

7. Semistrukturierte Daten und XML

Stefan DeBloch

Überblick

- **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

- 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	Adresse	Alter



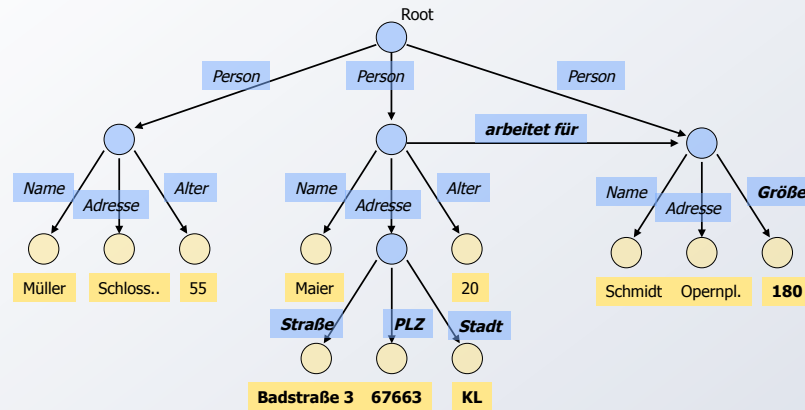
Müller	Schlossallee 1, ...	55
Maier	Badstraße 3, ...	20
Schmidt	Opernplatz 5, ...	35

Semistrukturierte Daten

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



- 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 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

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 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

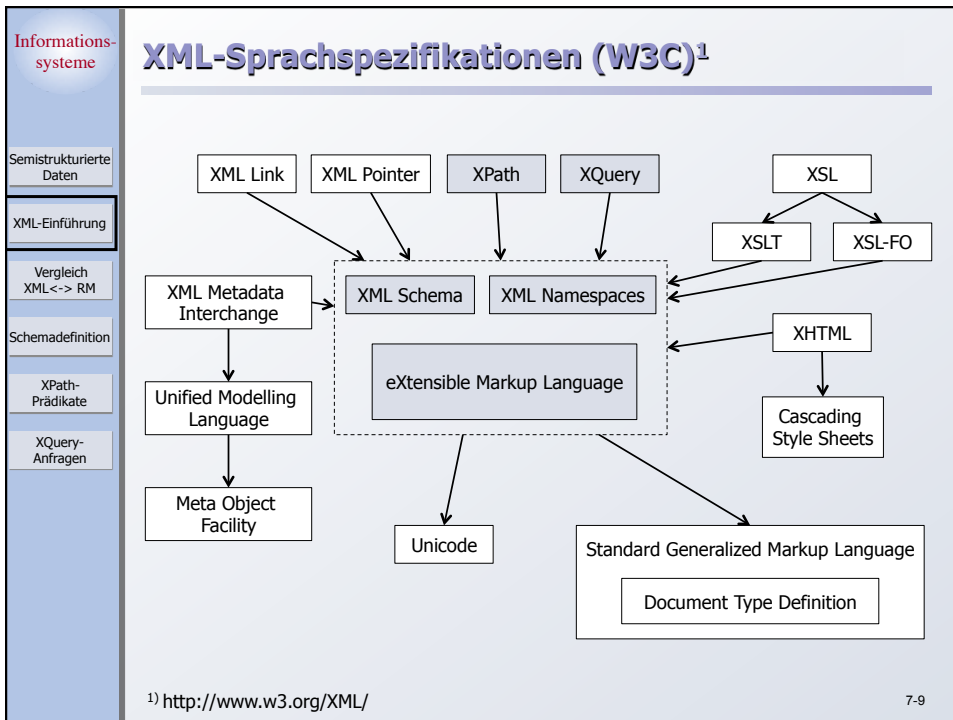
Nutzung von XML

■ Dokumentorientierte Sicht

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

■ Datenorientierte Sicht

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



Informationssysteme

Semistrukturierte Daten

XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen

XML-Dokumente

- sind Text (Unicode)
 - Markup (beginnt immer mit '<' oder '&')
 - (Start-/Ende-) Tags (z.B. <Person>, </Person>)
 - Referenzen (<, &, ...)
 - Deklarationen, Kommentare, Verarbeitungsanweisungen, ...
 - Daten (character data)
 - Zeichen '<' oder '&' müssen im Text durch Referenzen (z.B. <) 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.)

7-10

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 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

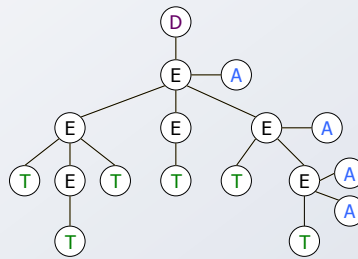
↪ Welches Konzept soll man wann nutzen?

Beispiel

```
<?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>
```

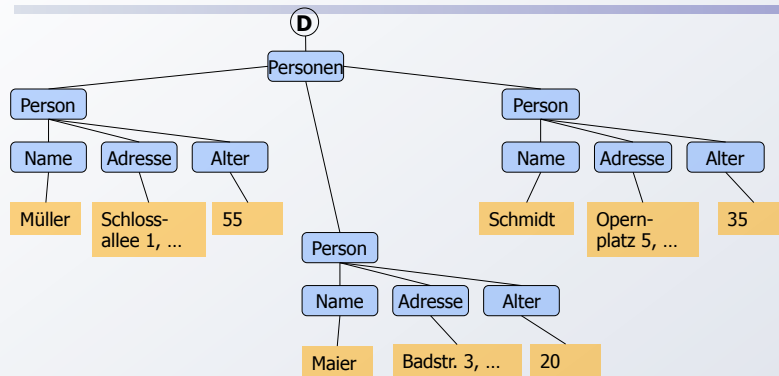
XML-Datenmodell

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



- Konzeptuelle Repräsentation: Bäume mit Knoten und Kanten
- Dokumentenordnung muss erhalten bleiben / wiederherstellbar sein: **Knotenordnung ist wichtig!**

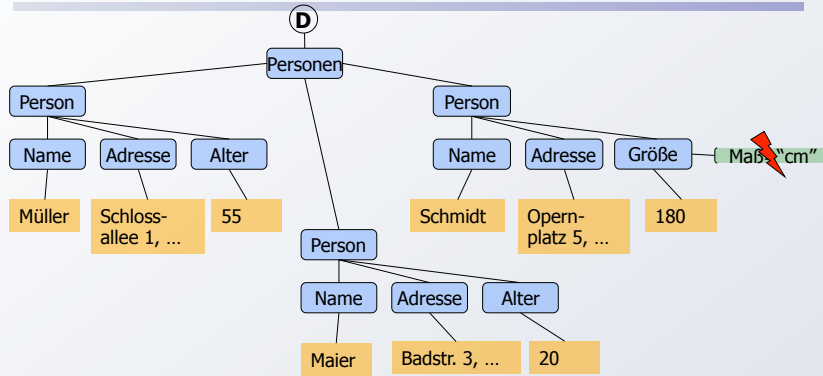
Beispiel: XML ↔ Relationenmodell



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

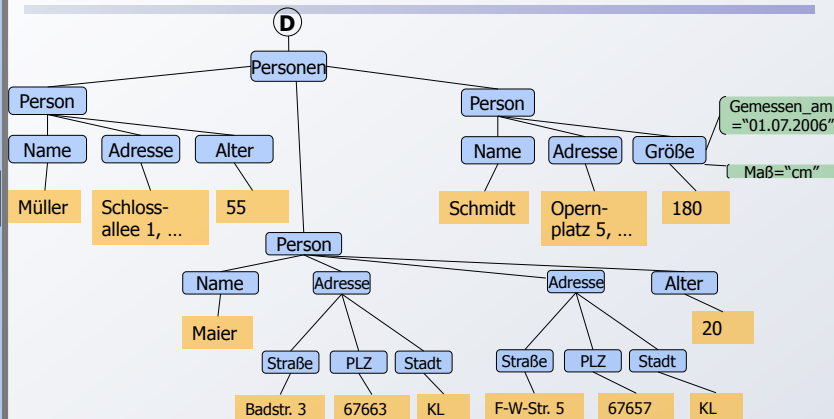


Person			
Name	Adresse	Alter	Größe
Müller	Schlossallee 1, ...	55	--
Maier	Badstraße 3, ...	20	--
Schmidt	Opernplatz 5, ...	--	180

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)



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

Erweitertes Beispiel: XML ↔ Relationenmodell (3)

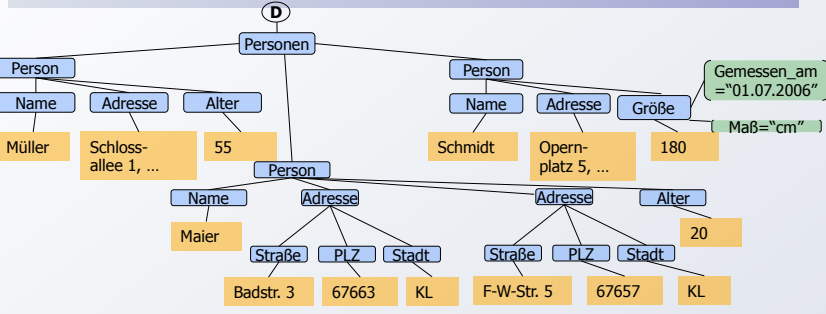


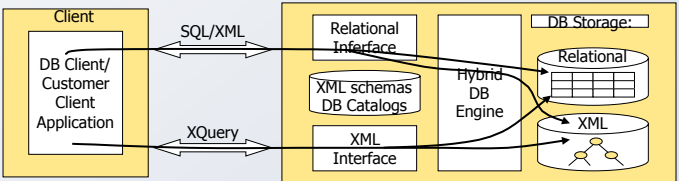
Abbildung im RM über mehrere Tabellen

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

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 ↔ 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

Informationssysteme

Semistrukturierte Daten

XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen

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, ...

7-19

Informationssysteme

Semistrukturierte Daten

XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen

Relationale, native oder hybride Speicherung?

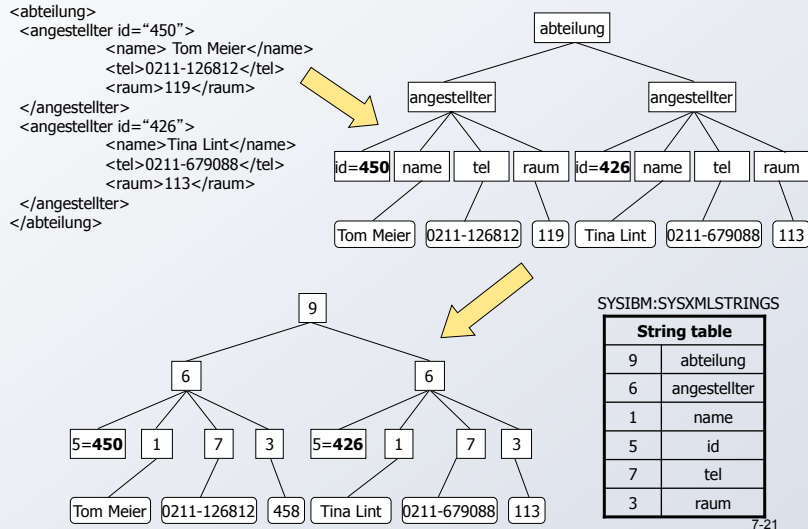
- Speicherung als „**lange Felder**“ (LOBs) ermöglichen keine feinkörnige Verwaltung, keine inhaltsbasierte Suche und keine Mehrbenutzer-Verarbeitung
- **Abbildung auf relationale Tabellen?**
 - dazu gibt es viele Lösungen: „Shredding“
 - **Leistungsverhalten** (Suche, IUD-Ops) ungeklärt
 - XML-Anfragesprachen (z.B. XQuery, XPath, DOM, SAX) sind auf SQL abzubilden
 - Nutzung des SQL-Optimizers!
 - Aber: Kontrolle des Mehrbenutzerbetriebs (Sperrern) ist sehr kompliziert, weil ein Dokument über n Tabellen verteilt sein kann
- **Hybrider Ansatz**
 - Beispiel: DB2-Viper `create table abt (ID char(8), ..., xmldoc xml)`
 - 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

ID	...	xmldoc
„V18“	...	<abteilung> ... <name> ... </name> </abteilung>
...

7-20

Transformation eines XML-Dokuments (Bsp. DB2)

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



7-21

Vogelperspektive – XML-Datenbankverwaltung

Datenorganisation	DB2 Viper MS SQLServer	10 ⁵ XSDs
Kollektionen von Dokumenten	...	
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 10⁵ Schemata um?

7-22

Informations- systeme Semistrukturierte Daten XML-Einführung Vergleich XML<-> RM Schemadefinition XPath- Prädikate XQuery- Anfragen	<h2>Schemadefinition für XML-Dokumente</h2>
	<ul style="list-style-type: none"> ■ XML-Dokument kann (optional) Schema besitzen <ul style="list-style-type: none"> • standardisierter Datenaustausch, ...
	<ul style="list-style-type: none"> ■ Schema schränkt die für das Dokument erlaubten Strukturen und Datentypen ein <ul style="list-style-type: none"> • Dokument heisst gültig (valid), falls es die Anforderungen des Schemas erfüllt • rekursive Schemadefinitionen erlaubt <ul style="list-style-type: none"> - Bauteil besteht aus weiteren Bauteilen ...
	<ul style="list-style-type: none"> ■ Zwei wichtige Ansätze <ul style="list-style-type: none"> • Document Type Definition (DTD) <ul style="list-style-type: none"> - im Dokument enthalten, oder - in einer separaten Datei gespeichert, im Dokument referenziert • XML Schema
	7-23

Informations- systeme Semistrukturierte Daten XML-Einführung Vergleich XML<-> RM Schemadefinition XPath- Prädikate XQuery- Anfragen	<h2>XML Document Type Definition (DTD)</h2>
	<ul style="list-style-type: none"> ■ Definition von Elementtypen <ul style="list-style-type: none"> • Welche Elemente dürfen vorkommen • Welche Attribute darf bzw. muss ein Element haben • Welche Unterelemente dürfen bzw. müssen in einem Element auftreten, und wie oft
	<ul style="list-style-type: none"> ■ DTD bietet keine Unterstützung für Datentypen <ul style="list-style-type: none"> • Daten sind, wie gehabt, nur Zeichenketten ohne weitere Einschränkungen
	<ul style="list-style-type: none"> ■ DTD-Syntax <ul style="list-style-type: none"> • <!ELEMENT element (subelements-specification) > • <!ATTLIST element (attributes) >
	7-24

Informations-
systeme

Semistrukturierte
Daten

XML-Einführung

Vergleich
XML<-> RM

Schemadefinition

XPath-
Prädikate

XQuery-
Anfragen

Elementspezifikation

- **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) )
  ...

```

7-25

Informations-
systeme

Semistrukturierte
Daten

XML-Einführung

Vergleich
XML<-> RM

Schemadefinition

XPath-
Prädikate

XQuery-
Anfragen

Attributspezifikation

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

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

7-26

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 Adressenelemente referenzieren, wenn diese eine ID hätten ...

7-27

Document Type Definition – Beispiel

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

7-28

Informations- systeme	<h2 style="text-align: center;">Schemabeschreibung mit XML Schema</h2>
Semistrukturierte Daten	<ul style="list-style-type: none"> ■ Unterstützung von Datenmodellierungskonzepten, u.a. <ul style="list-style-type: none"> • Datentypen <ul style="list-style-type: none"> - integer, string, ... • Constraints <ul style="list-style-type: none"> - min/max-Werte für mgl. Anzahl von Elementwiederholungen - Werteindeutigkeit (unique), Fremdschlüssel • Getypte Referenzen • Namensräume in XML (name spaces) • Benutzerdefinierte Typen <ul style="list-style-type: none"> - lokale Definition oder Referenz auf entsprechenden Namensraum - mehrfache Benutzung ■ Modularisierung und Wiederverwendung von Schemadefinitionen ■ Schemadefinition ist wiederum ein XML-Dokument ■ Deutlich komplizierter als DTDs
XML-Einführung	
Vergleich XML<-> RM	
Schemadefinition	
XPath- Prädikate	
XQuery- Anfragen	
	7-29

Informations- systeme	<h2 style="text-align: center;">Namensräume (Namespaces)</h2>
Semistrukturierte Daten	<ul style="list-style-type: none"> ■ Ziel: Vermeidung von Namenkollisionen bei Nutzung von verschiedenen Vokabularen (Schemata) im gleichen Dokument <ul style="list-style-type: none"> • Beispiel <ul style="list-style-type: none"> - 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 <ul style="list-style-type: none"> • „enthält“ eine Menge von Namen • durch URI „weltweit“ eindeutig identifiziert • kann in einem XML-Dokument oder Element genutzt werden <ul style="list-style-type: none"> - Deklaration als Default-Namespace - Definition und Angabe von “Kürzeln” (prefix) <ul style="list-style-type: none"> ▪ <lit:titel>, <pers:titel>, ...
XML-Einführung	
Vergleich XML<-> RM	
Schemadefinition	
XPath- Prädikate	
XQuery- Anfragen	
	7-30

XML Schema – Beispiel (1)

```
<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 Schema – Beispiel (2)

```
<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 – 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 version="1.0" encoding="UTF-8" ?>
<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 base="xsd:string" />
  <xsd:sequence base="xsd:string" />
  <xsd:element name="Person" minOccurs="0" maxOccurs="unbounded" />
  <xsd:complexType base="xsd:string" />
  <xsd:sequence base="xsd:string" />
  <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 />
```

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

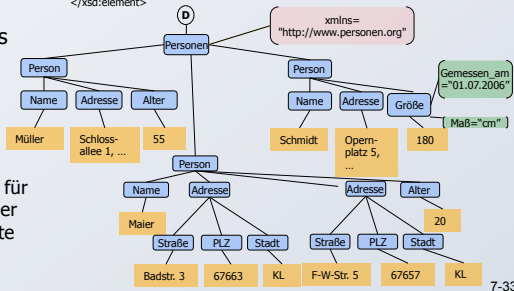
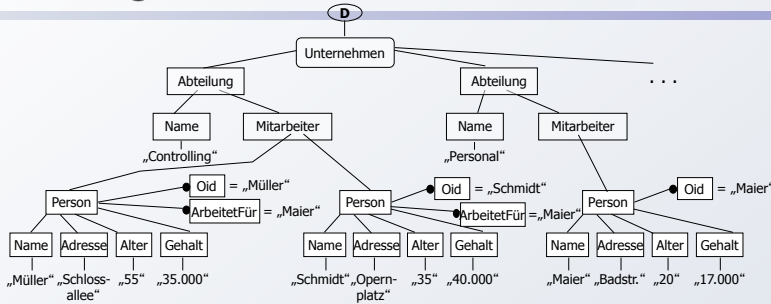


Abbildung ER-Modell -> XML Schema



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

Informations- systeme	<h2>Anfragesprachen</h2>
Semistrukturierte Daten	<ul style="list-style-type: none"> ■ Anfragen auf <ul style="list-style-type: none"> • (großen) XML-Dokumenten • XML-Daten in nativen XML-Datenbanken • logischen XML-Sichten auf beliebigen Datenquellen ■ Semistrukturierter Ansatz erfordert Umgang mit <ul style="list-style-type: none"> • schemalosen Daten/Dokumenten • heterogenen Dokumentstrukturen ■ Funktionalität <ul style="list-style-type: none"> • Pfadausdrücke zur Lokalisierung von Knoten im XML-Baum • Komplexe Anfragen ■ XML-Anfragesprache XQuery * <ul style="list-style-type: none"> • beinhaltet u. a. wesentliche Funktionen von XPath • W3C-Standard
XML-Einführung	
Vergleich XML <-> RM	
Schemadefinition	
XPath- Prädikate	
XQuery- Anfragen	
<p>* W. Lehner, H. Schöning: <i>XQuery – Grundlagen und fortgeschrittene Methoden</i>, dpunkt-Verlag, 2004. 7-35</p>	

Informations- systeme	<h2>Datenmodell für XML-Anfragen</h2>
Semistrukturierte Daten	<ul style="list-style-type: none"> ■ Ziel: Abgeschlossenheit bzgl. der Operationen der Anfrageverarbeitung <ul style="list-style-type: none"> • Baummodell nicht ausreichend ■ Datenmodell in XPath, XQuery <ul style="list-style-type: none"> • Objekte des Datenmodells sind Sequenzen <ul style="list-style-type: none"> - Sequenz hat 0, 1 oder mehrere Einträge (<i>items</i>) <ul style="list-style-type: none"> ▪ Entweder Knoten (<i>nodes</i>) oder atomare Werte (<i>atomic values</i>) - Einträge sind geordnet • Knoten als Sequenzeinträge <ul style="list-style-type: none"> - Repräsentieren Baumstruktur eines Dokuments bzw. Fragments <ul style="list-style-type: none"> ▪ 7 Knotentypen: Element, Dokument, Attribut, Namensraum, Verarbeitungsanweisung, Kommentar, Text - Knoten haben eine Identität (nicht verwechseln mit ID, IDREF) - Ordnung im Baum entspricht Dokumentordnung <ul style="list-style-type: none"> ▪ Eltern < Kinder ▪ Namenräume < andere Attribute < Kinder ▪ Geschwister in Dokumentreihenfolge ■ Anfrageausdrücke operieren auf einer oder mehreren Sequenzen und liefern wieder eine Sequenz
XML-Einführung	
Vergleich XML <-> RM	
Schemadefinition	
XPath- Prädikate	
XQuery- Anfragen	
<p>7-36</p>	

Informationssysteme

Semistrukturierte Daten

XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen

Pfadausdrücke in XPath, XQuery

- Pfadausdruck adressiert (selektiert) eine 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 Resultate erzeugen keine Fehler

7-37

Informationssysteme

Semistrukturierte Daten

XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen

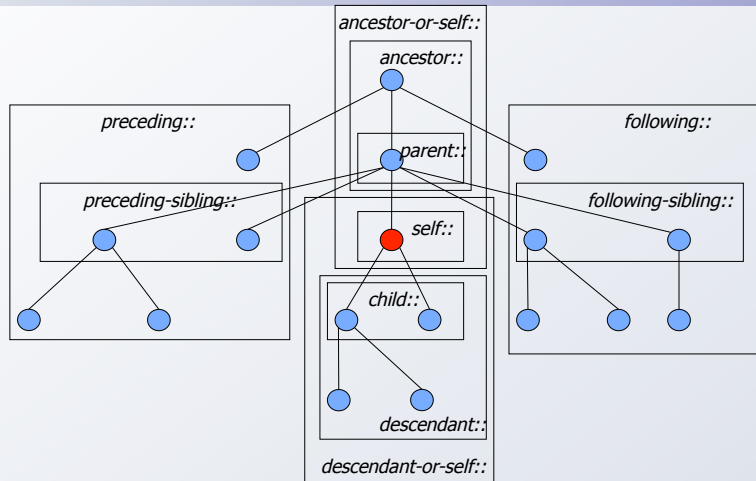
Pfadausdrücke – Schritte

- Initialer “/” bezeichnet Dokumentknoten (Wurzel)
- Ein **Achsen-schritt** 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

7-38

Mögliche Achsen



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

Knotentests

- **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

Semistrukturierte Daten

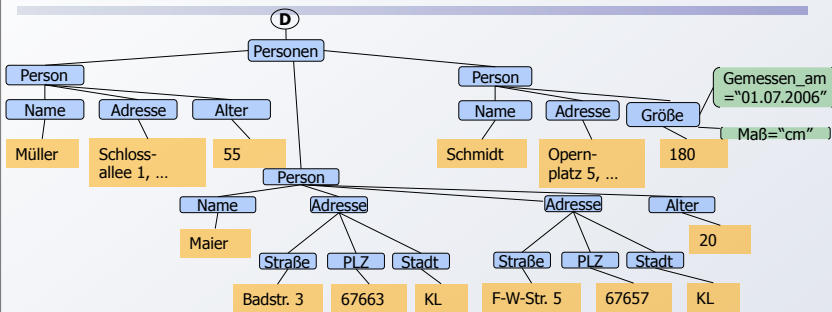
XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen



- `/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

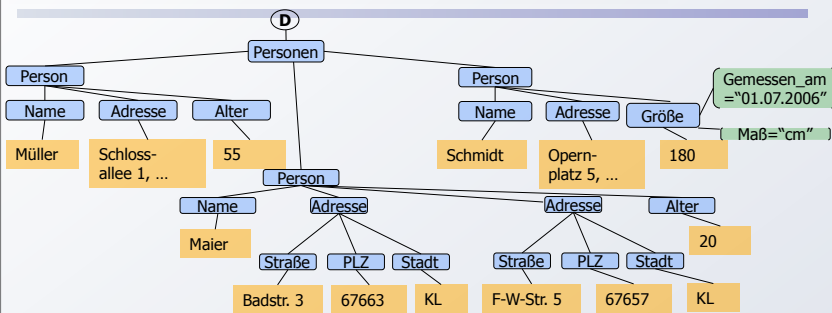
XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen

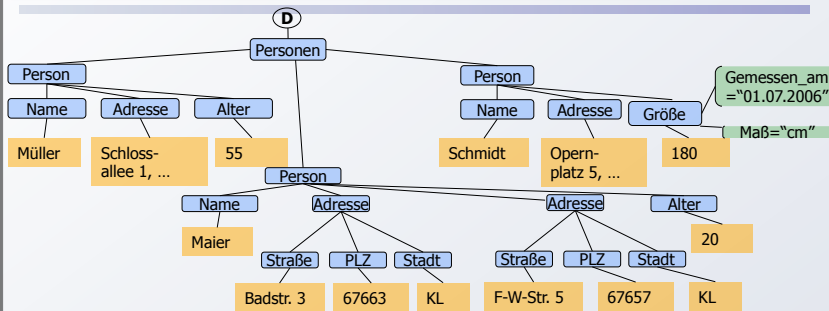


- `/descendant::Person/child::Größe/attribute::Maß`
 - Maß="cm"
- Was ändert sich bei `(.../attribute::*)`?
 - Gemessen_am="01.07.2006"
 - Maß="cm"
- `/descendant::Adresse/child::*`
 - `<Straße>Badstraße 3</Straße>`
`<PLZ>67663</PLZ>`
`<Stadt>KL</Stadt>`
`<Straße>F.-W.-Straße 5</Straße>`
`<PLZ>67657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

Prädikate

- 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



- 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

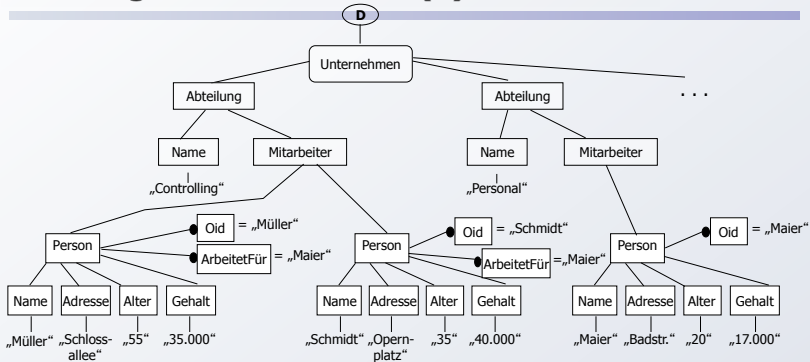
XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen



■ Nutzung von Funktionen

- `/descendant::Person`
`[fn:id(attribute::ArbeitetFür)/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::ArbeitetFür)/child::Name]`
 - Verwendung von `id()` in Filterschritt
 - > *Name von Personen, für die Personen älter als 30 arbeiten*

7-45

Nutzung von Funktionen (3)

Semistrukturierte Daten

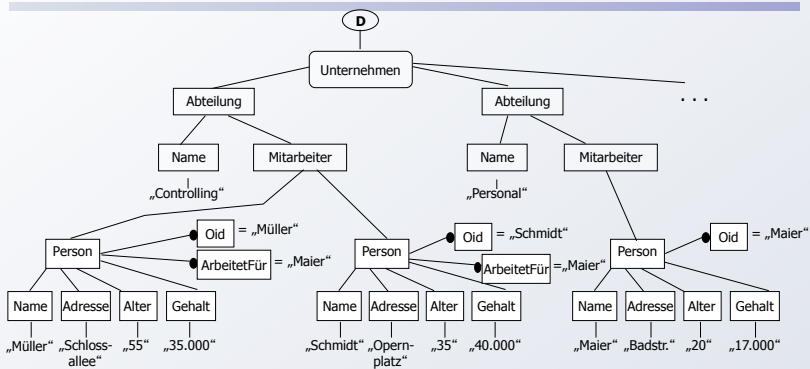
XML-Einführung

Vergleich XML <-> RM

Schemadefinition

XPath-Prädikate

XQuery-Anfragen



■ 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“?**

7-46

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

Konstruktoren

- Nutzung der XML-Dokumentsyntax, um Dokumentfragmente zu konstruieren

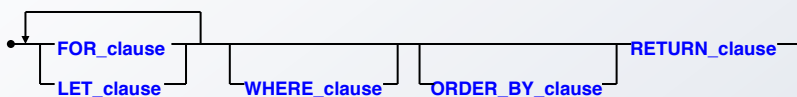
```
<buch isbn = "12345">
  <titel>Huckleberry Finn</titel>
</buch>
```

- 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): 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

Eingangssequenzen



FOR \$X,\$Y ..
LET \$Z ..

Tupelstrom

\$x	\$y	\$z
	o	o
	o	o
	o	o
...

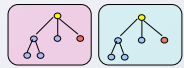
ok!
ok!
X

WHERE ..

\$x	\$y	\$z
	o	o
	o	o
...

ORDER BY ..

Ausgabesequenz



RETURN ..

\$x	\$y	\$z
	o	o
	o	o
...

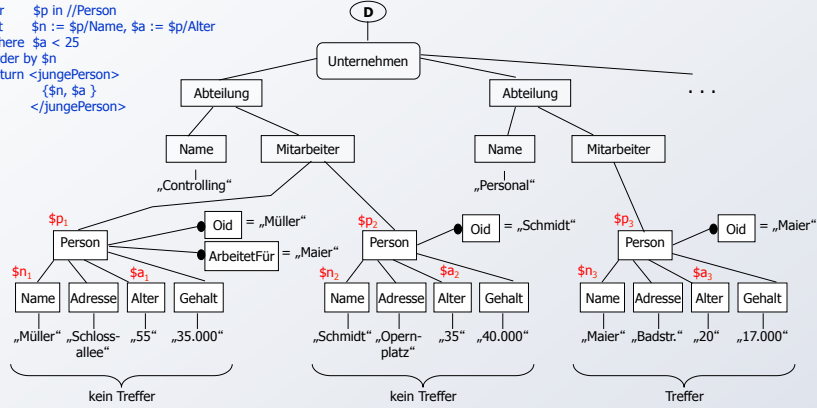
FLWOR – Beispiel

- 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>

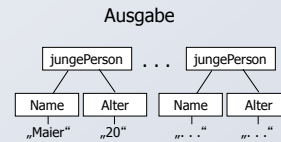
Auswertungsbeispiel 1 – FLWOR-Ausdruck

```

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



Let-Klausel

■ Erleichtert Sequenz-Auswertung

```

for
let
where
return ...
    
```

```

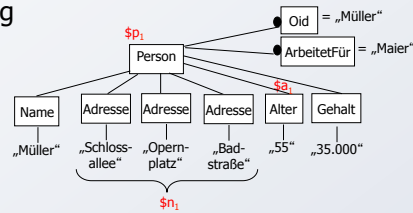
for $p in //Person
let $n := $p/Adresse, $a := $p/Alter
where $n = „Badstraße“
return ...
    
```

```

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 ...
    
```



Verbundoperationen

■ Verfolgung von Referenzen

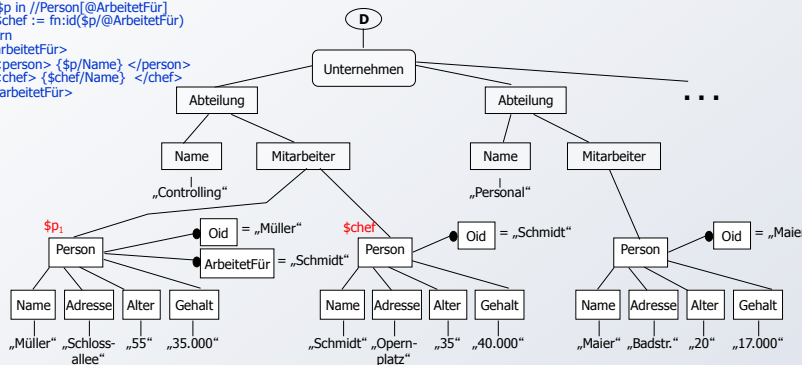
```
for $p in //Person[@ArbeitetFür]
let $chef := fn:id($p/@ArbeitetFür)
return
<arbeitetFür>
  <person> {$p/Name} </person>
  <chef> {$chef/Name} </chef>
</arbeitetFür>
```

■ 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>
```

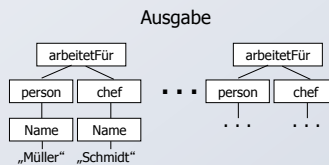
Auswertungsbeispiel 2 – Verfolgung von Referenzen

```
for $p in //Person[@ArbeitetFür]
let $chef := fn:id($p/@ArbeitetFür)
return
<arbeitetFür>
  <person> {$p/Name} </person>
  <chef> {$chef/Name} </chef>
</arbeitetFür>
```



Funktion fn:id() verfolgt ID-Referenzen IDREF und liefert alle XML-Elemente zurück, deren ID-Attribut den referenzierten Wert besitzen. (Attr. vom Typ ID/IDREF müssen im XML-Schema bekannt gegeben werden)

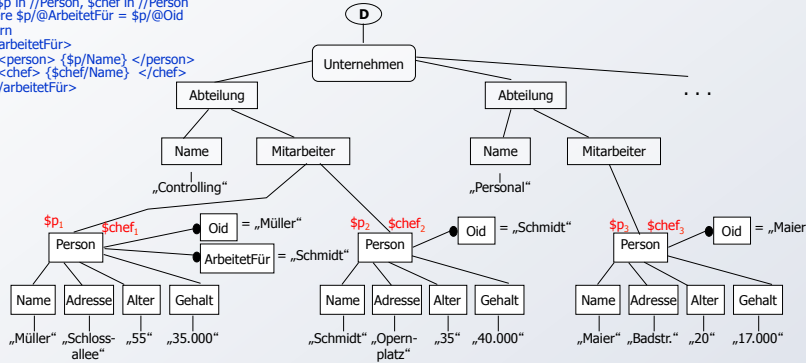
- 1) \$p wird im dargestellten XML-Fragment nur 1x gebunden (an Person Müller), da nur diese Person ein „ArbeitetFür“-ID-Attr. hat
- 2) für \$p wird \$chef an die Person „Schmidt“ gebunden
- 3) Ausgabe wird erzeugt



Auswertungsbeispiel 3 – Symmetrischer Verbund

```
for $p in //Person, $chef in //Person
where $p/@ArbeitetFür = $p/@Oid
return
```

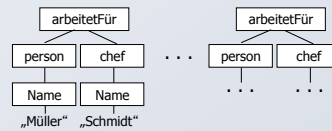
```
<arbeitetFür>
<person> {$p/Name} </person>
<chef> {$chef/Name} </chef>
</arbeitetFür>
```



1) Ausdruck in der for-Klausel ergibt klassischen Nested-Loops-Verbund, wobei jeweils \$p an der äußeren Verbundpartner und \$chef an den inneren Verbundpartner gebunden wird.

2) Für solche (\$p, \$chef)-Paare, für die die where-Bedingung gilt, wird eine Ausgabe erzeugt.

Ausgabe



7-57

Left Outer Join

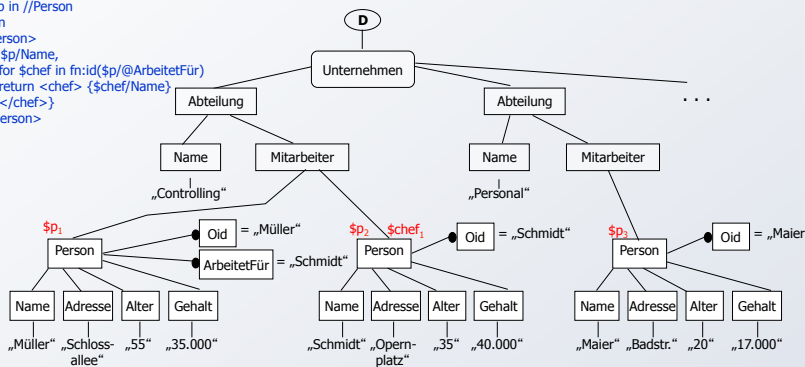
- Beispiel: Liste alle Personen mit Namen und Chef (falls vorhanden)

```
• for $p in //Person
return
  <person>
    { $p/Name,
      for $chef in fn:id($p/@ArbeitetFür)
      return <chef> {$chef/Name} </chef> }
  </person>
```

7-58

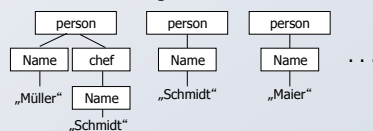
Auswertungsbeispiel 4 – Left Outer Join

```
for $p in //Person
return
<person>
{ $p/Name,
  for $chef in fn:id($p/@ArbeitetFür)
  return <chef> {$chef/Name}
}
</person>
```



- 1) Iteration über alle Personen, Binden von \$p
- 2) Für jede Person wird ein Element „Person“ erzeugt, welches das Element „Name“ der Person enthält
- 3) Falls eine Person ein Attribut „ArbeitetFür“ enthält, wird mit Hilfe der fn:id()-Funktion das „Person“-Element des Chefs an \$chef gebunden und dessen Name innerhalb eines „Chef“-Elements ausgegeben (passiert im dargestellten Fragment nur 1x)

Ausgabe



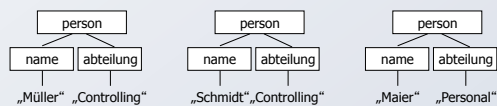
Geschachtelte FLWOR-Ausdrücke

```
for $p in fn:distinct-values(//Person)
order by $p/Name
return
<person>
<name> $p/Name/text() </name>
{ for $a in //Abteilung[./Person = $p]
  return <abteilung> $a/Name </abteilung>}
</person>
```

```
<Abteilung>
<Name>...</Name>
<Mitarbeiter>
  <Person>...<Person>
  <Person>...</Person>
</Mitarbeiter>
</Abteilung>
<Abteilung>
  <Name>
  ...
</Abteilung>
```

```
<person>
  <name>...</name>
  <abteilung>...</abteilung>
</person>
.
.
.
```

Ausgabe:

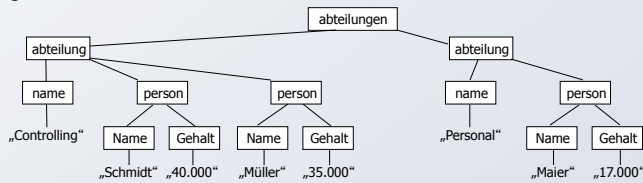


Sortierung & Schachtelung

- Suche alle Abteilungen, sortiert nach Name; liste alle Mitarbeiter absteigend nach Gehalt

```
<abteilungen>
{ for $a in //Abteilung
  order by $a/Name
  return <abteilung>
    <name> $a/Name/text() </name>
    { for $p in $a//Person
      order by $p/Gehalt descending
      return <person>
        { $p/Name }
        { $p/Gehalt }
      }
    }
</abteilungen>
```

Ausgabe:



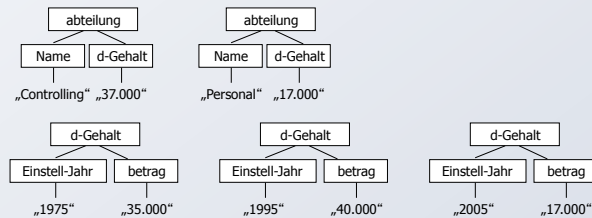
Gruppierung und Aggregation

- entlang einer XML-Hierarchie


```
for $a in //Abteilung
let $g := $a//Person/Gehalt
return
  <abteilung>
    { $a/Name }
    <d-Gehalt> {fn:avg($g)} </d-Gehalt>
  </abteilung>
```
- aufgrund von Werten


```
for $j in fn:distinct-values(//Person/Einstell-Jahr)
let $g := //Person[Einstell-Jahr = $j]/Gehalt
return <d-Gehalt> { $j }
  <betrag> {fn:avg($g)} </betrag>
</d-Gehalt>
```

Ausgaben:



Informations- systeme Semistrukturierte Daten XML-Einführung Vergleich XML<-> RM Schemadefinition XPath- Prädikate XQuery- Anfragen	<h2>Zusätzliche Funktionalität</h2> <ul style="list-style-type: none"> ■ Ausdrücke <ul style="list-style-type: none"> • arithmetische und konditionale Ausdrücke • Mengenoperationen • quantifizierte Ausdrücke • Typumwandlung ■ Funktionen <ul style="list-style-type: none"> • Definition und Nutzung ■ Anfrageverarbeitungsmodell <ul style="list-style-type: none"> • Statische Analyse und dynamische Evaluierung • Nutzung von Typinformation ■ ...
--	---

7-63

Informations- systeme Semistrukturierte Daten XML-Einführung Vergleich XML<-> RM Schemadefinition XPath- Prädikate XQuery- Anfragen	<h2>Zusammenfassung</h2> <ul style="list-style-type: none"> ■ Semistrukturierte Daten <ul style="list-style-type: none"> • selbstbeschreibend (Integration von Schema und Daten) ■ XML <ul style="list-style-type: none"> • Meta-Sprache zur Beschreibung von Dokumenten <ul style="list-style-type: none"> - Struktur und Inhalt - wohlgeformtes (<i>well-formed</i>) XML • Dokumentorientiert vs. Datenorientierte Sicht ■ Schemadefinition für XML Dokumente <ul style="list-style-type: none"> • DTDs <ul style="list-style-type: none"> - rudimentäre, struktur-orientierte Schemata • XML Schema <ul style="list-style-type: none"> - unterstützte effektivere Datenmodellierung mit XML ■ Anfrageverarbeitung mit XML <ul style="list-style-type: none"> • Pfadausdrücke (XPath) <ul style="list-style-type: none"> - wichtiges Konzept zur flexiblen Lokalisierung von Knoten in XML-Bäumen • XQuery <ul style="list-style-type: none"> - komplexe Anfragen und Transformationen auf XML-Daten/Dokumenten
--	---

7-64