7. Semistrukturierte Daten (XML, JSON)

- Einleitung
- XML-Dokumente und Schemas
  - XML-Dokumente
  - DTD (Document Type Definition)
  - XML Schema
- XML-Anfragesprachen
  - Xpath
  - XQuery
- XML-Datenbanken, SQL/XML
  - Datentyp XML und XML-Operatoren
  - Datenkonversion relational <-> XML
  - Auswertung von XML-Inhalten: XMLQUERY
- JSON-Format, SQL JSON

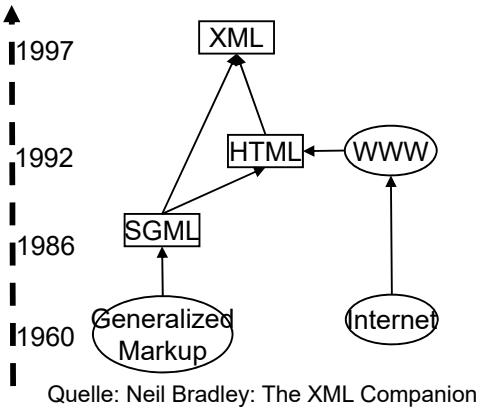
Strukturiertheit von Daten

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



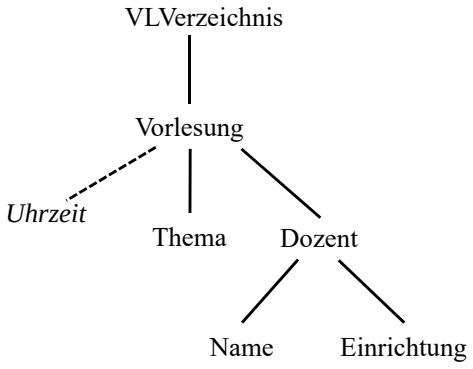
Quelle: Neil Bradley: The XML Companion

- starke XML-Verbreitung u.a. zum Datenaustauch
- leichtgewichtigere XML-Alternative: **JSON (JavaScript Object Notation)**
  - schemalose Repräsentation von Dokumenten / geschachtelten Datenobjekten
  - Unterstützung in SQL und durch Document Stores (NoSQL-Datenbanken), zB MongoDB

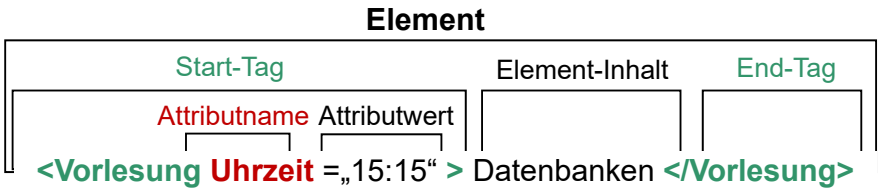
XML - Beispiel

- 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 (Document Type Definition) / 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)**
    Elemente A, B und Text treten in beliebiger Reihenfolge und Anzahl auf

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

- 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)

- **Auftretens**angabe 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

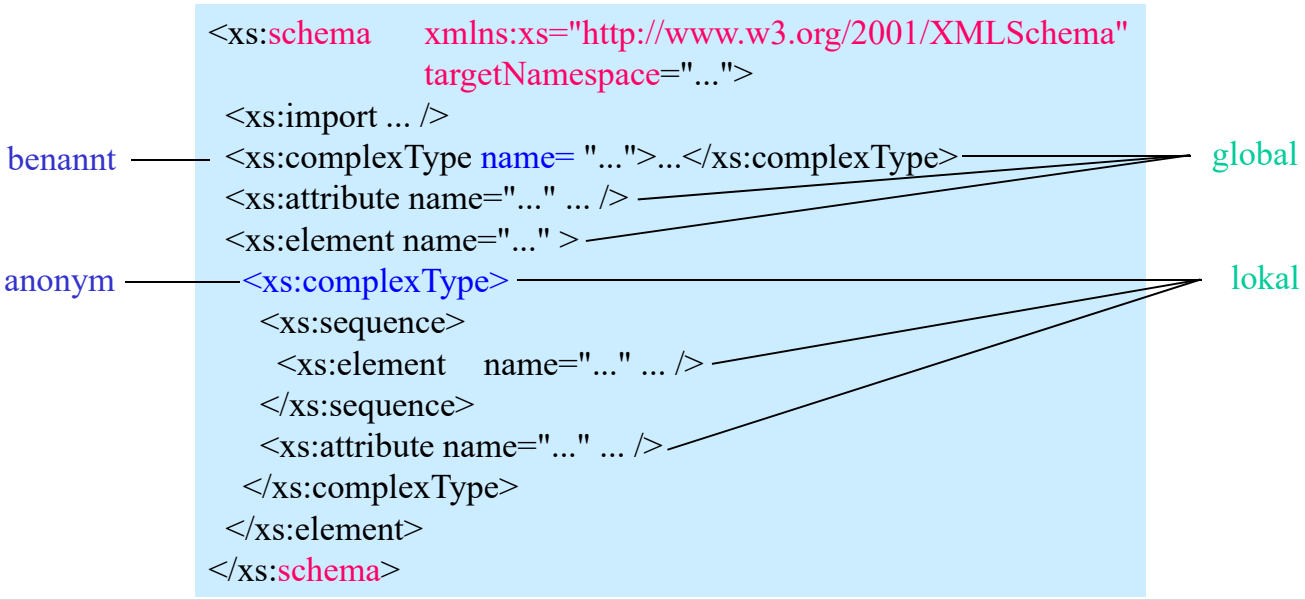
XML Schema*

- 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: 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>
```


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:complex Type> ... </xs:complex Type>
</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:

```
<1ELEMENT 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>
```


7. Semistrukturierte Daten (XML, JSON)

- Einleitung
- XML-Dokumente und Schemas
  - XML-Dokumente
  - DTD (Document Type Definition)
  - XML Schema
- XML-Anfragesprachen
  - Xpath
  - XQuery
- XML-Datenbanken, SQL/XML
  - Datentyp XML und XML-Operatoren
  - Datenkonversion relational <-> XML
  - Auswertung von XML-Inhalten: XMLQUERY
- JSON-Format, SQL JSON

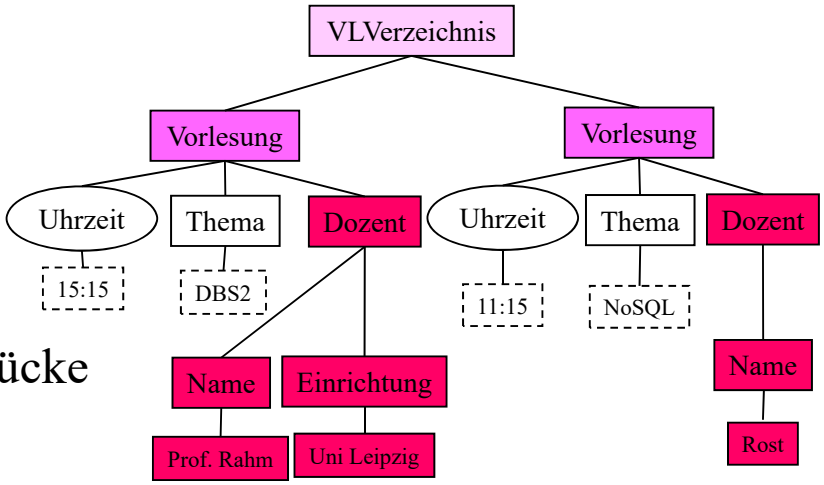
Einleitung

- deskriptive Anfragesprache für XML erforderlich
- Anfragesprachen wie SQL für XML unzureichend
  - Suche nach Inhalten auf beliebiger Hierarchieebene
  - Unterstützung für Pfadnavigation und Reihenfolgeabhängigkeiten
  - Unterstützung von Wildcards in Pfaden
  - Anfragemöglichkeit zu Metadaten und Daten
  - Anfragen auf schemalosen Daten
  - Unterstützung zur Neustrukturierung der Ergebnismenge
- Basisunterstützung durch XPath
  - ist Bestandteil von XQuery, XSLT …
  - Navigation und Selektion von Teildokumenten
  - 1999: V1.0, 2007: V2.0
- vollständige Anfragesprache: W3C XQuery
  - Recommendation (V1) seit Jan. 2007 , V3 seit 2014

XPath 1.0*

- Sprache zur Selektion von XML-Teildokumenten (Dokumentfragmente, Elemente, Attribute, Kommentare, Text, ...)
- Selektion durch schrittweise Navigation im Dokumentbaum

Beispiel:
/VLVerzeichnis/Vorlesung/Dozent

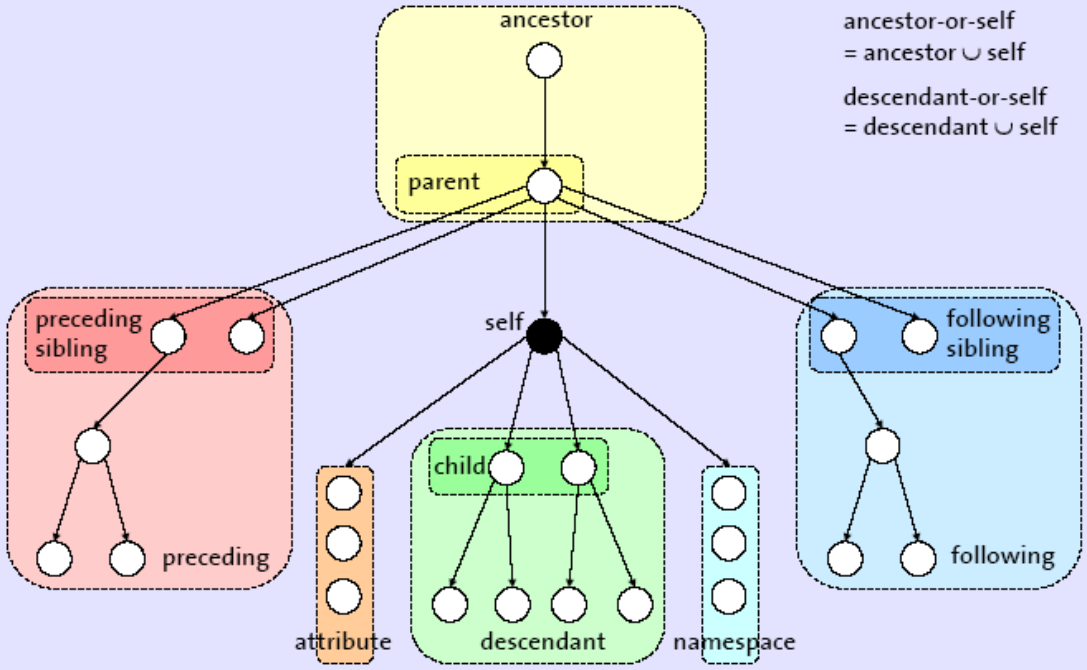


- absolute und relative Pfadausdrücke
  - absolut: /schritt1/schritt2
  - relativ: schritt1/schritt2
  - Abarbeitung von Pfadausdruck von links nach rechts
  - jeder Schritt liefert Knotenmenge oder Werte
- Syntax eines Schrittes: **Achse::Knotentest [Prädikat]**

* http://www.w3c.org/TR/xpath

XPath-Schritte: Achsen

- Achse: Richtung vom aktuellen Kontextknoten aus, in der die Knoten selektiert werden sollen
  - **13 Achsen**: ancestor, ancestor-or-self, child, descendant, descendant-or-self, following, following-sibling, parent, preceding, preceding-sibling, self, attribute, namespace)



XPath-Schritte: Knotentest

- **Knotentest**: selektiert Knoten aus der durch Achse vorgegebenen Menge (Selektion eines Knotens, wenn Test „wahr“ liefert)

Knotentest	selektierte Menge (aus Knotenmenge der Achse
Name	Elemente bzw. Attribute bzw. Namensraum mit diesem Namen
node()	alle Knoten
*	alle Elementknoten
text()	alle Textknoten
comment()	alle Kommentarknoten
processing-instruction()	alle Processing instructions

XPath-Schritte: Prädikate

- **Prädikat**: Filterausdruck, der nach dem Knotentest diejenigen Knoten selektiert, für die das Prädikat „wahr“ liefert
  - kann Vergleichsoperatoren (’=’, ’<’, ’<=’, ’!=’, ...)
  - logische Operatoren (and, or) und
  - Vereinigung von Knotenmengen (’|’) enthalten

  /child::VLVerzeichnis/child::Vorlesung[attribute::Uhrzeit=„15:15“]/child::Thema

  alternativ: */VLVerzeichnis/Vorlesung [@Uhrzeit=„15:15“]/Thema*
- Prädikat kann XPath-Ausdrücke enthalten: testet Existenz bestimmter Elemente/Attribute/Attributwerte

  /child::VLVerzeichnis/child::Vorlesung/child::Dozent[child::Einrichtung]/child::Name

  alternativ: */VLVerzeichnis/Vorlesung/Dozent [Einrichtung]/Name*
- bei Angabe von Zahlen werden Knoten der entsprechenden Kontextpositionen selektiert

  /child::*Produktliste/child::Produkt [position()=1]*

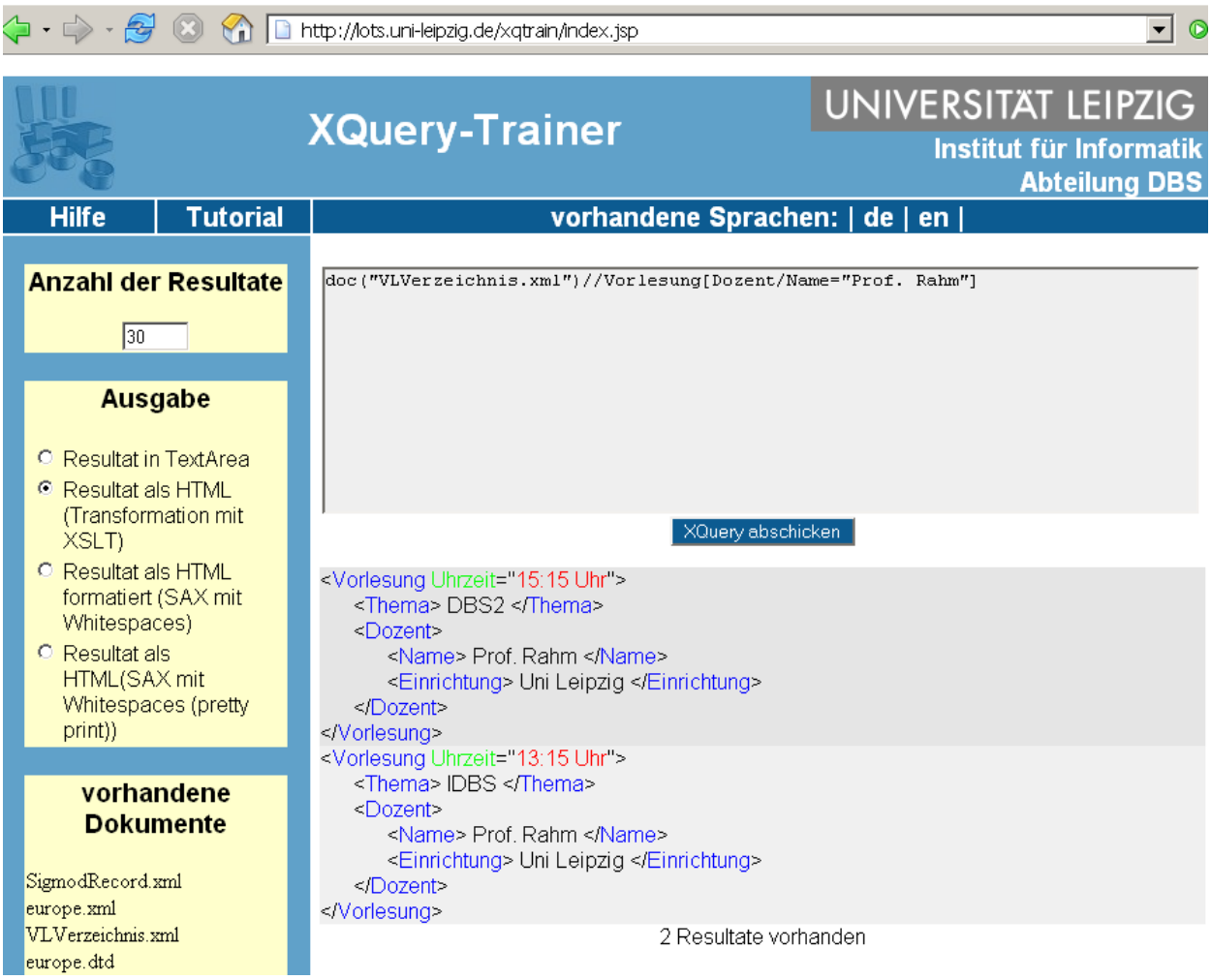
  alternativ: */Produktliste/Produkt [1]*

XPath: Abgekürzte Syntax/Funktionen

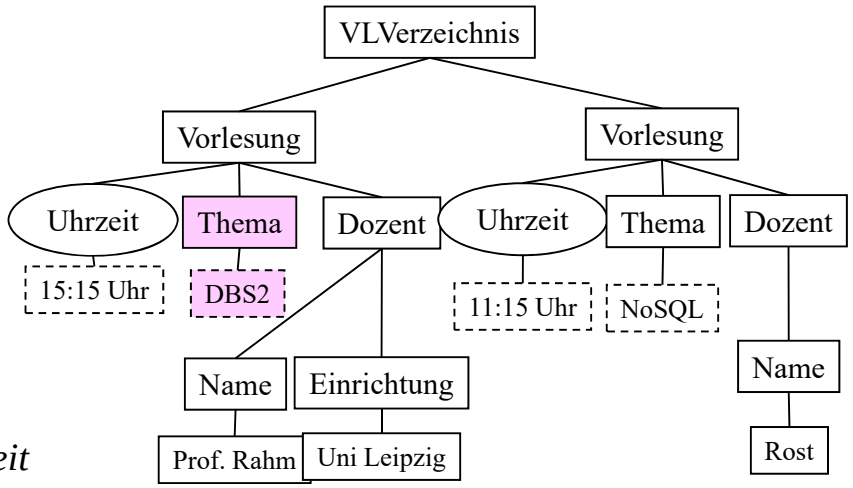
- XPath-Ausdrücke können durch abgekürzte Syntax vereinfacht werden

Langform	Abkürzung	Bemerkung
child::Knotentest	Knotentest	ohne Achsenangabe wird child-Achse verwendet
self::node()	.	aktueller Knoten
parent::node()	..	Elternknoten
descendant-or-self::node()	//	alle Nachkommen des aktuellen Knotens
attribute::Name	@Name	
[position()=X]	[X]	Prädikat zur Selektion von Knoten an Knotenposition X

- folgende Funktionen sind in XPath verfügbar (Auswahl):
  - für Knotenmenge: *name*(Knotenmenge), *last*(), *position*(), *count*(Knotenmenge), *id*(Name)
  - für Strings: *string*(Objekt), *concat*(String1, String2, ...), *starts-with*(String, Pattern), *contains*(String, Pattern), *substring*(String, Start[, Länge]), *string-length*()
  - für Zahlen: *number*(Objekt), *sum*(Knotenmenge), *round*(Zahl)
  - Boolesche Funktionen: *not* (Boolean), *true*(), *false*()



XPath: Beispiele



- Finde Uhrzeit der Vorlesung "DBS2"

//Thema[text() ="DBS2"]/../@Uhrzeit

//Vorlesung [Thema=„DBS2“]/@Uhrzeit

- Vorlesungen, deren Dozent "Prof. Rahm" oder „Rost" ist ?

//Vorlesung [Dozent/Name=„Prof. Rahm“ OR Dozent/Name=„Rost“]

XPath: Beispiele (2)

- Themen aller Vorlesungen, die „15:15" beginnen und deren Dozent zur "Uni Leipzig" gehört ?

  //Vorlesung [@Uhrzeit=„15:15“ AND Dozent/Einrichtung=„Uni Leipzig“]/Thema

- Wie viele Vorlesungen sind im Verzeichnis enthalten?

  count (//Vorlesung)

- Welche Einrichtungen enthalten den Text "Leipzig" ?

  //Einrichtung [contains(text(), „Leipzig“)]

XQuery*

- XQuery: W3C-Standardisierung für einheitliche XML-Anfragesprache
- abgeleitet von vorangegangenen proprietären XML-Anfragesprachen (Quilt, XPath, XQL, XML-QL, ...) sowie SQL und OQL

- FLWOR („flower“) –Syntax:

  For ... Let ... Where ... Order By ... Return)

- weitere Eigenschaften
  - funktionale Anfragesprache (Ausdrücke sind wieder als Parameter verwendbar)
  - komplexe Pfadausdrücke (basierend auf XPath 2.0)
  - Funktionen
  - konditionale und quantifizierte Ausdrücke
  - Ausdrücke zum Testen/Modifizieren von Datentypen
  - Elementkonstruktoren
  - Dokument-Änderungen mit „XQuery Update Facility“ (W3C Recommendation seit 2011)

* www.w3c.org/XML/Query

XQuery-Ausdruck: Beispiele

- Literale, z. B. 1, 0.5, "a string"
- arithmetische Ausdrücke: 1+1
- Funktionen, z. B. true(), concat("1","3"), count (...), avg (...)
- Konstruktoren, z. B.: <a x="1"/>,
  element a { attribute x { 1 } }
- XPath 2.0-Ausdruck: doc("VLV.xml")//Dozent/Name
- FLWOR-Ausdruck:

```
for     $i in doc("VLV.xml")//Vorlesung
where   $i/@Uhrzeit=„15:15"
order by $i/Thema
return  $i/Thema
```

Teilschritte

- Dokumentzugriff mit *doc*(URI) oder *collection*(URI)
- Auswahl von Dokumentfragmenten mit XPath 2.0
- Variablenbindungen
- Operationen (Selektion, Verbund, ...) auf den gebundenen Daten
- Erzeugung neuer Knoten

FLWOR-Ausdrücke

```
FLWOR-expr    ::= (FOR-expr | LET-expr)+
                  WHERE-expr?
                  ORDERBY-expr?
                  return Expr
FOR-expr      ::= for $var in Expr (, $var in Expr)*
LET-expr      ::= let $var := Expr (, $var := Expr)*
WHERE-expr    ::= where Expr
ORDERBY-expr  ::= (order by | stable order by) OrderSpec (, OrderSpec)*
OrderSpec     ::= Expr OrderModifier
OrderModifier ::= (ascending | descending)? (empty greatest | empty least)? (collation StringLiteral)?
```

- For/Let: Variablenbindung an Datenquellen / Sequenzen
- Where: Auswahlbedingung
- Order by: Sortieranweisung (ansonsten Sortierung in Dokumentreihenfolge)
- Return: Festlegung, wie Ergebnis aussehen soll

FLWOR – For/Let (1)

- For/Let: Ergebnis eines XQuery-Ausdrucks wird an Variable gebunden
- For-Ausdruck
  - für jedes Wurzelelement der Ergebnissequenz erfolgt Bindung an Variable (Iteration über die Sequenz)
  - Beispiel:
    - $d wird jeweils an Elemente der Sequenz von Dozentennamen gebunden (für jeden Namen genau einmal)
    - RETURN wird für jede Bindung einmal ausgeführt

```
for $d in doc("VLV.xml")//Dozent/Name
return <Ergebnis> { $d } </Ergebnis>
```

```
<Ergebnis>
   <Name>Prof. Rahm</Name>
</Ergebnis>
<Ergebnis>
   <Name>Dr. Köpcke</Name>
</Ergebnis>
```

- Let-Ausdruck
  - Ergebnis des XQuery-Ausdrucks wird geschlossen an Variable gebunden
  - Beispiel: RETURN wird einmal ausgeführt

```
let $d := doc("VLV.xml")//Dozent/Name
return <Ergebnis> { $d } </Ergebnis>
```

```
<Ergebnis>
   <Name>Prof. Rahm</Name>
   <Name>Dr. Köpcke</Name>
</Ergebnis>
```


FLWOR – For/Let (2)

- Kombination von let/for-Ausdrücken

```
let $j := ("A", "B", "C")
for $i in $j
return
   <i>{ $i }</i>
```

→

```
<i>A</i>
<i>B</i>
<i>C</i>
```

```
for $i in (1, 2, 3)
let $j := ("A", "B", "C")
return
   <tuple>
        <i>{ $i }</i> <j>{ $j }</j>
   </tuple>
```

→

```
<tuple> <i>1</i> <j>A B C</j> </tuple>
<tuple> <i>2</i> <j>A B C</j> </tuple>
<tuple> <i>3</i> <j>A B C</j> </tuple>
```

```
for    $i in (1, 2, 3),
       $j in ("A", "B", "C")
return
   <tuple>
        <i>{ $i }</i> <j>{ $j }</j>
   </tuple>
```

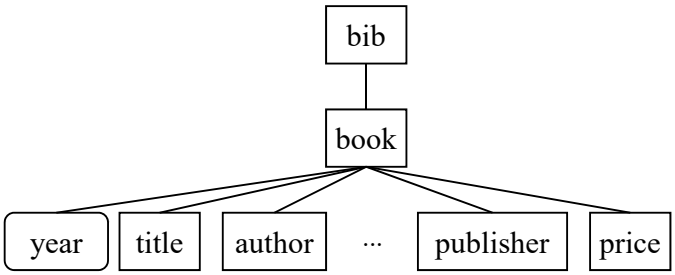
→

```
<tuple> <i>1</i> <j>A</j> </tuple>
<tuple> <i>1</i> <j>B</j> </tuple>
<tuple> <i>1</i> <j>C</j> </tuple>
<tuple> <i>2</i> <j>A</j> </tuple>
<tuple> <i>2</i> <j>B</j> </tuple>
<tuple> <i>2</i> <j>C</j> </tuple>
<tuple> <i>3</i> <j>A</j> </tuple>
<tuple> <i>3</i> <j>B</j> </tuple>
<tuple> <i>3</i> <j>C</j> </tuple>
```

XQuery: Beispielschema

- folgendes Schema (DTD) liegt den folgenden XQuery-Beispielen zugrunde
  (aus XQuery use cases)
- zugehöriges Dokument sei durch *bib.xml* referenzierbar

```
<!ELEMENT bib  (book* )>
<!ELEMENT book  (title, author+, publisher, price )>
<!ATTLIST book  year CDATA  #REQUIRED >
<!ELEMENT author  (#PCDATA)>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT publisher  (#PCDATA )>
<!ELEMENT price  (#PCDATA )>
```



XQuery: Reihenfolge/Sortierung

- Liste alle Buchtitel, in denen "Erhard Rahm" Erstautor ist

```
doc("bib.xml")//title[../author[1]="Erhard Rahm"]
```

```
for $b in doc("bib.xml")//book
where $b/author[1]="Erhard Rahm"
return $b/title
```

- Liste die ersten 10 Bücher

```
doc("bib.xml")//book[position() = (1 to 10)]
```

- Liste alle Titel von Büchern, die bei Springer nach 2000 publiziert wurden, in alphabetischer Reihenfolge

```
for $b in doc("bib.xml")/bib/book
where   $b/publisher=„Springer“ AND $b@year>2000
order by $b/title
return $b/title
```

XQuery: Verbundoperationen

- Liste alle Dozenten, die auch Buchautoren sind (innerer Verbund)
  (Annahme: Dozent hat Vorname/Name)

```
for $d in distinct-values(doc("VLVerzeichnis.xml")//Dozent/Name)
for $a in doc("bib.xml")//author[text() = $d]
return
  <DozentUndAutor>
    { $a/text() }
  </DozentUndAutor>
```

- Liste alle Dozenten und markiere Autoren mit einem Attribut 'author' (äußerer Verbund)

```
for $d in distinct-values(doc("VLVerzeichnis.xml")//Dozent/Name)
return
  <Dozent>
    {   for $a in doc("bib.xml")//author [text() = $d]
        return attribute author { "yes" },
      $d
    }
  </Dozent>
```


```
for $d in distinct-values(doc("VLVerzeichnis.xml")//Dozent/Name)
return
 <Dozent>
   {
    for $a in doc("bib.xml")//author[surname = substring-after(string($d)," ")]
    return
      attribute author { "yes" },
      $d
   }
 </Dozent>
```

XQuery abschicken

```
<Dozent author="yes"> Prof. Rahm </Dozent>
<Dozent> Dr. Sosna </Dozent>
<Dozent> Prof. Heyer </Dozent>
<Dozent> Prof. Gräbe </Dozent>
<Dozent> Prof. Fähnrich </Dozent>
```

5 Resultate vorhanden

7. Semistrukturierte Daten (XML, JSON)

- Einleitung
- XML-Dokumente und Schemas
  - XML-Dokumente
  - DTD (Document Type Definition)
  - XML Schema
- XML-Anfragesprachen
  - Xpath
  - XQuery
- 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

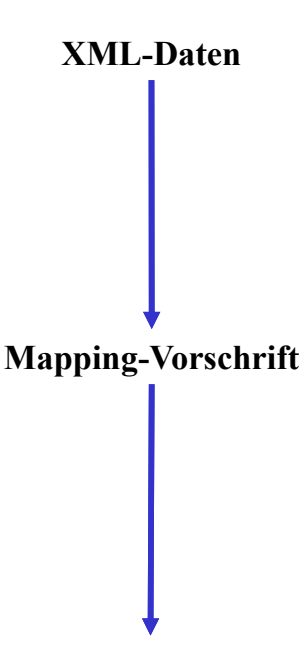
1. ganzheitliche Speicherung von Dokumenten (BLOB, UDT XML)
   - schnelles Einbringen der Daten und originalgetreue Dokumentrekonstruktion
   - hoher Aufwand für Queries und Änderungen
   - weniger geeignet für strukturierte Daten
2. Zerlegung (Dekomposition) der XML-Daten auf Tabellen

   2a) generisch: Graphmodell (Document object model, DOM)
   - allgemeine Lösung mit Tabellen für Elemente und Attribute
   - aufwändige (SQL-) Query-Verarbeitung mit vielen Joins

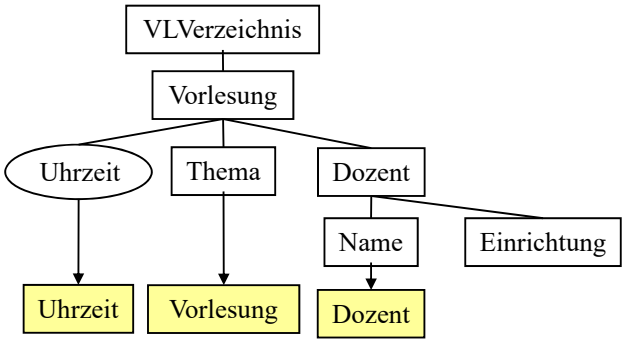
   2b) schemabasiert: aufgrund Tabellen-Mapping („Shredding“)
   - schneller Zugriff auf ausgewählte Teile
   - vereinfachte SQL-Queries
   - günstig für strukturierte Daten
3. Kombinationen (SQL/XML unterstützt 1 und 2b)

Schemabasierte Dekomposition: Beispiel

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




```
<Vorlesung Uhrzeit=„15:15">
   <Thema>DBS2</Thema>
   ...
</Vorlesung>
```



Vorlesung	Uhrzeit	Dozent
DBS2	15:15	Prof. Rahm

VLVerzeichnis

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 2023',
     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-Dokumenten
  - XMLEXISTS: Selektionsbedingung für XML-Dokumente (in SQL-WHERE-Klausel)
- XMLTABLE: Konvertierung eines XML-Anfrageergebnisses in Relation

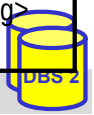
XMLELEMENT, XMLATTRIBUTES

- 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	11:15	Rost	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="11:15"><Thema>NoSQL</Thema><Dozent><Name>Rost</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 Univ, XMLQUERY(
  '//Vorlesung/Thema[../@Uhrzeit="15:15"]'
  PASSING Vorlesungen) AS X
FROM Vorlesungsverzeichnis
WHERE Semester = 'SoSe 2023'
```

Vorlesungsverzeichnis		
Univ	Semester	Vorlesungen
Uni LE	SoSe 2023	<VLVerzeichnis ...
Uni LE	WiSe 22/23	<VLVerzeichnis ...
Uni DD	SoSe 2023	<VLVerzeichnis ...

Univ	X (*kein well-formed XML*)
Uni LE	<Thema>DBS2</Thema><Thema>DWH </Thema>
Uni DD	<Thema>AlgoDat</Thema><Thema>Logik</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
..	..	<VLVerzeichnis> <Vorlesung Uhrzeit="15:15"> <Thema>DBS2</Thema> <Dozent> <Name>Rahm</Name> <Einrichtung>IfI</Einrichtung> </Dozent> </Vorlesung> <Vorlesung Uhrzeit="11:15"> <Thema>NoSQL</Thema> <Dozent> <Name>Rost</Name> <Einrichtung>IfI</Einrichtung> </Dozent> </Vorlesung> </VLVerzeichnis>

```
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	11:15	Rost

7. Semistrukturierte Daten (XML, JSON)

- Einleitung
- XML-Dokumente und Schemas
  - XML-Dokumente
  - DTD (Document Type Definition)
  - XML Schema
- XML-Anfragesprachen
  - Xpath
  - XQuery
- 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, PostgresSQL
- 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
- einfache Strukturfestlegung durch DTD
- XML Schema
  - umfassendes Typsystem und Unterstützung von Namensräumen
  - ermöglicht gemeinsame Nutzung verschiedener Schemas / globaler Typen in einem Dokument
- standardisierte XML-Anfragesprachen: XPath, XQuery
  - XPath: einfache Auswahl über Pfadausdrücke
  - Xquery: vollständige Anfragesprache, u.a. mit FLWOR-Ausdrücken
- SQL-Standard unterstützt UDT XML
  - Konversion XML -> relational: XMLTABLE
  - Konversion relational -> XML: XMLELEMENT, XMLATTRIBUTE …
- JSON: leichtgewichtigere XML-Alternative ohne Schema