

6. Semistrukturierte Daten (XML, JSON)

- Einleitung
- Einführung in XML
 - XML-Dokumente
 - DTD (Document Type Definition)
- XML Schema
 - Typen (simpleType, complexType)
 - Schemaaufbau
 - Element/Attribut-Deklarationen
 - Integritätsbedingungen
 - Nutzung mehrerer Schemas / Namensräume
- XML-Datenbanken, SQL/XML
 - Datentyp XML und XML-Operatoren
 - Datenkonversion relational <-> XML
 - Auswertung von XML-Inhalten: XMLQUERY
- JSON-Format, SQL JSON

Kap. 7: Anfragesprachen:
XPath, XQuery



Strukturiertheit von Daten

Unstrukturiert Prosa-Text	Semi-strukturiert	(stark) strukturiert Datenbanken
-------------------------------------	--------------------------	--

- regelmäßige Struktur durch Schema
 - Datensätze mit gleichartigem Aufbau
 - leichtere und effiziente automatische Verarbeitung
 - weniger flexibel
- semi-strukturierte Daten
 - Kompromiss zwischen unstrukturierten und stark strukturierten Daten
 - hohe Flexibilität
 - Text- und Daten-orientierte *Dokumente* möglich
 - optionales Schema



Repräsentation von semistrukturierten Daten / Dokumenten

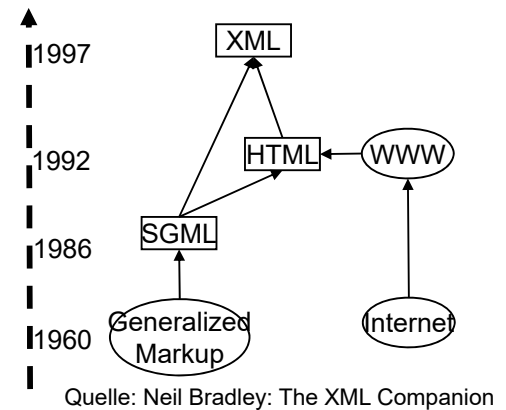
■ Markup-Sprache: Dokumentabschnitte werden durch Marker ausgezeichnet

– SGML (Standard Generalized Markup Language): hohe Komplexität

– HTML (HyperText Markup Language): SGML-Anwendung, feste Menge an Auszeichnungselementen

– XML (eXtensible Markup Language):

- Empfehlung des W3C - World Wide Web Consortium
- SGML-Kompatibilität, jedoch einfacher
- optionale Schemaunterstützung (DTD, XML Schema)
- Anfragemöglichkeiten (XPath, Xquery)
- XML-Datenbanken / SQL-Integration



■ starke XML-Verbreitung u.a. zum Datenaustausch

■ leichtgewichtige XML-Alternative: JSON (JavaScript Object Notation)

– schemalose Repräsentation von Dokumenten / geschachtelten Datenobjekten

– Unterstützung in SQL und durch Document Stores (NoSQL-Datenbanken), zB MongoDB



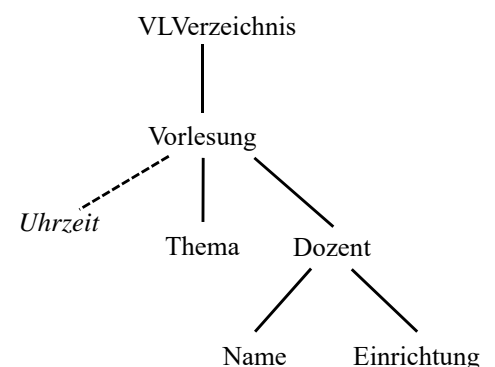
XML - Beispiel

■ XML-Dokumente haben Baumstruktur

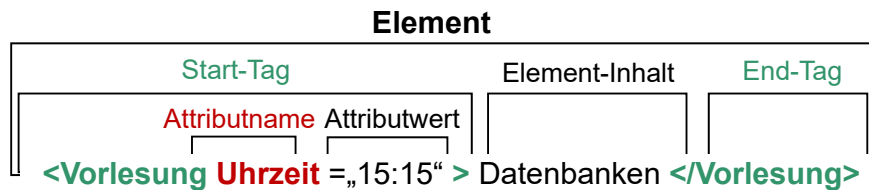
– Schachtelung von Elementen (= Baumknoten)

– Elemente können neben Unterelementen Attribute bzw. Inhalt (z.B. Text) aufweisen

```
<?XML version="1.0"?>
<!DOCTYPE Vorlesungsverzeichnis SYSTEM
"http://dbs.uni-leipzig.de/dtd/VLVerzeichnis.dtd">
<VLVerzeichnis>
  <Vorlesung Uhrzeit="15:15">
    <Thema>DBS2</Thema>
    <Dozent>
      <Name>Prof. Rahm</Name>
      <Einrichtung>Uni Leipzig</Einrichtung>
    </Dozent>
  </Vorlesung>
</VLVerzeichnis>
```



XML: Elemente und Attribute



■ Elemente

- Start- und End-Tag müssen vorhanden sein.
Ausnahme: leeres Element (Bsp. <Leer />)
- feste Reihenfolge von Geschwisterelementen (Reihung ist wichtig!)
- Element-Inhalt besteht entweder nur aus weiteren Elementen (**element content**) oder aus Zeichendaten optional vermischt mit Elementen (**mixed content**)

■ Attribute

- Attributwerte können nicht strukturiert werden
- Attributreihenfolge beliebig

■ Groß-/Kleinschreibung ist relevant (gilt für alle Bezeichner in XML)

■ weitere XML-Bestandteile (hier nicht behandelt): Entities, Processing Instructions, Kommentare



XML-Strukturdefinition: DTD

- **DTD** (**D**ocument **T**ype **D**efinition) / Schema optional
- XML-Dokumente sind **wohlgeformt**, d.h. syntaktisch korrekt
 - alle (außer leere) Elemente müssen ein Start-Tag und ein Ende-Tag haben
 - korrekte Schachtelung von Tags
 - eindeutige Attributnamen pro Element; Attributwerte in Hochkommas ...
- XML-Daten sind **gültig**: wohlgeformt und DTD- bzw- Schema-konform

```
<?XML version="1.0"?>
<!DOCTYPE Vorlesungsverzeichnis SYSTEM
"http://dbs.uni-leipzig.de/dtd/VLVerzeichnis.dtd">
<VLVerzeichnis>
  <Vorlesung Uhrzeit="15:15">
    <Thema>DBS2</Thema>
    <Dozent>
      <Name>Prof. Rahm</Name>
      <Einrichtung>Uni Leipzig</Einrichtung>
    </Dozent>
  </Vorlesung>
</VLVerzeichnis>
```

VLVerzeichnis.dtd:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT VLVerzeichnis (Vorlesung)* >
<!ELEMENT Vorlesung (Thema, Dozent) >
<!ATTLIST Vorlesung
  Uhrzeit CDATA #REQUIRED >
<!ELEMENT Thema (#PCDATA) >
<!ELEMENT Dozent (Name, Einrichtung?) >
<!ELEMENT Name (#PCDATA) >
<!ELEMENT Einrichtung (#PCDATA) >
```



DTD – Document Type Definition

- beschreibt Dokumentstruktur und legt damit einen Dokumenttyp fest
- im Dokument verweist *Dokumenttyp-Deklaration* auf DTD

- interne DTD

```
<!DOCTYPE Vorlesungsverzeichnis [<!ELEMENT VLVerzeichnis (Vorlesung)*>...]>
```

- externe DTD

```
<!DOCTYPE Vorlesungsverzeichnis SYSTEM „http://dbs.uni-leipzig.de/dtd/VLVerzeichnis.dtd“>
```

- Definition von Elementen in einer DTD

- Sequenz: (A , B) - vorgegebene Reihenfolge

```
<!ELEMENT Vorlesung (Thema, Dozent)
```

- Alternative: (A | B) - entweder A oder B (XOR)

```
<!ELEMENT Adresse (PLZ, Ort, (Str, Nr) | Postfach)>
```



DTD (2)

- Element-Wiederholung / Kardinalität:

- A? - 0..1 Mal

```
<!ELEMENT Dozent (Name, Einrichtung?)>
```

- A+ - 1..n Mal

```
<!ELEMENT Name (Vorname+, Nachname)>
```

- A* - 0..n Mal

```
<!ELEMENT VLVerzeichnis (Vorlesung)*>
```

- Mixed Content:

- (#PCDATA / A / B)*

```
<!ELEMENT Text (#PCDATA | Link)*>
```

Elemente A, B und Text treten in beliebiger Reihenfolge und Anzahl auf

- leeres Element (kein Element-Inhalt)

```
<!ELEMENT br EMPTY>
```



DTD (3)

■ Definition von Attributen in einer DTD

```
<!ATTLIST Elementname (Attributname Typ Auftreten)* >
```

■ Attribute gehören zu einem Element

■ jedes Attribut hat Namen, Typ und Auftretensangabe

■ mögliche Attribut-Typen

- CDATA
- ID
- IDREF/IDREFS
- Aufzählung möglicher Werte (wert1 | wert2 | ...)
- NMTOKEN/NMTOKENS, ENTITY/ENTITYS

■ ID- und IDREF-Attribute ermöglichen Querverweise innerhalb eines Dokumentes (Graphstruktur)



DTD (4)

■ Auftretensangabe eines Attributs

- Obligatheit: #REQUIRED - das Attribut muss angegeben werden
- Defaultwert-Regelung: #IMPLIED - es gibt keinen Defaultwert
defaultwert - wird angenommen, wenn Attribut nicht angegeben wird
- #FIXED defaultwert - Attribut kann nur den Defaultwert als Wert besitzen

■ Beispiele:

```
<!ATTLIST Entfernung Einheit CDATA #FIXED „km“>
```

```
<!ATTLIST Karosse Farbe („rot“ | „gelb“ | „blau“) „blau“>
```

```
<!ATTLIST Artikel id ID #REQUIRED>
```

```
<!ATTLIST Rechnungsposten Artikel IDREF #REQUIRED>
```



Elemente vs. Attribute

- Datenmodellierung oft durch Element als auch durch Attribut möglich
- Einsatzkriterien unter Verwendung einer DTD

	Element	Attribut
Inhalte	komplex	nur atomar
Ordnungserhaltung	ja	nein
Quantoren / Kardinalität	1 / ? / * / +	REQUIRED / IMPLIED
Alternativen	ja	nein
Identifikation	nein	ID / IDREF / IDREFS
Defaultwerte	nein	ja
Aufzählungstypen	nein	ja

- bei XML Schema anstelle DTD können Elemente alle Eigenschaften von Attributen erhalten; trotzdem Attribute zu bevorzugen wenn:
 - Aufzählungstypen mit atomaren Werten und Defaultwert zu modellieren sind
 - es sich um 'Zusatzinformation' handelt (Bsp. Währung, Einheit)
 - das Dokument effizient verarbeitet (geparsed) werden soll



6. Semistrukturierte Daten (XML, JSON)

- Einleitung
- Einführung in XML
 - XML-Dokumente
 - DTD (Document Type Definition)
- XML Schema
 - Typen (simpleType, complexType)
 - Schemaaufbau
 - Element/Attribut-Deklarationen
 - Integritätsbedingungen
 - Nutzung mehrerer Schemas / Namensräume
- XML-Datenbanken, SQL/XML
 - Datentyp XML und XML-Operatoren
 - Datenkonversion relational <-> XML
 - Auswertung von XML-Inhalten: XMLQUERY
- JSON-Format, SQL JSON



XML Schema-Sprachen

■ Schemainformationen wichtig für

- Korrektheit von Daten/Dokumenten: Validierung von Struktur und Datentypen
- effiziente DB-Speicherung und Verarbeitung von Anfragen
- Interoperabilität in verteilten Umgebungen: feste Formatierungen für Datenaustausch (z.B. in Schnittstellen für Web-Services)

■ Defizite der DTD

- zusätzliche Syntax (kein XML-Format) -> extra Parser notwendig
- kaum Typisierung von Elementinhalt (Text) und Attributwerten möglich
- mehrere DTDs pro Dokument sind nicht erlaubt
- keine Unterstützung von Namensräumen
- unzureichende Unterstützung der Interoperabilität und Wiederverwendbarkeit



XML Schema*

■ XML Schemabeschreibungssprache, entwickelt durch das W3C

- Recommendation seit 2001 , 2012: V1.1
- auch als *XML Schema Definition (XSD)* bezeichnet

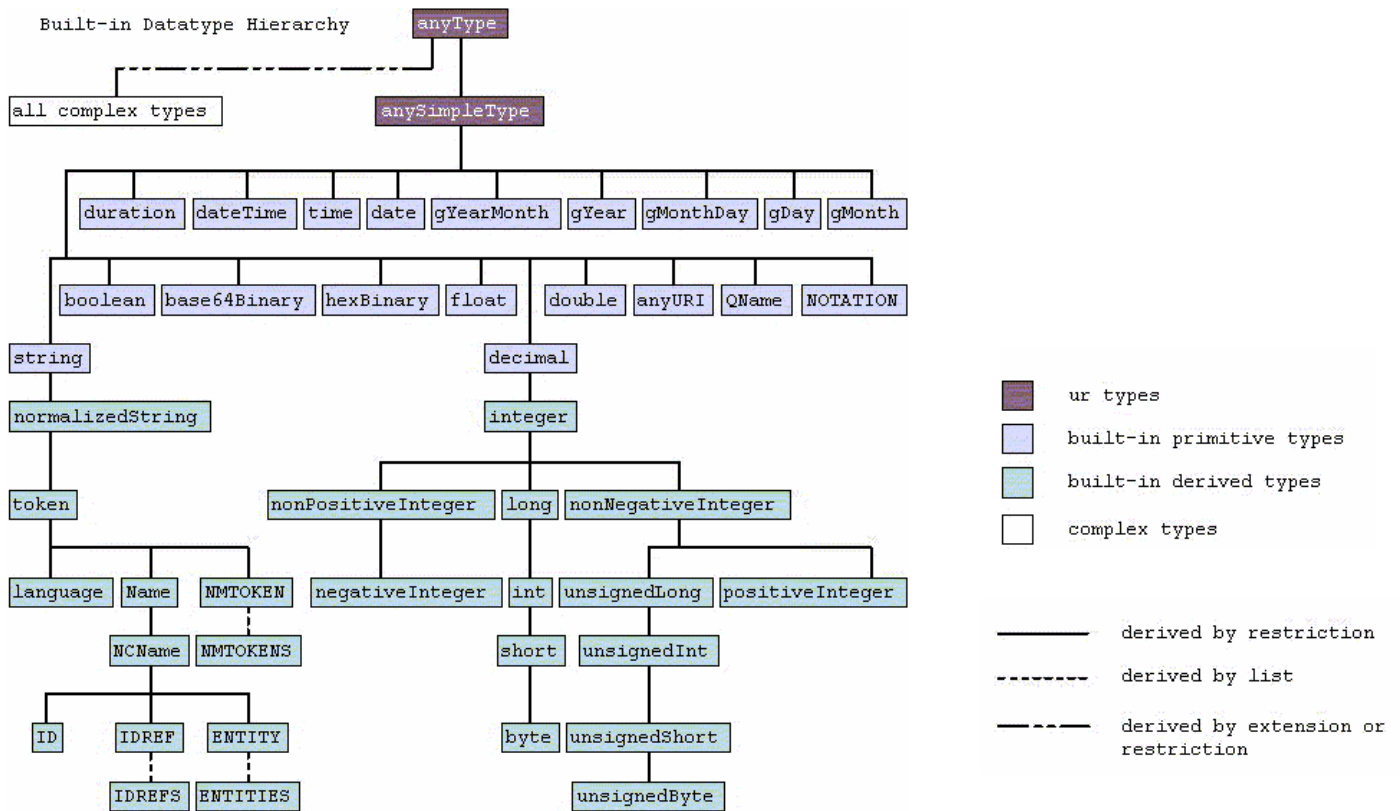
■ Eigenschaften

- verwendet XML-Syntax
- Unterstützung von Namenräumen (Namespaces)
- erweiterbares Typsystem (simple, complex types)
- erlaubt Definition einfacher Integritätsbedingungen (unique, key)
- komplexer und weniger kompakt gegenüber DTD

* <http://www.w3c.org/XML/Schema>



XML Schema: vordefinierte Typhierarchie



XML Schema: simpleType

- **simpleType**: benutzerdefinierter, einfacher Datentyp
 - hat keine Attribute oder Kindelemente
- 3 Möglichkeiten zur Bildung einfacher Typen
 - Einschränkung (restriction) von Eigenschaften (facets)
 - Listenbildung (list)
 - Vereinigung (union)
- Beispiel: simpleType-Definition mit length-Einschränkung
 - Präfix **xs:** bezeichne standardisierten XML-Schema-Namensraum

```
<xs:simpleType name="LoginName">
  <xs:restriction base="xs:string">
    <xs:length value="8" />
  </xs:restriction>
</xs:simpleType>
```

- **anonyme vs. benannte Typen**: ohne / mit name-Attribut
 - benannte Typen ermöglichen Mehrfach/Wiederverwendung



XML Schema: simpleType (2)

- Möglichkeiten der Einschränkung (restriction)
 - Längenbeschränkung für Zeichenketten /Listen: `length`, `minLength`, `maxLength`
 - Längenbeschränkung für Dezimalzahlen: `totalDigits`, `fractionDigits`
 - Wertebereichs-Unter/Obergrenzen: `minInclusive`, `maxInclusive`, `minExclusive`, `maxExclusive`
 - Aufzählung zulässiger Werte: `enumeration`
 - Muster: `pattern` value = <regulärer Ausdruck>
- Listenbildung (list): definiert leerzeichen-separierte Liste von simpleType-Werten
- Vereinigung (union): ermöglicht Typ mit mehrere Wertebereichen

```
<xs:simpleType name="BibId">  
  <xs:restriction base="xs:string">  
    <xs:pattern value="[A-Z][1-9][0-9]*" />  
  </xs:restriction>  
</xs:simpleType>
```

```
<xs:simpleType name="Kontakt">  
  <xs:union memberTypes="TelefonTyp EmailTyp"/>  
</xs:simpleType>
```

```
<xs:simpleType name="Lottozahlen">  
  <xs:list>  
    <xs:simpleType>  
      <xs:restriction base="xs:integer">  
        <xs:minInclusive value="1" />  
        <xs:maxInclusive value="49" />  
      </xs:restriction>  
    </xs:simpleType>  
  </xs:list>  
</xs:simpleType>
```



XML Schema: complexType

- komplexe Datentypen dienen der Beschreibung von Elementinhalten
 - Beschränkung (`restriction`) oder Erweiterung (`extension`) bestehender Typen
 - einfaches Inhaltsmodell (simple content): für Elemente ohne Subelemente
 - Default: komplexes Inhaltsmodell (complexContent), Basistyp anyType
- Elementstrukturierung bei `complexContent`
 - Reihung: `sequence`
 - Auswahl/Alternative: `choice`
 - Gruppe: `all` (jedes Element kann max. 1-mal auftreten; reihenfolgeunabhängig)
- Kardinalitätsrestriktionen: `minOccurs`, `maxOccurs`
 - Defaultwert: 1
 - unbeschränkte Anzahl: "unbounded"



XML Schema: complexType (2)

■ Definition durch Beschränkung (restriction)

```
<xs:complexType name="DozentTyp">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:sequence>
        <xs:element name="Name"
          type="xs:string"/>
        <xs:element name="Einrichtung"
          type="xs:string"
          minOccurs="0"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

≡

```
<xs:complexType name="DozentTyp">
  <xs:sequence>
    <xs:element name="Name"
      type="xs:string"/>
    <xs:element name="Einrichtung"
      type="xs:string"
      minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```



Vergleich Kardinalitätsrestriktionen

DTD	XML Schema	
	minOccurs	maxOccurs
1	1	1
?		
+		unbounded
*		unbounded



XML Schema: complexType (3)

■ Definition durch Erweiterung (extension):

```
<xs:complexType name="UniDozentTyp">
  <xs:complexContent>
    <xs:extension base="DozentTyp">
      <xs:sequence>
        <xs:element name="Adresse" type="AdressTyp"/>
        <xs:choice>
          <xs:element name="Mail" type="xs:string"/>
          <xs:element name="URL" type="xs:string"/>
        </xs:choice>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```



XML Schema: Schemaaufbau

■ Schema in *schema*-Element eingeschlossen

- Namensraum *http://www.w3.org/2001/XMLSchema* für Schemaelemente
- Spezifikation eines „target namespace“, dem das Schema zugeordnet wird

■ Schemakomponenten, die direkt unter *schema*-Element liegen, sind *global*; tiefer liegende Komponenten sind *lokal*

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="...">
  <xs:import ... />
  <xs:complexType name="...">...</xs:complexType>
  <xs:attribute name="..." ... />
  <xs:element name="..." >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="..." ... />
    </xs:sequence>
    <xs:attribute name="..." ... />
  </xs:complexType>
</xs:element>
</xs:schema>
```

benannt ———— **global**

anonym ———— **lokal**



XML Schema: Elementdeklaration

■ verschiedene Möglichkeiten der Elementdeklaration

- **direkte Deklaration** des Elementinhalts

```
<xs:element name=„Einrichtung“ type=„xs:string“ minOccurs=„0“ />
<xs:element name=„Dozent“>
  <xs:complexType> ... </xs:complexType>
</xs:element>
```

- Verweis auf **komplexen Datentyp**

```
<xs:element name=„Dozent“ type=„DozentTyp“ />
```

- Verweis auf **globale Elementdeklaration**

```
<xs:element ref=„Dozent“ />
```



Vergleich DTD - XML Schema Elementdeklaration

DTD:

```
<!ELEMENT Dozent (Titel?, Vorname+, Name, (Vorlesung | Seminar)*)>
```

XML Schema:

```
<xs:element name="Dozent">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Titel" type="xs:string" minOccurs="0" />
      <xs:element name="Vorname" type="xs:string" maxOccurs="unbounded" />
      <xs:element name="Name" type="xs:string" />
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Vorlesung" type="xs:string" />
        <xs:element name="Seminar" type="xs:string" />
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



XML Schema: Attributdeklaration

■ Attributdeklarationen

- global oder lokal am Ende einer ComplexType-Definition
- direkte Deklaration oder über *ref* bzw. *vordef.* Typen
- nur SimpleType-Datentypen erlaubt
- *Attribute für Attribute*: name, type, use (optional/required), default, fixed

```
<xs:element name="Vorlesung">
  <xs:complexType>
    <xs:sequence>...</xs:sequence>
    <xs:attribute name="Uhrzeit" type="xs:time" use="required"/>
  </xs:complexType>
</xs:element>
```

■ Beispiele

```
<xs:attribute name="Einheit" type="xs:string" fixed="km"/>
```

```
<xs:attribute name="Farbe" default="blau">
  <xs:simpleType>
    <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="rot" />
      <xs:enumeration value="gelb" />
      <xs:enumeration value="blau" />
    </xs:restriction>
  </simpleType>
</xs:attribute>
```

```
<xs:attribute name="id" type="xs:ID" use="required"/>
```

```
<xs:attribute name="Artikel" type="xs:IDREF" use="required"/>
```



XML Schema: Schlüsselbedingungen

■ Eindeutigkeitsbedingung: **unique**

- Selector-Ausdruck: Teilobjekte eines Dokuments, auf die Bedingung anzuwenden ist (XPath-Ausdruck)
- field-Ausdruck: Element- bzw. Attributwert, bzgl. dessen (Eindeutigkeit) getestet werden soll

```
<xs:element name="VLVerzeichnis">
  <xs:complexType>...</xs:complexType>
  <xs:unique name="UniqueBed">
    <xs:selector xpath=".,//Vorlesung" />
    <xs:field xpath="Thema/text()" />
  </xs:unique>
</xs:element>
```



XML Schema: Schlüsselbedingungen (2)

- Schlüsselbedingung: **key**
- Schlüsselreferenz: **keyref**

```
<xs:key name="ArtikelKey">
  <xs:selector xpath=„//Produkte/Artikel“ />
  <xs:field xpath="@aID" />
</xs:key>

<xs:keyref name="ArtikelKeyRef" refer="ArtikelKey">
  <xs:selector xpath="//Verzeichnis/Artikel" />
  <xs:field xpath="@artikelID" />
</xs:keyref>
```

```
<Produkte>
  <Artikel aID=„A1“>
    <Name> ... </Name>
  </Artikel>
...
<Verzeichnis>
  <Artikel artikelID=„A1“>
    ... </Artikel> ...
</Verzeichnis>
```

- Vergleich zu ID / IDREF
 - anwendbar für Attribute und Elemente
 - anwendbar für viele Typen
 - Einschränkung bzgl. Teildokumenten



XML Schema und Namensräume

- XML Schema baut auf und unterstützt Namensräume
- Namensraumzuordnung im Schema und im Dokument erforderlich
 - Attribut **targetNamespace** im Schema
 - Verweise im Schema auf definierte Elemente müssen qualifizierte Namen verwenden
 - kein Präfix erforderlich für Default-Namensraum

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://uni-leipzig.de/VLV"
  xmlns:v="http://uni-leipzig.de/VLV" >

  <element name="VLVerzeichnis">
    <complexType>
      <sequence>
        <element ref="v:Vorlesung"
          minOccurs="0" maxOccurs="unbounded" />
      </sequence>
    </complexType>
  </element>

  <element name="Vorlesung"> ... </element>
  <element name="Thema" type="string" /> ...
```

```
<?XML version="1.0"?>
<VLVerzeichnis xmlns="http://uni-leipzig.de/VLV">
  <Vorlesung Uhrzeit=„15:15“>
    <Thema>DBS2</Thema>
    <Dozent>
      <Name>Prof. Rahm</Name>
      <Einrichtung>Uni Leipzig</Einrichtung>
    </Dozent>
  </Vorlesung>
</VLVerzeichnis>
```



Nutzung mehrerer Schemas

■ import-Anweisung: Einbindung anderer Schemas / Namensräume

- Zugriff per Referenz auf globale Komponenten (Elemente, Attribute, Typen) des importierten Schemas
- Referenz muss qualifizierten Namen verwenden

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="urn:de.uni-leipzig.bibl"
  xmlns:bib="urn:de.uni-leipzig.bibl"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
<import namespace="http://purl.org/dc/elements/1.1/" />
...
<element name="book">
  <complexType>
    <sequence>
      <element ref = "dc:title" />
      <element ref = "dc:publisher" />
      <element ref = "dc:identifier" />
      <element name="location" type="string" />
      <element name="identifier" type="string" />
    </sequence>
  </complexType>
</element> ...
</schema>
```

```
<book xmlns="urn:de.uni-leipzig.bibl"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>Web und Datenbanken</dc:title>
  <dc:publisher>dpunkt-Verlag</dc:publisher>
  <dc:identifier>3-89864-189-9</dc:identifier>
  <location>Zweigstelle 1</location>
  <identifier>ST 6519</identifier>
</book>
```



6. Semistrukturierte Daten (XML, JSON)

■ Einleitung

■ Einführung in XML

- XML-Dokumente
- DTD (Document Type Definition)

■ XML Schema

- Typen (simpleType, complexType)
- Schemaaufbau
- Element/Attribut-Deklarationen
- Integritätsbedingungen
- Nutzung mehrerer Schemas / Namensräume

■ XML-Datenbanken, SQL/XML

- Datentyp XML und XML-Operatoren
- Datenkonversion relational <-> XML
- Auswertung von XML-Inhalten: XMLQUERY

■ JSON-Format, SQL JSON



XML-Datenbanken, SQL/XML

- mehrere Alternativen zur Speicherung von XML-Dokumenten in Datenbanken

1. ganzheitliche Speicherung von Dokumenten (BLOB, UDT XML)

- schnelles Einbringen der Daten und originalgetreue Dokumentrekonstruktion
- hoher Aufwand für Queries und Änderungen
- weniger geeignet für strukturierte Daten

2. Zerlegung (Dekomposition) der XML-Daten auf Tabellen

2a) generisch: Graphmodell (Document object model, DOM)

- allgemeine Lösung mit Tabellen für Elemente und Attribute
- aufwändige (SQL-) Query-Verarbeitung mit vielen Joins

2b) schemabasiert: aufgrund Tabellen-Mapping („Shredding“)

- schneller Zugriff auf ausgewählte Teile
- vereinfachte SQL-Queries
- günstig für strukturierte Daten

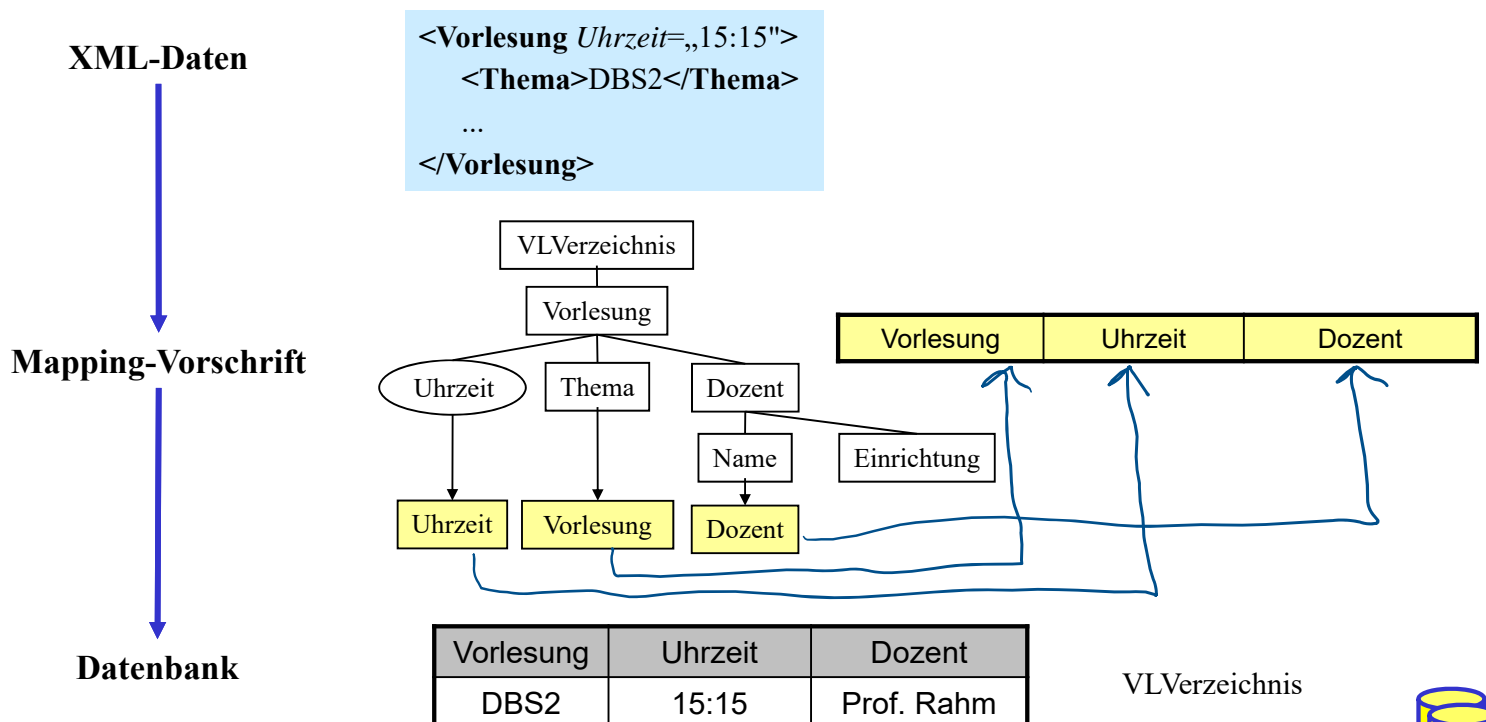
3. Kombinationen (SQL/XML unterstützt 1 und 2b)



Schemabasierte Dekomposition: Beispiel

■ manuelles Mapping

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



SQL-Datentyp: XML

- SQL-Standard Teil 14: SQL/XML
- XML-Dokument wird als Ganzes gespeichert
 - Beibehaltung der Originaldokumente/-daten (ähnlich LOB)
- DBMS erlaubt XPath/XQuery-Anfragen auf XML-Dokumenten
- Beispiel: Tabelle Vorlesungsverzeichnis

```
- CREATE TABLE Vorlesungsverzeichnis (  
    Universitaet VARCHAR (255),  
    Semester     VARCHAR (255),  
    Vorlesungen  XML  
)  
  
- INSERT INTO Vorlesungsverzeichnis  
(Universitaet, Semester, Vorlesungen)  
VALUES  
(`Uni Leipzig`, `SoSe 2020`,  
 XMLPARSE ( DOCUMENT`<VLVerzeichnis><Vorlesung  
    Uhrzeit="15:15"><Thema> DBS2 </Thema><Dozent>  
    <Name> Rahm </Name> <Einrichtung> IfI </Einrichtung> ...` )
```



SQL/XML-Operatoren

- Erweiterung der SQL-Anfragesprache zur XML-Verarbeitung
- Operatoren zur Erzeugung von XML-Daten
 - **XMLELEMENT**, **XMLATTRIBUTES** : Konstruktion von XML-Elementen bestehend aus Name, Attributen und Inhalt (Text oder andere XML-Elemente)
 - XMLAGG: Konstruktion gruppierter XML-Elemente
 - XMLFOREST, XMLCONCAT, XMLNAMESPACES, XMLDOCUMENT, ...
- Operatoren für Anfragen auf XML-Dokumenten
 - **XMLQUERY**: einfache XPath-Anfragen auf XML-Dokumente
 - **XMLEXISTS**: Selektionsbedingung für XML-Dokumente (in SQL-WHERE-Klausel)
- **XMLTABLE**: Konvertierung eines XML-Anfrageergebnisses in Relation



XMLELEMENT, XMLATTRIBUTES

- Erstellung von XML-Elementen bzw. -Attributen aus relationalen Tabellen
- Beispiel: Vorlesungen jeweils als XML-Fragment

```
SELECT Id, XMLELEMENT (Name "Vorlesung",
    XMLATTRIBUTES (Uhrzeit AS "Uhrzeit"),
    XMLELEMENT (Name "Thema", Thema),
    XMLELEMENT (Name "Dozent",
        XMLELEMENT (Name "Name", DName),
        XMLELEMENT (Name "Einrichtung", DEinr))) AS X
FROM Vorlesung
```

Vorlesung				
Id	Thema	Uhrzeit	DName	DEinr
1	DBS2	15:15	Rahm	IfI
2	NoSQL	9:15	Köpcke	IfI

Id	X (Typ:XML)
1	<Vorlesung Uhrzeit="15:15"><Thema>DBS2</Thema> <Dozent><Name>Rahm</Name><Einrichtung>IfI</Einrichtung> </Dozent></Vorlesung>
2	<Vorlesung Uhrzeit="9:15"><Thema>NoSQL</Thema> <Dozent><Name>Köpcke</Name><Einrichtung>IfI</Einrichtung> </Dozent></Vorlesung>



XMLQUERY

- Unterstützung einfacher XPath-Anfragen
 - Ergebnis ist Knotenmenge (serialisiert als konkatenierte XML-Elemente)
 - PASSING-Klausel gibt an, in welcher Tabellen-Spalte XML-Dokumente ausgewertet werden sollen
- Beispiel: Alle Themen von 15:15-Uhr-Vorlesungen

```
SELECT Universitaet, XMLQUERY(
    '//Vorlesung/Thema[../@Uhrzeit="15:15"]'
    PASSING Vorlesungen) AS X
FROM Vorlesungsverzeichnis
WHERE Semester = 'SoSe 2020'
```

Vorlesungsverzeichnis		
Univ.	Semester	Vorlesungen
Uni LE	SoSe 2020	<VLVerzeichnis ...
Uni LE	WiSe 19/20	<VLVerzeichnis ...
Uni DD	SoSe 2020	<VLVerzeichnis ...

Univ.	X (kein well-formed XML)
Uni LE	<Thema>DBS2</Thema><Thema>DWH </Thema>
Uni DD	<Thema>AlgoDat</Thema><Thema>eBusiness</Thema>



XMLTABLE

■ Konvertierung von XML-Anfrageergebnis in Relation

- Verwendung mehrerer XPath-Ausdrücke

■ Aufbau

- XPath-Ausdruck zur Selektion von Basiselementen
→ je ein Datensatz pro Basiselement in Ergebnisrelation
- pro Attribut der Relation ein XPath-Ausdruck
→ Auswertung des XPath-Ausdrucks relativ vom Basiselement

■ Einsatzbereiche

- Definition (relationaler) Sichten auf XML-Dokumente
- nutzerdefinierte Speicherung von XML-Daten in Tabellen
- Kombination von XML-Daten mit relationalen Daten (z.B. JOIN)



XMLTABLE (2)

■ Beispiel: relationale Darstellung des Vorlesungsverzeichnis

Vorlesungsverzeichnis		
Uni	Sem	Vorlesungen
..	..	<pre><VLVerzeichnis> <Vorlesung Uhrzeit="15:15"> <Thema>DBS2</Thema> <Dozent> <Name>Rahm</Name> <Einrichtung>IfI</Einrichtung> </Dozent> </Vorlesung> <Vorlesung Uhrzeit="9:15"> <Thema>NoSQL</Thema> <Dozent> <Name>Köpcke</Name> <Einrichtung>IfI</Einrichtung> </Dozent> </Vorlesung> </VLVerzeichnis></pre>

```
SELECT X.*
FROM Vorlesungsverzeichnis,
XMLTABLE (
  '//Vorlesung' PASSING Vorlesungen
  COLUMNS
    "THEMA" VARCHAR(255) PATH 'Thema',
    "ZEIT" VARCHAR(255) PATH '@Uhrzeit',
    "DNAME" VARCHAR(255) PATH 'Dozent/Name'
)AS X
```



Thema	Zeit	DName
DBS2	15:15	Rahm
NoSQL	9:15	Köpcke



6. Semistrukturierte Daten (XML, JSON)

- Einleitung
- Einführung in XML
 - XML-Dokumente
 - DTD (Document Type Definition)
- XML Schema
 - Typen (simpleType, complexType)
 - Schemaaufbau
 - Element/Attribut-Deklarationen
 - Integritätsbedingungen
 - Nutzung mehrerer Schemas / Namensräume
- XML-Datenbanken, SQL/XML
 - Datentyp XML und XML-Operatoren
 - Datenkonversion relational <-> XML
 - Auswertung von XML-Inhalten: XMLQUERY
- JSON-Format, SQL JSON



JSON-Datenrepräsentation

- **schemalose** Repräsentation von Dokumenten
- JSON (JavaScript Object Notation)
 - geschachtelte Objekt-Notation
 - Objekt { ... } umfasst Menge von Key-Value (Attribut/Wert-) Paaren
 - Datentypen: String, Zahl, Array [...], Boolean, Nullwert
- JSON einfacher als XML, leicht lesbar / schreibbar
 - besonders geeignet für „datenorientierte“ Objekte (statt Dokumenten mit viel Text)
 - starke Verbreitung in mobilen und Web Anwendungen
- Beispiel (JSON vs. XML)

```
{
  "Herausgeber": "Mastercard",
  "Nummer": "1234-5678-9012-3456",
  "Währung": "EURO",
  "Inhaber": {
    "Name": "Mustermann",
    "Vorname": "Max",
    "männlich": true,
    "Hobbys": [ "Reiten", "Golfen", "Lesen" ],
    "Kinder": [],
    "Partner": null
  }
}
```

```
<Kreditkarte Herausgeber="Mastercard"
  Nummer="1234-5678-9012-3456"
  Waehrung="EURO">
  <Inhaber Name="Mustermann" Vorname="Max"
    maennlich=true Partner="null">
    <Hobbys>
    <Hobby>Reiten</Hobby>
    <Hobby>Golfen</Hobby>
    <Hobby>Lesen</Hobby>
    </Hobbys>
    <Kinder />
  </Inhaber>
</Kreditkarte>
```



SQL/JSON

- wesentlicher Bestandteil von SQL:2016
 - Unterstützung u.a. in Oracle, DB2, MS SQL-Server, PostgreSQL
- kein vordefinierter Datentyp wie für XML, sondern Speicherung von JSON-Objekten als Strings/CLOBs
 - Test auf Gültigkeit (korrekte JSON-Syntax) als Constraint möglich

```
CREATE TABLE JTAB ( jcol CLOB CHECK (jcol IS JSON) )
```

- Erzeugen von JSON-Daten über Funktionen *json_object*, *json_array*, ...

```
UPDATE JTAB SET jcol = json_object ('id': 2345, 'name': 'Stefan') WHERE ...
```

- JSON-Abfragen über Funktionen *json_exists*, *json_value*, *json_query* unter Nutzung von Pfadausdrücken

```
... WHERE json_exists (jcol, '$.name')
```

- Funktion *json_table* zur Erzeugung von Tabellen aus JSON-Daten

```
SELECT JT.* FROM JTAB, json_table (jcol, '$[*]' COLUMNS (id NUMERIC PATH '$.id',  
name VARCHAR(255) PATH '$.name') AS JT ...
```

<https://modern-sql.com/blog/2017-06/whats-new-in-sql-2016>



Zusammenfassung

- XML:
 - flexibles Format für strukturierte und semistrukturierte Daten
 - dominierendes Austauschformat zwischen Web-Anwendungen
- DTD hat nur limitierte Schema-Eignung
 - mangelhafte Typisierung, keine Namensräume
- XML Schema
 - mächtigere Strukturbeschreibung als DTD
 - ermöglicht Typisierung von Elementinhalten und Attributwerten
 - zusätzliche Integritätsbedingungen
- Namensräume erlauben gemeinsame Nutzung verschiedener Schemas / globaler Typen in einem Dokument
- SQL-Standard unterstützt UDT XML
 - Konversion XML -> relational: XMLTABLE
 - Konversion relational -> XML: XMLELEMENT, XMLATTRIBUTE ...
- JSON: leichtgewichtiger XML-Alternative ohne Schema

