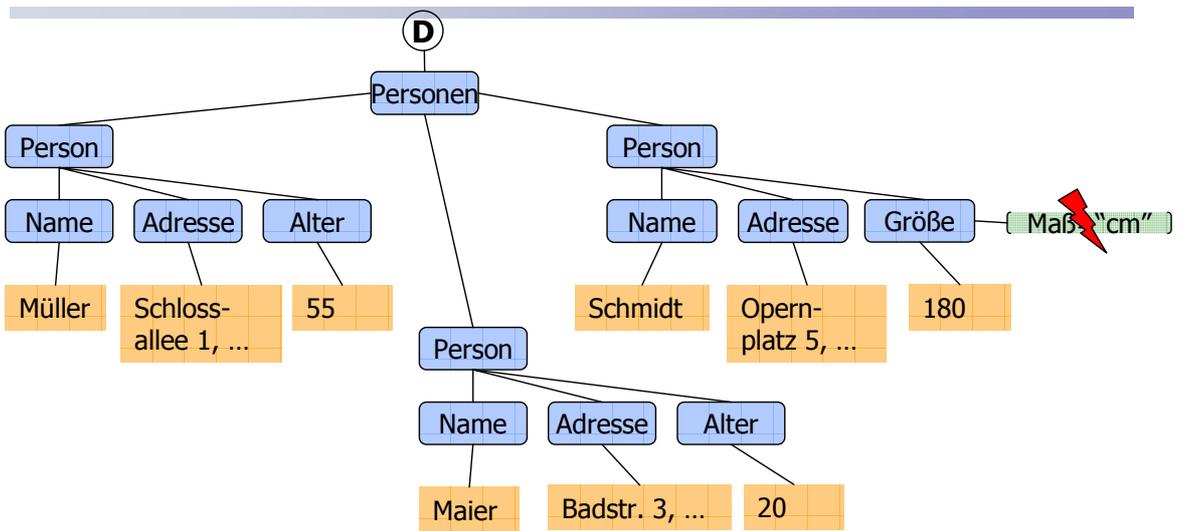
XML-Einführung

Vergleich XML ↔ RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen



Person			
Name	Adresse	Alter	Größe
Müller	Schlossallee 1, ...	55	--
Maier	Badstraße 3, ...	20	--
Schmidt	Opernplatz 5, ...	--	180

Einfache Abbildung im RM möglich

- bei Bedingungen für perfekte Abbildung,
- aber geringfügigen Strukturabweichungen (z. B. Alter/Größe)
- jedoch keine Aspekte (Attribute von XML-Elementen); sie müssten auf eine separate Spalte abgebildet werden

# Erweitertes Beispiel: XML ↔ Relationenmodell (2)

Semistrukturierte Daten

XML-Einführung

Vergleich XML ↔ RM

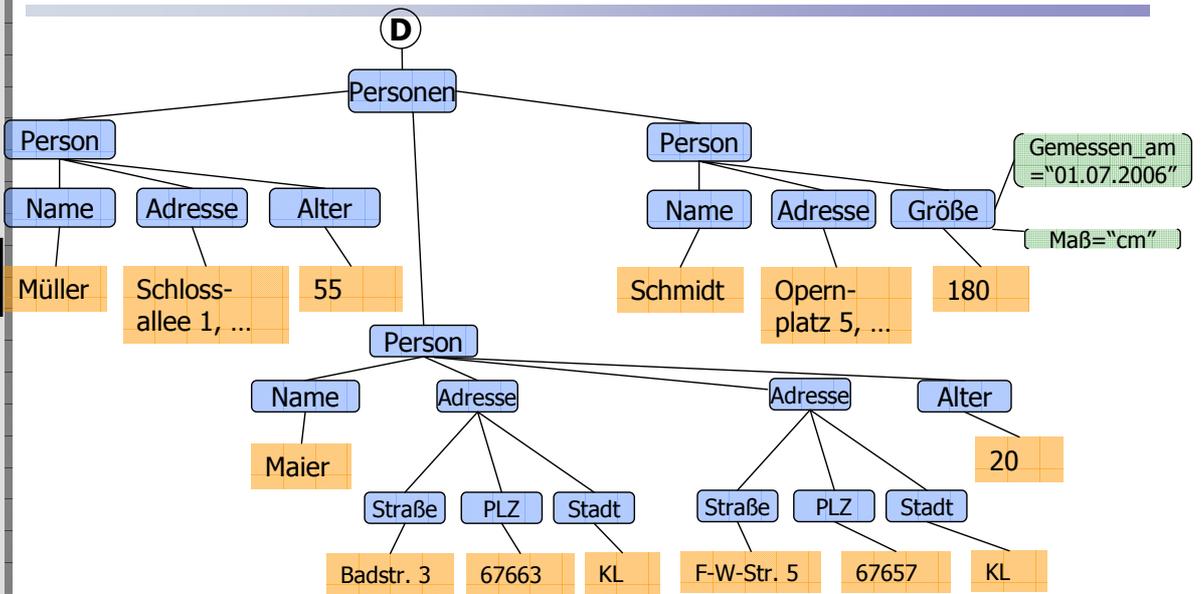
Schemadefinition

XPath-Prädikate

XQuery-Anfragen



© 2004 AG DBIS



Person			
Name	Adresse	Alter	Größe
Müller	Schlossallee 1, ...	55	--
Maier	?	20	--
Schmidt	Opernplatz 5, ...	--	180

Abbildung im RM in einer Tabelle scheitert, wenn Objektbeschreibung

- mehr als drei Stufen hat (zusammengesetzte Attribute),
- mehrwertige oder relationenwertige Attribute (n-stufige Wiederholungsgruppen) besitzt

7-17

# Erweitertes Beispiel: XML ↔ Relationenmodell (3)

Semistrukturierte Daten

XML-Einführung

Vergleich XML ↔ RM

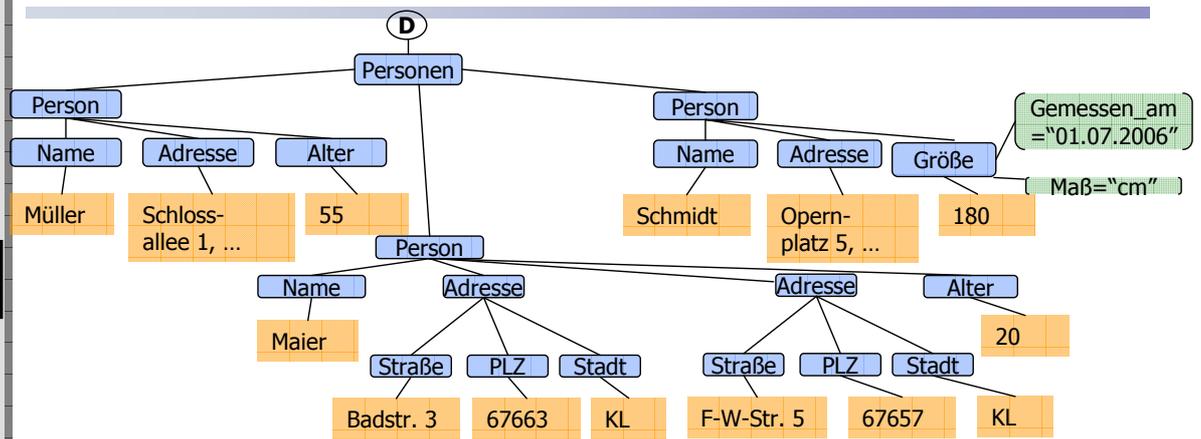
Schemadefinition

XPath-Prädikate

XQuery-Anfragen



© 2004 AG DBIS



Person					
LfdNr	Name	Adresse	AdNr	Alter	Größe
1	Müller	Schlossallee 1, ...	--	55	--
2	Maier	--	1	20	--
3	Schmidt	Opernplatz 5, ...	--	--	180

Adresse				
AdNr	LfdNr	Straße	PLZ	Stadt
1	1	Bachstr. 3	67663	KL
1	2	F-W-Str. 5	67657	KL
...				

Abbildung im RM über mehrere Tabellen

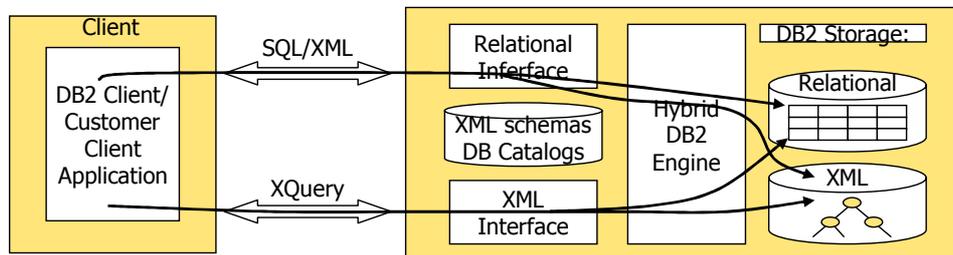
- wird zumindest sehr komplex und unverständlich (siehe Kap. 4),
- zusätzlich muss auch Ordnung erhalten werden,
- wird auch „Shredding“ genannt

7-18

## Welches Modell setzt sich durch?

- Zunehmende Entwicklung von verteilten Anwendungen auch im Internet
  - Standardisierte XML-Austauschformate sind für AW-Klassen vordefiniert
  - Nachrichten sind aber auch (in DBs zu speichernde) Daten!
  - ständiges Konvertieren: RM  $\leftrightarrow$  XML?
- Koexistenz
  - native XML-Verarbeitung löst RM nicht ab
  - beide Modelle bestehen nebeneinander und ergänzen sich!
- XML wird bereits intensiv für Kommunikation und Datenbeschreibung (Web Services, SOA) benutzt, weil es
  - wesentlich flexibler ist und die Datenintegration vereinfacht
  - selbstbeschreibend, erweiterbar, hersteller- und plattformunabhängig ist

B1 sieht eine hoch entwickelte relationale DB, die auch XML unterstützt



B2 sieht eine hoch entwickelte native XML-DB, die auch SQL unterstützt

7-19

## Warum XML-Datenbanksysteme (XDBS)?

- Erwünschte Eigenschaften:
  - Instanzen tragen Struktur und Daten (integrierte Metadaten)
  - Modellierungsflexibilität und Schemaevolution
  - DB- oder Applikations-kontrollierte Konsistenz
  - zusätzlich zu den in der Praxis bewährten DBS-Eigenschaften
    - ACID-Transaktionen
    - deklarative Anfrageverarbeitung
    - effiziente und parallele Verarbeitung von großen Datenvolumina
    - hohe Verfügbarkeit und Fehlertoleranz
    - Skalierbarkeit für transaktionale Arbeitslasten und Datenvolumina
- Beispiele:
  - **Dokument-zentrierte Sicht:** Dokumentkollektionen
    - Bücher, Artikel, Web-Seiten, ...
    - Applikationen: Struktur-sensitives Information Retrieval, ...
  - **Daten-zentrierte Sicht:** semi-strukturiertes Datenmodell
    - Nachrichten, Konfigurationsdateien, semi-strukturierte Daten per se
    - Applikationen: Informationsverwaltung für Administration, Krankenhäuser, E-Business, ...

7-20

# Relationale, native oder hybride Speicherung?

Speicherung als „lange Felder“ (LOBs) ermöglichen keine feinkörnige Verwaltung, keine inhaltsbasierte Suche und keine Mehrbenutzer-Verarbeitung

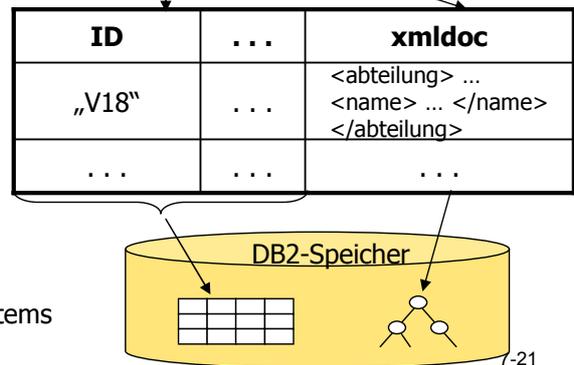
## Abbildung auf relationale Tabellen?

- dazu gibt es viele Lösungen: „Shredding“
- **Leistungsverhalten** (Suche, IUD-Ops) ungeklärt
- XML-Anfragesprachen (z.B. XQuery, XPath, DOM, SAX) sind auf SQL abzubilden
- Nutzung des SQL-Optimizers!
- Aber: Kontrolle des Mehrbenutzerbetriebs (Sperrungen) ist sehr kompliziert, weil ein Dokument über n Tabellen verteilt sein kann

## Hybrider Ansatz

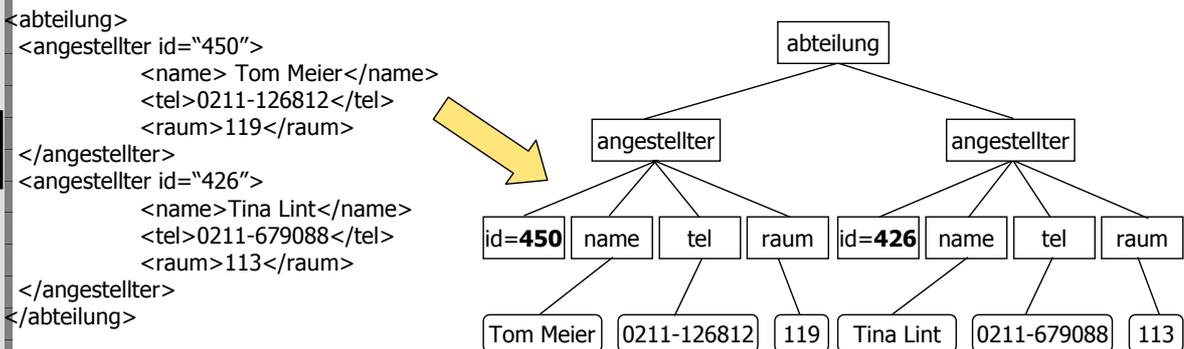
- Beispiel: DB2-Viper
- SQL mit XQuery-Teilfragen von SQL-Funktionen xmlQuery, xmlExists und xmlTable
- „stand-alone“ XQuery-Schnittstelle
  - bietet Zugriff auf XML-Spalten
  - db2-fn:xmlcolumn importiert eine ganze XML-Spalte als Sequenz von Items

```
create table abt (ID char(8), ..., xmldoc xml)
```

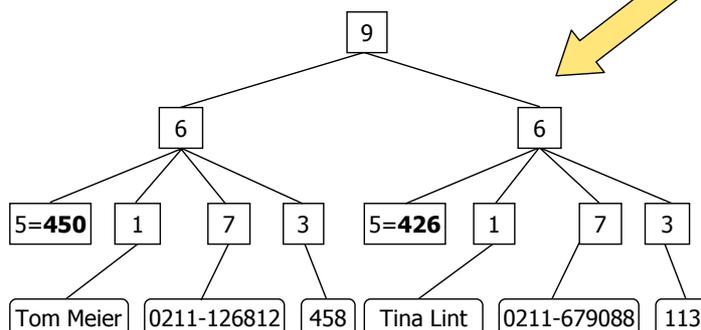


# Transformation eines XML-Dokuments (Bsp. DB2)

Transformation in einen internen XML-Baum; Elementnamen werden mithilfe eines Wörterbuchs (Dictionary) ersetzt



```
<abtteilung>
  <angestellter id="450">
    <name> Tom Meier </name>
    <tel>0211-126812</tel>
    <raum>119</raum>
  </angestellter>
  <angestellter id="426">
    <name>Tina Lint</name>
    <tel>0211-679088</tel>
    <raum>113</raum>
  </angestellter>
</abtteilung>
```



SYSIBM:SYSXMLSTRINGS

String table	
9	abtteilung
6	angestellter
1	name
5	id
7	tel
3	raum

Datenorganisation	Kollektionen von Dokumenten	DB2 Viper MS SQLServer ...	10 <sup>5</sup> XSDs
	einzelne isolierte Dokumente	XTC, Natix Timber	1 XSD
		ohne explizites Schema	schemabasiert
		Dokumente	

- „Ganze Welt“ ein einziges Dokument
  - heterogene Teilbäume
  - typische Index-unterstützte Ops: XPath-Verarbeitung, TWIGs, ...
  - unterschiedliche Sprachmodelle DOM, SAX, XPath, XQuery
- Kollektion von kleineren (<1MB) Dokumenten
  - heterogen
  - Filterung durch Indexeinsatz
  - XQuery, XSQL, SQL/XML
- Flexibilität und Schemaevolution
  - schemaloser Ansatz (Konsistenzgarantien, Suche?)
  - multiple oder sich entwickelnde Schemata oder
  - Schemata mit Erweiterungspunkten (Really Simple Syndication, Atom, ...)
- DB-kontrollierte Konsistenz
  - XSD-Entwurf zuerst
  - Schema-unterstützte Suche
  - strenge Konsistenzprüfung
  - Wie geht man mit 10<sup>5</sup> Schemata um?

- 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

## 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) >`

## Mögliche Unterelemente

- Name des Elements
- `#PCDATA` (parsed character data), d.h., beliebige 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, Adresse+, (Alter | Größe) )
  <!ELEMENT Name (#PCDATA)
  <!ELEMENT Adresse (#PCDATA | (Straße, PLZ, Stadt) )
  ...
```

## Für jedes Attribut

- Name
- Attributtyp
  - Zeichenkette (CDATA) oder Name Token
  - Aufzählungstyp
  - ...
- Vorkommen
  - Attribut(wert) muss vorhanden sein, oder
  - Attribut ist optional, oder
  - Defaultwert

## Beispiel

- `<!ATTLIST Größe Maß (cm | inches) #REQUIRED>`

## 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 Adressenelemente referenzieren, wenn diese eine ID hätten ...

```
<!DOCTYPE Personen [
  <!ELEMENT Personen (Person*)>
  <!ELEMENT Person (Name, Adresse+, (Alter|Größe))>
  <!ATTLIST Person oid ID
                    arbeitetFür IDREF>
  <!ELEMENT Name (#PCDATA)>
  <!ELEMENT Adresse (#PCDATA|(Straße, PLZ Stadt))>
  <!ELEMENT Alter (#PCDATA)>
  <!ELEMENT Größe (#PCDATA)>
  <!ATTLIST Größe Gemessen_am CDATA
                    Maß (cm | inches) #REQUIRED>
  <!ELEMENT Straße (#PCDATA)>
  <!ELEMENT PLZ (#PCDATA)>
  <!ELEMENT Stadt (#PCDATA)>
]>
```

- 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
    - lokale Definition oder Referenz auf entsprechenden Namensraum
    - mehrfache Benutzung
- Modularisierung und Wiederverwendung von Schemadefinitionen
- Schemadefinition ist wiederum ein XML-Dokument
- Deutlich komplizierter als DTDs

# Namensräume (Namespaces)

Semistrukturierte Daten

XML-Einführung

Vergleich XML<-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen



- **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 Schemata 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 – Beispiel (I)

Semistrukturierte Daten

XML-Einführung

Vergleich XML<-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen



```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.personen.org" xmlns="http://www.personen.org" >
  <xsd:element name="Personen">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Person" minOccurs="0" maxOccurs="unbounded" >
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Name" type="xsd:string"/>
              <xsd:element name="Adresse" type="AdresseTyp"
                minOccurs="1" maxOccurs="3"/>
              <xsd:choice>
                <xsd:element name="Alter" type="xsd:integer" />
                <xsd:element name="Größe" type="GrößeTyp" />
              </xsd:choice>
            </xsd:sequence>
            <xsd:attribute name="oid" type="xsd:ID" />
            <xsd:attribute name="arbeitetFür" type="xsd:IDREF" />
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

```

<xsd:complexType name="AdresseTyp" mixed="true" >
  <xsd:sequence>
    <xsd:element name="Straße" type="xsd:string"/>
    <xsd:element name="PLZ" type="PLZTyp"/>
    <xsd:element name="Stadt" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="GrößeTyp" >
  <xsd:simpleContent>
    <xsd:extension base="xsd:positiveInteger" >
      <xsd:attribute name="Gemessen_am" type="xsd:date" />
      <xsd:attribute name="Maß" type="MaßTyp" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="PLZTyp" >
  <xsd:simpleContent>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0-9]{5}"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="MaßTyp">
  <xsd:simpleContent>
    <xsd:restriction base="xsd:string">
      <enumeration value="cm"/>
      <enumeration value="inches"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
</xsd:schema>
  
```

## XML-Schema-Dokument

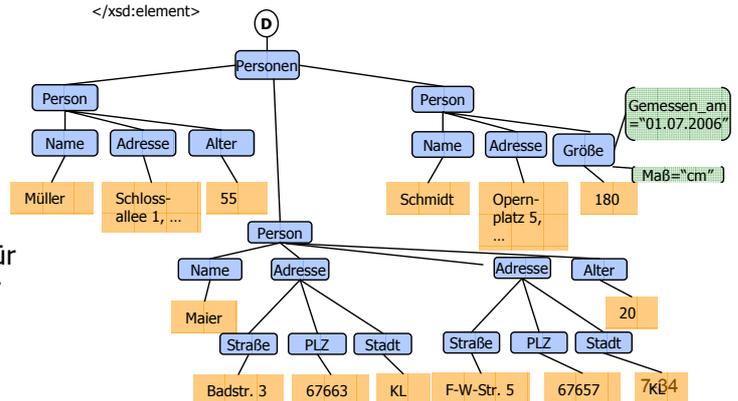
- deklariert Elemente/Attribute bzw. definiert (einfache oder zusammengesetzte) Datentypen
- ordnet (optional) die deklarierten/definierten Konzepte einem Namensraum zu (target namespace)

```

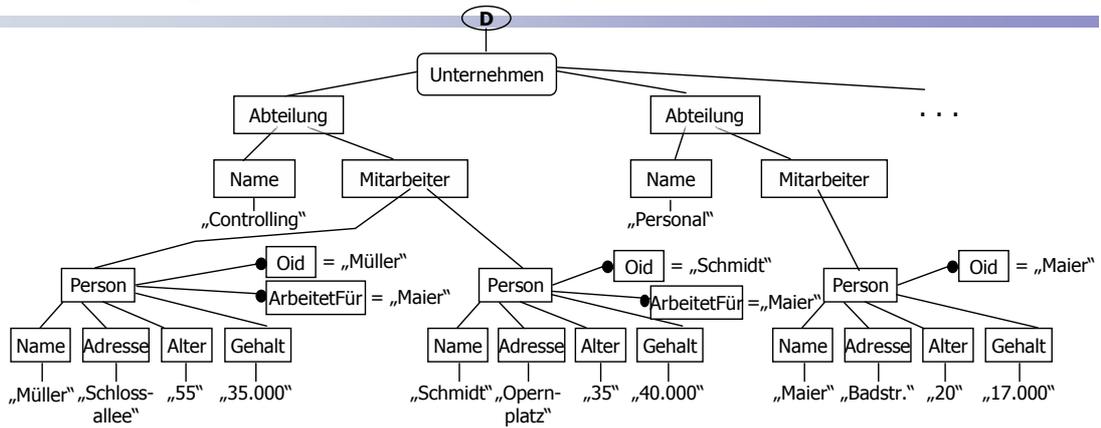
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.personen.org" xmlns="http://www.personen.org" >
  <xsd:element name="Personen">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Person" minOccurs="0" maxOccurs="unbounded" >
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Name" type="xsd:string"/>
              <xsd:element name="Adresse" type="AdresseTyp"
                minOccurs="1" maxOccurs="3"/>
              <xsd:choice>
                <xsd:element name="Alter" type="xsd:integer" />
                <xsd:element name="Größe" type="GrößeTyp" />
              </xsd:choice>
            </xsd:sequence>
            <xsd:attribute name="oid" type="xsd:ID" />
            <xsd:attribute name="arbeitetFür" type="xsd:IDREF" />
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
  
```

## XML-Dokument

- nutzt ein Schema durch Deklaration/Referenz des entsprechenden Namensraums
  - Nutzung mehrerer Schemata im gleichen Dokument
  - Namensraumdefinition für Wurzelement und/oder beliebige Unter-elemente



# 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 geschachtelten 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
- **Semistrukturierter 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)

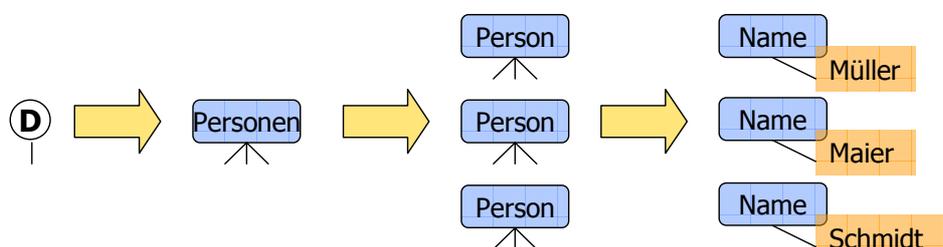
\* W. Lehner, H. Schöning: *XQuery – Grundlagen und fortgeschrittene Methoden*, dpunkt-Verlag, 2004. 7-36

# Datenmodell für XML-Anfragen

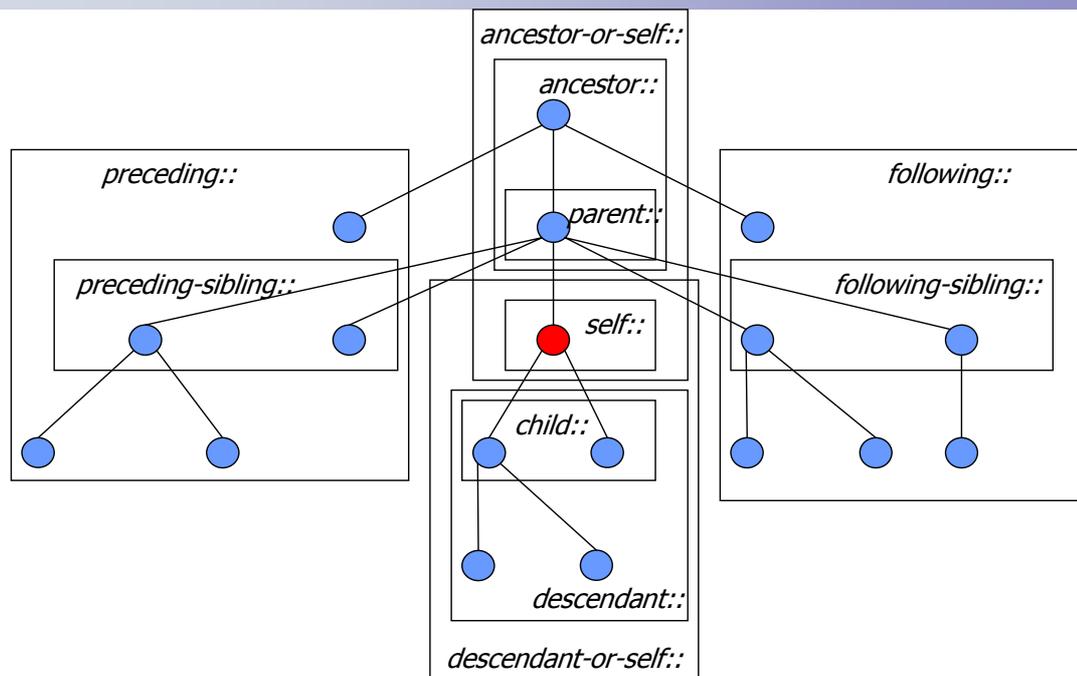
- **Ziel:** Abgeschlossenheit bzgl. der Operationen der Anfrageverarbeitung
  - Baummodell nicht ausreichend
- **Datenmodell in XPath, XQuery**
  - Objekte des Datenmodells sind Sequenzen
    - Sequenz hat 0, 1 oder mehrere Einträge (*items*)
      - Entweder Knoten (*nodes*) oder atomare Werte (*atomic values*)
    - Einträge sind geordnet
  - Knoten als Sequenzeinträge
    - Repräsentieren Baumstruktur eines 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 adressiert (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
    - Pfadanzug: 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 Resultate erzeugen keine Fehler



- Initialer "/" bezeichnet Dokumentknoten (Wurzel)
  - Ein **Achenschritt** hat i. Allg. drei Bestandteile
    - Achse – beschreibt Navigationsrichtung vom Kontextknoten ausgehend
    - Knotentest – Auswahl von Knoten aufgrund des Namens oder Typs
    - Optionale Prädikate – Weitere Selektion von sich qualifizierenden Knoten
    - Beispiel: Alle Personen älter als 30  
`child::Person[child::Alter > 30]`
- Syntax: **Achse::Knotentest [Prädikat] ...**
- Alternativ: **Filterschritt**
    - Anstelle von Achse::Knotentest kann ein Ausdruck stehen, der für Knoten aus dem Kontext weitere Knoten lokalisiert



- Ausgabe erfolgt in Dokumentenordnung (außer bei positionalen Prädikaten, z. B. `ancestor[1]` liefert "parent")
- Attribute und Namensräume sind hier weggelassen
  - Weitere (künstliche) Achse: *attribute::*

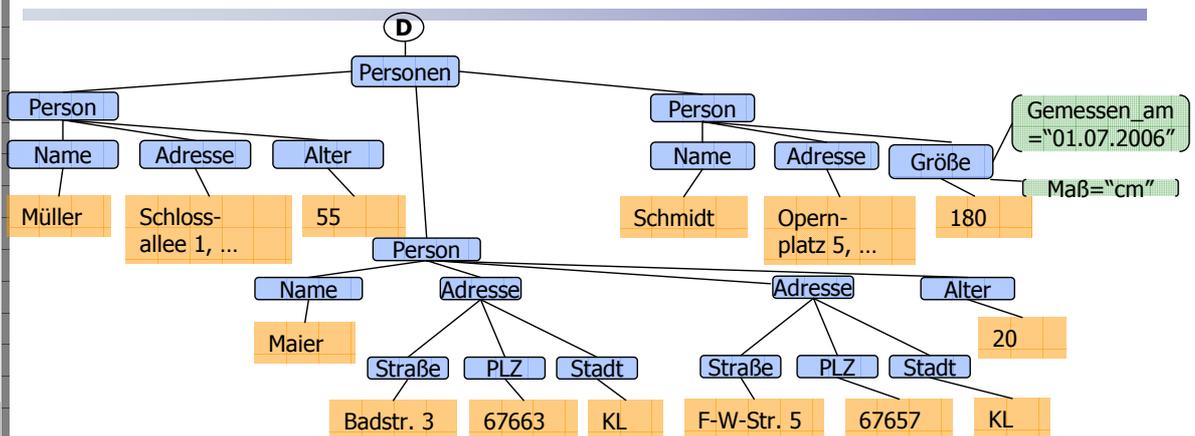
## Namenstest

- Element- oder Attributname
  - `child::name` – `<name>` Elementknoten
  - `child::*` - alle Elementknoten
  - `attribute::name`, `attribute::*` - analog für Attributnamen
- `namespace:name`, `namespace:*` – Beschränkung auf angegebenen Namensraum

## Knotentypstest

- Wählen nur Knoten des entsprechenden 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

- Ausgabe: Sequenz von Items (Item kann Wert oder einer von 7 verschiedenen Knotentypen sein!)

# Pfadausdrücke – Beispiele (2)

Semistrukturierte Daten

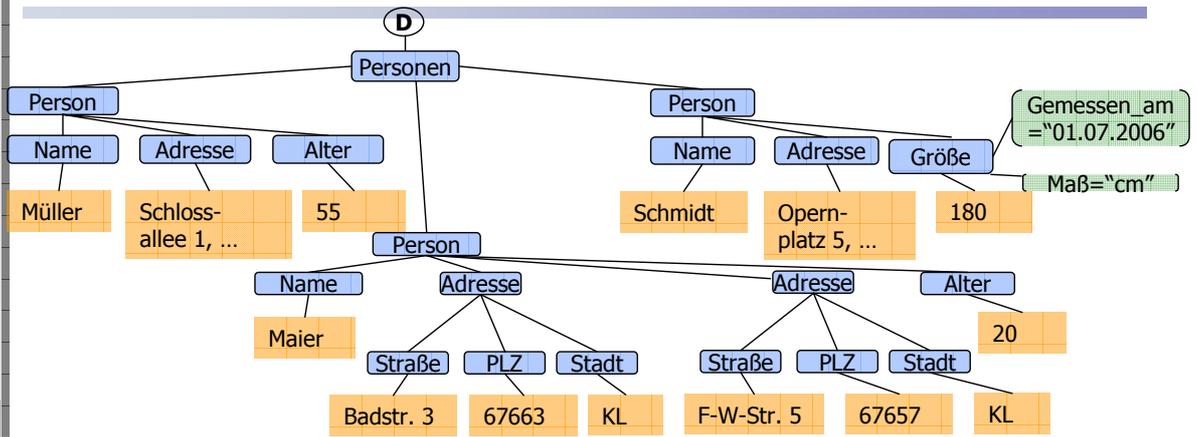
XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen



- `/descendant::Person/child::Größe/attribute::Maß`
  - Maß="cm"
- Was ändert sich bei `(../attribute::*)?`
  - Gemessen\_am="01.07.2006"
  - Maß="cm"
- `/descendant::Adresse/child::*`
  - `<Straße>Badstraße 3</Straße>`
  - `<PLZ>67663</PLZ>`
  - `<Stadt>KL</Stadt>`
  - `<Straße>F.-W.-Straße 5</Straße>`
  - `<PLZ>67657</PLZ>`
  - `<Stadt>KL</Stadt>`
  - ...
  - `child::*` qualifiziert **nur** Elementknoten
- `/descendant::Adresse`
  - wie ändert sich die Ausgabe?

# Prädikate

Semistrukturierte Daten

XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen



- Wertevergleiche (und viele Funktionen) betrachten nicht Knoten, sondern deren (getypte) Werte
  - "Atomisierung"
- Boolesche Ausdrücke
  - `/descendant::Person[child::Name = "Maier"]`
    - logische Konnektoren unterstützt
- Numerische Ausdrücke
  - `/descendant::Person[2]`
    - > liefert das zweite Personenelement
- Existenztests
  - `/descendant::Person[child::Größe]`
    - > Personenelemente, die ein Element „Größe“ besitzen
  - `/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

Semistrukturierte Daten

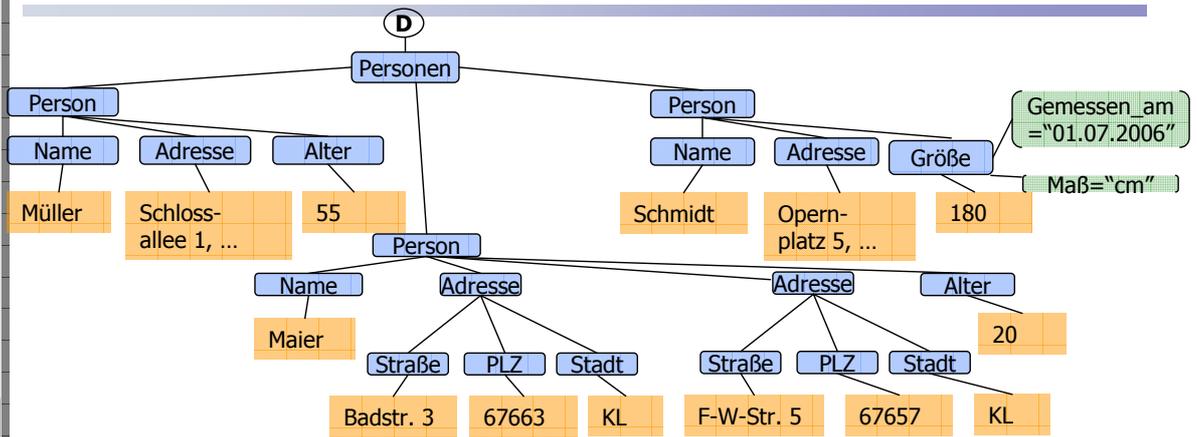
XML-Einführung

Vergleich XML<-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen



## Nutzung von Funktionen

- `/descendant::Person[child::Name="Maier"]  
/fn:string(child::Adresse[2])`

- `string()` liefert textuellen Wert eines Knotens

> *F.-W.-Str. 567657KL*

- `string()` funktioniert **nur**, wenn Sequenz aus einem Element besteht

# Nutzung von Funktionen (2)

Semistrukturierte Daten

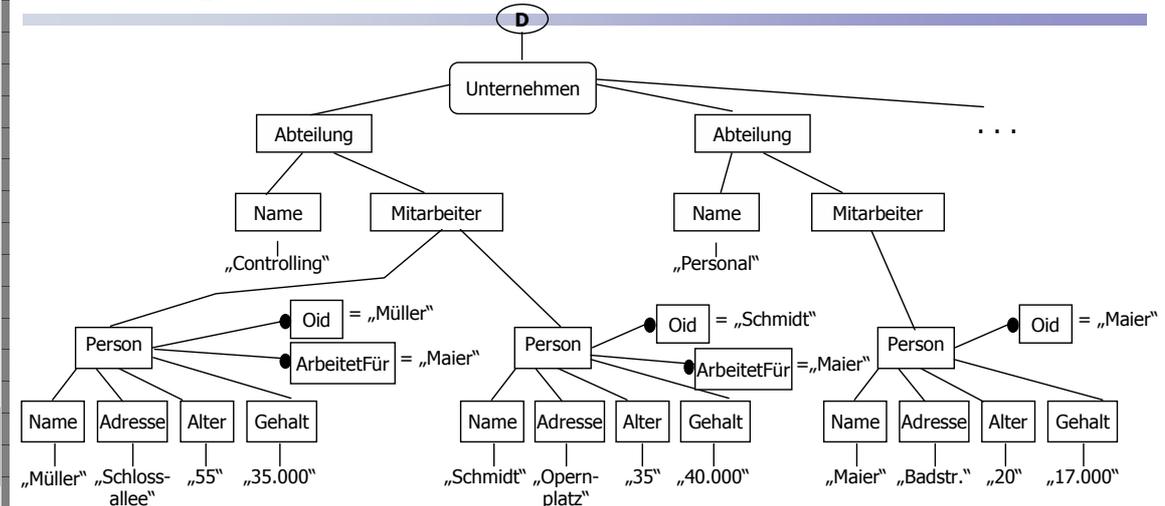
XML-Einführung

Vergleich XML<-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen



## Nutzung von Funktionen

- `/descendant::Person  
[fn:id(attribute::ArbeitsFür)/child::Alter < 30]/child::Name`

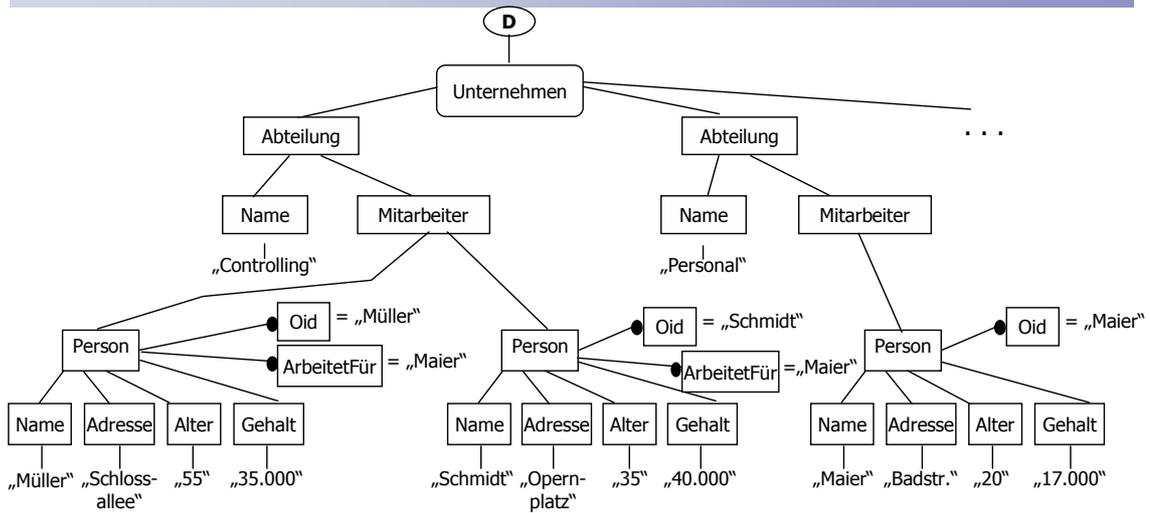
- Funktion `id()` liefert Elemente mit angegebenen ID-Wert

- > *Namen der Personen, die für Personen jünger als 30 arbeiten*

- `/descendant::Person[child::Alter > 30]  
/fn:id(attribute::ArbeitsFür)/child::Name`

- Verwendung von `id()` in Filterschritt

- > *Name von Personen, für die Personen älter als 30 arbeiten*



## Zugriff auf externe Dokumente

- `fn:doc("Personen.xml")/descendant::*`
  - > alle Elemente im angegebenen Dokument
    - auf beliebiger Schachtelungsebene
- **Vorsicht: Was bedeutet „jedes Element auf jeder Ebene wird ausgegeben“?**

- 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`

- 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 zusätzliche Konzepte in XQuery
  - Konstruktoren
  - FLWOR-Ausdrücke (gespr.: *flower*)

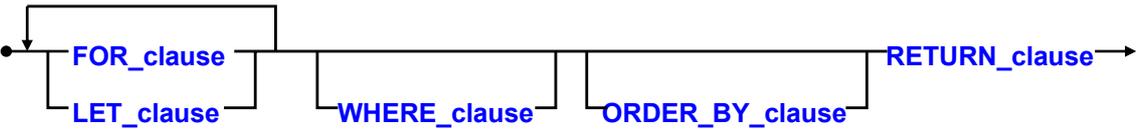
- Nutzung der XML-Dokumentsyntax, um Dokumentfragmente zu konstruieren
 

```
<buch isbn = "12345">
  <titel>Huckleberry Finn</titel>
</buch>
```
- Geschweifte Klammern, um Resultate eines Anfrageausdrucks als Inhalt zu berücksichtigen
 

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

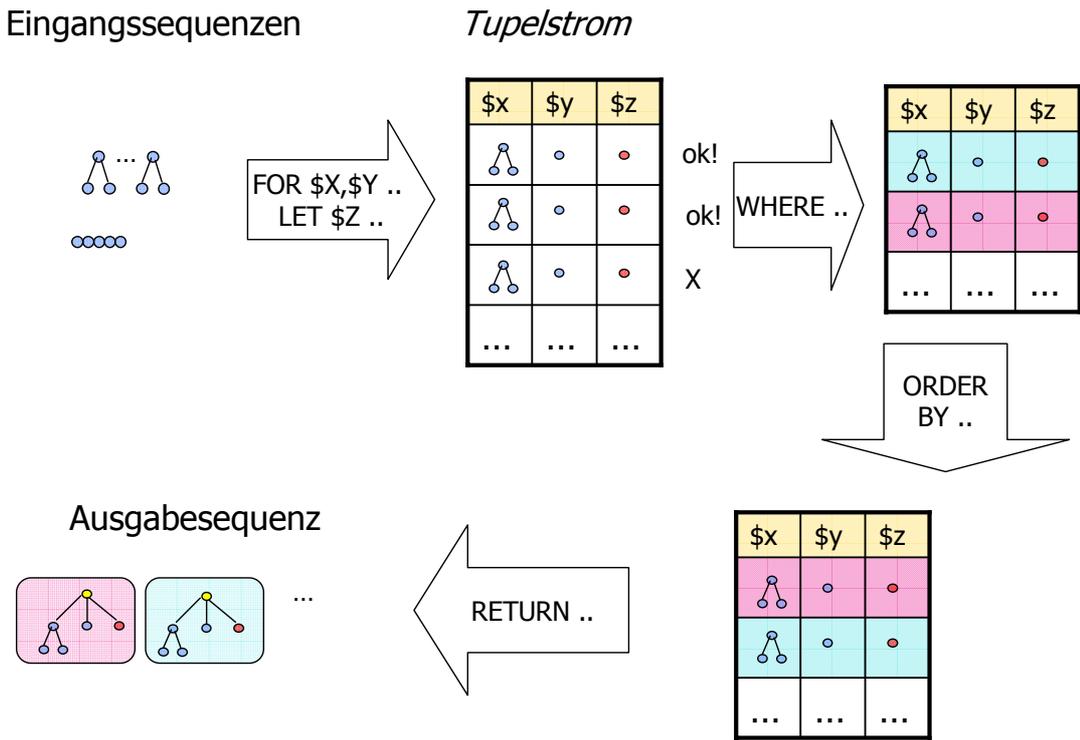
  - Variablenbelegungen sind hier durch umgebenden Ausdruck vorgegeben
  - Dynamische Berechnung von Namen und Inhalten möglich.
    - element {*name-expr*} {*content-expr*}
    - attribute {*name-expr*} {*content-expr*}

# FLWOR – Ausdrücke



- FOR, LET binden Variablen durch
  - Iteration über eine Sequenz von Knoten (FOR)
  - Auswertung eines Ausdrucks (LET): Ausdruck liefert Sequenz von Items, welche an Variable gebunden wird
- 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



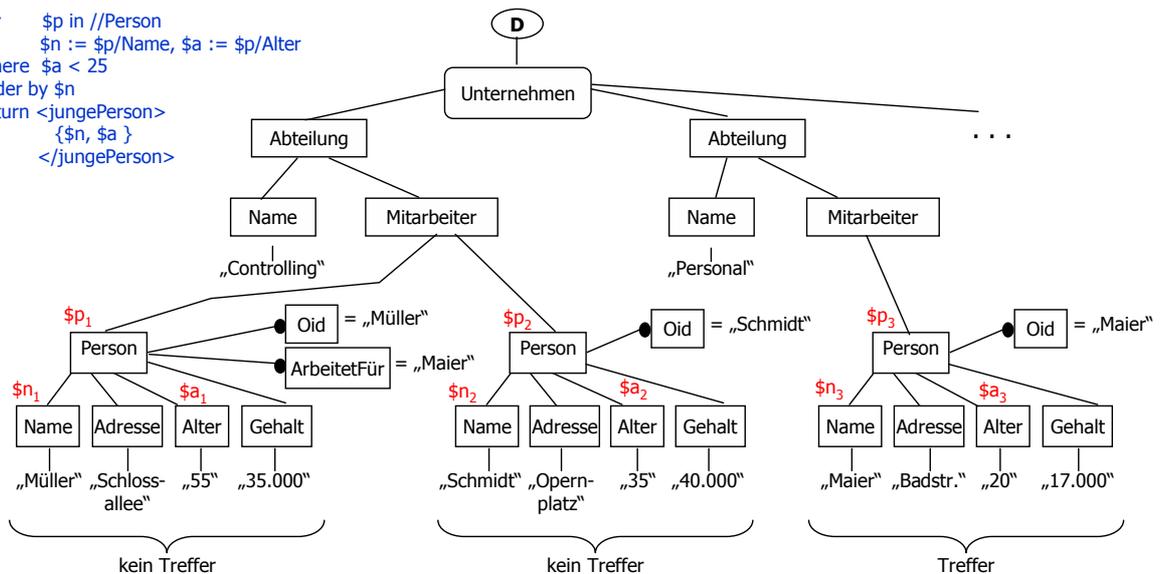
- Suche alle Personen (Name, Alter) jünger als 25
  - Variablen beginnen mit "\$"-Präfix
  - Anfrage ohne LET, WHERE:
 

```
for $p in //Person[Alter < 25]
order by $p/Name
return <jungePerson> {$p/Name, $p/Alter} </jungePerson>
```
  - Einfache (äquivalente) FLWOR-Ausdrücke (siehe Auswertungsbeispiel 1: Sequenzen Name und Alter bestehen in allen Fällen nur aus einem Element)
    - for \$p in //Person
 

```
let $n := $p/Name, $a := $p/Alter
where $a < 25
order by $n
return <jungePerson> {$n, $a } </jungePerson>
```
    - for \$p in //Person[Alter < 25]
 

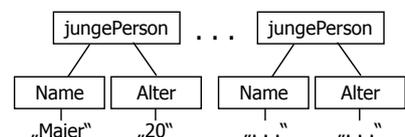
```
let $n := $p/Name, $a := $p/Alter
order by $n
return <jungePerson> {$n, $a } </jungePerson>
```

```
for $p in //Person
let $n := $p/Name, $a := $p/Alter
where $a < 25
order by $n
return <jungePerson>
  {$n, $a }
</jungePerson>
```



- 1) \$p iteriert über alle „Person“-Elemente und bindet jeweils die Kinder „Name“ und „Alter“ an \$n bzw. \$a.
- 2) Bei jedem Iterationsschritt wird überprüft, ob „where“-Bedingung zutrifft
- 3) Treffer werden in einer Sequenz gespeichert und nach \$n (Namen) sortiert
- 4) Ausgabe wird generiert

### Ausgabe



## Erleichtert Sequenz-Auswertung

- ```

for $p in //Person
let $n := $p/Adresse,
    $a := $p/Alter
where fn:count($n) > 2
return ...

```
- ```

for $p in //Person
let $n := $p/Adresse,
    $a := $p/Alter
where $n = „Badstraße“
return ...

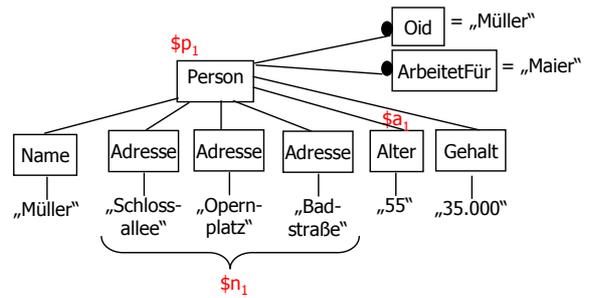
```
- ```

for $p in //Person
let $n := $p/Adresse,
    $a := $p/Alter
where $n EQ „Badstraße“
return ...

```
- ```

for $p in //Person
let $n := $p/Adresse,
    $a := $p/Alter
where $a < 25
return ...

```



- ### Semistrukturierte Daten

  - selbstbeschreibend (Integration von Schema und Daten)
- ### XML

  - Meta-Sprache zur 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