

Modul 1

Semistrukturierte Datenmodellierung

Julian Haslinger



Der vorliegende Foliensatz basiert vorwiegend auf:

Elliottte Rusty Harold, W. Scott Means: XML in a Nutshell: A Desktop Quick Reference, 3rd Edition, O'Reilly, 2005

Inhalt

- Strukturierte Daten
- Semistrukturierte Daten
- Datenmodelle für semistrukturierte Daten
- Einführung in XML

Strukturierte Daten

- Daten mit gleichartigem Aufbau werden zu einer Informationseinheit zusammengefasst:

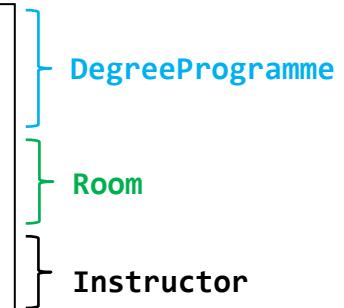
- Entität
- Objekt

[0307, Software Engineering, SE]
[0458, Medizin- und Bioinformatik, MBI]
[0456, Kommunikation Wissen Medien, KWM]
[3.009, FH3, LBS2]
[2.027, FH2, runtastic HS5]
[p20621, Josef Altmann]
[p22080, Julian Haslinger]

- Entitäten/Objekte mit gleichartigem Aufbau bzw. regelmäßiger Struktur werden gruppiert zu

- Entitätstypen
- Klassen

[0307, Software Engineering, SE]
[0458, Medizin- und Bioinformatik, MBI]
[0456, Kommunikation Wissen Medien, KWM]
[3.009, FH3, LBS2]
[2.027, FH2, runtastic HS5]
[p20621, Josef Altmann]
[p22080, Julian Haslinger]



Strukturierte Daten

- Entitäten/Objekte des gleichen Entitätstyps bzw. der gleichen Klasse haben die gleichen Eigenschaften (typisierte Attribute)

DegreeProgramme

[0307, Software Engineering, SE]
[0458, Medizin- und Bioinformatik, MBI]
[0456, Kommunikation Wissen Medien, KWM]

{[code:integer, name:string, abbreviation:string]}

Instructor

[p20621, Josef Altmann]
[p22080, Julian Haslinger]

{[instructorNumber:integer, name:string]}

Room

[3.009, FH3, LBS2]
[2.027, FH2, runtastic HS5]

{[roomNr:integer, building:string, description:string]}

... strikte Struktur durch Schema erzwungen!

Strukturierte Daten

Datenmodell

- Relationenmodell zur Beschreibung von strukturierten Daten
 - Entität ⇒ Tupel
 - Entitätstyp ⇒ Relation

DegreeProgramme	code	name	abbreviation
	0307	Software Engineering	SE
	0458	Medizin- u. Bioinformatik	MBI
	0456	Kommunikation Wissen Medien	KWM

} Schema
} Daten

- Datenbankschema
 - Trennung von Schema (Strukturinformation) und Daten (Instanz)
 - Vollständige Strukturbeschreibung vor der Datenspeicherung
- Daten
 - sind immer Instanzen des Schemas
 - Struktur und Typisierung festgelegt, keine Abweichungen möglich
 - tragen selbst keine Strukturinformationen
 - müssen mit Hilfe des Schemas interpretiert, manipuliert werden

Semistrukturierte Daten

Beispiel Instructor

p20621, Josef Altmann, 22610, 22699, josef.altmann@fh-hagenberg.at

p22080, Julian, Haslinger, julian.haslinger@fh-hagenberg.at, 06641234567

p23001, Norbert Niklas, norbert.niklas@fh-hagenberg.at

p20774, Barbara, Traxler, barbara.traxler@fh-hagenberg.at, 22033

- Struktur teilweise vorhanden
 - jede Zeile repräsentiert eine Entität
 - Entitäten können gruppiert werden (Instruktoren)
- Struktur ist aber nicht strikt
 - Entitäten haben keine regelmäßige Struktur und keine strenge Typisierung
 - Struktur weiterer Entitäten nicht fixiert/vorhersagbar
 - Schema ist optional

Semistrukturierte Daten

Anforderungen

- Daten mit wechselnder und nicht streng typisierter Struktur können nicht durch ein herkömmliches (Datenbank-)Schema beschrieben werden.
- Fehlt das Schema, so muss die Bedeutung der Struktur in den Daten(sätzen) selbst wiedergegeben werden.
- Datenaustausch erfordert flexible Austauschformate, denen kein einheitliches Schema zugrunde liegt.

... WEB!

... Dokumentenorientierte Sicht (WEB) versus
strikte Datenstrukturen (Datenbanken)

Semistrukturierte Daten

Datenmodell

Unstrukturiert
(Text)

Semistrukturiert
(formatierter Text)

(stark) Strukturiert
(Datenbanken)

- Wie können semistrukturierte Daten beschrieben/modelliert werden?

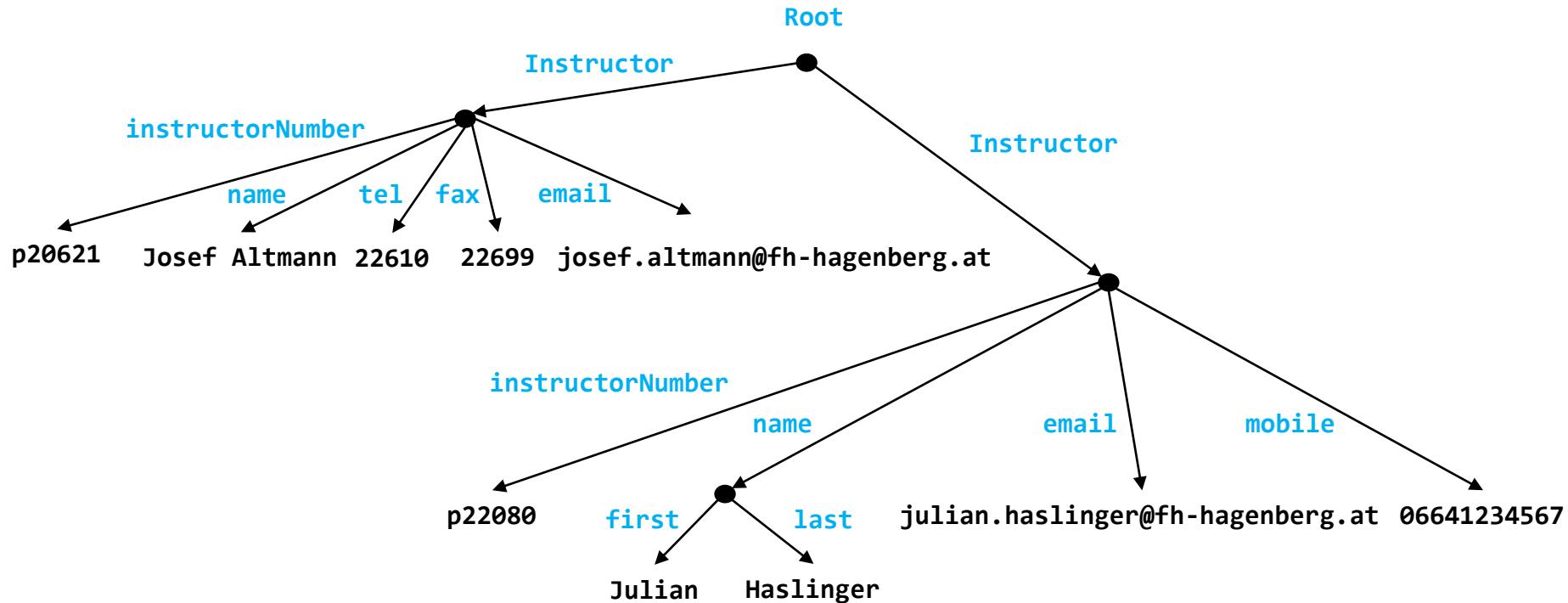
... Ideen dazu?

Semistrukturierte Daten

Datenmodell: Gerichteter Graph

p20621, Josef Altmann, 22610, 22699, josef.altmann@fh-hagenberg.at

p22080, Julian, Haslinger, julian.haslinger@fh-hagenberg.at, 06641234567



- Blattknoten: Daten (Zeichenketten)
- Innere Knoten, Kanten: Struktur/Schemainformation
- Kantenbeschriftung: Attributname
- *selbstbeschreibendes Datenmodell („Schemagraph“)*

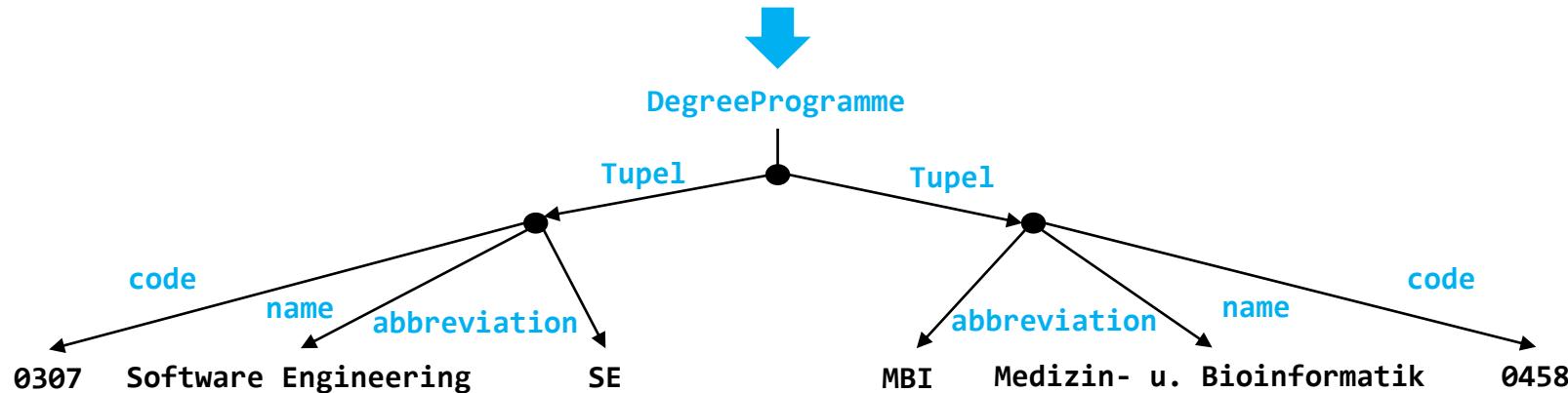
Semistrukturierte Daten

Abbildung relationaler Daten

- Strukturierte Daten sind nur ein Spezialfall von semistrukturierten Daten
- Relationale Daten können als Graph dargestellt werden

DegreeProgramme

	code	name	abbreviation
0307	Software Engineering	SE	
0458	Medizin- u. Bioinformatik	MBI	
0456	Kommunikation Wissen Medien	KWM	



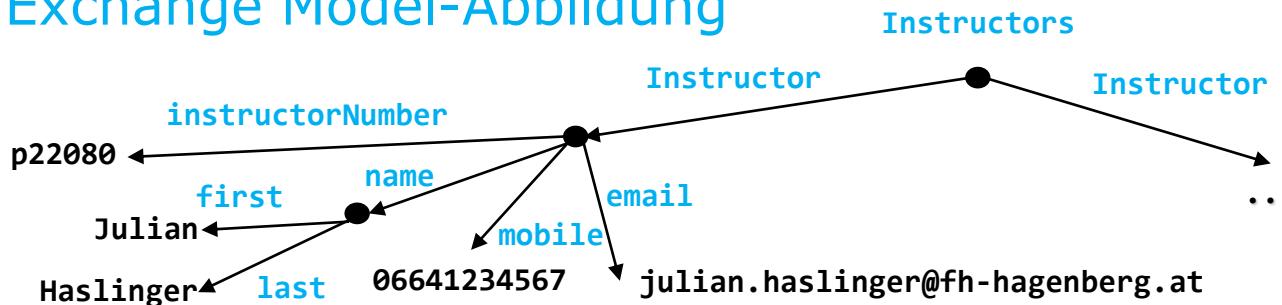
Semistrukturierte Datenmodellierung

■ Semistrukturierte Datenmodelle

- **Object Exchange Model**
 - OEM: Stanford University Database Group
- **Extensible Markup Language**
 - XML: w3.org/TR/xml/
- **JavaScript Object Notation**
 - JSON: json.org/
- **YAML Ain't Markup Language**
 - YAML: yaml.org/
- ...

Semistrukturierte Datenmodellierung

Object Exchange Model-Abbildung



```
{Instructors:  
  {Instructor:  
    {  
      instructorNumber: "p22080"  
      name:  
        {  
          first: "Julian"  
          last: "Haslinger"  
        }  
      email: "julian.haslinger@fh-hagenberg.at"  
      mobile: 06641234567  
    }  
    {Instructor:  
      {  
        instructorNumber: "p20621"  
        name: "Josef Altmann"  
        tel: 22610  
        fax: 22699  
        email: "josef.altmann@fh-hagenberg.at"  
      }  
    }  
  }  
}
```

OEM

Semistrukturierte Datenmodellierung

JSON/YAML-Abbildung

```
{
  "Instructors": [
    {
      "Instructor": {
        "instructorNumber": "p22080",
        "name": {
          "first": "Julian",
          "last": "Haslinger"
        },
        "email": "julian.haslinger@fh-hagenberg.at",
        "mobile": "06641234567"
      }
    },
    {
      "Instructor": {
        "instructorNumber": "p20621",
        "name": "Josef Altmann",
        "tel": 22610,
        "fax": 22699,
        "email": "josef.altmann@fh-hagenberg.at"
      }
    }
  ]
}
```

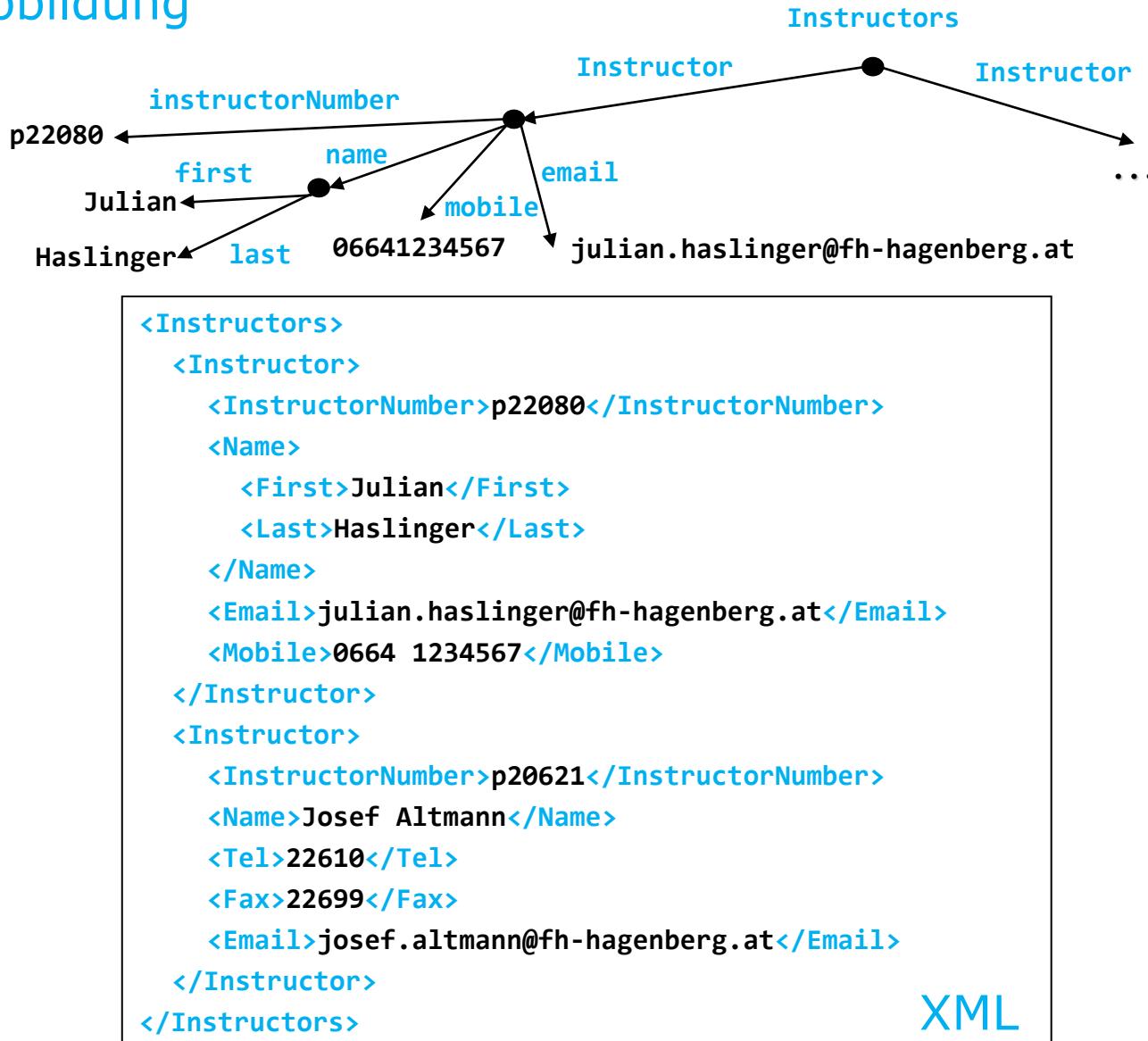
JSON

```
--- #YAML-Mapping
Instructors:
  - Instructor:
    instructorNumber: p22080
    name:
      first: Julian
      last: Haslinger
    email: julian.haslinger@fh-hagenberg.at
    mobile: 06641234567
  - Instructor:
    instructorNumber: p20621
    name: Josef Altmann
    tel: 22610
    fax: 22699
    email: josef.altmann@fh-hagenberg.at
```

YAML

Semistrukturierte Datenmodellierung

XML-Abbildung



Semistrukturierte Datenmodellierung

- Unterschiedliche Datenmodelle für semistrukturierte Daten
 - Object Exchange Model (OEM)
 - Extensible Markup Language (XML)
 - JavaScript Object Notation (JSON)
 - YAML Ain't Markup Language (YAML)
 - ...
- Unterschiede liegen in
 - der Syntax
 - den Beschreibungs- und Ausdrucksmöglichkeiten
 - der Abfrage- und Manipulationssprache
 - den Anwendungsbereichen

... but the goal is the same – modelling of semi-structured data!

Inhalt

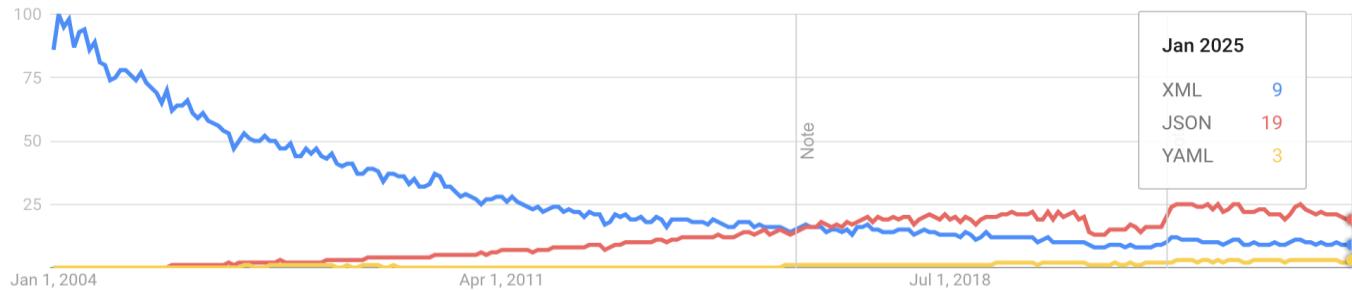
- Strukturierte Daten
- Semistrukturierte Daten
- Datenmodelle für semistrukturierte Daten
- **Einführung in XML**

Das Phänomen XML

- "XML is the ASCII of the 21st century."
- "XML is the ASCII of the Web" (Tim Bray, Co-Editor von XML 1.0)
- "If I invent another programming language, its name will contain the letter X." (N. Wirth, Software Pioniere Konferenz, Bonn 2001)

Google-Suche 2025:

JavaScript	5.900 Mio.
Database	2.680 Mio.
ChatGPT	1.320 Mio.
XML	553 Mio.
SQL	370 Mio.
JSON	312 Mio.
C++	237 Mio.
YAML	70 Mio.
"University of Applied Sciences Upper Austria"	100 K



Motivation für XML

Von HTML zu XML

- HTML (*HyperText Markup Language*) ist die "Lingua Franca" zur Beschreibung von Hypertextdokumenten im Web
- Grundkonzept: Auszeichnungen ("Markup") in Form von "Tags" (*start tag, end tag*)
- Einschränkungen:
 - Beschränkte Anzahl vordefinierter Tags
 - Erweiterungen um (proprietary) Tags
 - Tags beschreiben vorwiegend Layout-Aspekte
 - Strukturelle Tags fehlen
 - Suche im Web erschwert; nur einfache Anfragen möglich

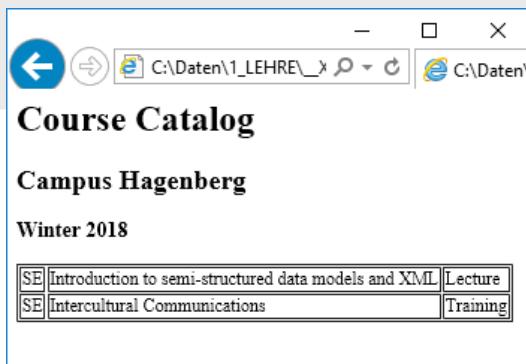
"The problem with HTML-WYSIWYG is that what you see is all you've got."
(Brian Kernighan, Entwickler von C)

Motivation für XML

Von HTML zu XML

HTML beschreibt das
Layout des Dokumentinhalts

```
<h1>Course Catalog</h1>
<h2>Campus Hagenberg</h2>
<h3>Winter 2018</h3>
<table border="1">
  <tr>
    <td>SE</td>
    <td>Introduction to ...</td>
  </tr>
  <tr>
    <td>SE</td>
    <td>Intercultural Communications</td>
  </tr>
  ...
</table>
```

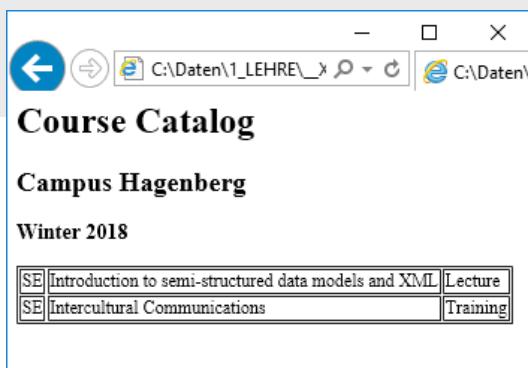


Motivation für XML

Von HTML zu XML

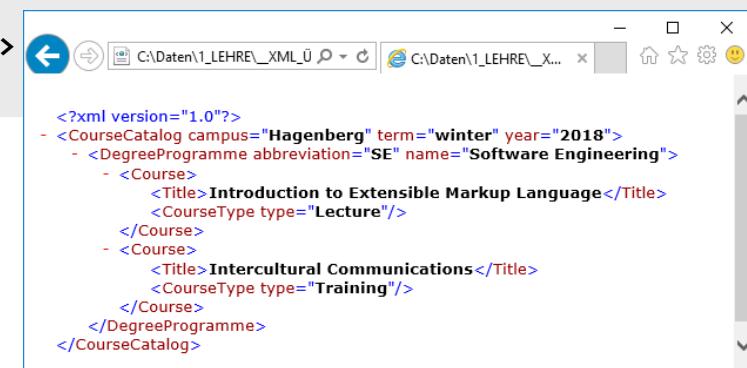
HTML beschreibt das Layout des Dokumentinhalts

```
<h1>Course Catalog</h1>
<h2>Campus Hagenberg</h2>
<h3>Winter 2018</h3>
<table border="1">
  <tr>
    <td>SE</td>
    <td>Introduction to ...</td>
  </tr>
  <tr>
    <td>SE</td>
    <td>Intercultural Communications</td>
  </tr>
  ...
</table>
```



XML beschreibt die Struktur u. Semantik des Dokumentinhalts

```
<CourseCatalog year="2018" term="winter" campus="Hagenberg">
  <DegreeProgramme name="Software Engineering" abbrev="SE">
    <Course>
      <Title>Introduction to ...</Title>
      <CourseType type="Lecture"/>
    </Course>
    <Course>
      <Title>Intercultural Communications</Title>
      <CourseType type="Training"/>
    </Course>
  </DegreeProgramme>
</CourseCatalog>
```



"XML will become the ASCII of the 21st century - basic, essential, unexciting."
(Tim Bray, Co-Editor of XML 1.0)

Was ist nun XML?

■ Generische Auszeichnungssprache (Markup-Sprache)

- textbasierte Sprache, die Dokumente mit zusätzlichen Tags („Markierungen“ = Strukturinformationen) versieht
- Kombination von Inhalt (Daten) und Informationen über den Inhalt (Metadaten) in einem Dokument
- keine Tags vorgegeben, beliebige Tags möglich
- Metasprache zur Definition von Sprachen

```
<CourseCatalog year="2018" term="winter" campus="Hagenberg">
  <DegreeProgramme name="Software Engineering" abbreviation="SE">
    <Course>
      <Title>Introduction to semi-structured data models and XML</Title>
      <CourseType type="Lecture"/>
    </Course>
    <Course>
      <Title>Intercultural Communications</Title>
      <CourseType type="Training"/>
    </Course>
  </DegreeProgramme>
</CourseCatalog>
```

Merkmale von XML

- **Layout-Unabhängigkeit**
 - Trennung Struktur u. Semantik des Inhalts von dessen Darstellung
- **Plattform- und Herstellerunabhängigkeit**
 - Lizenzfreie W3C-Standards
- **Erweiterbarkeit**
 - Tags und Attribute können neu definiert werden (Metasprache)
- **Strukturierbarkeit**
 - Tags können beliebig geschachtelt werden
- **Semistrukturertheit**
 - Inhalt kann nicht-strukturierte Teile enthalten
 - Information trägt einen Teil der Struktur mit sich
- **Selbstbeschreibend**
 - ... für den Menschen: einfach zu lesen u. zu erstellen
 - ... für die Maschine: einfach zu generieren u. zu parsen
- **Validierbarkeit**
 - XML-Dokumente können ein Dokumentenmodell, d.h. eine formale Beschreibung ihres Vokabulars und ihrer Grammatik aufweisen und gegenüber diesem validiert werden.

Merkmal – Wohlgeformtheit

■ Wohlgeformtheit *

- Es existiert genau ein Wurzelement
- Jedes Start-Tag muss ein dazugehöriges End-Tag besitzen
- Tags dürfen einander nicht überschneiden
- Attributwerte müssen in Anführungszeichen stehen (paarweise "... " oder '...')
- Element- und Attributbezeichner müssen XML-Namen sein (Namenskonvention)
- XML ist case-sensitive (SE != se)
- Ein Element darf nicht zwei Attribute mit gleichem Namen besitzen
- Kommentare dürfen nicht innerhalb Tags stehen
- Reservierte Zeichen < und & dürfen nicht innerhalb von Elementinhalten oder Attributwerten auftreten (müssen ggf. maskiert werden)
- ... es gibt noch mehr

```
<CourseCatalog year="2018" term="winter" campus="Hagenberg">
  <DegreeProgramme name="Software Engineering" abbreviation="SE">
    <Course>
      <Title>Introduction to semi-structured data models and XML</Title>
      <CourseType type="Lecture"/>
    </Course>
    <Course>
      <Title>Intercultural Communications</Title>
      <CourseType type="Training"/>
    </Course>
  </DegreeProgramme>
</CourseCatalog>
```

* grundlegende syntaktische Korrektheit,
unabhängig vom anwendungsspezifischen Einsatz

Merkmal – Wohlgeformtheit

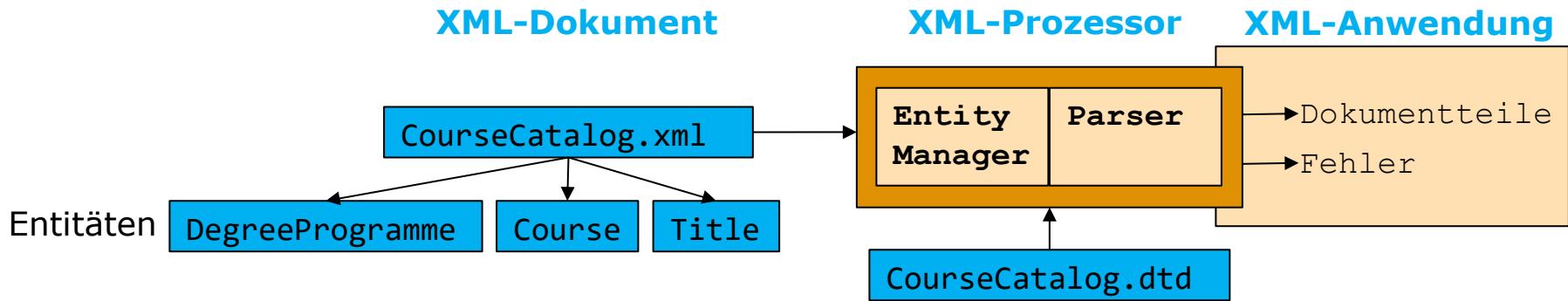
- Wohlgeformte Dokumente können zwar von jedem XML-Parser eingelesen werden, jedoch sagt die Wohlgeformtheit nichts darüber aus
 - welche Elementnamen überhaupt vorkommen dürfen,
 - in welcher Reihenfolge die Elemente im XML-Dokument erscheinen müssen,
 - welches Element Kindelement eines anderen sein darf
 - wie oft die Elemente im XML-Dokument erscheinen dürfen,
 - welche Attribute in bestimmten Elementen verwendet werden dürfen
 - ...

Merkmal – Gültigkeit

- Gültigkeit (Validität)
 - XML-Dokument ist wohlgeformt und
 - entspricht einem formalen Dokumentenmodell
- Formales Dokumentenmodell definiert
 - die ihr bekannten bzw. von ihr akzeptierten Elemente (**Vokabular**) sowie
 - die Dokumentenstruktur (**Grammatik**)
- Formales Dokumentenmodell kann mit Hilfe von Schemata definiert werden, z.B. mit
 - einer sog. **Document Type Declaration** (DTD, beschränkte Möglichkeiten) oder
 - einem **XML Schema** (aktuell)

Merkmal – Gültigkeit

- XML-Prozessoren lesen XML-Dokumente ein und
 - überprüfen entweder nur deren Wohlgeformtheit (nicht-validierende Prozessoren)
 - oder auch deren Validität (validierende Prozessoren)



- können in Anwendungen (z.B. Web-Browser, Textverarbeitung, DB-Server, Web-Server) eingebunden werden
- zerlegen ein XML-Dokument in seine Informationseinheiten und erstellen einen Baum

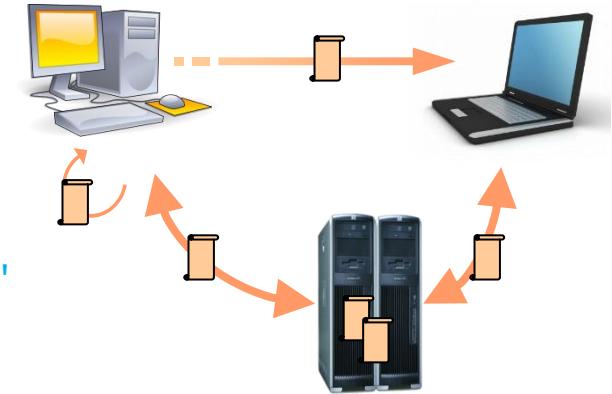
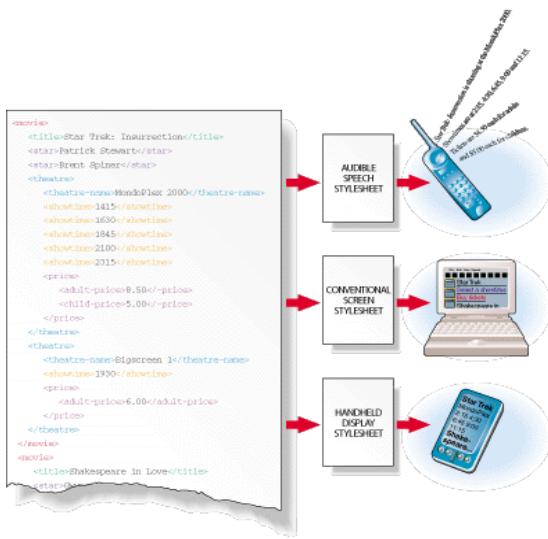
Anwendungsbereiche von XML

Wobei soll XML unterstützen?

■ Globale Sprache für den **Datenaustausch**

- über XML als reine Austauschnotation oder
- zusätzlich über gemeinsames Schema

"write once, read everywhere"

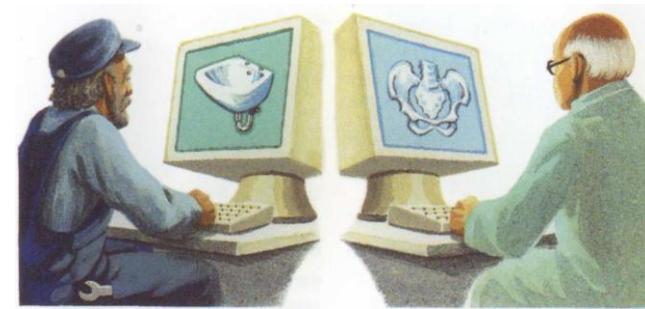


■ Multi-Delivery

- Derselbe Inhalt kann auf verschiedenen Geräten unterschiedlich präsentiert werden

■ Intelligente Suche

- Statt einfacher Schlagwortsuche in HTML-Dokumenten ist eine strukturbasierte Suche in XML-Dokumenten möglich.



Anwendungsbereiche von XML

XML-Periodensystem

ADF																									
AdsML	RoadXML																								
ARTS XML	RosettaNet																								
BIML	SportsML	DSD	AOS	GeoSciML	MusicXML	phyloXML	Topic Maps																		
BPEL	StratML	DSDL	BeerXML	GML	NeuroML	PMML	TumorML																		
CXML	UBL	DTD	CDA	HumanML	Oasis XML Catalog	PNML	UnitsML	ASF	MOWL	SDML	WS-Policy														
FpML	UDDI	NVDL	CellML	IDML	ODD	RecipeML	WaterML	CDF	NRL	SOAP	XACML														
GJXML	VAST	RELAX	CML	JATS	ODF	REML	WordML	EPP	OML	SPML	XBEL	XML-Enc	COLLADA	NCL	LOM										
ifcXML	WITSML	RELAX NG	DAISY	KML	OEB	RTML	xCal	Facelets VDL	OPML	SwA	XFA	XML-RPC	DotML	SBGN	MARCXML										
NIEM	XBRL	Schematron	DITA	LIDO	OMDoc	RuleML	XCES	GXA	RAML	Vexi	XHTML	XML-Sig	gbXML	SVG	MESH										
NITF	XFT	SQF	DocBook	MathML	OOXML	SBML	XDF	HTML	RSS	Web feed	XHTML Basic	XMPP	GraphML	SXBL	NewsML										
ODRL	XMLA	TREX	EAD	MEI	OpenMath	SMIL	XDXF	Microformats	RSS enclosure	WS-Discovery	XHTML Modularization	XOXO	GXL	VML	NewsML-G2										
OFX	XOMGL	XRules	EPUB	MML	OSIS	TEI	XML Infoset	MJML	SALT	WSDL	XINS	XUL	IMML	X3D	ONIX										
RailML	XPDL	XSD	FictionBook	MML	PDBML	ThML	XSL-FO	MMI	SAML	WSFL	XMI	InkML	JDF	XAML	SCORM	GPX	TMX	XPath	XUpdate						

Businesssprache

QS-Prüfung

Dokumentformat

Internetformat

Grafikformat

Metadatenstandard

Datenmanipulation

Softwareunterstützung:
 Dunkel: Volle Unterstützung
 Hell: Teilweise Unterstützung
 Grau: Keine Unterstützung
 Nicht hinterlegt: Keine Angabe

[oxygen XML Editor](#)
[Altova XMLSpy](#)
[Antenna House Formatter](#)

Standard-Zugehörigkeit:
[ISO](#)
[W3C](#)
[OASIS](#)

 Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.

Anwendungsbereiche von XML

Branchen – "Vertikalisierung von XML"

■ Dokumentenmodelle für ...

- | | |
|------------------------|---------------|
| ● Reiseindustrie | openTravel |
| ● Personalwesen | XML-HR |
| ● Sprachapplikationen | VoiceXML |
| ● Vektorgraphiken | SVG |
| ● Mobile Applikationen | WML |
| ● Gesundheitswesen | HL7 |
| ● Mathematik | MathML |
| ● Börsentransaktionen | FIXML |
| ● Finanzinformationen | OFX |
| ● eGovernment | eGovML |
| ● Chemie | CML |
| ● News | NewsML |
| ● Literatur | Gutenberg |
| ● Semantische Webs | RDF, OWL |
| ● Geodaten | GML |
| ● ... | OpenStreetMap |

■ ... Electronic Commerce

- **xCBL**: XML Common Business Library (Commerce One)
- **BizTalk**: Microsoft
- **cXML**: Commercial Extensible Mark-up Language
- **ebXML**: OASIS + XML/EDI
- **FpML**: Financial Products Markup Language
- **ebInterface**: Rechnungsstandard

■ ... E-Learning

- **IMS** Learning Design, Beschreibung von E-Learning-Content:
www.imsglobal.org/learningdesign/
- **SCORM** (Sharable Content Object Reference Model)

■ ... Büroanwendungen

- **ODF**: OpenDocument
- **MS**: Office Open XML

Anwendungsbereiche von XML

XML Standardisierung – "Horizontalisierung von XML"

■ **XML Namespaces**

- Unterstützung global eindeutiger Element- und Attributnamen

■ **XPath** (XML Path Language)

- Pfadausdrücke zur Navigation in XML-Dokumenten

■ **XQuery** (XML Query Language)

- XML-Anfragesprache

■ **XML Schema**

- Sprache zur Beschreibung von XML-Schemata in XML

■ **XSL** (Extensible Stylesheet Language)

- **XSLT**: Transformation von XML-Dokumenten (deklarativ)
- **XSL-FO**: Formatierung von XML-Dokumenten (deklarativ)

■ **DOM** (Document Object Model)

- API für den prozeduralen Zugriff auf XML-Dokumente

■ ...

"It takes ten minutes to understand (base) XML, but then ten month to understand the new technologies hung around it."
(Peter Chen)

Geschichte von XML

- 1945: Hypertext
- 1969: GML
- 1986: SGML (ISO Standard)
- 1989: HTML (Tim Berners-Lee, CERN)
- 1994: W3C gegründet
- 1996: SGML Subset Arbeitsgruppe gegründet
- 1998: XML 1.0
- ...
- 2006: XML 1.1
- ... laufend neue Industriestandards

The screenshot shows the W3C XML Recommendation page from February 10, 1998. The page header includes the W3C logo and the document identifier "REC-xml-19980210". Below the header, the title "Extensible Markup Language (XML) 1.0" is displayed, followed by the subtitle "W3C Recommendation 10-February-1998". The page contains several sections with hyperlinks to various XML-related documents. At the bottom, there are sections for "Editors" (Tim Bray, Jean Paoli, C. M. Sperberg-McQueen) and their contact information.

REC-xml-19980210

Extensible Markup Language (XML) 1.0

W3C Recommendation 10-February-1998

This version:

<http://www.w3.org/TR/1998/REC-xml-19980210>
<http://www.w3.org/TR/1998/REC-xml-19980210.xml>
<http://www.w3.org/TR/1998/REC-xml-19980210.html>
<http://www.w3.org/TR/1998/REC-xml-19980210.pdf>
<http://www.w3.org/TR/1998/REC-xml-19980210.ps>

Latest version:

<http://www.w3.org/TR/REC-xml>

Previous version:

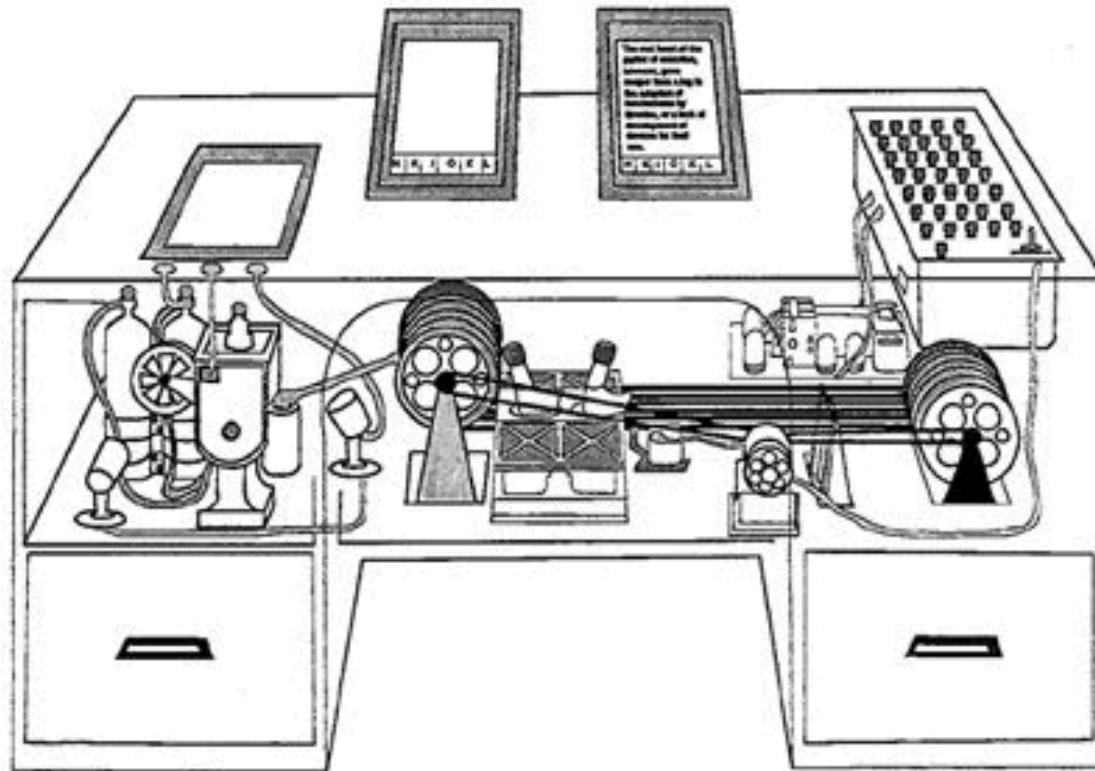
<http://www.w3.org/TR/PR-xml-971208>

Editors:

Tim Bray (Textuality and Netscape) <tbray@textuality.com>
Jean Paoli (Microsoft) <jpaoli@microsoft.com>
C. M. Sperberg-McQueen (University of Illinois at Chicago) <cmsmcq@uic.edu>

Geschichte von XML

Memex – Geburtsstunde von Dokumentenbeschreibungssprachen



Memex-Maschine (**M**emory **E**xtender)



Bush, Vannevar, "As we may think", 1945
[<https://www.w3.org/History/1945/vbush/>]

Geschichte von XML

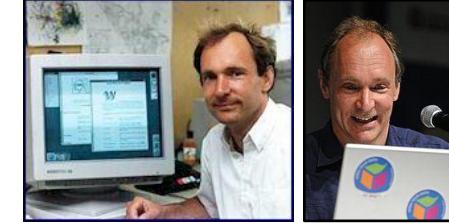
"Markup" und "Generic Coding"

- Informationsgehalt eines Dokuments wird von seiner äußereren Form getrennt
- Buchdruck: "Editorial structure tags" (Ende der 60er)
⇒ "Generic Markup"
- "Markup": Begriff aus dem Verlagswesen
 - Nach der inhaltlichen Überprüfung eines Werkes erfolgt Bearbeitung durch einen Layouter (Hinzufügen von Seitenformat, Zeichensätze, ...)
 - Zuerst manuell, danach elektronisch unterstützt (Steuerzeichen, Makros)
- Konzept ist weiterhin aktuell
 - Textverarbeitungsprogramme arbeiten mit Formatvorlagen
 - Darstellung nicht über Steuerzeichen, sondern als bereits formatierter Text
- Generic Coding (GenCode-Konzept)
 - Wesentliche Idee von SGML
 - Struktur und logische Elemente eines Textes kennzeichnen
 - Die Kennzeichnungen (Markup) beschreiben dann die Art der gekennzeichneten Elemente ⇒ sie beschreiben diese genauer
 - Nicht format-orientiert

Organisation hinter XML

World Wide Web Consortium (W3C)

- World Wide Web Consortium [www.w3.org]
 - 1994 am MIT gegründet
 - Direktor: Tim Berners-Lee
- Industiekonsortium
 - ca. 480 Mitgliedsunternehmen und Forschungseinrichtungen (aktuelle Mitglieder, [- Entwicklung von einheitlichen Technologien, die den Fortschritt des Webs fördern und seine Interoperabilität sicherstellen
 - Keine Normierungsorganisation im klassischen Sinn
 - kann Einhaltung von Normen nicht auf rechtlichem Wege einklagen
 - definiert deshalb "lediglich" Empfehlungen \(Recommendations\)
 - Prozesse bzw. Reifegrade von Standards im "Consortium Process Document" beschrieben.](http://www.w3.org/Consortium/Member>List)■ Aufgaben<ul style=)
- Produkte
 - Recommendations (90%)
 - Software (10%)



XML ...

- ... ist eine effiziente, flexible und einfache Metasprache, mit der Auszeichnungssprachen definiert werden können, d.h. eine Sprache zur Definition von weiteren Sprachen
- ... trennt Inhalt, Struktur und Layout
- ... ist sprach- und plattformunabhängig
- ... ist ein textbasiertes Format (Unicode)
- ... wurde speziell für das Internet/Web entwickelt
- ... dient vorwiegend als Dateiaustauschformat
- ... bildet den Kern einer "Technologie-Familie", die Validierung, Abfrage und Transformation unterstützt
- ... ist ein W3C-(Industrie-)Standard und offen!

Modul 2

XML Grundlagen

Julian Haslinger

<?xml?>



Der vorliegende Foliensatz basiert vorwiegend auf:

*Elliotte Rusty Harold, W. Scott Means: XML in a Nutshell: A Desktop Quick Reference, 3rd Edition, O'Reilly, 2005
Wilfried Grupe: XML Grundlagen, Technologien, Validierung, Auswertung, 1. Auflage, mitp Verlag, 2018*

Inhalt



- **XML-Dokument: Aufbau und Bestandteile**
- XML-Prolog: Dokumenteigenschaften
- DTD: Dokumenttypdefinition
- Entities und Verweise

XML-Dokument 1/7

Aufbau

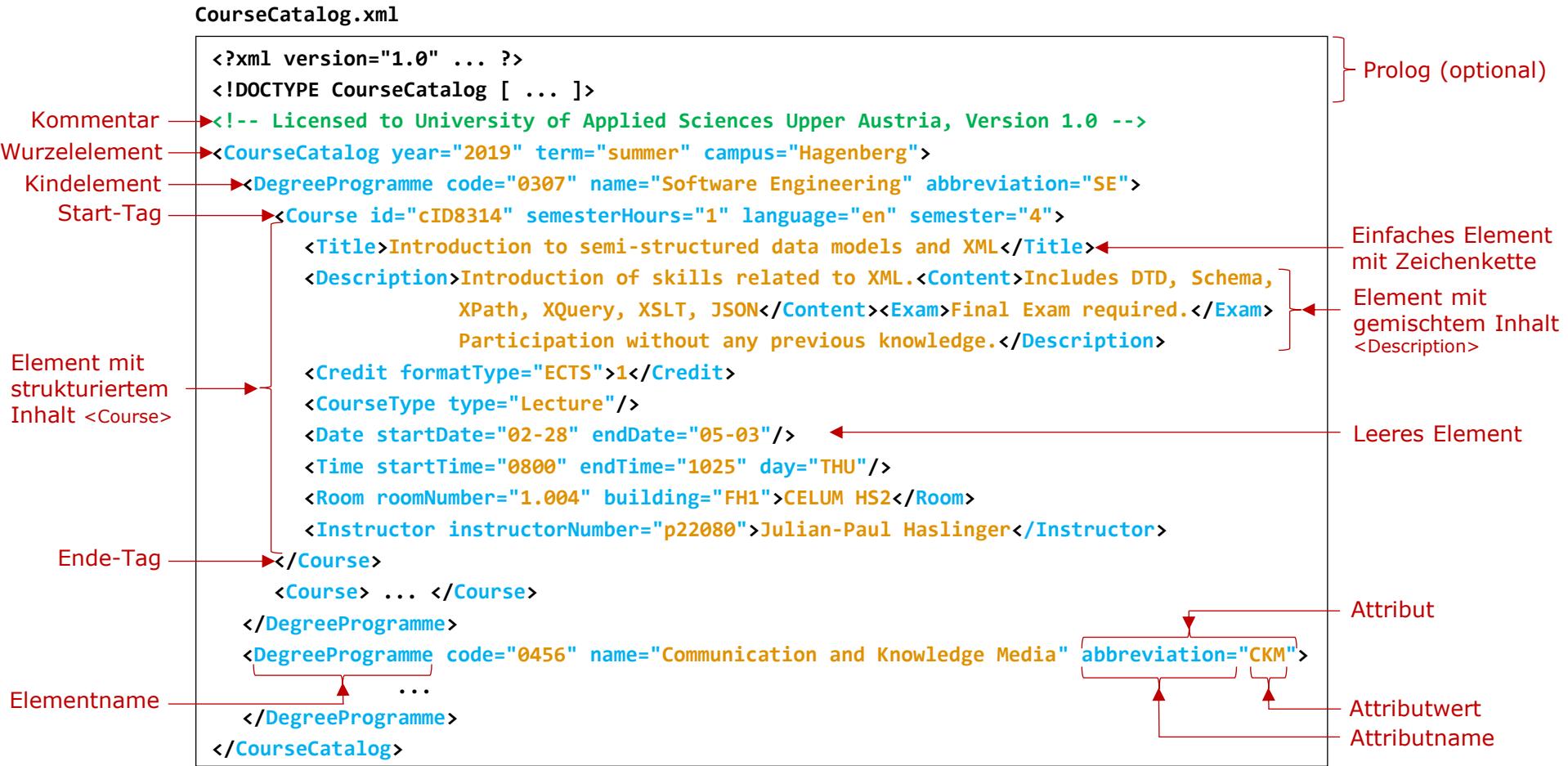
CourseCatalog.xml

XML-Deklaration	Prolog
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>	
<!DOCTYPE CourseCatalog [...]>	Dokumenttyp-Deklaration (DTD)
<!-- Licensed to University of Applied Sciences Upper Austria, Version 1.0 -->	
<CourseCatalog year="2019" term="summer" campus="Hagenberg">	
<DegreeProgramme code="0307" name="Software Engineering" abbreviation="SE">	
<Course id="cID8314" semesterHours="1" language="en" semester="4">	
<Title>Introduction to semi-structured data models and XML</Title>	
<Description>Introduction of skills related to XML.<Content>Includes DTD, Schema, XPath, XQuery, XSLT, JSON</Content><Exam>Final Exam required.</Exam>	
Participation without any previous knowledge.</Description>	
<Credit formatType="ECTS">1</Credit>	
<CourseType type="Lecture"/>	
<Date startDate="02-28" endDate="05-03"/>	
<Time startTime="0800" endTime="1025" day="THU"/>	
<Room roomNumber="1.004" building="FH1">CELUM HS2</Room>	
<Instructor instructorNumber="p22080">Julian-Paul Haslinger</Instructor>	
</Course>	
<Course> ... </Course>	
</DegreeProgramme>	
<DegreeProgramme code="0456" name="Communication and Knowledge Media" abbreviation="CKM">	
...	
</DegreeProgramme>	
</CourseCatalog>	

XML-Dokumentinstanz

XML-Dokument 2/7

Beispiel: CourseCatalog.xml



XML-Dokument 3/7

Elemente und Attribute: Namensregeln

- Element- und Attributnamen müssen gültige XML-Namen sein:
- [letter|_|:] [letter|'0..9'|'.'|'-'|'_'|':']*
- letter umfassen A-Z, a-z, sowie andere Schriftzeichen wie bspw. ä, ê, γ, ζ, Ω
- Verwendung von ':' ist Namensräumen vorbehalten
- Zeichenkette "XML" oder "xml" darf nicht am Anfang von Namen auftreten (reserviert für Processing Instructions)
- keine Längenbeschränkung für Namen
- Unterscheidung zw. Klein- und Großschreibung

```
<!-- FALSCH -->  
<Kurs Katalog>...</Kurs Katalog>  
<2.Semester>...</2.Semester>  
<Raum/Nr>...</Raum/Nr>  
<xMlINFO>...</xMlINFO>
```

XML-Dokument 4/7

Elemente und Attribute

- Leere Elemente können in Lang- oder Kurzform geschrieben werden
 - <HerstellerNr nr="h1234"></HerstellerNr> oder
 - <HerstellerNr nr="h1234"/>
- Attributwerte müssen unter Anführungszeichen gesetzt werden
 - <Smartphone name="iPhone 16"/> oder
 - <Smartphone name='iPhone 16' /> oder
 - <Smartphone name='iPhone"16'" /> oder
 - <Smartphone name="iPhone'16'" />
- Element kann beliebige Anzahl von eindeutigen Attributen enthalten
 - <Smartphone name="iPhone 16" jahr="2024"/>
 - <!-- FALSCH -->
<Smartphone name="iPhone 11" name="iPhone XR" jahr="2019" />

XML-Dokument 5/7

Kommentare

- Können sich über mehrere Zeilen erstrecken
 - beginnt mit <!-- und endet mit -->
 - zwischen Start-Tag und Ende-Tag eines Elements
 - vor oder nach der Elementwurzel
- Einschränkungen
 - nicht vor dem XML-Prolog und innerhalb von Tags erlaubt
 - keine Schachtelung von Kommentaren erlaubt
 - zwei oder mehr aufeinander folgende "-" im Kommentar nicht erlaubt

```
<!--
```

Ein Kommentar kann jede beliebige Art von Text sein, kann Leerzeichen, Zeilenumbrüche und <tagname></tagname> enthalten!

```
-->
```

XML-Dokument 6/7

Verarbeitungsanweisung (Processing Instruction - PI)

- Verarbeitungsanweisung für eine Anwendung, die ein XML-Dokument verarbeitet
 - Der Inhalt einer Verarbeitungsanweisung wird vom XML-Prozessor unverändert an die Anwendung weitergegeben
 - Können an beliebiger Stelle im XML-Dokument stehen
 - Verarbeitungsanweisung beginnt mit einem Namen
 - identifiziert die Zielanwendung oder gibt der Anweisung einen Namen
 - muss ein gültiger XML-Name sein
 - Danach können beliebige Anweisungen stehen

Inhalt



- XML-Dokument: Aufbau und Bestandteile
- **XML-Prolog: Dokumenteigenschaften**
- DTD: Dokumenttypdefinition
- Entities und Verweise

Prolog 2/4

XML-Deklaration ...

- ... ist optional
- ... muss am Anfang der Datei stehen
- ... liefert Informationen an den XML-Parser für die Verarbeitung des Dokuments
- ... legt grundlegende Eigenschaften des XML Dokuments fest:
 - XML-Versionsnummer (**version**)
 - Art der Zeichenkodierung (**encoding**)
 - ggf. Existenz einer DTD (**standalone**)

Prolog 3/4

XML-Deklaration - Parameter

■ **version**

- deklariert verwendete XML-Version
- Parser unterstützen Version "1.0" (ev. auch "1.1")

■ **encoding** (optional)

- definiert die im XML-Dokument verwendete Zeichenkodierung¹
- Standard-Zeichenkodierung ist "UTF-8"

■ **standalone** (optional)

- definiert, ob externe DTD existiert
- Standardwert ist "no"

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
```

Prolog 4/4

XML-Deklaration - Zeichenkodierung

■ XML-Parser

- müssen gemäß XML-Spezifikation intern mit Unicode (UTF-8 oder UTF-16) arbeiten

■ Unicode

- kann alle nationalen Zeichen darstellen
- insgesamt ca. 65.000 Zeichen

■ encoding-Attribut

- Zeichenkodierung der betreffenden XML-Datei
- Fehlt das Attribut, dann wird Kodierung mit Unicode angenommen
- Beachte: XML-Parser müssen gemäß XML-Spezifikation nur Unicode verarbeiten können!

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
...  
<Course id="cID8314">  
  <Title>Introduction to XML</Title>  
  ...  
</Course>  
...
```

XML-Dokument

gespeichert als

Unicode
UTF-8

XML-Datei

Inhalt

- XML-Dokument: Aufbau und Bestandteile
- XML-Prolog: Dokumenteigenschaften
- **DTD: Dokumenttypdefinition**
- Entities und Verweise

DTD

Zweck

- Beschreibt Vokabular und Grammatik für eine Menge von XML-Dokumentinstanzen
 - deklariert eine Menge von erlaubten Elementen (Vokabular)
 - definiert ein Inhaltsmodell für jedes Element (Grammatik)
 - deklariert für jedes Element eine Menge von zulässigen Attributlisten
- Stellt Mechanismen zur Verwaltung des Dokumentenmodells bereit
 - Textersetzung
 - Einbindung von Teilen des Dokumentenmodells aus externer Datei

DTD

Charakteristika

- XML-Dokument darf nur eine DTD einbinden
- DTD muss im XML-Dokument nach dem Prolog, jedoch vor der Elementwurzel eingebunden werden
- DTD legt nicht die Elementwurzel eines XML-Dokuments fest
 - dies erfolgt durch das XML-Dokument selbst innerhalb der **DOCTYPE**-Deklaration
 - kann ein beliebiges Element der DTD sein

CourseCatalog.xml

```
<?xml version="1.0" ... ?>
<!DOCTYPE CourseCatalog [ ... <!-- Interne CourseCatalog-DTD --> ... ]>
<CourseCatalog>
  ...
</CourseCatalog>
```

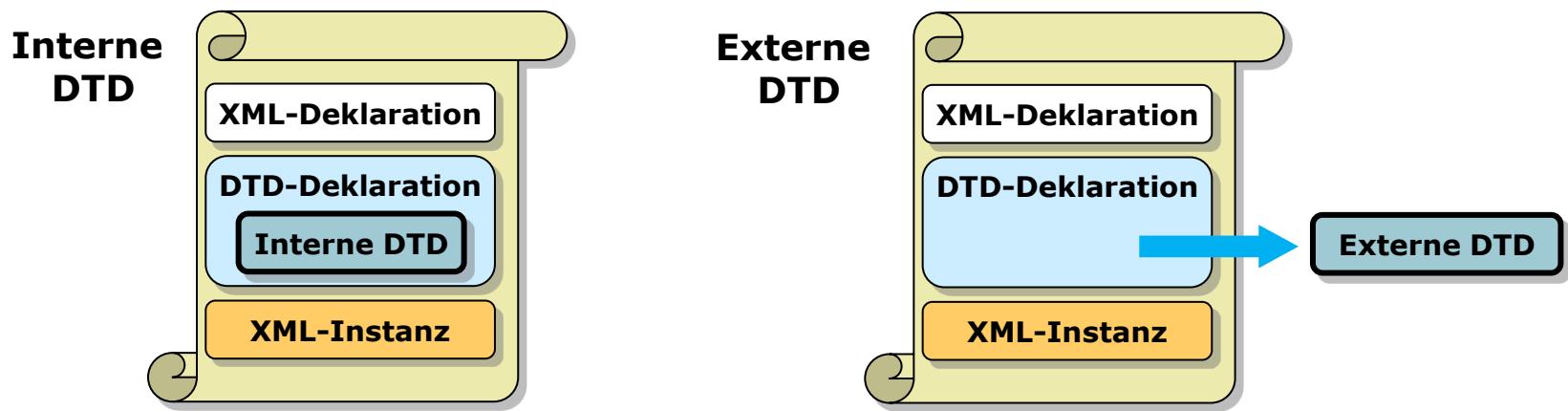
The diagram shows the XML code with annotations:

- An orange arrow points from the **DOCTYPE** declaration to the opening tag **<CourseCatalog>**, labeled "Definition".
- A red bracket below the opening tag **<CourseCatalog>** is labeled "Verwendung" (Usage).
- A red bracket to the right of the opening tag is labeled "Wurzelement" (Root element).

DTD

Einbindung von DTD's in XML-Dokumenten - 3 Alternativen

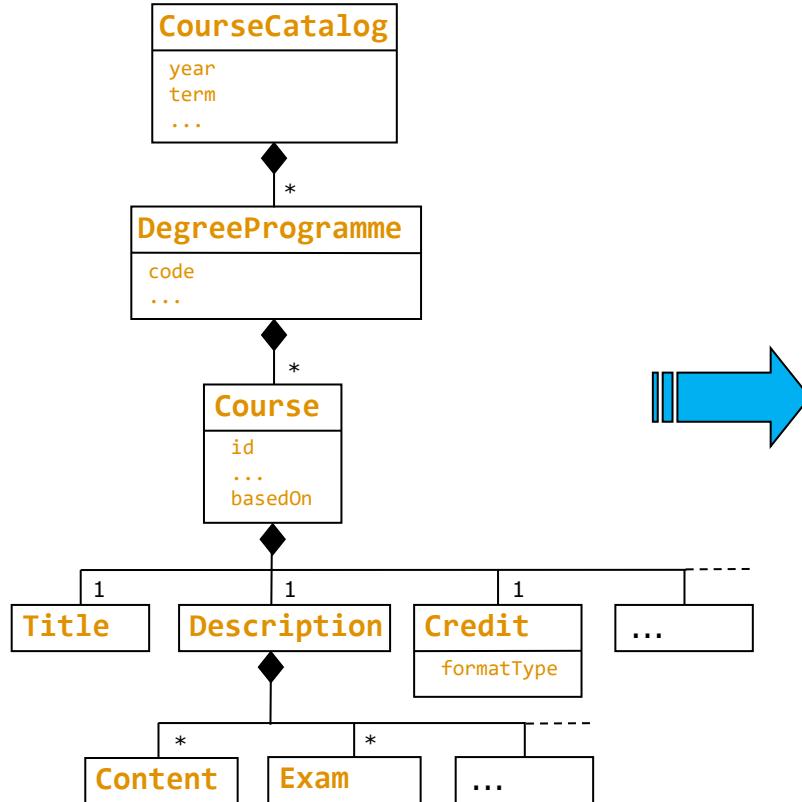
1. Interne DTD, d.h. im Dokument selbst definiert
`<!DOCTYPE CourseCatalog [...]>`
2. Externe DTD, d.h. in eigener Datei (*.dtd)
`<!DOCTYPE CourseCatalog SYSTEM "CourseCatalog.dtd">` oder
`<!DOCTYPE CourseCatalog SYSTEM "http://www.fh-hagenberg.at/dtds/CourseCatalog.dtd">`
3. Externe und interne DTD, d.h. externe ergänzt interne DTD



DTD

Beispiel: CourseCatalog.dtd

CourseCatalog.uml



CourseCatalog.dtd

```

<!-- CourseCatalog.dtd Version 1.0 -->
<!ELEMENT CourseCatalog (DegreeProgramme*)>
<!ATTLIST CourseCatalog year CDATA #REQUIRED>
<!ATTLIST CourseCatalog term (summer|winter) #REQUIRED>
...
<!ELEMENT DegreeProgramme (Course*) >
<!ATTLIST DegreeProgramme code CDATA #REQUIRED>
...
<!ELEMENT Course (Title, Description, Credit, ...)>
<!ATTLIST Course id ID #REQUIRED>
<!ATTLIST Course basedOn IDREFS #IMPLIED>
...
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Description (#PCDATA|Content|Exam)*>
<!ELEMENT Content (#PCDATA)>
<!ELEMENT Exam (#PCDATA)>
...
<!ELEMENT Credit (#PCDATA)>
<!ATTLIST Credit formatType (ECTS|CP1) "ECTS">
...
  
```

The DTD defines the XML structure. It starts with the root element CourseCatalog, which contains multiple DegreeProgramme elements. Each DegreeProgramme element has an attribute year and an attribute term with values summer or winter. The DegreeProgramme element also contains multiple Course elements. Each Course element has an attribute id. The Course element is defined to contain multiple Title, Description, and Credit elements. The Description element can contain Content or Exam elements. The Credit element has an attribute formatType with values ECTS or CP¹, with a default value of ECTS. Ellipses indicate additional elements and attributes.

Legende:

XML Element	1 : genau eines
XML Attribut	1..* : ein oder mehrere
	* : null oder mehrere
	◆ : besteht aus

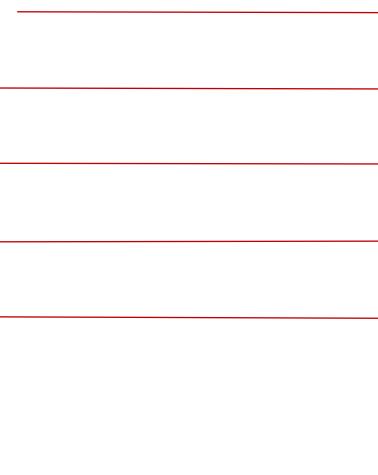
¹ CP ... Credit Points

DTD

Inhaltsmodelle von Elementen

■ Inhaltsmodelle:

1. Unstrukturierter Inhalt
2. Strukturierter Inhalt
3. Gemischter Inhalt
4. Leerer Inhalt
5. Beliebiger Inhalt



```
<!ELEMENT Elementname (Inhaltsmodell)>
```

Elementtypdeklaration

Inhaltsmodell legt den erlaubten Inhalt eines Elements fest.

DTD

1. Unstrukturierter Inhalt

DTD:

```
<!ELEMENT Title (#PCDATA)>
```

Dokumentinstanz:

```
<Title>Introduction to semi-structured data models and XML</Title>
```

- Element enthält einfachen Text ohne Kindelemente
- Inhalt auch als *Parsed Character Data* (**PCDATA**) bezeichnet
 - *character data*: einfache Zeichenkette
 - *parsed*: Zeichenkette wird vom Parser analysiert, um End-Tag zu identifizieren
 - reservierte Symbole < und & nicht erlaubt (hierfür Zeichenentitäten < bzw. & benutzen)

DTD

2. Strukturierter Inhalt

DTD:

```
<!ELEMENT CourseCatalog (DegreeProgramme)>
```

Dokumentinstanz:

```
<CourseCatalog>
  <DegreeProgramme> ... </DegreeProgramme>
</CourseCatalog>
```

- (Eltern-)Element hat ein oder mehrere Kindelemente
- kein Text vor, nach oder zwischen den Kindelementen
- Elemente können beliebig tief geschachtelt werden

DTD

2. Strukturierter Inhalt - Sequenz

DTD:

```
<!ELEMENT Course (Title, Description)>
```

Dokumentinstanz:

```
<Course>
  <Title> ... </Title>
  <!-- Kommentare sind erlaubt -->
  <Description> ... </Description>
</Course>
```

- Kindelemente (**Title**, **Description**) sind nur in der angegebenen Reihenfolge erlaubt

DTD

2. Strukturierter Inhalt - Alternative

DTD:

```
<!ELEMENT Course (Title, (Room | Location))>
```

Dokumentinstanz:

```
<Course>
  <Title> ... </Title>
  <Room> ... </Room>
</Course>

<Course>
  <Title> ... </Title>
  <Location> ... </Location>
</Course>
```

- Genau eines der alternativen Kindelemente ist erlaubt

DTD

2. Strukturierter Inhalt - Häufigkeitsangaben

- Notwendig (einmal)
`<!ELEMENT CourseCatalog (DegreeProgramme)>`
- Notwendig und wiederholbar (ein- oder mehrmals)
`<!ELEMENT Course (Instructor+)>`
- Optional (null- oder einmal)
`<!ELEMENT Course (Comment?)>`
- Optional und mehrmals (null- oder mehrmals)
`<!ELEMENT DegreeProgramme (Course*)>`

- DTD kennt keine Möglichkeit, Wiederholbarkeit genauer, z. B. "1 bis 5 mal", zu definieren!

DTD

3. Gemischter Inhalt

DTD:

```
<!ELEMENT Description (#PCDATA | Content | Exam | Tool)*>
```

Dokumentinstanz:

```
<Description>
    Introduction of skills related to XML.<Content>Includes
    DTD, Schema, XPath, XQuery, XSLT, JSON</Content>
    <Exam>Final Exam required.</Exam>Participation without
    any previous knowledge.
</Description>
```

- alle aufgezählten Elemente können in beliebiger Reihenfolge beliebig häufig vorkommen
- zwischen den Elementen dürfen Zeichenketten vorkommen
- Anmerkung zur DTD-Syntax: **#PCDATA** (am Beginn) und ***** sind obligatorisch für den gemischten Inhalt

DTD

4. Leerer Inhalt

DTD:

```
<!ELEMENT Date EMPTY>
```

Dokumentinstanz:

```
<Date></Date>
<!-- oder -->
<Date/>
```

- weder Text noch Kindelement(e) als Inhalt
- Warum leere Elemente?
 - Elemente können Attribute besitzen (wird später behandelt)
 - Bestimmte Stellen in einem Dokument markieren

DTD

5. Beliebiger Inhalt

DTD:

```
<!ELEMENT Date ANY>
```

- Alle in der DTD definierten Elemente können als Inhalt beliebig oft und in beliebiger Reihenfolge vorkommen
- Inhalt muss wohlgeformt sein
- ANY sollte nicht verwendet werden (schlechter Stil!), da man damit die Strukturüberprüfung verhindert

DTD

Datentypen für Elementdeklaration

- (nur) drei verschiedene Datentypen:
 - Strukturierter Inhalt
 - **#PCDATA**: unstrukturierter Inhalt ohne reservierte Symbole < und &.
 - Gemischter Inhalt
 - **EMPTY**: leerer Inhalt, Element kann aber Attribute haben
 - **ANY**: beliebiger Inhalt (strukturiert, unstrukturiert, gemischt oder leer)
- Beachte: Datentypen wie **INTEGER**, **FLOAT** etc. stehen nicht zur Verfügung.

DTD

Attributdeklaration

DTD:

```
<!ATTLIST Elementname  
        Attributname Attributtyp Attributbedingung  
        ...  
>
```

- Attributnamen müssen innerhalb eines Elements eindeutig sein
- Attributbedingung
 - Obligatorischer Attributwert **#REQUIRED**
 - Optionaler Attributwert **#IMPLIED**
 - [Fixer] Vorgabewert **[#FIXED] "wert"**

#REQUIRED
#IMPLIED
[#FIXED] "wert"

} Kein Vorgabewert möglich

DTD

Attributdeklaration - Attributtypen

DTD:

```
<!ATTLIST CourseCatalog  
    year CDATA #REQUIRED>
```

Dokumentinstanz:

```
<CourseCatalog year="2019">  
    ...  
</CourseCatalog>
```

- **CDATA** (character data)
 - Beliebige Zeichenkette
 - <, " und & nicht erlaubt (entsprechend zu maskieren)

DTD

Attributdeklaration - Attributtypen

DTD:

```
<!ATTLIST Course
    id      ID      #REQUIRED
    basedOn IDREFS #IMPLIED>
```

Dokumentinstanz:

```
<Course id="cID_65"> ... </Course>
<Course id="cID_66"> ... </Course>
<Course id="cID_75" basedOn="cID_65 cID_66"> ... </Course>
```

■ **ID, IDREF(S)**

- **ID** gewährleistet Eindeutigkeit von Attributwerten innerhalb eines Dokuments, d.h., zwei Attribute vom Typ **ID** dürfen niemals gleichen Wert haben
- pro Element ist nur ein Attribut vom Typ **ID** erlaubt
- **IDREF** ist eine Referenz auf ein Attribut vom Typ **ID**
- referentielle Integrität (ungetypt!) wird durch XML-Parser geprüft
- Werte von **ID**- u. **IDREF(S)**-Attributen müssen gültige XML-Namen sein, d.h. dürfen z.B. nicht mit Zahlen beginnen

DTD

Attributdeklaration - Attributtypen

DTD:

```
<!ATTLIST CourseCatalog  
    year NMTOKEN #REQUIRED>
```

Dokumentinstanz:

```
<CourseCatalog year="2019">  
    ...  
</CourseCatalog>
```

■ NMTOKEN(S)

- XML-Namenstoken sind eine erweiterte Form von XML-Namen
- können zusätzlich mit "0..9", "." und "-" beginnen
- Leerzeichen in XML-Namenstoken sind nicht erlaubt
- NMTOKEN ist ein restriktiverer Datentyp wie CDATA

DTD

Attributdeklaration - Attributtypen

DTD:

```
<!ATTLIST CourseCatalog  
    term (summer|winter) #REQUIRED>
```

Dokumentinstanz:

```
<CourseCatalog term="summer">  
    ...  
</CourseCatalog>
```

- Aufzählungstyp
 - vorgegebene Wertemenge bestehend aus erlaubten Attributwerten (= XML-Namenstoken)

DTD

Entwurfsentscheidung: Element vs. Attribut

- keine allgemeingültige Antwort, aber Anhaltspunkte
 - Element muss verwendet werden, wenn
 - Inhalt weiter strukturiert werden soll
 - Reihenfolge relevant ist (bei Attributen beliebig!)
 - Elemente mehrmals vorkommen sollen (Attribut kann pro Element nur einmal vorkommen!)
 - Attribut muss verwendet werden, wenn man
 - Aufzählungstyp, Vorgabewert, fixer Wert, ID/IDREF einsetzen möchte
- Faustregel
 - Attribute für einfache, unstrukturierte Zusatz- bzw. Metainformationen für Elemente geeignet
 - Alternative Bedingungen sollten durch Attributwerte repräsentiert werden und nicht durch die An- bzw. Abwesenheit von Elementen
 - Elemente sollen für die eigentlichen Daten genutzt werden oder als "künstliches" Gruppierungselement
 - Einheitliche Darstellung mit Elementen eleganter (aber speicherintensiver), Darstellung mit Attributen kompakter

Inhalt



- XML-Dokument: Aufbau und Bestandteile
- XML-Prolog: Dokumenteigenschaften
- DTD: Dokumenttypdefinition
- **Entities und Verweise**

DTD

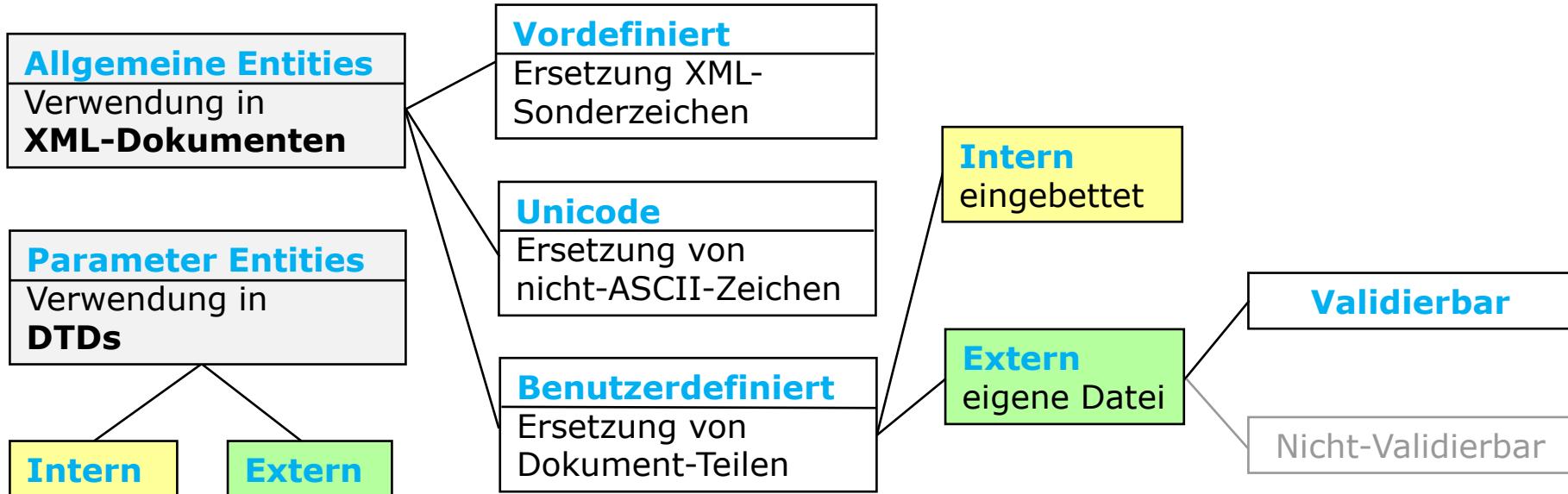
Zweck

- Beschreibt Vokabular und Grammatik für eine Menge von XML-Dokumentinstanzen
 - deklariert eine Menge von erlaubten Elementen (Vokabular)
 - definiert ein Inhaltsmodell für jedes Element (Grammatik)
 - deklariert für jedes Element eine Menge von zulässigen Attributlisten
- Stellt Mechanismen zur Verwaltung des Dokumentenmodells bereit
 - Textersetzung
 - Einbindung von Teilen des Dokumentenmodells aus externer Datei

Entities

Überblick

- Referenzierbare, mit Namen versehene Textteile
 - eines XML-Instanzdokuments
 - oder einer DTD
- Zweck: Zeichen-/Textersetzung, Modularisierung
- Verarbeitung: Referenzen werden beim Parse-Vorgang expandiert



Entities

Vordefinierte (od. Namendefinierte) Entities

- Zweck:
 - Verwendung von XML-Sonderzeichen (z.B. <, >, &, ', ") im Text
- 5 vordefinierte Entities
 - `&` & (Kaufmännisches Und)
 - `<` < (Kleiner als)
 - `>` > (Größer als)
 - `'` ' (Apostrophe: einfache Anführung)
 - `"` " (Doppelte Anführung)
- Beispiel:
 - `<formel>x < y</formel>`
- Verwendung:
 - als Element (#PCDATA) oder Attributwert
- Alternative: CDATA-Abschnitt
 - Beispiel:
`<formel>x <![CDATA[<]]> y</formel>`
 - CDATA-Abschnitt darf Begrenzer ']]>' nicht enthalten
 - CDATA-Abschnitt darf nicht geschachtelt werden

Entities

Unicode (od. Zahlendefinierte) Entities

- Zweck:
 - Repräsentation von Zeichen, die am Eingabegerät nicht verfügbar sind
 - Unicode-Standard [<http://www.unicode.org/>]

- Unicode klassifiziert Zeichen nach Buchstaben, Zahlen, Interpunktionszeichen, Symbolen (allgemein, technisch, mathematisch) etc.
- Eindeutige Zuordnung Zeichen <-> Zahl
- Unterstützt 25 lebende Sprachen (Cyrillic, Hebrew, Hiragana etc.)
- Insgesamt ca. 60.000 verschiedene Zeichen
- Verwendung:
 - in Element- oder Attributwert
 - beliebige Unicode-Zeichen werden über deren Nummer (dezimal oder hexadezimal) referenziert

```
<?xml version="1.0" encoding="iso-2022-jp"?>
<!DOCTYPE 週報 SYSTEM "weekly-iso-2022-jp.dtd">
<!-- 週報サンプル -->
<週報>
  <業務報告リスト>
    <業務報告>
      <業務名>XMLエディターの作成</業務名>
      <業務コード>X3355-23</業務コード>
      <予定項目リスト>
        <予定項目>
          <P>XMLエディターの基本仕様の作成</P>
        </予定項目>
      </予定項目リスト>
    <実施事項リスト>
      <実施事項>
        <P>XMLエディターの基本仕様の作成</P>
      </実施事項>
    <実施事項>
      <P>競合他社製品の機能調査</P>
    </実施事項>
    <実施事項リスト>
      <実施事項>
        <P>XMLとは何かわからない。</P>
      </実施事項>
    </実施事項リスト>
    <問題点対策>
      <P>XMLとは何かわからない。</P>
    </問題点対策>
  </業務報告>
</業務報告リスト>
</週報>
```

**©; © und ©
stellen das copyright
Zeichen dar!**

Entities

Benutzerdefinierte Interne Entities

- Zweck:
 - Platzhalter für wiederkehrende Textfragmente (Text oder wohlgeformtes Markup wird mit einem Namen versehen)
- Deklaration in DTD:

```
<!ENTITY entityName "Ersetzungstext">
```

 - Ersetzungstext ist bei der Deklaration anzugeben
 - zyklische Referenzen in Entities nicht erlaubt
- Verwendung im XML-Dokument:

```
&entityName;
```

 - in Element- oder Attributwerten des XML-Dokuments

Entities

Benutzerdefinierte Externe Analysierte Entities

■ Zweck:

- Modularisierung des XML-Instanzdokuments

■ Deklaration in DTD:

```
<!ENTITY instructors SYSTEM "InstructorList.xml">
```

- Ersetzungstext befindet sich in eigener Datei

■ Charakteristika:

- prinzipiell wohlgeformt, darf mehrere Elementwurzeln aufweisen
- kann keine DTD einbinden

■ Verwendung:

- in Elementwerten des XML-Dokuments und in Entities selbst
- zyklische Referenzen nicht erlaubt
- NICHT in Attributwerten

Entities

Benutzerdefinierte Externe Nicht-Analysierte Entities

DEPRECATED

■ Zweck:

- Einbindung von Dateien, die Nicht-XML-Formate enthalten (z.B. nicht-wohlgeformtes XML, Bilder, Videos)

■ Deklaration in DTD:

```
<!NOTATION JPEG SYSTEM "/usr/local/bin/jpeg_viewer">
<!ENTITY image SYSTEM "http://www.images.com/image.jpeg" NDATA JPEG>
<!ELEMENT Picture EMPTY>
<!ATTLIST Picture src ENTITY #REQUIRED>
```

- **NDATA** kennzeichnet ein nicht-analysiertes (ungeparstes) Entity
- **NOTATION**-Deklaration identifiziert eine Applikation, die Dateien im definierten Format verarbeiten kann

■ Verwendung:

```
<Picture src="image"/>
```

- ausschließlich als Attributwert vom Typ **ENTITY**
- Syntax: Name des Entity in Anführungsstrichen (kein &...;)
- Parser informiert aufrufende Applikation, dass ein nicht-analysiertes Entity existiert - keine Expansion!

■ **(Ausdrucksstärkere) Alternative: W3C's XLink-Standard**

Entities

Benutzerdefinierte, allgemeine Entities - Beispiel

Deklaration

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE CourseCatalog SYSTEM "http://www.fh-hagenberg.at/dtds/CourseCatalog.dtd" [ 
  <!ENTITY nonBreakingSpace "&#160;">
  <!ENTITY finalExam "Final Exam required.">
  <!ENTITY roomCelum "<Room roomNumber='1.004' building='FH1'>CELUM HS2</Room>">
  <!ENTITY instructorsCourseID8314 SYSTEM "InstructorList.xml"> ← Externes, analysiertes Entity
  ...
]>
<CourseCatalog year="2019" term="summer" campus="Hagenberg">
  <DegreeProgramme code="0307" name="Software&nonBreakingSpace;Engineering" abbreviation="SE">
    <Course id="cID8314" semesterHours="1" language="en" semester="4">
      <Title>Introduction to semi-structured data models and XML</Title>
      <Description>Introduction of skills related to XML.<Content>Includes DTD, Schema,  

          XPath, XQuery, XSLT, JSON</Content><Exam>&finalExam;</Exam>  

          Participation without any previous knowledge.</Description>
      <Credit formatType="ECTS">1</Credit>
      <CourseType type="Lecture"/>
      <Date startDate="03-10" endDate="10-01"/>
      <Time startTime="0800" endTime="1025" day="THU"/>
      &roomCelum;
      &instructorsCourseID8314;
    </Course>
  </DegreeProgramme>

```

} Interne Entities

} Externes, analysiertes Entity

InstructorList.xml

```

<Instructor instructorNumber="p22088">Julian Haslinger</Instructor>
<Instructor instructorNumber="p20623">Barbara Traxler</Instructor>

```

Verwendung

Entities

Parameter Entities

■ Zweck:

- Modularer Aufbau und Wiederverwendung von DTDs

■ Deklaration in DTD:

```
<!ENTITY % entityName "Ersetzungstext">  
<!ENTITY % entityName SYSTEM "URI">
```

- % mit Leerzeichen

■ Anwendung nur in DTD:

```
%entityName;
```

- % ohne Leerzeichen
- Namen und Inhaltsmodellen von Elementen
- Attributdeklarationen

Internes Parameter-Entity

```
<!ENTITY % name  
      "(FirstName, LastName)">  
...  
<!ELEMENT Instructor %name;>  
<!ELEMENT Examiner %name;>
```

Externes Parameter-Entity

```
<!ENTITY % courseTypeDefinition  
      SYSTEM "CourseTypeDefinition.mod">  
...  
%courseTypeDefinition;  
...
```

Entities

Parameter Entities - Redeklareration (~Überschreiben)

- Ein in einer externen DTD deklariertes Parameter Entity kann in der internen DTD eines XML-Dokuments überschrieben werden
- Damit kann die externe DTD den Erfordernissen einzelner XML-Dokumente direkt innerhalb derselben angepasst werden, ohne die externe DTD ändern zu müssen
- Verwendung Parameter Entity als eine Art "**Customization Hook**"

CourseCatalog.dtd

```
...
<!ENTITY % name
  "(FirstName, LastName)">
...
```

CourseCatalog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CourseCatalog SYSTEM "CourseCatalog.dtd">
[ <!ENTITY % name
  "(FirstName, LastName, Middlename)">
  ... ]>
```

Entities

Beispiel: CourseCatalog.dtd

Parameter Entities

```
<!-- * Licensed to University of Applied Sciences Upper Austria *
     * http://www.fh-ooe.at/dtds/CourseCatalog.dtd Version 1.0 * -->
<!-- Basic Attributes -->
<!ENTITY % degreeProgrammeAttributes "code CDATA #REQUIRED
                                         name CDATA #REQUIRED
                                         abbreviation CDATA #IMPLIED">
<!ENTITY % courseAttributes "id ID #REQUIRED
                               semesterHours CDATA #REQUIRED
                               language (en|de|fr|es|it) #IMPLIED
                               semester CDATA #REQUIRED
                               basedOn IDREFS #IMPLIED">

<!ELEMENT CourseCatalog (DegreeProgramme*)>
...
<!ELEMENT DegreeProgramme (Course*)>
<!ATTLIST DegreeProgramme %degreeProgrammeAttributes;>

<!ELEMENT Course (Title, Description, Credit, CourseType, ...)>
<!ATTLIST Course %courseAttributes;>
...
```

Entities

Beispiel: CourseCatalog.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE CourseCatalog SYSTEM "http://www.fh-hagenberg.at/dtds/CourseCatalog.dtd"> [
    <!ENTITY % courseAttributes "
        id ID #REQUIRED
        semesterHours CDATA #REQUIRED
        language (en|de|fr|es|it) #IMPLIED
        semester CDATA #REQUIRED
        basedOn IDREFS #IMPLIED
        maxNumParticipants CDATA #IMPLIED">
]
<CourseCatalog year="2019" term="summer" campus="Hagenberg">
    <DegreeProgramme code="0307" name="Software Engineering" abbreviation="SE">
        <Course id="cID8314" semesterHours="1" language="en" semester="4" maxNumParticipants="40">
            <Title>Introduction to semi-structured data models and XML</Title>
            <Description>Introduction of skills related to XML. ....</Description>
            <Credit formatType="ECTS">1</Credit>
            <CourseType type="Lecture"/>
            <Date startDate="02-28" endDate="05-03"/>
            <Time startTime="0800" endTime="1025" day="THU"/>
            <Room roomNumber="1.004" building="FH1">CELUM HS2</Room>
            <Instructor instructorNumber="p22080">Julian-Paul Haslinger</Instructor>
        </Course>
    </DegreeProgramme>
```

The diagram illustrates the structure of the XML document. A large red bracket on the left, labeled 'Interne DTD', covers the entire content of the document except for the DOCTYPE declaration. A red bracket on the right, labeled 'Externe DTD', covers the URL 'http://www.fh-hagenberg.at/dtds/CourseCatalog.dtd'. A red curly brace on the right, labeled 'Redeklaration Parameter Entity', encloses the entire content of the internal DTD declaration, from the '<!ENTITY' start to the '>' end.

Tipps zum Erstellen von DTDs

- Modularisierung von DTDs (externe DTD)
- Deklaration von logisch zusammengehörigen Elementen in voneinander getrennten Abschnitten
- Verwendung von Leerzeichen und Tabulatoren, um DTD zu formatieren und damit lesbarer zu machen
- Verwendung von Kommentaren
- Definition von Parameter-Entities für wiederkehrende Teile von Deklarationen
- Versionierung von DTDs

Zusammenfassung

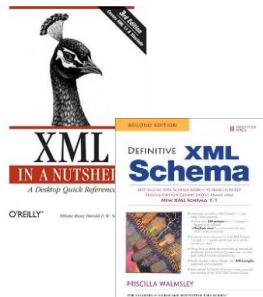
- DTD unterstützt (u.a. mit dem Entity-Konzept) eine kompakte und relativ schnelle Strukturdefinition von XML-(Text-)Dokumenten
- Schwächen
 - Keine XML-Syntax
 - Fehlende Datentypisierung, insbesondere beim Elementinhalt
 - Unzureichende Kardinalitätsangaben
 - Keine Namensräume
 - Eingeschränkte Wiederverwendbarkeit und Erweiterbarkeit
- XML Schema behebt diese Schwächen

Modul 3

XML Namensräume

Julian Haslinger

<?xml?>



Der vorliegende Foliensatz basiert vorwiegend auf:

*Elliotte Rusty Harold, W. Scott Means: XML in a Nutshell: A Desktop Quick Reference, 3rd Edition, O'Reilly, 2005
Priscilla Walmsley: Definitive XML Schema, 2nd Edition, Prentice Hall, 2012*

Inhalt

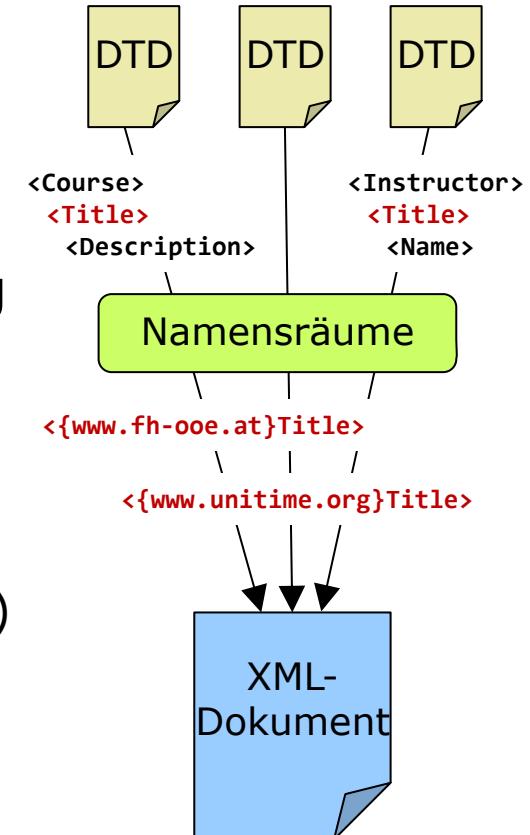
- **Einführung**
- Deklaration
- Namensräume und DTDs

- Anhang: Namensraum-Beispiele

Namensräume 1/7

Einführung

- Vermeidung von Namenskonflikten bei der Verwendung unterschiedlicher Markup-Vokabulare
- XML-Namensraum ist ein Mechanismus, der Elemente und Attribute global eindeutig identifiziert
 - Elemente und Attribute eines Markup-Vokabulars (z.B. MathML) werden einem od. mehreren Namensräumen zugeordnet
 - erlaubt die Wiederverwendung (das Mischen) von Markup-Vokabularen
- Namespaces in XML 1.0 (3rd ed., 2009)
 - <https://www.w3.org/TR/xml-names/>
- Namespaces in XML 1.1 (2nd ed., 2006)
 - <https://www.w3.org/TR/2006/REC-xml-names11-20060816/>

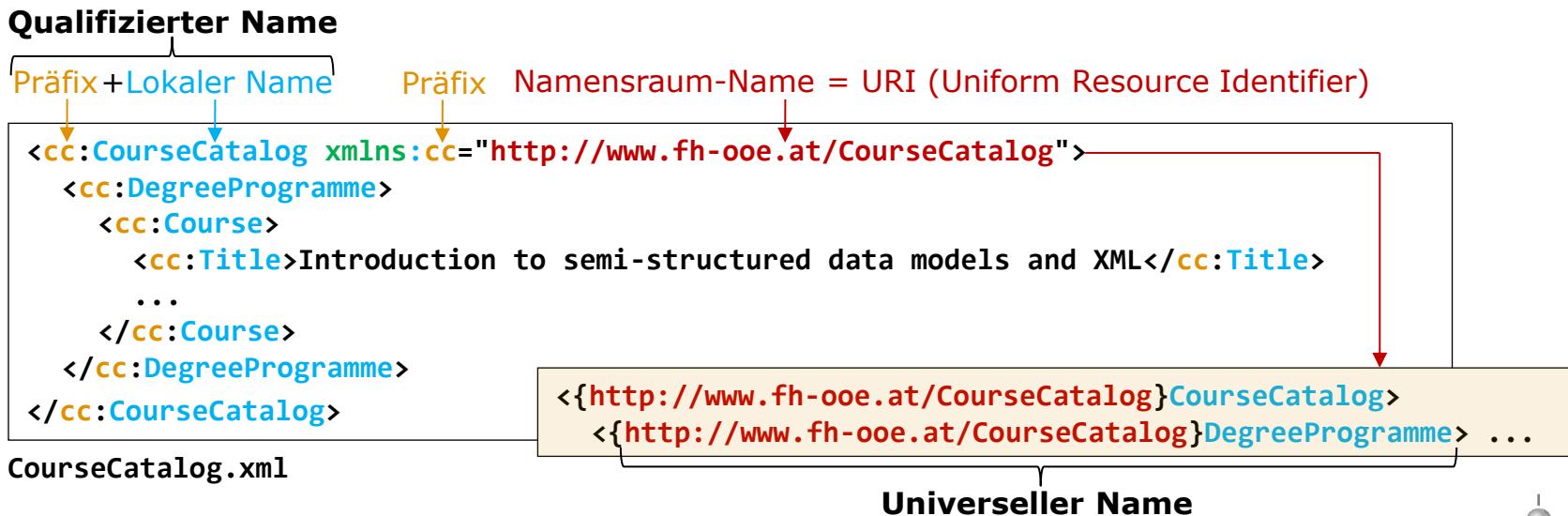


Namensräume 2/7

Namensraumdeklaration mit Präfix

URI = Uniform Resource Identifier
 IRI = Internationalized Resource Identifier

- Namensräume werden über URIs identifiziert (Konvention)
- Jeder Element- od. Attributname ist einem Namensraum zugeordnet
- Zuordnung erfolgt über ein **Präfix**, das mit Hilfe des vordefinierten Attributs **xmlns** an eine **URI** gebunden ist
- Zuordnung ist für das Element gültig, in dem sie getroffen wurde, und für alle Subelemente, bis eine Redefinition des Präfixes erfolgt



Namensräume 3/7

Namensraumdeklaration mit Default-Namensraum

- Default-Namensraum wird über das vordefinierte Attribut **xmlns** - ohne Präfix - deklariert
- Nichtqualifizierte Subelementnamen unterliegen automatisch dem Default-Namensraum, **Attributnamen nicht**
- Default-Namensraum kann in Subelementen überschrieben werden

CourseCatalog.xml

```
<CourseCatalog xmlns="http://www.fh-ooe.at/CourseCatalog">
    <DegreeProgramme code="0307" name="Software Engineering">
        <Course id="cID_8314" semesterHours="1">
            <Title>Introduction to semi-structured data models and XML</Title>
            ...
        </Course>
    </DegreeProgramme>
</CourseCatalog>
```

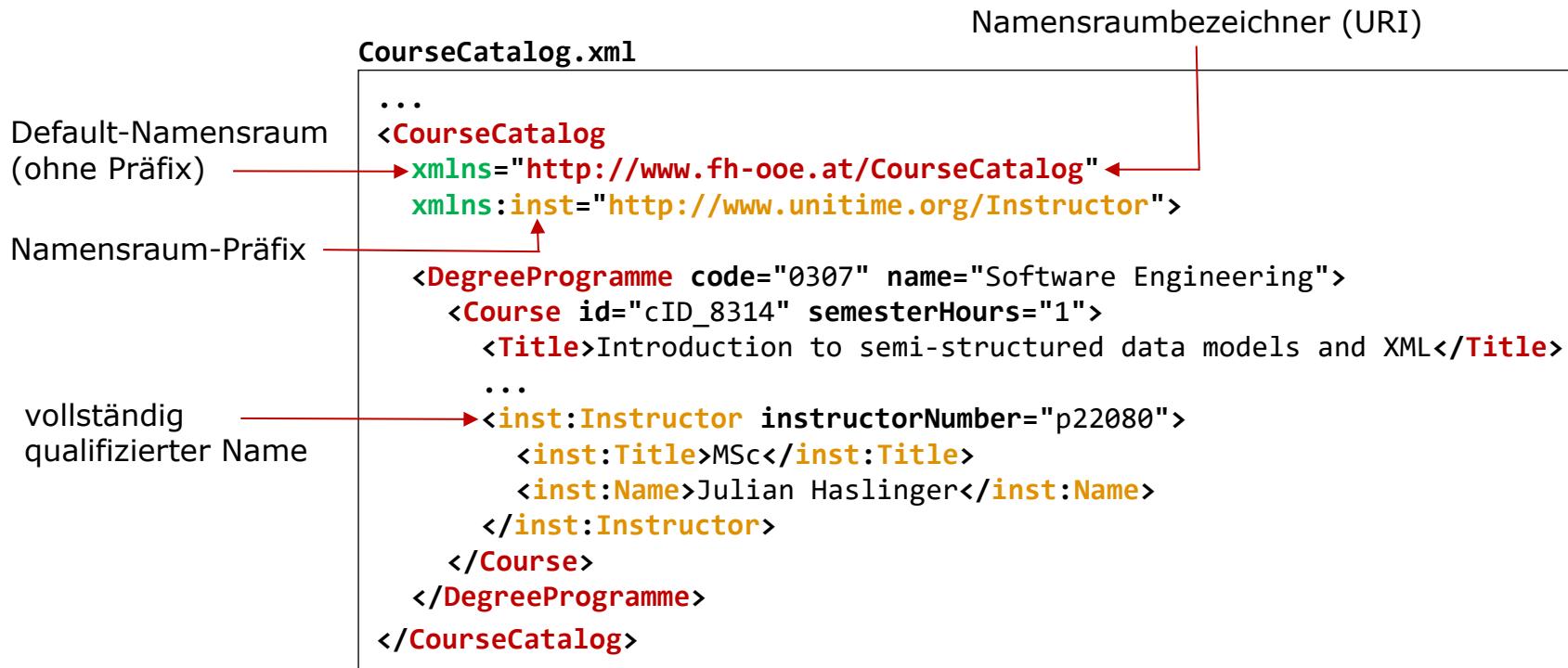
Namensräume 4/7

Regeln

- Elementnamen mit Präfix (qualifizierter Name) gehören zu dem Namensraum, den das Präfix definiert.
- Elementnamen ohne Präfix, für die die Deklaration eines Default-Namensraum gültig ist, gehören zu dem dort als Default deklarierten Namensraum.
- Elementnamen ohne Präfix, für die keine Deklaration eines Default-Namensraums gültig ist, unterliegen einem leeren Namensraum.
- Attributnamen mit Präfix (qualifizierter Name) gehören zu dem Namensraum, den das Präfix definiert.
 - Ansonsten gehören Attribute zu keinem Namensraum (Annahme, dass Attribut nur in Verbindung mit dem jeweiligen Element sinnvoll ist)

Namensräume 5/7

Deklaration und Verwendung



- Der Namensraum des Elements `<inst:Instructor>` ist `http://www.unitime.org/Instructor`
- Der Namensraum des Elements `<CourseCatalog>` ist der Default-Namensraum `http://www.fh-ooe.at/CourseCatalog`
- Alle Attribute unterliegen dem leeren Namensraum

Namensräume 6/7

und DTDs

- Namensräume sind prinzipiell **unabhängig von DTDs**
 - können sowohl in Dokumenten mit als auch ohne DTD verwendet werden
- Jedoch müssen alle im XML-Dokument qualifizierten Elemente und Attribute auch entsprechend in der DTD deklariert werden
 - <**cc:CourseCatalog**> ... <!ELEMENT **cc:CourseCatalog** (...)>
 - <**cc:Title**> ... <!ELEMENT **cc:Title** (#PCDATA)>
- In der DTD kann ein Default-Namensraum (als Attribut) spezifiziert werden
 - <!ATTLIST **cc:CourseCatalog** xmlns
 CDATA #FIXED "http://www.fh-ooe.at/CourseCatalog">

Namensräume 7/7

Beispiele für Namensräume

- SVG
 - <http://www.w3.org/2000/svg>
- MathML
 - <http://www.w3.org/1998/Math/MathML>
 - Mathematical Markup Language zur Beschreibung von mathematischen Sachverhalten
- XHTML
 - <http://www.w3.org/1999/xhtml>
- SMIL
 - <http://www.w3.org/TR/REC-smil>
 - Synchronized Multimedia Integration Language zur Beschreibung von Multimedia-Präsentationen/-Animationen
- XSL
 - <http://www.w3.org/1999/XSL/Transform>
 - <http://www.w3.org/1999/XSL/Format>
- ...

Anhang

Namensräume

Beispiele

Namensräume

Beispiele

- Die Deklaration eines Namensraumes ist auf das Element (und dessen Nachkommen) beschränkt, indem er deklariert wurde.

```
<aaa xmlns:lower="http://fh-ooe.at/ssd4/lower">
  <lower:BBB xmlns:lower="http://fh-ooe.at/ssd4/upper">
    <lower:x11/>
    <sss xmlns:lower="http://fh-ooe.at/ssd4/other">
      <lower:x111/>
    </sss>
  </lower:BBB>
  <lower:x111/>
</aaa>
```

Namensräume

Beispiele

- Attribute gehören selbst dann nicht zu einem Namensraum, wenn ein Standard-Namensraum für das entsprechende Element definiert ist.

```
<aaa
    xmlns="http://fh-ooe.at/ssd4/lower"
    xmlns:upper="http://fh-ooe.at/ssd4/upper"
    xmlns:number="http://fh-ooe.at/ssd4/number">
    <bbb zz="11">
        <ccc WW="22" xmlns="http://fh-ooe.at/ssd4/upper"/>
    </bbb>
    <upper:BBB sss="**" number:S111="???" />
    <number:x111/>
</aaa>
```

Namensräume

Beispiele

- Selbst wenn Standard-Namensräume benutzt werden, können Namensräume für ausgewählte Elemente explizit gesetzt werden.

```
<aaa
    xmlns:upper="http://fh-ooe.at/ssd4/upper"
    xmlns:number="http://fh-ooe.at/ssd4/number">

    <bbb xmlns="http://fh-ooe.at/ssd4/lower">
        <ccc/>
        <upper:www/>
        <number:c456/>
    </bbb>
    <BBB xmlns="http://fh-ooe.at/ssd4/upper">
        <CCC/>
        <upper:www/>
        <number:c456/>
    </BBB>
</aaa>
```

Namensräume

Beispiele

- Standard-Namensräume können wieder "un"-deklariert werden, indem ein Leerstring als Wert für das xmlns-Attribut gebraucht wird.

```
<aaa xmlns="http://fh-ooe.at/ssd4/lower">
  <bbb>
    <ccc xmlns="">
      <ddd></ddd>
    </ccc>
  </bbb>
</aaa>
```

Modul 4

XPath

Julian Haslinger



Der vorliegende Foliensatz basiert vorwiegend auf:

Kay, M.: *XPath 2.0 Programmer's Reference (3rd ed.)*, Wiley, 2004.

Lehner, W., Schöning, H.: *XQuery*, dpunkt.verlag, 2004.

Simpson, J.: *XPath and XPointer*, O'Reilly, 2002.

Becher, M.: *XML (1. Auflage)*, Springer Campus, 2009.

Harold, R., Means, S.: *XML in a Nutshell: A Desktop Quick Reference (3rd ed.)*, O'Reilly Media, 2009.

Inhalt

- **Einführung**
- Datenmodell
- Pfadausdrücke
- Erweiterte Ausdrücke
- Funktionen und Operatoren
- Zusammenfassung

XPath-Versionen (W3C-Standards):

- XPath 1.0, Nov. 1999, ~28 PDF-Seiten
- **XPath 2.0**, Jan. 2007, ~78 PDF-Seiten
- XPath 3.0, Apr. 2014, ~100 PDF-Seiten
- XPath 3.1, März 2017, ~171 PDF-Seiten

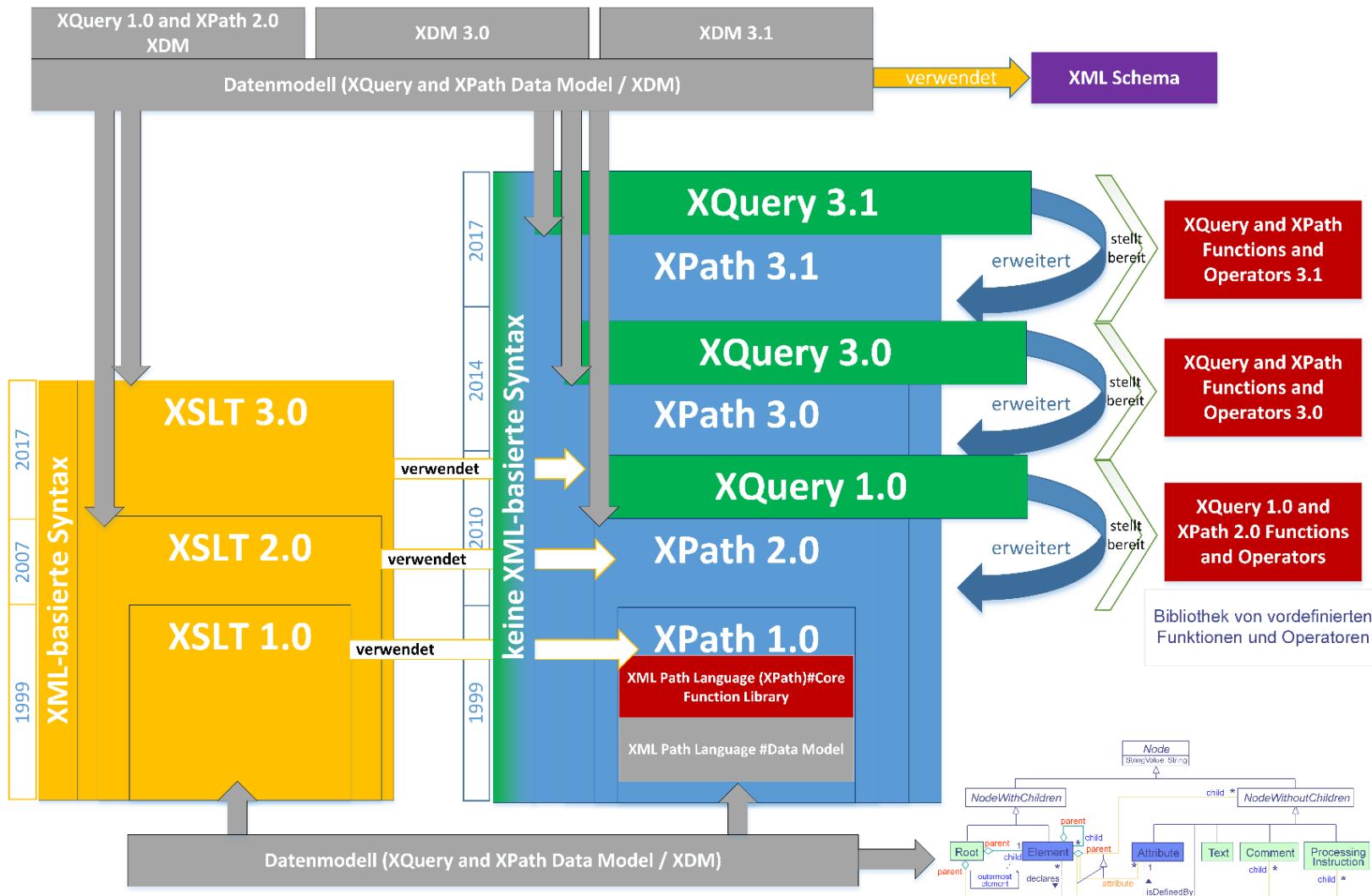
XPath

XML Path Language

- Einführung
 - Selektion (Lokalisieren) von Knoten und Inhalten (Dokumentteilen)
 - in vielen XML-Standards verwendet: XQuery, XSLT, **XML Schema**, etc.
 - keine XML-Syntax - eigener (Pfadbeschreibungs-) Standard
 - Selektionskriterien: Element- und Attributnamen, Inhalt, Typ, etc.
- Grundprinzip der Verarbeitung
 - Navigation in einer Baumstruktur, ähnlich zur Navigation in einem Dateisystem
 - Ausgangspunkt ist ein **Kontext** in Form eines Baumknotens
 - Kontext wird von einem XPath-Ausdruck vorgegeben
 - **Navigation** und **Filter** modifizieren den Kontext
 - Ergebnis eines XPath-Ausdrucks = **zuletzt berechneter Kontext** (= Knotensequenz)
 - Erzeugen und Ändern von Knoten wird nicht unterstützt (*read only!*)
- W3C-Recommendations:
 - XPath 1.0, Nov. 1999
 - **XPath 2.0**, Jan. 2007
 - XPath 3.0, April 2014
 - XPath 3.1, Jänner 2017

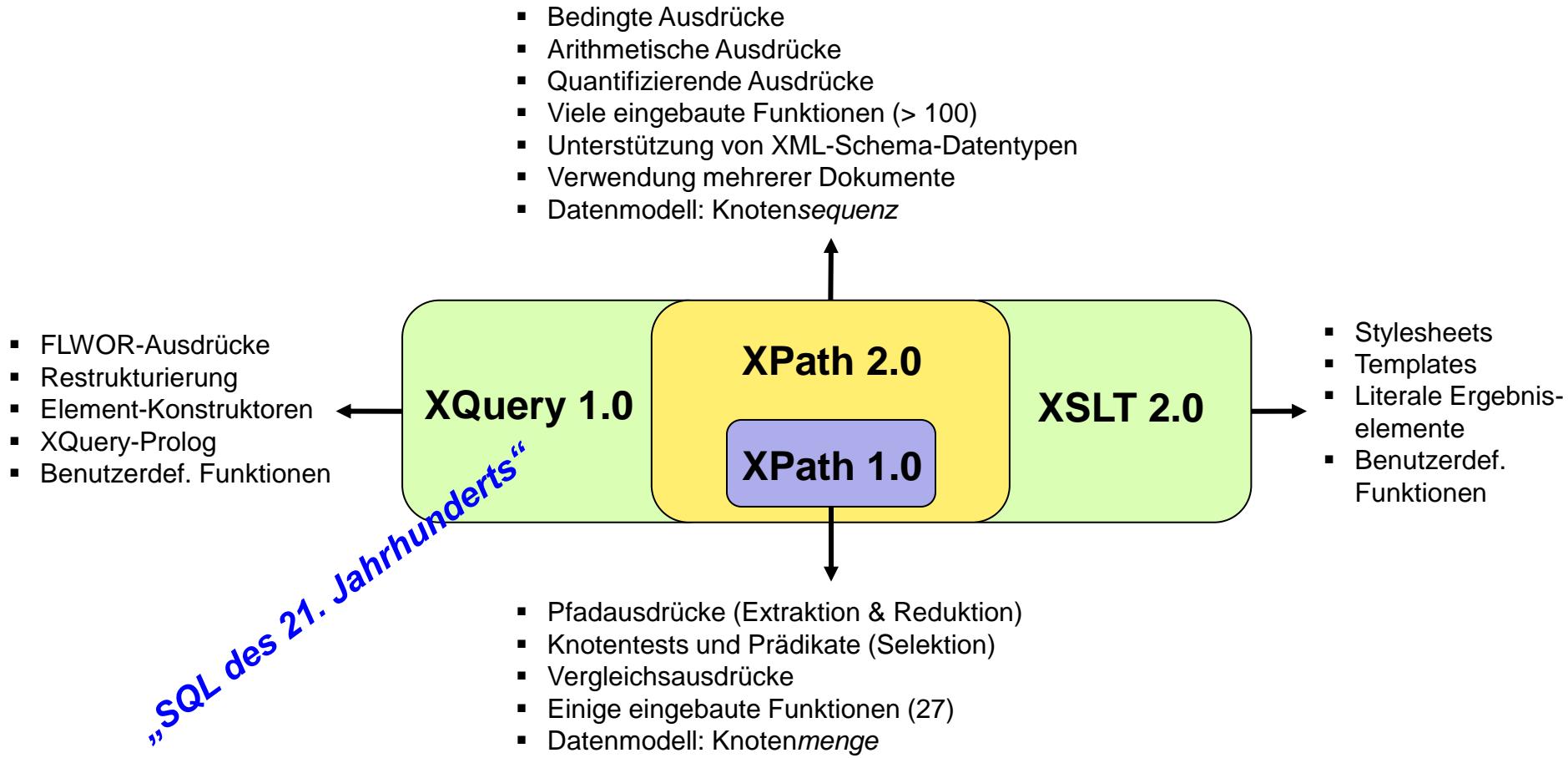
XPath

Versionen



XPath

Zusammenhang bis XPath 2.0



XPath

Sprachumfang (Erweiterungen in XPath 3.0 / 3.1)

XPath 3.1

- Neue Datentypen map und array

XPath 3.0

- Anonyme Funktionen
- Dynamische Funktions-Aufrufe
- Vereinigungs-Typen (Union)
- Namespace-Literale
- String-Konkatenations-Operator
- Mapping-Operator

XPath 2.0

- Bedingte Ausdrücke
- Arithmetische Ausdrücke
- Quantifizierende Ausdrücke
- Viele eingebaute Funktionen (> 100)
- Unterstützung von XML-Schema-Datentypen
- Verwendung mehrerer Dokumente
- Datenmodell: Knotensequenz

XPath 1.0

- Pfadausdrücke (Extraktion & Reduktion)
- Knotentests und Prädikate (Selektion)
- Vergleichsausdrücke
- Einige eingebaute Funktionen (27)
- Datenmodell: Knotenmenge

→ Mehr zu XPath 3.0/3.1 in eigenem Foliensatz

XPath 3.1

XPath 3.0

XPath 2.0

XPath 1.0

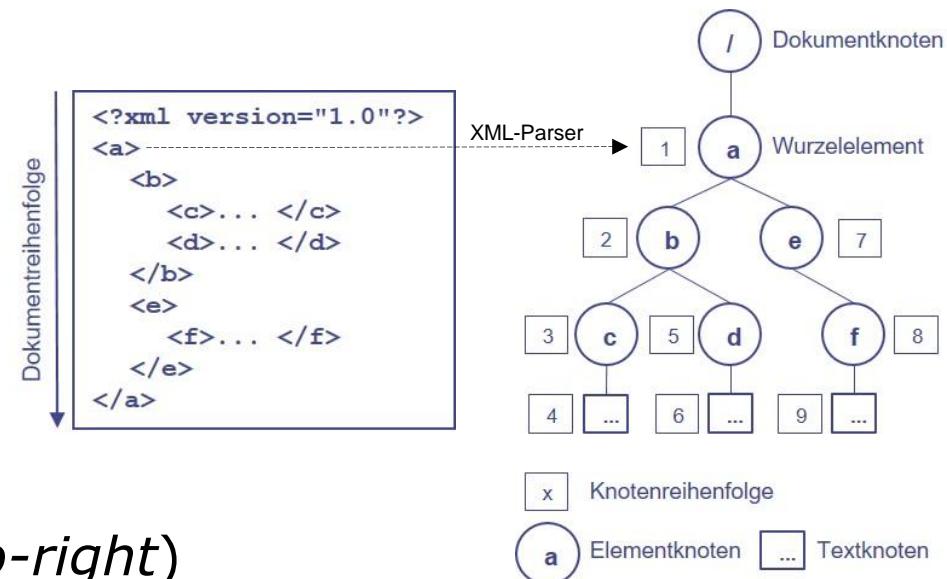
Inhalt

- Einführung
- **Datenmodell**
 - Knoten
 - atomarer Wert
 - Sequenz
- Pfadausdruck
- Erweiterte Ausdrücke
- Funktionen und Operatoren
- Zusammenfassung

XPath

Datenmodell

- XPath basiert auf dem **XQuery and XPath Data Model (XDM)**
 - siehe www.w3.org/TR/xpath-datamodel/
- XDM ist das Datenmodell für XPath, XQuery und XSLT
- Basiskomponenten von XDM sind:
 - Knoten
 - atomare Werte
 - Sequenzen
 - (Funktionen)
- XML-Dokument wird als Baum dargestellt
- Knoten im Baum sind geordnet (*top-down, left-to-right*)
- Somit: Navigation im Baum möglich

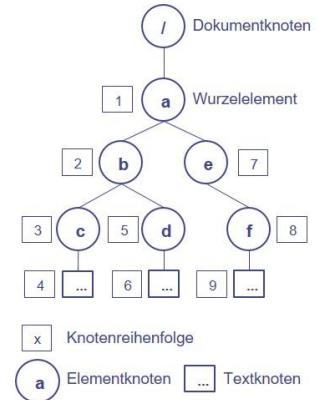


XPath

Datenmodell – Knoten

■ 7 Knotenarten im Dokumentenbaum

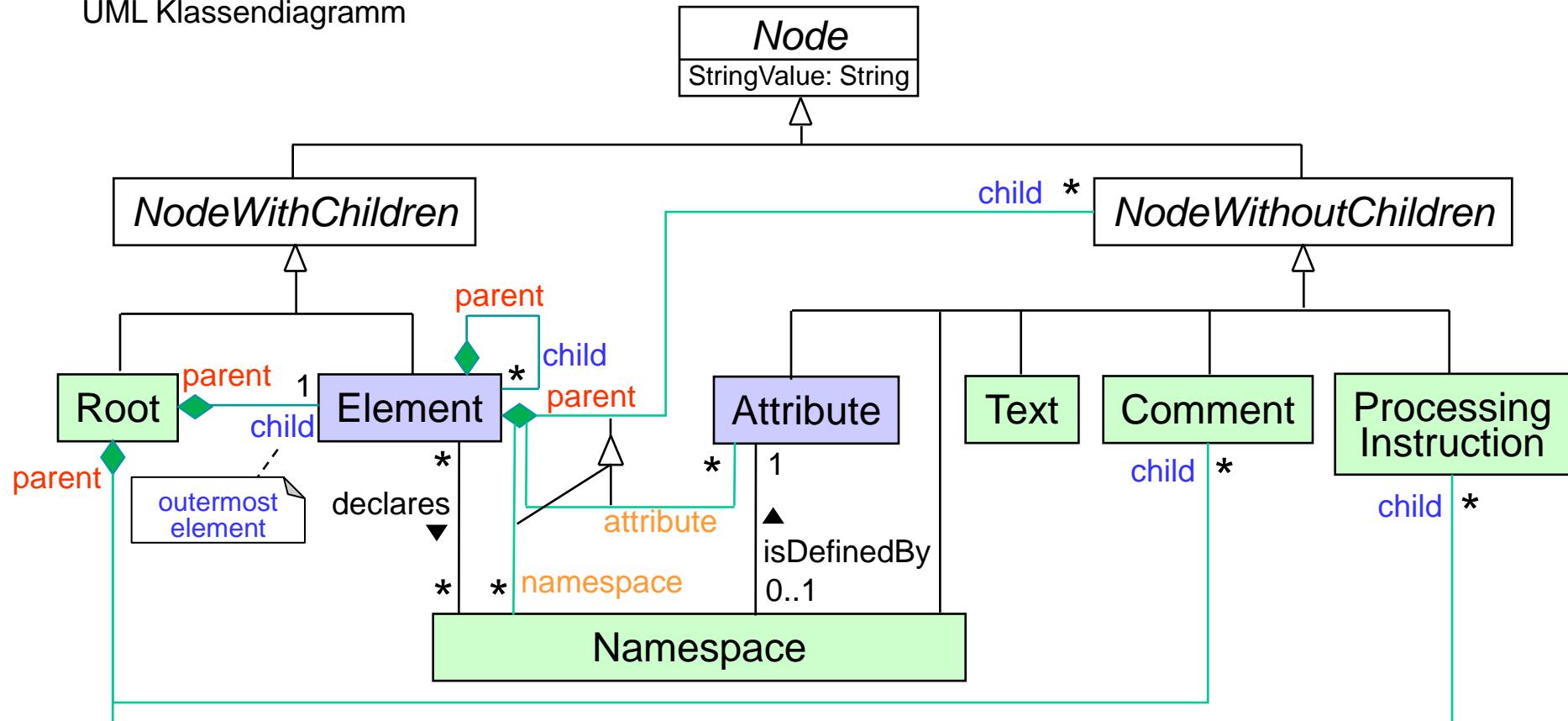
- **Dokumentknoten** bzw. Wurzelknoten
 - umfasst das gesamte XML-Dokument
- **Elementknoten**
 - für jedes Element im XML-Dokument
- **Attributknoten**
 - zu entsprechenden Elementknoten zugeordnet
- **Namensraumknoten**
 - für alle NS-Präfixe und einen etwaigen Default-NS; zu entsprechenden Elementknoten zugeordnet
- **Verarbeitungsanweisungsknoten**
- **Kommentarknoten**
 - enthalten die in <!-- ... --> enthaltene Zeichenkette
- **Textknoten**
 - enthalten ausschließlich Zeichendaten



XPath

Knotentypen

UML Klassendiagramm



Hinweis: Root ist nicht die Elementwurzel, sondern repräsentiert das gesamte XML-Dokument ("Dokumentwurzel")

XPath

Datenmodell – Knoteninformationen

Node
StringValue: String

■ Knoten liefern folgende Informationen

- **Knotename**: qualifizierter Name bei Element- und Attributknoten, Präfix bei Namensknoten
- **Elternknoten**: jeder Knoten außer dem Dokumentknoten hat genau einen Elternknoten
- **Kindknoten**: nur bei Dokumentknoten und Elementknoten
- **Attribute**: nur bei Elementknoten
- **Namensräume**: nur bei Elementknoten, die Namensraumknoten enthalten die aktuelle Präfix-Bindungen
- **Typ**: Typinformationen zum Knoten
- **String-Wert**: siehe nächste Folie

XPath

Datenmodell – String-Wert eines Knotens

Node
<code>StringValue: String</code>

- String-Wert (**StringValue**) liefert bei
 - **Dokumentknoten**: Konkatenation aller Textknotennachfolger im Dokument
 - **Elementknoten**: Konkatenation aller Textknoten unterhalb eines Elementknotens
 - **Attributknoten**: normalisierter Attributwert
 - **Namensraumknoten**: Namensraum-URI
 - **Kommentarknoten**: Inhalt des Kommentars, ohne `<!-- ... -->`
 - **Textknoten**: Zeicheninhalt

String-Wert für Elementknoten CourseCatalog

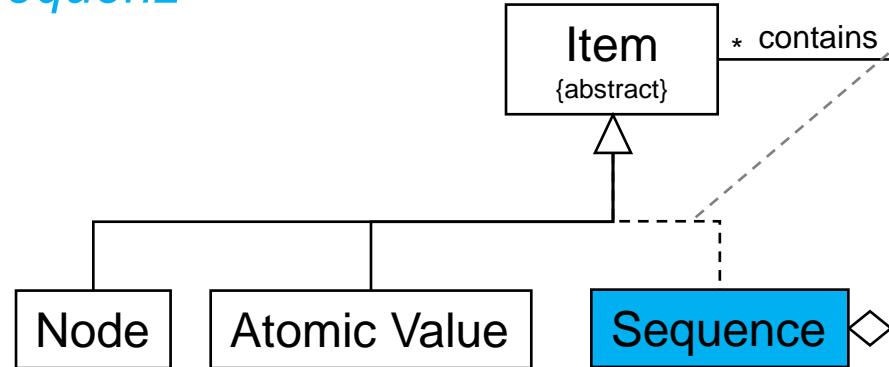
<u>CourseCatalog:Element</u>
<code>StringValue := '...'</code>

XPath

Datenmodell – Sequenz

„Alles ist eine Sequenz“

Hinweis: Syntaktisch korrekt; geschachtelte Sequenzen werden automatisch entschachtelt.



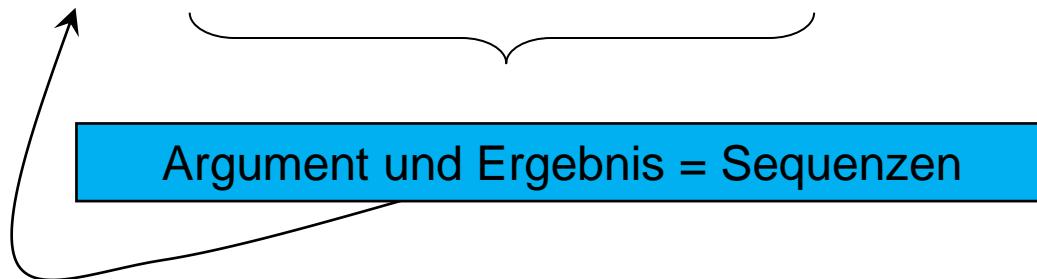
- Ergebnis jedes XPath-Ausdrucks ist eine **Sequenz**
- Sequenz besteht aus beliebig vielen **Items**
- Item ist entweder ein **Knoten** oder ein **atomarer Wert** (string, boolean, decimal, ...)
- **Duplikate** sind in Sequenzen erlaubt
- Sequenzen können **nicht verschachtelt** sein bzw. werden automatisch entschachtelt

XPath

Datenmodell – Sequenz

- Konsequenz aus „*Alles ist eine Sequenz*“
 - Jeder Operand eines Ausdrucks ist eine Sequenz
 - Jedes Ergebnis eines Ausdrucks ist eine Sequenz
- Zwei Eigenschaften
 - Abgeschlossen
 - Jede Operation erzeugt als Ergebnis wieder eine Sequenz
 - Ermöglicht Komposition
 - Ausdrücke können beliebig geschachtelt werden
- Beispiel

- `count(//Course/@semesterHours)`



XPath

Beispiel-XML-Dokument

CourseCatalog.xml

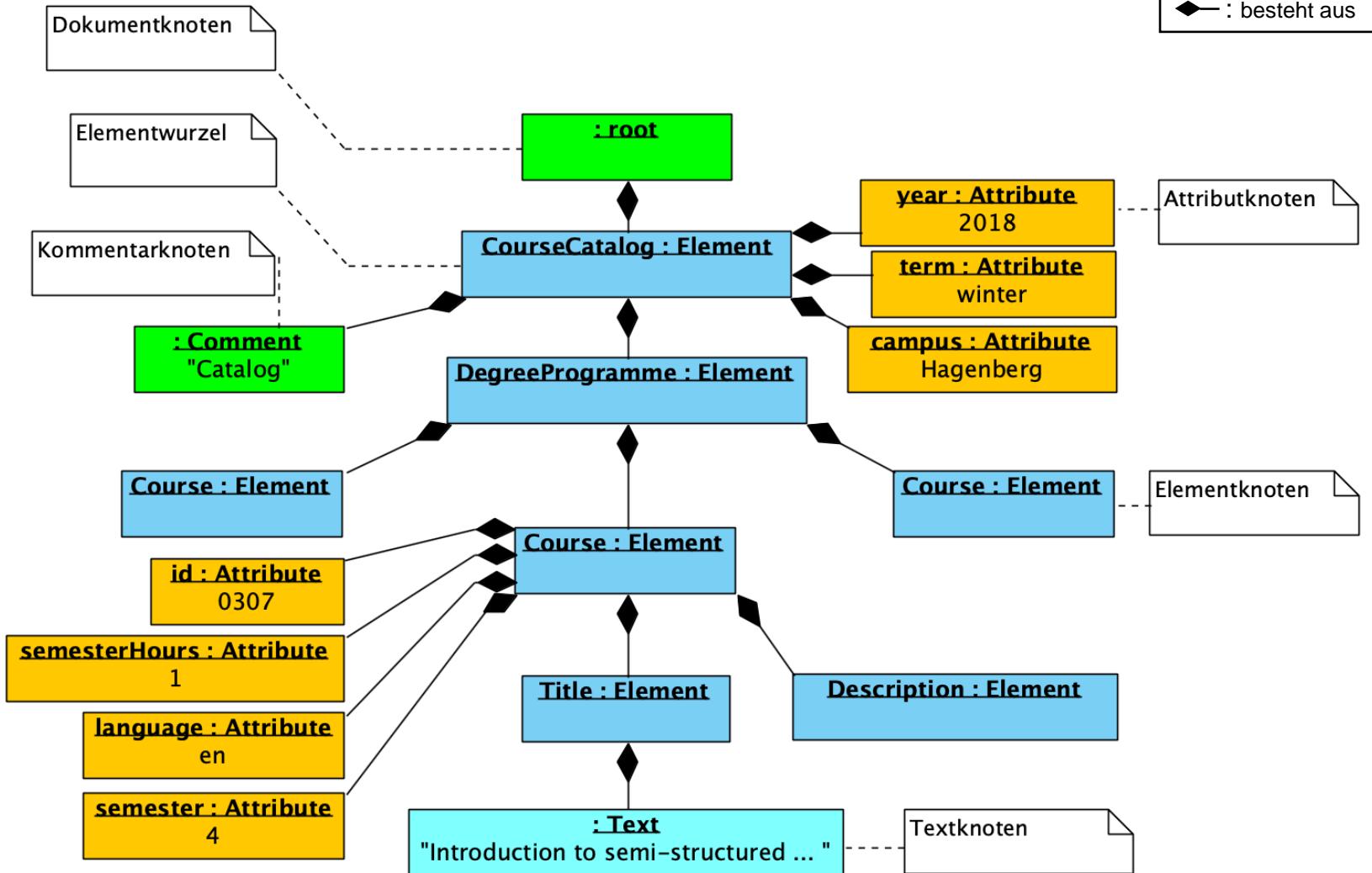
```
<?xml version="1.0" encoding="UTF-8"?>
<CourseCatalog year="2019" term="summer" campus="Hagenberg">
    <!-- Software Engineering -->
    <DegreeProgramme code="0307" name="Software Engineering" abbreviation="SE">
        <Course id="cID_8314" semesterHours="1" language="en-US" semester="4">
            <Title>Introduction to semi-structured data models and XML</Title>
            <Description>Introduction of skills related to XML.<Content>Includes DTD, Schema,
                XPath, XQuery, XSLT, JSON</Content><Exam>Final Exam required.</Exam>
                Participation without any previous knowledge.</Description>
            <Credit formatType="ECTS">1</Credit>
            <CourseType type="Lecture"/>
            <Date startDate="02-28" endDate="05-03"/>
            <Time startTime="0800" endTime="1025" day="THU"/>
            <Room roomNumber="1.004" building="FH1">CELUM HS2</Room>
            <Instructor instructorNumber="p22080">Julian Haslinger</Instructor>
        </Course>
        <Course> ... </Course>
    </DegreeProgramme>
    <DegreeProgramme code="0456" name="Communication and Knowledge Media" abbreviation="CKM">
        ...
    </DegreeProgramme>
</CourseCatalog>
```

Knotenname: Knotentyp
Knotenwert

◀ : besteht aus

XPath

Datenmodell – Beispiel CourseCatalog.xml



XPath

Datenmodell – Sequenz

■ Beispiele

- Geordnet
 - (1, 2, 3, 4) **ist verschieden von** (4, 3, 2, 1)
- Duplikate erlaubt
 - (1, 2, 3, 4) **ist verschieden von** (1, 1, 2, 3, 4)
- Heterogene Einträge (Items) möglich
 - (1, 2, 3, "Hallo")
 - (1, 2, "Hallo", `function($a as xs:double, $b as xs:double) as xs:double {$a * $b}`, "Test") (ab XPath 3.0)
- Keine Schachtelung möglich (flach)
 - (1,2, (1,2,3,4), 3) **ist äquivalent zu** (1, 2, 1, 2, 3, 4, 3)

Inhalt

- Einführung
- Datenmodell
- Pfadausdruck
 - Lokalisierungsschritt
 - Achsen
 - Knotentest
 - Filter (Prädikat)
- Erweiterte Ausdrücke
- Funktionen und Operatoren
- Zusammenfassung

XPath

Pfadausdruck

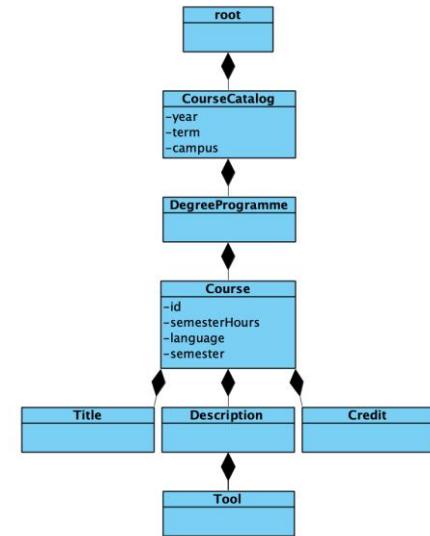
- Pfadausdruck dient zur Lokalisierung und dem Zugriff auf Bestandteile eines XML-Dokuments (als Instanz des Datenmodells beschrieben)
 - Absoluter Pfad `/CourseCatalog/DegreeProgramme/Course`
 - Auswertung beginnt bei der Dokumentwurzel ("`/`") UNABHÄNGIG vom aktuellen Kontext
 - Relativer Pfad `DegreeProgramme/Course`
 - Auswertung beginnt beim aktuellen Kontextknoten (z.B. durch vorangegangenen Lokalisierungsschritt bestimmt)
- Pfadausdruck besteht aus
 - **Lokalisierungsschritten** und **Achsen**: geben die Richtung an, in der die zu selektierenden Knoten gesucht werden
 - **Knotentest**: filtert die durch die Lokalisierungsschritte selektierten Knoten
 - **Prädikate** (optional): kann die Knotenmenge weiter filtern

XPath

Pfadausdruck – Lokalisierungsschritt

■ Lokalisierungsschritt selektiert eine Knotenmenge

- der nächste Schritt operiert auf dieser Knotenmenge
 - (jeder Knoten ist dabei einmal der Kontextknoten)
- pro Schritt kann die Knotenmenge wachsen oder schrumpfen
- Abarbeitung zusammengesetzter Schritte erfolgt "von links"



■ Hierarchieoperatoren / und //

- **/**
Dokumentwurzel (*root node*)
- **//Course**
alle Course-Elemente, beliebig tief im Kontext (ausgehend von Dokumentwurzel)
- **//Course/Title**
alle Title-Subelemente von Course-Elementen, beliebig tief im Kontext

XPath

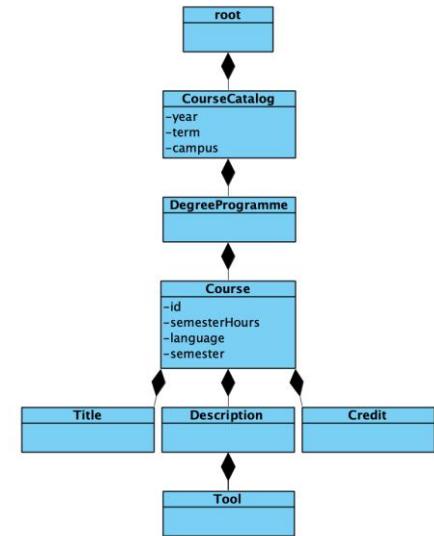
Pfadausdruck – Lokalisierungsschritt

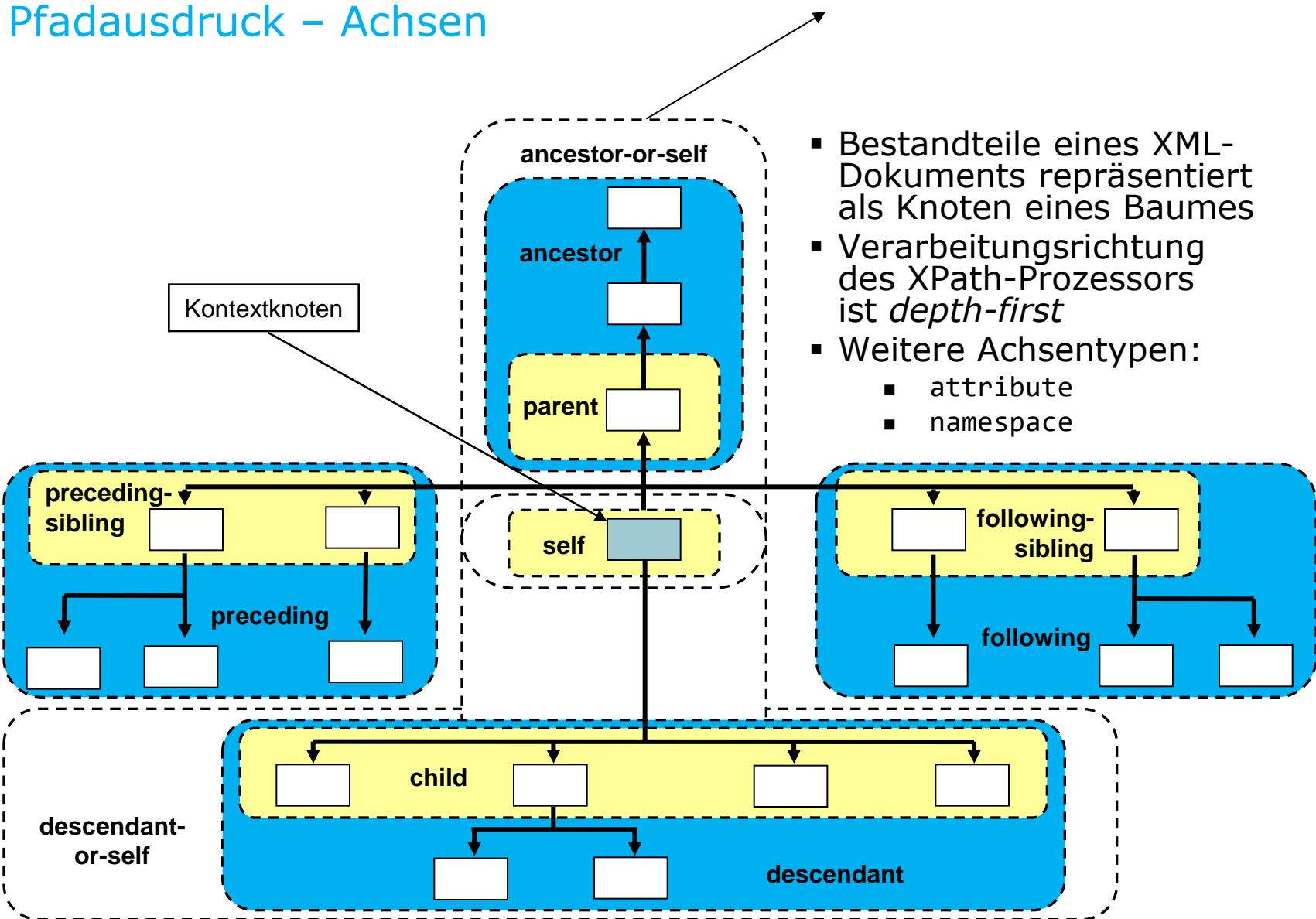
- Zugriff auf beliebige Elemente *

 - `/*`
Elementwurzel
 - `//*`
alle Elemente, inklusive Elementwurzel
 - `//DegreeProgramme/*/*`
alle Credit-Elemente, sofern sie Enkel des DegreeProgramme-Elements sind

- Zugriff auf Attribute @

 - `//@*`
alle Attribute aller Elemente
 - `//Course/@semester`
alle semester-Attribute aller Course-Elemente





XPath

Pfadausdruck – Achsen

- Verarbeitungsrichtung des XPath-Prozessors ist *depth-first*
- Dokumentausschnitt und Aufbau der Ergebnismenge bei unterschiedlichen Ausdrücken:

```
<CourseCatalog>
  <DegreeProgramme>
    <Course>
      <Title/>
      <Description><Content/><Exam/></Description>
      <Credit/>
      <CourseType/>
      <Date/>
      <Time/>
      <Room/>
      <Instructor/>
    </Course>
  </DegreeProgramme>
</CourseCatalog>
```

Kontext-Knoten im letzten Schritt: *Course*

Ergebnis 1:

- Alle **direkten** Kind-Elemente (*/**) von *Course*
- (*Content* und *Exam* sind keine Kind-Elemente von *Course*).

Ergebnis 2:

- Ergebnis: Alle Kind-Elemente von *Course*
- (inkl. *Content* und *Exam* als eigene Knoten)

```
/CourseCatalog/DegreeProgramme/Course/*
<Title/><Description/><Credit/>
<CourseType/><Date/><Time/><Room/><Instructor/>
```

```
/CourseCatalog/DegreeProgramme/Course//*
<Title/><Description/><Content/><Exam/><Credit/>
<CourseType/><Date/><Time/><Room/><Instructor/>
```

XPath

Pfadausdruck – Lokalisierungsschritte und Achsen

- Achsenname - Navigation über Achsenbezeichnung

Langform

`child::element-name`

`attribute::attributename`

`/descendant-or-self::node()/child::element-name`

`self::node()`

`parent::node()`

Kurzform

`element-name`

`@attributename`

`//element-name`

`.`

`..`

Beispiele

Attributwerte aller
DegreeProgramme-
Elemente

`/child::CourseCatalog/child::DegreeProgramme/attribute::name`
(kürzer) `/CourseCatalog/DegreeProgramme/@name`

Alle Course-Elemente,
unabhängig von der
Position im Baum

`/descendant-or-self::node()/child::Course`
(kürzer) `//Course`

XPath

Pfadausdruck – Knotentest

- Knotentest gibt ein Filterkriterium für die Auswahl von Knoten an
- Knotentest erfolgt durch
 - **Namenstest**: Angabe des Knotennamens
 - `/CourseCatalog/DegreeProgramme/Course`
 - **Wildcard-Test** mittels *
 - `//*` (: gibt alle Kind-Elementknoten zurück :)
 - `//@*` (: gibt alle Attributknoten zurück :)
 - **Kindtest**: Überprüfen des Knotentyps
 - `comment()`: wählt alle Kommentarknoten aus
z.B. `/CourseCatalog/comment()`
 - `attribute()`: wählt alle Attributknoten
 - `node()`: wählt alle Knoten unabhängig von ihrem Typ
 - `text()`: wählt alle Textknoten aus
 - ...

XPath

Pfadausdruck – Filter und Prädikate

- Lokalisierungsschritt kann Filter (Prädikate) enthalten
 - Auswertung der Filterbedingung für alle im Knotentest ausgewählten Knoten
 - Filter sind XPath-Ausdrücke in eckigen Klammern
- Typische Anwendungen von Filtern sind: Finde die Knoten, bei denen
 - ein bestimmter Attributwert mit einer Zeichenkette übereinstimmt
 - ein Elementinhalt mit einer Zeichenkette übereinstimmt,
 - ein Element ein bestimmtes Kindelement enthält oder
 - ein Knoten eine bestimmte Position hat.
- Zur Formulierung der Filter können Operatoren verwendet werden
 - Arithmetische Operatoren: +, -, *, div, mod
 - Logische Operatoren: and, or, not
 - Vergleichsoperatoren: =, !=, <, <=, >, >=, ...

XPath

Filter – Existenztest

- **//Course[Title]**
 - alle Course-Elemente, die ein Title-Element enthalten

- **//Course[Title]/Description[Tool]**
 - alle Description-Elemente mit Tool-Element in Course-Elementen, die ein Title-Element enthalten

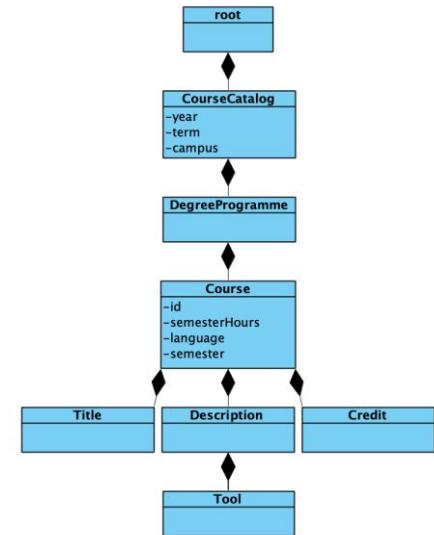
- **//DegreeProgramme[Course/Title]**
 - alle DegreeProgramme-Elemente, die ein Course-Element enthalten, das ein Title-Element als Kind hat

- **//Course[Title and Description]**
 - alle Course-Elemente mit Title- und Description-Subelementen

- **//Course[starts-with>Title, "Introduction to"]**
 - alle Course-Elemente, deren Title-Element mit „Introduction to“ beginnt.

- **//Course[@id = "cID7540"]**
 - alle Course-Elemente, deren Attribut id den Wert cID7540 hat

- **//DegreeProgramme[Course[@id = "cID7540"]]**
 - alle DegreeProgramme-Elemente, die ein Course-Element enthalten, deren Attribut id den Wert cID7540 hat

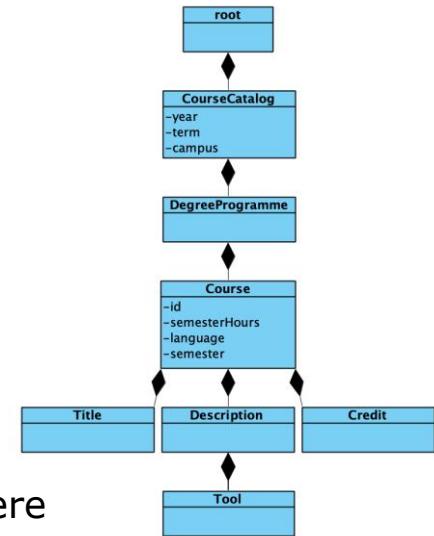


XPath

Filter – Funktionen und Filterlisten

- XPath stellt Bibliothek von Funktionen bereit, die in Filtern verwendet werden können
 - Beispiel: Filter über Kontextposition des Knotens

<code>//Credit[1]</code>	alle ersten Credit-Elemente von den entsprechenden Eltern-Knoten (auch mehrere möglich)
oder	
<code>//Credit[position()=1]</code>	
<code>(//Credit)[1]</code>	das erste Credit-Element im gesamten Dokument
<code>(//Credit)[position()=1]</code>	
<code>//Course[Last()]</code>	alle letzten Credit-Elemente von den entsprechenden Eltern-Knoten (auch mehrere möglich)
<code>(//Credit)[Last()]</code>	das letzte Credit-Element im gesamten Dokument



Auswertung von Filterlisten

- `//Course[starts-with(@id, "cID")][Last()]/Title`
- alle Course-Elemente, bei denen die id mit "cID" beginnt, aus dieser Knotenmenge wird der letzte Title-Knoten zurückgegeben. [Reihenfolge der Filter ist von Bedeutung – Auswertung von links nach rechts]

Inhalt

- Einführung
- Datenmodell
- Pfadausdruck
- **Erweiterte Ausdrücke**
 - Vergleichsausdruck
 - Schleifenausdruck
 - Konditionaler Ausdruck
 - Quantifizierender Ausdruck
- Funktionen und Operatoren
- Zusammenfassung

XPath

Vergleichsausdruck

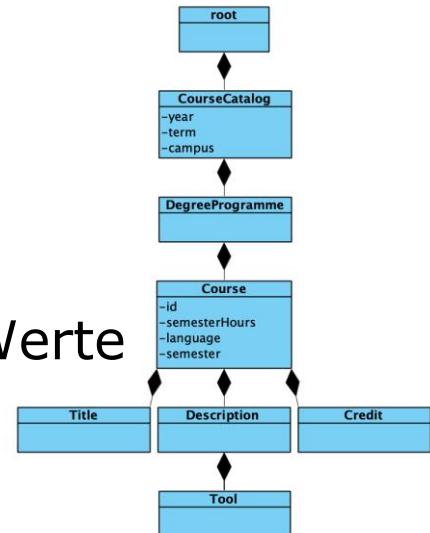
- 3 Arten von Vergleichsausdrücken
 - Wertevergleich: `eq`, `ne`, `lt`, `le`, `gt` und `ge`
 - Allgemeiner Vergleich: `=`, `!=`, `<`, `<=`, `>` und `>=`
 - Knotenvergleich: `is`, `<<` und `>>`

XPath

Vergleichsausdruck – Wertevergleich

- Operatoren: **eq**, **ne**, **lt**, **le**, **gt** und **ge**
- wird verwendet, um zwei einzelne (atomare) Werte gleichen Typs zu vergleichen
- bei Sequenzen mit mehr als einem Item wird ein Fehler erzeugt

- Beispiele:
 - `(//Course)[1]/Credit/@formatType eq "ECTS"`
→ true
 - `//Course/Credit/@formatType eq "ECTS"`
→ Fehler, da die @formatType-Sequenz mehrere Items enthält



XPath

Vergleichsausdruck – Allgemeiner Vergleich

- Operatoren: `=`, `!=`, `<`, `<=`, `>` und `>=`
- Vergleich von Sequenzen mit beliebig vielen Items
- Auswertung von Ausdrücken der Form `x <comp> y`:
 - Wert jedes Items aus `x` wird mit jedem Item-Wert aus `y` verglichen (entsprechend `<comp>`)
 - liefert einer der Vergleiche `true`, so liefert der gesamte Ausdruck `true`
- Beispiele:
 - `//*` $=$ "1"
→ `true`, da einer der Textknoten im Dokumentenbaum den Wert '1' enthält (ECTS)).
 - `//@*` $=$ "ECTS"
→ `true`, da ein Attributknoten den Wert "ECTS" enthält

XPath

Vergleichsausdruck – Knotenvergleich

- Operatoren: **is**, **<<** und **>>**
- Vergleich von zwei Knoten x **<comp>** y
- **is** liefert true, wenn x derselbe Knoten wie y ist
- das Ergebnis von **<<** und **>>** wird von der Dokumentenreihenfolge bestimmt
 - **<<** liefert true, wenn x ein Vorgänger von y ist
 - **>>** liefert true, wenn x ein Nachfolger von y ist
- Beispiele:
 - **/Course[1] is /Course[1]**
→ true
 - **//Course[1] << //Course[2]**
→ true (Course-Element 1 kommt vor dem Course-Element 2)

XPath

Schleifenausdruck – for

- Ergebnis des Ausdrucks wird elementweise an die Variable **\$instructor** gebunden und das nachfolgende Anfragekonstrukt für jedes Element einzeln ausgeführt

- ```
for $instructor in //Instructor
 return upper-case($instructor/@instructorNumber)
```

P20621 P22080 P22100 P22101
- ```
for $ects in //Credit[@formatType="ECTS"] return $ects
```

```
<?xml version="1.0" encoding="UTF-8"?>  
<Credit formatType="ECTS">1</Credit>  
<Credit formatType="ECTS">1,5</Credit>  
<Credit formatType="ECTS">0,5</Credit>
```

- Schachtelung von "for"-Ausdruck erlaubt, da Ergebnis wieder eine Sequenz ist

- ```
count(for $hours in //@semesterHours return $hours)
```

3

# XPath

## Konditionaler Ausdruck – if

- Testausdruck in Klammern entscheidet, ob das Ergebnis der **then**- bzw. der **else**-Klausel zurückgeliefert wird.
  - `if (//DegreeProgramme[1]/@code = "0307")  
 then "SE"  
 else "not SE"`
- Schachtelung von Ausdrücken erlaubt  
⇒ Ausdrucksstärke!
  - `sum(  
 for $course in //Course  
 return  
 if ($course/CourseType/@type = "LabSession")  
 then $course/@semesterHours  
 else  
 ())`

# XPath

## Quantifizierender Ausdruck – some/every

### ■ Existenzielle Quantifizierung

- `some $courseTypes in //Course/CourseType satisfies  
$courseTypes/@type = "LabSession"`
  - → true

### ■ Universelle Quantifizierung

- `every $courseType in //Course/CourseType satisfies  
($courseType/@type != "Training")`
  - → false
- `every $x in (1 to 10) satisfies ($x > 11)`
  - → false

### ■ Ausdrucksstärker, da beliebige Bedingung nach **satisfies** möglich ist (nicht nur `=`, `!=`, `<`, ...)

- `some $x in (1 to 10) satisfies $x * $x < 10`
  - → true

# Inhalt

- Einführung
- Datenmodell
- Pfadausdruck
- Erweiterte Ausdrücke
- **Funktionen und Operatoren**
- Zusammenfassung

# XPath

## Funktionen und Operatoren

- Funktionen und Operatoren, die (u.a.) in XPath verwendet werden können, werden als Katalog in **XQuery and XPath Functions and Operators (FO)** definiert
  - siehe <https://www.w3.org/TR/xquery-operators/>
- Standard wird zwischen unterschiedlichen Technologien geteilt (u.a. XPath, XQuery und XSLT)
- Konkrete Syntax der Funktionen und Operatoren wird in den jeweiligen Technologien beschrieben

# XPath

## Operatoren – Sequenzen

### ■ Vereinigung

- $(A, B) \text{ union } (A, B) \rightarrow (A, B)$
- $(A, B) \text{ union } (B, C) \rightarrow (A, B, C)$

### ■ Durchschnitt

- $(A, B) \text{ intersect } (A, B) \rightarrow (A, B)$
- $(A, B) \text{ intersect } (B, C) \rightarrow (B)$

### ■ Differenz

- $(A, B) \text{ except } (A, B) \rightarrow ()$
- $(A, B) \text{ except } (B, C) \rightarrow (A)$

# XPath

## Operatoren – Vergleiche

- Wertevergleich
  - `eq`, `ne`, `lt`, `le`, `gt` und `ge`
- Allgemeiner Vergleich
  - `=`, `!=`, `<`, `<=`, `>` und `>=`
- Knotenvergleich
  - `is`, `<<` und `>>`

# XPath

## Sequenzfunktionen

### ■ Strukturfunktionen

- `insert-before((7, 9, 10), 2, 8)` → `(7, 8, 9, 10)`
- `remove((1, 2, 3), 2)` → `(1, 3)`
- `index-of((10, 20, 30), 20)` → `2`
- `empty(())` → `true`
- `exists((1, 2, 3))` → `true`
- ...

### ■ Aggregatfunktionen

- `sum(1, 2, 3)` → `6` (: Kommentar :)
- `count(1, 2, 3)` → `3`
- `avg(1, 2, 3)` → `2`
- `min(1, 2, 3)` → `1`
- `max(1, 2, 3)` → `3`

# XPath

## Zeichenkettenfunktionen

- Konvertierung von Klein- bzw. Großbuchstaben
  - `upper-case('Introduction')` → `'INTRODUCTION'`
  - `lower-case()`
- Konkatenation
  - `concat('FH3', '.', '108')` → `'FH3.108'`
- div. weitere Zeichenketten-Funktionen
  - `ends-with()`,
  - `starts-with()`,
  - `substring-before()`,
  - `string-length()`,
  - `contains()`,
  - `normalize-space`, etc.

# XPath

... und viele weitere Funktionen

- Funktionen für boolesche Werte
  - `boolean()`, `false()`, `true()`, `not()`
- Funktionen für numerische Werte
  - `abs()`, `ceiling()`, `floor()`, `round()`
- Funktionen für Zeitangaben und Zeitdauer
  - `current-date()`, `current-time()`, `current-dateTime()`
- Funktionen für die Knotensuche
  - `collection()`, `doc()`, `id()`, `root()`
- Funktionen auf Knoten
  - `name()`, `local-name()`, `namespace-uri()`
- Funktionen für Konvertierungen
  - `number()`, `string()`, `boolean()`
- Kontextfunktionen
  - `position()`, `last()`

→ siehe [www.w3.org/TR/xpath-functions/](http://www.w3.org/TR/xpath-functions/)

# Inhalt

- Einführung
- Datenmodell
- Pfadausdruck
- Erweiterte Ausdrücke
- Funktionen und Operatoren
- **Zusammenfassung**

# Zusammenfassung

- XPath dient zum Navigieren in XML-Dokumenten und zur Selektion (von Teilen) von XML-Dokumenten
  - XPath 2.0 wird oft als Sprache zur Verarbeitung von Sequenzen, die auch zur Navigation in XML-Dokumenten verwendet werden kann, bezeichnet
- Bildet auch gemeinsame Grundlage von XQuery und XSLT
- XPath 2.0 Recommendation
  - siehe [www.w3.org/TR/xpath20/](http://www.w3.org/TR/xpath20/)
- Verwendet dasselbe Datenmodell wie andere XML-Technologien (**XML-DM**)
- Unterstützt eine Vielzahl von eingebauten Funktionen und Operatoren (**FO**)
  - siehe [www.w3.org/TR/xpath-functions/](http://www.w3.org/TR/xpath-functions/)
- XPath 3.0 und 3.1 sinnvolle Erweiterungen bei fortgeschrittener Anwendung von XPath

# Ressourcen

## ■ W3C-Recommendations

- XPath 3.1: <https://www.w3.org/TR/xpath-31/>
- XPath 3.0: <https://www.w3.org/TR/xpath-30/>
- XPath 2.0: <https://www.w3.org/TR/xpath20/>
- XPath 1.0: <https://www.w3.org/TR/1999/REC-xpath-19991116/>

## ■ News zum Thema XPath

- <https://www.xml.com/news/?tag=xpath>

## Modul 5

---

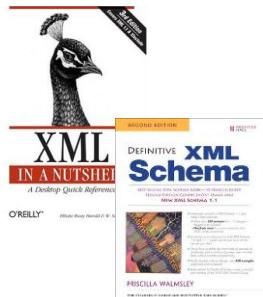
# XML Schema

## XML Schema Definition Language (XSD)

---

Julian Haslinger

<?xml?>



**Der vorliegende Foliensatz basiert vorwiegend auf:**

*Elliotte Rusty Harold, W. Scott Means: XML in a Nutshell: A Desktop Quick Reference, 3<sup>rd</sup> Edition, O'Reilly, 2005*  
*Priscilla Walmsley: Definitive XML Schema, 2<sup>nd</sup> Edition, Prentice Hall, 2012*

# Inhalt

- **Einführung**
- Elemente und Attribute
- Vordefinierte Datentypen
- Benutzerdefinierte Datentypen
- Schlüssel und Schlüsselreferenzen
- Modularisierung und Komposition
- Modellierungsmuster
  
- Anhang I: DTD versus XML Schema
- Anhang II: Facetten
- Anhang III: Entwurfsrichtlinien
- Anhang IV: Annotationen
- Anhang V: XML Schema 1.1 – Erweiterungen

# Einführung 1/5

## DTD versus XML Schema

### ■ Nachteile DTDs

- keine XML-Syntax
  - wenige Datentypen
    - Elementinhalte nur Text
    - Wenige Attributtypen
  - Keine benutzerdefinierte Datentypen
  - Eingeschränkte Wiederverwendung (nur Parameter Entities, keine Vererbung)
  - Nur einfache Integritätsbedingungen formulierbar
  - ID, IDREF(S): Einschränkungen
  - Keine Unterstützung von XML-Namensräumen
- **zur Beschreibung von Textdokumenten ausreichend**

### ■ Vorteile XML Schema

- XML als Syntax
  - zahlreiche vordefinierte Datentypen
    - für Elemente und Attribute
  - Benutzerdefinierte einfache und komplexe Datentypen
  - Wiederverwendungskonzepte auf Typ-, Element- und Attributebene (strukturelle Vererbungsmechanismen)
  - Komplexe Integritätsbedingungen formulierbar
  - Schlüssel, Referenz: flexibles Konzept
  - Unterstützung von XML-Namensräumen
- **zur Beschreibung von Daten besser geeignet**

# Einführung 2/5

## DTD versus XML Schema

XML-Dokument

CourseCatalog.xsd

Elementdeklaration

```

<?xml version="1.0"?>
<xss:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
 targetNamespace="http://www.fh-ooe.at/CourseCatalog">

 ...
 <xss:element name="CourseCatalog" type="cc:CourseCatalogType"></xss:element>
 ...
 <xss:complexType name="CourseCatalogType">
 <xss:sequence>
 <xss:element name="DegreeProgramme" type="cc:DegreeProgrammeType" minOccurs="0" maxOccurs="unbounded"/>
 </xss:sequence>
 <xss:attribute name="term" type="cc:termType" use="required"/>
 <xss:attribute name="campus" type="cc:campusType" use="required"/>
 <xss:attribute name="year" type="xs:gYear" use="required"/>
 </xss:complexType>
 ...
 <xss:simpleType name="termType">
 <xss:restriction base="xs:string">
 <xss:enumeration value="summer"/>
 <xss:enumeration value="winter"/>
 </xss:restriction>
 </xss:simpleType>
 ...
</xss:schema>

```

Annotations:

- Namensraum des XML Schema-Standards: `xmlns:xs="http://www.w3.org/2001/XMLSchema"`
- Namensraum des CourseCatalog Schemas: `xmlns:cc="http://www.fh-ooe.at/CourseCatalog"`
- Zielnamensraum für das Instanzdokument: `targetNamespace="http://www.fh-ooe.at/CourseCatalog"`
- Inhaltsmodell: `<xss:sequence>`
- Attributdeklaration: `<xss:attribute ...>`
- Vordefinierte einfache Datentypen: `<xss:restriction base="xs:string">`
- Einschränkung durch Facetten: `<xss:enumeration value="summer"/>`, `<xss:enumeration value="winter"/>`
- Benutzerdefinierter einfacher Datentyp: `<xss:element name="DegreeProgramme" type="cc:DegreeProgrammeType" minOccurs="0" maxOccurs="unbounded"/>`
- Benutzerdefinierter komplexer Datentyp: `<xss:complexType name="CourseCatalogType">`
- Kardinalität: `minOccurs="0" maxOccurs="unbounded"`

Datentypdefinition  
(komplexer Datentyp)Datentypdefinition  
durch Ableitung  
(einfacher Datentyp)

CourseCatalog.dtd

```

...
<!ELEMENT CourseCatalog (DegreeProgramme*)>
<!ATTLIST CourseCatalog year CDATA #REQUIRED
 term (summer|winter) #REQUIRED
 campus (Hagenberg|Linz|Steyr|Wels) #REQUIRED>
...

```

# Einführung 3/5

## XML Schema

- Unterscheidung von **Dokumentenschema** und konkreten Ausprägungen, den sog. **Instanz-Dokumenten**
- XML Schema ist Datendefinitionssprache zur Festlegung
  - der **Struktur** von Instanz-Dokumenten
  - des **Datentyps** jedes einzelnen Elements/Attributs
- **XML Schema 1.0** (2004)
  - Part 0: Primer Second Edition
    - <https://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>
  - Part 1: Structures Second Edition
    - <https://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>
  - Part 2: Datatypes Second Edition
    - <https://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>
- **XML Schema Definition Language (XSD) 1.1** (2012)
  - Part 1: Structures
    - <https://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/>
  - Part 2: Datatypes
    - <https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/>

# Einführung 4/5

## Deklaration von Namensräumen im Schema

- Namensraum für eigenes Vokabular
  - Namensraum (Namespace: NS) des zu definierenden Schemas kann über **targetNamespace** festgelegt werden.
- Namensraum des XML-Schema-Standard-Vokabulars
  - NS des XML Schema-Standards (definiert **<xs:element>**, **<xs:attribute>**,...) muss angegeben werden.
  - Weitere NS können eingebunden werden.
- Ein NS kann als Default-NS definiert werden
  - zu definierender NS, XML-Schema-NS oder anderer NS
  - für alle anderen muss ein Präfix verwendet werden

CourseCatalog.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.fh-ooe.at/CourseCatalog"
 xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
 elementFormDefault="qualified"
 attributeFormDefault="unqualified">
 ...
</xs:schema>
```

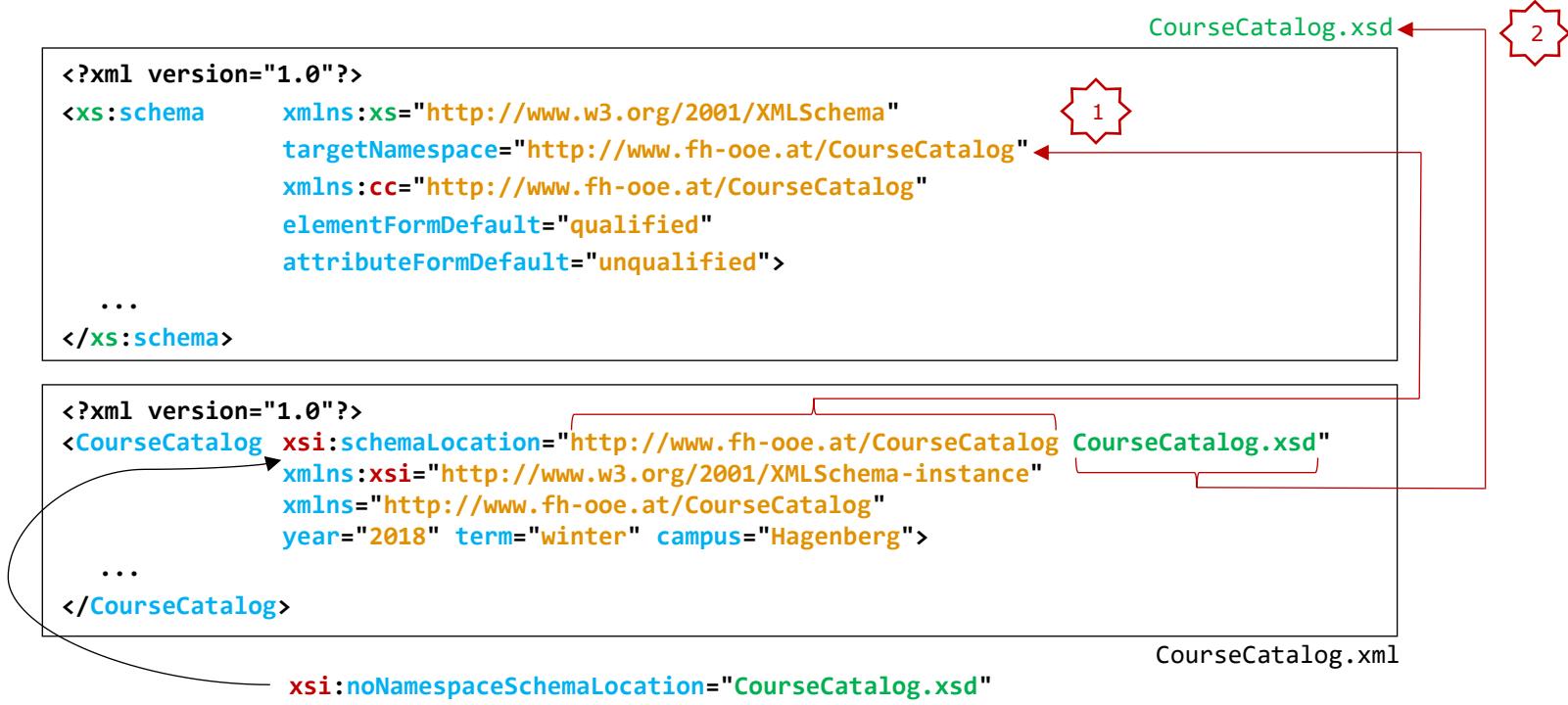
# Einführung 5/5

## Verwendung von Namensräumen im XML-Dokument

- Schema eines XML-Dokuments wird im Wurzelement durch Attribut `schemaLocation` bestimmt

1 Komponente: `targetNamespace` des Schemas

2 Komponente: Pfadangabe des Schema-Dokuments



# Inhalt

- Einführung
- **Elemente und Attribute**
- Vordefinierte Datentypen
- Benutzerdefinierte Datentypen
- Schlüssel und Schlüsselreferenzen
- Modularisierung und Komposition
- Modellierungsmuster
  
- Anhang I: DTD versus XML Schema
- Anhang II: Facetten
- Anhang III: Entwurfsrichtlinien
- Anhang IV: Annotationen
- Anhang V: XML Schema 1.1 – Erweiterungen

# Aufbau XML Schema-Dokument

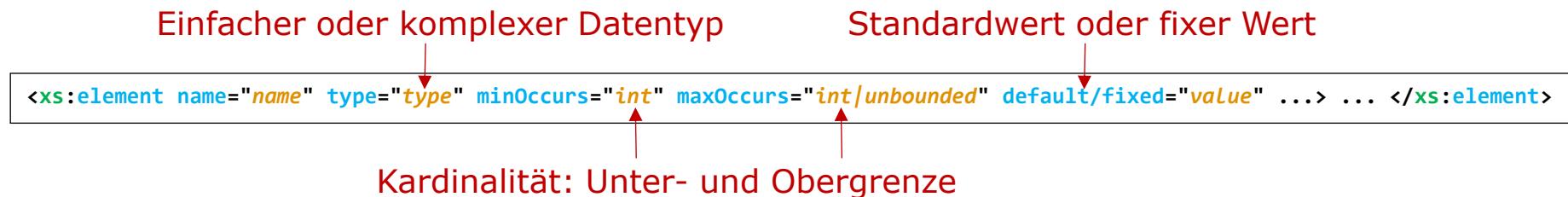
- Wurzelement <**xs:schema** ...>
  - mit Angabe eines **targetNamespace**, also dem Namensraum, in dem die Definitionen und Deklarationen gelten sollen
- Darin enthalten sind
  - Elementdeklarationen
  - Attributdeklarationen
  - Datentypdefinitionen (einfache und komplexe Datentypen)

CourseCatalog.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.fh-ooe.at/CourseCatalog"
 xmlns:cc="http://www.fh-ooe.at/CourseCatalog" ...>

 <xs:element name="CourseCatalog" ...> ... </xs:element> <!-- Elementdeklaration -->
 ...
 <xs:attribute name="campus" ...> ... </xs:attribute> <!-- Attributdeklaration -->
 ...
 <xs:simpleType name="termType"> ... </xs:simpleType> <!-- Datentypdefinition -->
 ...
 <xs:complexType name="CourseCatalogType"> ... </xs:complexType> <!-- Datentypdefinition -->
 ...
</xs:schema>
```

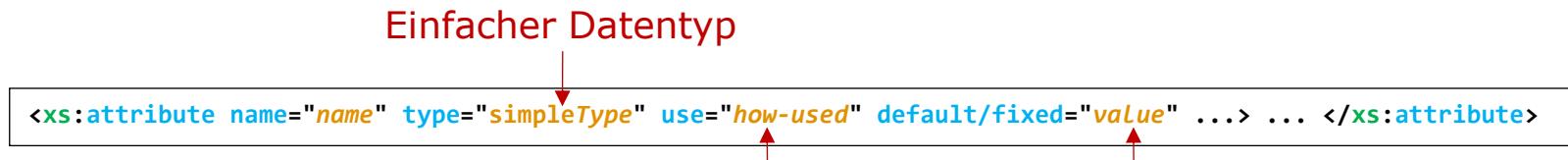
# Element Deklaration



- Die Anzahl der Vorkommen eines Elements innerhalb eines XML-Instanzdokuments kann durch die Attribute
  - **minOccurs="int"** (Untergrenze, Standardwert "1" )
  - **maxOccurs="int|unbounded"** (Obergrenze, Standardwert "1")im <**xs:element**>-Tag festgelegt werden.
- Mit **default** kann ein Vorgabewert für das Element angegeben werden.
- Mit **fixed** kann ein fester, nicht veränderbarer Wert festgelegt werden.

vgl. [https://www.w3.org/TR/xmlschema-1/#cElement\\_Declarations](https://www.w3.org/TR/xmlschema-1/#cElement_Declarations)

# Attribut Deklaration



**required** Attribut obligatorisch

**optional** Attribut optional (Standardwert)

**prohibited** Attribut unzulässig

**default** `use` muss **optional** sein

**fixed** nur relevant, wenn `use` nicht gesetzt

**default** und **fixed** schließen sich gegenseitig aus, d.h. in einer Attribut-deklaration dürfen beide Attribute nicht gleichzeitig vorkommen.

- Attribute können nur einfache Datentypen aufnehmen

# Elemente und Attribute 1/3

## Lokale und globale Datentypen

- Elemente und Attribute können lokale oder globale Datentypen verwenden
  - Element/Attribut mit **lokalem** (anonymen) Datentyp

```
<xs:element name="name" minOccurs="int" maxOccurs="int/unbounded">
 <xs:complexType> ... </xs:complexType>
</xs:element>
```

```
<xs:attribute name="name" use="how-used" default/fixed="value" ...>
 <xs:simpleType> ... </xs:simpleType>
</xs:attribute>
```

- Element/Attribut mit **globalem** (benannten) Datentyp  
(global = direktes Kindelement von `<xs:schema>`)

```
<xs:element name="name" type="typeName" minOccurs="int" ... </xs:element>
```

```
<xs:attribute name="name" type="typeName" use="how-used" ... </xs:attribute>
```

# Elemente und Attribute 2/3

## Globale Elemente und Attribute

- Globale Element- und Attributdeklarationen treten als direktes Kindelement von `<xs:schema>` auf
- Globale Deklarationen sind überall im Schema sichtbar und können an beliebigen Stellen mit dem Attribut `ref` referenziert werden (Wiederverwendung).
  - Verweis auf bereits bestehendes Element bzw. Attribut

```
<xs:element ref="name" minOccurs="int" maxOccurs="int/unbounded" ... />
```

```
<xs:attribute ref="name" use="how-used" default/fixed="value" ... />
```

- Einschränkungen bei Deklaration von globalen Elementen und Attributen
  - Verwendung von `ref`-Attribut nicht erlaubt
  - Kardinalität darf nicht eingeschränkt werden

# Elemente und Attribute 3/3

## Global/Lokal - Beispiel

Elementdeklaration  
 Datentypdefinition

Globales Element mit  
globalem Datentyp

Globaler Datentyp

Lokales Element mit  
globalem Datentyp

Lokales Attribut mit  
globalem Datentyp

Globaler Datentyp

Referenz auf  
globales Element

Globales Element mit  
vordefiniertem Datentyp

CourseCatalog.xsd

```

<xs:schema ... >
 <xs:element name="CourseCatalog" type="cc:CourseCatalogType"></xs:element>
 <xs:complexType name="CourseCatalogType">
 <xs:sequence>
 <xs:element name="DegreeProgramme" type="cc:DegreeProgrammeType"
 minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="term" type="cc:termType" use="required"/>
 <xs:attribute name="campus" type="cc:campusType" use="required"/>
 <xs:attribute name="year" type="xs:gYear" use="required"/>
 </xs:complexType>
 ...
 <xs:complexType name="CourseType">
 <xs:sequence>
 <xs:element ref="cc:Title"/>
 <xs:element name="Description" type="cc:DescriptionType"/>
 </xs:sequence>
 </xs:complexType>
 ...
 <xs:simpleType name="termType">
 <xs:restriction base="xs:string">
 <xs:enumeration value="summer"/>
 <xs:enumeration value="winter"/>
 </xs:restriction>
 </xs:simpleType>
 ...
 <xs:element name="Title" type="xs:string"/>
</xs:schema>
```

Verweis auf globale  
Elementdeklaration

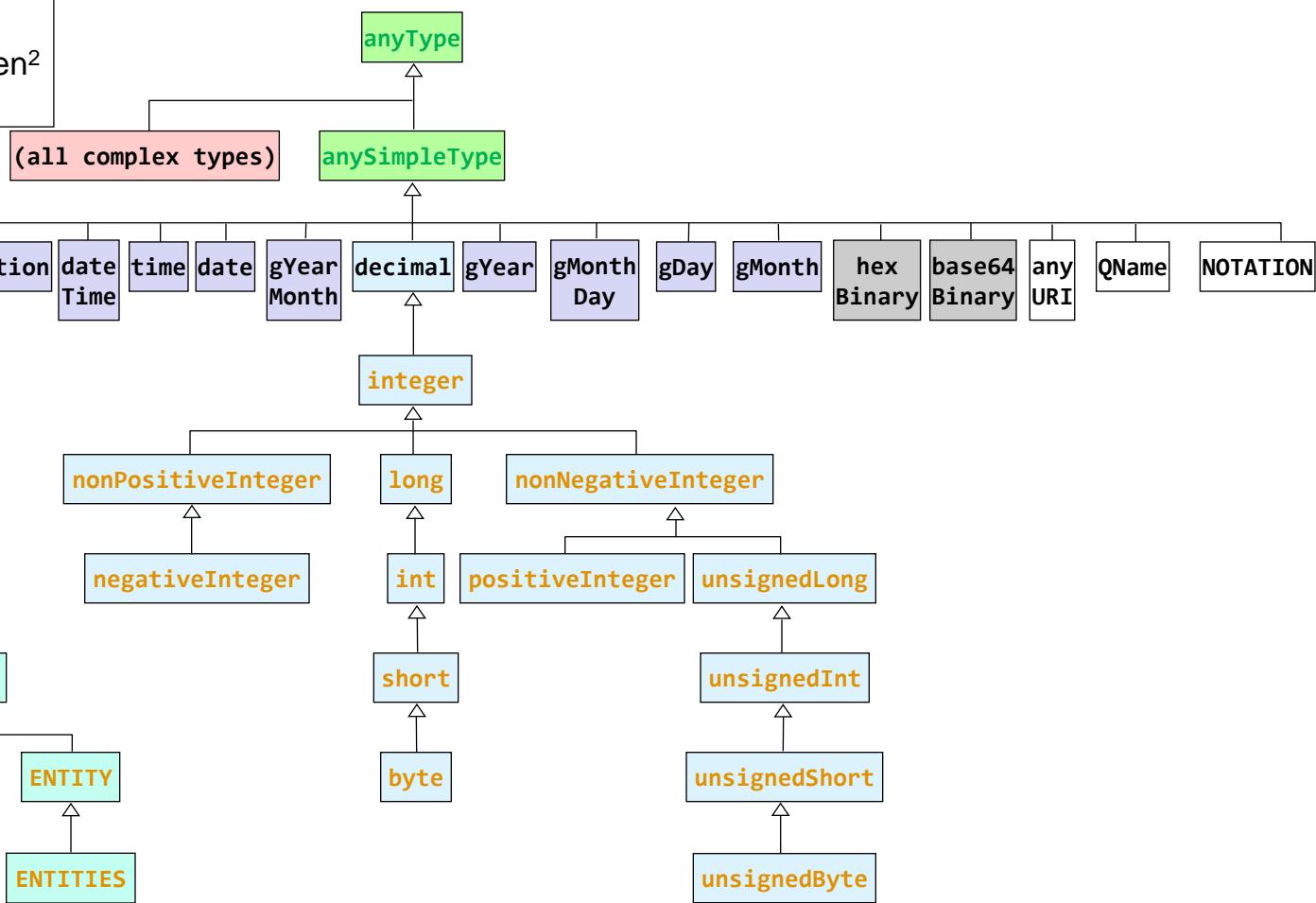
# Inhalt

- Einführung
  - Elemente und Attribute
  - **Vordefinierte Datentypen**
  - Benutzerdefinierte Datentypen
  - Schlüssel und Schlüsselreferenzen
  - Modularisierung und Komposition
  - Modellierungsmuster
- 
- Anhang I: DTD – XML Schema – Vergleich
  - Anhang II: Facetten – Wertebereichseinschränkung
  - Anhang III: XML Schema Entwurfsrichtlinien
  - Anhang IV: XML Schema Dokumentation
  - Anhang V: XML Schema 1.1 – Erweiterungen

# Vordefinierte Datentypen 1/4

## Typhierarchie von W3C XML Schema Datentypen

- Ur-Typen<sup>1</sup>
- Primitive (atomare) Typen<sup>2</sup>
- Abgeleitete Typen<sup>3</sup>



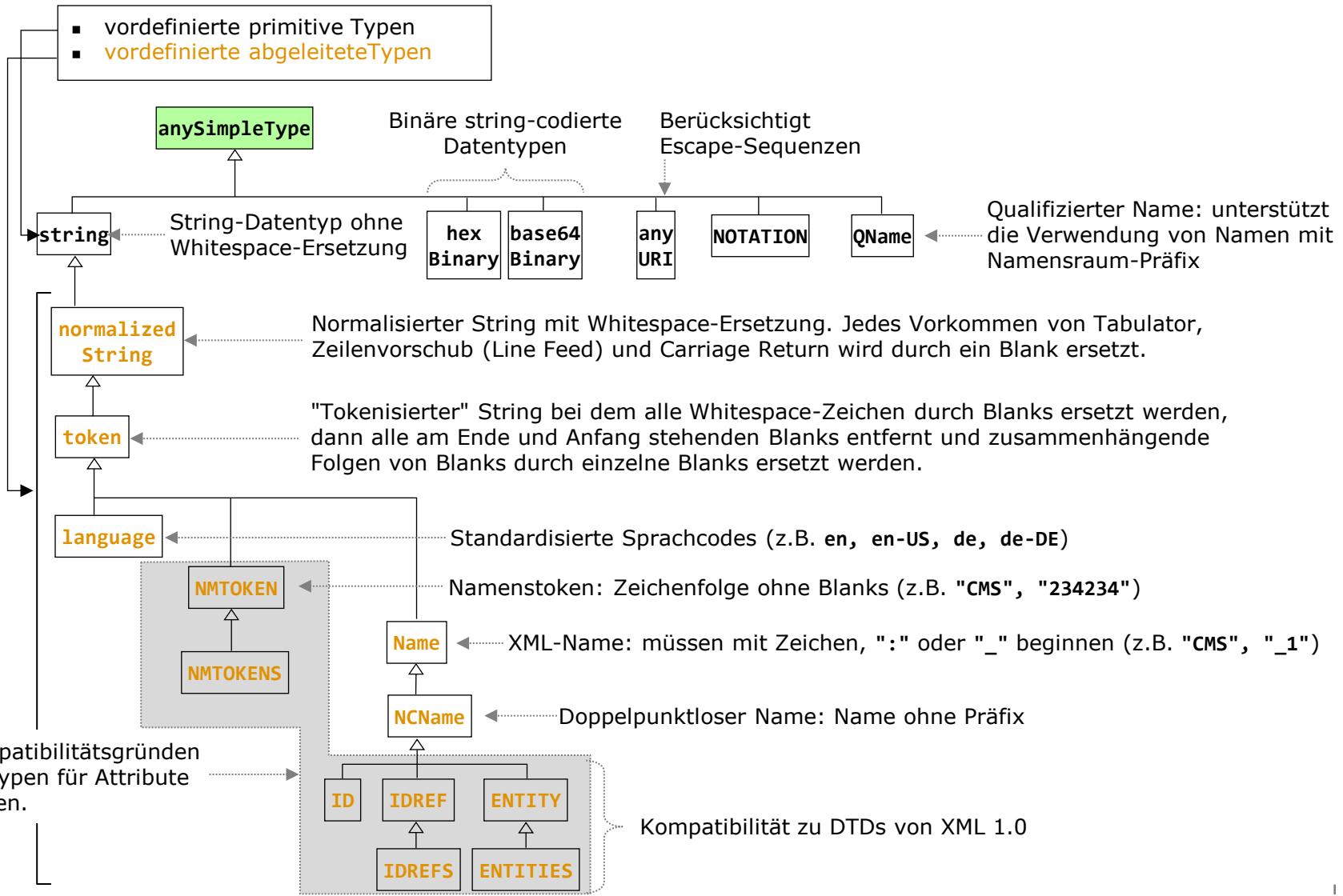
<sup>1</sup> vgl. <https://www.w3.org/TR/xmlschema-2/#built-in-datatypes>

<sup>2</sup> vgl. <https://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes>

<sup>3</sup> vgl. <https://www.w3.org/TR/xmlschema-2/#built-in-derived>

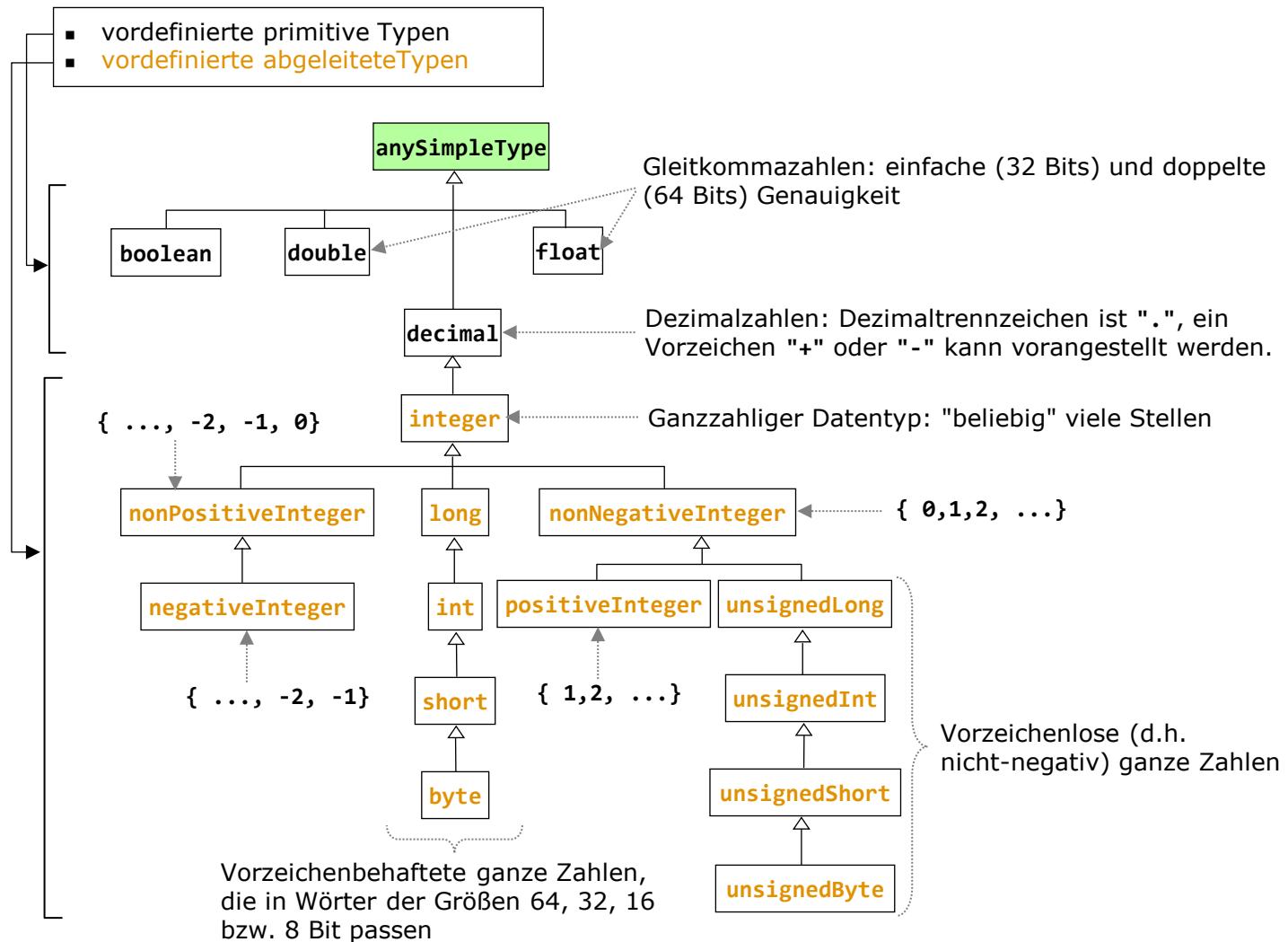
# Vordefinierte Datentypen 2/4

## Zeichenketten-Datentypen



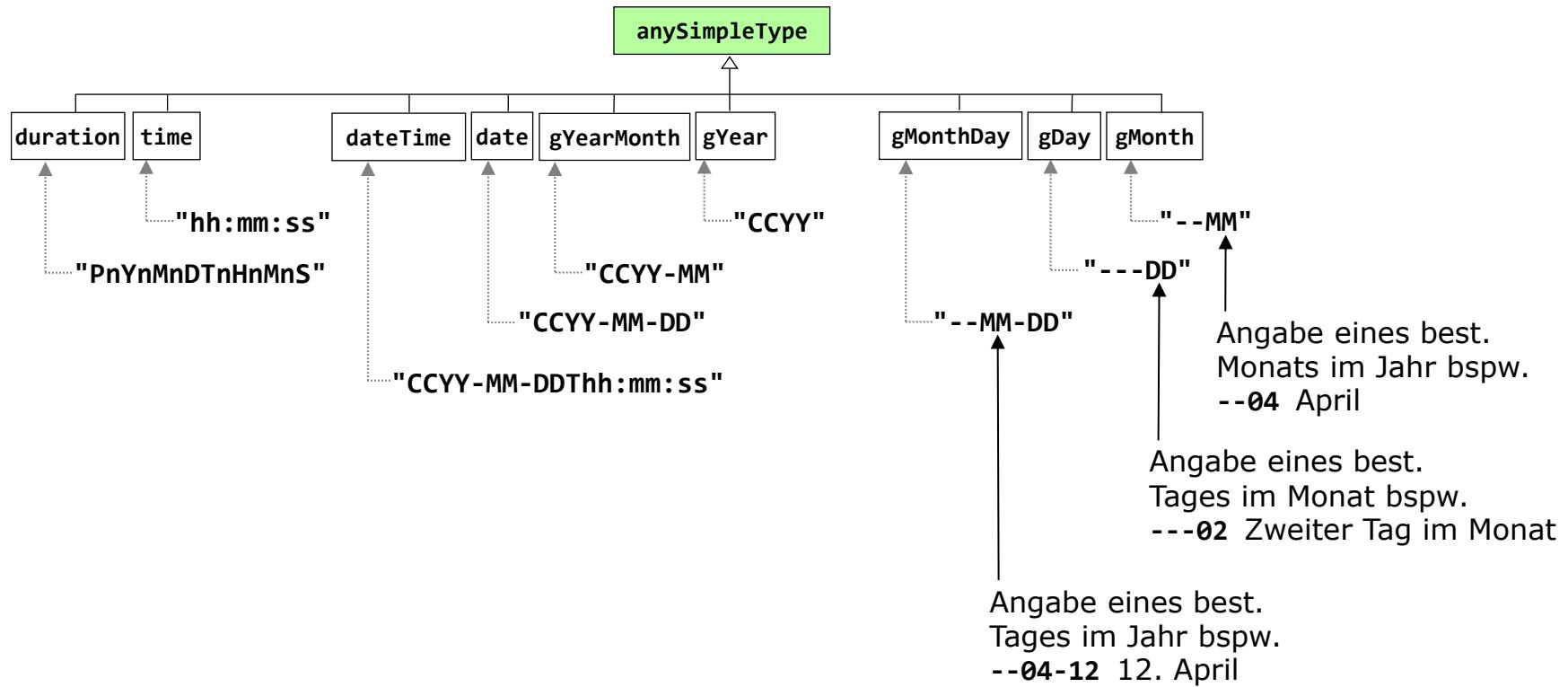
# Vordefinierte Datentypen 3/4

## Numerische Datentypen



# Vordefinierte Datentypen 4/4

## Datums- und Zeit-Datentypen

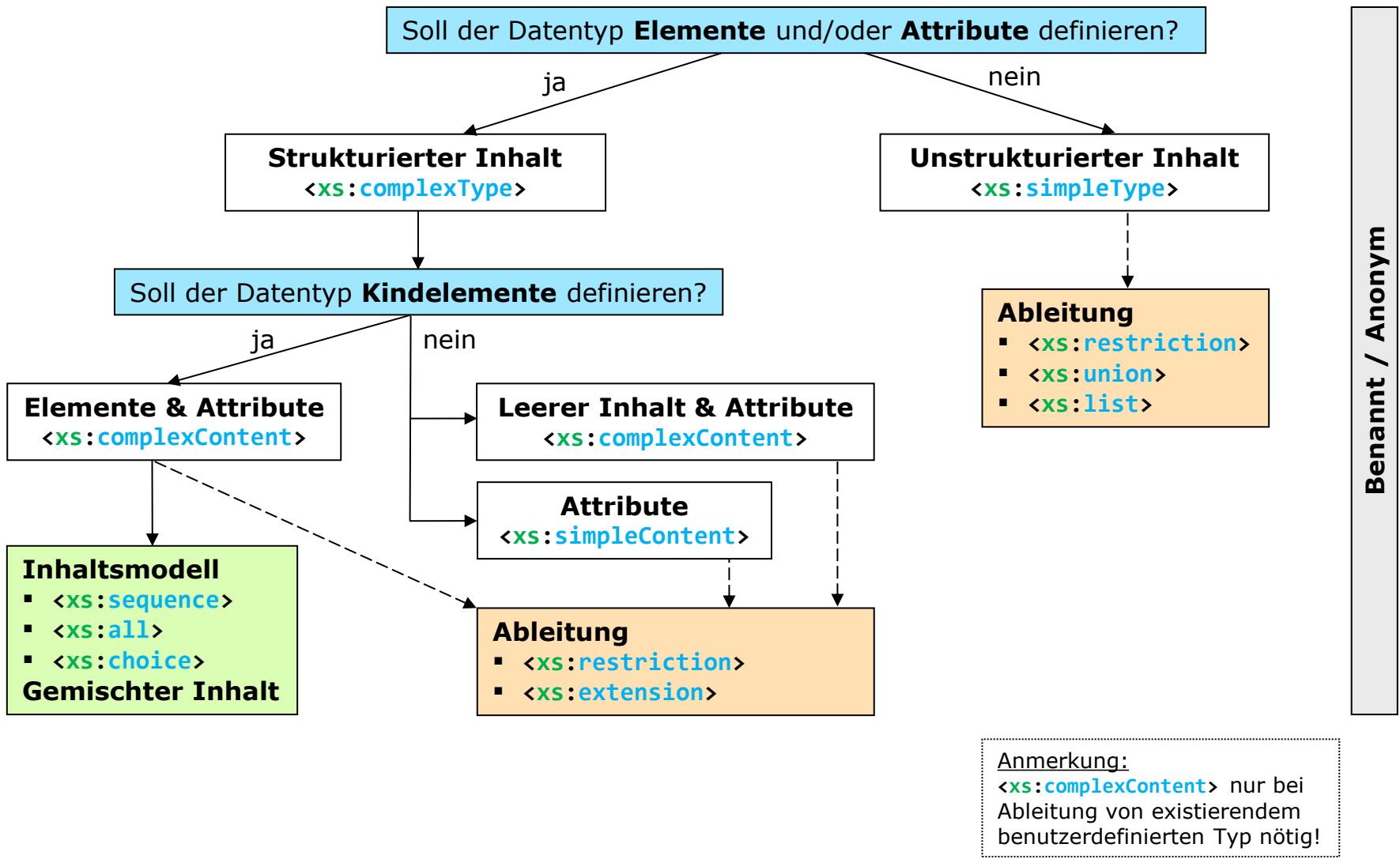


# Inhalt

- Einführung
  - Elemente und Attribute
  - Vordefinierte Datentypen
  - **Benutzerdefinierte Datentypen**
  - Schlüssel und Schlüsselreferenzen
  - Modularisierung und Komposition
  - Modellierungsmuster
- 
- Anhang I: DTD – XML Schema – Vergleich
  - Anhang II: Facetten – Wertebereichseinschränkung
  - Anhang III: XML Schema Entwurfsrichtlinien
  - Anhang IV: XML Schema Dokumentation
  - Anhang V: XML Schema 1.1 – Erweiterungen

# Benutzerdefinierte Datentypen

## Alternativen



# Benutzerdefinierte Datentypen

## Alternativen - Beispiele

- Benutzerdefiniert
- Vordefiniert

### Nicht abgeleitet

```
<xs:decimal>
```

Einfach

### Abgeleitet

```
<xs:simpleType name="creditType">
 <xs:restriction base="xs:decimal">
 <xs:fractionDigits value="1"/>
 <xs:minInclusive value="0.5"/>
 <xs:maxInclusive value="30"/>
 </xs:restriction>
</xs:simpleType>
```

```
<xs:complexType
 name="CourseType">
 <xs:sequence>
 <xs:element .../>

 </xs:sequence>
 <xs:attribute .../>
</xs:complexType>
```

Komplex

```
<xs:complexType name="InternationalCourseType">
 <xs:complexContent>
 <xs:extension base="cc:CourseType" >
 <xs:sequence>
 <xs:element name="Prerequisites" type="xs:string"/>
 </xs:sequence>
 <xs:attribute name="languageCertificate"
 type="cc:certificateType" use="required"/>
 </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

# Benutzerdefinierte Datentypen

## Abgeleitete Einfache Datentypen – <xs:simpleType>

- **Einschränkung** eines vordefinierten Datentyps
  - <xs:restriction>
  - Wertebereich wird eingeschränkt
- **Vereinigung** von vordefinierten Datentypen (Erweiterung)
  - <xs:union>
  - Werte des neuen Datentyps müssen zumindest einem der vereinigten Datentypen entsprechen
- **Liste** von Werten eines vordefinierten Datentyps (oder wiederum eines List-Datentyps)
  - <xs:list>
  - Werte sind durch *Whitespace* getrennt

# Benutzerdefinierte Datentypen

Abgeleitete Einfache Datentypen `<xs:simpleType>` – `<xs:restriction>`

- Datentypdefinition mit
  - Attribut `base` existierenden einfachen Datentyp referenzieren
  - `<xs:restriction>` als Sub-Element des `<xs:simpleType>`-Elements neu definieren
- 12 mögliche Einschränkungen (Facetten), abhängig vom Basisdatentyp:

**Facetten:**

- `length`
- `minLength`
- `maxLength`
- `pattern`
- `enumeration`
- `minInclusive`
- `maxInclusive`
- `minExclusive`
- `maxExclusive`
- `totalDigits`
- `fractionDigits`
- `whitespace`
- `assertion`

► Anhang II

CourseCatalog.xsd

```

<xs:simpleType name="courseType">
 <xs:restriction base="xs:string"> ← Basisdatentyp
 { Facetten
 <xs:enumeration value="Lecture"/>
 <xs:enumeration value="Seminar"/>
 <xs:enumeration value="LabSession"/>
 <xs:enumeration value="PracticeSession"/>
 <xs:enumeration value="Training"/>
 } Werteliste
 </xs:restriction>
</xs:simpleType>

<xs:element name="CourseType" type="cc:courseType"/>

```



CourseCatalog.xml

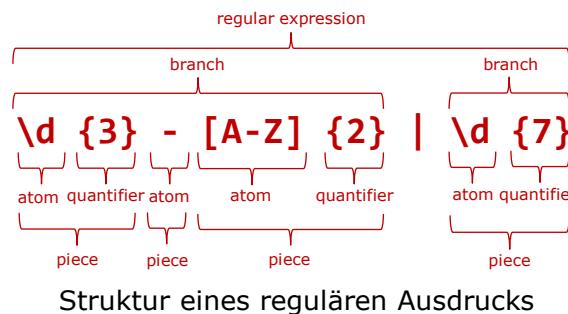
```
<CourseType>Lecture</CourseType>
```

# Benutzerdefinierte Datentypen

Abgeleitete Einfache Datentypen `<xs:simpleType>` – `<xs:restriction>`

## Einschränkung: `<xs:pattern>`

- regulärer Ausdruck schränkt Wertebereich ein



CourseCatalog.xsd

```

<xs:simpleType name="instructorNumberType">
 <xs:restriction base="xs:string">
 <xs:pattern value="p[1-4]{1}[\d]{4}"/>
 </xs:restriction>
</xs:simpleType> Regulärer Ausdruck

<xs:element name="Instructor" type="cc:instructorNumberType"/>

```

CourseCatalog.xml

```

<Instructor>p22080</Instructor>

```

# Benutzerdefinierte Datentypen

Abgeleitete Einfache Datentypen `<xs:simpleType>` – `<xs:restriction>`

- Einschränkung: `<xs:fractionDigits>`
  - Anzahl der Nachkommastellen begrenzt
- Einschränkung: `<xs:minInclusive>`
  - eingeschlossene Untergrenze begrenzt Wertebereich
- Einschränkung: `<xs:maxInclusive>`
  - eingeschlossene Obergrenze begrenzt Wertebereich

CourseCatalog.xsd

```
<xs:simpleType name="creditType">
 <xs:restriction base="xs:decimal">
 <xs:fractionDigits value="1"/> ← Nachkommastellen
 <xs:minInclusive value="0.5"/> ← Untergrenze
 <xs:maxInclusive value="30"/> ← Obergrenze
 </xs:restriction>
</xs:simpleType>

<xs:element name="Credit" type="cc:creditType "/>
```

CourseCatalog.xml

```
<Credit>2.5</Credit>
```

# Benutzerdefinierte Datentypen

## Abgeleitete Einfache Datentypen – <xs:union>

- Ableitung durch Vereinigung von einfachen Datentypen
  - über **memberTypes**-Attribut existierende Datentypen referenzieren

CourseCatalog.xsd

|                                                                                                                                                                                                                                                                                                                                                   |                                                                                                                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> &lt;xs:simpleType name="letterGradeType"&gt;   &lt;xs:restriction base="xs:token"&gt;     &lt;xs:enumeration value="A"/&gt;     &lt;xs:enumeration value="B"/&gt;     &lt;xs:enumeration value="C"/&gt;     &lt;xs:enumeration value="D"/&gt;     &lt;xs:enumeration value="F"/&gt;   &lt;/xs:restriction&gt; &lt;/xs:simpleType&gt; </pre> | <pre> &lt;xs:simpleType name="numericalGradeType"&gt;   &lt;xs:restriction base="xs:integer"&gt;     &lt;xs:minInclusive value="1"/&gt;     &lt;xs:maxInclusive value="5"/&gt;   &lt;/xs:restriction&gt; &lt;/xs:simpleType&gt; </pre> |
| <pre> &lt;xs:simpleType name="gradeType"&gt;   &lt;xs:union memberTypes="cc:letterGradeType cc:numericalGradeType"/&gt; &lt;/xs:simpleType&gt; </pre> <p style="text-align: center;">Datentypvereinigung</p> <pre> &lt;xs:element name="Grade" type="cc:gradeType"/&gt; </pre>                                                                    |                                                                                                                                                                                                                                        |

CourseCatalog.xml

```

<Grade>2</Grade>
<Grade>B</Grade>

```

# Benutzerdefinierte Datentypen

## Abgeleitete Einfache Datentypen – <xs:list>

- Ableitung durch Auflistung von atomaren Werten eines einfachen Datentyps
  - über **itemType**-Attribut existierenden Datentyp referenzieren

CourseCatalog.xsd

```
<xs:simpleType name="gradeType">
 <xs:union memberTypes="cc:letterGradeType cc:numericalGradeType"/>
</xs:simpleType>

<xs:simpleType name="combinedGradeType">
 <xs:list itemType="cc:gradeType"/>
</xs:simpleType> <simpleType> oder <union>

<xs:element name="CombinedGrades" type="cc:combinedGradeType"/>
```

CourseCatalog.xml

```
<CombinedGrades>2 B 1 F</CombinedGrades>
```

# Benutzerdefinierte Datentypen

## Komplexe Datentypen – <xs:complexType>

- Geschachtelte Elemente
  - nur innerhalb eines komplexen Datentyps möglich
- Attribute
  - nur innerhalb eines komplexen Datentyps möglich
  - unabhängig davon, ob geschachtelte Elemente vorhanden oder nicht
- Leerer Inhalt – empty content
  - weist keine geschachtelten Elemente auf
  - nur innerhalb eines komplexen Datentyps möglich
- Gemischter Inhalt – mixed content
  - Datentyp kann geschachtelte Elemente und Text enthalten
  - im Gegensatz zu DTDs sind für die geschachtelten Elemente folgende Eigenschaften spezifizierbar:
    - Reihenfolge
    - Kardinalität

# Benutzerdefinierte Datentypen

<xs:complexType> – Geschachtelte Elemente

## ■ Sequenz - <xs:sequence>

```
<xs:complexType name="CourseType">
 <xs:sequence>
 <xs:element name="Title" type="xs:string" minOccurs="1" maxOccurs="1"/>
 <xs:element name="Description" type="cc:DescriptionType" minOccurs="1" maxOccurs="1"/>
 ...
 </xs:sequence>
</xs:complexType>

<xs:element name="Course" type="cc:CourseType"/>
```

## ■ Auswahl – <xs:choice>

- von den angeführten Elementen darf nur eines auftreten

## ■ Menge – <xs:all>

- Reihenfolge der Elemente beliebig
- jedes Element erscheint maximal einmal

## ■ Kardinalität wird durch **minOccurs** u. **maxOccurs** ausgedrückt

- Einschränkung bei <xs:all>: **minOccurs** kann nur die Werte **0** od. **1** annehmen, **maxOccurs** muss den Wert **1** aufweisen
-  **minOccurs** und **maxOccurs** dürfen **> 1** sein

# Benutzerdefinierte Datentypen

<xs:complexType> – Geschachtelte Elemente und Attribute

- Attribute werden am Ende der Typ-Definition angeführt

```
<xs:complexType name="CourseType">
 <xs:sequence>
 <xs:element name="Title" type="xs:string" minOccurs="1" maxOccurs="1"/>
 <xs:element name="Description" type="cc:DescriptionType" minOccurs="1" maxOccurs="1"/>
 ...
 </xs:sequence>
 <xs:attribute name="id" type="cc:idCourseType" use="required"/>
 <xs:attribute name="semesterHours" type="xs:decimal" use="required"/>
 <xs:attribute name="language" type="xs:language" use="optional"/>
 <xs:attribute name="basedOn" use="optional">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:pattern value="cID_[\d]{4}"/>
 </xs:restriction>
 </xs:simpleType>
 </xs:attribute>
</xs:complexType>
```

# Benutzerdefinierte Datentypen

<xs:complexType> – Atomarer Elementinhalt mit Attribut

CourseCatalog.xsd

```
<xs:complexType name="RoomType">
 <xs:simpleContent>
 <xs:extension base="xs:string">
 <xs:attribute name="roomNumber" type="xs:string" use="required"/>
 <xs:attribute name="building" type="xs:string" use="required"/>
 </xs:extension>
 </xs:simpleContent>
</xs:complexType>

<xs:element name="Room" type="cc:RoomType"/>
```

CourseCatalog.xml

```
<Room roomNumber="2.020" building="FH2">karriere.at Audimax</Room>
```

Atomarer Elementinhalt

# Benutzerdefinierte Datentypen

<xs:complexType> – Leerer Elementinhalt mit Attribut

CourseCatalog.xsd

```
<xs:complexType name="DateType">
 <xs:attribute name="startDate" type="xs:gMonthDay" use="required"/>
 <xs:attribute name="endDate" type="xs:gMonthDay" use="required"/>
</xs:complexType>

<xs:element name="Date" type="cc:DateType"/>
```

CourseCatalog.xml

```
<Date startDate="--10-03" endDate="--01-24"></Date>
```

# Benutzerdefinierte Datentypen

<xs:complexType> – Gemischter Elementinhalt

CourseCatalog.xsd

```
<xs:complexType name="DescriptionType" mixed="true">
 <xs:sequence>
 <xs:element name="Content" type="xs:string" minOccurs="0" maxOccurs="1"/>
 <xs:element name="Exam" type="xs:string" minOccurs="0" maxOccurs="1"/>
 <xs:element name="Tool" type="xs:string" minOccurs="0" maxOccurs="1"/>
 </xs:sequence>
</xs:complexType>

<xs:element name="Description" type="cc:DescriptionType"/>
```

CourseCatalog.xml

```
<Description>Introduction of skills related to XML.<Content>Includes DTD, Schema, XPath,
XQuery, XSLT, JSON</Content><Exam>Final Exam required.</Exam>Participation without any
previous knowledge.</Description>
```

- Reihenfolge und Anzahl des Auftretens von Kindelementen wird kontrolliert!

# Benutzerdefinierte Datentypen

<xs:complexType> – Ableitung von komplexen Typen

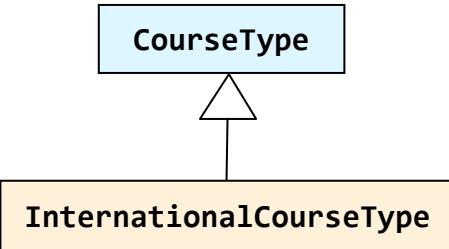
- Erweiterung
  - <xs:extension>
  - zusätzliche geschachtelte Elemente und/oder Attribute
- Einschränkung
  - <xs:restriction>
  - Wertebereich
  - Kardinalität

---

- Abstrakte Datentypen
  - <xs:complexType> mit Attribut **abstract="true"**
- Verbot der Ableitung
  - <xs:complexType> mit Attribut **final**
  - mit Ausprägungen: **#all, restriction, extension**

# Benutzerdefinierte Datentypen

<xs:complexType> – Ableitung durch Erweiterung



```

<xs:complexType name="CourseType">
 <xs:sequence>
 <xs:element name="Title" type="xs:string"/>
 <xs:element name="Description" type="cc:DescriptionType"/>
 </xs:sequence>
 <xs:attribute name="id" type="cc:idCourseType" use="required"/>
 <xs:attribute name="semesterHours" type="xs:decimal" use="required"/>
</xs:complexType>

```

```

<xs:complexType name="InternationalCourseType">
 <xs:complexContent>
 <xs:extension base="cc:CourseType">
 <xs:sequence>
 <xs:element name="Prerequisites" type="xs:string" minOccurs="0" maxOccurs="1"/>
 </xs:sequence>
 <xs:attribute name="languageCertificate" type="cc:certificateType" use="required"/>
 </xs:extension>
 </xs:complexContent>
</xs:complexType>

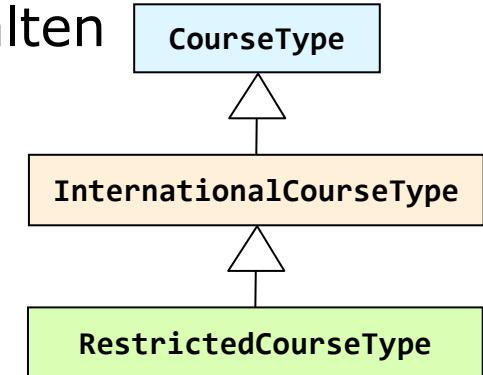
```

- Elemente werden am Ende angefügt
- Erweiterung muss innerhalb eines <xs:complexContent>-Elements vorgenommen werden

# Benutzerdefinierte Datentypen

<xs:complexType> – Ableitung durch Einschränkung

- Die Deklarationen des Basistyps, die beibehalten werden sollen, müssen wiederholt werden!
- Einschränkung muss innerhalb eines <xs:complexContent>-Elements vorgenommen werden



```

<xs:complexType name="RestrictedCourseType">
 <xs:complexContent>
 <xs:restriction base="cc:InternationalCourseType">
 <xs:sequence>
 <xs:element name="Title" type="xs:string"/>
 <xs:element name="Description" type="cc:DescriptionType"/>
 </xs:sequence>
 <xs:attribute name="id" type="cc:idCourseType" use="required"/>
 <xs:attribute name="semesterHours" type="xs:decimal" use="required"/>
 <xs:attribute name="languageCertificate" type="cc:certificateType" use="required"/>
 </xs:restriction>
 </xs:complexContent>
</xs:complexType>

```

# Benutzerdefinierte Datentypen

<xs:complexType> – Typsubstitution im Instanzdokument

- Statisch
- Dynamisch
  - Festlegung des abgeleiteten Datentyps im XML-Dokument über Attribut **type** aus dem XML Schema Instance (**xsi**) Namensraum

**CourseCatalog.xml**

```

<?xml version="1.0" encoding="UTF-8"?>
<CourseCatalog>
 ...
 <Course id="cID_7555" semesterHours="1" language="en" semester="6">
 <Title>Intercultural Communications</Title>
 <Description>Different types of fake news.</Description>
 </Course>
 ...
 <Course xsi:type="InternationalCourseType" id="cID_8840" semesterHours="2">
 <language="en" semester="2" languageCertificate="TOEFL">
 <Title>Data modelling and database design</Title>
 <Description>Introduction to database design</Description>
 <Prerequisites>At least 15 ECTS in CS required</Prerequisites>
 </Course>
 ...
</CourseCatalog>

```

Element Course mit Datentyp CourseType

Hinweis für Schema-Prozessor

Element Course mit Datentyp InternationalCourseType  
Erweiterung um Prerequisites und languageCertificate

**CourseCatalog.xsd**

```

<xsd:element name="Course" type="cc:CourseType"/>

```

# Inhalt

- Einführung
  - Elemente und Attribute
  - Vordefinierte Datentypen
  - Benutzerdefinierte Datentypen
  - **Schlüssel und Schlüsselreferenzen**
  - Modularisierung und Komposition
  - Modellierungsmuster
- 
- Anhang I: DTD versus XML Schema
  - Anhang II: Facetten
  - Anhang III: Entwurfsrichtlinien
  - Anhang IV: Annotationen
  - Anhang V: XML Schema 1.1 – Erweiterungen

# Schlüssel und Schlüsselreferenzen 1/2

- Eigenschaften eines **Schlüssels** `<xs:key>`
  - Wert(kombination) muss eindeutig sein
  - Wert muss vorhanden sein
- Als **Schlüsselkomponenten** können definiert werden `<xs:field>`
  - Elemente (nur einfache Datentypen)
  - Attribute
  - Kombinationen von Elementen u. Attributen
- **Gültigkeitsbereich** kann definiert werden `<xs:selector>`
- **Referenz** auf Schlüssel `<xs:keyref>`
- Weiters können Elemente, Attribute bzw. Kombinationen davon als eindeutig spezifiziert werden `<xs:unique>`
  - Wert(kombination) muss eindeutig sein
  - Wert muss nicht vorhanden sein

# Schlüssel und Schlüsselreferenzen 2/2

Beispiel: <xs:key>

CourseCatalog.xsd

```

<xs:element name="CourseCatalog" type="cc:CourseCatalogType">
 <xs:key name="courseIdKey">
 <xs:selector xpath="cc:DegreeProgramme/cc:Course"/>
 <xs:field xpath="@id"/>
 </xs:key>
 <xs:keyref name="refCourseIdKey" refer="cc:courseIdKey">
 <xs:selector xpath="cc:DegreeProgramme/cc:Course"/>
 <xs:field xpath="@basedOn"/>
 </xs:keyref>
</xs:element>

```

Schlüssel

Referenz

Schlüssel innerhalb von <CourseCatalog> eindeutig!

Definition von **Schlüssel** und **Referenz** müssen immer gemeinsam und lokal zu einem Element erfolgen!

CourseCatalog.xml

```

<CourseCatalog ... >
 <DegreeProgramme code="0307" name="Software Engineering" ...>
 <Course id="cID_8314" ...>
 ...
 </Course>
 <Course id="cID_8315" basedOn="cID_8314" ...>
 ...
 </Course>
 ...
 </DegreeProgramme>
</CourseCatalog>

```

# Inhalt

- Einführung
  - Elemente und Attribute
  - Vordefinierte Datentypen
  - Benutzerdefinierte Datentypen
  - Schlüssel und Schlüsselreferenzen
  - **Modularisierung und Komposition**
  - Modellierungsmuster
- 
- Anhang I: DTD versus XML Schema
  - Anhang II: Facetten
  - Anhang III: Entwurfsrichtlinien
  - Anhang IV: Annotationen
  - Anhang V: XML Schema 1.1 – Erweiterungen

# Modularisierung und Komposition 1/8

## Innerhalb eines Schemas

- Wurzelement `<xs:schema>` eines XML Schemas enthält alle Schema-Komponenten (Datentypen, Elemente, Attribute,...) als Kindelemente
- Globale versus lokale Deklaration von Datentypen, Elementen und Attributen
- Beziehungen: lokal geschachtelte Elemente versus über Referenzen (`<xs:keyref>` oder `<xs:ref>`) realisierte Beziehungen
- Gruppierung von Elementen und Attributen
  - Zweck: Modularisierung, Wiederverwendung

JETZT NEU

# Modularisierung und Komposition 2/8

## Innerhalb eines Schemas

### ■ Elementgruppe

- Zusammenfassung von Elementen zu einer Elementgruppe
- Referenzierung über Gruppennamen
- Einschränkung: keine rekursiven Bezüge erlaubt!

```
<xs:schema ...>
 <xs:group name="CourseDescriptionGroup">
 <xs:sequence>
 <xs:element name="Title" type="xs:string"/>
 <xs:element name="Description" type="cc:DescriptionType"/>
 <xs:element name="Credit" type="cc:CreditType"/>
 <xs:element name="CourseType" type="cc:CourseTypeType"/>
 </xs:sequence>
 </xs:group>
 <xs:complexType name="CourseType">
 <xs:sequence>
 <xs:group ref="cc:CourseDescriptionGroup" minOccurs="1" maxOccurs="1"/>
 <xs:element name="Date" type="cc:DateType"/>
 ...
 </xs:sequence>
 </xs:complexType>
 ...
</xs:schema>
```

# Modularisierung und Komposition 3/8

## Innerhalb eines Schemas

### ■ Attributgruppe

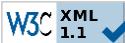
- Zusammenfassung von Attributen zu einer Attributgruppe
- Referenzierung über Gruppennamen
- Bessere Wiederverwendbarkeit

```
<xs:schema ...>
 <xs:attributeGroup name="IdentifierDegreeProgrammeGroup">
 <xs:attribute name="code" type="cc:codeType" use="required"/>
 <xs:attribute name="type" type="xs:string" use="required"/>
 <xs:attribute name="abbreviation" type="xs:string" use="optional"/>
 </xs:attributeGroup>

 <xs:complexType name="DegreeProgrammeType">
 <xs:sequence> ... </xs:sequence>
 <xs:attributeGroup ref="cc:IdentifierDegreeProgrammeGroup"/>
 </xs:complexType>
 ...
</xs:schema>
```

# Modularisierung und Komposition 4/8

## Aufbau von Schema-Bibliotheken

- Einbindung anderer Schemata durch
  - <xs:include>
  - <xs:redefine>
  - <xs:import>
  -  <xs:override> ►Anhang V
- <xs:include>, <xs:redefine> und <xs:import> Elemente müssen als Subelemente von <xs:schema> vor anderen Deklarationen angeführt werden

# Modularisierung und Komposition 5/8

## Schema-Inklusion

### ■ Inkludieren eines Schemas - <xs:include>

- Inkludiertes Schema muss den gleichen Namensraum wie das inkludierende Schema oder keinen Namensraum haben
- Komponenten des inkludierten Schemas können so verwendet werden, als wären sie direkt im inkludierenden Schema deklariert worden

Catalog.xsd

```
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.fh-ooe.at/CourseCatalog"
 xmlns:cc="http://www.fh-ooe.at/CourseCatalog" ...>

 <xs:include schemaLocation="Course.xsd"/>

 <xs:element name="CourseCatalog" .../>
 ...
 <xs:complexType name="DegreeProgrammType">
 <xs:sequence>
 <xs:element name="Course" type="cc:CourseType"/>
 ...
 </xs:complexType>
</xs:schema>
```

Course.xsd

```
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.fh-ooe.at/CourseCatalog"
 xmlns:cc="http://www.fh-ooe.at/CourseCatalog" ...>

 ...
 <xs:complexType name="CourseType">
 <xs:sequence>
 <xs:element name="Title" type="xs:string"/>
 ...
 </xs:sequence>
 ...
 </xs:complexType>
 ...
</xs:schema ...>
```

# Modularisierung und Komposition 6/8

## Schema-Inklusion mit Ableitung

### ■ Inkludieren eines Schemas - <xs:include>

- Ableitung durch Erweiterung
- Ableitung durch Einschränkung

Catalog.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
 targetNamespace="http://www.fh-ooe.at/CourseCatalog" ...>

 <xs:include schemaLocation="Course.xsd"/>

 <xs:complexType name="InternationalCourseType">
 <xs:complexContent>
 <xs:extension base="cc:CourseType">
 <xs:sequence>
 <xs:element name="Prerequisites" type="xs:string" minOccurs="0" maxOccurs="1"/>
 </xs:sequence>
 <xs:attribute name="languageCertificate" type="cc:certificateType" use="required"/>
 </xs:extension>
 </xs:complexContent>
 </xs:complexType>
</xs:schema>
```

# Modularisierung und Komposition 7/8

## Schema-Inklusion mit Redefinition

- Inkludieren u. Redefinieren eines Schemas - <**xs:redefine**>
  - Gleiche Funktionalität wie <**xs:include**>
  - Zusätzlich können inkludierte Komponenten
    - <**xs:simpleType**> (Einschränkung)
    - <**xs:complexType**> (Einschränkung und Erweiterung)
    - <**xs:group**> <**xs:attributeGroup**> (Einschränkung und Erweiterung)
  - neu definiert werden

Catalog.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
 targetNamespace="http://www.fh-ooe.at/CourseCatalog" ...>
 <xs:redefine schemaLocation="Course.xsd">
 <xs:simpleType name="dayType">
 <xs:restriction base="cc:dayType"> <!-- restrict MON to SAT -->
 <xs:enumeration value="MON"/>
 ...
 <xs:enumeration value="SAT"/>
 </xs:restriction>
 </xs:simpleType>
 </xs:redefine>
 <xs:element name="CourseCatalog" ...>
 ...
</xs:schema>
```

# Modularisierung und Komposition 8/8

## Schema-Import

### ■ Importieren eines Schemas - <xs:import>

- Importiertes Schema kann einen beliebigen Namensraum (ungleich dem aktuellen Namensraum) oder keinen Namensraum haben

Catalog.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
 xmlns:cr="http://www.fh-ooe.at/CourseReservation" ←
 targetNamespace="http://www.fh-ooe.at/CourseCatalog" ...>

 <xs:import
 namespace="http://www.fh-ooe.at/CourseReservation" →
 schemaLocation="CourseReservations.xsd">
 </xs:import

 <xs:element name="Reservations" type="cr:CourseReservationsType"/>

 <xs:element name="CourseCatalog" .../>
 ...
</xs:schema>
```

# Inhalt

- Einführung
- Elemente und Attribute
- Vordefinierte Datentypen
- Benutzerdefinierte Datentypen
- Schlüssel und Schlüsselreferenzen
- Modularisierung und Komposition
- **Modellierungsmuster**
  
- Anhang I: DTD versus XML Schema
- Anhang II: Facetten
- Anhang III: Entwurfsrichtlinien
- Anhang IV: Annotationen
- Anhang V: XML Schema 1.1 – Erweiterungen

# Modellierungsmuster 1/8

## Beziehungen / Global vs. Lokal / Element vs. Typ

- Beziehungen
  - Realisierung durch geschachtelte Elemente oder über Referenzen
- Globale Element/Attribut-Deklarationen
  - Voraussetzung für Wiederverwendung in gleichem/anderem Schema
  - Wurzelement muss immer global sein
- Lokale Element/Attribut-Deklarationen
  - falls Deklaration nur im Zusammenhang mit deklarierten Typ sinnvoll
- Lokale Elementdeklarationen
  - können mit unterschiedlicher Struktur aber gleichem Namen in unterschiedlichen Typen auftreten
- Lokale Attributdeklarationen
  - sinnvoll, da Attribute meist eng an Elemente gekoppelt sind
- **Entwurfsmuster**
  - Russische Matrjoschka (Russian Doll)
  - Salamischeiben (Salami Slice)
  - Jalousien (Venetian Blind)
  - Garten Eden (Garden of Eden)

vgl. Roger Costello: Schema structure patterns, [www.xfront.com/GlobalVersusLocal.pdf](http://www.xfront.com/GlobalVersusLocal.pdf)

# Modellierungsmuster 2/8

- Entwurfsmuster unterscheiden sich in der Sichtbarkeit der Elementdeklarationen und Typdefinitionen

|                 |                | Elementdeklarationen  |                       |
|-----------------|----------------|-----------------------|-----------------------|
|                 |                | Lokal                 | Global                |
| Typdefinitionen | Anonym/Lokal   | <i>Russian Doll</i>   | <i>Salami Slice</i>   |
|                 | Benannt/Global | <i>Venetian Blind</i> | <i>Garden of Eden</i> |

- Globale Elementdeklarationen und Typdefinitionen
  - Direkt unter dem Wurzelement `<xs:schema>`
  - Wiederverwendung in anderen Schemata durch `<xs:include>`, `<xs:redefine>` und `<xs:import>` möglich
- Lokale Elementdeklarationen und Typdefinitionen
  - Elementdeklarationen und Typdefinitionen sind verschachtelt
  - Wiederverwendbarkeit ist nur sehr eingeschränkt gegeben

# Modellierungsmuster 3/8

## Russische Matrjoschka (Russian Doll Design)



- Elementdeklarationen ineinander schachteln
  - Ein einziges globales Element
    - sonst nur lokale Deklarationen
  - vermeidet globale Typdefinitionen
- Vorteile
  - Struktur offensichtlich (entspricht Struktur des XML-Dokuments)
  - Vermeidung von Seiteneffekten
    - restriktive Strukturen möglich
- Nachteile
  - tiefe Schachtelungstiefe der Elemente (Redundanzen)
  - keine Wiederverwendung von Deklarationen und Typen
  - keine Erweiterbarkeit (Ableitung)
  - nur eine XML-Schema-Datei möglich

### CourseCatalog.xsd (Ausschnitt)

```

<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
 targetNamespace="http://www.fh-ooe.at/CourseCatalog"
 elementFormDefault="qualified" attributeFormDefault="unqualified">

 <xs:element name="CourseCatalog">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="DegreeProgramme" maxOccurs="unbounded">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Course" maxOccurs="unbounded">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Title" type="xs:string"/>
 <xs:sequence>
 <xs:attribute name="semester" type="xs:decimal" use="required"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 </xs:sequence>
 <xs:attribute name="code" use="required">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:pattern value="[\d]{4}"/>
 </xs:restriction>
 </xs:simpleType>
 </xs:attribute>
 <xs:attribute name="name" type="xs:string" use="required"/>
 <xs:attribute name="abbreviation" type="xs:string" use="optional"/>
 </xs:complexType>
 </xs:element>
 </xs:sequence>
 <xs:attribute name="term" use="required">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:enumeration value="summer"/>
 <xs:enumeration value="winter"/>
 </xs:restriction>
 </xs:simpleType>
 </xs:attribute>
 </xs:complexType>
 </xs:element>
</xs:schema>

```

# Modellierungsmuster 4/8

## Salamischeiben-Stil (Salami Slice)



- Globale Elementdeklarationen
  - Verwendung globaler Elemente per Referenz (**ref**-Attribut)
  - jedes globale Element kann Wurzelement sein
- Lokale Typdeklarationen
- Vorteile
  - Wiederverwendung von globalen Elementdeklarationen
  - mehrere Wurzelemente möglich
- Nachteile
  - große Menge an globalen Elementen (ev. unübersichtlicher)
  - Seiteneffekte bei Änderungen möglich
  - keine Erweiterbarkeit (Ableitung)

### CourseCatalog.xsd (Ausschnitt)

```

<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:cc="http://www.fl-ooe.at/CourseCatalog"
 targetNamespace="http://www.fl-ooe.at/CourseCatalog"
 elementFormDefault="qualified" attributeFormDefault="unqualified">

 <xs:element name="CourseCatalog">
 <xs:complexType>
 <xs:sequence>
 | <xs:element ref="cc:DegreeProgramme" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute ref="cc:term" use="required"/>
 </xs:complexType>
 </xs:element>

 <xs:element name="DegreeProgramme">
 <xs:complexType>
 <xs:sequence>
 | <xs:element ref="cc:Course" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute ref="cc:code" use="required"/>
 </xs:complexType>
 </xs:element>

 <xs:element name="Course">
 <xs:complexType>
 <xs:sequence>
 | <xs:element name="Title" type="xs:string"/>
 </xs:sequence>
 <xs:attribute name="semester" type="xs:decimal" use="required"/>
 </xs:complexType>
 </xs:element>

 <xs:attribute name="term">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:enumeration value="summer"/>
 <xs:enumeration value="winter"/>
 </xs:restriction>
 </xs:simpleType>
 </xs:attribute>
 ...
</xs:schema>
```

# Modellierungsmuster 5/8

## Jalousien-Design (Venetian Blinds Design)



### Globale Typdeklarationen

- Elemente sind lokal deklariert (Ausnahme Wurzelement)

### Vorteile

- Wiederverwendung von Typen
  - zu jedem Element und Attribut existiert ein benannter Typ
  - Typen können aus anderen Schemata importiert werden
- Erweiterbarkeit (Ableitung und `<xs:redefine>`)

### Nachteil

- große Menge an globalen Typen (ev. unübersichtlicher)

#### CourseCatalog.xsd (Ausschnitt)

```

<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:cc="http://www.fh-oeoe.at/CourseCatalog"
 targetNamespace="http://www.fh-oeoe.at/CourseCatalog"
 elementFormDefault="qualified" attributeFormDefault="unqualified">

 <xs:element name="CourseCatalog" type="cc:CourseCatalogType"></xs:element>

 <xs:complexType name="CourseCatalogType">
 <xs:sequence>
 <xs:element name="DegreeProgramme" type="cc:DegreeProgrammeType"
 maxOccurs="unbounded"/>
 <xs:sequence>
 <xs:attribute name="term" type="cc:termType" use="required"/>
 </xs:complexType>

 <xs:complexType name="DegreeProgrammeType">
 <xs:sequence>
 <xs:element name="Course" type="cc:CourseType" maxOccurs="unbounded"/>
 <xs:sequence>
 <xs:attribute name="code" type="cc:codeType" use="required"/>
 </xs:complexType>

 <xs:complexType name="CourseType">
 <xs:sequence>
 <xs:element name="Title" type="xs:string"/>
 <xs:sequence>
 <xs:attribute name="semester" type="xs:decimal" use="required"/>
 </xs:complexType>

 <xs:simpleType name="termType">
 <xs:restriction base="xs:string">
 <xs:enumeration value="summer"/>
 <xs:enumeration value="winter"/>
 </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="codeType">
 <xs:restriction base="xs:string">
 <xs:pattern value="[\d]{4}"/>
 </xs:restriction>
 </xs:simpleType>
 </xs:complexType>
 </xs:sequence>
 </xs:complexType>
</xs:schema>

```

# Modellierungsmuster 6/8

## Vergleich

► Anhang III  
Entwurfsrichtlinien

- *Russian Doll* für **restiktive Strukturen**
  - Struktur der Instanz stark durch Schema vorgegeben
- *Salami Slice* für **variable Strukturen**
  - Struktur der Instanz kann stark schwanken, da aus verschiedenen Wurzelementen ausgewählt werden kann
- *Venetian Blinds* ebenfalls für **variable Strukturen**
  - Struktur der Instanz kann schwanken, falls Typvererbung genutzt wird
- In der Praxis Mischformen, bspw. **Garden of Eden**

# Modellierungsmuster 7/8

## Mischform – Garten Eden (Garden of Eden)



- Einflüsse
  - *Venetian Blinds* → alle Typdefinitionen global
  - *Salami Slice* → alle Elementdefinitionen global
- Jedes Element wird unter dem Wurzelement definiert
- Vorteile
  - Schemata sind stark wiederverwendbar, da alle Elemente und Typen global definiert wurden.
  - Sinnvoll vor allem bei der Entwicklung von Bibliotheken mit umfangreichem Anwendungsbereich (oder wenn der Anwendungsbereich vorab noch nicht genau bekannt ist)
- Nachteile
  - Viele unterschiedliche Wurzelemente möglich
  - Datenkapselung durch die globalen Elemente/Typen schwierig
  - Oft schwierig zu lesen und zu interpretieren

# Modellierungsmuster 8/8

## Mischform – Garten Eden (Garden of Eden)

CourseCatalog.xsd (Ausschnitt)

```

<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
 targetNamespace="http://www.fh-ooe.at/CourseCatalog"
 elementFormDefault="qualified" attributeFormDefault="unqualified">

 <xs:element name="CourseCatalog" type="cc:CourseCatalogType"></xs:element>

 <xs:complexType name="CourseCatalogType">
 <xs:sequence>
 <xs:element ref="cc:DegreeProgramme" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="term" type="cc:termType" use="required"/>
 </xs:complexType>

 <xs:element name="DegreeProgramme" type="cc:DegreeProgrammeType"/>

 <xs:complexType name="DegreeProgrammeType">
 <xs:sequence>
 <xs:element name="Course" type="cc:CourseType" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute ref="cc:code" use="required"/>
 </xs:complexType>

 <xs:complexType name="CourseType">
 <xs:sequence>
 <xs:element name="Title" type="xs:string"/>
 </xs:sequence>
 <xs:attribute name="semester" type="xs:decimal" use="required"/>
 </xs:complexType>

 <xs:simpleType name="termType">
 <xs:restriction base="xs:string">
 <xs:enumeration value="summer"/>
 <xs:enumeration value="winter"/>
 </xs:restriction>
 </xs:simpleType>

 <xs:attribute name="code">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:pattern value="[d]{4}"/>
 </xs:restriction>
 </xs:simpleType>
 </xs:attribute>
</xs:schema>
```



# Inhalt

- Einführung
- Elemente und Attribute
- Vordefinierte Datentypen
- Benutzerdefinierte Datentypen
- Schlüssel und Schlüsselreferenzen
- Modularisierung und Komposition
- Modellierungsmuster
  
- Anhang I: DTD versus XML Schema
- Anhang II: Facetten
- Anhang III: Entwurfsrichtlinien
- Anhang IV: Annotationen
- Anhang V: XML Schema 1.1 – Erweiterungen

## Anhang I

---

# DTD versus XML Schema

---

Gegenüberstellung

# Vergleich DTD – XML Schema 1/6

## Allgemeines

|                    | <b>DTD</b>                | <b>XML Schema</b>  |
|--------------------|---------------------------|--------------------|
| <b>Syntax</b>      | eigene Syntax             | benutzt XML Syntax |
| <b>Struktur</b>    | relativ einfache Struktur | komplexe Struktur  |
| <b>Namensräume</b> | x                         | ✓                  |

# Vergleich DTD – XML Schema 2/6

## Elemente

|                               | DTD                                                   | XML Schema                                        |
|-------------------------------|-------------------------------------------------------|---------------------------------------------------|
| <b>Defaultwerte</b>           | x                                                     | ✓                                                 |
| <b>Definition des Inhalts</b> | Text, Elemente, gemischter Inhalt (Text und Elemente) | einfache Typen, komplexe Typen                    |
| <b>Reihenfolge</b>            | mittels "," definierbar                               | < <b>xs:sequence</b> >                            |
| <b>Keine Reihenfolge</b>      | x                                                     | < <b>xs:all</b> >                                 |
| <b>Alternative</b>            | mittels " " definierbar                               | < <b>xs:choice</b> >                              |
| <b>Kardinalität</b>           | "?", "*", "+"                                         | <b>minOccurs</b> und <b>maxOccurs</b> (flexibler) |

# Vergleich DTD – XML Schema 3/6

## Attribute

|                     | DTD | XML Schema |
|---------------------|-----|------------|
| <b>Defaultwerte</b> | ✓   | ✓          |
| <b>Optionalität</b> | ✓   | ✓          |

# Vergleich DTD – XML Schema 4/6

## Datentypen

|                                 | <b>DTD</b>                                                                              | <b>XML Schema</b>                                                                      |
|---------------------------------|-----------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| <b>Vordefinierte Datentypen</b> | wenige Datentypen;<br>nur String-Datentypen, z.B.<br><b>CDATA, ID, ...</b>              | zahlreiche Datentypen;<br>Vielfalt von Datentypen, z.B.<br><b>string, integer, ...</b> |
| <b>Benutzerdef. Datentypen</b>  | ✗                                                                                       | ✓                                                                                      |
| <b>Wertebereiche</b>            | durch Aufzählen aller Werte<br>(nur für Attribute)                                      | verschiedenste Möglichkeiten<br><b>&lt;xs:length&gt;, ...</b>                          |
| <b>Muster für Datentypen</b>    | eingeschränkt u. kompliziert<br>realisierbar (z.B. durch<br>Kardinalitätsspezifikation) | mittels <b>&lt;xs:pattern&gt;</b><br>möglich                                           |

# Vergleich DTD – XML Schema 5/6

## Vererbung

|                                                   | DTD | XML Schema                                                                   |
|---------------------------------------------------|-----|------------------------------------------------------------------------------|
| Ableiten von vordef. und einfachen Datentypen     | ✗   | mittels <code>&lt;xs:base&gt;</code>                                         |
| Ableiten von komplexen Datentypen (Erweiterung)   | ✗   | mittels <code>&lt;xs:base&gt;</code> und <code>&lt;xs:extension&gt;</code>   |
| Ableiten von komplexen Datentypen (Einschränkung) | ✗   | mittels <code>&lt;xs:base&gt;</code> und <code>&lt;xs:restriction&gt;</code> |

# Vergleich DTD – XML Schema 6/6

- Die wichtigsten Vorteile von DTD's:
  - schnell und einfach zu erstellen
    - zur Erstellung einfacher Dokumente gut geeignet
- Die wichtigsten Vorteile von XML Schema:
  - zahlreiche vordefinierte Datentypen
  - eigene Datentypen definierbar (Vererbungshierarchie)
  - integrieren Namensräume
  - keine eigene Syntax, sondern selbst XML-Sprache
    - zum Modellieren komplexer Dokumente gut geeignet

## Anhang II

---

# Facetten

---

Wertebereichseinschränkung bei  
einfachen Datentypen

---

# Facetten 1/2

## Einschränkung des Wertebereiches

|                       |                                                                                                                                               |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <code>string</code>   | <code>length, minLength, maxLength, pattern, enumeration, whitespace, assertion</code>                                                        |
| <code>boolean</code>  | <code>pattern, whiteSpace, assertion</code>                                                                                                   |
| <code>float</code>    | <code>pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, assertion</code>                              |
| <code>double</code>   | <code>pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, assertion</code>                              |
| <code>decimal</code>  | <code>totalDigits, fractionDigits, pattern, whiteSpace, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive, assertion</code> |
| <code>duration</code> | <code>pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, assertion</code>                              |
| <code>dateTime</code> | <code>pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, assertion</code>                              |
| <code>time</code>     | <code>pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, assertion</code>                              |
| <code>date</code>     | <code>pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, assertion</code>                              |

vgl. <https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/#built-in-datatypes>

# Facetten 2/2

## Einschränkung des Wertebereiches

|              |                                                                                                     |
|--------------|-----------------------------------------------------------------------------------------------------|
| gYearMonth   | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, assertion |
| gYear        | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, assertion |
| gMonthDay    | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, assertion |
| gDay         | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, assertion |
| gMonth       | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, assertion |
| hexBinary    | length, minLength, maxLength, pattern, enumeration, whiteSpace, assertion                           |
| base64Binary | length, minLength, maxLength, pattern, enumeration, whiteSpace, assertion                           |
| anyURI       | length, minLength, maxLength, pattern, enumeration, whiteSpace, assertion                           |
| QName        | length, minLength, maxLength, pattern, enumeration, whiteSpace, assertion                           |
| NOTATION     | length, minLength, maxLength, pattern, enumeration, whiteSpace, assertion                           |

vgl. <https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/#built-in-datatypes>

## **Anhang III**

---

# **Entwurfsrichtlinien**

---

---

# Entwurfsrichtlinien 1/6

- Verwende globale und lokale Elementdeklarationen
  - Globale Elementdeklarationen können mithilfe von Verweisen in anderen Schemateilen oder anderen Schemadokumenten wieder verwendet werden.
  - Globale Elementdeklarationen können in Ersetzungsgruppen (substitution group) verwendet werden.
  
- Lokale Elementdeklarationen können mit gleichen Namen in unterschiedlichen Typen auftreten.
- Lokale Elementdeklarationen sollten dann eingesetzt werden, wenn die Elementdeklaration nur im Zusammenhang mit dem deklarierten Typ sinnvoll ist.

# Entwurfsrichtlinien 2/6

- Verwende globale und lokale Attributdeklarationen
  - Globale Attributdeklarationen können in mithilfe von Verweisen in anderen Schemateilen oder anderen Schemadokumenten wieder verwendet werden.
  - Lokale Attributdeklarationen sollten dann eingesetzt werden, wenn die Elementdeklaration nur im Zusammenhang mit dem deklarierten Typ sinnvoll ist.
  - Lokale Attributdeklarationen sind vorzuziehen, da Attribute gewöhnlich eng an die ihnen übergeordneten Elemente gekoppelt sind.

# Entwurfsrichtlinien 3/6

- Definiere `elementFormDefault` immer als `qualified`
  - Lokale Elemente im XML-Dokument sind somit auch dem Zielnamensraum zuzuordnen (→ Dokumentation)
- Verwende Attributgruppen und Elementgruppen
  - Benannte Auflistung von Attributen/Elementen können an einer einzigen Stelle deklariert werden und ein oder mehrere Schemata können dann darauf verweisen.
- Verwende integrierte einfache Typen (44 Datentypen)
  - Schränke die verwendeten Typen auf eine Menge ein, die bewältigt werden kann.
- Bevorzuge für Identitätseinschränkungen `<xs:key>`, `<xs:keyref>` und `<xs:unique>` gegenüber `ID`/`IDREF`

# Entwurfsrichtlinien 4/6

## ■ Verwende komplexe Typen

- Benannte komplexe Typen ermöglichen Typableitung und Wiederverwendung (im internen und externen Schemadokumenten)
- Anonyme Typen sollten nur dann verwendet werden, wenn Verweise auf den Typ nicht außerhalb der Elementdeklaration benötigt werden und keine Typableitung gebraucht wird.

## ■ Vermeide Standard- oder feste Werte

- Durch Standard- und feste Werte werden neue Daten nach der Prüfung in das XML-Dokument eingefügt und dadurch die Daten verändert.
- Das bedeutet, dass ein XML-Dokument mit einem Schema mit Standardwerten, das nicht geprüft wurde, nicht vollständig ist.

# Entwurfsrichtlinien 5/6

- Verwende Einschränkungen von einfachen Typen
- Verwende Erweiterungen von komplexen Typen
  - Wiederverwendung durch Erweiterung ist eine leistungsstarke Funktion und entspricht den Konzepten der objektorientierten Programmierung.
- Verwende Einschränkungen von komplexen Typen mit Vorsicht
  - Eine Vielzahl von Nuancen der Ableitung durch Einschränkung in komplexen Typen führt oft zu Programmierfehler.
  - Ableitung durch Einschränkung von komplexen Typen entspricht nicht den Konzepten der objektorientierten Programmierung.

# Entwurfsrichtlinien 6/6

- Verwende Platzhalter `any`/`anyAttribute`, um fest definierte Punkte für die Erweiterbarkeit bereitzustellen
- Vermeide Typen- und Gruppenneudefinition mit `<xs:redefine>`
  - Alle Verweise auf den ursprünglichen Typ oder Gruppe in beiden Schemata verweisen auf den neu definierten Typ, während die ursprüngliche Definition verdeckt wird.
  - Verwende `<xs:override>` (XML Schema 1.1), um Element- und Attributdeklarationen, Typdefinitionen sowie Element- und Attributgruppen zu überschreiben/ersetzen
- Mache Typnamen erkennbar
  - Über „`...Typ(e)`“ oder andere Schreibweise
- Validiere Schemata mit mehreren Schemaprozessoren
- Versuche nicht, XML Schema meisterhaft zu beherrschen
  - Das würde Monate dauern!

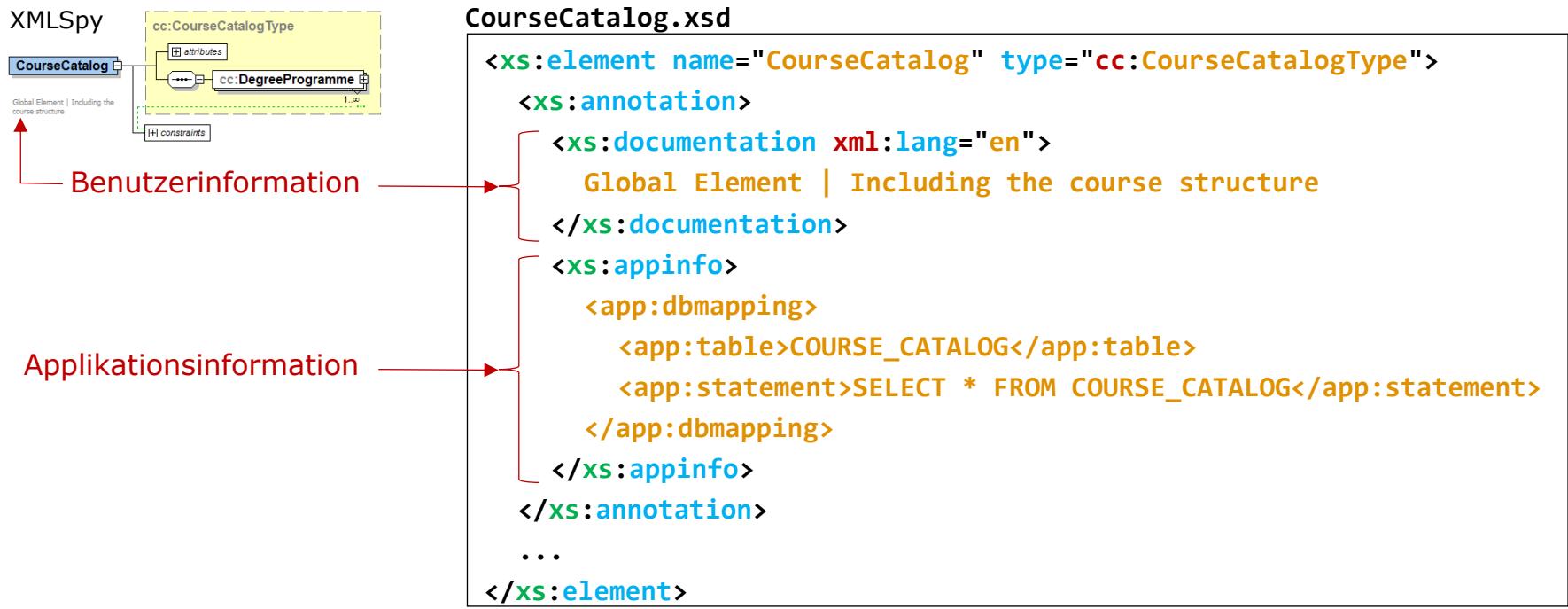
## Anhang IV

---

# Annotationen

# Annotationselement

- `<xs:annotation>` kann allen XML Schema-Elementen als erstes Kindelement hinzugefügt werden und enthält die optionalen Elemente
  - `<xs:documentation>` für menschenlesbare Dokumentation - Benutzer
  - `<xs:appinfo>` für maschinenlesbare Zusatzinformation - Applikation (z.B. Metadaten, Verarbeitungsanweisungen, Programmteile)



## Anhang V

---

# XML Schema Definition Language (XSD) 1.1

---

Erweiterungen

---

Part 1: Structures <https://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/>  
Part 2: Datatypes <https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/>

# XML Schema 1.0 – Schwachstelle

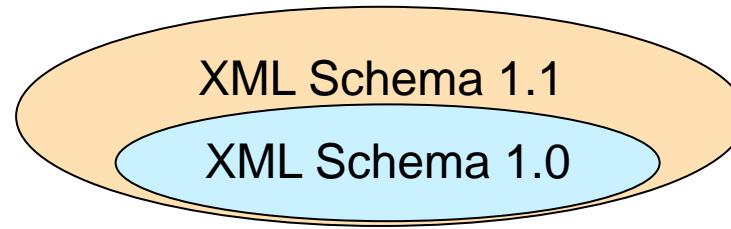
## Co-Occurrence Constraints / Co-Constraints

- Unterstützung von besseren Beschränkungen (constraints) bzw. Zusicherungen (assertions)
  - Komplexere Beschränkungen und Zusicherungen (die mehr als ein Element betreffen) mussten in der jeweiligen Applikationslogik implementiert werden (XML Schema 1.0 bietet hier nur grundlegende Möglichkeiten an).
  - Vergleiche: Schematron und RelaxNG (weitere Schema-Sprachen) – Möglichkeiten deutlich ausgeprägter.

```
<xs:complexType name="intRange">
 <xs:attribute name="min" type="xs:int"/>
 <xs:attribute name="max" type="xs:int"/>
 <xs:assert test="@min <= @max"/> <!-- co-constraint -->
</xs:complexType>
```

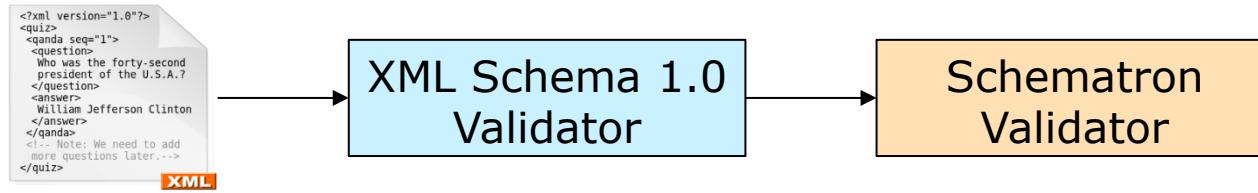
# XML Schema 1.1

- W3C Recommendation seit April 2012
- Neuerungen (Auswahl)
  - Regel-basierte Validierung - Zusicherungen `<xs:assert>` und `<xs:assertion>`
  - Bedingte Typisierung / Datentyp-Alternativen `<xs:alternative>`
  - Standard-Attributgruppen und Schemaweite Attribute
  - Offener Inhalt `<xs:openContent>` `<xs:defaultOpenContent>`
  - Attribute an Kindelemente vererben (inheritable)
  - Schemata wiederverwenden über `<xs:override>` und `<xs:error>`
  - Aufweichung der Reihenfolge von Elementen (`all`)
  - Ersetzungsgruppen für Wörter (substitution)
- XML Schema 1.1 baut auf XML Schema 1.0 auf



# XML Schema 1.1 - Regel-basierte Validierung

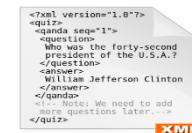
- XML Schema 1.0 unterstützt die Definition und Validierung von Grammatik-Regeln
  - Werden die richtigen Elemente / Attribute verwendet?
  - Werden die richtigen Attribut-Werte verwendet?
  - ...
- XML Schema 1.0 unterstützt keine Definition und Validierung von Geschäftsregeln
  - Zusicherungen innerhalb eines XML-Instanzdokumentes, die vom jeweiligen Geschäftsfall abhängig sind
  - Dafür ist mit XML Schema 1.0 ein weiterer Verarbeitungsschritt notwendig!



# XML Schema 1.1 - Regel-basierte Validierung

## <xs:assert> / <xs:assertion>

- Element- oder Attributwerte können mithilfe von XPath-2.0-Ausdrücken validiert werden.
  - Ähnlich zu XML-Schemasprache *Schematron* oder *RelaxNG*
  - Beispiel: <**xs:assertion test="xpath"**>
    - Attributwert für **test** muss ein gültiger XPath-2.0-Ausdruck sein, der **true** oder **false** zurückgibt.
    - Spezielle Variable **\$value**, um auf den zu prüfenden **simpleContent**-Wert zugreifen zu können
  - Evaluierung erfolgt im Kontext des Elternknotens
  - Zugriff nur auf Nachfahren eines Elementes (u.a. wegen Kompatibilität zu „streaming validation“ – SAX)
  - Zugriff auf andere Dokumente nicht erlaubt (kein **doc(...)**)
- Geschäftsregeln werden dadurch Teil des XML-Schemas und müssen nicht mehr in der jeweiligen Anwendung implementiert werden
  - Vorteil für Lesbarkeit und Wartung



# XML Schema 1.1 - Regel-basierte Validierung

## <xs:assert> / <xs:assertion>

### ■ Assertion für einfachen Datentyp

- <xs:assertion>-Facette
- Zugriff auf den Wert des einfachen Datentyps über \$value

```
<xs:simpleType name="SizeType">
 <xs:restriction base="xs:integer">
 <xs:assertion test="$value != 0"/>
 </xs:restriction>
</xs:simpleType>
```

### ■ Assertion für komplexen Datentyp

- <xs:assert>-Element für Zusicherungen über Element- und Attributwerte
- Zugriff auf Elemente / Attribute über deren Namen

```
<xs:complexType name="PointType">
 <xs:attribute name="X" type="xs:integer"/>
 <xs:attribute name="Y" type="xs:integer"/>
 <!-- XPath-2.0-Ausdruck -->
 <xs:assert test="(@X eq @Y) or (@X lt @Y)"/>
</xs:complexType>
```

# XML Schema 1.1 - Regel-basierte Validierung

## <xs:assert> – Beispiele

```

<xs:complexType name="ProductType">
 <xs:sequence>
 <xs:element name="number" type="xs:integer"/>
 <xs:element name="name" type="xs:string"/>
 <xs:element name="size" type="SizeType"/>
 </xs:sequence>
 <xs:attribute name="dept" type="xs:string"/>
 <xs:assert test="
 (@dept eq 'ELEC' and number gt 500) or
 (string-length(@dept) lt 4))"/>
</xs:complexType>

```

→ <xs:complexType>  
 <xs:assert>-Element

Vererbung möglich – alle  
**test**-Ausdrücke müssen  
**true** sein

Gültig, da **test**-Ausdruck **true** ergibt

```

<Product dept="ELEC">
<number>501</number>
<name>iPhone XS</name>
<size>10</size>
</Product>

```

Nicht gültig, da **string-length(@dept) >= 4**

```

<Product dept="ELECTRONICS">
<number>200</number>
<name>iPhone XS</name>
<size>10</size>
</Product>

```

# XML Schema 1.1 - Regel-basierte Validierung

## <xs:assertion> – Beispiele

```

<xs:simpleType name="DepartmentCodeType">
 <xs:restriction base="xs:token">
 <xs:length value="3"/>
 <xs:assertion test="not(contains($value, 'X'))"/>
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="RestrictedDepartmentCodeType">
 <xs:restriction base="DepartmentCodeType">
 <xs:assertion test="substring($value,2,2) != '00'"/>
 <!-- Assertion-Facette für "simpleType" -->
 </xs:restriction>
</xs:simpleType>

<xs:element name="Department" type="RestrictedDepartmentCodeType"/>

```

<xs:simpleType>  
 <xs:assertion>-Facette

Vererbung möglich → alle  
 test-Ausdrücke müssen  
 true sein

Gültig

<Department>K20</Department>

Nicht gültig

<Department>X20</Department>

# XML Schema 1.1 - Bedingte Typisierung

## <xs:alternative>

- Elementtyp kann, je nach Attribut-Werten im Instanz-Dokument, dynamisch zugewiesen werden
- Elementdeklaration wird um <xs:alternative>-Sequenz erweitert
  - Attribut **test**: Bedingung, die zutreffen muss
  - Attribut **type**: Typ, der dem Element dynamisch zugewiesen wird
- Bedingte Typisierung kann auch über Zusicherungen nachgebaut werden
  - Dabei werden keine vordefinierten Datentypen ausgewählt, sondern bestimmte Nachfahren über XPath-Ausdrücke zugelassen/verboten.

# XML Schema 1.1 - Bedingte Typisierung

## <xs:alternative> – Beispiele

```

<xs:complexType name="PersonType">
 <xs:sequence>
 <xs:element name="Vorname" type="xs:string"/>
 <xs:element name="Nachname" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="kind" type="xs:string"/>
</xs:complexType>

<xs:complexType name="TeacherType">
 <xs:complexContent>
 <xs:extension base="PersonType">
 <xs:sequence>
 <xs:element name="PersonalNummer" type="xs:string"/>
 </xs:sequence>
 </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:element name="Persons">
 <xs:complexType>
 <xs:sequence maxOccurs="unbounded">
 <xs:element name="Person" type="PersonType">
 <xs:alternative test="@kind eq 'teacher'" type="TeacherType"/>
 </xs:element>
 </xs:sequence>
 </xs:complexType>
</xs:element>

```

```

<Person kind="teacher">
 <Vorname>Julian</Vorname>
 <Nachname>Haslinger</Nachname>
 <PersonalNummer>P22080</PersonalNummer>
</Person>

```

} Typ abhängig von  
test-Auswertung

# XML Schema 1.1 - Standard-Attribute und Attributgruppen

- Grundidee: Bestimmte Attribute oder Attributgruppen sollen in einem XML-Dokument bei allen/vielen Elementen verwendet werden.
- Möglichkeiten in XML Schema 1.0:
  - Erstellung einer Attributgruppe und explizite Modellierung zu jedem komplexen Datentypen
  - Alle Datentypen erben von einem „Grunddatentyp“, der nichts außer die Attribute definiert
- Lösung in XML Schema 1.1: Schemaweite Attribute
  1. `<xs:attributeGroup>` definieren
  2. `<xs:schema ... defaultAttributes="Attributgruppe">`
  3. Attribute werden automatisch Teil aller `<xs:complexType>`-Definitionen
- `defaultAttributesApply="false"` zum Deaktivieren für einzelne Elemente

# XML Schema 1.1 - Standard-Attribute und Attributgruppen – Beispiele

```

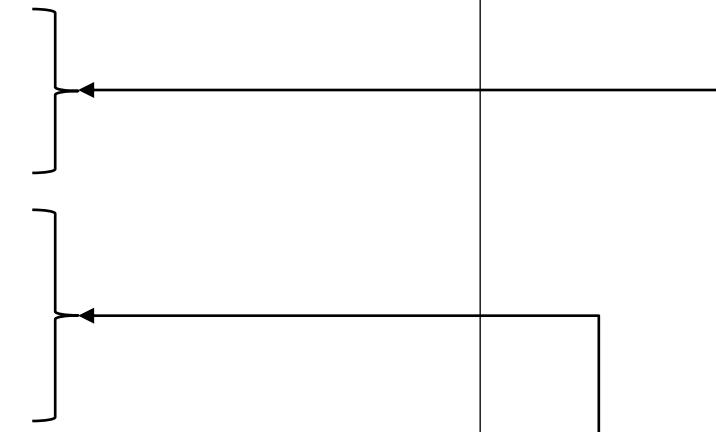
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning" xmlns:fh="http://www.fh-oe.at/xmlschema/xml11"
 targetNamespace="http://www.fh-oe.at/xmlschema/xml11" elementFormDefault="qualified"
 defaultAttributes="fh:DefaultAttributes" vc:minVersion="1.1">

 <xs:attributeGroup name="DefaultAttributes">
 <xs:attribute name="CreatedAt" type="xs:gYear"/>
 <xs:attribute name="CreatedBy" type="xs:string"/>
 </xs:attributeGroup>

 <xs:complexType name="ProductType" defaultAttributesApply="false">
 <xs:simpleContent>
 <xs:extension base="xs:string"/>
 </xs:simpleContent>
 </xs:complexType>

 <xs:complexType name="OrderType">
 <xs:sequence>
 <xs:element name="OrderName"/>
 </xs:sequence>
 <xs:attribute name="OrderNr" type="xs:integer" use="required"/>
 </xs:complexType>

```



```

<Order OrderNr="12" CreatedAt="2017" CreatedBy="JH">
 <OrderName>Order_1</OrderName>
</Order>

```

```
<Product>iPhone XS</Product>
```

- } Element mit **Default**-Attributen
- } Element mit deaktivierten **Default**-Attributen

# XML Schema 1.1 - Offener Inhalt

<xs:openContent> / <xs:defaultOpenContent>

- Grundidee: Erhöhung der XML-Schema-Flexibilität
- XML Schema 1.0: Geänderte Anforderungen an das Schema können oftmals schwierig ohne Hilfe der Schema-Entwickler eingearbeitet werden.
  - starre Schemata
- XML Schema 1.1: Während der Entwicklung eines Schemas bereits „offenen Inhalt“ (Sub-Elemente, die nicht im Inhaltsmodell definiert wurden) einplanen. Die Instanz-Dokumente können dann sofort geändert werden und das Schema kann (muss aber nicht) später nachgezogen (um die neuen Elemente) erweitert werden.
  - Definition auf Schema-Ebene oder für einzelne komplexe Datentypen
  - Definition von offenem Inhalt beinhaltet eine „Element-Wildcard“, z.B.
    - <xs:any namespace="#any" processContents="skip"/>

NS, aus welchem die Elemente stammen dürfen  
z.B. **##any**, **##other**, **##local**, **##targetNamespace**, explizite Angabe

Validierung: **strict**, **lax**, **skip**

# XML Schema 1.1 - Offener Inhalt

<xs:openContent> / <xs:defaultOpenContent>

- Offener Inhalt in komplexen Datentypen (**openContent**)
  - <xs:openContent>-Element als Kindelement oder als Teil von <xs:restriction> / <xs:extension>
  - Attribut: **mode**
    - **interleave** - offener Inhalt kann überall im **complexType** vorkommen
    - **suffix** - offener Inhalt darf nur am Ende einer Sequenz vorkommen
    - **none** - **complexType** verwendet **defaultOpenContent** nicht
- Offener Inhalt im gesamten XML-Schemadokument (**defaultOpenContent**)
  - Oftmals Anforderung, offenen Inhalt für viele komplexe Datentypen zuzulassen.
  - <xs:defaultOpenContent> als Kindelement vom Schema definieren

# XML Schema 1.1 - Offener Inhalt

## <xs:openContent> – Beispiele

```

<xs:complexType name="PersonType">
 <xs:sequence>
 <xs:element name="Vorname" type="xs:string"/>
 <xs:element name="Nachname" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="kind" type="xs:string"/>
</xs:complexType>

<xs:complexType name="TeacherType">
 <xs:complexContent>
 <xs:extension base="PersonType">
 <xs:openContent mode="interleave">
 <xs:any namespace="##any" processContents="skip"/>
 </xs:openContent>
 <xs:sequence>
 <xs:element name="PersonalNummer" type="xs:string"/>
 </xs:sequence>
 </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:element name="Person" type="PersonType">

```

```

<Person>
 <Vorname>Julian</Vorname>
 <MittlererName>Paul</MittlererName>
 <Nachname>Haslinger</Nachname>
 <PersonalNummer>p22080</PersonalNummer>
</Person>

```

Offener Inhalt für Elemente vom Typ **TeacherType**

**interleave**: Offener Inhalt kann im ganzen Element vorkommen



Offener Inhalt: Neues Element <**MittlererName**>



# XML Schema 1.1 - Offener Inhalt

## <xs:defaultOpenContent> – Beispiele

```
<xs:schema ...>
 <xs:defaultOpenContent mode="suffix" appliesToEmpty="true">
 <xs:any/>
 </xs:defaultOpenContent>
...
</xs:schema>
```

Offener Inhalt:  
Kann in jedem Element  
vorkommen;  
auch in Elementen, die „leer“  
definiert  
wurden (**appliesToEmpty**).

```
<Persons>
 <Teacher>
 <Vorname>FirstName Teacher</Vorname>
 <Nachname>LastName Teacher</Nachname>
 <PersonalNummer>P22080</PersonalNummer>
 </Teacher>

 <Student>
 <Vorname>FirstName Student</Vorname>
 <Nachname>LastName Student</Nachname>
 <NeuesElement attribute="Wert"></NeuesElement>
 </Student>
</Persons>
```

**suffix**: Offener Inhalt darf nur  
am Ende eines Elements  
vorkommen (default: **interleave**)

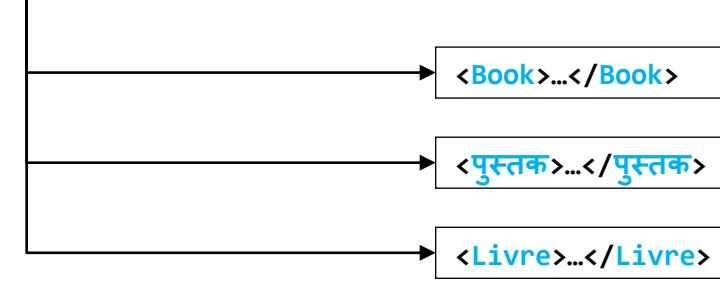
# XML Schema 1.1 – Ersetzungsgruppen

## Erweiterung zu XML Schema 1.0

- Grundidee: Element kann durch ein anderes Element (Namen) ersetzt werden
  - Beispiel: Element <Book> soll im Instanzdokument durch mehrere andere Elemente ersetzt werden können.
    - Beschreibung von beiden bzw. mehreren Elementen
    - Angabe der Elemente in **substitutionGroup**-Attribut von <Book>
  - Beispiel: Angabe von mehreren Ersetzungen

```
<xs:element name="Buch"/>
<xs:element name="Livre"/>

<xs:element name="Book" substitutionGroup="Buch Livre"/>
...
</xs:element>
```



# XML Schema 1.1 - Wiederverwendung

<xs:override> und Datentyp <xs:error>

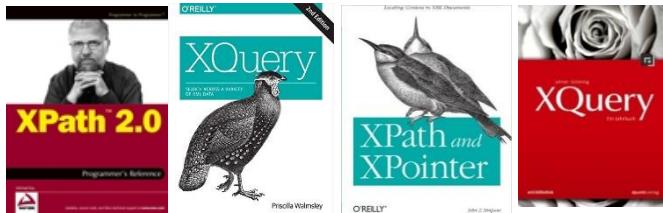
- XML Schema 1.0: Über <xs:redefine> konnten Teile von bestehenden Schemata übernommen und modifiziert werden.
  - Globale Datentypen aus anderem Schema erweitern / einschränken
- XML Schema 1.1: Neues Element <xs:override>
  - Global deklarierte Items aus anderen Schemata können im eigenen Schema überschrieben / ersetzt werden.
    - Elemente, Attribute, Datentypen (<xs:simpleType>, <xs:complexType>), Element- und Attributgruppen
  - Über Datentyp <xs:error> werden Teile des importierten Schemas von der weiteren Verwendung ausgeschlossen

# XQuery

## XML Query Language

Julian Haslinger

```
for $instructorNr in //@instructorNumber
Let $pNr := replace($instructorNr, 'p', '')
Where starts-with($pNr, '22')
Order by $pNr descending
Return $pNr
```



**Der vorliegende Foliensatz basiert vorwiegend auf:**

Kay, M.: XPath 2.0 Programmer's Reference (3<sup>rd</sup> ed.), Wiley, 2004.

Lehner, W., Schöning, H.: XQuery, dpunkt.verlag, 2004.

Simpson, J.: XPath and XPointer, O'Reilly, 2002.

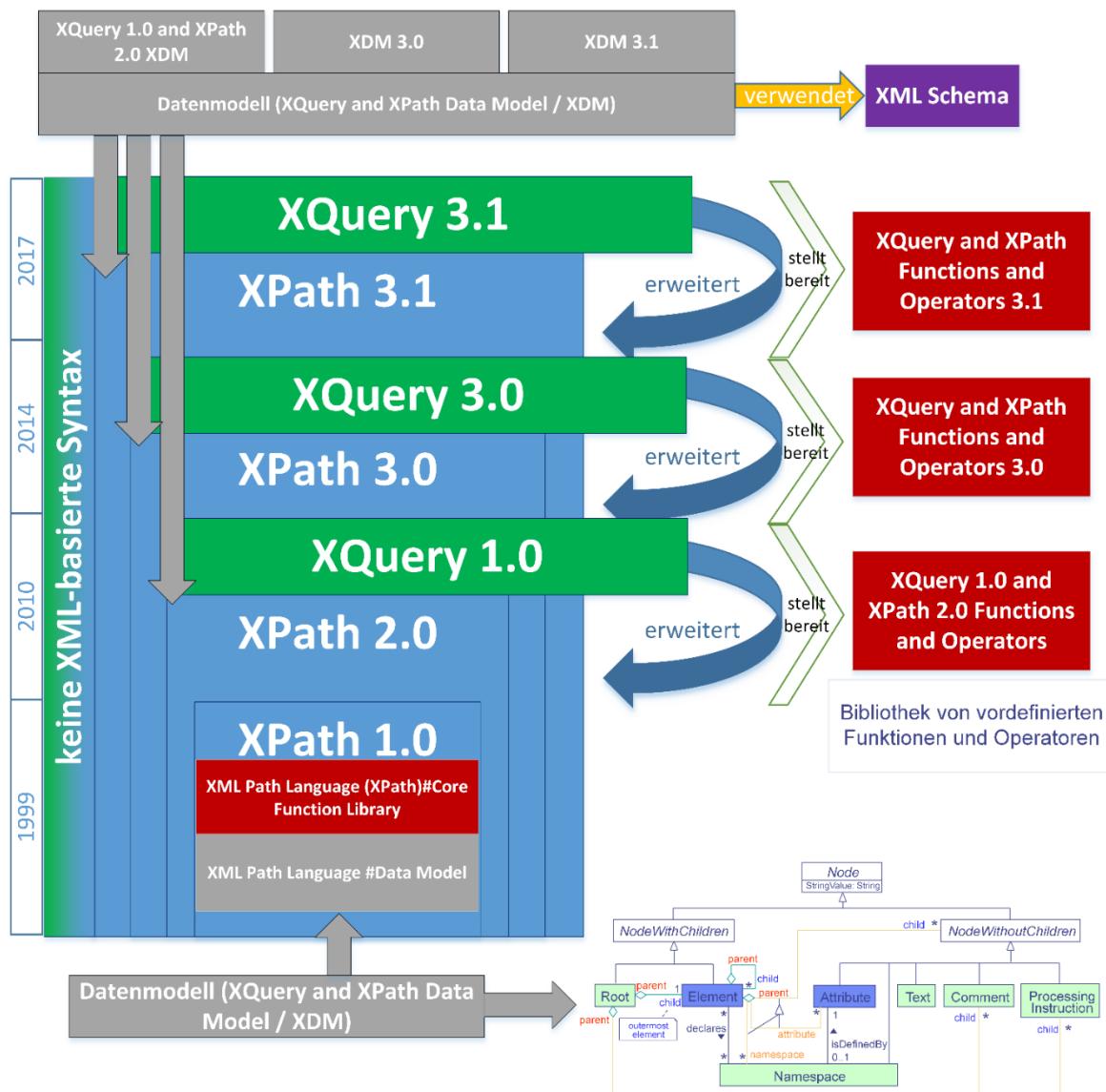
Walmsley, P.: XQuery (2<sup>nd</sup> ed.), O'Reilly, 2015/2017.

# Inhalt

- Einführung
- XQuery 1.0
  - Grundlagen
  - `for`- und `let`-Klausel
  - Hinzufügen von Elementen/Attributen
  - Konditionale Ausdrücke
  - Verbund
  - Quantifizierende Ausdrücke
  - Gruppierung
  - Sortierung und Aggregation
  - Programmstruktur
  - Dokumentation
- Anhang: XQuery Update Facility 1.0

# XQuery

## Versionen



# XQuery 1.0

## Sprachumfang (Auswahl)

(::) XQuery  
@XQuery

When your language is a superset of XPath, you've got a pretty powerful hammer! [twitter.com/notessensei/st...](https://twitter.com/notessensei/status/905426178311376896)  
[pic.twitter.com/ucPQSScuxc](https://pic.twitter.com/ucPQSScuxc)

2:43 PM - Sep 6, 2017

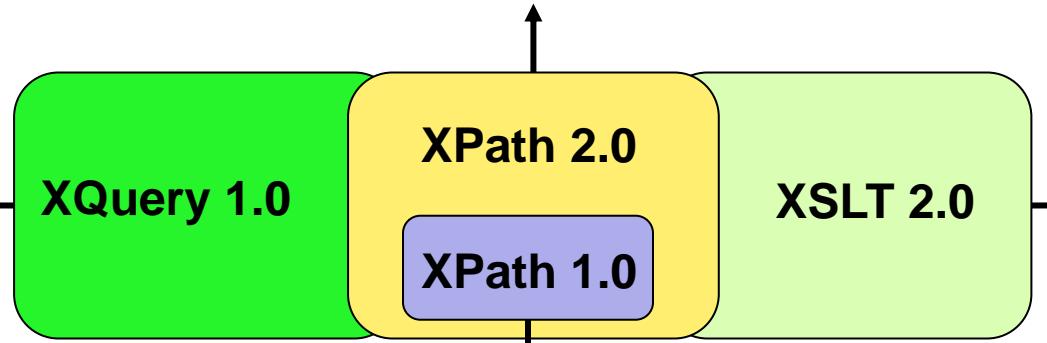
8 See XQuery's other Tweets



- Bedingte Ausdrücke
- Arithmetische Ausdrücke
- Quantifizierende Ausdrücke
- Viele eingebaute Funktionen (> 100)
- Unterstützung von XML-Schema-Datentypen
- Verwendung mehrerer Dokumente
- Datenmodell: Knoten**sequenz**

[<https://twitter.com/XQuery/status/905426178311376896>]

- FLWOR-Ausdrücke
- Restrukturierung
- Element-Konstruktoren
- XQuery-Prolog
- Benutzerdef. Funktionen



- Stylesheets
- Templates
- Literale Ergebnis-elemente
- Benutzerdef. Funktionen

- „SQL des 21. Jahrhunderts“*
- Pfadausdrücke (Extraktion & Reduktion)
  - Knotentests und Prädikate (Selektion)
  - Vergleichsausdrücke
  - Einige eingebaute Funktionen (27)
  - Datenmodell: Knoten**menge**

# Inhalt

- Einführung
- XQuery 1.0
  - Grundlagen
  - **for**- und **let**-Klausel
  - Hinzufügen von Elementen/Attributen
  - Konditionale Ausdrücke
  - Verbund
  - Quantifizierende Ausdrücke
  - Gruppierung
  - Sortierung und Aggregation
  - Programmstruktur
  - Dokumentation
- Anhang: XQuery Update Facility 1.0

80% der Sprachkonzepte  
von XPath 2.0

Ähnlichkeit zu SQL

W3C-Standards:

- XQuery 1.0, März 2007
  - viele Zwischenschritte: 2003/2004/2005
- XQuery 3.0, April 2014
- XQuery 3.1, März 2017

# XQuery

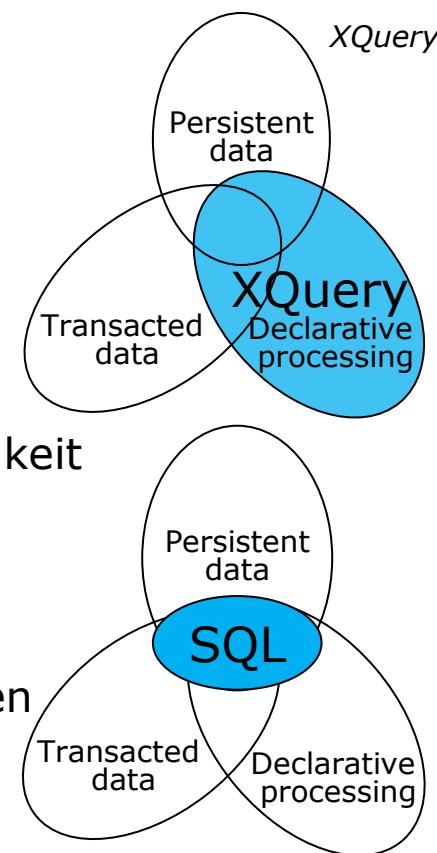
## Grundlagen – Warum XQuery?

### ■ Warum eine Abfragesprache für XML?

- Logische/physische Datenunabhängigkeit
  - Abstraktes Datenmodell gewährleistet Unabhängigkeit von konkreter physischer Speicherung
- Deklarative Programmierung
  - Beschreibe das *WAS*, nicht das *WIE*
  - Gemeinsamkeiten mit funktionalen und imperativen Programmiersprachen sowie mit Abfragesprachen

### ■ Warum eine native Abfragesprache? Warum nicht SQL?

- Eigenheiten von XML müssen berücksichtigt werden
- Hierarchisch, geordnet, Text-basiert, ev. Schema-lose Struktur



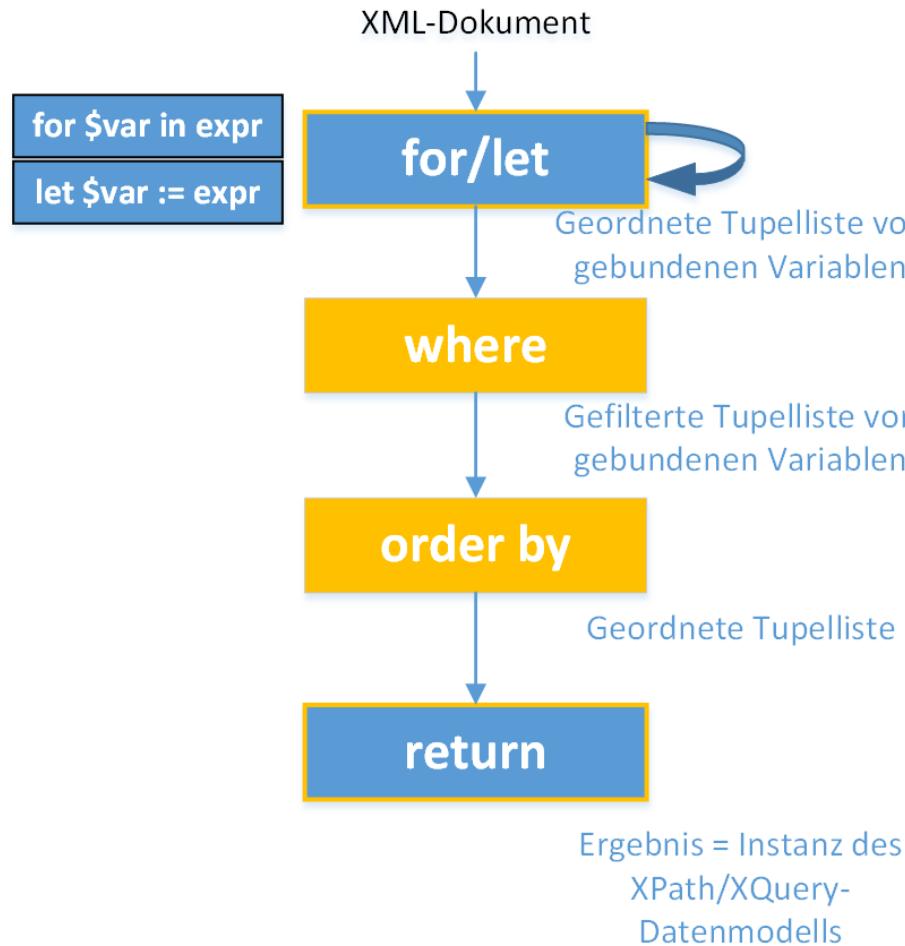
# XQuery 1.0

Grundlagen – XQuery = **80 %** XPath 2.0 + **20 %** ...

- FLWOR (**for-let-where-order-return**)-Ausdruck
  - entspricht **SELECT-FROM-WHERE** in SQL
- Generieren von (u.a.) XML
  - Element- und Attributkonstruktoren, Transformation, ...
- Sortieren u. Gruppieren von Ergebnissen
- Operatoren für Typen
  - Typprüfung während statischer Analyse- und / oder dynamischer Evaluierungs-Phase
- Benutzerdefinierte Funktionen
  - Modularisierung von umfangreichen Anfragen
  - Rekursive Verarbeitung von Daten
- Streng typisiert
  - statisch wie dynamisch

# XQuery 1.0

## Grundlagen – FLWOR ['flower'] Ausdruck 1/2



Iteration (vgl. [FROM](#) in SQL) und Var. Bindung

Variablen werden an Ausdrucksweise gebunden (basiert auf XPath)

Selektion (vgl. [WHERE](#) in SQL)

Filterung von Tupeln auf Basis von Prädikaten (optional)

Sortierung (vgl. [ORDER BY](#) in SQL)

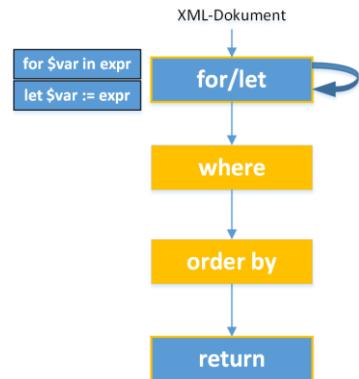
Sortierung von Tupeln auf Basis von Prädikaten (optional)

Generierung (vgl. [SELECT](#) in SQL)

Ergebnisgenerierung (einzelne Knoten, XML-Fragment oder atomare Werte)

# XQuery

## Grundlagen – FLWOR ['flower'] Ausdruck 2/2



Variablenbindung

Funktionsaufruf

*XPath-Ausdruck*

```
for $course in doc("CourseCatalog.xml")//Course
 where contains($course/Title, 'XML')
 return $course/Title
```

Variablenreferenzierung  
(+ XPath)

Vordefinierte Funktion  
(aus „Functions and Operators“)

```
doc("CourseCatalog.xml")//Course[contains>Title, 'XML']/Title
```

### FLWOR-Ausdrücke

unterstützen

- Sortierung
- Verbund und Gruppierung
- Hinzufügen von Elementen und Attributen zum Ergebnis

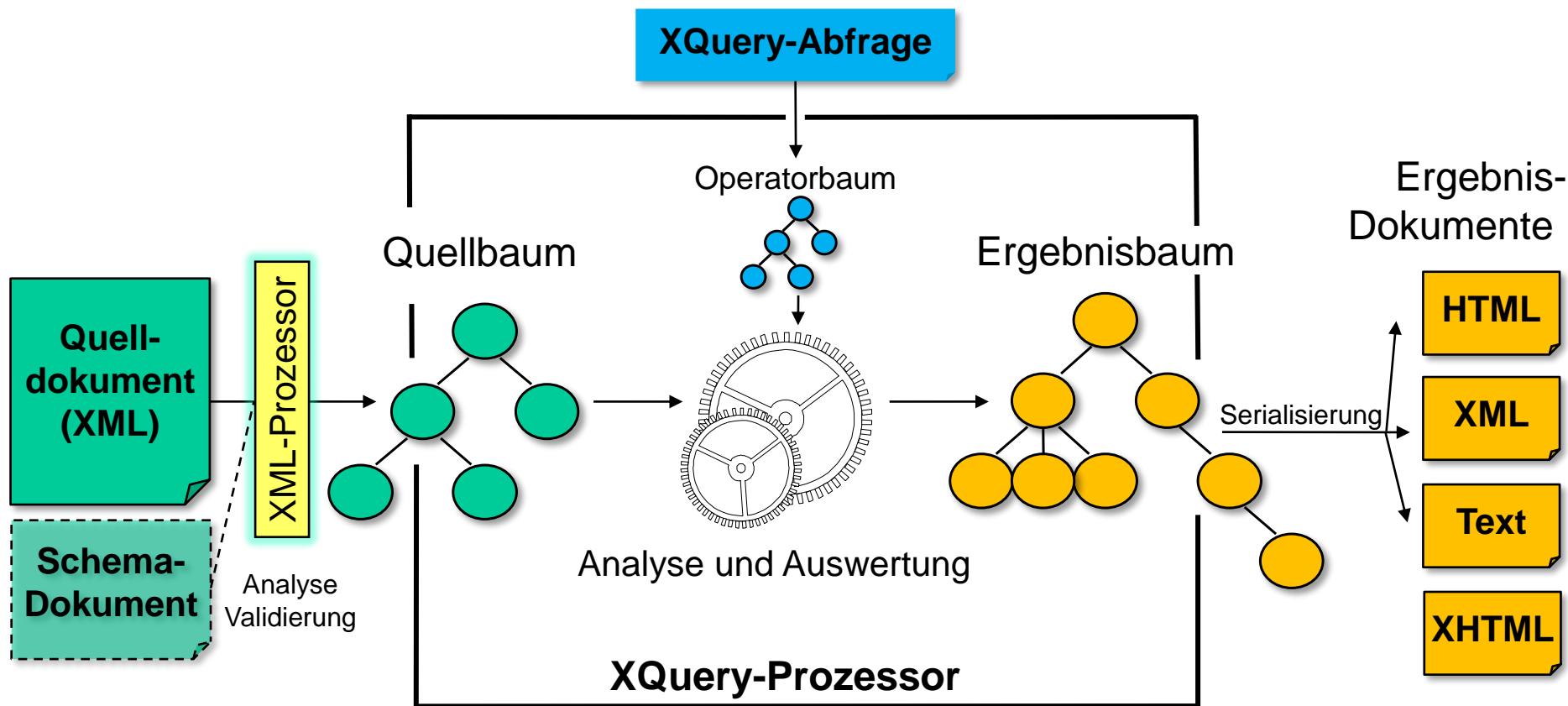
```
<Title>Introduction to semi-structured data models and XML</Title>
<Title>Introduction to semi-structured data models and XML</Title>
```

### Pfadausdrücke

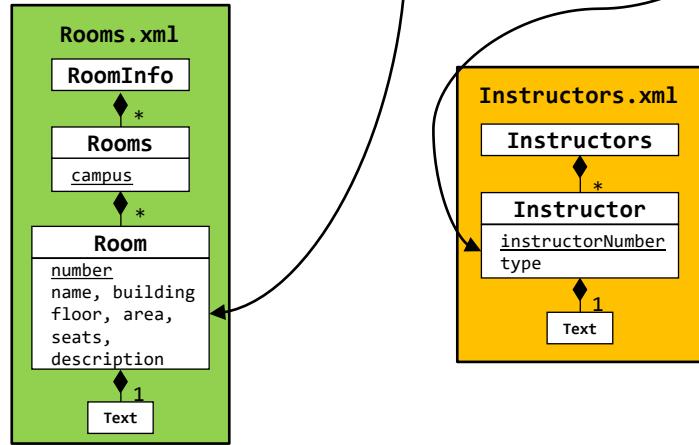
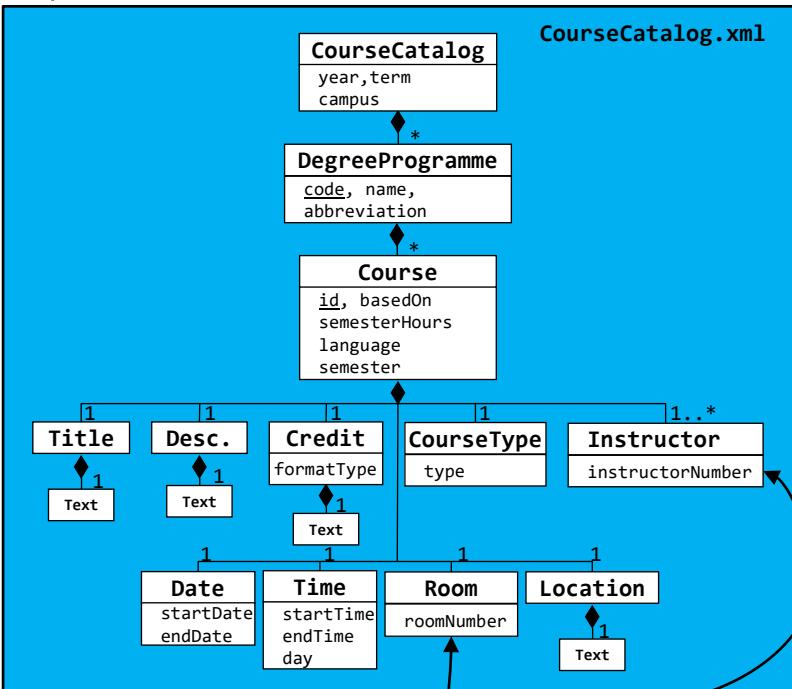
- Gute Unterstützung, wenn in Abfragen ausgewählte Elemente und Attribute **unverändert** in das Ergebnis übernommen werden!

# XQuery

## Grundlagen – Verarbeitungsmodell



# Beispiel Kurskatalog



**Instructors.xml**

```

<Instructors>
 <Instructor instructorNumber="p20621" type="HBL">Josef Altmann</Instructor>
 <Instructor instructorNumber="p22080" type="NBL">Julian Haslinger</Instructor>
 <Instructor instructorNumber="p20622" type="NBL">Norbert Niklas</Instructor>
 <Instructor instructorNumber="p20623" type="HBL">Barbara Traxler</Instructor>
</Instructors>

```

**RoomInfo**

```

<Rooms campus="Hagenberg">
 <Room name="LBS1" number="3.008" building="FH 3" floor="0" description="SE Labor 1" area="77.72" seats="24"/>
 <Room name="LBS2" number="3.009" building="FH 3" floor="0" description="SE Labor 2" area="77.72" seats="24"/>
<!-- ... -->
 <Room name="HS3" number="2.025" building="FH 2" floor="0" description="bet-at-home.com Hörsaal 3" area="156.05" seats="120"/>
 <Room name="HS4" number="1.004" building="FH 1" floor="0" description="CELUM HS2" area="95.20" seats="90"/>
<!-- ... -->
</RoomInfo>

```

**CourseCatalog**

```

<CourseCatalog year="2019" term="summer" campus="Hagenberg">
 <DegreeProgramme code="0307" name="Software Engineering" abbreviation="SE">
 <Course id="cID_8314" semesterHours="1" language="en" semester="4">
 <Title>Introduction to semi-structured data models and XML</Title>
 <Description>Introduction of skills related to XML. <Content>Includes DTD, Schema, XPath, XQuery, XSLT, JSON</Content> <Exam>Final Exam required.</Exam> Participation without any previous knowledge.</Description>
 <Credit formatType="ECTS">1</Credit>
 <CourseType type="Lecture"/>
 <Date startDate="--02-28" endDate="--05-03"/>
 <Time startTime="08:00:00" endTime="10:25:00" day="THU"/>
 <Room roomNumber="1.004"/>
 <Instructor instructorNumber="p22080"/>
 </Course>
<!-- ... -->
 </DegreeProgramme>
</CourseCatalog>

```

# XQuery

## for / let-Klausel

{ } Auswertungskontext

- **let-Klausel mit to-Bereichsoperator**

```
let $i := (1 to 3)
return <value>{$i}</value>
```

<value>1 2 3</value>

- **for-Klausel mit to-Bereichsoperator**

```
for $i in (1 to 3)
return <value>{$i}</value>
```

<value>1</value>
<value>2</value>
<value>3</value>

- **Schachtelung von for-Klauseln**

```
for $i in (1 to 2)
for $j in ('a', 'b')
return
<result>
 $i is {$i} and $j is {$j}
</result>
```

<result>\$i is 1 and \$j is a</result>
<result>\$i is 1 and \$j is b</result>
<result>\$i is 2 and \$j is a</result>
<result>\$i is 2 and \$j is b</result>

- **for-Klausel mit mehreren Variablenbindungen**

```
for $i in (1 to 2), $j in ('a', 'b')
return
<result>
 $i is {$i} and $j is {$j}
</result>
```

Sequenz mit 2 Items: a, b

# XQuery

## for / let-Klausel

### Einfache for / let-Klausel

```
for $course in doc("CourseCatalog.xml")//Course
let $courseType := $course/CourseType/@type
where $courseType = 'Training'
return $course/Title
```

```
<Title>Intercultural Communications</Title>
```

### Mehrere for / let-Klauseln

```
let $doc := doc("CourseCatalog.xml")
for $course in $doc//Course
let $courseType := $course/CourseType/@type
let $courseTitle := $course/Title
where $courseType = 'Training'
return $courseTitle
```

} Reihenfolge könnte  
zusätzlich geändert  
werden

```
<Title>Intercultural Communications</Title>
```

# XQuery

## for / let-Klausel – Namespaces 1/2

- Einfache for / let-Klausel – Elemente in bestimmten Namespace

```
declare default element namespace "http://www.fh-ooe.at/CourseCatalog";
for $course in doc("CourseCatalogWithSchema.xml")//Course
let $courseType := $course/CourseType/@type
where $courseType = 'Training'
return $course/Title
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Title xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.fh-ooe.at/CourseCatalog"
 xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
> Intercultural Communications</Title>
```

Default-Namespace wird übernommen

Andere Namespaces aus dem Quell-Dokument

# XQuery

## Hinzufügen von Elementen/Attributen

### (1) Unveränderte Übernahme aus Eingangsdokument

- Einfache Elemente
- Komplexe Elemente – zusätzlich Attribute und Sub-Elemente
  - Elemente, Attribute, etc. nicht änderbar

### (2) Direkter Element-/Attributkonstruktor – Mischung aus ...

- Literale – direkte Übernahme in das Ausgabedokument
- Ausdrücke in {} werden zu Knoten (Elemente, Attribute, etc.) und atomare Werte ausgewertet
- Direkter Elementkonstruktor
  - muss auf XML-Syntax basieren

### (3) Berechneter Konstruktor

- Berechnung des Namens und des Inhalts von Elementen
- Transformation eines Eingangsdokuments in ein Ausgangsdokument (mit geänderten Elementen, Attributen und Inhalten)

# XQuery

## (1) Unveränderte Übernahme aus Eingangsdocument

### ■ Übernahme von einfachen Elementen

```
for $course in
doc("CourseCatalog.xml")//Course[@id="cID_8314"]
return $course/Title
```

<Title>Introduction to semi-structured data models and XML</Title>

### ■ Übernahme von komplexen Elementen

```
for $course in
doc("CourseCatalog.xml")//Course[@id="cID_8314"]
return $course
```

<Course id="cID\_8314" semesterHours="1" language="en" semester="4">  
 <Title>Introduction to semi-structured data models and XML</Title>  
 <Description>Introduction of skills related to XML.<Content>Includes DTD, Schema, XPath,  
 XQuery, XSLT, JSON</Content>  
 <Exam>Final Exam required.</Exam>Participation without any previous knowledge. </Description>  
 <Credit formatType="ECTS">1</Credit>  
 <CourseType type="Lecture"/>  
 <Date startDate="--02-28" endDate="--05-03"/>  
 <Time startTime="08:00:00" endTime="10:25:00" day="THU"/>  
 <Room roomNumber="1.004"/>  
 <Instructor instructorNumber="p22080"/>  
</Course>

# XQuery

## (2) Direkter Konstruktor 1/7

- Schließe Ergebnis (`Title`-Elemente) in `ul`-Element ein

Literaler Inhalt

```

{
for $course in
doc("CourseCatalog.xml")//Course[@id="cID_8314"]
return $course/Title
}

```

<ul>
 <Title>Introduction to semi-structured data models and XML</Title>
</ul>

- Schließe jedes `Title`-Element in ein `li`-Element ein

```

{
for $course in
doc("CourseCatalog.xml")//Course[@id="cID_8314"]
return {$course/Title}
}

```

<ul>
 <li>
 <Title>Introduction to semi-
 structured data models and XML</Title>
 </li>
</ul>

Literaler Inhalt

# XQuery

## (2) Direkter Konstruktor 2/7

- Neue Attribute hinzufügen, Attributwert/Elementinhalt übernehmen

```
<ul type="Courses">
{
for $course in doc("CourseCatalog.xml")//Course
order by $course/Time/@startTime descending
return <li time ="{$course/Time/@startTime}">{data($course/Title)}
}

```

Neuer Attributname & -wert

Neuer Attributname &  
Übernahme von existierendem Wert

Übernahme Elementinhalt  
via data()-Funktion

```
<ul type="Courses">
 <li time="11:20:00">Introduction to semi-structured data models and XML
 <li time="10:30:00">Data modelling and database design
 <li time="09:00:00">Intercultural Communications
 <li time="08:00:00">Introduction to semi-structured data models and XML

```

# XQuery

## (2) Direkter Konstruktor 3/7

- Elementinhalt als Attributwert mit Präfix (ECTS\_) übernehmen

```
for $course in doc("CourseCatalog.xml")//Course
order by $course/Time/@startTime
return
 <new_course
 ects="ECTS_{$course/Credit}"
 title="{$course/Title}">
 </new_course>
```

data()-Funktion nicht notwendig –  
Automatische Atomisierung

```
<new_course ects="ECTS_1"
 title="Introduction to semi-structured data models and XML"/>
<new_course ects="ECTS_0.5" title="Intercultural Communications"/>
<new_course ects="ECTS_2" title="Data modelling and database design"/>
<new_course ects="ECTS_1.5"
 title="Introduction to semi-structured data models and XML"/>
```

# XQuery

## (2) Direkter Konstruktor 4/7

- Attribute/Element übernehmen und bestimmte Attribute/Elemente eliminieren

```
for $course in doc("CourseCatalog.xml")//Course
where contains($course/Title, "Intro")
order by $course/Time/@startTime
return
<new_course>
 {$course/(@* except @id, * except (Credit, CourseType, Location,
 Description, Instructor, Date)),
 }
</new_course>
```

course-**Attribute** in new\_course-Attribute übernehmen  
Ausnahme: Attribut @id wird nicht übernommen.

course-**Elemente** übernehmen und  
als Sub-Elemente in new\_course-Element aufnehmen  
Eliminiere bestimmte Sub-Elemente von  
course-Element

```
<new_course semesterHours="1" language="en" semester="4">
 <Title>Introduction to semi-structured data models and XML</Title>
 <Time startTime="08:00:00" endTime="10:25:00" day="THU"/>
 <Room roomNumber="1.004"/>
</new_course>
<new_course semesterHours="2" language="en" semester="4" basedOn="cID_8314">
 <Title>Introduction to semi-structured data models and XML</Title>
 <Time startTime="11:20:00" endTime="14:35:00" day="SAT"/>
 <Room roomNumber="3.108" building="FH3">XORTEX LBS3</Room>
</new_course>
```

# XQuery

## (2) Direkter Konstruktor 6/7

- Eingeschlossener Ausdruck erzeugt Elemente

```
for $course in doc("CourseCatalog.xml")//Course
return
ECTS: {$course/Credit}
```

```
ECTS: <Credit formatType="ECTS">1</Credit>
ECTS: <Credit formatType="ECTS">1.5</Credit>
ECTS: <Credit formatType="ECTS">0.5</Credit>
ECTS: <Credit formatType="ECTS">2</Credit>
```

- Eingeschlossener Ausdruck erzeugt Attribute

```
for $course in doc("CourseCatalog.xml")//Course
return
{$course/@id}ECTS: {$course/Credit}
```

```
<li id="cID_8314">ECTS: <Credit formatType="ECTS">1</Credit>
<li id="cID_8315">ECTS: <Credit formatType="ECTS">1.5</Credit>
<li id="cID_7555">ECTS: <Credit formatType="ECTS">0.5</Credit>
<li id="cID_8840">ECTS: <Credit formatType="ECTS">2</Credit>
```

# XQuery

## (3) Berechneter Konstruktor 1/4

- Neues Element erzeugen (dynamisch berechneter Element-Name)

```
for $course in doc("CourseCatalog.xml")//Course
where contains($course/@id, '8314')
return
 element {$course/@id} {data($course/Title)}
```

```
<cID_8314>Introduction to semi-structured data models and XML</cID_8314>
```

- Neues Element erzeugen

```
for $course in doc("CourseCatalog.xml")//Course
where contains($course/@id, '8314')
return
 element Kurs {data($course/Title)}
```

```
<Kurs>Introduction to semi-structured data models and XML</Kurs>
```

# XQuery

## (3) Berechneter Konstruktor 2/4

- Neues Attribut erzeugen (dynamisch berechneter Name)

```
for $course in doc("CourseCatalog.xml")//Course
where contains($course/@id, '8314')
return
 element Kurs { attribute {$course/CourseType/@type} {$course/@id}}
```

```
<Kurs Lecture="cID_8314"/>
```

- Neues Attribut erzeugen

```
for $course in doc("CourseCatalog.xml")//Course
where contains($course/@id, '8314')
return
 element Kurs { attribute ID {$course/@id}}
```

```
<Kurs ID="cID_8314"/>
```

# XQuery

## (3) Berechneter Konstruktor 4/4

- Inhalt in XML-Markup transformieren
  - Attributinhalt ⇒ Elemente
  - Expliziter Elementkonstruktor

```
for $wd in distinct-values(doc("CourseCatalog.xml")//@day)
return
element {$wd}
{ doc("CourseCatalog.xml")//Course[Time/@day = $wd]/Title}
```

```
<THU>
 <Title>Introduction to semi-structured data models and XML</Title>
</THU>
<SAT>
 <Title>Introduction to semi-structured data models and XML</Title>
</SAT>
<MON>
 <Title>Data modelling and database design</Title>
</MON>
```

# XQuery

## Konditionale Ausdrücke - if

- Einfacher konditionaler Ausdruck
  - Teil von Return-Klausel

```
for $course in doc("CourseCatalog.xml")//Course
let $type := $course/CourseType/@type
return
 if ($type = 'Lecture') then
 <VO>{$course/@id, data($course/Title)}</VO>
 else if ($type = 'LabSession') then
 <UE>{$course/@id, data($course/Title)}</UE>
 else
 <Sonstiges>{$type, data($course/Title)}</Sonstiges>
```

Rückgabe-Wert  
muss einem  
XQuery-Datentyp  
entsprechen

```
<VO id="cID_8314">Introduction to semi-structured data models and XML</VO>
<UE id="cID_8315">Introduction to semi-structured data models and XML</UE>
<Sonstiges type="Training">Intercultural Communications</Sonstiges>
<VO id="cID_8840">Data modelling and database design</VO>
```

# XQuery

## Verbunde 1/3

```
<info>Course Introduction to
semi-structured data models and
XML is in room 3.108</info>
```

### ■ Verbund mit Verbundprädikat

```
for $course in doc("CourseCatalog.xml")//Course
for $room in doc("Rooms.xml")//Room[@number = $course/Room/@roomNumber]
return
<info>
{
 concat('Course ', $course/Title, ' is in room ', $room/@number)
}
</info>
```

### ■ Verbund mit Verbundprädikat in der `where`-Klausel

```
for $course in doc("CourseCatalog.xml")//Course
for $room in doc("Rooms.xml")//Room
where $room/@number = $course/Room/@roomNumber
return
<info>
{
 concat('Course ', $course/Title, ' is in room ', $room/@number)
}
</info>
```

# XQuery

## Verbunde 2/3

### ■ Verbund mit 3 XML-Dokumenten in where-Klausel

```
for
$course in doc("CourseCatalog.xml")//Course,
$room in doc("Rooms.xml")//Room,
$instructor in doc("Instructors.xml")//Instructor
```

where

```
$room/@number = $course/Room/@roomNumber and
$course/Instructor/@instructorNumber = $instructor/@instructorNumber
```

return

```
<info>
{
 concat('Course "', $course/Title, '" is in room FH',
 $room/@number, ' (lecturer: ', $instructor, ')')
}
```

```
<info>Course "Introduction to semi-structured data models and XML" is in room FH3.108
(lecturer: Barbara Traxler)</info>
```

# XQuery

## Verbunde 3/3

### ■ Äußerer Verbund

```
for $course in doc("CourseCatalog.xml")//Course
return
<info course = "{$course/Title}">
{
 attribute room {
 for $room in doc("Rooms.xml")//Room
 where $room/@number = $course/Room/@roomNumber
 return $room/@number
 }
}
</info>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<info course="Introduction to semi-structured data models and XML" room="" />
<info course="Introduction to semi-structured data models and XML"
 room="3.108" />
<info course="Intercultural Communications" room="" />
<info course="Data modelling and database design" room="" />
```

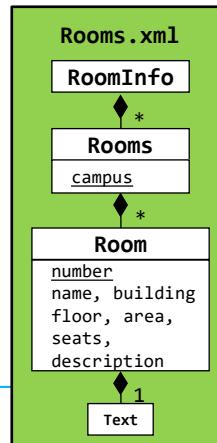
# XQuery

## Duplikat-Eliminierung und Gruppierung

### ■ Gruppierung der Räume nach Gebäude (@building)

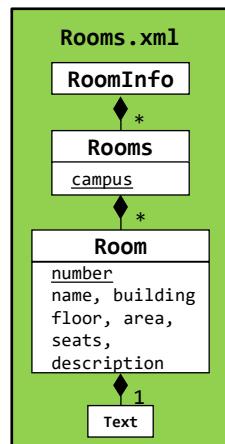
```
for $fh in distinct-values(doc("Rooms.xml")//Room/@building)
let $rooms := doc("Rooms.xml")//Room[@building = $fh]
order by $fh
return
 <fh id="{{$fh}}>
 {
 for $room in $rooms
 order by $room/@number
 return
 <room>{$room/@name}</room>
 }
</fh>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<fh id="FH 1">
 <room name="HS2"/>
</fh>
<fh id="FH 2">
 <room name="AM"/>
 <room name="HS3"/>
 <room name="SRC1"/>
 <room name="LBB2"/>
 <room name="LBB1"/>
 <room name="LBS5"/>
 <room name="SRC2"/>
</fh>
<fh id="FH 3">
 <room name="LBS1"/>
 <room name="LBS2"/>
 <room name="LBS3"/>
 <room name="LBS4"/>
</fh>
```



# XQuery

## Duplikateliminierung, Sortierung und Aggregation 1/2



### Aggregation – `sum`

```
for $fh in distinct-values(doc("Rooms.xml")//Room/@building)
let $rooms := doc("Rooms.xml")//Room[@building = $fh]
order by $fh
return
 <fh
 id ="{$fh}"
 totalArea ="{sum($rooms/@area)}"
 seats ="{sum($rooms/@seats)}"
 />
```

```
<fh id="FH 1" totalArea="95.2" seats="90"/>
<fh id="FH 2" totalArea="842.61" seats="497"/>
<fh id="FH 3" totalArea="310.88" seats="92"/>
```

### Aggregation – `sum` (mit `group by`, XQuery 3.0)

```
for $rooms in doc("Rooms.xml")//Room
group by $building := $rooms/@building
order by $building
return
 <fh
 id ="{$building}"
 totalArea ="{sum($rooms/@area)}"
 seats ="{sum($rooms/@seats)}"
 />
```

# Inhalt

- Einführung
- XQuery 1.0
  - Grundlagen
  - **for**- und **let**-Klausel
  - Hinzufügen von Elementen/Attributen
  - Konditionale Ausdrücke
  - Verbund
  - Quantifizierende Ausdrücke
  - Gruppierung
  - Sortierung und Aggregation
  - **Programmstruktur**
  - **Dokumentation**
- Anhang: XQuery Update Facility 1.0

# Programmstruktur – Beispiel

Verwendung einer Bibliothek mit rekursiven Funktionen

```
xquery version "1.0";
module namespace rec = "http://www.fh-ooe.at/xquery/recursion";
}

declare function rec:factorial($x as xs:integer)
as xs:integer
{
 if ($x <= 1) then
 1
 else
 $x * rec:factorial($x - 1)
};
```

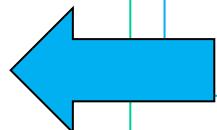
library.xqm

```
xquery version "1.0";
import module namespace r = "http://www.fh-
ooe.at/xquery/recursion" at "library.xqm";

for $i in -10 to 4
let $fac := r:factorial($i)
return
 if ($i >= 0) then
 <val i='{$i}' fac='{$fac}'/>
 else
 ()
```

main.xq

```
<val i="0" fac="0"/>
<val i="1" fac="1"/>
<val i="2" fac="2"/>
<val i="3" fac="6"/>
<val i="4" fac="24"/>
```



# Inhalt

- Einführung
- XQuery 1.0
  - Grundlagen
  - `for`- und `let`-Klausel
  - Hinzufügen von Elementen/Attributen
  - Konditionale Ausdrücke
  - Verbund
  - Quantifizierende Ausdrücke
  - Gruppierung
  - Sortierung und Aggregation
  - Programmstruktur
  - Dokumentation
- Anhang: XQuery Update Facility 1.0

# Exkurs: XQuery Update Facility 1.0

## Instructors.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Instructors>
 <Instructor instructorNumber="p20621" type="HBL">Josef Altmann</Instructor>
 <Instructor instructorNumber="p22080" type="NBL">Julian Haslinger</Instructor>
 <Instructor instructorNumber="p20622" type="NBL">Norbert Niklas</Instructor>
 <Instructor instructorNumber="p20623" type="HBL">Barbara Traxler</Instructor>
</Instructors>
```

<https://www.w3.org/TR/xquery-update-10/>

XQuery Update Facility 3.0: <https://www.w3.org/TR/xquery-update-30/>

# XQuery Update Facility 1.0

## ■ Löschen

- Knoten, der durch den Ausdruck selektiert wird, wird gelöscht

```
delete node doc("Instructors.xml")//Instructor[@instructorNumber = 'p22080']
```



```
<?xml version="1.0" encoding="UTF-8"?>
<Instructors>
 <Instructor instructorNumber="p20621" type="HBL">Josef Altmann</Instructor>
 <Instructor instructorNumber="p22080" type="NBL">Julian Haslinger</Instructor>
 <Instructor instructorNumber="p20622" type="NBL">Norbert Niklas</Instructor>
 <Instructor instructorNumber="p20623" type="HBL">Barbara Traxler</Instructor>
</Instructors>
```

# XQuery Update Facility 1.0

## ■ Einfügen

```
insert node
 <Instructor>Julian Haslinger</Instructor>
before
doc("Instructors.xml")//Instructor[1]
```

Mögliche Positionen  
**before / after / into**  
**as first / as last**



```
<?xml version="1.0" encoding="UTF-8"?>
<Instructors>
 <Instructor>Julian Haslinger</Instructor>
 <Instructor instructorNumber="p20621" type="HBL">Josef Altmann</Instructor>
 <Instructor instructorNumber="p20622" type="NBL">Norbert Niklas</Instructor>
 <Instructor instructorNumber="p20623" type="HBL">Barbara Traxler</Instructor>
</Instructors>
```

# XSLT

## eXtensible Stylesheet Language (for) Transformations

Julian Haslinger

```
<xsl:template match="/">
```



**Der vorliegende Foliensatz basiert vorwiegend auf:**

DuCharme, B., *XSLT Quickly*, Manning, 2001.

Kay, M., *XSLT 2.0 Programmer's Reference (4<sup>th</sup> ed.)*, Wiley, 2008.

Bongers, F., *XSLT 2.0 und XPath 2.0 (2. Auflage)*, Galileo Computing, 2008.

Tidwell, D., *XSLT, 2<sup>nd</sup> ed.*, O'Reilly, 2008.

# Inhalt

- Einführung in XSL
- XSLT Grundlagen
- XSLT 2.0/3.0 Erweiterungen
- XSLT versus XQuery
- Anhang I: Verarbeitungsmodell
- Anhang II: Stylesheets
- Anhang III: Template Rules
- Anhang IV: Beispiel

# Einführung in XSL 1/5

## Motivation für Stylesheets

- **Stylesheet** = Datei, die Regeln für die Strukturierung, Transformation und Formatierung des Inhalts einer Quelldatei enthält
- **Regeln** werden nicht in einer Applikation fest codiert
  - Prozessor zur Verarbeitung von Stylesheets ist generisch
- Ein und derselbe Inhalt ist
  - unterschiedlich **strukturierbar**
    - Umordnen, Erweitern, Einschränken etc.
    - Umwandeln in andere Text-basierte Formate, z.B.
      - » HTML für die Präsentation im Web
      - » CSV (*comma-separated values*) für Datenbank-Anwendungen
      - » JSON (JavaScript Object Notation) für Web-Anwendungen
  - unterschiedlich **formatierbar**
    - verschiedene Ausgabeformate – (X)HTML, PDF, SVG, EPUB etc.
    - verschiedene Ausgabegeräte - PC, Smartphones etc.
- Content-Provider müssen sich nicht um Layout-Fragen kümmern
- Verschiedene Sprachen zur Erstellung von Stylesheets:
  - DSSSL („[dɪsl]“, Document Style Semantics and Specification Language) für SGML
  - CSS (Cascading Style Sheets) für HTML und X(HT)ML
  - XSL (Extensible Stylesheet Language) für X(HT)ML

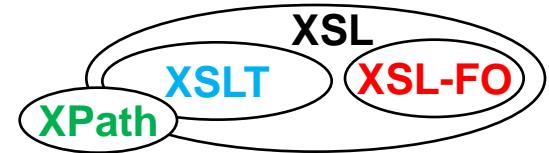
# Einführung in XSL 2/5

## CSS vs. XSL

	CSS	XSL
Anwendbar auf HTML-Dokumente	ja	nein
Anwendbar auf XML-Dokumente	ja	ja
Transformation	nein	ja
Syntax	proprietär	XML
Dokument als Baum repräsentiert	nein	ja
Anwendungsbereich	einfach	komplex

# Einführung in XSL 3/5

## XSL-Transformation und Formatierung



### ■ XSLT: XSL Transformations

- Sprache zur Transformation der Struktur eines XML-Dokuments
- Ausgabe in XML (primäres Ziel des W3C), aber auch in (X)HTML oder beliebigem anderen ASCII-Format
- verwendet XPath zur Adressierung von Dokumentteilen
- W3C-REC:
  - XSLT 1.0, Nov. 1999, 120 Seiten
  - XSLT 2.0, Jan. 2007, 210 Seiten
  - XSLT 3.0, Feb. 2017

### ■ XSL-FO: XSL Formatting Objects

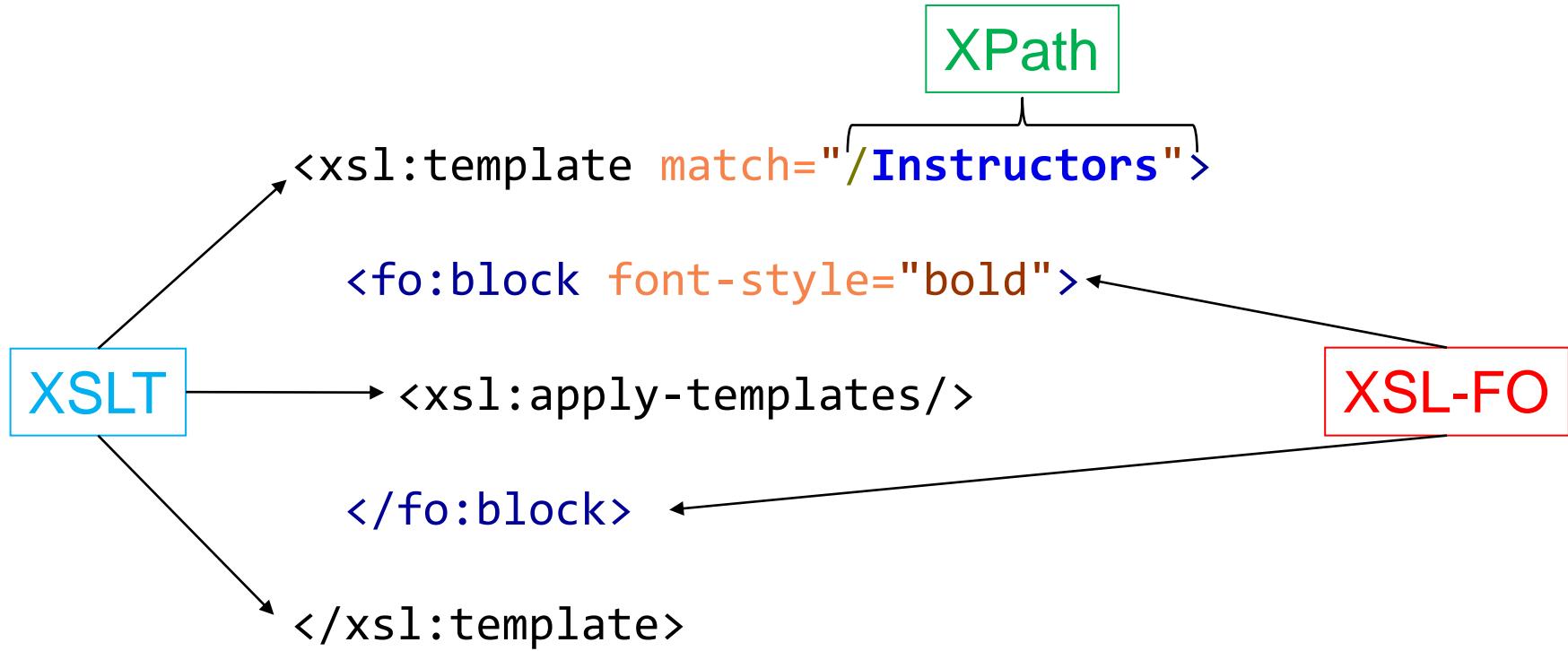
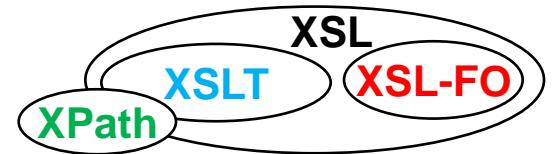
- XML-Vokabular zur Spezifikation von Formatierungssemantik, unabhängig von konkreten Ausgabeformaten
- im (professionellen) Print/Publishing-Bereich eingesetzt
- W3C Recommendation:
  - Extensible Stylesheet Language (XSL) 1.0, 15. Oct. 2001, ca. 420 Seiten
  - Extensible Stylesheet Language (XSL) 1.1, 5. Dec. 2006
- W3C-WD (Working Draft):
  - Extensible Stylesheet Language (XSL) 2.0, 2012

Michael Kay:

"XSLT ist das SQL des Web"

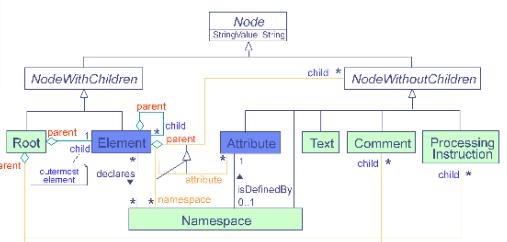
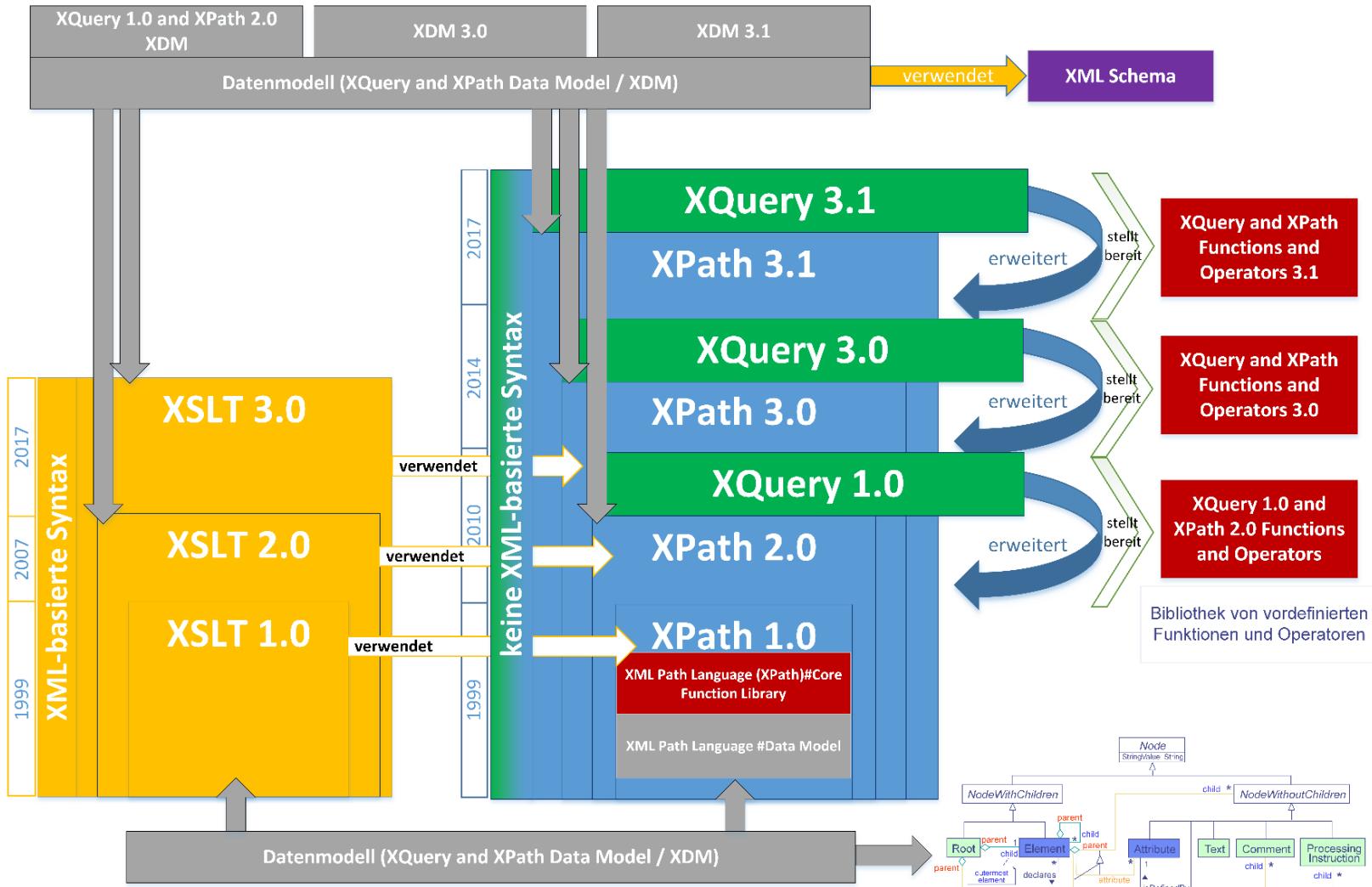
# Einführung in XSL 4/5

## XSL Beispiel – Template Rules



# Einführung in XSL 5/5

## Zusammenhang XPath – XSLT – XQuery



# Inhalt

- Einführung in XSL
- [XSLT Grundlagen](#)
- XSLT 2.0/3.0 Erweiterungen
- XSLT versus XQuery
- Anhang I: Verarbeitungsmodell
- Anhang II: Stylesheets
- Anhang III: Template Rules
- Anhang IV: Beispiel

# XSLT Grundlagen 1/8

## Wozu Transformieren?

### ■ Beispiel Datenaustausch:

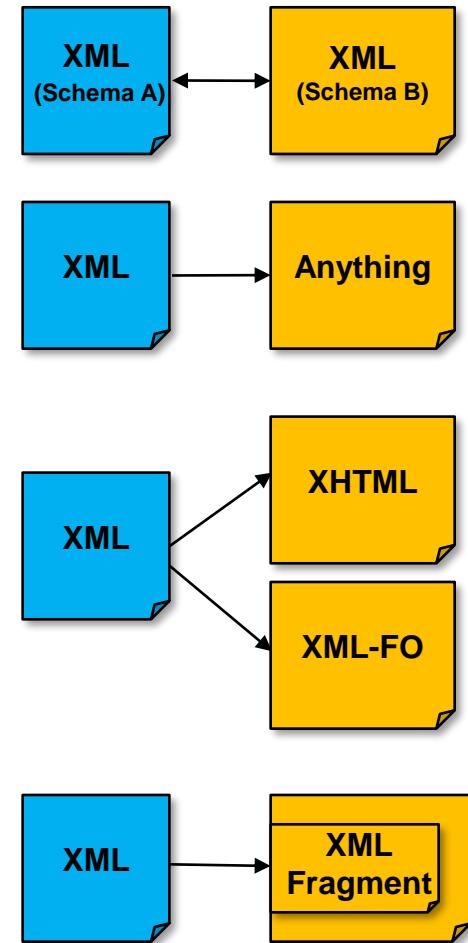
- falls kein gemeinsames Schema verwendet wird → Umstrukturierung der XML-Dokumente
- falls Empfänger-Applikation kein XML versteht → Transformation z.B. in CSV für DB-Anwendungen oder JSON für Web-Anwendungen

### ■ Beispiel Multi-Delivery (Publishing):

- XML muss in entsprechendes Format des Ausgabegerätes transformiert werden (HTML, XHTML, XSL-FO (PDF), Office Open XML, EPUB, etc.)

### ■ Beispiel Datenintegration:

- Extraktion von XML-Dokument-Teilen und Integration in andere XML-Dokumente (z.B. Kundenadresse aus Kundenliste extrahieren und in Rechnung integrieren)

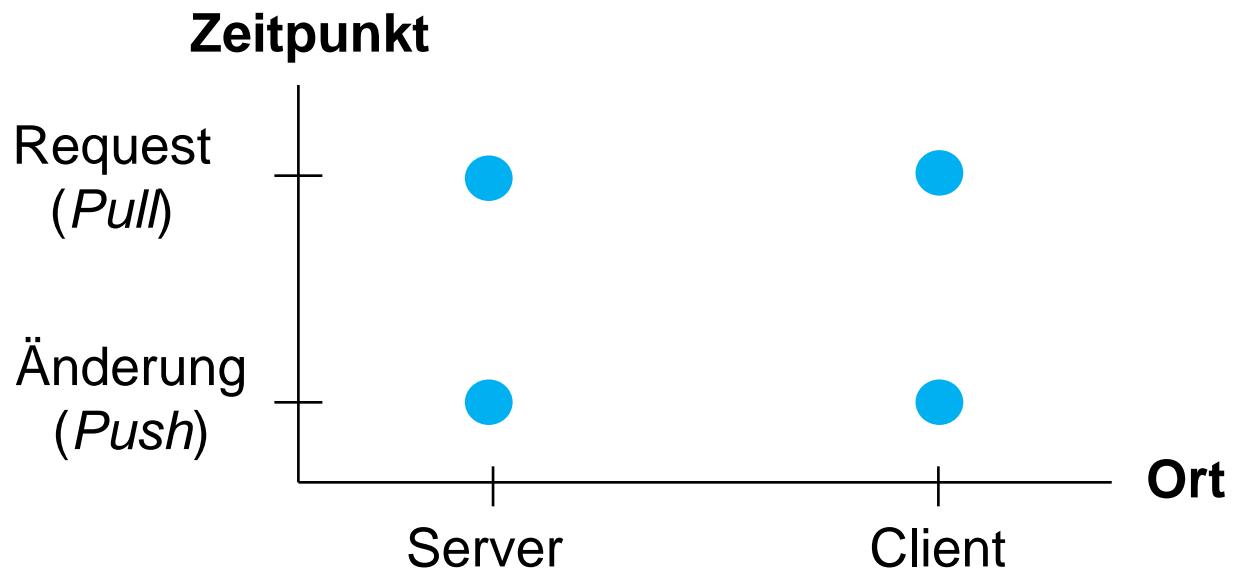


# XSLT Grundlagen 2/8

## Wann/Wo Transformieren?

Kategorien von XSLT-Prozessoren:

- Teil des Web Browsers: z.B. IE
- Teil des Web (Application) Servers:  
z.B. <http://xml.apache.org/cocoon>
- Standalone: z.B. Saxon  
<http://saxon.sourceforge.net/>  
oder Xalan <http://xml.apache.org/xalan-j/>



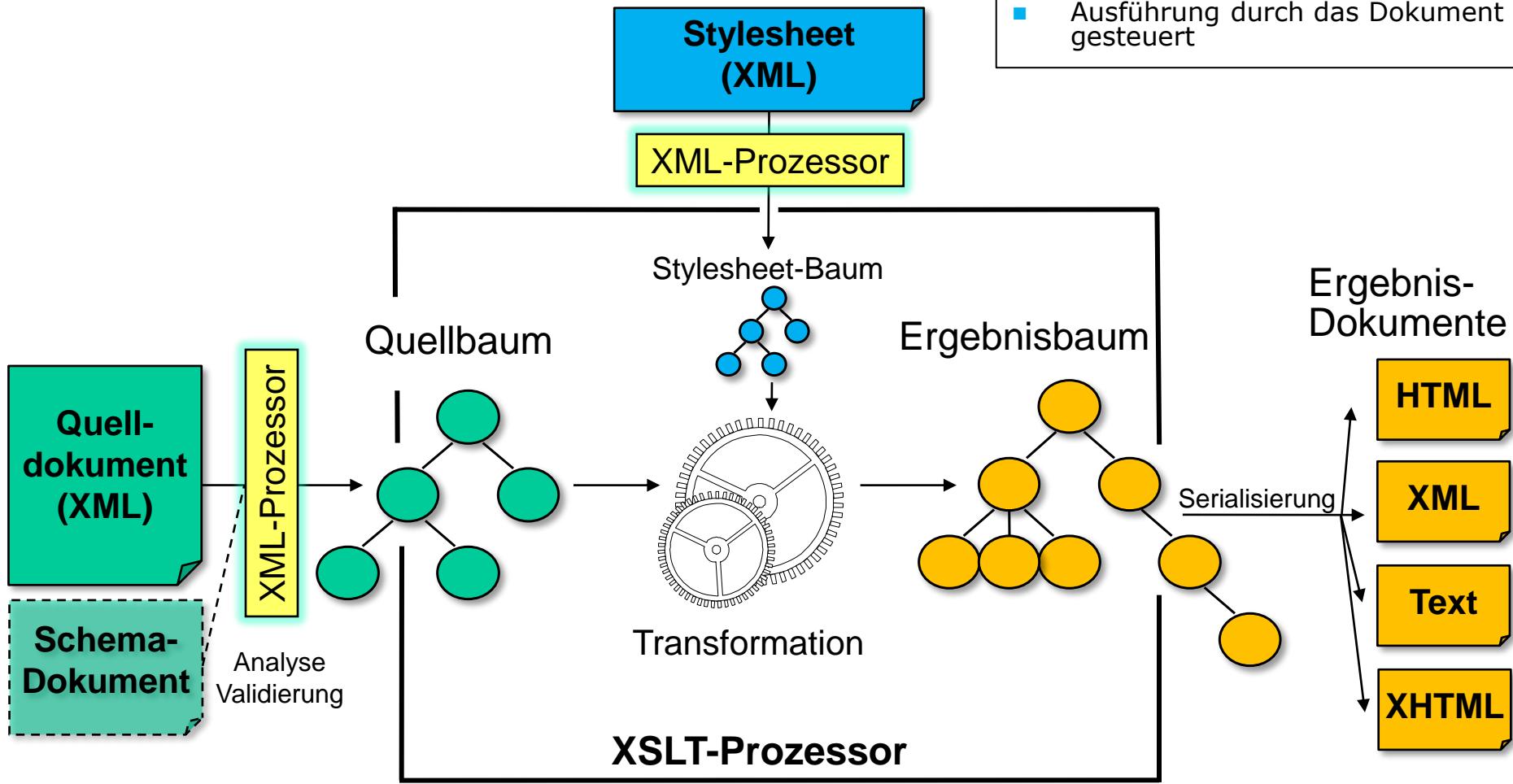
# XSLT Grundlagen 3/8

## Sprachcharakteristika

- XSLT basiert auf **XML-Syntax**
  - XSLT-Stylesheet ist ein wohlgeformtes XML-Dokument
  - Wiederverwendung von XML-Prozessoren zum Parsen
- XSLT verwendet **XPath**
  - zur Selektion von zu verarbeitenden Elementen, etc.
- XSLT ist **ressourcenintensiv**
  - gesamtes Dokument im Hauptspeicher (Ausnahmen in XSLT 3.0)
- XSLT erlaubt die **Einbindung** anderer **Programmiersprachen**
  - Java, JavaScript etc.
- XSLT ist **regelbasiert**
  - ein Stylesheet enthält eine Menge von "Template Rules"
  - wenn ein bestimmtes Muster ("Pattern") - definiert durch XPath - im Input auftaucht ("Kontext"), dann wird ein bestimmter Output produziert ("Action")
- XSLT benötigt **XSLT-Prozessoren**
  - auf Client- oder Serverseite (Altova RaptorXML Server, Altova XMLSpy, Saxon, Xalan-Java, Xalan-C++, MS XML Core Services, etc.)

# XSLT Grundlagen 4/8

## Verarbeitungsmodell



Programmsteuerung durch Template Rules:

- Rekursion als Normalfall
- Selektion der Template Rules durch XSLT-Prozessor
- Ausführung durch das Dokument gesteuert

# XSLT Grundlagen 5/8

## Beispiel: Quell- und Ergebnisdokument

CourseCatalog.xml

```
<CourseCatalog year="2019" term="summer" campus="Hagenberg">
 <DegreeProgramme code="0307" name="Software Engineering" abbreviation="SE">
 <Course id="cID_8314" semesterHours="1" language="en" semester="4">
 <Title>Introduction to semi-structured data models and XML</Title>
 <Description>Introduction of skills related to XML.<Content>Includes DTD, Schema, XPath, XQuery, XSLT, JSON</Content>
 <Exam>Final Exam required.</Exam>Participation without any previous knowledge.
 </Description>
 <Credit formatType="ECTS">1</Credit>
 <CourseType type="Lecture"/>
 <Date startDate="--02-28" endDate="--05-03"/>
 <Time startTime="08:00:00" endTime="10:25:00" day="THU"/>
 <Room roomNumber="1.004" building="FH1">CELUM HS2</Room>
 <Instructor instructorNumber="p22080">Julian Haslinger</Instructor>
 </Course>
<!-- ... -->
```

### Kurskatalog: 0307 (Software Engineering)

#### Kurse

ID	Typ	Raum / Location	Titel
cID_7555	Training	White House and Moskovskij Kreml	Intercultural Communications
cID_8314	Lecture	FH1.004	Introduction to semi-structured data models and XML
cID_8315	LabSession	FH3.108	Introduction to semi-structured data models and XML



### Kurskatalog: 0458 (Medical and Bioinformatics)

#### Kurse

ID	Typ	Raum / Location	Titel
cID_8840	Lecture	FH2.010	Data modelling and database design

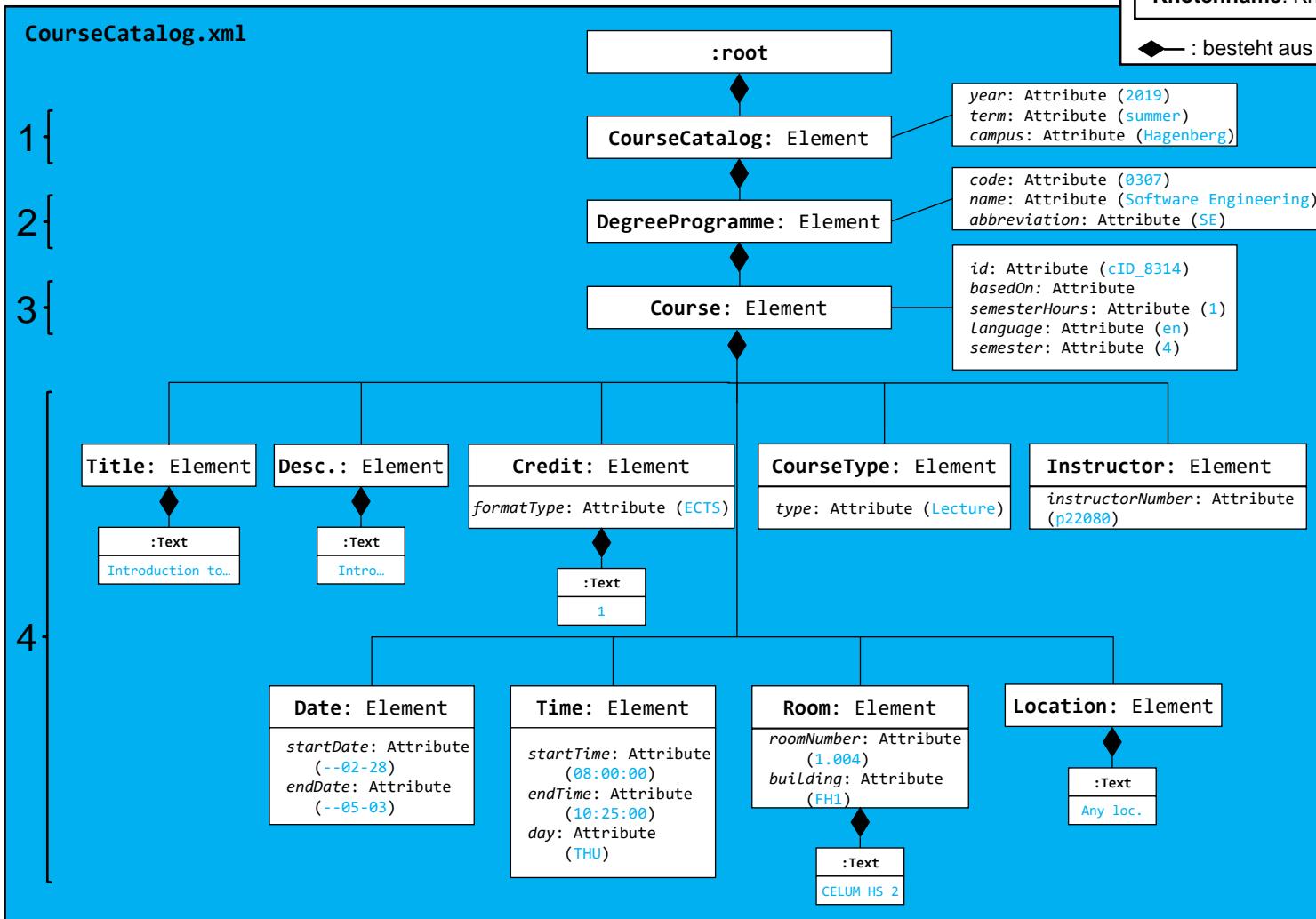
# XSLT Grundlagen 6/8

## Beispiel: Quelldokument im XPath-Datenmodell

Legende:

Knotenname: Knotentyp (Knotenwert)

◆ : besteht aus



# XSLT Grundlagen 7/8

## Beispiel: Stylesheet

```

<xsl:stylesheet version="2.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 1 <xsl:template match="/">
 <html><head/>
 <body>
 <h1>Kurskataloge</h1>
 <xsl:apply-templates/>
 </body>
 </html>
 </xsl:template>

```

Template Rule

```

 2 <xsl:template match="DegreeProgramme">
 <table border="1">
 <tbody>
 <tr bgcolor="#00B0F0">
 <th>Nr.</th>
 <th>Titel</th>
 <th>ECTS</th>
 </tr>
 <xsl:apply-templates/>
 </tbody>
 </table>
 </xsl:template>

```

Anwenden weiterer  
Template Rules auf  
Kindknoten von  
<DegreeProgramme>

```

 3 <xsl:template match="Course">
 <tr>
 <td>
 <xsl:number/>
 </td>
 <xsl:apply-templates select="Title | Credit"/>
 </tr>
 </xsl:template>

```

```

 4 <xsl:template match="Title | Credit">
 <td>
 <xsl:value-of select=". />
 </td>
 </xsl:template>
</xsl:stylesheet>

```

```

<html>
 <head/>
 <body>
 <h1>Kurskataloge</h1>
 <table border="1">
 <tbody>
 <tr bgcolor="#00B0F0">
 <th>Nr.</th>
 <th>Titel</th>
 <th>ECTS</th>
 </tr>
 <tr>
 <td>1</td>
 <td>Introduction to semi-structured data models and XML</td>
 <td>1</td>
 </tr>
 <tr>
 <td>2</td>
 <td>Introduction to semi-structured data models and XML</td>
 <td>1.5</td>
 </tr>
 <tr>
 <td>3</td>
 <td>Intercultural Communications</td>
 <td>0.5</td>
 </tr>
 </tbody>
 </table>
 <table border="1">
 <tbody>
 <tr bgcolor="#00B0F0">
 <th>Nr.</th>
 <th>Titel</th>
 <th>ECTS</th>
 </tr>
 <tr>
 <td>4</td>
 <td>Data modelling and database design</td>
 <td>2</td>
 </tr>
 </tbody>
 </table>
 </body>
</html>

```

Kurskataloge

Nr.	Titel	ECTS
1	Introduction to semi-structured data models and XML	1
2	Introduction to semi-structured data models and XML	1.5
3	Intercultural Communications	0.5

Nr.	Titel	ECTS
4	Data modelling and database design	2

Fortlaufende Nummerierung  
gemäß Position der <Course>-  
Knoten im Quelldokument

Ausgabe des Inhalts der aktuellen  
<Title>- und <Credit>-Knoten

# XSLT Grundlagen 8/8

## Überblick Sprachumfang

Definition der Stylesheet-Struktur

```
<xsl:stylesheet>
<xsl:include>
<xsl:import>
```

Definition des Output-Formats

```
<xsl:output>
```

Definition und Aufrufkontrolle von Template Rules

```
<xsl:template>
<xsl:apply-templates>
<xsl:call-template>
```

Kontrollstrukturen

```
<xsl:if>
<xsl:choose>
<xsl:when>
<xsl:otherwise>
<xsl:for-each>
```

Generierung von Output

```
<xsl:value-of>
<xsl:element>
<xsl:attribute>
<xsl:comment>
<xsl:processing-instruction>
<xsl:text>
```

Kopieren von Source- nach Zieldokument

```
<xsl:copy>
<xsl:copy-of>
```

Definition von Variablen und Parametern

```
<xsl:variable>
<xsl:param>
<xsl:with-param>
```

Sortierung und Nummerierung

```
<xsl:sort>
<xsl:number>
```

# Inhalt

- Einführung in XSL
- XSLT Grundlagen
- **XSLT 2.0/3.0 Erweiterungen**
- XSLT versus XQuery
- Anhang I: Verarbeitungsmodell
- Anhang II: Stylesheets
- Anhang III: Template Rules
- Anhang IV: Beispiel

# XSLT 2.0 Erweiterungen 1/6

XSLT 2.0 – Zahlen und Fakten:

- 49 (bisher 35) XSLT-Elemente
- 19 (bisher 9) XSLT-Funktionen
- 111 (bisher 27) XPath-Funktionen

- Unterstützung von XPath 2.0 (bspw. Sequenzen)
- Vereinfachung von Stylesheets durch
  - Gruppierungsfunktion
  - Wiederverwendung von benutzerdefinierten Funktionen in XPath-2.0-Ausdrücken
  - Zeichenkettenverarbeitung mit regulären Ausdrücken
  - Verarbeitung mehrerer Eingabedokumente
  - Erzeugung mehrerer Ausgabedokumente
  - Zusätzliche Formatoptionen
- Verarbeitung von beliebigen Textdateien
- Unterstützung von XML Schema-Datentypen
  - Verwendung zur Deklaration und Prüfung von Inhalten

# XSLT 2.0 Erweiterungen 2/6

## Gruppierung

### ■ **xsl:for-each-group**

- Zugriff auf Gruppen mit Funktion `current-group()`
- Zugriff auf Schlüssel mit Funktion `current-grouping-key()`
- Attribute zum Gruppieren nach Inhalten/Werten: `group-by`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" indent="yes"/>
 <xsl:template match="/">
 <xsl:for-each-group group-by="CourseType/@type"
 select="CourseCatalog/DegreeProgramme/Course">
 <xsl:sort select="current-grouping-key()" order="descending"/>
 <xsl:variable name="type" select="current-grouping-key()"/>

 <lvacount type="{current-grouping-key()}">
 <count>
 <xsl:value-of select="count(current-group())"/>
 </count>
 <xsl:for-each select="current-group()">
 <lva>
 <xsl:value-of select="Title"/>
 </lva>
 </xsl:for-each>
 </lvacount>
 </xsl:for-each-group>
 </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<lvacount type="Training">
 <count>1</count>
 <lva>Intercultural Communications</lva>
</lvacount>
<lvacount type="Lecture">
 <count>2</count>
 <lva>Introduction to semi-structured data models and XML</lva>
 <lva>Data modelling and database design</lva>
</lvacount>
<lvacount type="LabSession">
 <count>1</count>
 <lva>Introduction to semi-structured data models and XML</lva>
</lvacount>
```

# XSLT 2.0 Erweiterungen 3/6

## Benutzerdefinierte Funktionen

### ■ **xsl:function**

- Element(e) unterhalb von `xsl:stylesheet` platzieren
- Argumente als Parameter (`xsl:param`)
- Funktionsnamen (eigenen) Namensraum zuweisen
- Rückgabe von atomaren Werten oder *Sequenzen*
- Anwendung innerhalb von XPath-Ausdrücken
- Attribut `override (yes|no)` ermöglicht Überschreiben Prozessor-eigener Funktionen mit identischem Namen

```
<!-- Factorial function -->
<xsl:function name="fh:factorial" as="xs:decimal">
 <xsl:param name="x" as="xs:decimal"></xsl:param>
 <xsl:sequence select="if($x le 1) then 1 else $x * fh:factorial($x - 1)"/>
</xsl:function>
 <!-- Testing factorial -->
 <xsl:text>5! = </xsl:text><xsl:value-of select="fh:factorial(5)" />
```

5! = 120

# XSLT 2.0 Erweiterungen 4/6

## Mehrere Ausgabedokumente

### ■ **xsl:result-document**

- Ausgabe beliebig vieler Ergebnisdokumente innerhalb einer Transformation
- Erzeugung mehrerer **xsl:output**-Elemente unterhalb von **xsl:stylesheet**
- **xsl:result-document** verweist über **format**-Attribut auf die deklarierten Namen und enthält den Dateinamen für die Ausgabe als **href**-Attribut

```
<!-- . . . -->
<xsl:output name="CourseList_TextFormat" method="text"/>
<xsl:output name="CourseList_HTMLFormat" method="html"/>

<xsl:template match="/">
 <xsl:result-document href=".//CourseList.txt" format="CourseList_TextFormat">
 <xsl:text>This is the TXT version of the output (...)</xsl:text>
 </xsl:result-document>

 <xsl:result-document href=".//CourseList.html" format="CourseList_HTMLFormat">
 <xsl:text>This is the HTML version of the output (...)</xsl:text>
 </xsl:result-document>
<!-- . . . -->
</xsl:template>
<!-- . . . -->
```

# XSLT 2.0 Erweiterungen 5/6

## Verarbeitung von Textdateien

### ■ **xsl:unparsed-text**

- Liefert den Inhalt einer Textdatei als Zeichenkette zurück

Kurs: Introduction to semi-structured data models and XML (LVA-Leiter: p22080)

Kurs: Introduction to semi-structured data models and XML (LVA-Leiter: p20623)

Kurs: Intercultural Communications (LVA-Leiter: p22100 p22101)

CourseList\_0307\_Software\_Engineering\_Text.txt

```
<xsl:output method="xml"/>
<xsl:template match="/">
 <xsl:variable name="text" select="unparsed-text('CourseList_0307_Software_Engineering_Text.txt')"/>
 <lectures>
 <xsl:for-each select=" tokenize($text, '(\r)|(\n)|(\r\n)')">
 <xsl:if test="string-length(.) > 0">
 <lecture>
 <xsl:value-of select="normalize-space(substring-after(., ':'))"/>
 </lecture>
 <xsl:text>
</xsl:text><!-- add a new line to the output -->
 </xsl:if>
 </xsl:for-each>
 </lectures>
</xsl:template>
```

```
<lectures>
 <lecture>Introduction to semi-structured data models and XML (LVA-Leiter: p22080)</lecture>
 <lecture>Introduction to semi-structured data models and XML (LVA-Leiter: p20623)</lecture>
 <lecture>Intercultural Communications (LVA-Leiter: p22100 p22101)</lecture>
</lectures>
```

# XSLT 2.0 Erweiterungen 6/6

## XML Schema-Datentypen

### Deklaration und Prüfung von Inhalten

- **as-Attribut** für Elemente wie `xsl:param`, `xsl:variable` etc.
- Verwendung von Konstruktorfunktionen wie `xs:date(...)`, `xs:integer(...)`, `xs:string(...)` etc.
- Prüfoperationen: `instance of` / `cast as` / `castable as`

<code>&lt;xsl:variable name="stringVar" as="xs:string"&gt; LVA &lt;/xsl:variable&gt;</code>	<code>&lt;xsl:value-of select="\$stringVar"/&gt;</code>	LVA	Leiter
		LVA	Leiter
<code>&lt;xsl:variable name="tokenVar" as="xs:token"&gt; LVA Leiter &lt;/xsl:variable&gt;</code>	<code>&lt;xsl:value-of select="\$tokenVar"/&gt;</code>	LVA	Leiter
		LVA	Leiter
<code>&lt;xsl:variable name="dateVar" as="xs:date" select="xs:date('2019-03-20')"/&gt;</code>	<code>&lt;xsl:value-of select="\$dateVar"/&gt;</code>	2019-03-20	true
	<code>&lt;xsl:value-of select="\$dateVar instance of xs:date"/&gt;</code>		
<code>&lt;xsl:variable name="dateTimeVar" as="xs:dateTime" select="current-dateTime()"/&gt;</code>	<code>&lt;xsl:value-of select="\$dateTimeVar"/&gt;</code>	2019-03-20T00:14:36.649+01:00	false
	<code>&lt;xsl:value-of select="\$dateTimeVar castable as xs:string"/&gt;</code>		true
	<code>&lt;xsl:value-of select="\$dateTimeVar castable as xs:date"/&gt;</code>		true

# XSLT 3.0 Erweiterungen

## ■ Wesentliche Erweiterung

- Unterstützung zur Transformation von Streams
  - In diesem Modus müssen weder Quell- noch Ergebnisdatei immer komplett im Speicher gehalten werden

## ■ Zusätzliche Erweiterungen

- Unterstützung vom Datentyp Map (ursprünglich für XSLT entwickelt, wurde es dann in XPath 3.1 übernommen)
- Funktionen für Import und Export von JSON
- Packages zur weiteren Modularisierung, Wiederverwendung und Erweiterung von Code
  - Stylesheets als Module strukturieren und über `<xsl:include>` / `<xsl:import>` einbinden
  - Package als Sammlung von Modulen mit einer definierten Schnittstelle (`public`, `private`, `abstract`, `final`)
  - Packages können separat kompiliert werden
- Funktionen höherer Ordnung (XPath 3.0)
- Try/Catch-Funktionalität

# Inhalt

- Einführung in XSL
- XSLT Grundlagen
- XSLT 2.0/3.0 Erweiterungen
- **XSLT versus XQuery**
- Anhang I: Verarbeitungsmodell
- Anhang II: Stylesheets
- Anhang III: Template Rules
- Anhang IV: Beispiel

# XSLT vs. XQuery 1/2

## Funktionale Überschneidungen

- Gleiches Datenmodell, gleiches Typsystem und gleiche Bibliothek von eingebauten („built-in“) Funktionen
  - XPath 2.0 als gemeinsame Untermenge
  - Ausdruckbasierte Sprache, deklarativ
  - Seiteneffektfrei
- Gleiche Mechanismen für
  - Funktionsdefinition
  - Element- und Attribut-erzeugung
- Beide Sprachen sind Turing-vollständig

XSLT feature	Present in 1.0?	XQuery equivalent
<code>xsl:for-each</code>	yes	<code>for</code> clause in a FLWOR expression
XPath <code>for</code> expression	no	<code>for</code> clause in a FLWOR expression
<code>xsl:variable</code>	yes	<code>let</code> clause in a FLWOR expression or global variable declaration
<code>xsl:sort</code>	yes	<code>order by</code> clause in a FLWOR expression
<code>xsl:if</code> , <code>xsl:choose</code>	yes	Conditional expressions (if-then-else)
Literal result elements	yes	Direct constructors
<code>xsl:element</code>	yes	Computed constructors
<code>xsl:attribute</code>	yes	Computed constructors
<code>xsl:function</code>	no	User-defined functions
Named templates	yes	User-defined functions
<code>xsl:value-of</code>	yes	An enclosed expression in curly braces inside an element constructor
<code>xsl:copy-of</code>	yes	The path or other expression that would appear in the <code>select</code> attribute
<code>xsl:sequence</code>	no	The path or other expression that would appear in the <code>select</code> attribute
<code>xsl:include</code>	yes	Module import
<code>xsl:template</code>	yes	No direct equivalent; can be simulated with user-defined functions

Walmsley, Priscilla, XQuery, O'Reilly, March 2007.

# XSLT vs. XQuery 2/2

## Zusammenfassung

### ■ XSLT – Vorteile

- Template Rules (Vorlagen)
- Formatierung (numbers, dates, times)
- Reguläre Ausdrücke für Zeichenkettenverarbeitung
- Mehrere Ausgabedokumente
- Verarbeitung von beliebigen Textdateien
- XML-Syntax ist erweiterbar und robust (aber umfangreich ...)
  - Geeignet für semistrukturierte Daten (gemischter Inhalt, generische Schemata) und für Transformationen

### ■ XQuery – Vorteile

- SQL-ähnliche FLWOR-Ausdrücke
- Einfacher zu optimieren
- Einfache Einbettung von XQuery in Java, C# etc., da keine XML-Syntax
  - Geeignet für strukturierte Daten (Struktur durch Schema fixiert)

### ■ Automatische Übersetzer zwischen XSLT und XQuery vorhanden!

Michael Kay:

***"XQuery is for query, XSLT is for transformation"***

# Inhalt

- Einführung in XSL
- XSLT Grundlagen
- XSLT 2.0/3.0 Erweiterungen
- XSLT versus XQuery
- Anhang I: Verarbeitungsmodell
- Anhang II: Stylesheets
- Anhang III: Template Rules
- Anhang IV: Beispiel

## Anhang I

# XSLT Verarbeitungsmodell

## Anhang I: Verarbeitungsmodell 1/3

### Mehrere Dokumente pro Transformationsprozess

- Mehrere Quelldokumente
  - können im Stylesheet über die `document()`-Funktion geladen werden
- Mehrere Stylesheets
  - durch `<xsl:include>` oder `<xsl:import>` einbindbar
- Mehrere Ergebnisdokumente
  - können mit der `<xsl:result-document>`-Anweisung erzeugt werden (ab XSLT 2.0)

## Anhang I: Verarbeitungsmodell 2/3

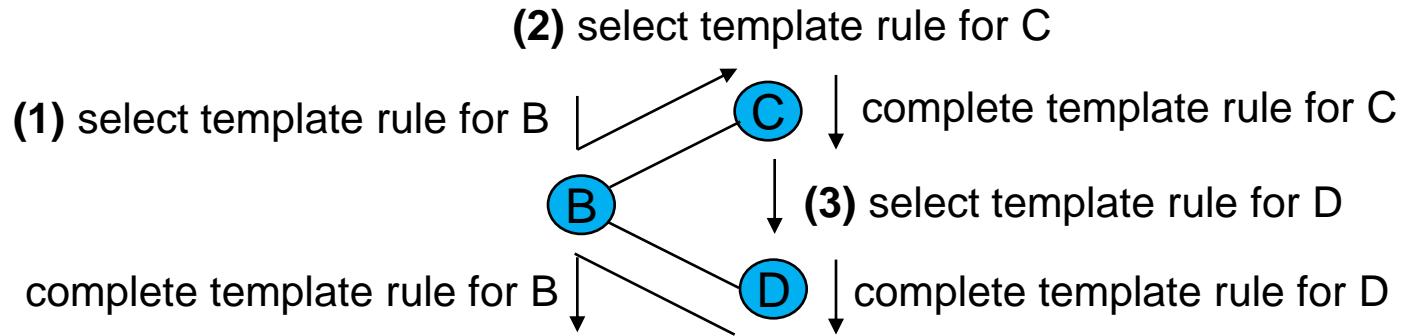
### Transformationsprozess innerhalb eines Stylesheets

- Der Transformationsprozess traversiert das Quelldokument und beginnt dabei an der Dokumentwurzel ("root node").
- Das Stylesheet wird nach einer Template Rule durchsucht, welche auf diesen Knoten anwendbar ist.
- Die Template Rule wird abgearbeitet ("instantiiert"), indem der der Template Rule entsprechende Teilbaum im Stylesheet traversiert wird.

## Anhang I: Verarbeitungsmodell 3/3

### Transformationsprozess innerhalb eines Stylesheets

- Eine Template Rule ist nicht vollständig abgearbeitet, solange ihr End-Tag nicht erreicht ist
  - mehrere Template Rules können dadurch zur gleichen Zeit offen sein



- Für jedes Element i.d. aktuellen Template Rule prüft der Prozessor, ob es sich um ein XSLT-Element handelt, d.h. über den XSLT-Namensraum definiert wurde
  - falls ja, wird der Output entsprechend der Template Rule aufbereitet
  - falls nein, wird das Element (z.B. ein HTML-Tag) unverändert in das Ergebnisdokument geschrieben ("literales Ergebniselement")

## Anhang II

# XSLT Stylesheets Aufbau und Modularisierung

## Anhang II: Stylesheets 1/4

### Aufbau

- XSLT-Namensraum muss immer angegeben werden  
Standardpräfix ist "xsl:"
- Stylesheet-Element (= Elementwurzel) muss Versions-Attribut aufweisen  
Zweck: Vorwärtskompatibilität

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" <!--
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 <xsl:output method="html" encoding="ISO-8859-1" indent="yes"/>

 <xsl:template match="/" <!-- -->
 </xsl:template>

</xsl:stylesheet>
```

- Definiert, in welcher Form der Baum des Ergebnisdokuments serialisiert werden soll - optional (`xml = default, xhtml, text, html`)

## Anhang II: Stylesheets 2/4

### Aufbau - Funktionen

- XML-Schema-Namensraum bei Verwendung von XSD-Funktionalität

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:fh="http://www.fh-ooe.at/se/ssd4/xslt/functions">

 <xsl:output method="html" encoding="ISO-8859-1" indent="yes"/>

 <xsl:template match="/">
 <xsl:value-of select="fh:factorial(5)" />
 </xsl:template>

 <xsl:function name="fh:factorial" as="xs:decimal">
 <xsl:param name="x" as="xs:decimal"/>
 <xsl:sequence select="
 if ($x le 1) then 1 else $x * fh:factorial($x - 1)" />
 </xsl:function>

</xsl:stylesheet>
```

- Bei benutzerdefinierten Funktionen: Eigenen Namensraum einbinden

## Anhang II: Stylesheets 3/4

### Einbindung in XML-Dokumente

#### ■ Externes referenziertes Stylesheet

- im XML-Dokument existiert eine Referenz auf ein externes Stylesheet

```
<?xml-stylesheet type="text/xml"
 href="styles/coursecatalog.xsl"?>
```

- wird über eine XML Processing Instruction eingebunden
- `type` gibt Sprache an, in der das Stylesheet geschrieben ist
- `href` gibt URI des zu verarbeitenden Stylesheets an

#### ■ Externes nicht-referenziertes Stylesheet

- es gibt keine Beziehung zwischen XML Dokument und Stylesheet
- beide werden unabhängig voneinander an den XSLT-Prozessor übergeben

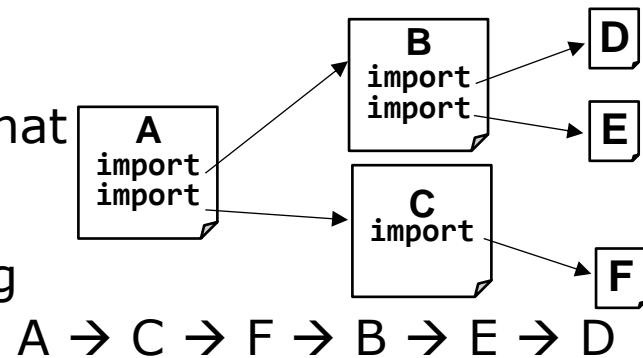
#### ■ Eingebettetes Stylesheet

- direkt im XML-Dokument

## Anhang II: Stylesheets 4/4

### Modularisierung

- Ein Stylesheet kann mittels `<xsl:include>` oder `<xsl:import>` andere Stylesheets referenzieren
  - deren Inhalt wird an der Stelle der Referenzierung expandiert
  - müssen als Subelemente von `<xsl:stylesheet>` spezifiziert werden
  - direkte oder indirekte Rekursion ist nicht erlaubt
  - Einsatz: unverändert inkludieren vs. Importiertes überschreiben
- Inkludieren `<xsl:include href="URI">`
  - inkludierte Stylesheets haben gleiche Priorität wie inkludierendes
  - bei Konflikt aufgrund gleicher Template Rules - Fehler
- Importieren `<xsl:import href="URI"/>`
  - bei Konflikt - importierendes Stylesheet hat Vorrang gegenüber importiertem
  - falls mehrere importiert werden, hat das als letztes importierte Stylesheet Vorrang



## Anhang III

# XSLT Template Rules

## Definition und Anwendung

## Anhang III: Definition von Template Rules 1/2

### xsl:template

#### ■ **match** (optional)

- XPath-Ausdrücke - beschränkt auf die Achsen child, attribute und descendant-or-self
- „Oder“-Verknüpfung durch | oder **union** realisierbar

```
<xsl:template
 match = "pattern"
 name = "qname"
 priority = "number"
 mode = "qname">
 <!-- (xsl:param*, template body) -->
</xsl:template>
```

#### ■ **name** - benanntes Template (optional)

- Aufruf über **<xsl:call-template name="qname"/>** als Alternative zu **pattern** matching
- das **match**-Attribut kann, muss aber in diesem Fall nicht angegeben werden
- Template Rule ist damit unabhängig von bestimmten Knoten

## Anhang III: Definition von Template Rules 2/2

### xsl:template

#### ■ priority (optional)

- falls mehrere Template Rules zu einem Knoten matchen, wird nur jene mit der höchsten **priority** verwendet (numerischer Wert)
- Default-Priorität wird automatisch berechnet gemäß des Spezialisierungsgrades von **match**
- Default-Prioritätswerte: -0,5 (generell - z.B. "\*") bis +0,5 (speziell - z.B. `course[@id="cID_8314"]`)
- beliebiges Ändern der Default-Priorität möglich
- Fehler bei gleicher Priorität

#### ■ mode (optional)

- ermöglicht die Definition mehrerer Template Rules für einen Knoten, wobei jedoch nur eine - je nach **mode** – abgearbeitet wird
- Beispiel: `mode="color"` bzw. `mode="b/w"`

## Anhang III: Anwenden von Template Rules 1/10

### xsl:apply-templates

- Kindknoten des aktuellen Knotens werden nicht automatisch verarbeitet
  - dies muss explizit über **xsl:apply-templates** spezifiziert werden
- Einfachste Form ist ein leeres Element
  - agiert nur als Platzhalter
  - bestimmt den Punkt, an dem der Prozessor alle Kindknoten des aktuellen Knotens mit weiteren passenden Template Rules verarbeiten und gegebenenfalls den entsprechenden Output ausgeben soll

```
<xsl:apply-templates
 select = "node-set-expr"
 mode = "qname">
 <!-- (xsl:sort |
 xsl:with-param)* -->
</xsl:apply-templates>
```

```
<template match="Course">
 <xsl:apply-templates/>
</template>
```

## Anhang III: Anwenden von Template Rules 2/10

### xsl:apply-templates

#### ■ **select** (optional)

- Template Rule wird nur auf ausgewählte Knoten angewendet
- spezifiziert durch XPath-Ausdruck, der eine Knotenmenge liefert
- die Reihenfolge der Verarbeitung der Knoten kann durch **xsl:sort** spezifiziert werden

```
<xsl:template match="Course">
 <course>
 <xsl:apply-templates select="Instructor" mode="xml">
 <xsl:sort select="@instructorNumber" order="ascending"/>
 </xsl:apply-templates>
 <xsl:apply-templates select="Title"/>
 </course>
</xsl:template>
```

```
<xsl:sort
 select = "node-set-expr"
 order = ("ascending" |
 "descending")>
 <!-- -->
</xsl:sort>
```

#### ■ **mode** (optional)

- wende nur Template Rules mit **mode="xml"** an

## Anhang III: Anwenden von Template Rules 3/10

### xsl:call-template

- Verarbeitet eine Template Rule mit dem spezifizierten Namen `name`
  - steuert daher explizit den Traversierungsprozess, ähnlich wie `xsl:for-each`  
(das über die Knotenliste iteriert)
  - Template Rules können rekursiv aufgerufen werden
- Der aktuelle Knoten sowie die aktuelle Knotenliste des XSLT-Prozessors ändert sich NICHT!
  - im Unterschied zu `xsl:apply-templates`

```
<xsl:call-template
 name = "qname">
 <!-- xsl:with-param -->
</xsl:call-template>
```

## Anhang III: Anwenden von Template Rules 4/10

### Auswertungskontext im XSLT-Prozessor

- Template Rule wird immer in einem Kontext instantiiert
- Kontext besteht aus zwei Komponenten:
  - **Aktueller Knoten** (current node)  
Knoten des Quelldokuments den der XSLT-Prozessor gerade verarbeitet und
  - **Aktuelle Knotenliste** (current node list)  
Knoten die sich aus einer Selektion durch `xsl:apply-templates` oder `xsl:for-each` ergeben haben
- Aus diesem Kontext ergibt sich der Auswertungskontext für XPath-Ausdrücke

## Anhang III: Anwenden von Template Rules 5/10

### Verarbeitungsformen

#### ■ **Pattern-Matching** (Push-basierter)-Ansatz

- Kindknoten werden nicht in der Template Rule des Elternknoten behandelt
- Verarbeitung dieser Kindknoten wird durch `<xsl:apply-templates>` an passende `<xsl:template>` Template Rules delegiert
  - Ausführung ist durch den Dokumentinhalt bestimmt („**Content-Driven**“)

#### ■ **Imperativer** (Pull-basierter)-Ansatz

- Kindknoten werden explizit selektiert und im Elternknoten behandelt

`<xsl:for-each>`  
`<xsl:value-of>`  
`<xsl:call-template>`

- Ausführung durch das Stylesheet bestimmt („**Program-Driven**“)

#### ■ Beide Ansätze können gemeinsam angewendet werden

# Anhang III: Anwenden von Template Rules 6/10

## Verarbeitungsformen - Beispiel

- XML: 

```
<p>It was a dark and <i>stormy</i> night.</p>
```

### Content-Driven (Push)

```
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
 <xsl:apply-templates/>
 </xsl:template>
 <xsl:template match="p">
 <para><xsl:apply-templates/></para>
 </xsl:template>
 <xsl:template match="b">
 <xsl:apply-templates/>
 </xsl:template>
 <xsl:template match="i">
 <Italics><xsl:apply-templates/></Italics>
 </xsl:template>
</xsl:stylesheet>
```

### Program-Driven (Pull)

```
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="p">
 <para>
 <xsl:for-each select="node()">
 <xsl:choose>
 <xsl:when test="self::text()">
 <xsl:value-of select="."/>
 </xsl:when>
 <xsl:when test="self::b">
 <xsl:value-of select="."/>
 </xsl:when>
 <xsl:when test="self::i">
 <Italics><xsl:value-of select="."/></Italics>
 </xsl:when>
 </xsl:choose>
 </xsl:for-each>
 </para>
 </xsl:template>
</xsl:stylesheet>
```

- Was passiert, wenn
  - **b** in *i* eingebettet wird?
  - **b** und *i* in ein von **p** verschiedenes Elternelement eingeschlossen wird?
  - ein neues *inline*-Element hinzugefügt wird?

# Anhang III: Anwenden von Template Rules 7/10

## Parameterisierung

- Globale und lokale Parameter (`name`) mit Defaultwerten (`select` - optional)
  - Global: Definition direkt unterhalb der Elementwurzel (als "Top-Level Elemente")
  - Lokal: Definition innerhalb einer Template Rule

```
<xsl:template match="image">
 <xsl:param name="align" select="'left'"/>

</xsl:template>
```

- Übergabe eines Wertes für einen lokalen Parameter über `xsl:with-param` erfolgt bei
  - `xsl:apply-templates` oder
  - `xsl:call-template`

```
<xsl:param
 name = "qname"
 select = "expression"
 <!-- template body -->
/>
```

Default-Wert -  
XPath

Zugriff auf  
Parameterwert  
über XPath;  
z.B. als Attribut-  
wert verwendet  
**"Attribute Value  
Template"**

## Anhang III: Anwenden von Template Rules 8/10

### Parameterisierung - Übergabe

- **xsl:with-param** Elemente müssen direkte Subelemente sein von
  - **xsl:apply-templates** oder
  - **xsl:call-template**
- **name** (obligatorisch)
  - **xsl:template** erhält den Parameterwert über diesen Namen
- **select** (optional)
  - ein XPath-Ausdruck liefert den Parameterwert
  - **xsl:with-param** ist in diesem Fall ein leeres Element
  - ist **xsl:with-param** kein leeres Element, darf es kein **select** Attribut haben, stattdessen steht der Parameterwert im Elementinhalt
- Beispiel:

```
<xsl:with-param
 name = "qname"
 select = "expr."
 <!-- template body -->
</xsl:with-param>
```

```
<xsl:apply-templates select="images/image[1]">
 <xsl:with-param name="align" select="'right'" />
</xsl:apply-templates>
```

## Anhang III: Anwenden von Template Rules 9/10

### xsl:value-of

#### ■ Elementinhalt

- Quelldokument:

<para>All the <emph>text</emph> here is the value</para>

- Template Rule: <xsl:value-of select=".."/>
- Output: All the text here is the value

```
<xsl:value-of
 select="expr."
/>
```

aktuelles Element



#### ■ Attributwerte

- Quelldokument:

<para type="secret">This is a secret.</para>

- Template Rule:

```
<xsl:template match="para">
 <P><xsl:text>[TYPE: </xsl:text>
 <xsl:value-of select="@type"/>
 <xsl:text>] </xsl:text>
 <xsl:apply-templates/></P>
 </xsl:template>
```

- Output:

<P>[TYPE: secret ] This is a secret. </P>

## Anhang III: Anwenden von Template Rules 10/10

### Built-In Template Rules

- Für jeden der **7 Knotentypen des XPath-Datenmodells** existiert eine "Built-In Template Rule" - Diese kommt zum Einsatz, wenn für einen Knoten **keine passende Template Rule** gefunden wird
- Im Falle eines **leeren Stylesheets** führen die Built-In Template Rules dazu, dass der Inhalt des Dokuments ohne Markup ausgegeben wird
- Element- und Wurzelknoten (root node)
  - für jedes Subelement wird implizit **xsl:apply-templates** aufgerufen, wodurch passende Template Rules ausgeführt werden
- Textknoten
  - werden durch **xsl:value-of** in das Ergebnisdokument übernommen
- Kommentar-, Namensraum- und Processing-Instruction-Knoten werden nicht übernommen

```
<xsl:template match="*|/*">
 <xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template match="text() | @*|>
 <xsl:value-of select="."/>
</xsl:template>
```

```
<xsl:template match="comment()|processing-instruction()"/>
```

## Anhang IV

# XSLT Stylesheets

## Beispiel: Kurskatalog

# Anhang IV: Beispiel 1/6

CourseCatalog.xml

```
<CourseCatalog year="2019" term="summer" campus="Hagenberg">
 <DegreeProgramme code="0307" name="Software Engineering" abbreviation="SE">
 <Course id="cID_8314" semesterHours="1" language="en" semester="4">
 <Title>Introduction to semi-structured data models and XML</Title>
 <Description>Introduction of skills related to XML.<Content>Includes DTD, Schema, XPath, XQuery, XSLT, JSON</Content>
 <Exam>Final Exam required.</Exam>Participation without any previous knowledge.
 </Description>
 <Credit formatType="ECTS">1</Credit>
 <CourseType type="Lecture"/>
 <Date startDate="--02-28" endDate="--05-03"/>
 <Time startTime="08:00:00" endTime="10:25:00" day="THU"/>
 <Room roomNumber="1.004" building="FH1">CELUM HS2</Room>
 <Instructor instructorNumber="p22080">Julian Haslinger</Instructor>
 </Course>
<!-- ... -->
```

## Kurskatalog: 0307 (Software Engineering)

### Kurse

ID	Typ	Raum / Location	Titel
cID_7555	Training	White House and Moskovskij Kreml	Intercultural Communications
cID_8314	Lecture	FH1.004	Introduction to semi-structured data models and XML
cID_8315	LabSession	FH3.108	Introduction to semi-structured data models and XML

Aufgabe:  
Transformation von  
CourseCatalog.xml in eine  
HTML-Repräsentation



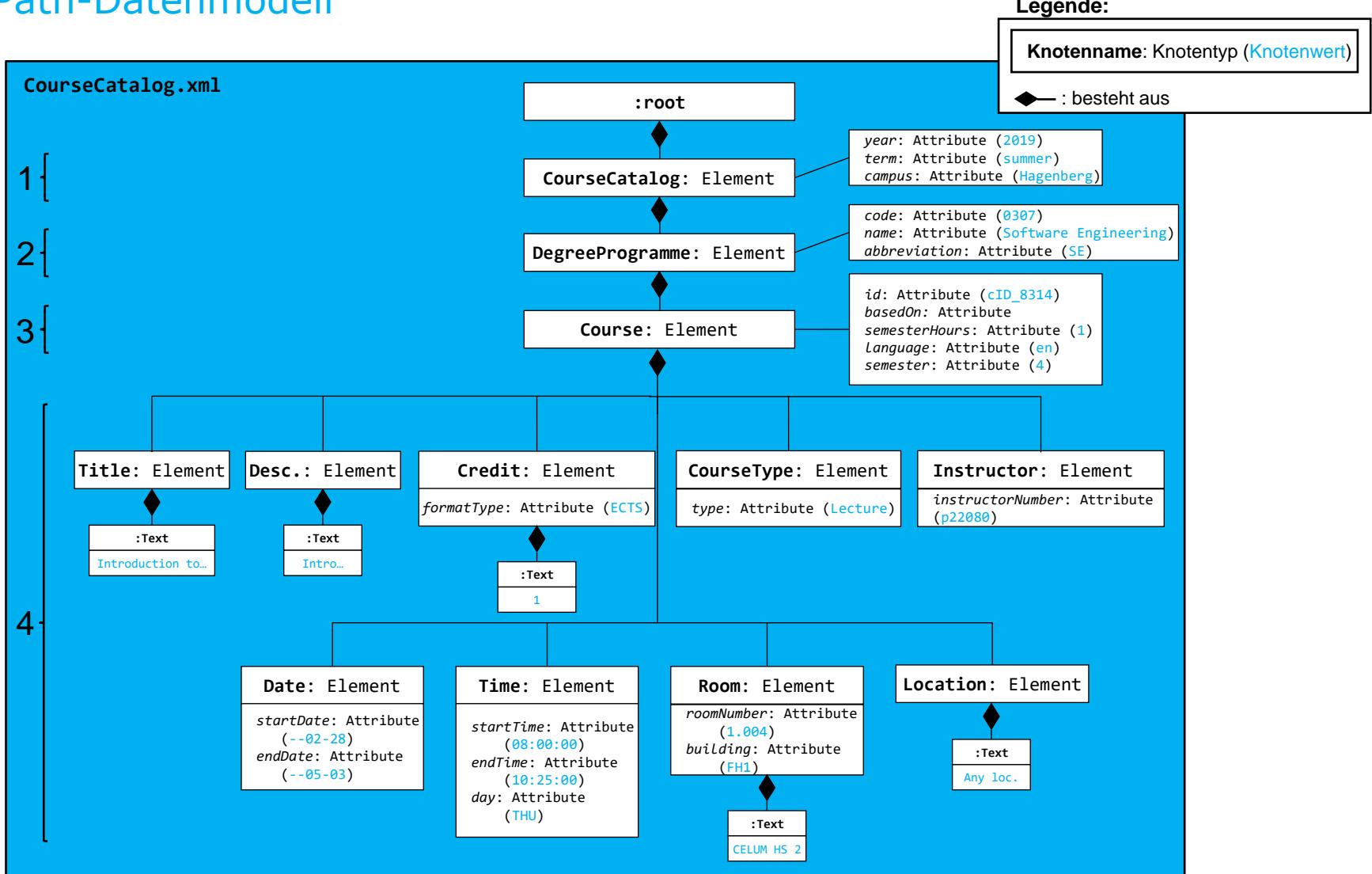
## Kurskatalog: 0458 (Medical and Bioinformatics)

### Kurse

ID	Typ	Raum / Location	Titel
cID_8840	Lecture	FH2.010	Data modelling and database design

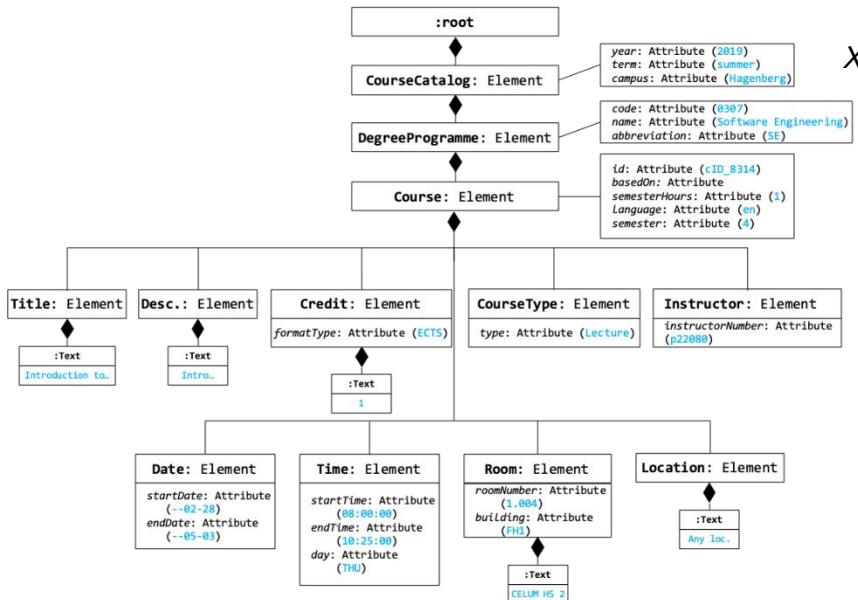
# Anhang IV: Beispiel 2/6

## XPath-Datenmodell



## Anhang IV: Beispiel 3/6

### Stylesheet



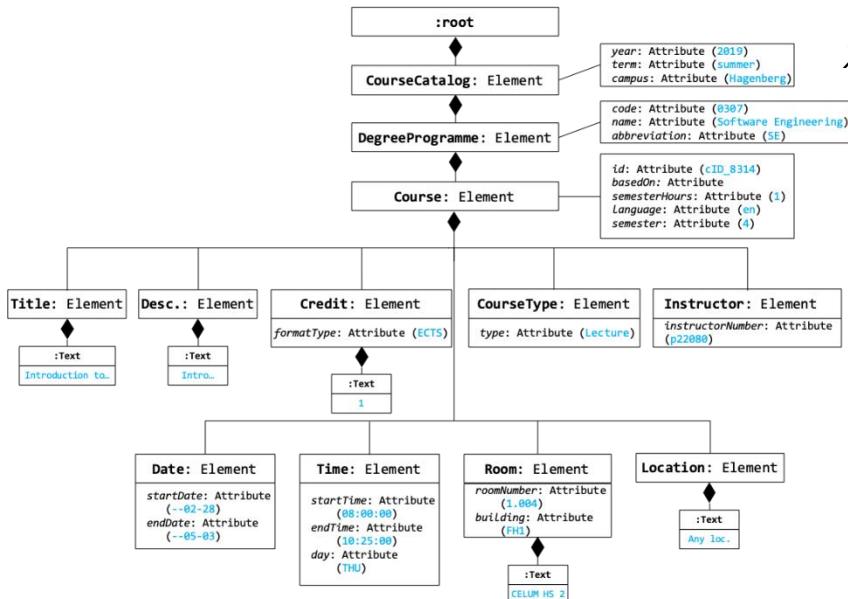
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="html" encoding="ISO-8859-1" indent="yes"/>

 <xsl:template match="/">
 <html>
 <head/>
 <body>
 <xsl:apply-templates/>
 </body>
 </html>
 </xsl:template>

 <xsl:template match="CourseCatalog">
 <xsl:apply-templates/>
 </xsl:template>
```

# Anhang IV: Beispiel 4/6

## Stylesheet



```

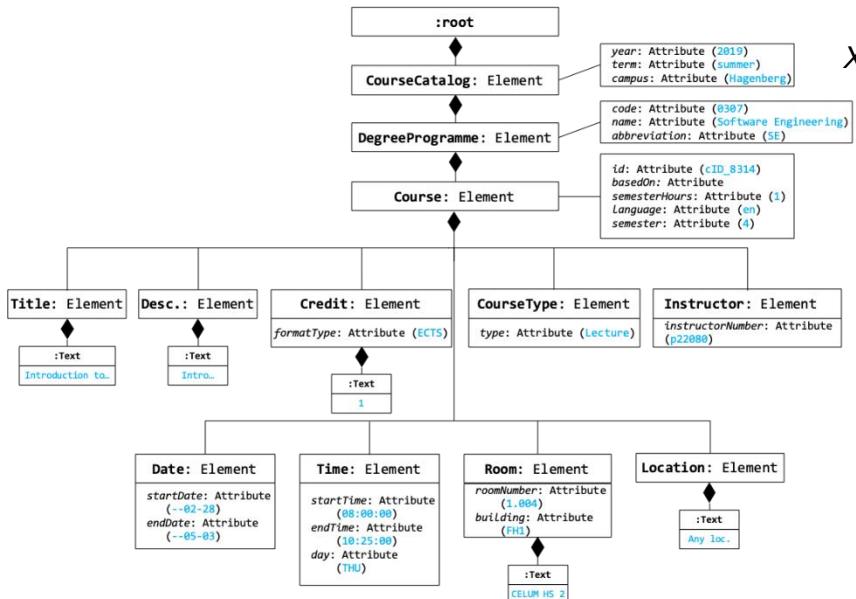
<xsl:template match="DegreeProgramme">
 <h1>Kurskatalog: <xsl:value-of select="@code"/> (<xsl:value-of select="@name"/>)</h1>
 <h2>Kurse</h2>
 <table border="1">
 <tbody>
 <tr bgcolor="#00B0F0"> [Redacted]
 <th>ID</th>
 <th>Typ</th>
 <th>Raum / Location</th>
 <th>Titel</th>
 </tr>

 <xsl:apply-templates select="Course">
 <xsl:sort select="@id"/>
 </xsl:apply-templates>
 </tbody>
 </table>
</xsl:template>

```

# Anhang IV: Beispiel 5/6

## Stylesheet



```

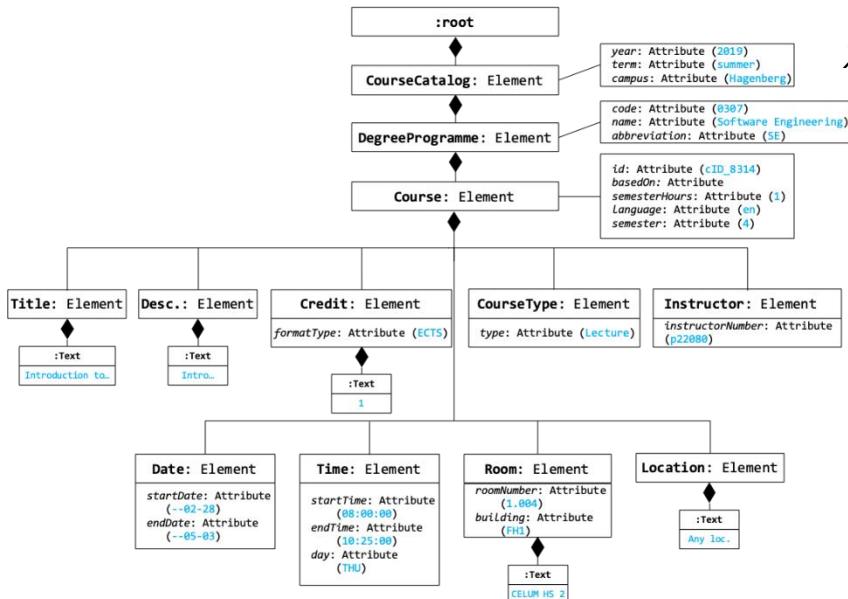
<xsl:template match="Course">
 <tr>
 <td>
 <xsl:value-of select="@id"/>
 </td>
 <xsl:apply-templates select="CourseType"/>
 <xsl:apply-templates select="Room | Location"/>
 <xsl:apply-templates select="Title"/>
 </tr>
</xsl:template>

<xsl:template match="Room">
 <td>
 <xsl:value-of select="concat('FH', @roomNumber)"/>
 </td>
</xsl:template>

```

# Anhang IV: Beispiel 6/6

## Stylesheet



```

<xsl:template match="Location">
 <td>
 <xsl:value-of select=". "/>
 </td>
</xsl:template>

<xsl:template match="CourseType">
 <td>
 <xsl:value-of select="@type"/>
 </td>
</xsl:template>

<xsl:template match="Title">
 <td>
 <xsl:value-of select=". "/>
 </td>
</xsl:template>
</xsl:stylesheet>

```

# XSLT 2.0 – PDF-Transformation

Julian Haslinger

# XSLT: PDF-Transformation

## Beispiel: CourseCatalog

- Benötigte Dateien:
  - Quell-Dokument
    - ◆ *CourseCatalog.xml*
  - XSLT-Stylesheet mit Templates zur Erstellung von PDF-Dokumenten (u.a. XSL-FO-Elemente)
    - ◆ *CourseCatalog\_PDF.xslt*
- → Bereits in Beispiel-Projekt vorhanden:  
**M0\_XML\_CourseCatalog-XQuery-XSLT**

# XSLT 2.0 – PDF-Transformation mit Oxygen XML Editor

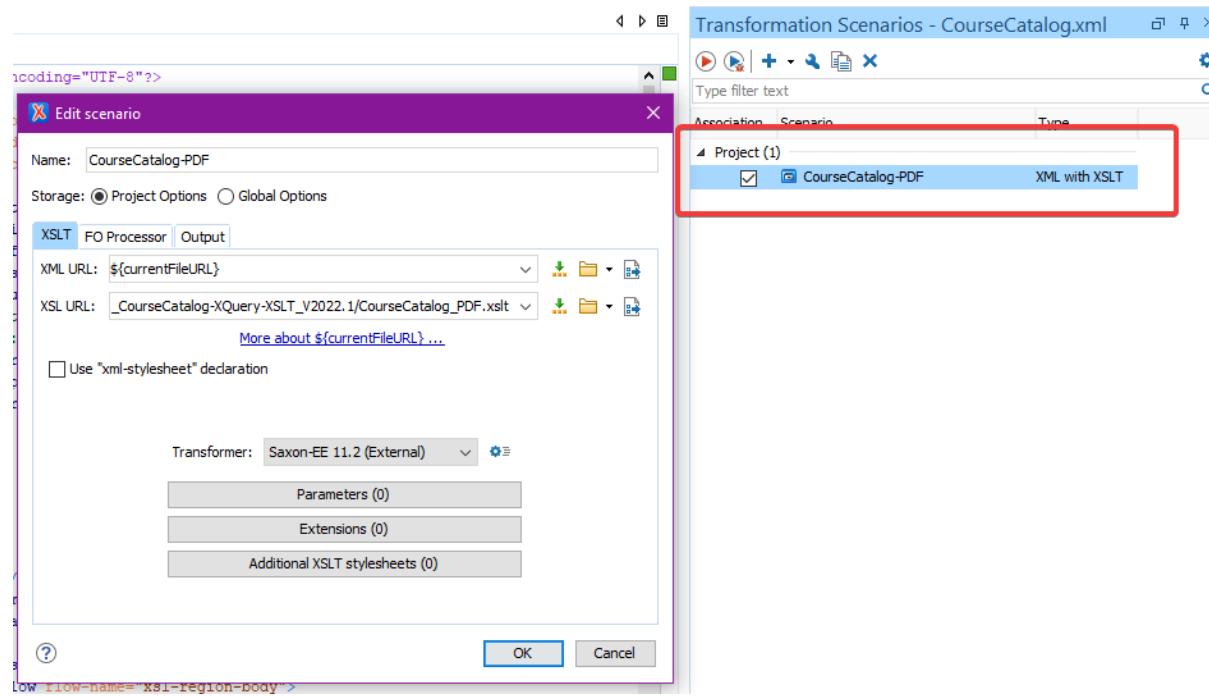
Julian Haslinger



# XSLT: PDF-Transformation

Beispiel: CourseCatalog (Oxygen XML Editor)

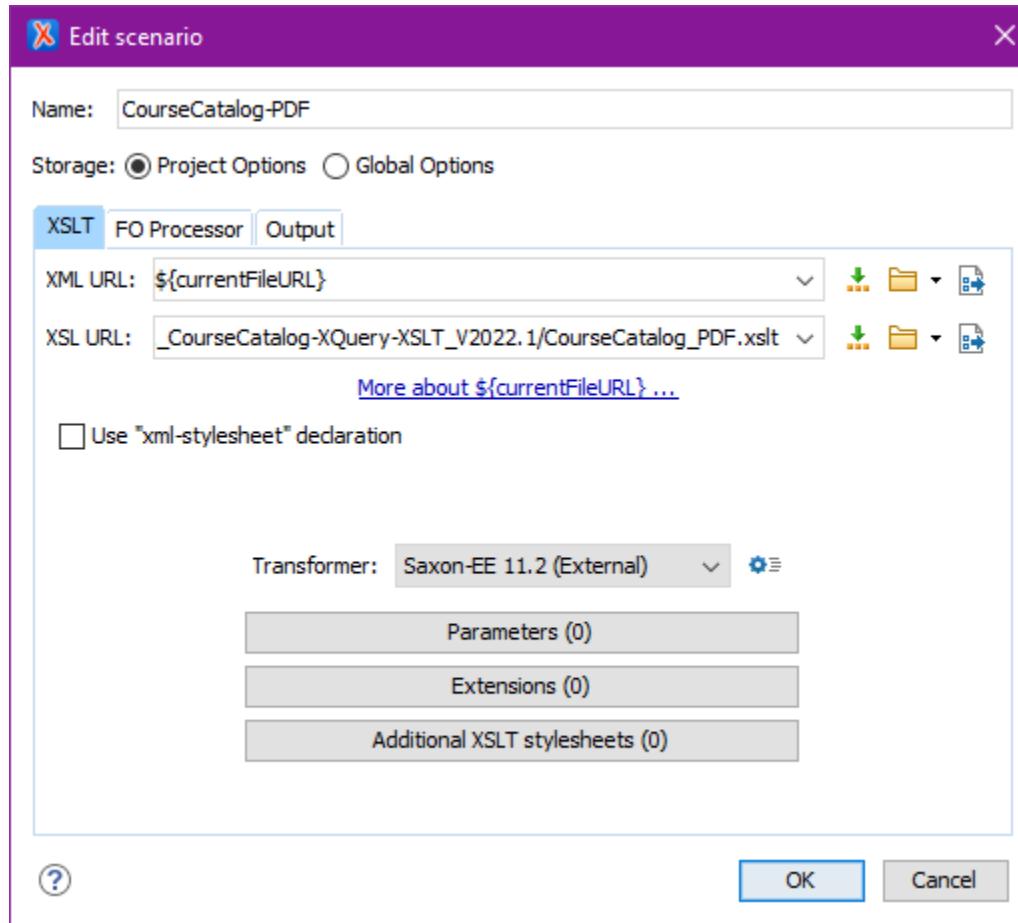
- *Transformation Scenario* erstellen



# XSLT: PDF-Transformation

Beispiel: CourseCatalog (Oxygen XML Editor)

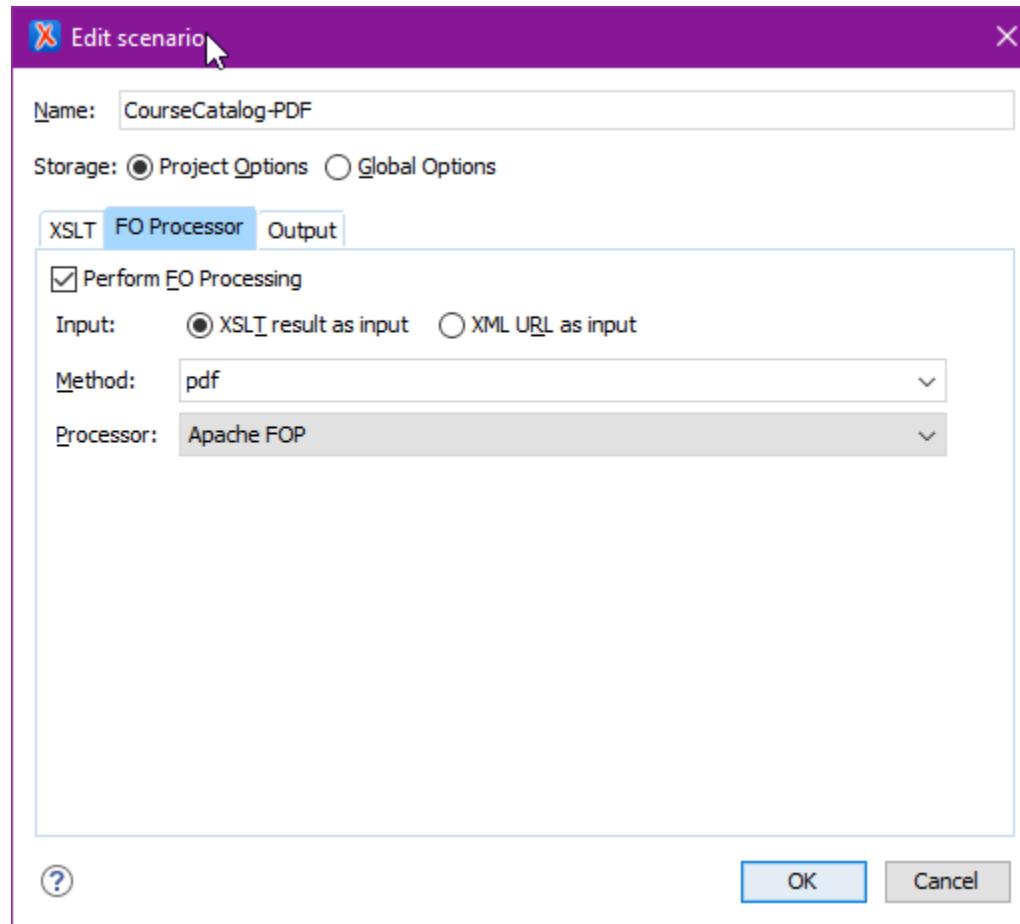
- Einstellungen *Transformation Scenario - XSLT*



# XSLT: PDF-Transformation

Beispiel: CourseCatalog (Oxygen XML Editor)

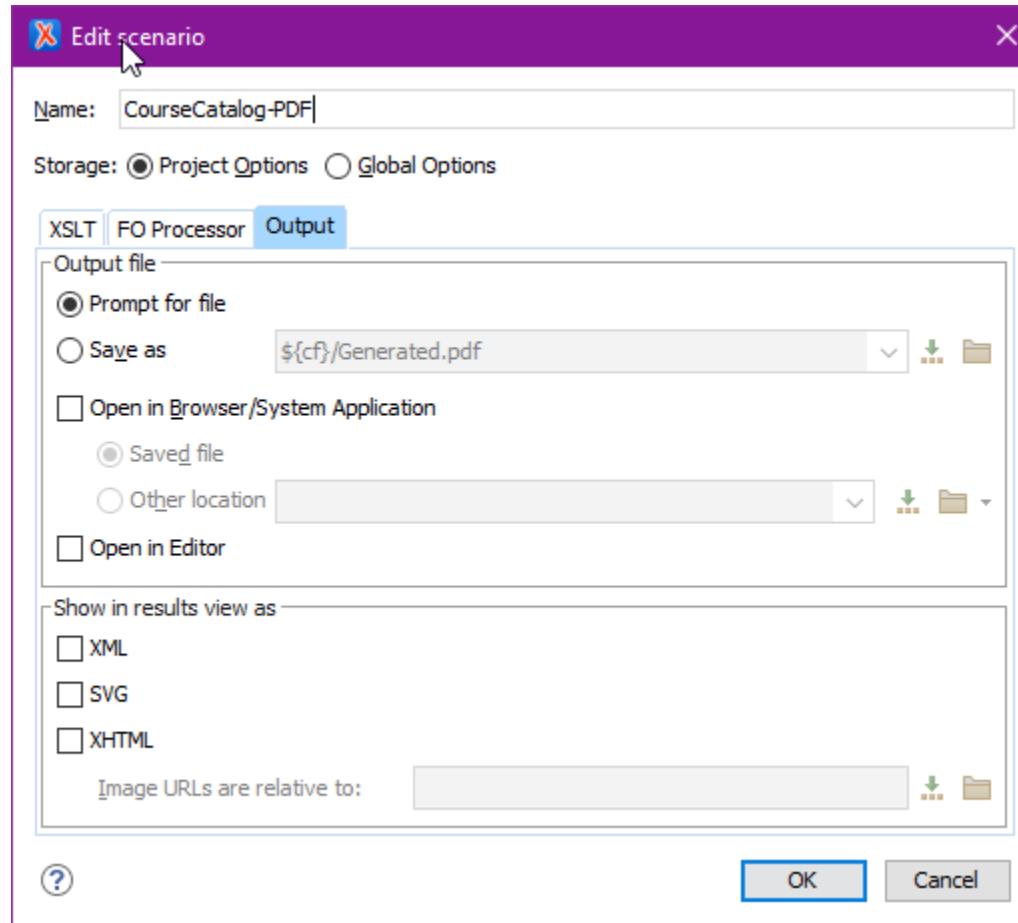
- Einstellungen *Transformation Scenario – FO Processor*



# XSLT: PDF-Transformation

Beispiel: CourseCatalog (Oxygen XML Editor)

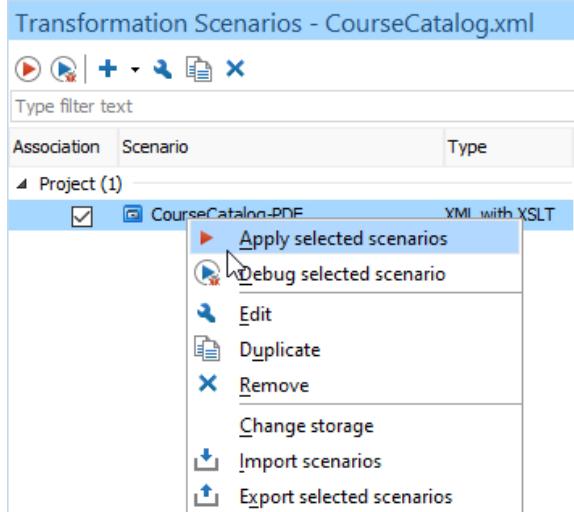
- Einstellungen *Transformation Scenario - Output*



# XSLT: PDF-Transformation

Beispiel: CourseCatalog (Oxygen XML Editor)

- Transformation Scenario ausführen und PDF-Datei speichern



Kurskatalog (Studiengang "Software Engineering")

LVA-Nummer	Typ	Raum	Titel	LVA-Leiter
cID_8314	Lecture	FH1.004	Introduction to semistructured data models and XML	p22080 - Julian Haslinger
cID_8315	LabSession	FH3.108	Introduction to semi-structured data models and XML	p20623 - Barbara Traxler
cID_7555	Training	(-)	Intercultural Communications	p22100 p22101 - (no more info)

Kurskatalog (Studiengang "Medical and Bioinformatics")

LVA-Nummer	Typ	Raum	Titel	LVA-Leiter
cID_8840	Lecture	FH2.010	Data modelling and database design	p20621 - Josef Altmann

# XSLT 2.0 – PDF-Transformation mit Altova XMLSpy

Julian Haslinger



# XSLT: PDF-Transformation

## Beispiel: CourseCatalog (Altova XMLSpy)

- Projekt öffnen

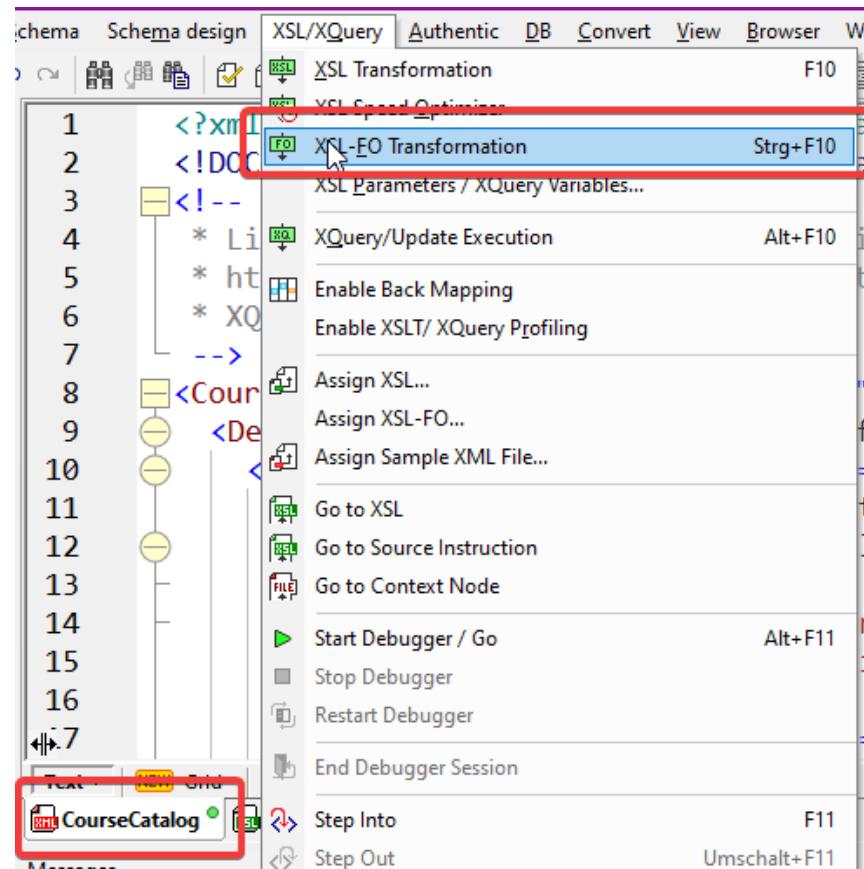
The screenshot shows the Altova XMLSpy interface with the following details:

- Project Explorer:** Shows the "CourseCatalog" project with several XML and XSL files listed.
- Main Editor:** Displays an XSLT template (CourseCatalog.xslt) with XML code. The code includes a DTD reference, comments about licensing, and a specific course entry for "Introduction to semi-structured data models and XML".
- Bottom pane:** Shows the generated XML output for the course entry, including fields like Title, Description, and Credit.
- Status bar:** Shows the XMLSpy version (Enterprise Edition v2022 rel. 2), registration information, and system status (Ln 11, Col 31).

# XSLT: PDF-Transformation

Beispiel: CourseCatalog (Altova XMLSpy)

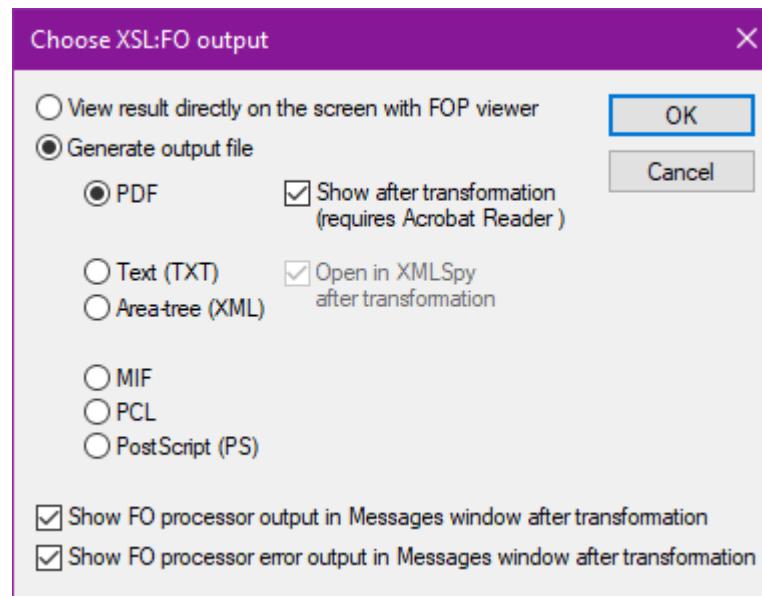
- XML-Datei auswählen und
  - XSL/XQuery – XSL-FO Transformation wählen



# XSLT: PDF-Transformation

Beispiel: CourseCatalog (Altova XMLSpy)

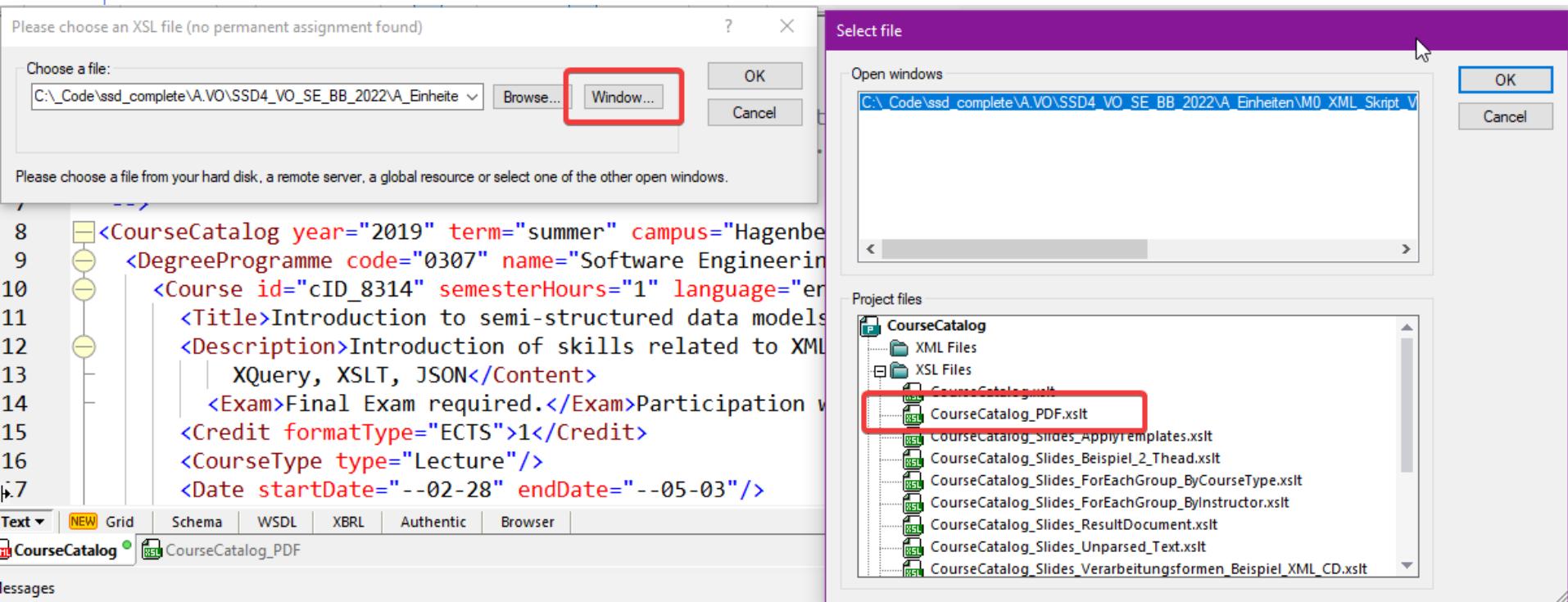
- XSL-FO Transformation - Einstellungen



# XSLT: PDF-Transformation

Beispiel: CourseCatalog (Altova XMLSpy)

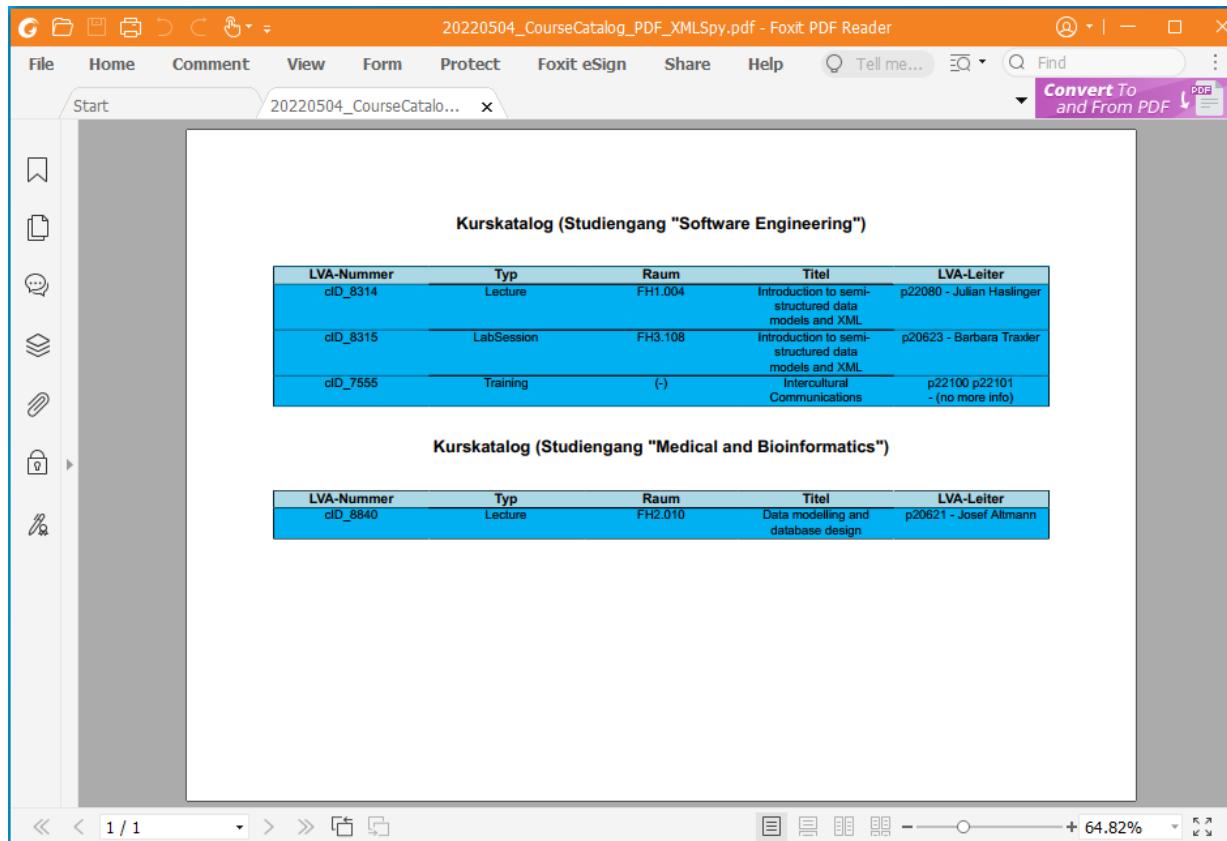
- XSLT-Datei für die Transformation wählen



# XSLT: PDF-Transformation

Beispiel: CourseCatalog (Altova XMLSpy)

- Dateinamen wählen → PDF wird erzeugt und geöffnet



# XPath, XQuery, XSLT 3.0+

Julian Haslinger

<?xml?>

Der Foliensatz basiert u.a. auf:

- W3C: XPath 3.0 / 3.1 Recommendation
- W3C: XQuery 3.0 / 3.1 Recommendation
- W3C: XSLT 3.0 Recommendation
- Online verfügbar auf [www.w3.org](http://www.w3.org)



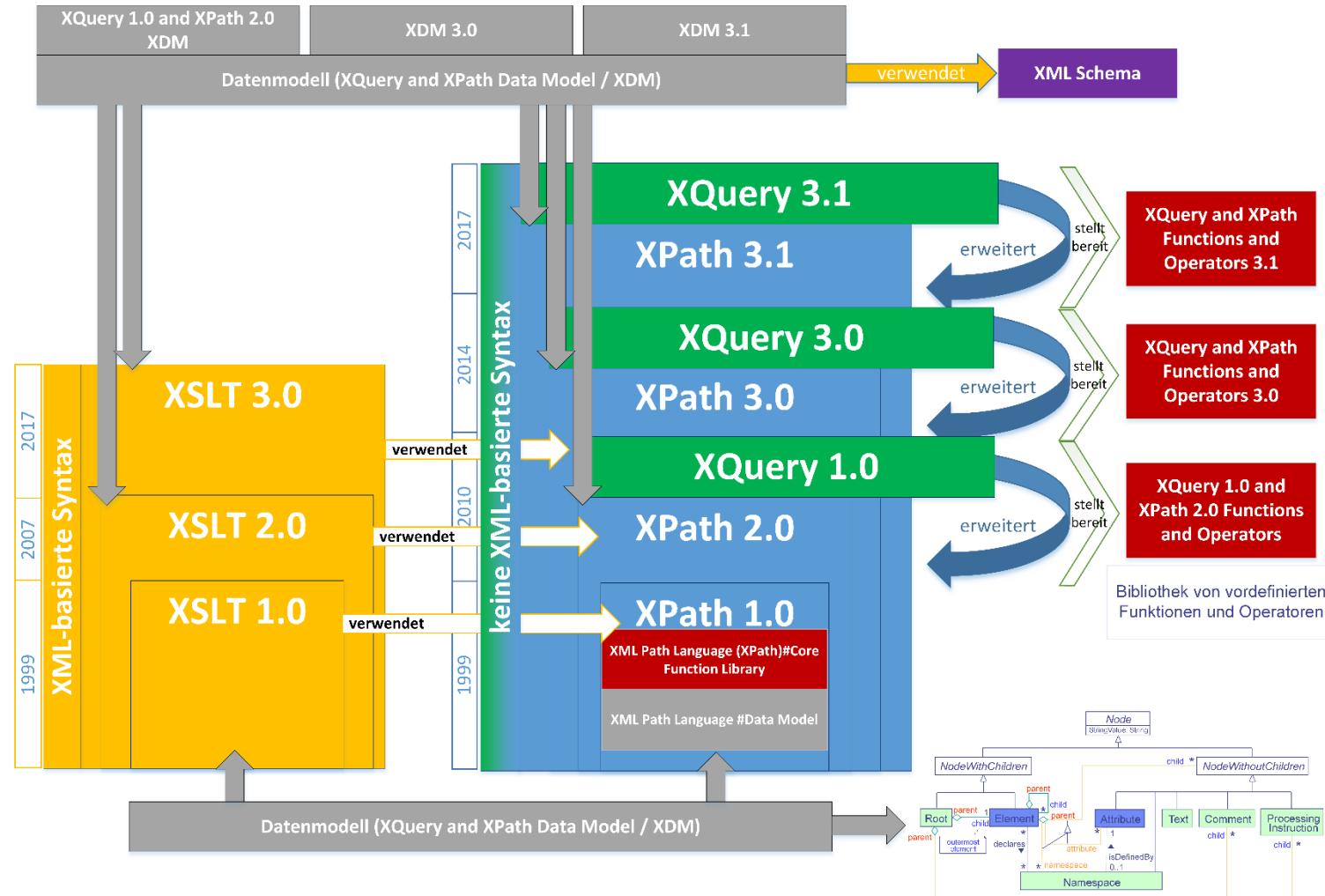
# Inhalt

---

- Einführung
- XQuery and XPath Data Model
- Funktionen höherer Ordnung
- XQuery and XPath Functions and Operators
- Maps und Arrays
- JSON-Verarbeitung
- XPath 3.0+
- XQuery 3.0+
- XSLT 3.0
- Literatur / W3C-Spezifikationen

# XPath, XSLT und XQuery

## Versionen und Zusammenhang



# Inhalt

- Einführung ✓
- XDM – XQuery and XPath Data Model
- Funktionen höherer Ordnung
- XQuery and XPath Functions and Operators
- JSON-Verarbeitung
- XPath 3.0 / 3.1
- XQuery 3.0 / 3.1
- XSLT 3.0

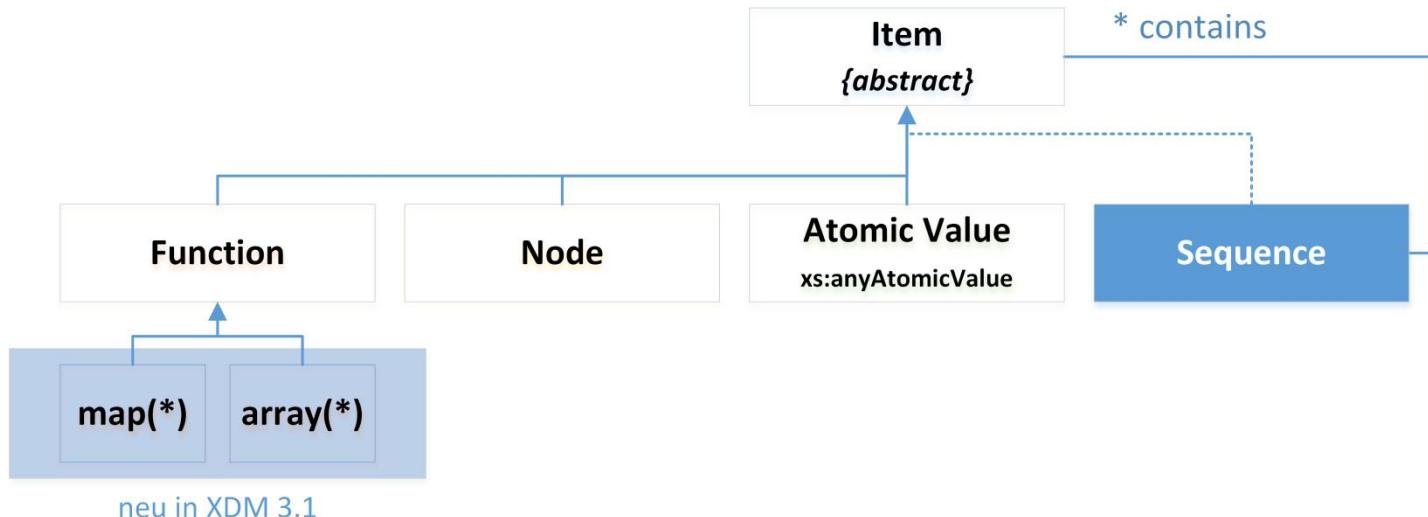
**XDM**

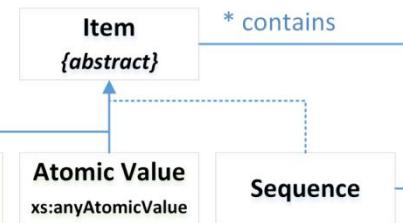
XQuery and XPath Data Model

# XDM 3.0/3.1

## XQuery and XPath Data Model

- Datenmodell für
  - XPath 3.0+, XQuery 3.0+, XSLT 3.0
- Function Items
  - Maps und Arrays (beide *immutable*)



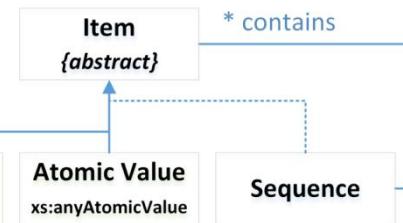


## XDM 3.0/3.1

### Funktionen als eigener Datentyp

- Funktionen sind vollwertige Bestandteile in Typ-Hierarchie
- Beschreibung von Funktionen höherer Ordnung
  - Funktionen, die andere Funktionen als Argumente hat oder eine Funktion zurückgibt
- Dynamischer Funktionsaufruf
  - (Anonyme) Funktionen können zuvor einer Variablen zugewiesen und danach aufgerufen werden
  - Beispiel:
    - let \$f := function(\$a as xs:double, \$b as xs:double)  
 as xs:double { \$a \* \$b }
    - return \$f(10,10) → 100
    - ...Typangaben optional, aber **empfohlen**
- Ausdrücke für Inline-Funktionen
  - Anonyme Funktionen können innerhalb einer anderen Funktion implementiert und verwendet werden

let \$f := function(*Argumente*  
*Rumpf*  
*Rückgabewert*  
 as xs:double { \$a \* \$b } )  
 return \$f(10,10) → 100  
 ...Typangaben optional, aber **empfohlen**



## XDM 3.0/3.1

### Funktionen als eigener Datentyp

- Funktionen werden als „*function items*“ (siehe XDM), oder auch „Funktionen höherer Ordnung („*lambdas*“) definiert (siehe nächstes Kapitel)
  - Funktionen können somit nicht nur einfach aufgerufen, sondern auch
    - ◆ anonym deklariert (*anonymous functions*) und an Variablen gebunden,
    - ◆ innerhalb anderer Funktionen deklariert,
    - ◆ als Parameter an Funktionen übergeben,
    - ◆ als Rückgabewerte von Funktionen definiert oder
    - ◆ teilweise angewandt werden (*partially applied functions*).

# Inhalt

---

- Einführung ✓
- XDM – XQuery and XPath Data Model ✓
- Funktionen höherer Ordnung
- XQuery and XPath Functions and Operators
- JSON-Verarbeitung
- XPath 3.0 / 3.1
- XQuery 3.0 / 3.1
- XSLT 3.0

# Funktionen höherer Ordnung

# Funktionen höherer Ordnung

## Basics

- Inline- bzw. anonyme Funktion
  - 1. Neue anonyme Funktion deklarieren, z.B.
    - ◆ let \$func := function(\$a){\$a}
    - ◆ let \$func := function(\$a as xs:string){\$a}
  - 2. Funktions-Item verwenden
    - ◆ return \$func('Hello, World!')
  - Ergebnis: „**Hello, World!**“
- Vordefinierte Funktionen verwenden
  1. Funktion an Variable binden (Angabe von Stelligkeit/*arity*, da es mehrere Implementierungen einer Funktion geben kann)
    - ◆ let \$f := math:pow#2
  2. Funktions-Item verwenden / Funktion aufrufen
    - ◆ return \$f(5, 2)
  - Ergebnis: **25**

# Funktionen höherer Ordnung

## Basics

- Teilweise angewandte Funktion (partially applied functions)
  - Im ersten Schritt werden nur einige der nötigen Argumente übergeben (den Rest mit ? angeben), um im zweiten Schritt die Funktion anzuwenden

```
let $f := function($a, $b, $c)
 {$a => concat($b) => concat($c)},
$paf := $f(?, ':', ?)
 ↑ ↑
 | |
return $paf("Hallo", ' Student!')
```

*Neue Funktion \$paf mit 2 Platzhalter*

# Funktionen höherer Ordnung

## Basics

- Teilweise angewandte Funktion (partially applied functions)

```
let $multiply :=
 function($base, $number) { $base * $number },
```

```
 $fMultiplyBy10 := $multiply(10, ?)
```

*Neue Funktion  
\$fMultiplyBy10  
mit 2 Platzhalter*

```
return
```

```
for-each(1 to 10, $fMultiplyBy10)
```

*for-each wendet für jedes Item (in der Sequenz) die angegebene Funktion an.*

# Funktionen höherer Ordnung

## Basics

- Funktionen als Parameter übergeben

- Funktionen können nicht nur aufgerufen, sondern, durch die vollständige Integration in das XQuery- und XPath-Datenmodell, u.a. auch als Parameter übergeben werden.

```
let $main := function($f1, $items)
 { for $i in $items return $f1($i)},
 $uc := function($str) {$str},
 $arr := ['a','b','c']
return $main($uc, $arr)
```

*Funktion wird als Argument übergeben und aufgerufen*

*Funktion*      *Array*       $\Rightarrow$  ['a','b','c']

# Implementierung von Funktionen

## Tipps

- Funktionen typisieren (Parameter / Argument, Rückgabewerte)
  - Standard: `item()*`
- Wiederverwendbarkeit von Funktionen ausnutzen!
  - Datentypen sinnvoll ausnutzen und überlegen, ob bestimmte Funktionen nicht für mehrere Datentypen verwendet werden können („generisch“).
  - Beispiel
    - ◆ `function($a as xs:string){$a}` oder
    - ◆ `function($a as item()*){$a}`?

# Grundlegende Higher-order functions

## (Auszug)

- **fn:for-each(\$seq as item()\*, \$f as function(item()) as item()\*) as item()\***
  - Wendet die Funktion \$f auf jedes Item in der Sequenz \$seq an.
- **fn:filter(\$seq as item()\*, \$f as function(item()) as xs:boolean) as item()\***
  - Liefert die Items aus \$seq zurück, für die die Auswertung von \$f true ergibt.
- **fn:fold-left(\$seq as item()\*, \$zero as item()\*, \$f as function(item(), item()) as item()\*) as item()\***
  - Verarbeitet die Sequenz \$seq von links nach rechts, wendet dabei die Funktion \$f nacheinander auf die Items und und baut dabei einen akkumulierten Rückgabewert auf.
- **fn:for-each-pair(\$seq1 as item()\*, \$seq2 as item()\*, \$f as function(item(), item()) as item()\*) as item()\***
  - Wendet Funktion \$f für jedes Paar aus den beiden übergebenen Sequenzen an. Terminierte, sobald durch die kürzere Sequenz iteriert wurde.
- **fn:sort**
- **fn:apply(\$function as function(\*), \$array as array(\*)) as item()\***
  - Wendet Funktion \$function auf das Array \$array an
  - fn:apply(fn:concat#3, ["a", "b", "c"]) → "abc"

# Inhalt

- Einführung ✓
- XDM – XQuery and XPath Data Model ✓
- Funktionen höherer Ordnung ✓
- XQuery and XPath Functions and Operators
  - Maps und Arrays
  - Weitere Neuerungen
- JSON-Verarbeitung
- XPath 3.0 / 3.1
- XQuery 3.0 / 3.1
- XSLT 3.0

FO

XQuery and XPath Functions and  
Operators

## FO 3.0/3.1

### *XQuery and XPath Functions and Operators*

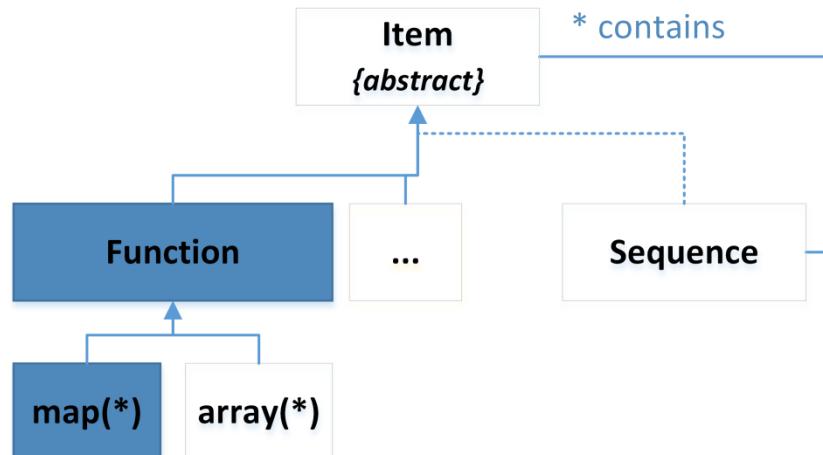
- Funktionsbibliothek legt die zu implementierenden (Datentyp-)Funktionen und Operatoren fest
  - Müssen / sollen von XPath-, XSLT- und XQuery-Implementierungen unterstützt werden
  - Konkrete Syntax der einzelnen Funktionen und Operatoren wird von der jeweiligen Zielsprache definiert
- Aktuelle Versionen
  - *XQuery and XPath Functions and Operators 3.0* (2014)
  - *XQuery and XPath Functions and Operators 3.1* (2017)

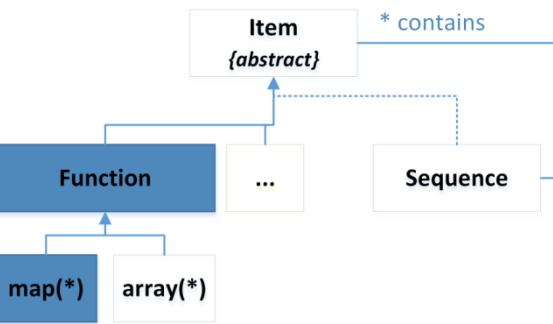
# Maps und Arrays

# Maps

(oder: *dictionaries, assoziative arrays...*)

- Map dient zur Speicherung von Schlüssel-Werte-Paaren
  - Erzeugen einer Map: `map{"1": "FH"}`
  - Zugriff u.a. wie Funktionsaufruf
    - ◆ `map{"1": "FH"}("1")` → „FH“
  - Jeder Schlüssel (atomarer Wert) kann nur einmal vorkommen, Werte können beliebige Sequenzen sein (z.B. auch Funktionen)

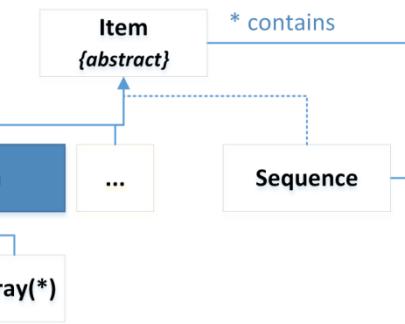




## Maps

- Vordefinierte Funktionen für Maps (11):
  - <https://www.w3.org/TR/xpath-functions-31/#map-functions>, u.a.
- Standard-Funktionen können um eigene Funktionen erweitert werden.

op:same-key	map:find
map:merge	map:put
map:size	map:entry
map:keys	map:remove
map:contains	map:for-each
map:get	



# Maps

## Funktionen auf Arrays - Auswahl

- **map:put(...)**

- Fügt einen neuen Eintrag an der angegebenen Stelle ein.  
Falls der Eintrag bereits vorhanden ist, wird er überschrieben
  - ◆ let \$m := map{1:'a'}
  - ◆ return map:put(\$m, 2, 'b')
  - ◆ → {1:'a', 2:'b'}

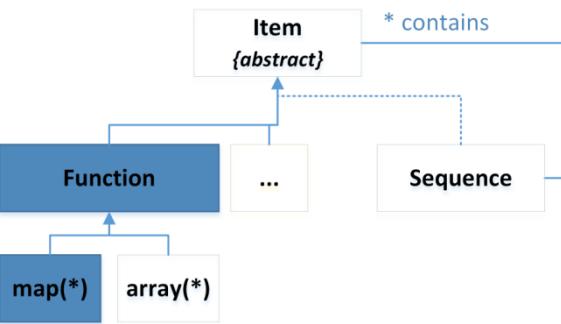
- **map:get(...)**

- Gibt den Eintrag an der angegebenen Stelle zurück.

```
let $m := map{1:'a',
 'Adder': function($one, $two){$one + $two}}
```

```
return map:get($m, 'Adder')(3,4)
→ 7
```

gibt Funktion  
zurück, die später  
aufgerufen wird



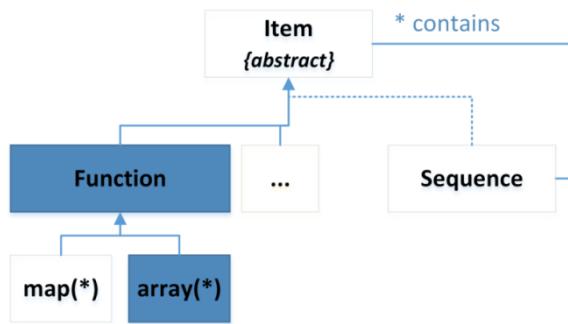
# Maps

## Funktionen auf Arrays - Auswahl

- **map:keys(...)**
  - Gibt alle Schlüssel in der Map zurück

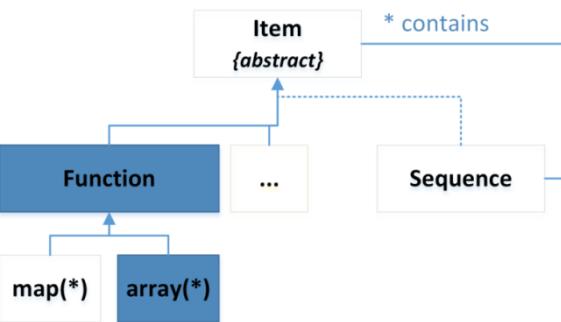
```
let $m := map{1:'a', 2:'b', 3:'c'}
return map:keys($m)
→ 1,2,3
```
- **map:remove(...)**
  - Löscht alle Einträge mit dem übergebenen Schlüssel(n)

```
let $m := map{1:'a', 2:'b', 3:'c'}
return map:remove($m, map:keys($m))
→ {} (leere Map)
```



## Arrays

- Geordnete Werteliste
  - Werte sind über deren Index [1..AnzahlElemente] referenzierbar
  - Erzeugen eines Arrays z.B.:
    - ◆ `array{5, 10, 6, 2}` oder
    - ◆ `[ 5, 10, 6, 2 ]` oder
    - ◆ `[1 to 3]` (Array mit einer Sequenz von 1 – 3, `[(1,2,3)]`)
- Arrays können geschachtelt werden
  - `[1,2,3,4,[45],[100]]` → Array mit zwei weiteren Arrays
  - `array:flatten([1,2,3,4,[45],[100]])` liefert „flaches“ Array



## Arrays

- Array kann Elemente mit unterschiedlichen Datentypen aufnehmen

$\text{xs:integer}$        $\text{xs:string}$        $\text{function(item())*} \text{ as } \text{xs:string}$   
↓           ↓           ↓  
[1, 2, 'Hallo', function(\$a){lower-case(\$a)}](4)

- → Gibt Funktion an Stelle 4 zurück (kann danach weiterverwendet werden)
- [1, 2, 'Hallo', function(\$a){lower-case(\$a)}](4)('FH HAGENBERG')
  - ◆ → 'fh hagenberg'

## Arrays

- Vordefinierte Funktionen für Arrays (18):
  - <https://www.w3.org/TR/xpath-functions-31/#array-functions>, u.a.
- Standard-Funktionen können um eigene Funktionen erweitert werden.

array:size	array:reverse
array:get	array:join
array:put	array:for-each
array:append	array:filter
array:subarray	array:fold-left
array:remove	array:fold-right
array:insert-before	array:for-each-pair
array:head	array:sort
array:tail	array:flatten

# Arrays

## Funktionen auf Arrays - Auswahl

- Leeres Array: []
- array:size(...):
  - array:size([1,2,'Hallo', function(\$a){lower-case(\$a)}])
  - Array mit Zahlen / Zeichenkette und Funktion
- array:get(...) – Position im Array (beginnend bei 1)
  - array:get([1,2,3,4], 2) → 2
  - Zugriff mit neuem Lookup-Operator: [1,2,3,4]?2 → 2
  - Verwendung wie Funktionsaufruf: [1,2,3,4](2) → 2
- array:put(...)
  - Erzeugt neues Array mit neuem Wert an übergebener Position
  - array:put([1,2,3,4], 3, 10) → [1,2,10,4] (vorhandene Stellen werden überschrieben)

# Arrays

## Funktionen auf Arrays - Auswahl

- `array:append(...):`
  - `array:append([1,2], 'FH Hagenberg')` → [1,2,'FH Hagenberg']
- `array:head(...)` / `array:tail(...):`
  - `array:head([1,2,3,4])` → 1
  - `array:head([1,2,3,4])` → 4
- `array:reverse(...):`
  - `array:reverse([1,2,3])` → [3,2,1]
- `array:sort(...):`
  - `array:sort([2,1.1,3])` → [1.1,2,3]
  - `array:sort([2,1.1,3,'a'])` → Fehler!

# Arrays

## Funktionen auf Arrays - Auswahl

- **array:filter(...):**

- Erzeugt ein Array mit den Elementen aus dem Ursprungs-Array, für die eine Filter-Funktion true zurückliefert.
- `array:filter([1,2,3], function($item){$item < 3})` → [1,2]
- `array:filter([1,2,3], function($item){$item mod 3 = 0})` → [3]
- ...oder auch:
  - `let $f := function($item){$item < 3},`
  - `$a := [1,2,3,4,5]`
  - `return array:filter($a, $f)`

Variablen werden deklariert,  
Funktion wird an eine Var. gebunden

```
array:filter($array as array(*),
 $function as function(item()* as xs:boolean) as array(*)
```

# Arrays

## Funktionen auf Arrays - Auswahl

- **array:for-each-pair(...):**
  - Für zwei Arrays wird paarweise für Elemente eine Funktion ausgeführt.
    - ◆ Terminiert, sobald alle Elemente vom kleineren Array abgearbeitet wurden.
  - `let $a1 := ['A', 'B', 'C', 'D'],`
  - `$a2 := [3, 2, 1],`
  - `$f := function($item1, $item2){concat($item1, $item2)}`
  - `return array:for-each-pair($a1, $a2, $f)`
  - **→ ['A3', 'B2', 'C1']**

*Variablen werden deklariert,  
Funktion wird an eine Var. gebunden*

```
array:for-each-pair($array1 as array(*),
 $array2 as array(*),
 $function as function(item()* , item()*) as item()*) as array(*)
```

# Lookup-Operator für Maps und Arrays (?)

- Einfachere Rückgabe von Werten in Maps und Arrays („syntactic sugar“) - Syntax: <datenkomponente>?<spezifizierer>
- Arrays (z.B. `let $arr := ([[1,2,3], [1,2,5], [1,2]])`)
  - Auswahl wohl Werten in geschachtelten Arrays
    - ◆ `return $arr?3?2`
  - Drittes geschachteltes Array wird ausgewählt
    - ◆ `return $arr?2?*`
  - Alle Werte werden zurückgegeben
    - ◆ `return $arr?*`
- Maps (z.B. `let $map := map {"Mo" : "1", "Di" : "2", "Mi" : "3"}`)
  - Wert zum Schlüssel "Mo"
    - ◆ `return $map?Mo`
    - ◆ `return map:get($map, "Mo")`
  - Alle Werte werden zurückgegeben
    - ◆ `return $map?*`

# Weitere Neuerungen

# Operatoren

## Pfeil-Operator „=>“

- Wendet eine Funktion auf einen Wert an
  - (Teilweise) komplexe Schachtelung (bei Funktionsaufrufen) kann verringert werden
- Der Wert wird als erstes Argument für den Funktionsaufruf verwendet
- „Syntactic sugar“ für normale Funktionsaufrufe
  - Ohne Pfeil-Operator:
    - ◆ `function($value, $argumente)`
  - Mit Pfeil-Operator:
    - ◆ `$value => function($argumente)`
- Beispiel: "fh hagenberg" => `upper-case()` => `contains('X')` → false
  - (XPath 2.0: `contains(upper-case("fh hagenberg"), "X")`)

# Operatoren

## „!“ und „||“

- Mapping-Operator „!“ (auch: *bang / map operator*)
  - Syntax: E1 ! E2
  - Für jedes Item in Ausdruck E1 wird der Ausdruck E2 angewendet (ähnlich for-Ausdruck)
    - ◆ Beispiel: (1 to 2) ! (. + 4) → 5, 6
- Konkatenations-Operator ( || )
  - Verknüpfen von Zeichenketten
    - ◆ Beispiel: "Hello" || ' ' || "World" → ,Hello World‘
  - Syntaktisch gleich zu fn:concat(\$s1, \$s2)

# Inhalt

- Einführung ✓
- XDM – XQuery and XPath Data Model ✓
- Funktionen höherer Ordnung ✓
- XQuery and XPath Functions and Operators ✓
  - Maps und Arrays
  - Weitere Neuerungen
- JSON-Verarbeitung
- XPath 3.0 / 3.1
- XQuery 3.0 / 3.1
- XSLT 3.0

# JSON-Verarbeitung

## Verarbeitung von JSON-Zeichenketten 1/2

- Möglichkeit zur Verarbeitung von JSON
  - `parse-json()`, `json-doc()`
- Funktion `parse-json()`
  - Verarbeitet eine JSON-Zeichenkette
  - Ergebnis wird in Map/Array gespeichert
- Funktion `json-doc()`
  - Verarbeitet eine externe JSON-Ressource
  - Ergebnis wird in Map/Array gespeichert
- Zugriff auf JSON u.a. mit dem Lookup-Operator  
„?“

## Verarbeitung von JSON-Zeichenketten 2/2

- Mehrere Funktionen zum Umgang mit JSON-Dateien
- fn:parse-json
  - Liest JSON ein – Rückgabe als Array/Map
  - `parse-json('{"Eintrag":{"Titel":"Beispiel-Glossar"}}')?Eintrag?Titel`  
→ „Beispiel-Glossar“
- fn:json-doc
  - Liest eine externe JSON-Ressource ein
- fn:json-to-xml
  - Liefert XML-Repräsentation zu JSON zurück
  - `fn:json-to-xml('{"Eintrag":{"Titel":"Beispiel-Glossar"}}')`
  - → XML-Repräsentation von JSON-Zeichenkette
- fn:xml-to-json
  - Liefert JSON-Repräsentation zu XML zurück
  - `fn:xml-to-json(`  
 `fn:json-to-xml`  
 `('{"Eintrag":{"Titel":"Beispiel-Glossar"}}'))`

# Inhalt

- Einführung ✓
- XDM – XQuery and XPath Data Model ✓
- Funktionen höherer Ordnung ✓
- XQuery and XPath Functions and Operators ✓
- JSON-Verarbeitung ✓
- XPath 3.0 / 3.1
- XQuery 3.0 / 3.1
- XSLT 3.0

# XPath 3.0 / 3.1

## Neuerungen im Überblick

### W3C-Standards:

- XPath 1.0, Nov. 1999, ~ 44 Seiten
- XPath 2.0, Jan. 2007, ~ 250 Seiten
- XPath 3.0, Apr. 2014
- XPath 3.1, März 2017

# XPath 3.0/3.1

## Neuerungen (Auswahl)

- Rückblick XPath 1.0 und XPath 2.0
  - XPath 1.0 (November 1999)
    - ◆ 4 Datentypen: node, boolean, string und number
    - ◆ Eingebaute Funktionsbibliothek: 27 Funktionen
      - [XML Path Language \(XPath\) #corelib](#)
  - XPath 2.0 (Jänner 2007, Dezember 2010)
    - ◆ Umfassenderes Typ-System (Typen von XML-Schema bekannt)
    - ◆ Erweiterte Funktionsbibliothek: ca. 100 Funktionen
      - [XQuery 1.0 and XPath 2.0 Functions and Operators \(Second Edition\)](#)
    - ◆ Wichtig: Sequenz-Konzept

# XPath 3.0

## Neuerungen (Auswahl)

- XPath 3.0 (April 2014)
  - Erweiterte Funktionsbibliothek (über 200)
  - Funktionen höherer Ordnung
    - ◆ Dynamischer Funktionsaufruf
    - ◆ Inline-Funktionen und Erstellung eigener Funktionsbibliotheken
  - Mapping- bzw. „Bang“-Operator „!“
  - Konkatenationsoperator „||“

# XPath 3.1

## Neuerungen (Auswahl)

- XPath 3.1 (März 2017)
  - Unterstützung für neue komplexe Datentypen `map` und `array`
    - ◆ In vorigen XPath-Versionen waren Elementstrukturen und Sequenzen die einzigen komplexen Datentypen
    - ◆ Als leichtgewichtige Datenstrukturen zur einfacheren Verwendung eingeführt (vor allem bei Kombination von XML/JSON-Verarbeitung)
  - Pfeil-Operator „=>“

# Inhalt

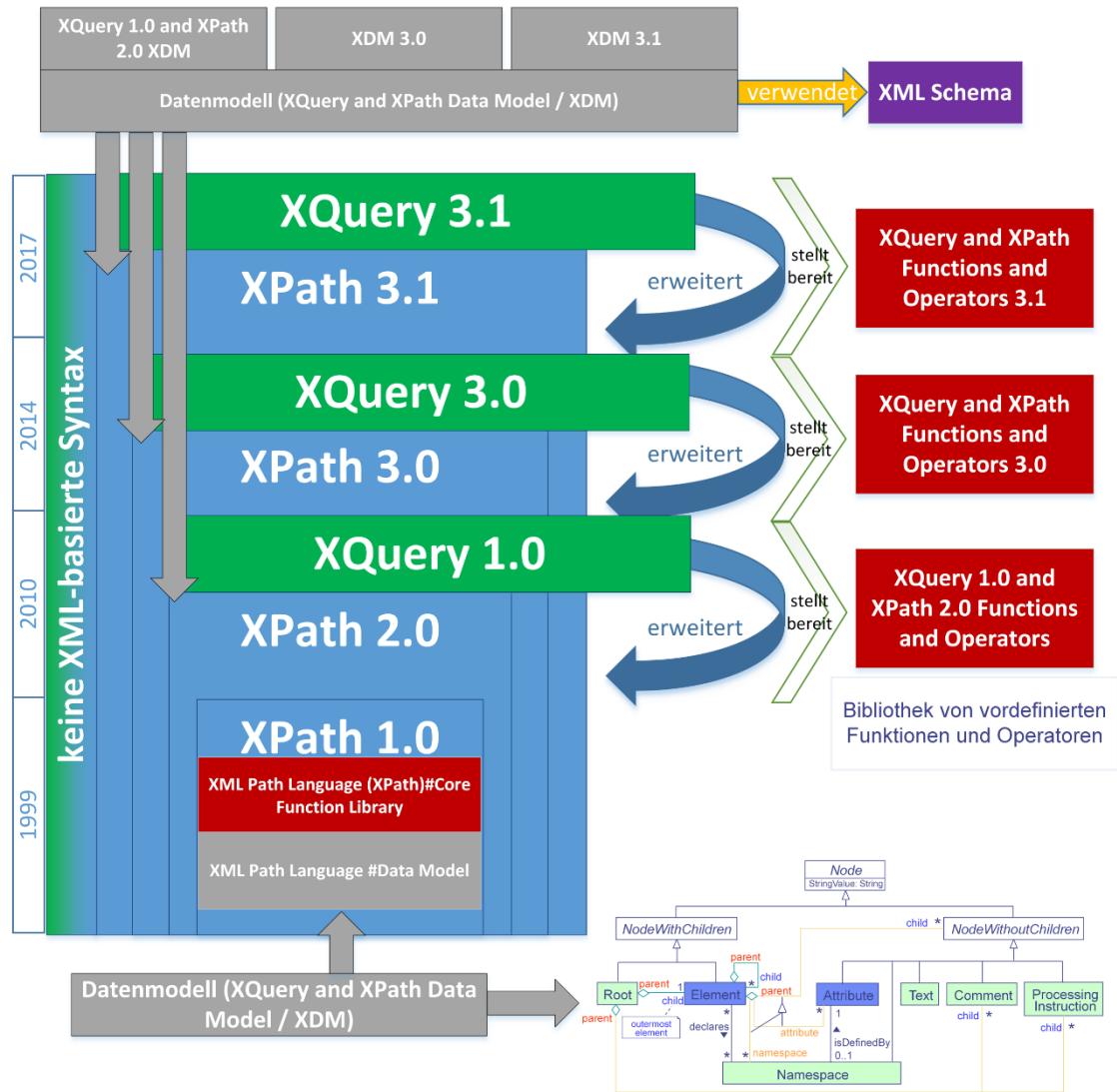
- Einführung ✓
- XDM – XQuery and XPath Data Model ✓
- Funktionen höherer Ordnung ✓
- XQuery and XPath Functions and Operators ✓
- JSON-Verarbeitung ✓
- XPath 3.0 / 3.1 ✓
- XQuery 3.0 / 3.1
- XSLT 3.0

# XQuery 3.0 / 3.1

Neuerungen im Überblick

# XQuery 3.0 / 3.1

- baut auf XPath auf bzw. erweitert es
- verwendet XDM
- implementiert FO
  - (Functions and Operators)
- erweitert XQuery 1.0 (es gibt keine Version 2.0 von XQuery)

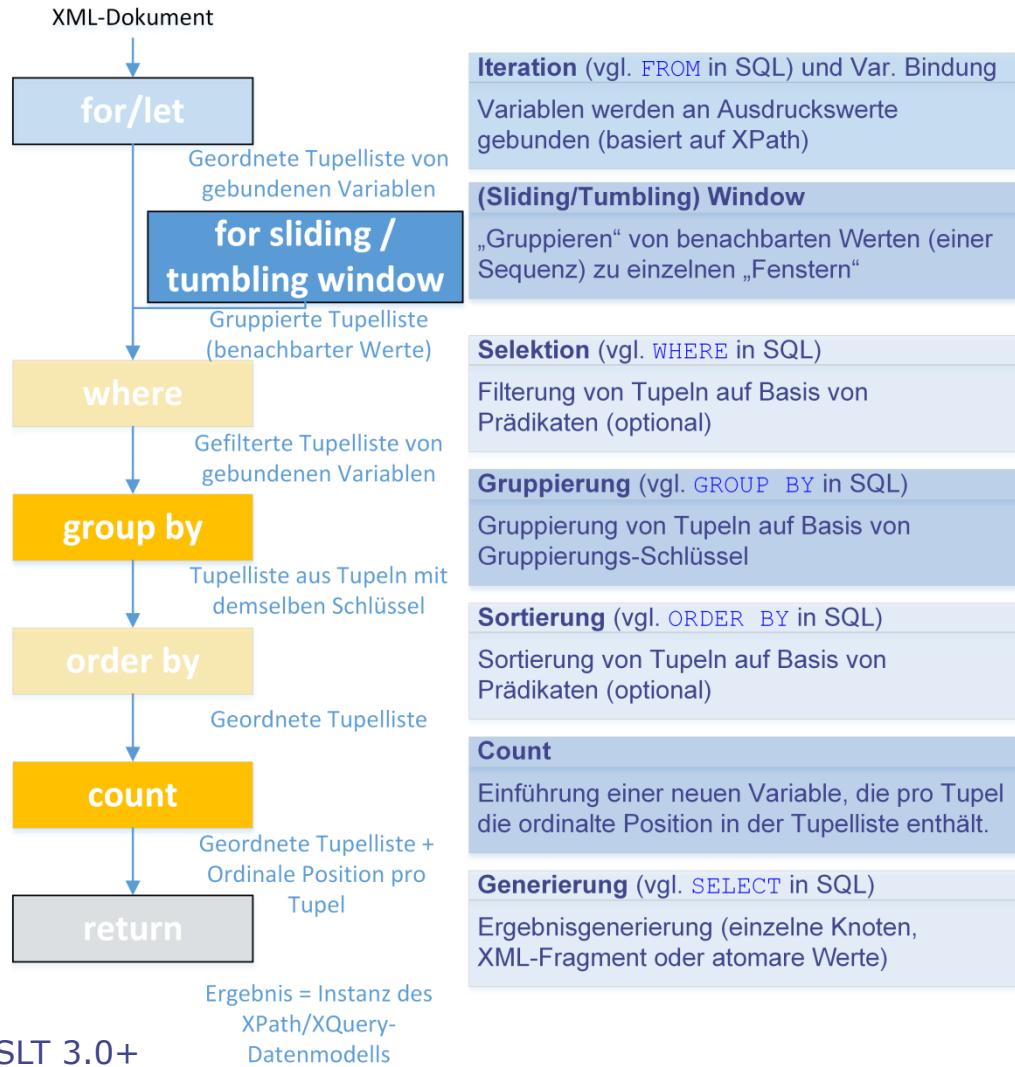


## XQuery 3.0 / 3.1

- Wichtigste Neuerungen in XQuery 3.0 und 3.1
  - Erweiterung der FLWOR-Ausdrücke
    - ◆ u.a. group by, count, (sliding / tumbling) window...
  - Vollständige, funktionale Programmiersprache
    - ◆ u.a. Funktionen höherer Ordnung
  - Unterstützung für neue Datentypen map und array (ab XQuery 3.1)
    - ◆ mit vielen Standard-Funktionen zu deren Verarbeitung
  - Unterstützung von JSON (ab XQuery 3.1)
    - ◆ Parsen, Serialisieren...
  - Unterstützung von neuen Operatoren
    - ◆ !, || (XQuery 3.0)
    - ◆ ?, => (XQuery 3.1)

# XQuery 3.0

## Erweiterung - FLWOR ['flower'] Ausdruck



# XQuery 3.0

## Erweiterung - FLWOR ['flower'] – group by



### ■ „group by“ – Klausel

- Werte-basierte Gruppierung, basierend auf Gruppierungs-Schlüssel

```
xquery version "3.0";
for $n in 1 to 10
group by $mod := $n mod 3
return
 if ($mod = 0) then
 <mod0>{$n}</mod0>
 else if ($mod = 1) then
 <mod1>{$n}</mod1>
 else
 <other>{$n}</other>
```

*Ergebnis: Alle Items,  
die über diese Eigenschaft  
gruppiert wurden*

mod0  
mod1  
other

<mod0>3 6 9</mod0>  
<mod1>1 4 7 10</mod1>  
<other>2 5 8</other>



# XQuery 3.0

## Erweiterung - FLWOR ['flower'] – window

- Zur Weiterverarbeitung der Daten werden einzelne Tupel-Sequenzen gebildet, die auf eine angegebene Fenster-Funktion passen
  - Jedes Tupel ist dabei ein Fenster (window)
- Zur Erstellung der window-Sequenzen können bis zu 9 beliebig benannte Variablen definiert werden, die an bestimmte Items in der Sequenz gebunden werden. Deren Eigenschaften sind:
  - Window-variable: Variable, an die die window-Sequenzen gebunden werden
  - Start-item, Start-item-position, Start-previous-item, Start-next-item
  - End-item, End-item-position, End-previous-item, End-next-item
- Zwei window-Arten
  - sliding window: Werte der Ursprungs-Sequenz können überlappen
  - tumbling window: Werte der Ursprungs-Sequenz können nicht überlappen

# XQuery 3.0

## Erweiterung - FLWOR ['flower'] – sliding window



### ■ sliding window

- Die Fenster können überlappen

```
for $w in (1, 2, 3, 4)
```

```
 start at $s when true()
```

```
 only end at $e when $e - $s eq 2
```

```
return $w
```

→ 1, 2, 3, 2, 3, 4

Überlappung der Werte aus der Anfangs-Sequenz

Ende darf vom Start nur 2 Elemente entfernt sein



# XQuery 3.0

## Erweiterung - FLWOR ['flower'] – tumbling window

### tumbling window

- Die Fenster können nicht überlappen

```
for tumbling window $w in (1, 2, 3, 4, 5, 6, 7)
```

```
 start at $s when fn:true()
```

```
 only end at $e when $e - $s eq 2
```

```
return <window>{ $w }</window>
```

```
→ <window>1,2,3</window><window>4,5,6</window>
```

kommt nicht  
im Ergebnis vor

Ende darf vom Start

nur 2 Elemente  
entfernt sein

Keine Überlappung der Werte aus der  
Anfangs-Sequenz

ohne only-Schlüsselwort:

```
<window>1,2,3</window><window>4,5,6</window><window>7</window>
```



# XQuery 3.0

## Erweiterung - FLWOR ['flower'] – window - only

- Schlüsselwort „only“ liefert nur „komplette“ Fenster

```
for $w in (2, 4, 6, 8, 10, 12, 14)
 start at $s when fn:true()
 only end at $e when $e - $s eq 2
return <window>{ $w }</window>
```

→

```
<window>2 4 6</window>
<window>4 6 8</window>
<window>6 8 10</window>
<window>8 10 12</window>
<window>10 12 14</window>
```

```
for $w in (2, 4, 6, 8, 10, 12, 14)
 start at $s when fn:true()
 end at $e when $e - $s eq 2
return <window>{ $w }</window>
```

→

```
<window>2 4 6</window>
<window>4 6 8</window>
<window>6 8 10</window>
<window>8 10 12</window>
<window>10 12 14</window>
<window>12 14</window>
<window>14</window>
```

# XQuery 3.0

## Erweiterung - FLWOR ['flower'] – count



- Zähl-Variable innerhalb der FLWOR-Ausdrücke – count
  - Variable, die für jedes Tupel an die ordinale Position des Tupels im Stream gebunden ist.

```
xquery version "3.1";

<products>{
 for $product in doc("products_lite.xml")/*/product
 order by $product/name
 count $number
 return
 <product number="{$number}">{data($product/name)}</product>
</products>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<products>
 <product>
 <name>iPhone 7</name>
 </product>
 <product>
 <name>iPhone 7 Plus</name>
 </product>
 <product>
 <name>iPhone 6</name>
 </product>
 <product>
 <name>iPhone 5</name>
 </product>
</products>
```



```
<products>
 <product number="1">iPhone 5</product>
 <product number="2">iPhone 6</product>
 <product number="3">iPhone 7</product>
 <product number="4">iPhone 7 Plus</product>
</products>
```

# XQuery 3.0

## try/catch - Fehlerbehandlung

- Abfrage auf spezifische Fehler oder allgemeine Fehler

- Zugriff auf Fehler-Beschreibung und Fehler-Code

```
try {
 1 + '2'
} catch err:XPTY0004 {
 'Specific error: ' || $err:description
} catch * {
 'Error ['] || $err:code || ']: ' || $err:description
}
```

# XQuery 3.0

## switch - Anweisung

- Rückgabewert hängt von Auswertung (switch) ab
- Default-Zweig notwendig

```
let $sample-xml :=
 parse-xml("<Tests><Test lva= 'ADE' Note='1' /></Tests>"),
 $gradeInADE := sum($sample-xml/Tests/Test[@lva='ADE']/@Note)
return
switch ($gradeInADE)
 case (1) return "Grade of " || $gradeInADE || " -> Great!"
 case (2) return $gradeInADE || "? That's ok."
 case (3) return $gradeInADE || "? Almost OK."
 case (4) return $gradeInADE || "? Somewhat bad"
 case (5) return "Better luck next time!"
default return "No grade!"
```

# Inhalt

- Einführung ✓
- XDM – XQuery and XPath Data Model ✓
- Funktionen höherer Ordnung ✓
- XQuery and XPath Functions and Operators ✓
- JSON-Verarbeitung ✓
- XPath 3.0 / 3.1 ✓
- XQuery 3.0 / 3.1 ✓
- **XSLT 3.0**

# XSLT 3.0

## XSLT Stylesheets Neuerungen im Überblick

### XSL Transformations (XSLT) Version 3.0

W3C Recommendation 8 June 2017



**This version:**

<https://www.w3.org/TR/2017/REC-xslt-30-20170608/>

**Latest version:**

<https://www.w3.org/TR/xslt-30/>

**Previous versions:**

<https://www.w3.org/TR/2017/PR-xslt-30-20170418/>

<https://www.w3.org/TR/2017/CR-xslt-30-20170207/>



**Michael Kay**

@michaelhkay



Done and dusted: ten years' work. [w3.org/TR/xslt-30/](https://w3.org/TR/xslt-30/)

3:01 PM - Jun 8, 2017



155



128

people are talking about this

# XSLT 3.0 Erweiterungen

- Hauptfeature: Unterstützung zur Transformation von Streams
  - In diesem Modus müssen weder Quell- noch Ergebnis-Dokument immer komplett im Speicher gehalten werden
    - ◆ <xsl:source-document streamable="yes" href="...">
- Zusätzliche Erweiterungen
  - Unterstützung vom Datentyp `map` (ursprünglich für XSLT entwickelt, wurde er dann in XPath 3.1 übernommen)
  - Unterstützung vom Datentyp `array` (wenn XPath 3.1 für den XSLT-Prozessor implementiert wurde)
  - Funktionen für Import, Export und Verarbeitung von JSON
  - Packages zur weiteren Modularisierung, Wiederverwendung und Erweiterung von Code
    - ◆ Stylesheets als Module strukturieren und über `<xsl:include>` / `<xsl:import>` einbinden
    - ◆ Package als Sammlung von Modulen mit einer definierten Schnittstelle (`public`, `private`, `abstract`, `final`)
    - ◆ Packages können separat kompiliert werden
  - Funktionen höherer Ordnung (XPath 3.0)
  - Verbesserung der Fehlerbehandlung (`try/catch`)

# XSLT 3.0

## Streaming

- Grundidee: Verarbeitung von Dokumenten (Quell- und Resultat-Dokumente) „on the fly“
  - Die Quell- und Ergebnis-Dokumente müssen nicht mehr durch komplette Bäume im Speicher repräsentiert werden
    - ◆ Ressourcen sparer
  - Dokument wird als Event-Sequenz verarbeitet
  - Problem: Streaming ist nicht für alle XSLT-Funktionen möglich
    - ◆ XPath-Ausdrücke könnten grundsätzlich das gesamte Dokument erfordern
    - ◆ Lösung: Definition eines Teils von XSLT, der im Streaming-Modus garantiert verwendet werden kann – siehe „Streamability“  
[<https://www.w3.org/TR/xslt-30/#streamability>]
- Neue Sprachkonstrukte (Auswahl):
  - ◆ `xsl:iterate`: Einfaches Streaming
  - ◆ `xsl:accumulator`: Werte beim Streaming von Dokumenten lesen, akkumulieren, zusammenfassen, längerfristig speichern
  - ◆ `xsl:fork / xsl:merge`: Manipulation vom Datenfluss (Aufteilen und Zusammenführen von Streams)

# XSLT 3.0

## Software (Auswahl)

Prozessor / Software	Plattform	Version	Link
Saxon	Java / .NET	10.3	<a href="http://saxon.sourceforge.net">http://saxon.sourceforge.net</a> <a href="http://www.saxonica.com">http://www.saxonica.com</a>
Altova RaptorXML Server	Windows	2021	<a href="https://www.altova.com/raptorml.html">https://www.altova.com/raptorml.html</a>
Altova XMLSpy	Windows	2021	<a href="https://www.altova.com/xmlspy.html">https://www.altova.com/xmlspy.html</a>
oXygen XML Editor – Streaming über Saxon-EE	Multi	23.1	<a href="https://www.oxygenxml.com/">https://www.oxygenxml.com/</a>
Stylus Studio	Windows	X16	<a href="http://www.stylusstudio.com">http://www.stylusstudio.com</a>

- Achtung: Unterschiedlicher Umfang bei der Implementierung der XSLT-3.0-Empfehlung

# Inhalt

- Einführung ✓
- XDM – XQuery and XPath Data Model ✓
- Funktionen höherer Ordnung ✓
- XQuery and XPath Functions and Operators ✓
- JSON-Verarbeitung ✓
- XPath 3.0 / 3.1 ✓
- XQuery 3.0 / 3.1 ✓
- XSLT 3.0 ✓

# **W3C Standards**

**XPath 3.0+**

**XQuery 3.0+**

**XSLT 3.0**

# Wichtige W3C Standards

## XDM und FO

- XQuery and XPath Data Model
  - XQuery and XPath Data Model 3.0
    - ◆ <https://www.w3.org/TR/xpath-datamodel-30/>
    - ◆ W3C Recommendation (April 2014)
  - XQuery and XPath Data Model 3.1
    - ◆ <https://www.w3.org/TR/xpath-datamodel-31/>
    - ◆ W3C Recommendation (März 2017)
- XPath and XQuery Functions and Operators -  
Funktions- und Operatoren-Bibliothek für XPath,  
XQuery und XSLT
  - XPath and XQuery Functions and Operators 3.0
    - ◆ <https://www.w3.org/TR/xpath-functions-30/>
    - ◆ W3C Recommendation (April 2014)
  - XPath and XQuery Functions and Operators 3.1
    - ◆ <https://www.w3.org/TR/xpath-functions-31/>
    - ◆ W3C Recommendation (März 2017)

# XPath 3.0 / 3.1

## W3C Standards

- XPath

- Version 3.0 W3C Recommendation (2014)
  - ◆ <https://www.w3.org/TR/xpath-30/>
- Version 3.1 W3C Recommendation (2017)
  - ◆ <https://www.w3.org/TR/xpath-31/>

# XQuery 3.0 / 3.1

W3C Standards

- XQuery
  - Version 3.0 W3C Recommendation (2014)
    - ◆ <https://www.w3.org/TR/xquery-30/>
  - Version 3.1 W3C Recommendation (2017)
    - ◆ <https://www.w3.org/TR/xquery-31/>
- Achtung: Viele XQuery-Prozessoren implementieren vor allem den XQuery-3.1-Standard noch nicht komplett

## XSLT 3.0

### W3C Standards

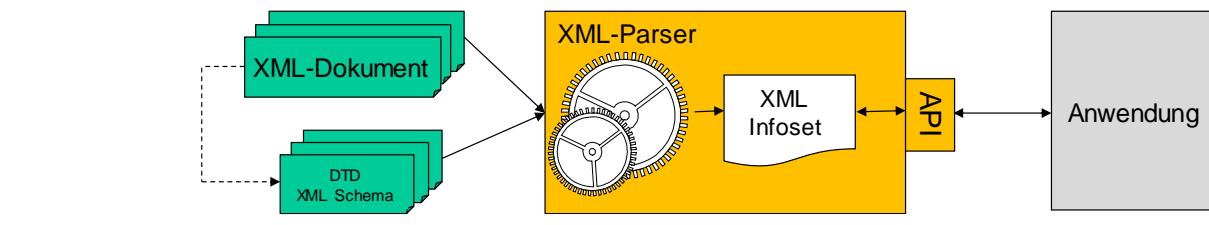
- XSL Transformations (XSLT) Version 3.0
  - <https://www.w3.org/TR/xslt-30/>
  - W3C Recommendation (Feb. 2017)

## Modul 9

# XML-Verarbeitung

## Programmierschnittstellen

Julian Haslinger



# Inhalt

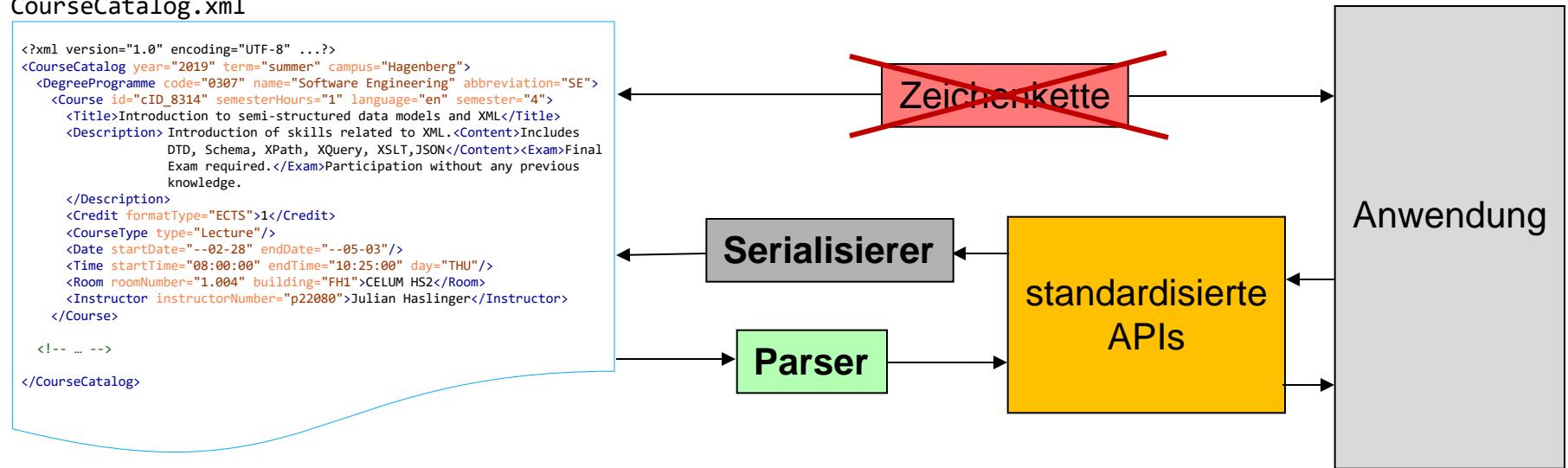
- **Einführung**
- Kategorien von XML-Parser
- Aufbau XML-Parser
- SAX – Simple API for XML
- DOM – Document Object Model
- StAX – Streaming API for XML
- Schema-Übersetzer – XML Data Binding
- Zusammenfassung

# Einführung 1/3

## Architektur

CourseCatalog.xml

```
<?xml version="1.0" encoding="UTF-8" ...?>
<CourseCatalog year="2019" term="summer" campus="Hagenerberg">
 <DegreeProgramme code="0307" name="Software Engineering" abbreviation="SE">
 <Course id="cID_8314" semesterHours="1" language="en" semester="4">
 <Title>Introduction to semi-structured data models and XML</Title>
 <Description>Introduction of skills related to XML. <Content>Includes DTD, Schema, XPath, XQuery, XSLT, JSON</Content><Exam>Final Exam required.</Exam>Participation without any previous knowledge.</Description>
 <Credit formatType="ECTS">1</Credit>
 <CourseType type="Lecture"/>
 <Date startDate="--02-28" endDate="--05-03"/>
 <Time startTime="08:00:00" endTime="10:25:00" day="THU"/>
 <Room roomNumber="1.004" building="FH1">CELUM HS2</Room>
 <Instructor instructorNumber="p22089">Julian Haslinger</Instructor>
 </Course>
 <!-- ... -->
</CourseCatalog>
```



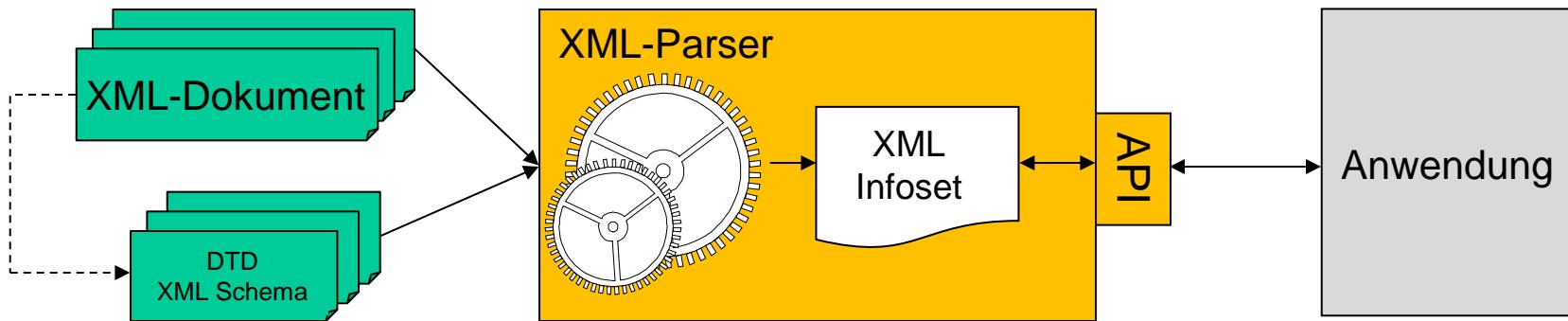
- **Parser**
  - analysiert XML-Dokument und liefert Tags, Text-Inhalte und Attribut-Wert-Paare
- **Serialisierer**
  - generiert aus Datenstrukturen XML-Dokument

# Einführung 2/3

## XML-Parser

### ■ XML-Parser ...

- befindet sich zwischen der Anwendung und dem XML-Dokument
- liest ein XML-Dokument und evtl. DTD/Schema zur Bestimmung des Dokumentinhalts ("parst" das Dokument)
- prüft auf Wohlgeformtheit und optional auf Gültigkeit
- stellt der Anwendung den Dokumentinhalt über eine abstrakte Programmierschnittstelle (API) zur Verfügung



# Einführung 3/3

## XML-Parser

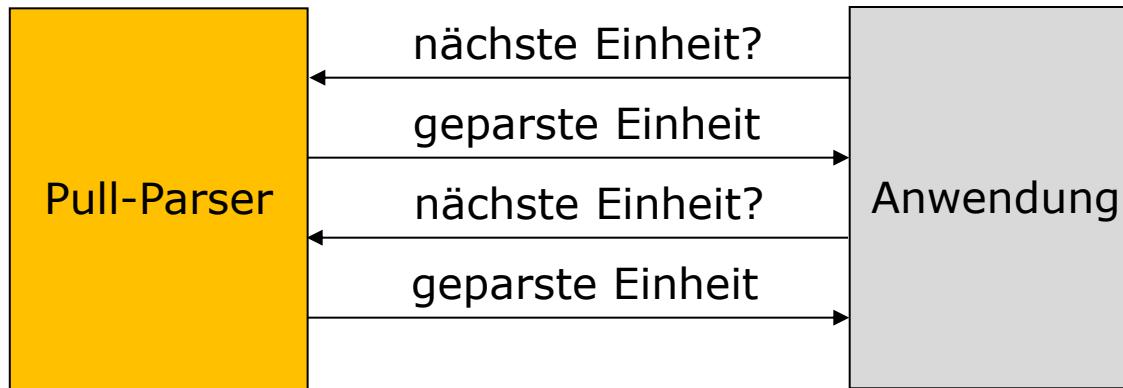
- XML-Parser bereitet das XML-Dokument auf
  - bestimmt Vorgabewerte (Default-Werte) für Attribute gemäß DTD/XML Schema
  - löst Referenzen auf externe Entities (in #PCDATA) und interne Entities (in Attributwerten und #PCDATA) auf – das kann lokal oder über ein Netzwerk erfolgen
  - normalisiert Attributwerte (z.B. CR, LF, Tab, Space)
  - unterstützt verschiedene Zeichensätze
  - ...
- Conclusio
  - Benutzung eines Parsers spart Arbeit!

# Kategorien von Parser 1/4

- Validierender vs. nicht-validierender Parser
  - Wird die Validität des Dokumentes untersucht?
    - DTD oder XML Schema erforderlich
- Pull- vs. Push-Parser
  - Wer hat die Kontrolle über das Parsen?
    - Anwendung oder Parser
- Einschritt- vs. Mehrschritt-Parser
  - Wird das XML-Dokument in einem Schritt vollständig geparsst oder Schritt für Schritt?
- Beachte: Kategorien unabhängig voneinander, können kombiniert werden.

# Kategorien von Parser 2/4

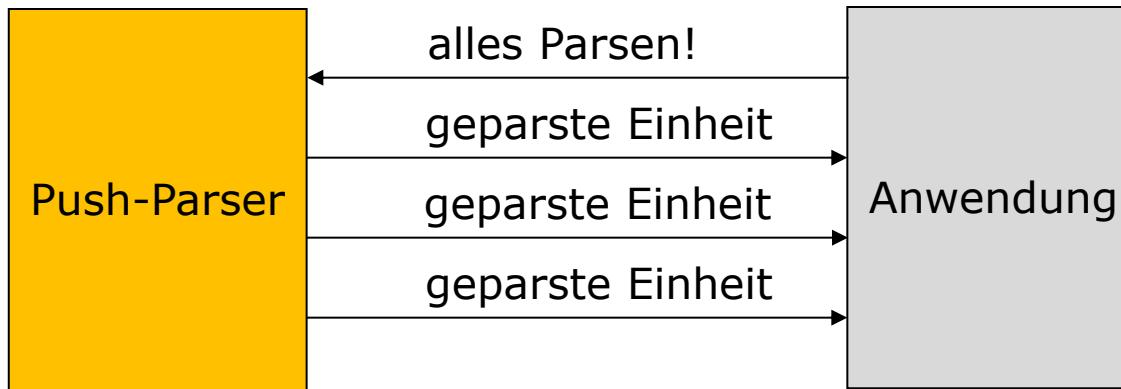
## Pull-Parser



- Anwendung hat Kontrolle über den Parse-Vorgang.
- Analyse der nächsten syntaktischen Einheit muss aktiv angefordert werden.
- Beachte: "Pull" aus Perspektive der Anwendung.

# Kategorien von Parser 3/4

## Push-Parser



- Parser hat Kontrolle über den Parse-Vorgang.
- Sobald Parser eine syntaktische Einheit analysiert hat, übergibt Parser das Analyseergebnis.
- Beachte: "Push" aus Perspektive der Anwendung.

# Kategorien von Parser 4/4

## XML-Parser

	Einschritt	Mehrschritt
Pull	<b>DOM</b>	<b>StAX</b>
Push		<b>SAX</b>

## Implementierungen:

### JAXP in J2SE



JAXP: Java API for XML Processing

- von Oracle (ehemals Sun Microsystems)

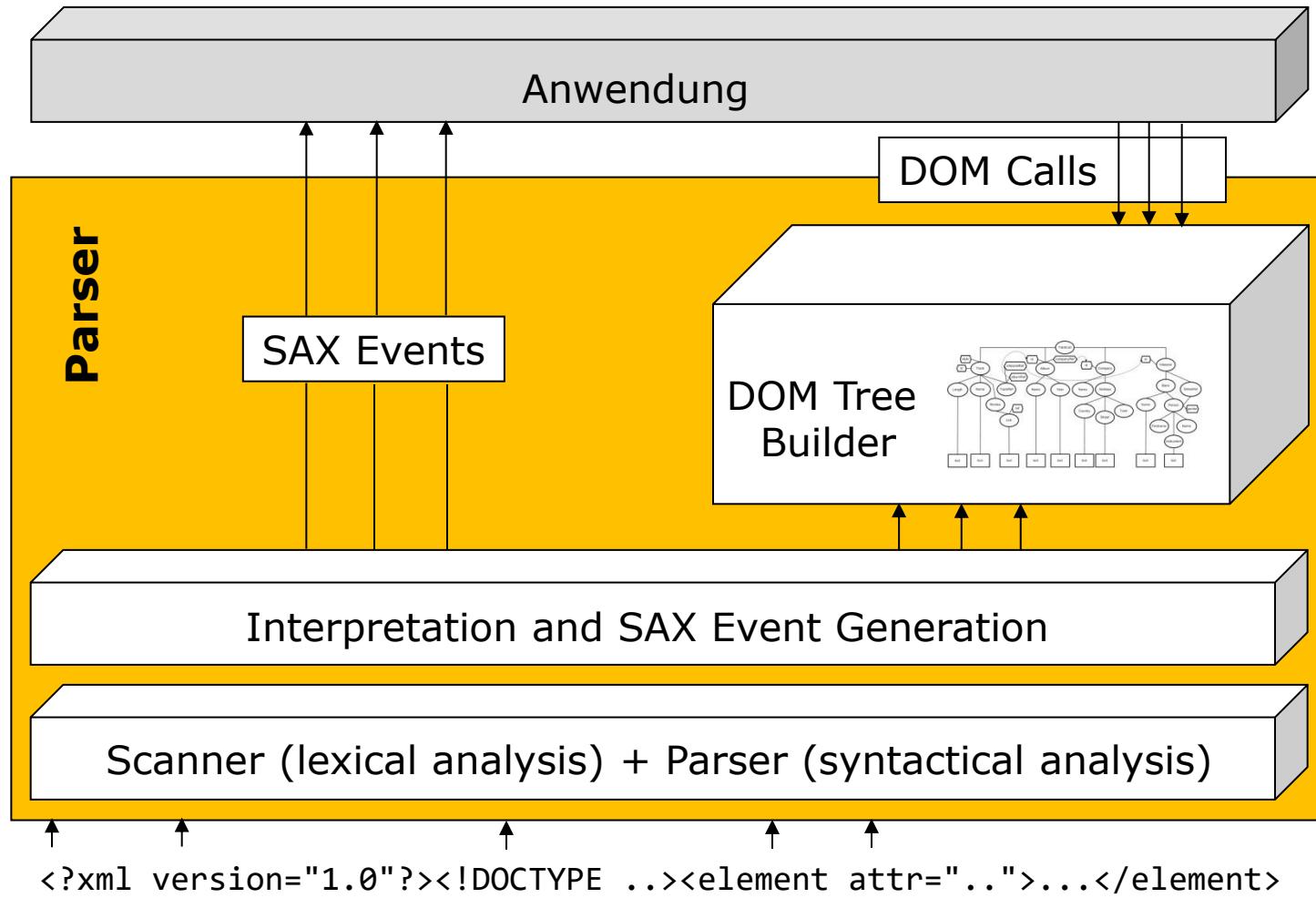
### .NET Framework



- von Microsoft

- **DOM**: Document Object Model
- **SAX**: Simple API for XML
- **StAX**: Streaming API for XML

# Architektur DOM/SAX-Parser



# Inhalt

- Einführung
- Kategorien von XML-Parser
- Aufbau XML-Parser
- **SAX – Simple API for XML**
  - Einführung
  - Grundlagen
  - Architektur
  - Anwendung
  - Zusammenfassung
- DOM – Document Object Model
- StAX – Streaming API for XML
- Schema-Übersetzer – XML Data Binding
- Zusammenfassung

# Einführung

## SAX – Simple API for XML Parsing

- Mehrschritt-Push-Parser
- Ursprünglich nur Java-API
  - inzwischen werden nahezu alle gängigen Sprachen unterstützt:  
z.B.: C, C++, C#, Python, Perl, JavaScript
  - auch in Microsoft XML Core Services (MSXML) und  
in .NET (.NET-XML-Klassen) integriert
  - Kein W3C-Standard, sondern De-facto-Standard (Public Domain)
- keine "formale" Standardisierung (wie z.B. von W3C)
  - entwickelt von der XML Community  
unter der Leitung von David Megginson  
⇒ [www.saxproject.org](http://www.saxproject.org)
  - aktuell: SAX 2.0.2 (April 2004)

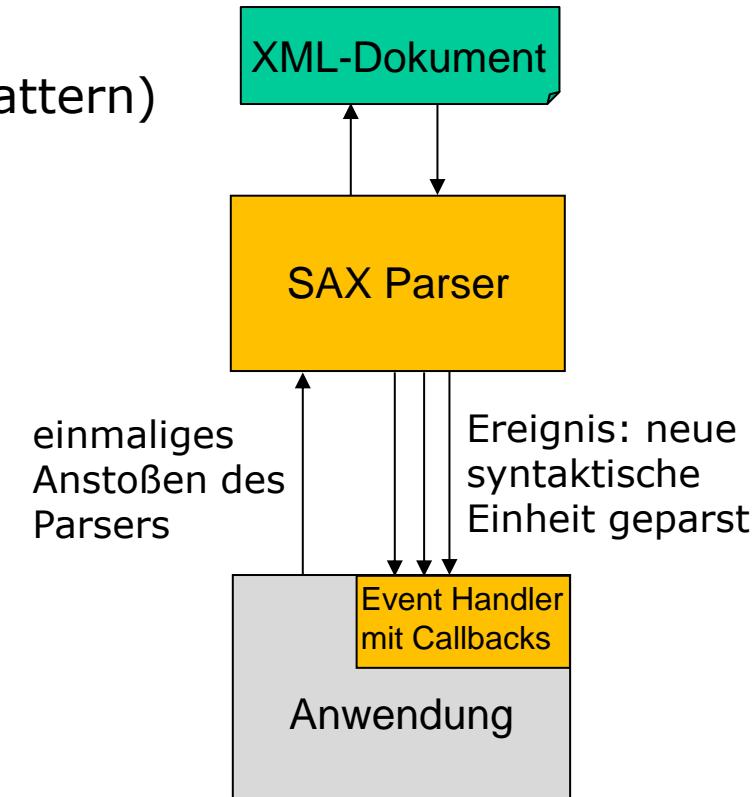
SOURCEFORGE.NET®

# Grundlagen

## Ereignisbasiertes Parsen

### SAX realisiert ereignisgesteuertes Vorgehen

- Parser definiert API-Signaturen für Callback-Methoden (vgl. Observer Pattern)
- Anwendung implementiert Callback-Methoden und registriert Callbacks mittels Event Handler beim Parser
- Parser liest XML-Dokument; initiiert von Anwendung
- Parser liefert ein Ereignis je Start-Tag, End-Tag etc. in Form von Callback-Methodenaufrufen
- Ausschließlich sequentielle Abarbeitung eines XML-Dokuments



# Grundlagen

## Beispiel

CourseCatalog.xml

```

...
<CourseCatalog year="2019" term="summer" campus="Hagenberg">
 <DegreeProgramme code="0307" name="Software Engineering">
 <Course id="cID_8314" semesterHours="1" semester="4">
 <Title>
 Introduction to ...
 </Title>
 <Credit formatType="ECTS">
 1
 </Credit>
 ...
 </Course>
 ...
</DegreeProgramme>
...
</CourseCatalog>
```

→ Parser ruft **startDocument()** auf.

→ Parser ruft **startElement("CourseCatalog", AttributeList( length=3, {name='year', type='NMTOKEN', value='2019'} ... )** auf.

→ Parser ruft **startElement("DegreeProgramme", AttributeList( length=2, {name='code', type='CDATA', value='0307'} ... )** auf.

→ Parser ruft **startElement("Course", AttributeList( length=3, {name='id', type='ID', value='cID\_8314'} ... )** auf.

→ Parser ruft **startElement("Title", null)** auf.

→ Parser ruft **characters("Introduction to...", ...)** auf.

→ Parser ruft **endElement("Title")** auf.

→ Parser ruft **startElement("Credit", AttributeList( length=1, {name='formatType', type='NMTOKEN', value='ECTS'})** auf.

→ Parser ruft **characters("1", ...)** auf.

→ Parser ruft **endElement("Credit")** auf.

...

→ Parser ruft **endElement("Course")** auf.

...

→ Parser ruft **endElement("DegreeProgramme")** auf.

...

→ Parser ruft **endElement("CourseCatalog")** auf.

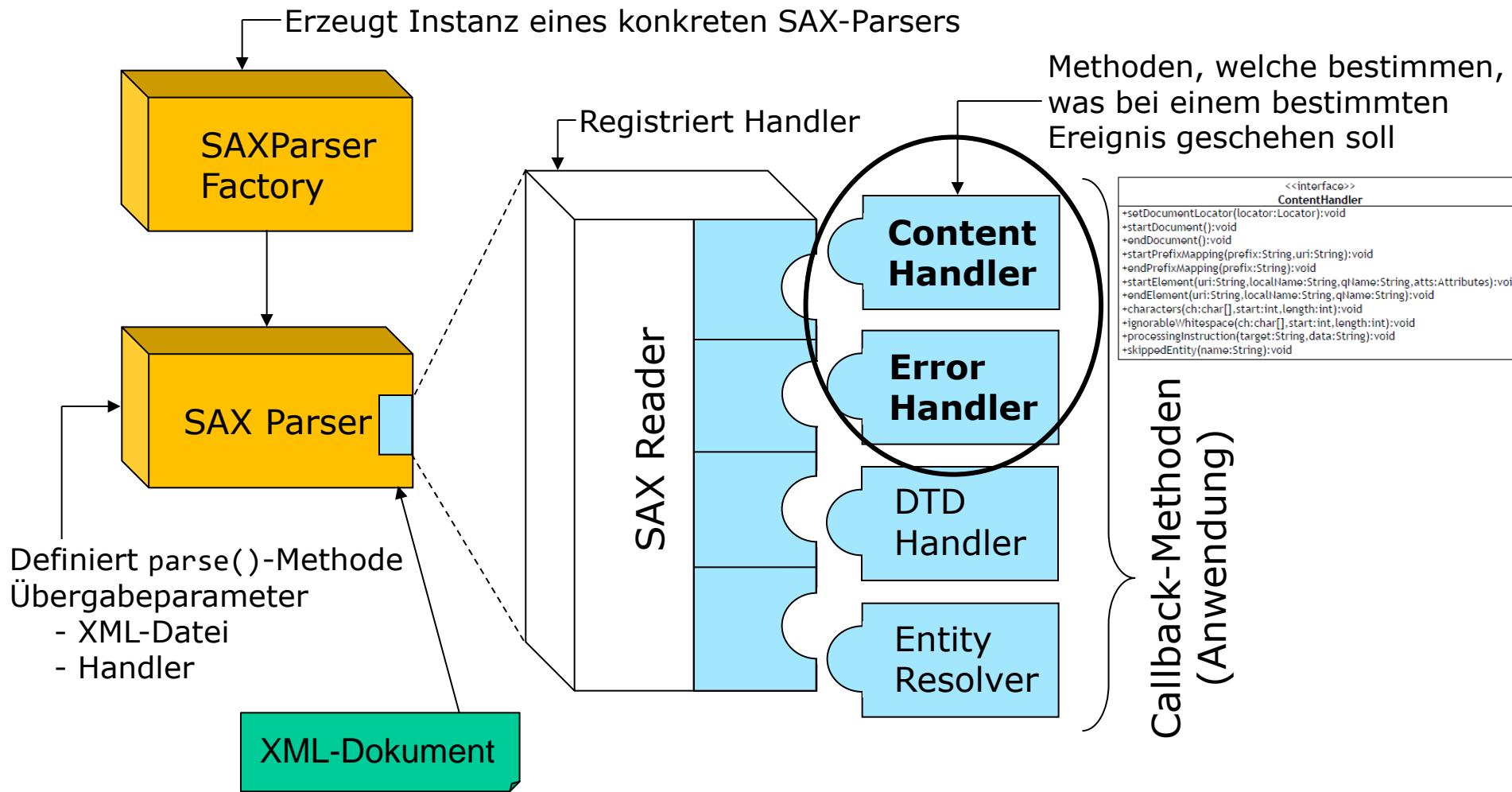
→ Parser ruft **endDocument()** auf.

- Ereignisfolge: Sobald Einheit geparst wurde, wird Anwendung benachrichtigt.
- Beachte: Es wird kein Parse-Baum aufgebaut!

# Architektur

Die **Abstrakte Fabrik** ist ein **Erzeugungsmuster**. Es definiert eine Schnittstelle zur Erzeugung einer Familie von Objekten, wobei die konkreten Klassen der zu instanzierenden Objekte dabei nicht näher festgelegt werden. Eine Abstrakte Fabrik vereinigt die Verantwortungen

- Zusammenfassung der Objektgenerierung an einer Stelle und
- Möglichkeit zu abstrakten Konstruktoren



# Callback-Methoden (Auszug)

## Content-Handler 1/5

- Methoden des Handlers (also der Anwendung), die vom Parser aufgerufen werden
- für jede syntaktische Einheit eigene Callback-Methode, u.a.:
  - `startDocument` und `endDocument`
  - `startElement` und `endElement`
  - `characters`
  - `processingInstruction`
- **DefaultHandler**
  - Standard-Implementierung der Callback-Methoden = leere Methoden!
  - Anwendung definiert Subklasse und überschreibt Callback-Methoden

# Callback-Methoden

## Content-Handler 2/5

- **void startDocument()**

**throws SAXException**

- Beginn des Dokuments. Methode wird als erste vor allen anderen Methoden aufgerufen

- **void endDocument()**

**throws SAXException**

- Ende des Dokuments. Methode wird nach dem Abschluss des Parse-Vorgangs aufgerufen.

# Callback-Methoden

## Content-Handler 3/5

■ **void startElement( String namespaceURI,  
String localName,  
String qName,  
Attributes attrs)**  
**throws SAXException**

- Methodenaufruf, wenn Parser mit der Elementverarbeitung beginnt.
- **namespaceURI**: Namensraum-Bezeichner oder null
- **localName**: lokaler Elementname ohne Präfix
- **qName**: Elementname mit Namensraumpräfix (falls vorhanden)
- **atts**: Liste aller Attribute des Elements (nur bei startElement)
  - Zugriff auf Attribute über Position (Index) oder Namen
  - Vorgabewerte (Defaultwerte) für Attribute werden durch Parser bereitgestellt (Achtung bei nicht validierenden Parsern!)

### Attributverarbeitung

```
public void startElement (String namespaceURI, String localName, String qName, Attributes attrs) throws SAXException) {
 for (int i = 0; i < attrs.getLength(); i++) {
 String name = attrs.getLocalName(i);
 String type = attrs.getType(i);
 String value = attrs.getValue(i); [...] }
}
```

# Callback-Methoden

## Content-Handler 4/5

■ **void endElement( String namespaceURI,  
String localName,  
String qName)  
throws SAXException**

- Methode wird beim Ende eines Elements aufgerufen.
- **namespaceURI**: Namensraum-Bezeichner oder null
- **localName**: lokaler Elementname ohne Präfix
- **qName**: Elementname mit Präfix

# Callback-Methoden

## Content-Handler 5/5

■ **void characters( char[] ch,  
                  int start,  
                  int length)**  
**throws SAXException**

- Lesen von Zeichenketten (Textknoten). Methode wird (einmal od. mehrmals) aufgerufen, wenn Parser eine Zeichenkette liest.
  - **ch**: Zeichenkette, die gesamten Inhalt des Vaterknotens enthält
  - **start**: Anfang der Zeichenkette
  - **length**: Länge der Zeichenkette
- Zeichenkette kann wie folgt erzeugt werden

```
StringBuffer text = new StringBuffer();
...
void characters(char[] ch, int start, int length) throws SAXException {
 text.append(ch, start, length);
}
```

# Beispiel

CourseCatalog.xml

```
<CourseCatalog year="2019" term="summer" campus="Hagenberg">
 <DegreeProgramme code="0307" name="Software Engineering" abbreviation="SE">
 <Course id="cID_8314" semesterHours="1" language="en" semester="4">
 <Title>Introduction to semi-structured data models and XML</Title>
 <Credit formatType="ECTS">1</Credit>
 <CourseType type="Lecture"/>
 <Room roomNumber="1.004" building="FH1">CELUM HS2</Room>
 <Instructor instructorNumber="p22080">Julian Haslinger</Instructor>
 </Course>
 <Course id="cID_8315" semesterHours="2" language="en" semester="4">
 <Title>Introduction to semi-structured data models and XML</Title>
 <Credit formatType="ECTS">1,5</Credit>
 <CourseType type="LabSession"/>
 <Room roomNumber="3.108" building="FH3">XORTEX LBS3</Room>
 <Instructor instructorNumber="p20623">Barbara Traxler</Instructor>
 </Course>
 </DegreeProgramme>
</CourseCatalog>
```



Gesucht ist die Ausgabe  
in folgender Form:

```

DegreeProgramme: Software Engineering

Course: cID_8314
Title: Introduction to semi-structured data models and XML
CourseType: Lecture
Room: CELUM HS2
Instructor: Julian Haslinger

Course: cID_8315
Title: Introduction to semi-structured data models and XML
CourseType: LabSession
Room: XORTEX LBS3
Instructor: Barbara Traxler

```

Dazu sind

- ein SAX-Parser und
  - passende Callback-Methoden
- notwendig!

# Erzeugen eines SAX Parser

## (Java API for XML Processing)

```
...
import org.xml.sax.*; ← Definiert SAX-Interface
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory; ← Definiert SAXParserFactory, welche
import javax.xml.parsers.SAXParser; ← einen SAXParser erzeugt.
...
try {
 //Get instance of SAXParserFactory
 SAXParserFactory myFactory = SAXParserFactory.newInstance();

 myFactory.setValidating(true); // Validating against DTD

 //Get instance of SAXParser
 SAXParser mySaxParser = myFactory.newSAXParser();
 //Create user-defined handler
 MyHandlerImpl myHandler = new MyHandlerImpl();
 //Start parsing
 mySaxParser.parse(new File("CourseCatalog.xml"), myHandler);
}
catch (... Exceptions) { ... }
}
```

Registrieren vom benutzerdef.  
Handler beim SAX-Parser

Zu parsende Datei, kann  
auch URL oder Stream sein

# Implementieren der Callback-Methoden

```
...

public class MyHandlerImpl extends DefaultHandler {
 private boolean printChars; // Context for printing

 public MyHandlerImpl() {
 super();
 }

 public void startDocument() throws SAXException {
 printChars = false;
 }
```

# Implementieren der Callback-Methoden

```
public void startElement (String namespaceURI, String localName,
 String qName, Attributes attrs) throws SAXException {

 if (qName.equals("DegreeProgramme")) {
 System.out.println("*****");
 System.out.println("DegreeProgramme: " + attrs.getValue("name"));
 System.out.println("*****");
 } else if (qName.equals("Course")) {
 System.out.println("Course: " + attrs.getValue("id"));
 } else if (qName.equals("Title")) {
 System.out.print("Title: ");
 printChars = true;
 } else if (qName.equals("CourseType")) {
 System.out.println("CourseType: " + attrs.getValue("type"));
 } else if (qName.equals("Room")) {
 System.out.print("Room: ");
 printChars = true;
 } else if (qName.equals("Instructor")) {
 System.out.print("Instructor: ");
 printChars = true;
 }
}
```

```

DegreeProgramme: Software Engineering

Course: cID_8314
Title: Introduction to semi-structured data models and XML
CourseType: Lecture
Room: CELUM HS2
Instructor: Julian Haslinger

Course: cID_8315
Title: Introduction to semi-structured data models and XML
CourseType: LabSession
Room: XORTEX LBS3
Instructor: Barbara Traxler

```

# Implementieren der Callback-Methoden

```
public void characters(char ch[], int start, int length)
 throws SAXException {
 if (printChars) {
 System.out.println(new String(ch, start, length));
 printChars = false;
 }
}

public void endElement (String namespaceURI, String localName,
 String qName) throws SAXException {

 if (qName.equals("Course")) {
 System.out.println("-----");
 }
}
```

```

DegreeProgramme: Software Engineering

Course: cID_8314
Title: Introduction to semi-structured data models and XML
CourseType: Lecture
Room: CELUM HS2
Instructor: Julian Haslinger

Course: cID_8315
Title: Introduction to semi-structured data models and XML
CourseType: LabSession
Room: XORTEX LBS3
Instructor: Barbara Traxler

```

# Fehlerbehandlung

- SAX Parser überprüft immer Wohlgeformtheit eines XML-Dokuments.
- kann auch die Gültigkeit bzgl. einer DTD oder eines Schemas überprüfen
  - ⇒ Syntax- und Strukturfehler werden vom SAX-Parser erkannt
  - ⇒ Callback-Methoden können wohlgeformte und gültige Dokumente voraussetzen.

# Callback-Methoden

## Error-Handler

- `void warning(SAXParseException e)`  
    `throws SAXException`
  - Fehler in der DTD. Bspw. unnötige (doppelte) Deklarationen für Entities, Attribute etc.
- `void error(SAXParseException e)`  
    `throws SAXException`
  - XML-Dokument ist nicht gültig.
- `void fatalError(SAXParseException e)`  
    `throws SAXException`
  - XML-Dokument ist nicht wohlgeformt, Parser bricht ab.
- SAXParseException enthält Zusatzinformation
  - `e.getLineNumber()` // Zeilenummer
  - `e.getColumnNumber()` // Spaltennummer
  - `e.getLocalizedMessage()` // Fehlermeldung

# Zusammenfassung 1/2

## ■ Vorteile

- Geeignet für große und einfach strukturierte XML-Dokumente
- Geeignet für Streaming-Anwendungen (d.h. Verarbeiten des Dokuments, bevor es vollständig übertragen ist)
- Geringe Speicheranforderungen, performant

## ■ Nachteile

- Zugriff ausschließlich sequentiell (forward-only)
- Anwendung muss Puffer-Datenstrukturen nachbilden, um Zugriff auf Kontext eines Ereignisses zu haben (SAX ist zustandslos!)
- Weniger geeignet für komplexe Bearbeitung (die Navigationsmöglichkeiten erfordern)
- Manipulation (Ändern und Schreiben) der XML-Dokumente nicht möglich – ausschließlich lesender Zugriff

# Zusammenfassung 2/2

- SAX ist geeignet, wenn
  - es genügt, ein XML-Dokument einmal zu durchlaufen
  - man nur einzelne Bestandteile eines XML-Dokuments benötigt
  - die Puffer-Datenstruktur "einfacher" als das XML-Dokument ist
  - der Kontext der benötigten Daten nicht von besonderem Interesse ist
  - Speicher-Restriktionen bestehen (bspw. im Kontext von mobilen Geräten)

# Inhalt

- Einführung
- Kategorien von XML-Parser
- Aufbau XML-Parser
- SAX – Simple API for XML
- **DOM – Document Object Model**
  - Einführung
  - Grundlagen
  - Architektur
  - Baumstruktur
  - Anwendung
  - Zusammenfassung
- StAX – Streaming API for XML
- Schema-Übersetzer – XML Data Binding
- Zusammenfassung

# Einführung

## DOM – Document Object Model

- Einschritt-Pull-Parser
- W3C-Standard ([www.w3c.org/DOM/](http://www.w3c.org/DOM/))
- Hauptspeicherrepräsentation (=DOM-Baum) des XML-Dokuments wird erstellt
- Verarbeitung von XML- als auch von HTML-Dokumenten
- Plattform- und programmiersprachenunabhängig



# Einführung

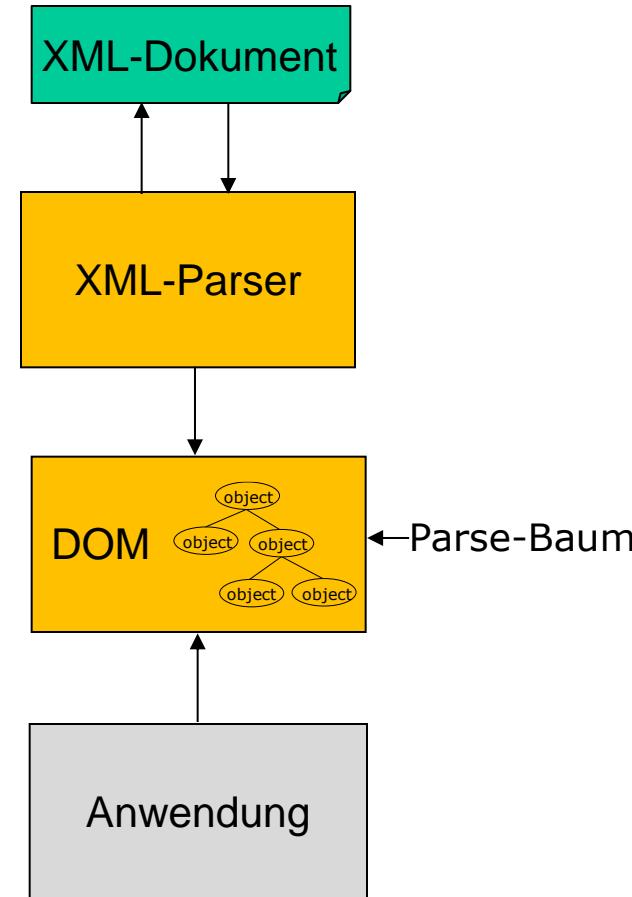
## Geschichtliche Entwicklung

- **Ursprüngliche Idee:** Web-Browsern den Zugriff und die Manipulation von Elementen auf einer Web-Seite zu ermöglichen ("DOM Level 0" – Entwicklung von Netscape)
- DOM wurde ständig erweitert und stellt heute ein mächtiges API bereit, ...
  - das die Knoten-Typen definiert, die in einem XML Dokument vorkommen und
  - das die Methoden und Eigenschaften für deren Zugriff und Manipulation festlegt
- DOM-Versionen in Form von Levels:
  - Levels bauen jeweils aufeinander auf
  - Jedem Level sind Module der DOM-Gesamtarchitektur zugeordnet
  - Aktuelle Version: DOM4 (November 2015)
    - <https://www.w3.org/DOM/DOMTR>
    - <https://www.w3.org/TR/domcore/>

# Grundlagen

## Charakteristika von DOM

- Plattform- und programmiersprachen-unabhängiges API
- Definiert die logische Struktur eines Dokuments in Form eines Baumes (DOM-Baum), bestehend aus Knoten
- Dokument muss vollständig geparsst werden, um den DOM-Baum aufzubauen
- Unterstützt das Lesen, sowie die Änderung von Inhalt und Struktur eines XML-Dokuments
- API definiert Klassen für Knoten des DOM-Baums



# Architektur 1/2

## Module

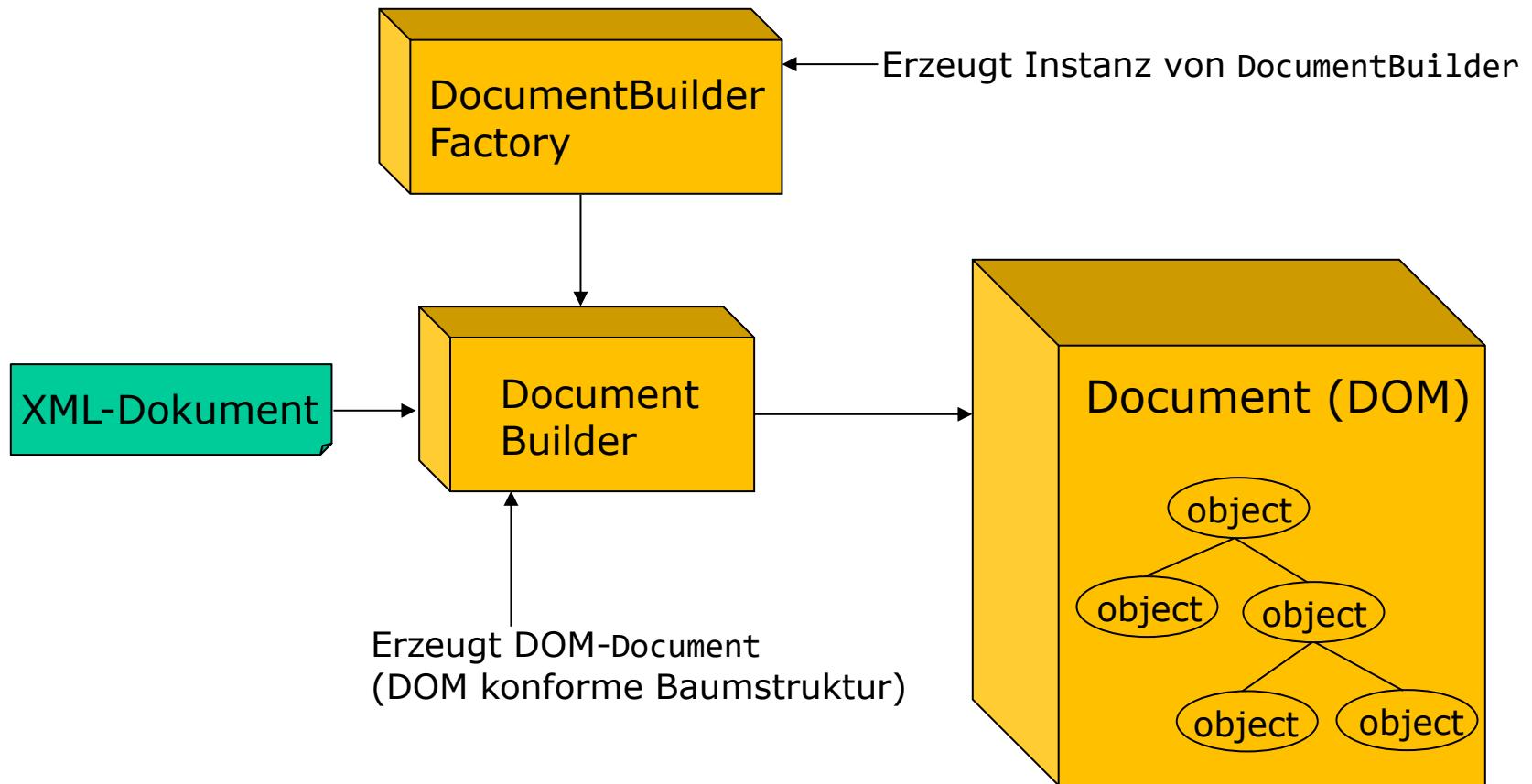
### ■ Untergliederung der DOM-Architektur in Module

⇒ Implementierungen können aber auch nur Teile von DOM unterstützen

- **DOM Core** (Baumdarstellung eines Dokuments)
- **DOM XML** (Erweiterung des DOM Core um XML-spezifische Merkmale wie Processing Instructions, CDATA, Entities, ...)
- **DOM HTML** (Verarbeitung von HTML Dokumenten)
- **DOM Events** (Maus, Tastatur, ...)
- **DOM Cascading Style Sheets** (Formatierung von Dokumenten)
- **DOM Load and Save** (Laden und Speichern von Dokumenten)
- **DOM Validation** (Modifikation des DOM-Baums unter Beibehaltung der Validität)
- **DOM Path** (XPath-Anfragen an einen DOM-Baum)

# Architektur 2/2

## Anwendung eines DOM-Parsers



# Baumstruktur 1/5

## DOM-Knoten

- XML-Dokument wird als Baumstruktur dargestellt
- Komponenten werden als Knoten repräsentiert

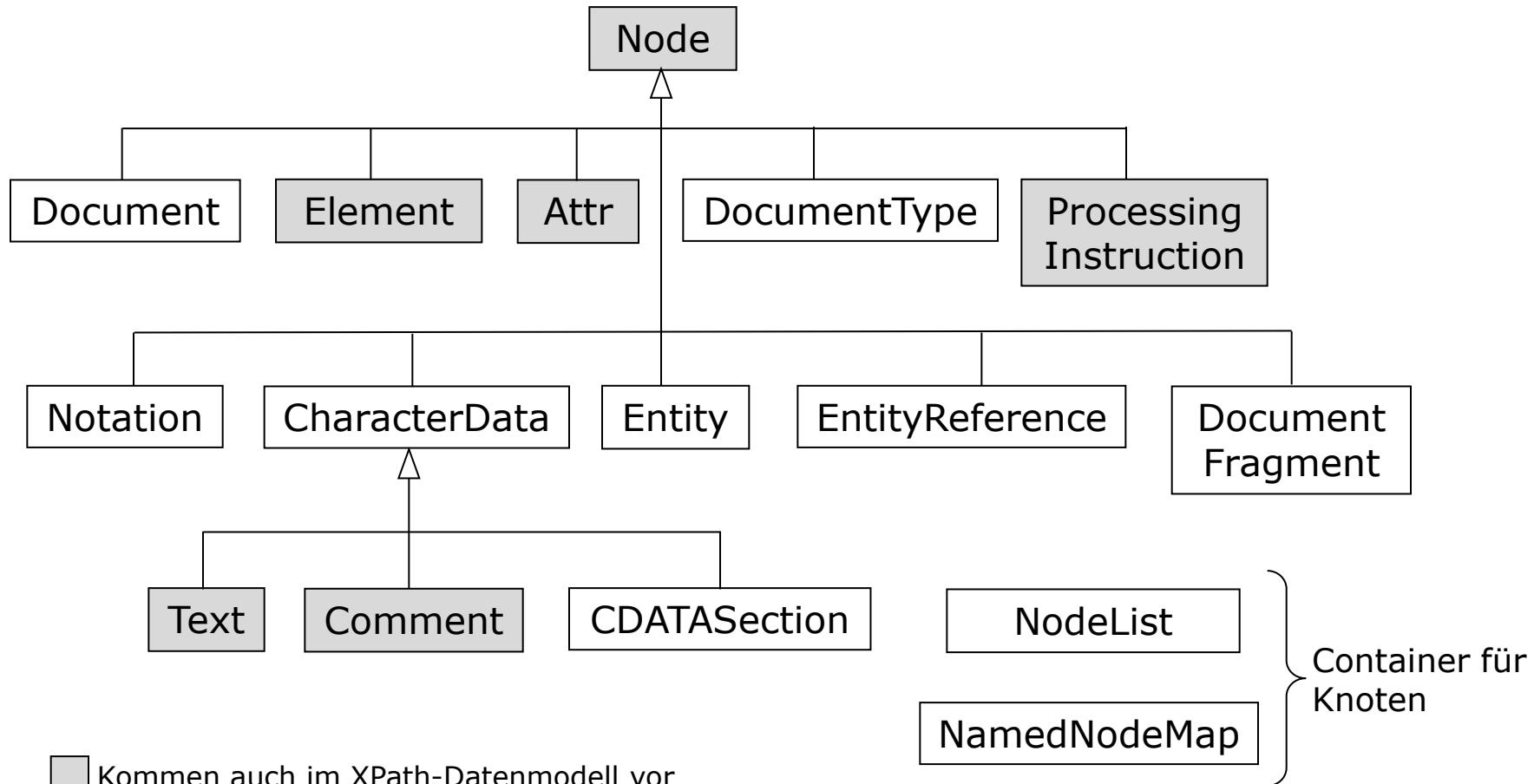
XML Komponente	DOM Repräsentation
Tag z.B.: < <b>Course</b> >	Elementknoten
Attribut z.B.: <... <b>id="cID_8314"</b> >	Attributknoten
Text z.B.: ...> <b>CELUM HS2</b> <...	Textknoten

Eine Liste sämtlicher Transformationsregeln (über 40) ist zu finden unter:  
<http://www.w3.org/TR/DOM-Level-3-Core/core.html>

# Baumstruktur 2/5

DOM-Knotentypen ([www.w3c.org](http://www.w3c.org))

- DOM basiert auf einem generischen Objektmodell



# Baumstruktur 3/5

## Node Interface (Interface Description Language)

```

interface Node {
 // NodeType
 const unsigned short ELEMENT_NODE = 1;
 const unsigned short ATTRIBUTE_NODE = 2;
 const unsigned short TEXT_NODE = 3;
 const unsigned short CDATA_SECTION_NODE = 4;
 const unsigned short ENTITY_REFERENCE_NODE = 5;
 const unsigned short ENTITY_NODE = 6;
 const unsigned short PROCESSING_INSTRUCTION_NODE = 7;
 const unsigned short COMMENT_NODE = 8;
 const unsigned short DOCUMENT_NODE = 9;
 const unsigned short DOCUMENT_TYPE_NODE = 10;
 const unsigned short DOCUMENT_FRAGMENT_NODE = 11;
 const unsigned short NOTATION_NODE = 12;

 readonly attribute DOMString nodeName; // The name of this node.
 attribute DOMStringnodeValue; // The value of this node.
 ...
 readonly attribute unsigned short nodeType; // A code representing the type of the underlying object.
 readonly attribute Node parentNode; // The parent of this node.
 readonly attribute NodeList childNodes; // A NodeList that contains all children of this node.
 readonly attribute Node firstChild; // The first child of this node.
 readonly attribute Node lastChild; // The last child of this node.
 readonly attribute Node previousSibling; // The node immediately preceding this node.
 readonly attribute Node nextSibling; // The node immediately following this node.
 readonly attribute NamedNodeMap attributes; // A NamedNodeMap containing attributes of this node
 // (if it is an Element) or null otherwise.

 // Modified in DOM Level 2:
 readonly attribute Document ownerDocument; // The Document object associated with this node.

 // Modified in DOM Level 3:
 ...
}

```

# Baumstruktur 4/5

## Node Interface (Interface Description Language)

```
...
Node insertBefore(in Node newChild, in Node refChild) raises(DOMException);
// Modified in DOM Level 3:
Node replaceChild(in Node newChild, in Node oldChild) raises(DOMException);
// Modified in DOM Level 3:
Node removeChild(in Node oldChild) raises(DOMException);
// Modified in DOM Level 3:
Node appendChild(in Node newChild) raises(DOMException);
boolean hasChildNodes();
Node cloneNode(in boolean deep);
// Modified in DOM Level 3:
void normalize();
// Introduced in DOM Level 2:
boolean isSupported(in DOMString feature,
 in DOMString version);
// Introduced in DOM Level 2:
readonly attribute DOMString namespaceURI;
// Introduced in DOM Level 2:
...

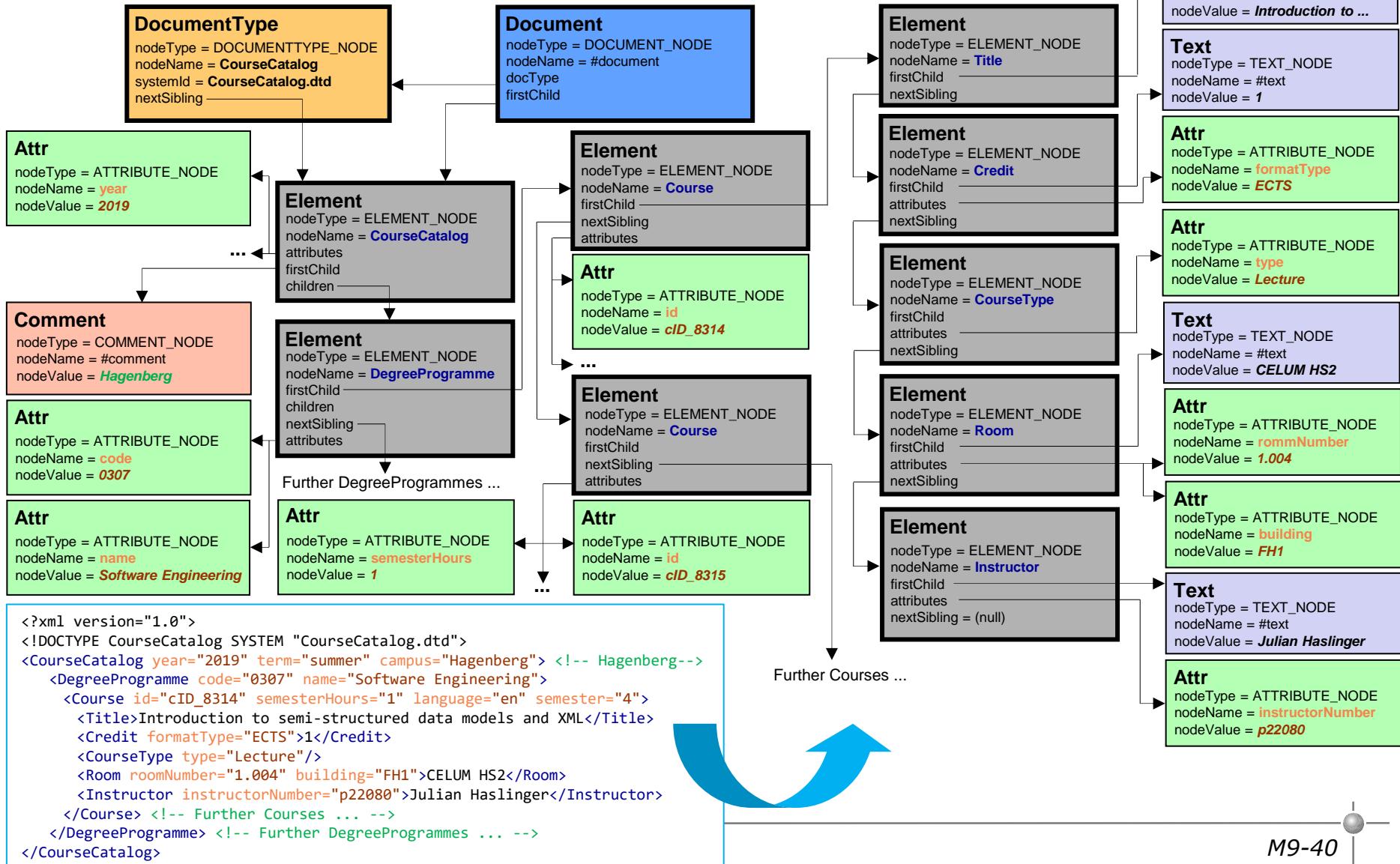
// Introduced in DOM Level 2:
boolean hasAttributes();
// Introduced in DOM Level 3:
readonly attribute DOMString baseURI;
...

};

}
```

# Baumstruktur 5/5

## XML-Dokument als DOM-Baum (nicht vollständig!)

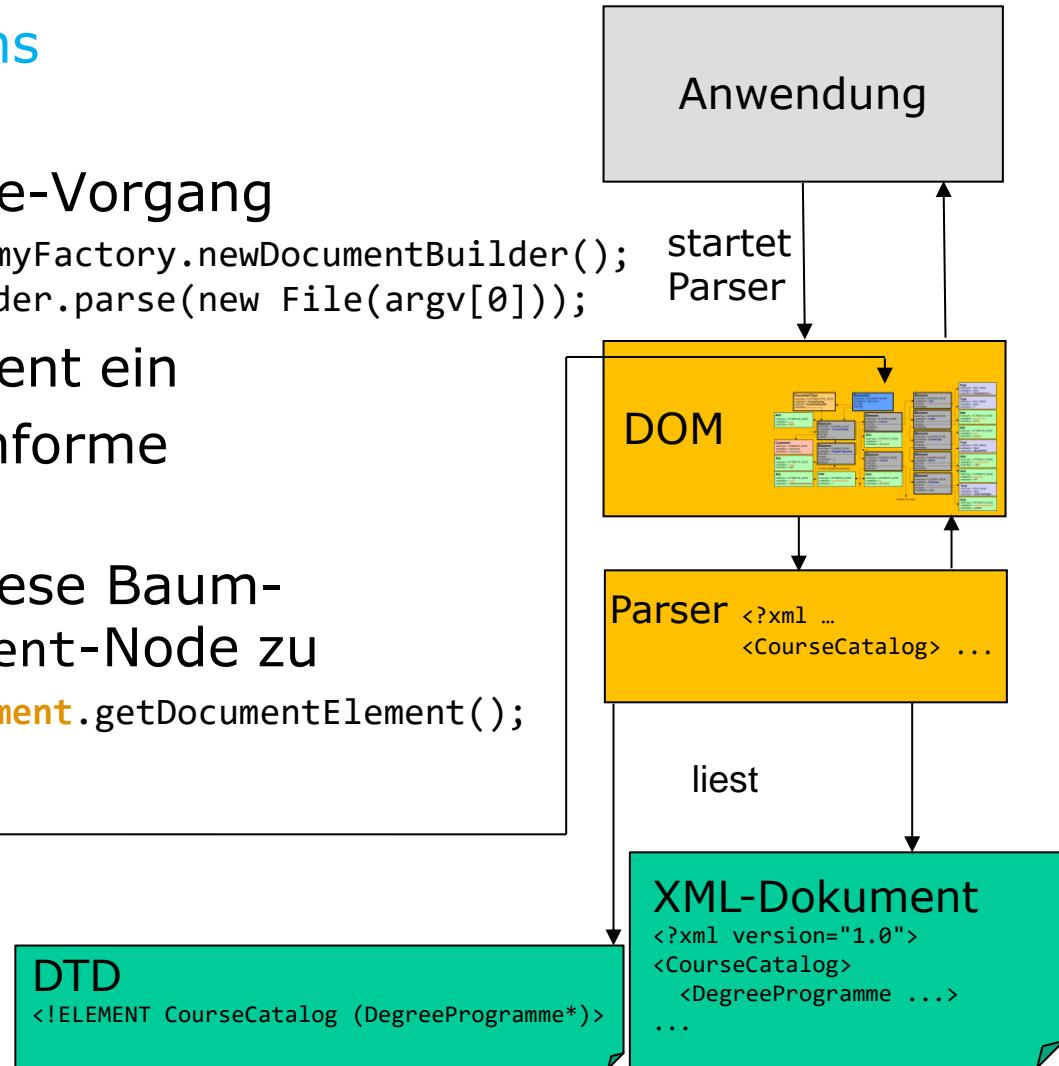


# Anwendung 1/12

## Erzeugen eines DOM-Baums

- Anwendung startet Parse-Vorgang
  - `DocumentBuilder myBuilder = myFactory.newDocumentBuilder();`  
`Document myDocument = myBuilder.parse(new File(argv[0]));`
- Parser liest XML-Dokument ein
- Parser erzeugt DOM-konforme Baumstruktur
- Anwendung greift auf diese Baumstruktur über den Document-Node zu

- `Element rootElement = myDocument.getDocumentElement();`



# Anwendung 2/12

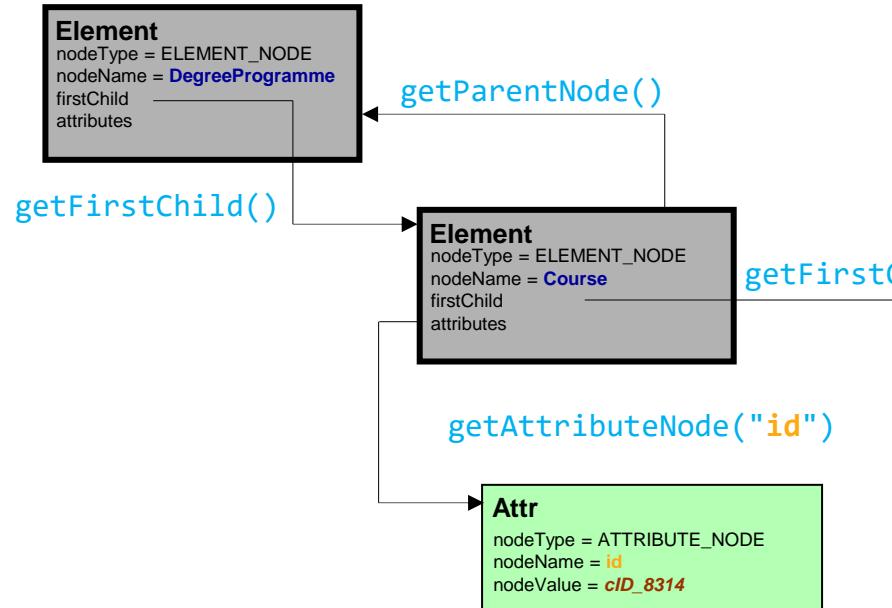
## Erzeugen eines DOM-Baums

```
import javax.xml.parsers.*; ← Definiert DocumentBuilderFactory und
... DocumentBuilder Klassen
import org.w3c.dom.*; ← Definiert DOM-API für XML-Dokumente
... (W3C-Standard)

public void load() {
 //Get instance of DocumentBuilderFactory
 DocumentBuilderFactory myFactory = DocumentBuilderFactory.newInstance();
 //By default, parsers won't validate documents
 //Set this to true to turn on validation against DTD
 myFactory.setValidating(true); // Validierung gegen DTD
 //By default, parsers won't ignore whitespaces within element contents
 //Set this to true to ignore whitespaces
 myFactory.setIgnoringElementContentWhitespace(true); [...]
try {
 //Get instance of DocumentBuilder
 DocumentBuilder myBuilder = myFactory.newDocumentBuilder();
 myBuilder.setErrorHandler(new MyErrorHandler());
 //Create document object - DOM
 Document myDocument = myBuilder.parse(new File("CourseCatalog.xml"));
 Element courseCatalog = myDocument.getDocumentElement(); //Root element
 ...
}
catch (ParserConfigurationException exp) { ... }
catch (SAXException exp) { ... }
catch (Exception exp) { ... }
```

# Anwendung 3/12

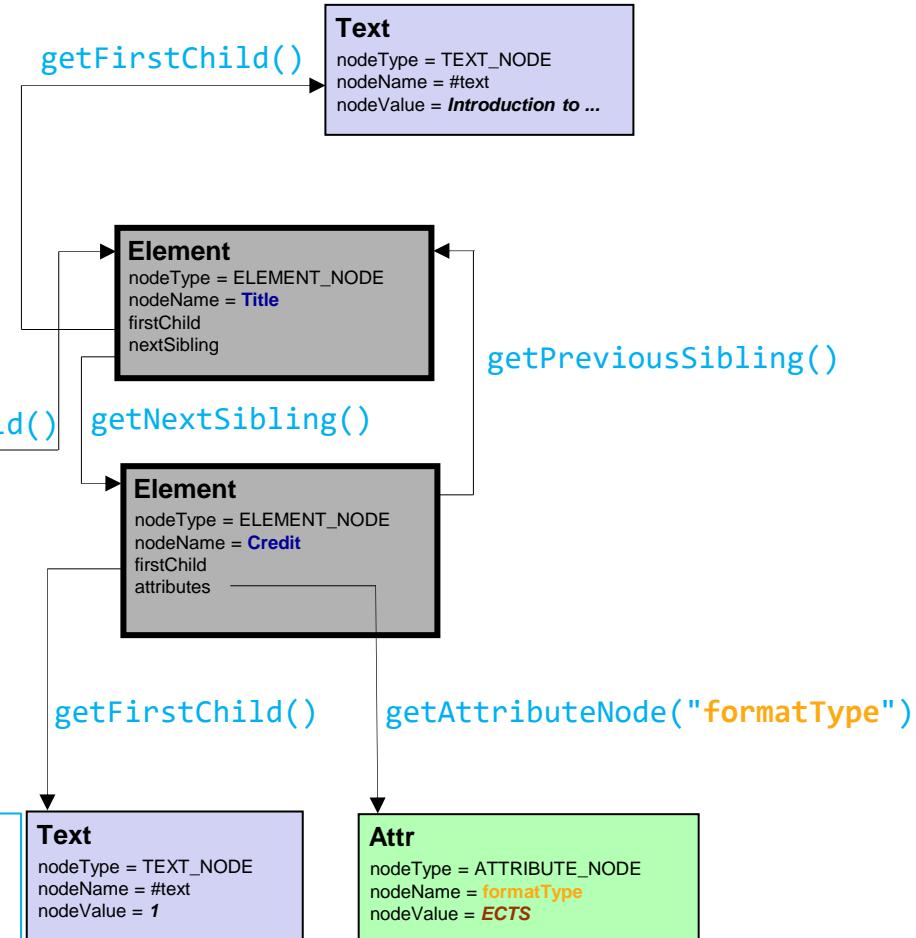
## Navigieren im DOM-Baum



```

...
<DegreeProgramme code="0307" name="Software Engineering">
 <Course id="cID_8314" semesterHours="1" language="en" semester="4">
 <Title>Introduction to semi-structured data models and XML</Title>
 <Credit formatType="ECTS">1</Credit>
 <CourseType type="Lecture"/>
 <Room roomNumber="1.004" building="FH1">CELUM HS2</Room>
 <Instructor instructorNumber="p22080">Julian Haslinger</Instructor>
 </Course> <!-- Further Courses ... -->
</DegreeProgramme> <!-- Further DegreeProgrammes ... -->
...

```



# Anwendung 4/12

## Informationsextraktion

### Ausgabe:

```
DegreeProgramme: Software Engineering
Course: cID_8314
Title: Introduction to ...
...
```

```
NodeList dpList = courseCatalog.getChildNodes(); //Child element DegreeProgramme

// Iterate DegreeProgrammes
for (int i = 0; i < dpList.getLength(); i++) {
 Element dp = (Element)dpList.item(i);
 String name = dp.getAttribute("name").getValue();
 System.out.println(dp.getNodeName() + ": " + name);

 NodeList cList= dp.getElementsByTagName("Course");

 // Iterate Courses
 for (int j = 0; j < cList.getLength(); j++) {
 Element c = (Element)cList.item(j);
 String id = c.getAttribute("id").getValue();
 System.out.println(c.getNodeName() + ": " + id);

 NodeList tList= c.getElementsByTagName("Title");
 Element t = (Element)tList.item(0);
 Text tText = (Text)t.getFirstChild();
 System.out.println(t.getNodeName() + ": " + tText.getNodeValue());
 }
}
```

# Anwendung 5/12

## Erzeugen von DOM-Knoten

- Knotentyp Document stellt Methoden zum Erzeugen von Knoten zur Verfügung (Auszug):

Methode	Erstellt Knoten vom Typ ...
<code>createAttribute()</code>	Attribut
<code>createElement()</code>	Element
<code>createTextNode()</code>	Text
<code>createComment()</code>	Kommentar

- Methoden erzeugen Knoten von einem bestimmten Typ, mit denen weitergearbeitet werden kann

# Anwendung 6/12

## Beispiel: Erzeugen von DOM-Knoten

- Beispiel: Neues Element erzeugen

```
newElementNode = myDocument.createElement("Course");
```

- Beispiel: Attribut zu einem Element-Knoten hinzufügen

```
newAttributeNode = myDocument.createAttribute("id");
newAttributeNode.setValue("cID8315");
newElementNode.setAttributeNode(newAttributeNode);
```

# Anwendung 7/12

## Bearbeiten eines DOM-Baums

- Knotentyp Node stellt Methoden zum Verändern der Struktur des XML-Dokuments zur Verfügung (Auszug):

Methode	Erstellt Knoten vom Typ ...
<code>appendChild(new)</code>	Fügt neuen Knoten <code>new</code> an das Ende der Knotenliste hinzu
<code>insertBefore(new, ref)</code>	Fügt neuen Knoten <code>new</code> vor Knoten <code>ref</code> in der Knotenliste ein
<code>removeChild(old)</code>	Löscht Knoten <code>old</code> aus der Knotenliste
<code>replaceChild(new, old)</code>	Ersetzt Knoten <code>old</code> durch Knoten <code>new</code> in der Knotenliste

# Anwendung 8/12

## Beispiel: Bearbeiten eines DOM-Baums

- Beispiel: Neues Element hinzufügen

```
currentElementNode.appendChild(newElementNode);
```

- Beispiel: Bestehendes Element löschen

```
oldElementNode = currentElementNode.removeChild(
 currentElementNode.childNodes().item(1));
```

# Anwendung 9/12

## Abfragen eines DOM-Baums

- Beispiel: Abfragen eines DOM-Baums mittels **XPath** und **XPathFactory**

```
...
Document myDocument = myBuilder.parse(new File("CourseCatalog.xml"));
Element courseCatalog = myDocument.getDocumentElement(); //Root element

XPathFactory xPathFactory = XPathFactory.newInstance();
XPath xPath = xPathFactory.newXPath();

String expression = "//*[text() [contains(., 'Haslinger')]]";

Double nodeCount = (Double) xPath.evaluate("count(" + expression + ")",
 courseCatalog, XPathConstants.NUMBER);
NodeList nodes = (NodeList) xPath.evaluate(expression, courseCatalog,
 XPathConstants.NODESET);

System.out.println("XPath evaluation (" + nodeCount.intValue() + " nodes match):");
for (int i = 0; i < nodes.getLength(); i++) {
 System.out.println(i + ". Node: " + ((Element)nodes.item(i)).getTextContent());
}
catch (XPathExpressionException e) {e.printStackTrace();} // XPathExpressionException
...
```

# Anwendung 10/12

## Transformieren eines DOM-Baums

### ■ Beispiel: Transformieren eines DOM-Baums mittels

**Transformer** und **TransformerFactory**

- Stylesheet und Transformer erzeugen

```
TransformerFactory myFactory = TransformerFactory.newInstance();
File styleSheetFile = new File("myXSL.xsl");
StreamSource styleSheet = new StreamSource(styleSheetFile);
Transformer myTransformer = myFactory.newTransformer(styleSheet);
```

- Quelldokument erzeugen

```
File inFile = new File("myXML.xml");
Document myDocument = myBuilder.parse(inFile);
DOMSource mySource = new DOMSource(myDocument);
```

- Ergebnisdokument erzeugen

```
File outFile = new File("myHTML.html");
StreamResult myResult = new StreamResult(
 new FileOutputStream(outFile));
```

- Transformation durchführen

```
myTransformer.transform(mySource, myResult);
```

# Anwendung 11/12

## Serialisieren eines DOM-Baums

- Serialisieren eines DOM-Baums mittels XSL-Transformation
  - Quelle (DOM-Baum) in Ziel (Datei) transformieren  
(Achtung: es erfolgt keine wirkliche Datentransformation)
- Beispiel: DOM-Baum serialisieren
  - Transformer erzeugen

```
TransformerFactory myFactory = TransformerFactory.newInstance();
Transformer myTransformer = myFactory.newTransformer();
```
  - Ausgabeeigenschaften setzen (für DOCTYPE und Leerzeilen)  

```
myTransformer.setOutputProperty("doctype-system", "myDTD.dtd");
myTransformer.setOutputProperty("indent", "yes");
```
  - Transformation mit Dokument als Quelle und FileOutputStream als Ziel durchführen  

```
myTransformer.transform(
 new DOMSource(doc), // source
 new StreamResult(new FileOutputStream(file)) // target
);
```

# Anwendung 12/12

## Serialisieren eines DOM-Baums

- Serialisieren eines DOM-Baums mittels **XMLSerializer**
  - DOM Level 3 stellt Interface zur Verfügung (in JDK 6 verfügbar)
- Beispiel: DOM-Baum serialisieren

```
OutputFormat format = new OutputFormat(DOM_document);

FileOutputStream fos = new FileOutputStream(resultFile);
XMLSerializer mySerializer = new XMLSerializer(fos, format);

mySerializer.serialize(DOM_document);
```

# Zusammenfassung 1/3

## Struktur von DOM

- Baumstruktur wird im Speicher verwaltet
  - ⇒ Verarbeitung großer XML-Dokumente kann Probleme mit sich bringen
- DOM API enthält Schnittstellen für das Navigieren, Manipulieren, Abfragen und Erstellen von XML-Dokumenten
- DOM API spezifiziert nur die Schnittstelle der Methoden (z.B.: `createElement(String name)`)
  - ⇒ Implementierung den Parsern (bspw. von Apache, Oracle, Microsoft) überlassen
- Struktur und Inhalt des Baumes kann mittels DOM-Methoden verändert werden

# Zusammenfassung 2/3

## Vor- und Nachteile

### ■ Vorteile

- W3C Standard von vielen Parsern implementiert
- Einfache Verarbeitung von XML-Dokumenten
- Bietet auch die Möglichkeit XML-Dokumente zu erstellen und zu modifizieren

### ■ Nachteile

- Hoher Speicherbedarf bei großen XML-Dokumenten
- Hohe Geschwindigkeitsverluste bei großen XML-Dokumenten
- Nicht für Streaming geeignet; Traversieren erst möglich, wenn XML-Dokument vollständig geladen
- Abstrahiert nicht von der XML-Syntax

# Zusammenfassung 3/3

## DOM versus SAX

### ■ DOM ist **dokumenten-orientiert**

- DOM wird vom W3C entwickelt (definierter Weg)
- Das gesamte XML-Dokument wird im Hauptspeicher als DOM-Baum aufgebaut
- DOM-Baum kann traversiert bzw. bearbeitet werden
  - Verwendet man bei Änderungen der Struktur oder der Daten

### ■ SAX ist **ereignis-orientiert**

- SAX wird von einer informellen Gruppe definiert (quasi Standard)
- Das Dokument muss nicht als Ganzes in den Hauptspeicher geladen werden
  - ⇒ wesentlich schneller bei großen Dokumenten
- Direkte Manipulation von XML-Dokument nicht möglich; kein wahlfreier Zugriff
  - Verwendet man zum gezielten Auslesen von bestimmten Teilen in XML-Dokumenten

# Literatur

- W3C DOM Spezifikation

[www.w3.org/DOM](http://www.w3.org/DOM)

- Java

<http://docs.oracle.com/javase/tutorial/jaxp/index.html>

- Microsoft XML Core Services bzw. System.Xml

<https://msdn.microsoft.com/en-us/library/ms763742.aspx>

<https://msdn.microsoft.com/library/system.xml.aspx>

- W3Schools.com: XML DOM Tutorial

[https://www.w3schools.com/xml/dom\\_intro.asp](https://www.w3schools.com/xml/dom_intro.asp)

# Inhalt

- Einführung
- Kategorien von XML-Parser
- Aufbau XML-Parser
- SAX – Simple API for XML
- DOM – Document Object Model
- **StAX – Streaming API for XML**
  - Einführung
  - Cursor-API
  - Iterator-API
  - Zusammenfassung
- Schema-Übersetzer – XML Data Binding
- Zusammenfassung

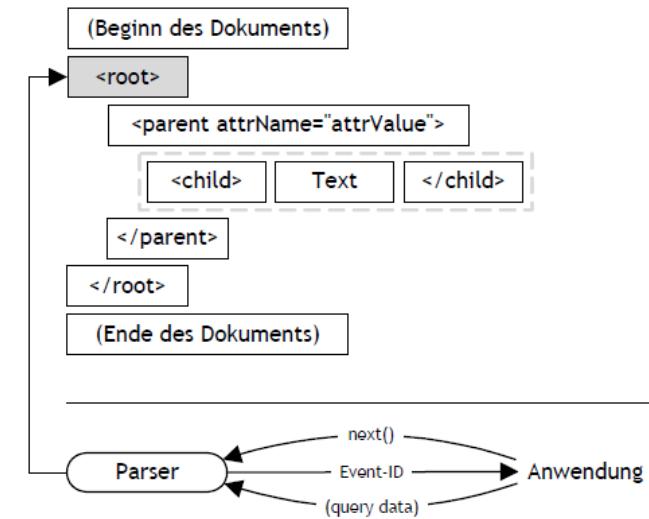
# Einführung

## Streaming API for XML

- Jüngster Ansatz zum Parsen von XML-Dokumenten (2004)
- Ereignisorientierter (Mehrschritt-) Pull-Parser
  - Einzelne Bestandteile des XML-Dokuments werden „auf Anfrage“ sequentiell vom XML-Parser geholt.
- Gründe für Pull-basierten Parser:
  - Verarbeitung, wenn Applikation bereit
  - Zustandsverwaltung
- verbindet die Vorteile von SAX und DOM
  - effizientes Arbeiten mit großen Dokumenten
  - Schreiben/Erzeugen von XML-Dokumenten
- StAX gliedert sich in eine
  - Cursor-basierte API
  - Iterator-basierte API
- seit Java 6.0 Teil der Java Development Kits (JDK)

# Cursor-API

- Logischer Cursor wird über einem Strom von **XML-Elementen** geführt
- In jedem von der Anwendung ausgelösten Schritt wird der Cursor von einem ereignisauslösenden Element zum Nächsten geschoben
- Cursor verwaltet Ereigniseigenschaften
  - ressourcensparend
- Ereignisauslösende Elemente:
  - StartDocument, StartElement etc. (vgl. SAX)



# Cursor-API

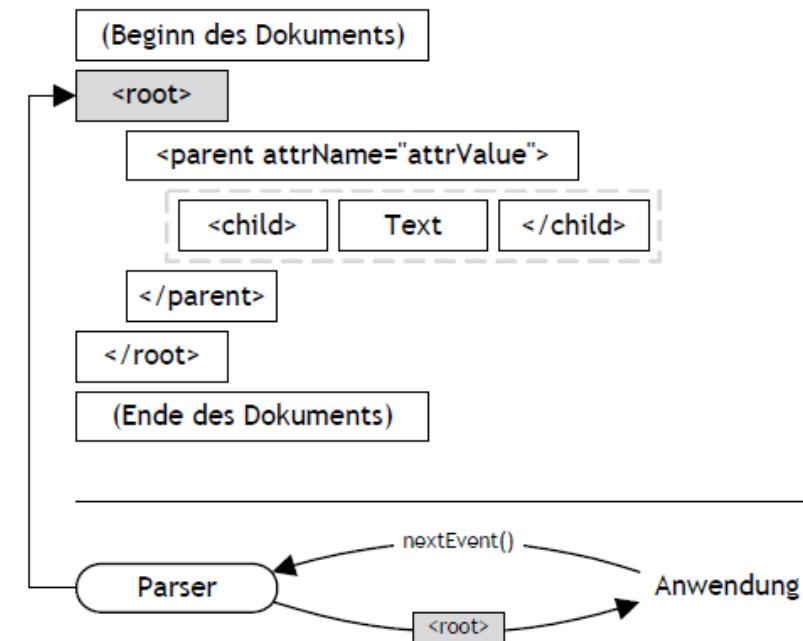
## Beispiel

```
URL u = new URL("http://www.fh-hagenberg.at/CourseCatalog.xml");
InputStream in = u.openStream();
XMLInputFactory myFactory = XMLInputFactory.newInstance();
XMLStreamReader myParser = myFactory.createXMLStreamReader(in);

try {
 int event = myParser.getEventType();
 while (true) {
 switch (event) {
 case XMLStreamConstants.START_ELEMENT:
 System.out.println("Start Element: " + myParser.getLocalName());
 for(int i = 0, n = myParser.getAttributeCount(); i < n; ++i)
 System.out.println("Attribute: " + myParser.getAttributeName(i)
 + "=" + myParser.getAttributeValue(i));
 break;
 // add cases for each event of interest
 }
 if (! myParser.hasNext())
 break;
 event = myParser.next();
 } finally {
 myParser.close();
 }
}
```

# Iterator-API

- Iterator iteriert über einem Strom von **Ereignisobjekten**
- **Ereignisobjekte** enthalten Informationen zu den Ereignissen
- Ereignisobjekte können objekt-orientiert verarbeitet werden
- Verwalten vieler Objekte steigert Ressourcenbedarf



# Iterator-API

## Beispiel

```
...
XMLInputFactory inputFactory = XMLInputFactory.newInstance();
XMLEventReader myReader = inputFactory.createXMLEventReader(input);

try {
 while (myReader.hasNext()) {
 XMLEvent e = myReader.nextEvent();
 if (e.isCharacters() && ((Characters) e).isWhiteSpace())
 continue;
 System.out.println(e);
 }
} finally {
 myReader.close();
}
```

# Serialisieren von XML-Dokumenten

- Cursor- und Iterator-API unterstützen das Serialisieren von XML-Dokumenten
- Zentrale Klassen:
  - `XMLStreamWriter` und `XMLEventWriter`

## Serialisieren mit Cursor-API:

```
XMLOutputFactory factory = XMLOutputFactory.newInstance();
XMLStreamWriter writer = factory.createXMLStreamWriter(
 new FileOutputStream("c:/CourseCatalog.xml"));

// XML-Header wird erzeugt
writer.writeStartDocument();
// Zuerst wird das Wurzelement mit Attribut geschrieben
writer.writeStartElement("CourseCatalog");
writer.writeAttribute("year", "2019"); [...]
// Kindelement DegreeeProgramme mit Attributen erzeugen
writer.writeStartElement("DegreeProgramme");
writer.writeAttribute("code", "0307"); [...]
writer.writeEndElement();
writer.writeEndElement();
writer.writeEndDocument();
writer.close();
```

```
<?xml version="1.0">
<CourseCatalog year="2019" ...>
 <DegreeProgramme code="0307" .../>
</CourseCatalog>
```

# Zusammenfassung

- Dritte Variante auf XML-Daten zuzugreifen
- Durch die verschiedenen API- Realisierungen flexiblerer Umgang
- Je nach Wahl ähnlich performant wie SAX
- Unterstützt die Serialisierung von XML-Dokumenten

# Zusammenfassung

## Vergleich SAX - DOM - StAX

Kriterien	SAX	DOM	StAX Cursor-API	StAX Iterator-API
API-Klassifikation	Push-Mehrschritt	Pull-Einschritt In Memory Tree	Pull-Mehrschritt	Pull-Mehrschritt
Wiederholter Zugriff	✗ sequentiell	✓ wahlfrei, (Objektmodell)	✗ sequentiell	✗ sequentiell
Eignung für Verarbeitung großer Dokumente	++	--	++	+
XML Lesen	✓	✓	✓	✓
XML Serialisieren	✗	✓	✓	✓
XML Manipulation	✗	✓	✗	✗
XPath-Unterstützung	✗	✓	✗	✗
Validierung	✓	✓	optional	optional

# Literatur

- JSR 173: Streaming API for XML

<http://jcp.org/en/jsr/detail?id=173>

- XML Pull Parsing

<http://www.xmlpull.org/index.shtml>

- Streaming API for XML

<https://docs.oracle.com/javase/tutorial/jaxp/stax/>

# Inhalt

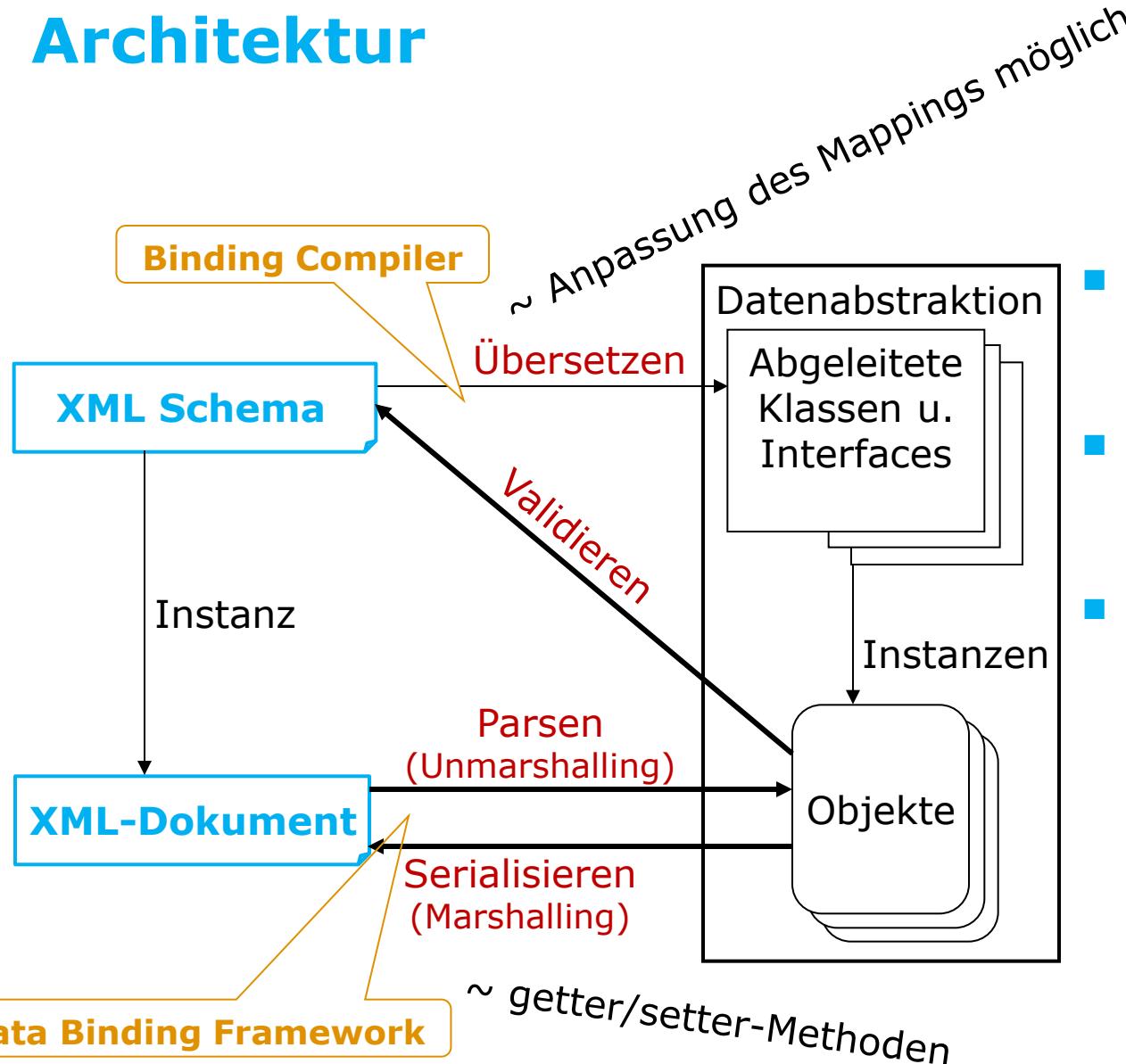
- Einführung
- Kategorien von XML-Parser
- Aufbau XML-Parser
- SAX – Simple API for XML
- DOM – Document Object Model
- StAX – Streaming API for XML
- **Schema-Übersetzer – XML Data Binding**
- Zusammenfassung

# Einführung

## XML Data Binding

- Abbildung von XML Schema in Java- oder C#-Klassen
- Direkte Verbindung zwischen Daten eines XML-Dokuments und den Objekten einer Programmiersprache

# Architektur



- Klassen/Interfaces/Methoden werden generiert
- Zweck: Schnittstelle, die von XML abstrahiert
- Lesen, Modifizieren und Erstellen von XML-Dokumenten

# XML Data Binding Frameworks

- JAXB (Java Architecture for XML Binding)
  - XML Data Binding Framework
  - Bestandteil von JDK
  - Erzeugt aus jedem komplexen XML-Teil (mit Kindknoten) eine Klasse.
  - Für die Elemente werden *Accessor/Getter* und *Mutator/Setter* (getXxx und setXxx) erzeugt.
- Apache XMLBeans, JiBX, Castor, Enhydra Zeus, JBind, ...
- .Net Data Binding, z.b. XMLDataProvider
- ...

# Beispiel

## JAXB - Generieren von Klassen/Interfaces

```
...
<xsd:complexType name="USAddress">
 <xsd:sequence>
 <xsd:element name="name" type="xsd:string"/>
 <xsd:element name="street" type="xsd:string"/>
 <xsd:element name="city" type="xsd:string"/>
 <xsd:element name="state" type="xsd:string"/>
 <xsd:element name="zip" type="xsd:decimal"/>
 </xsd:sequence>
 <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
...
address.xsd
```

```
package address;
public interface USAddress {
 String getState();
 void setState(String value);
 java.math.BigDecimal getZip();
 void setZip(java.math.BigDecimal value);
 String getCountry();
 void setCountry(String value);
 ...
 String getName();
 void setName(String value);
}
```

USAddress.java

```
...
JAXBContext jc = JAXBContext.newInstance("address");
ObjectFactory objFactory = new ObjectFactory();
USAddress address = objFactory.createUSAddress();
```

ObjectFactory: Zur Objekterzeugung

**Generierung**  
USAddress.java,  
USAddressImpl.java,  
ObjectFactory, ...

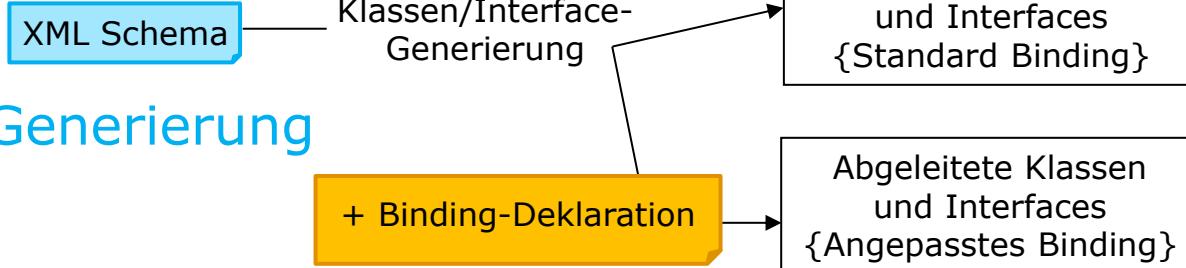
**Anwendung**

# Beispiel

## JAXB - Anpassen der Generierung

### ■ Binding Deklaration

- Separate XML-Datei (eigenes XML-Vokabular)
- "Inline" in der XML Schema-Datei im `annotation`-Element



```
...
<xs:annotation>
 <xs:appinfo>
 <jaxb:globalBindings generateIsSetMethod="true"
 bindingStyle="modelGroupBinding"
 choiceContentProperty="true" >
 ...
 <xxj:serializable uid="12343"/>
 <jaxb:javaType name="short"
 xmlType="xs:long"
 printMethod="javax.xml.bind.DatatypeConverter.printShort"
 parseMethod="javax.xml.bind.DatatypeConverter.parseShort"/>
 </jaxb:globalBindings>
 </xs:appinfo>
</xs:annotation>
...
```

uid=12343 wird generierten Klassen zugewiesen

Globale Anpassungen

isSet-Methode wird für alle Klassen generiert (stellt fest, ob Default-Wert verwendet wurde)

xs:long wird auf Java-Typ short abgebildet

Spezielle Methoden für das Marshalling und Unmarshalling sollen verwendet werden

Inline Binding Deklaration

# Zusammenfassung

## ■ Vorteile

- Abstraktion von den Low-Level-APIs von DOM und SAX
- Abstraktion von den Details des Parsing-Vorgangs
- Binding-Verhalten kann angepasst werden
- Entwicklungszeit und Fehleranfälligkeit werden verringert

## ■ Nachteile

- Höhere Einarbeitungszeit
- Hoher Speicherverbrauch bei großen XML-Dokumenten
- Änderung im XML-Schema führen zu einer Neugenerierung der Klassen

# Literatur

## ■ XML Data Binding Resources

- [www.rpbourret.com/xml/XMLDataBinding.htm](http://www.rpbourret.com/xml/XMLDataBinding.htm)

## ■ Bücher

- B. McLaughlin: Java und XML Data Binding.  
O'Reilly & Associates, 2002



# Inhalt

---

- Einführung
- Kategorien von XML-Parser
- Aufbau XML-Parser
- SAX – Simple API for XML
- DOM – Document Object Model
- StAX – Streaming API for XML
- Schema-Übersetzer – XML Data Binding
- **Zusammenfassung**

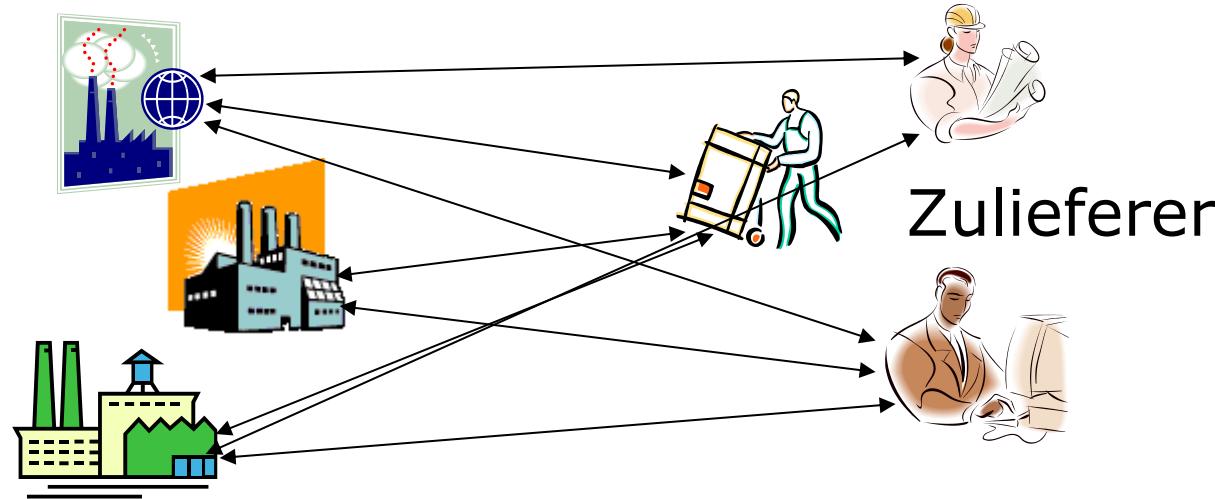
# Zusammenfassung

## Warum noch XML lernen?

- XML Data Binding (Schema-Übersetzer): XML Schema kann in Java- oder C#-Klassen übersetzt werden.
- Hiermit können XML-Dokumente gelesen, modifiziert und erstellt werden, ohne dass XML sichtbar ist.

Warum sich also noch mit XML und  
XML Schema beschäftigen?

# Typisches E-Business-Projekt



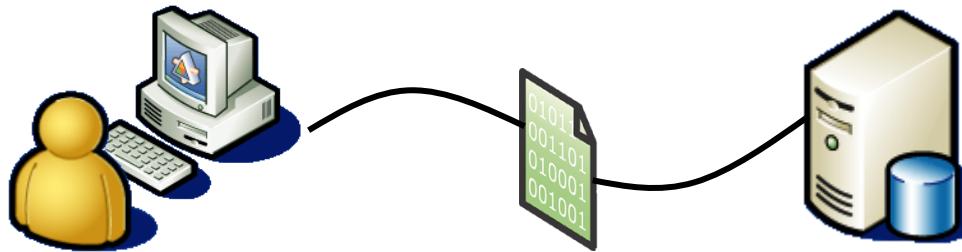
- Branchenvertreter wollen über das Internet miteinander Geschäfte abwickeln
- Sie müssen sich auf ein Austauschformat einigen.

# E-Business-Projekt: Phase I

- Welche Geschäftsdaten sollen ausgetauscht werden?
  - Gibt es bereits einen passenden Standard?
  - Wie sollen die Geschäftsdaten in XML repräsentiert werden?
  - Gibt es bereits einen geeigneten XML-Standard?
- Ziel: Branchenstandard in Form eines XML Schemas

Software-Architekten entwickeln gemeinsam  
einen XML-basierten Branchenstandard.

# E-Business-Projekt: Phase II



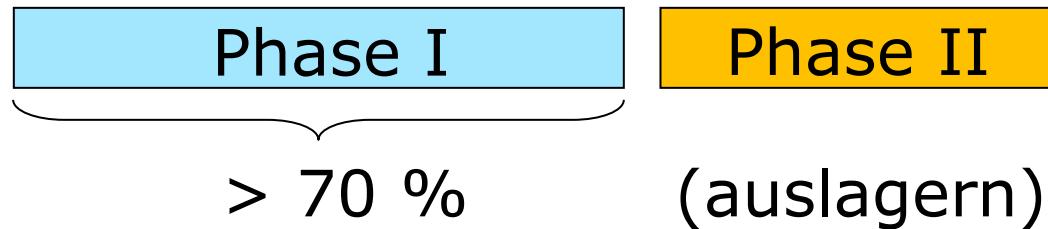
- Gegeben
  - Branchenstandard in Form eines XML-Schemas
  - gemeinsames Verständnis der XML-Syntax
- Aufgabe
  - Realisierung der Schnittstelle zwischen betriebsinterner Software und XML-Standard.

Programmierer können Schema-Übersetzer einsetzen und von XML abstrahieren.

# Warum sich mit XML beschäftigen?

- **Phase I:** Software-Architekten beschäftigen sich intensiv mit entsprechender Branche, XML und XML Schemata
- **Phase II:** Programmierer können Schema-Übersetzer einsetzen und von XML abstrahieren

Projektaufwand



## Modul 10

---

# XML und Datenbanken

---

Julian Haslinger



---

**Der vorliegende Foliensatz basiert vorwiegend auf:**

Klettke, Meike, Meyer, Holger: *XML & Datenbanken*, dpunkt.verlag, Jan. 2003

Türker, Can, *SQL:1999 & SQL:2003*, dpunkt, Feb. 2003

*Oracle XML DB Developer's Guide, 12c, Release 1 Dec. 2016*

# Inhalt

---

- Motivation
- Speichertechniken
- Abfragetechniken
- XML in Oracle

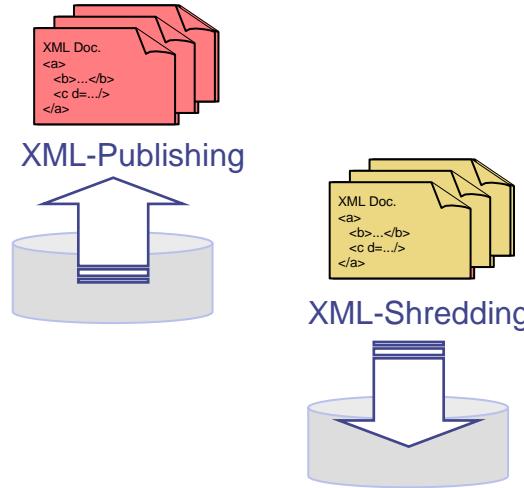
# Motivation

## XML und Datenbanken - Wozu?

- Existierende DB verwalten große Datenbestände
  - ⇒ **Publizieren** dieser Daten als XML-Dokumente
- Verwaltung von existierenden XML-Dokumenten
  - ⇒ **Speichern** in DB mit beschreibenden Attributen (Meta-Information)
- Vorteile von Datenbanken
  - Effiziente Speicherung großer, strukturierter Datenmengen
  - Abfragesprachen (SQL)
  - Optimierung
  - Benutzersichten, Datenschutz u. Datensicherheit
  - Mehrbenutzerfähigkeit – feinere Granularität als nur “Dokument”
  - Transaktionen und Recovery

→ DB sind essentielle Komponenten existierender IT-Infrastrukturen  
Die Wichtigkeit von DBs für Web-Applikationen steigt!

*“... The Web is one huge database...”*



# Motivation

## Herausforderung: Unterschiedliche Kategorien von XML-Dokumenten

### Datenorientierte XML-Dokumente

- Daten sind strukturiert und regulär, fein-granular und streng typisiert
- Ordnung der Elemente ist nicht signifikant
- Struktur wird meist durch ein Schema (z.B. DTD, XML Schema) festgelegt
- ⇒ Beispiele: Bestellungen, Rechnungen, Fahrpläne

```
<Bestellung bestellnummer="1012">
 <Kundennummer>8596</Kundennummer>
 <Position posNr="1">
 <Produktnummer>14896612</Produktnummer>
 <Anzahl>2</Anzahl> ...
 </Position> ...
</Bestellung>
```

### Dokumentenorientierte XML-Dokumente

- Daten sind semistrukturiert, irregulär, grob-granular und schwach typisiert
- Ordnung der Elemente ist signifikant
- Gemischter Inhalt kommt vor
- Struktur oft nicht mehr durch ein Schema fixierbar oder Schema ist sehr generisch
- ⇒ Beispiel: Geschäftsberichte, Schadensberichte

```
<Schadensbericht>
 Ein schweres <Ursache>Feuer</Ursache> verwüstete das
 Gebäude und forderte <Todesopfer>12</Todesopfer>
 Menschenleben. Erste polizeiliche Untersuchungen
 deuten auf eine <Motiv>Brandstiftung</Motiv> hin.
</Schadensbericht>
```

### Mischformen

- ⇒ Beispiel: Reiseinformationen, Produktkataloge

```
<Hotel>
 <Hotelname id=="4711" url=="www.sporthotel.at">
 Berger's Sporthotel</Hotelname>
 <Kategorie>4</Kategorie> ...
 <Anreisebeschreibung>Aus Richtung <emph>Salzburg</emph>
 kommend, fahren Sie über <emph>Lofer</emph> nach ...
 </Anreisebeschreibung>
 </Hotel>
```

# Inhalt

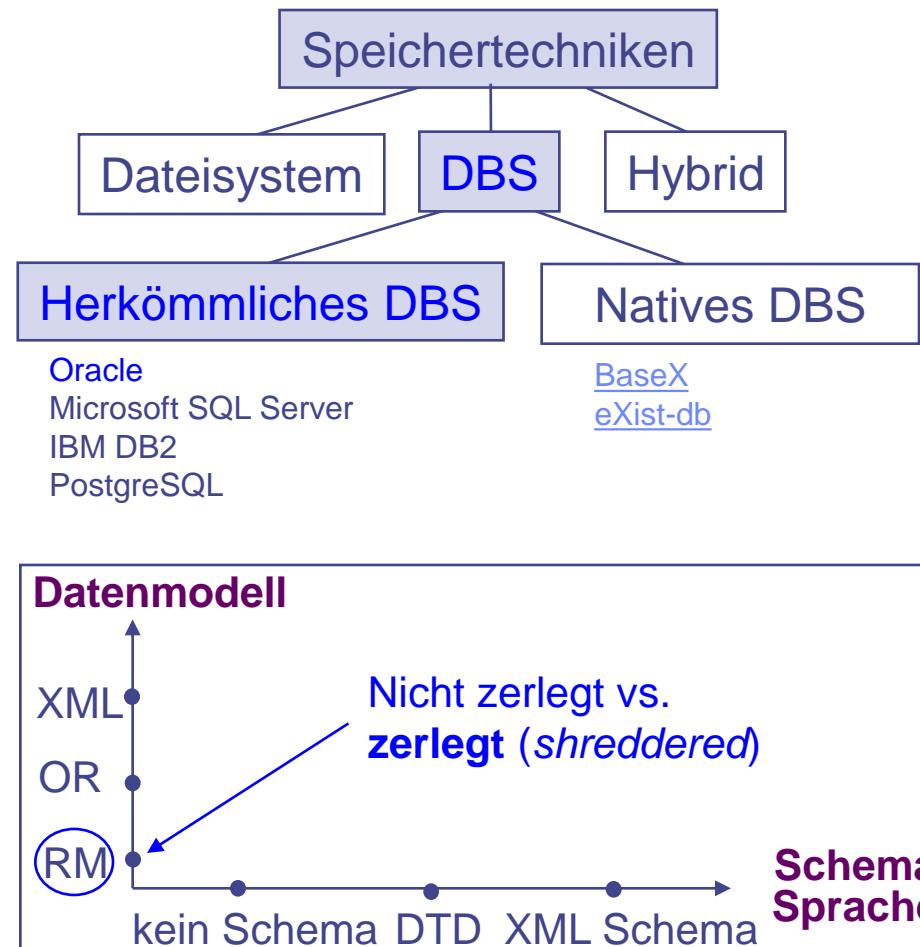
---

- Motivation ✓
- Speichertechniken
- Abfragetechniken
- XML in Oracle

# Speichertechniken

## Überblick

- Dateisystem
  - XML-Dokumente als Dateien auf Betriebssystemebene
  - Zusätzliche Speicherung von beschreibenden Attributen und Verweis auf Datei in DB möglich
- DBS
  - XML-Dokumente als Einheit oder zerlegt, u.U. gemeinsam mit beschreibenden Attributen, in der DB gespeichert
- Hybrid
  - XML-Dokumente oder Teile davon im Dateisystem und in der Datenbank
  - Redundante und nicht redundante Speicherung möglich



# Speichertechniken

## Relationale Speicherung: Datenmodell-Heterogenität

Datenmodell Ebene M2	<b>Relationale Konzepte</b>	<b>XML Konzepte</b>
	Relation → Attribut	↓ Element Typ    ---→ Attribut
Schema Ebene M1	<b>Relationales Schema</b>  Relation A → Attribut X Relation B → Attribut Y ...                         ...	<b>DTD / XML Schema (optional)</b>  -- Element Typ a    --> Attribut x -> Element Typ b    ---> Attribut y ...                         ...
Instanz Ebene M0	<b>Relationale Datenbank</b>  Tupel      → Wert	<b>XML-Dokument</b>  -> Element    -> Attribut ↓            ↓ Element Wert    Attribut Wert Attribut Wert

Legende: → ... besteht aus  
----→ ... kann bestehen aus

# Speichertechniken

## Relationale Speicherung: Datenmodell-Heterogenität

	RDBS	XML
<b>Struktur</b>	flach	geschachtelt
<b>Werte</b>	in Attributen gespeichert	in Attributen u. Elementen gespeichert
<b>Ordnung</b>	Tupel sind nicht geordnet	Elemente sind geordnet
<b>Schema</b>	notwendig	nicht notwendig
	ist vor d. Instanzen zu erstellen	kann nach Instanzen erstellt werden
	ist nicht Teil der Instanzen	implizites Schema in Form von Tags als Teil der Instanzdaten (selbst- beschreibend)

# Speichertechniken

## Abbildung auf Fixes Schema

### ■ Fixes Schema

Schema ist unabhängig vom Anwendungsbereich und vom Schema, zu dem abgebildet wird

- ohne Dekomposition: XML-Dokument wird als eine Einheit/ein Wert gespeichert (bspw. CLOB – textbasiert bzw. mittels *XMLType*)
- mit Dekomposition: XML-Dokument wird zerlegt (“shredding”)
  - ähnlich zu XML-API-Ansatz (DOM-Ansatz)

### ■ Beispiel für fixes Schema (mit Dekomposition)

Elemente	<u>DocID</u>	<u>ElemID</u>	<u>ElemName</u>	<u>ElemVorgänger</u>	<u>ElemPosition</u>	<u>ElemWert</u>

Attribute	<u>DocID</u>	<u>AttrName</u>	<u>ElemID</u>	<u>AttrWert</u>



# Speichertechniken

## Abbildung auf Abgeleitetes/Benutzerdefiniertes Schema

- Abgeleitetes Schema
  - Schema wird aus dem jeweils anderen Schema abgeleitet
    - ähnlich zu XML Data Binding-Ansatz
- Benutzerdefiniertes Schema
  - Schema entspricht dem Anwendungsbereich, wurde jedoch unabhängig vom Schema zu dem abgebildet wird entworfen
- Beispiel für benutzerdefiniertes od. abgeleitetes Schema

```
<!ELEMENT PDAKatalog (Hersteller*)>
<!ELEMENT Hersteller (HerstellerNr, PDA+)>
<!ATTLIST Hersteller Name CDATA #REQUIRED>...
```

Hersteller	HerstellerNr	Name	PDA	Name	Gewicht	HerstellerNr

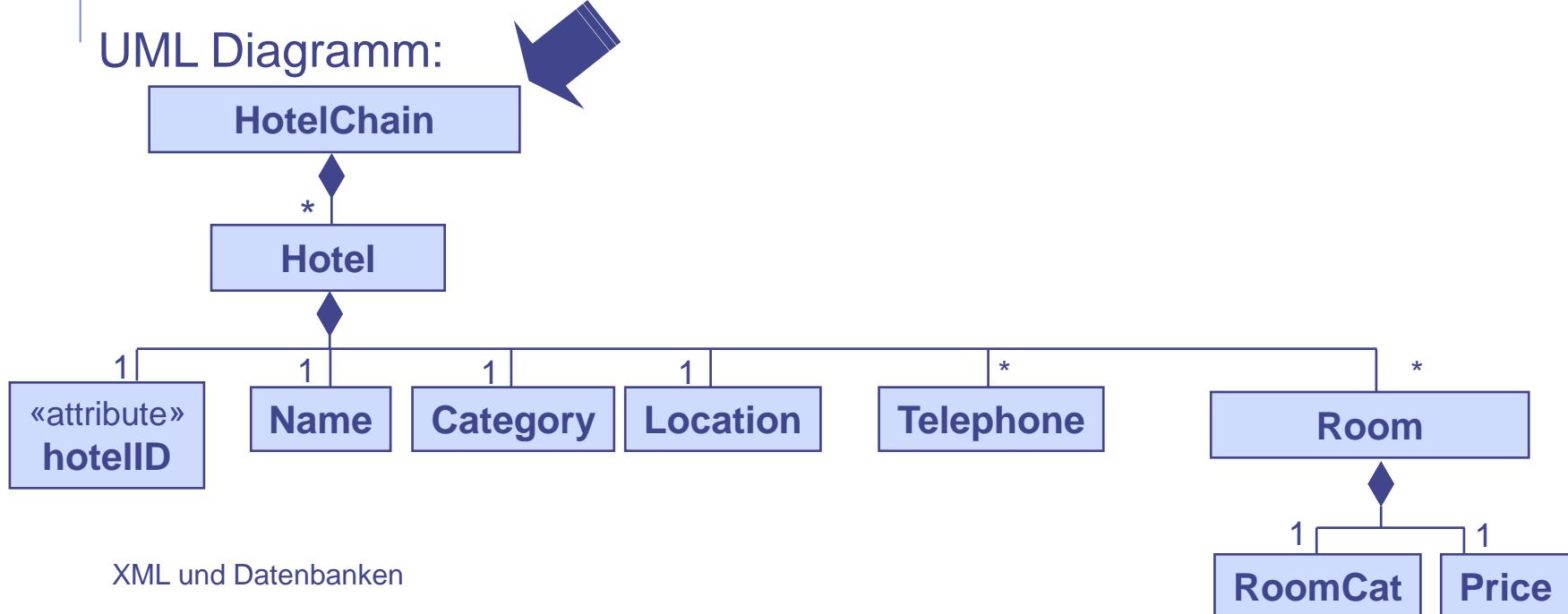
# Speichertechniken

## Beispiel

DTD:

```
<!ELEMENT HotelChain (Hotel*)>
<!ELEMENT Hotel (Name, Category, Location, Telephone*, Room*)>
 <!ATTLIST Hotel hotelID CDATA #REQUIRED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Category (#PCDATA)>
<!ELEMENT Location (#PCDATA)>
<!ELEMENT Telephone (#PCDATA)>
<!ELEMENT Room (RoomCat, Price)>
<!ELEMENT RoomCat (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
```

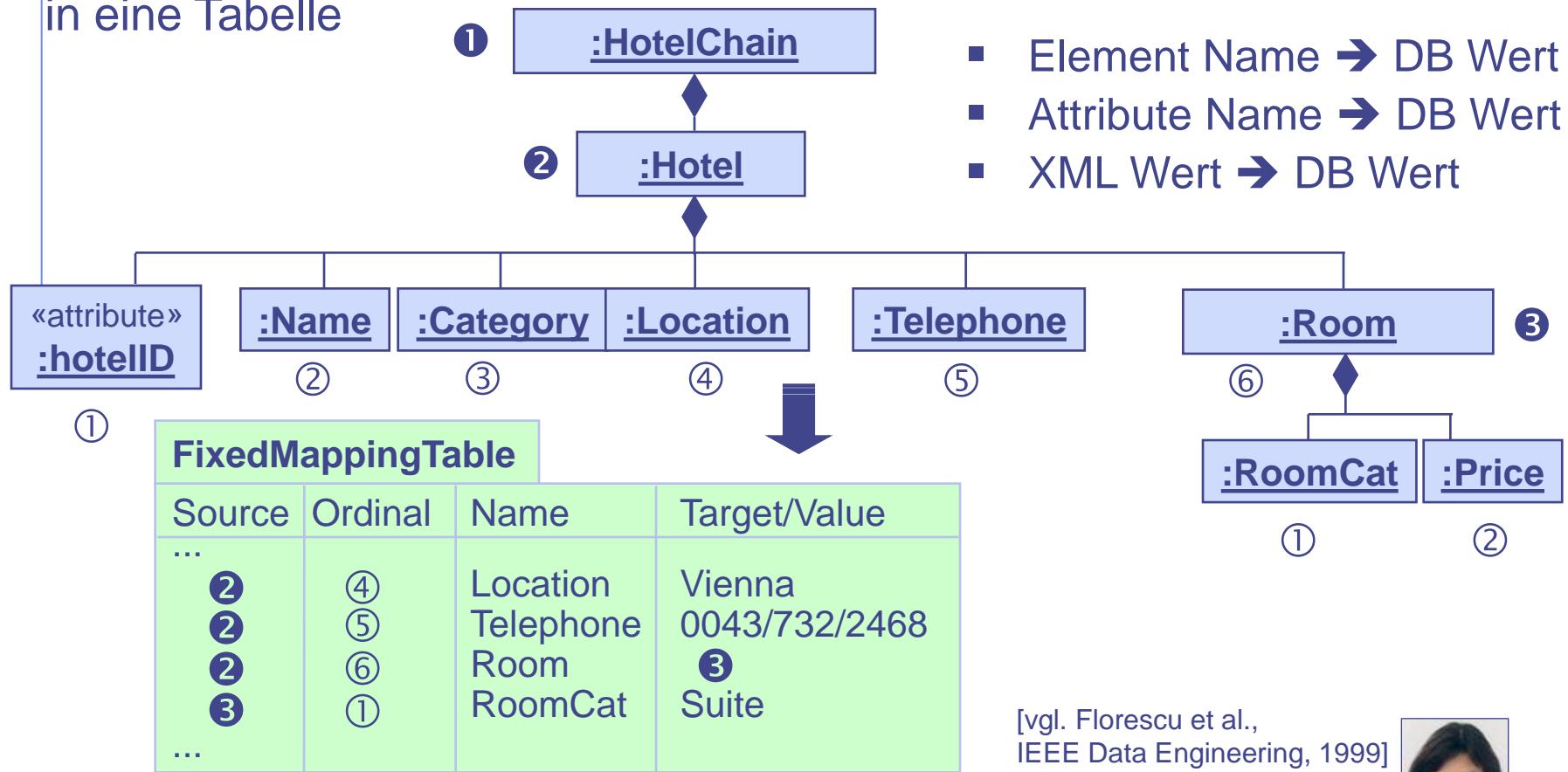
UML Diagramm:



# Speichertechniken

Abbildung auf fixes RDB-Schema

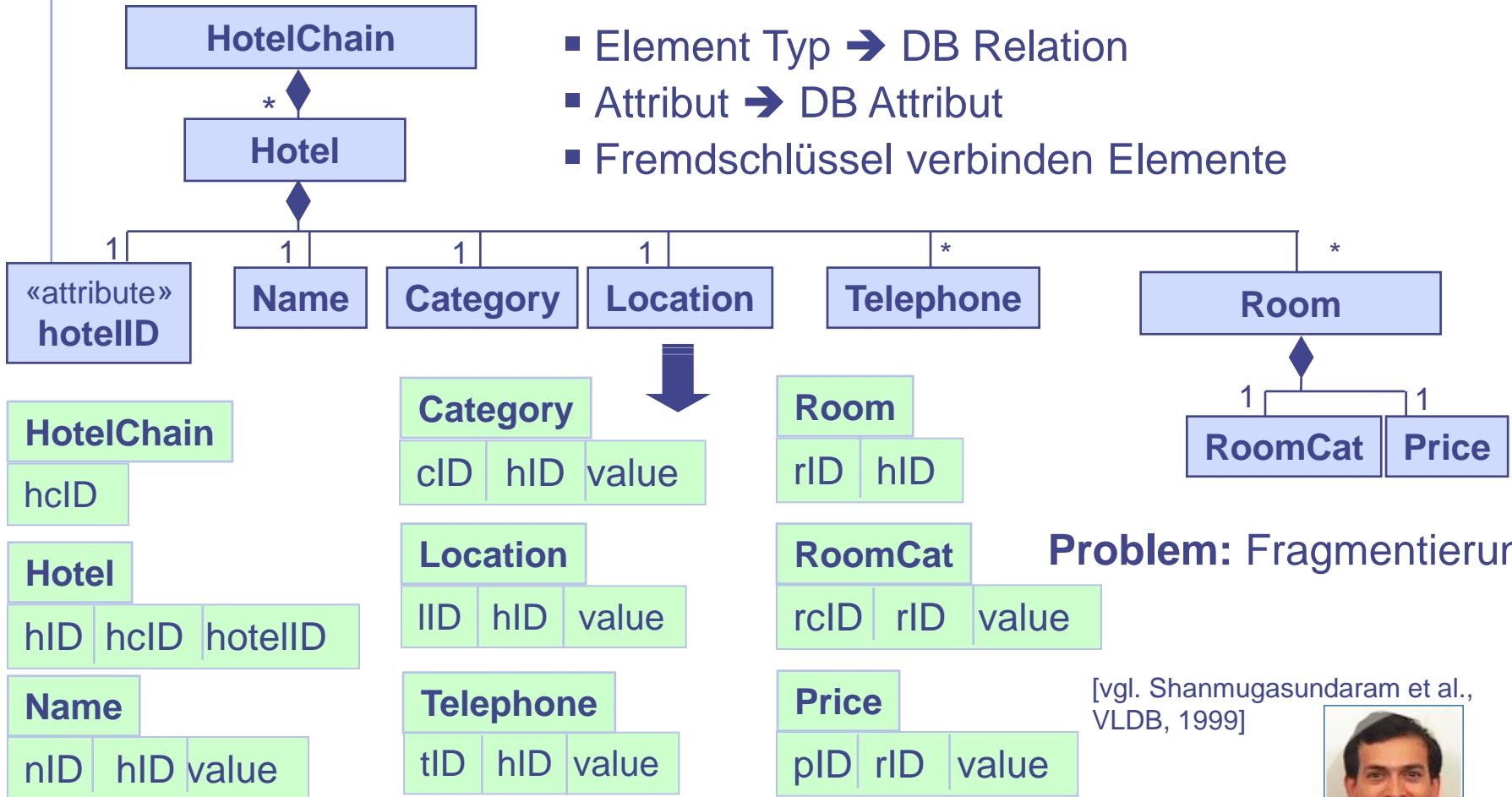
Beispiel: Dekomposition des Dokuments (Inhalt u. Schemainformation) in eine Tabelle



# Speichertechniken

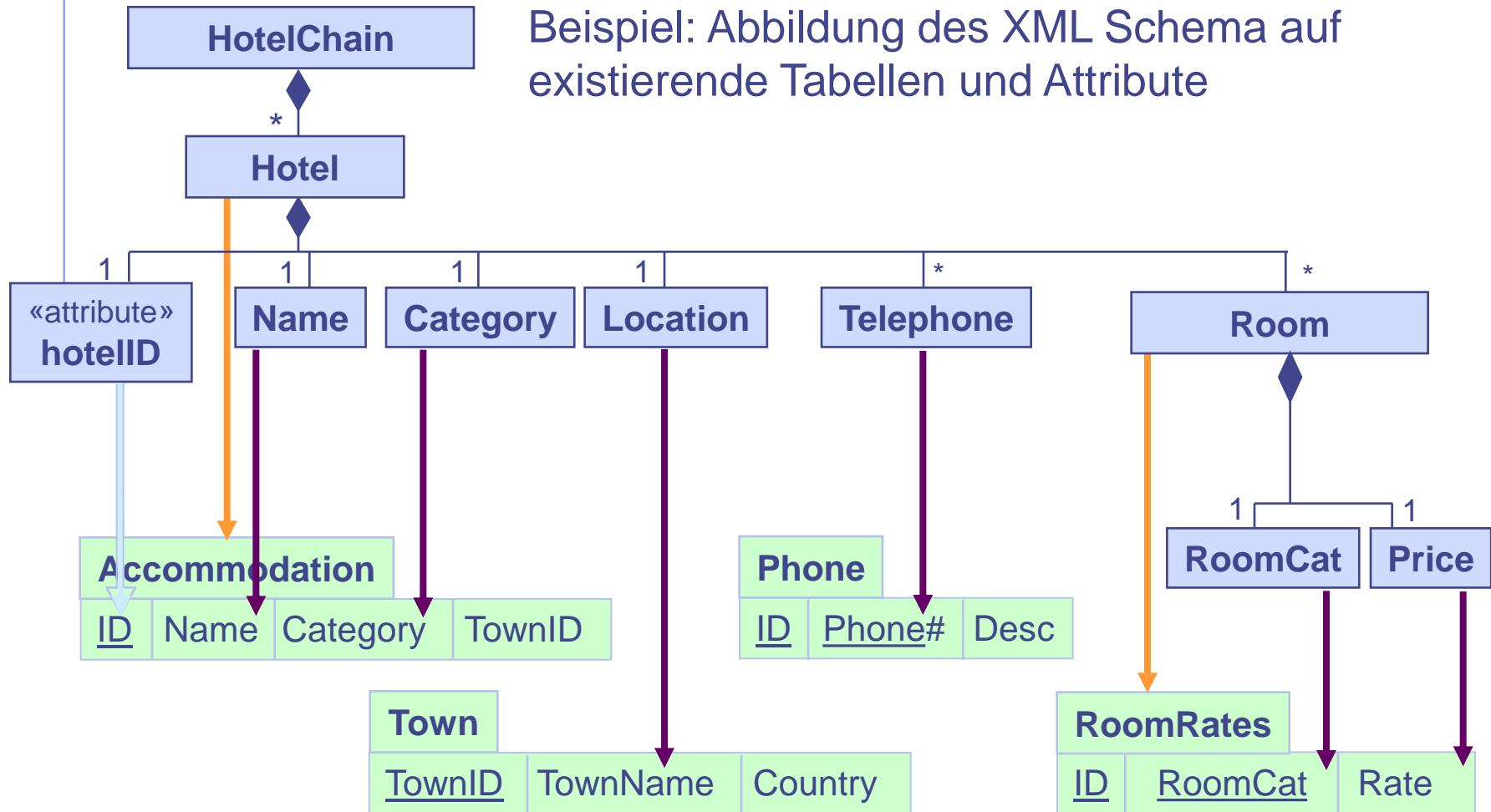
Abbildung auf abgeleitetes RDB-Schema

Beispiel: Dekomposition des XML Schema in Tabellen



# Speichertechniken

Abbildung auf benutzerdefiniertes RDB-Schema



# Speichertechniken

## Abbildungen – Vorteile und Nachteile

- Fix
  - Domäne wird durch das fixe Schema nicht abgebildet
  - Abfragen und Optimierung schwierig zu realisieren
  - + Fix auf DB-Seite:
    - ♦ kein Schema auf XML-Seite notwendig
    - ♦ für dokumentenorientiertes XML geeignet
- Abgeleitet
  - anderes Schema muss vorhanden sein
- Benutzerdefiniert
  - + Schema kann autonom vom anderen Schema entworfen werden
  - + Daten existierender DBs können genutzt werden
  - bei benutzerdefinierten Schemata auf beiden Seiten: Heterogenitäten
- Abgeleitet / benutzerdefiniert
  - + Domäne wird abgebildet
  - + Abfragen u. Optimierung können genutzt werden
  - + für datenorientiertes XML geeignet

# Speichertechniken

## Repräsentation des Abbildungswissens

- “Template-Driven” – Abbildungswissen fix-verdrahtet
  - in Transformationsprogrammen
  - in Abfragen

```
<?xml version="1.0" ?>
<Accommodation xmlns:sql="urn:schemas-ms-com:xml-sql">
 <sql:query>
 SELECT * FROM Accommodation FOR XML AUTO,ELEMENTS
 </sql:query>
</Accommodation>
```

- “Model-Driven” – Abbildungswissen reifizieren  
(= vergegenständlichen ~ als Metadaten speichern)
  - in Datei speichern, z.B. als XML-Dokument
  - in DB speichern

```
<?xml version="1.0" ?>
<Schema xmlns="urn:schemas-ms-com:xml-data"
 xmlns:dt="urn:schemas-microsoft-com:datatypes"
 xmlns:sql="urn:schemas-microsoft-com:xml-sql">
 <ElementType name="Phone" content="textOnly" />
 <ElementType name="Accommmodation" sql:relation="Accommmodation">
 <element type="Phone" sql:relation="Phone" sql:field="Number">
 <sql:relationship
 key-relation="Accommmodation"
 key="AcID"
 foreign-key="AccID"
 foreign-relation="Phone" />
 </element>
 </ElementType>
</Schema>
```

# Inhalt

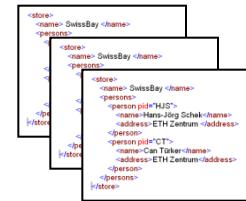
---

- Motivation ✓
- Speichertechniken ✓
- Abfragetechniken
- XML in Oracle

# Abfragetechniken

## Lesende vs. ändernde Abfragen

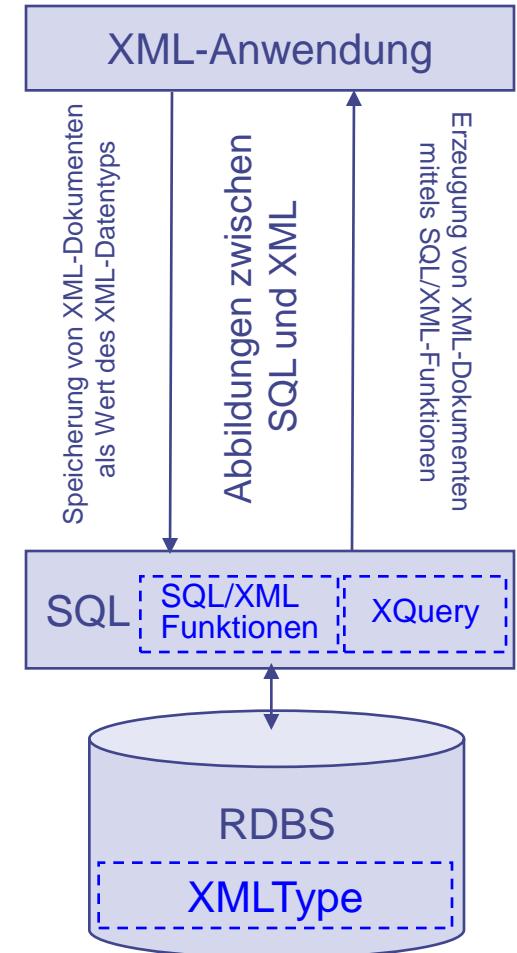
- Lesende („Read-Only“) Abfragen
  - XML-zentriert
    - ◆ Zugriff mit XML-basierter Sprache
    - W3C XQuery Standard
  - DB-zentriert
    - ◆ Zugriff mit SQL-basierter Sprache
    - SQL/XML – Teil des SQL2003-Standard
  - Herstellerspezifische Mechanismen
    - ◆ Weder DB- noch XML-zentriert
- Ändernde Abfragen
  - **XQuery Update Facility** (W3C Recommendation)
    - ◆ <https://www.w3.org/TR/xquery-update-10/> (2011)
    - ◆ <https://www.w3.org/TR/xquery-update-30/> (2017)



# Abfragetechniken

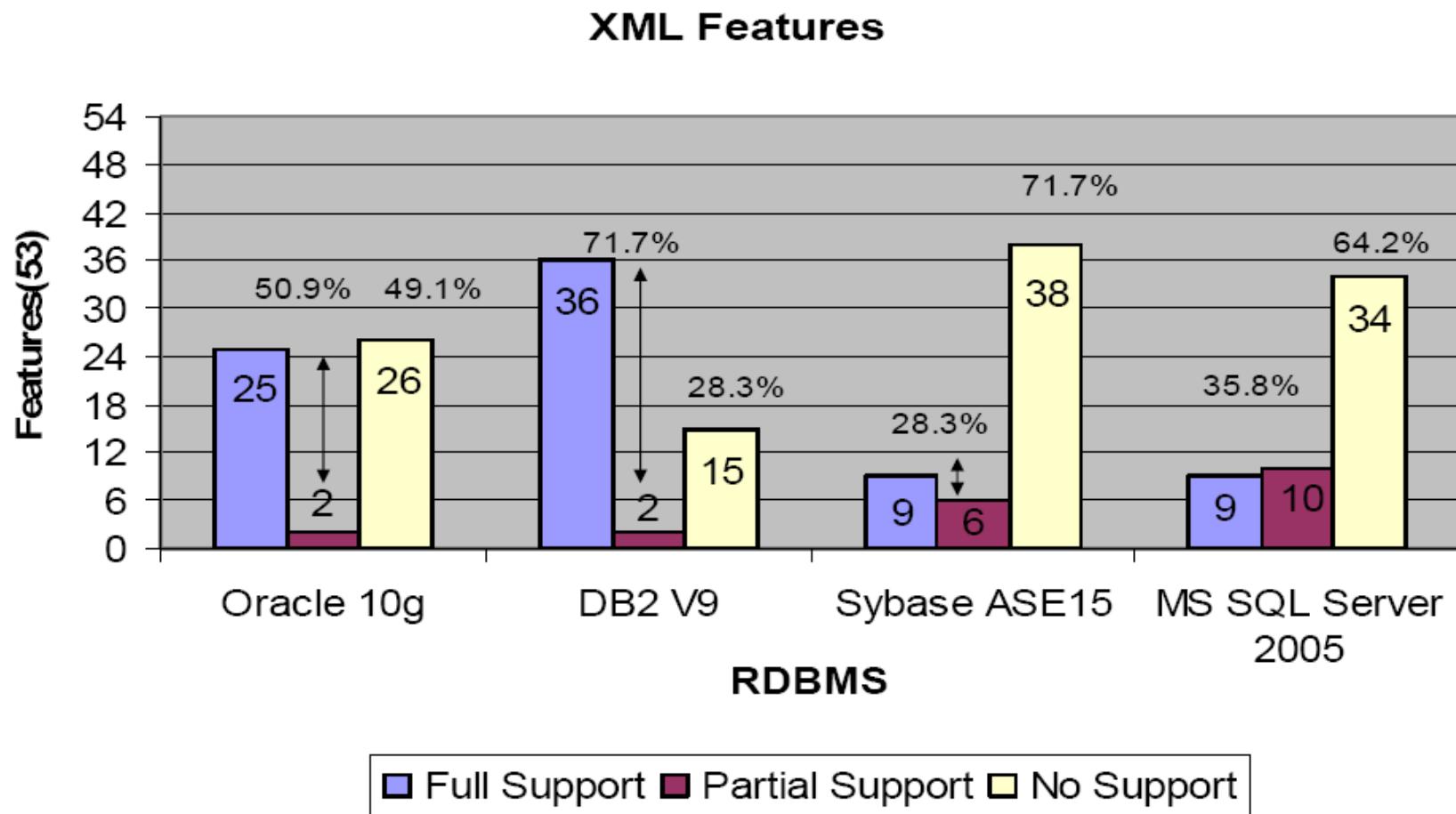
SQL:2003: SQL/XML

- XML-Speicherung
  - Basisdatentyp **XML**
  - Automatische Abbildung („Shredding“)
  
  
  
  
  
  
- XML-Erzeugung
  - XML-Funktionen zum Generieren von XML-Dokumenten aus relationalen Tabellen (z.B. **XMLElement**, **XMLAttributes**)
  - Einbettung von XQuery in SQL (**XMLQuery**- und **XMLTable**-Funktionen)



# Abfragetechniken

## SQL:2003: RDBS Konformität



# Abfragetechniken

## SQL:2008: SQL/XML-Erweiterungen

- Integration von XQuery-Datenmodell
  - XML-Datentyp basiert auf XQuery-Datenmodell
    - ◆ Beliebige Sequenzen
    - ◆ XML Schema und Validierungsunterstützung
    - ◆ ...
- Fortgeschrittene XQuery-Funktionen
  - **XMLQuery**-Funktion
    - ◆ Einbetten von XQuery in SQL
    - ◆ Erzeugen von XML
  - **XMLTable**-Funktion
    - ◆ Umwandeln von XQuery-Ergebnis in SQL-Tabelle

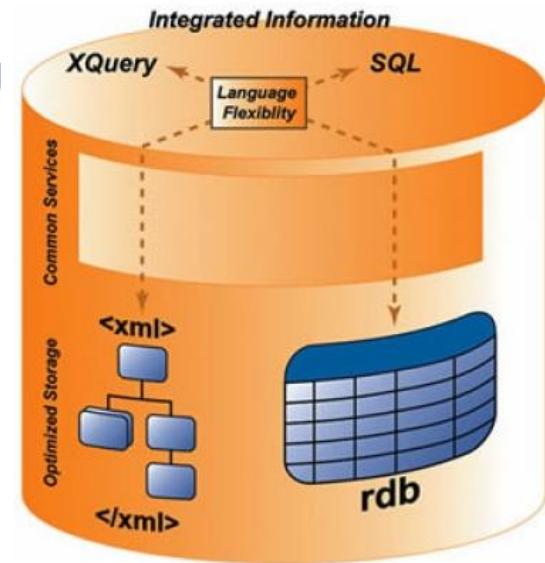


Figure „IBM DB2“ from an article of Holger Seubert

# Abfragetechniken

## SQL/XML-Standard: Vor- und Nachteile

- Vorteile
  - Unterstützung der SQL-Infrastruktur (z.B. Sichten, Trigger, PL/SQL)
  - Transaktionsunterstützung
  - Skalierbarkeit, Zugriffsschutz etc.
  - Globale Optimierung (XML und relational)
  - Standard durch Datenbanken unterstützt (z.B. Oracle, DB2, SQL Server)
- Nachteile
  - Kombination unterschiedlicher Abfragesprachen (SQL und XQuery)
  - XQuery nicht vollständig von DB-Engines unterstützt
    - ◆ kein XML Update wie in XQuery

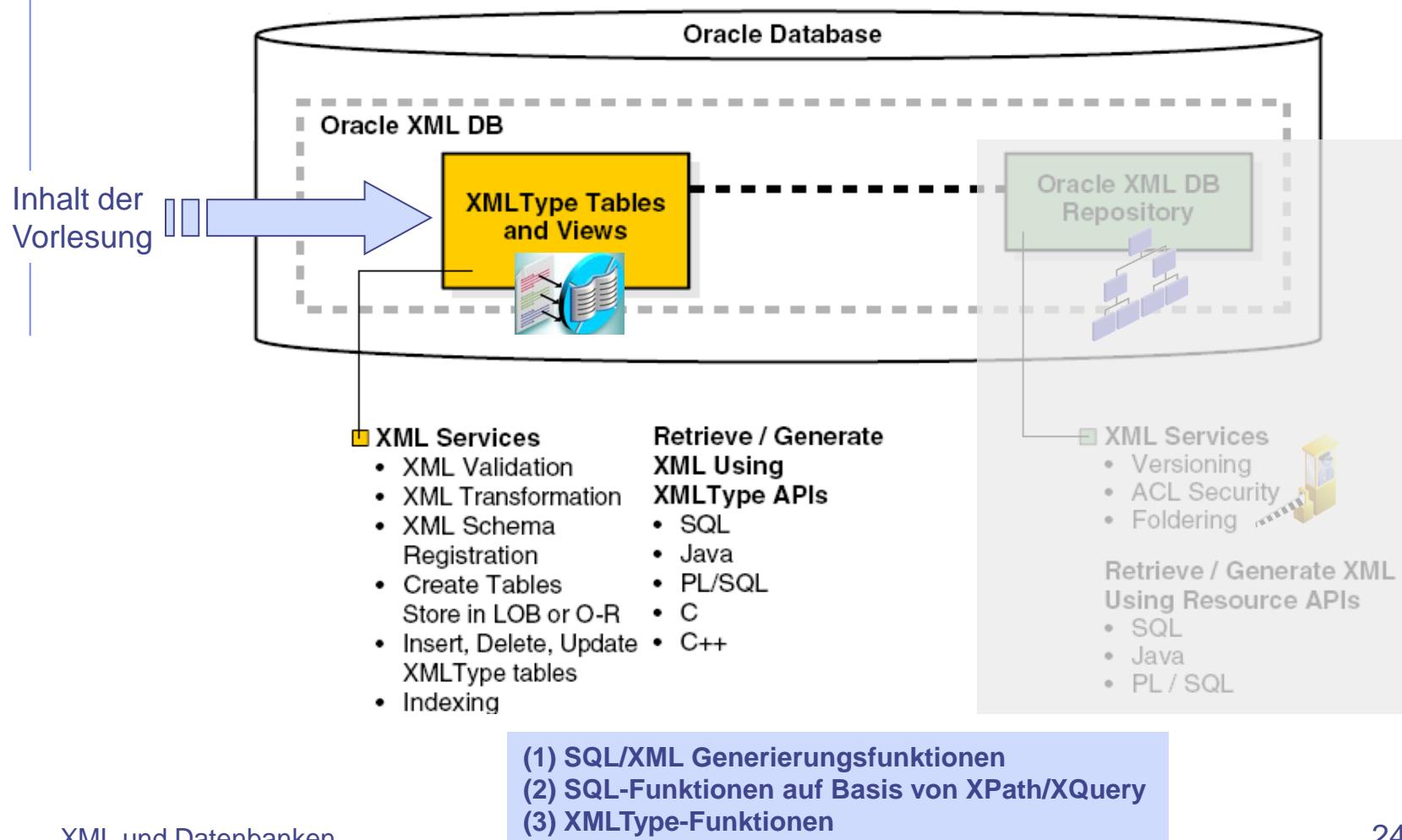
# Inhalt

---

- Motivation ✓
- Speichertechniken ✓
- Abfragetechniken ✓
- XML in Oracle
  - Speicheroptionen
  - Abfragen von XML-Daten
  - Manipulation von XML-Daten
  - XMLType-Sichten

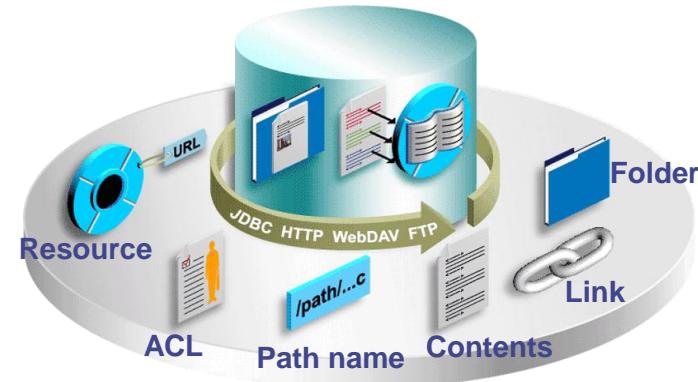
# XML in Oracle

## Speicheroptionen: XMLType-Tabellen/-Sichten vs. Repository



# XML in Oracle

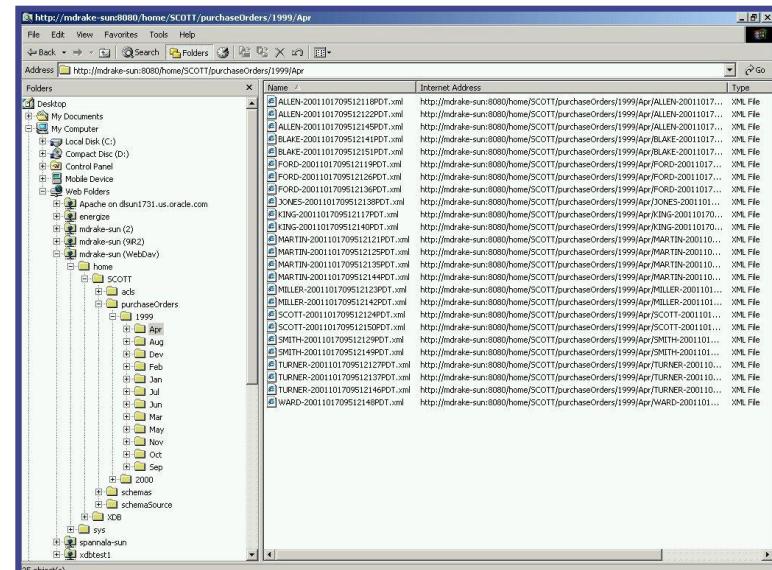
## Speicheroptionen: Repository



- Stellt Dateisystemfunktionalität bereit
- Bietet Pfadnamen für Ressourcen (XML, XSL, XML Schema, etc.), ähnlich wie bei Dateisystemen
- Hilfreich für inhaltsorientierte Anwendungen
- Abfrage und Manipulation mittels XPath
- Zugriff mit Internet-Protokolle (WebDAV, FTP, HTTP, etc.)
- Unterstützt Zugriffskontrolllisten (ACL) und Versionierung von Ressourcen
  - Check-In/Check-Out
  - Versionshistorie
  - Vorgänger/Nachfolger abrufen

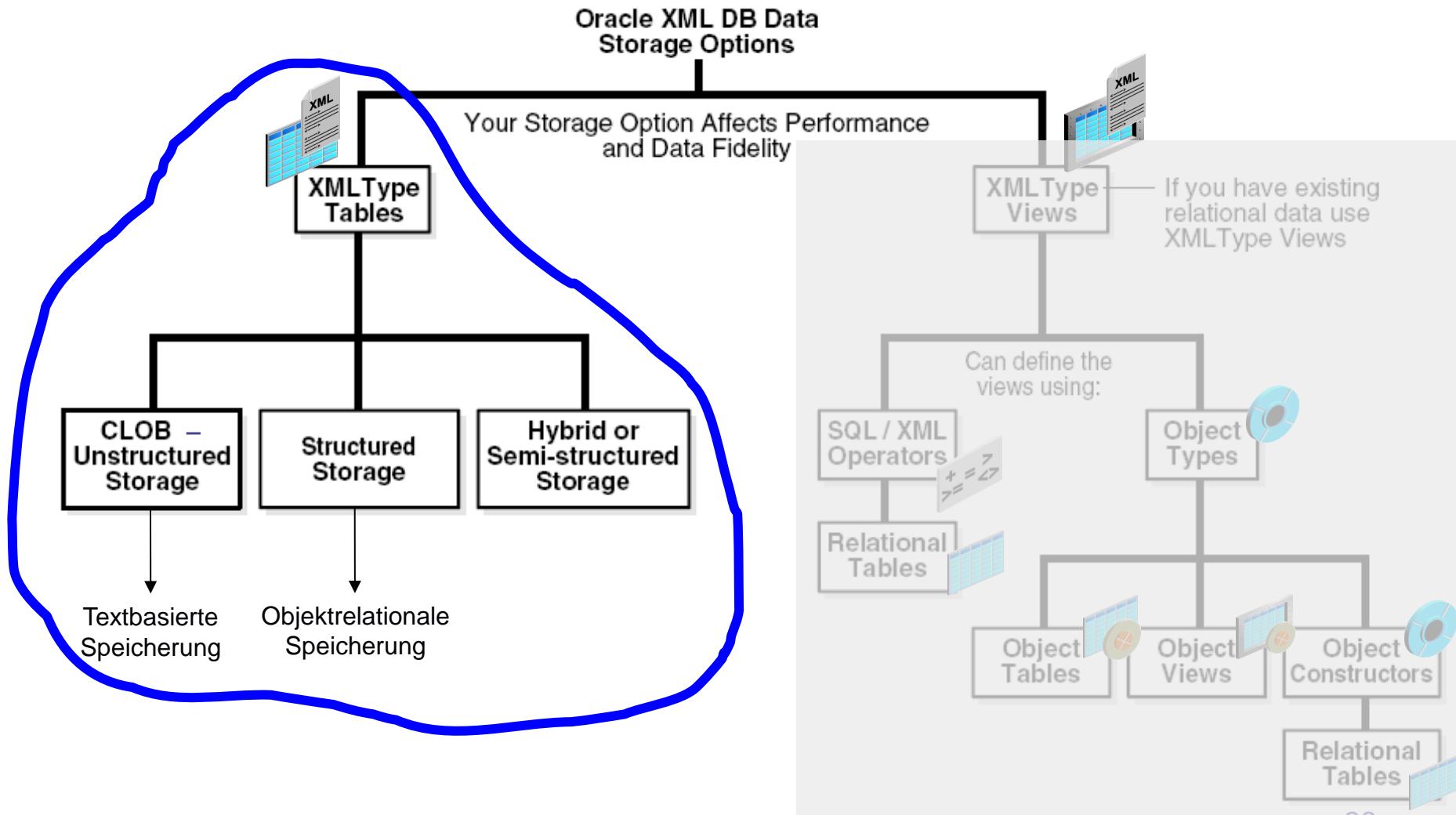


XML und Datenbanken Ressource



# XML in Oracle

## Speicheroptionen: XMLType-Tabellen und XMLType-Sichten



# XML in Oracle

## Speicheroptionen: Unstrukturierte vs. strukturierte Speicherung

Unstrukturierte (CLOB) Speicherung	Strukturierte Speicherung
Ermöglicht eine <b>einfachere Generierung</b>	Erfordert eine <b>komplexere Generierung</b>
Ist bei <b>Schema-Änderungen</b> flexibel	Bietet begrenzte Flexibilität
Bewahrt <b>Dokumenttreue</b> (Document Fidelity)	Bewahrt <b>DOM-Treue</b> (DOM Fidelity)
Unterstützt <b>Aktualisierungen des gesamten Dokuments</b>	Unterstützt <b>schrittweise Aktualisierungen</b>
Bietet bessere <b>Performance für das Speichern und Abrufen</b>	Bietet bessere <b>Performance für DML-Operationen</b>
Benötigt <b>mehr Speicherplatz</b> aufgrund von Tags und Leerstellen	Benötigt <b>weniger Speicherplatz</b> , da nur Daten gespeichert werden, keine Metadaten

# XML in Oracle

## Speicheroptionen: Eigenschaften von XMLType

- SQL2003-Datentyp für das Speichern von XML-Daten
  - XMLType ermöglicht der DB, XML zu verstehen!
- Verwendung zur Definition von
  - Tabellenspalten
  - Objekttypattribute
  - Objekttabellen
  - Sichten
  - PL/SQL-Variablen, -Parameter und -Rückgabewerte
- Funktionalität
  - **Konstruktoren** für die Erstellung von XMLType-Instanzen bei Einfügeoperationen
  - XPath-basierte **SQL-Built In-Funktionen** für die Abfrage und Manipulation von XML-Inhalten
  - SQL-Operationen für die Generierung von XML aus SQL-Inhalten
  - Referenzen auf registrierte **XML Schemata**
  - **Validierung** von XML-Inhalten
  - **XSL-Transformation** von XML-Inhalten

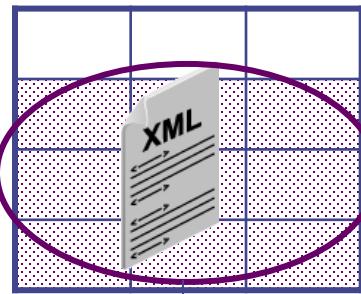
# XML in Oracle

## Speicheroptionen im Detail

①

### XMLType Table

```
CREATE TABLE xml_asTable
OF XMLType;
```

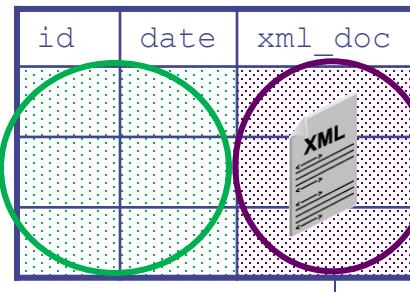


XML Document



### XMLType Column

```
CREATE TABLE xml_asColumn
(id NUMBER,
date DATE
xml_doc XMLType);
```



XML Document

XML document  
or fragments  
stored as CLOB

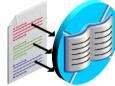
**XMLType is realised on basis of OR-concepts**

XML tags  
mapped to  
table columns

②

**Unstructured  
(CLOB)  
Storage**

OPTIONAL



Change of storage method  
possible using DB import/export

**Structured  
Storage**

MUST

**Structured  
Storage**

③

XML und Datenbanken

**Schema-based**

xsd Associating the XMLType table with an XML Schema

**Schema  
Registration**

④

# XML in Oracle

## Speicheroptionen: XML Schema registrieren 1/3

- PL\*SQL Package DBMS\_XMLSHEMA stellt Funktionalität für das
  - Registrieren und

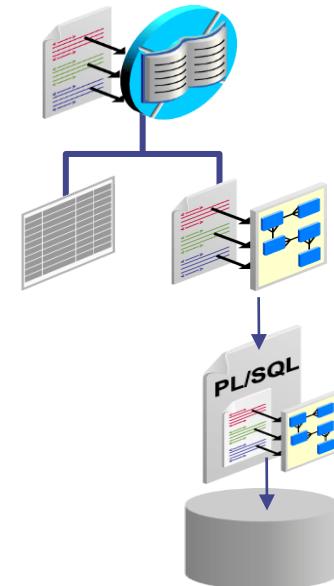
```
DBMS_XMLSHEMA.registerSchema (
 'http://localhost:8080/po.xsd' ,
 '/home/HR/xsd/po.xsd') ;
```

Oracle-interner eindeutiger Bezeichner für das zu registrierende Schema

Gültiges Schemadokument (Pfad zum Server-Verzeichnis oder direkt als Parameter)

- Löschen von Schemata zur Verfügung

```
DBMS_XMLSHEMA.deleteSchema (
 'http://localhost:8080/po.xsd' ,
 DBMS_XMLSHEMA.DELETE CASCADE FORCE
);
```



# XML in Oracle

## Speicheroptionen: XML Schema registrieren 2/3

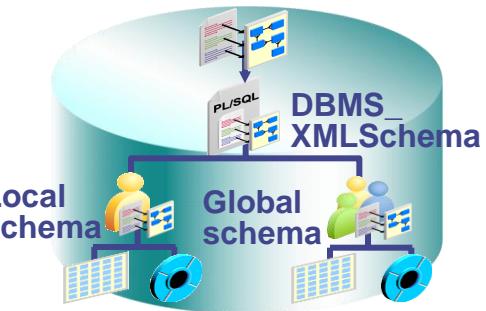
- XML Schema direkt als Parameter angeben

```
DBMS_XMLSCHEMA.registerSchema (
 'http://localhost:8080/po.xsd' ,
 '<xsd:schema version="1.0"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="region">
 <xsd:complexType>
 <xsd:sequence>
 <xsd:element name="region_id" type="xsd:int"/>
 <xsd:element name="region_name"
 type="xsd:string"/>
 </xsd:sequence>
 </xsd:complexType>
 </xsd:element>
 </xsd:schema>') ;
```

# XML in Oracle

## Speicheroptionen: XML Schema registrieren 3/3

- Standardaktionen beim Registrieren
  - parsen / teilweise validieren und speichern des XML Schema
  - erstellen von Typen und Tabellen sowie aktualisieren des Data Dictionary
    - DD-View: USER\_XML\_SCHEMAS
- Local
  - Local: Nur für den Eigentümer sichtbar (Default=TRUE)
  - Global: Für alle DB-Benutzer sichtbar und verwendbar
- GenTypes
  - Generiert Objekttypen auf Basis von complexTypes im Schema (Default=TRUE)
    - DD-View: USER\_TYPES
- GenBeans
  - Generiert Java Beans (Default = FALSE)
- GenTables
  - Generiert eine XMLType-Tabelle für jedes globale Element im Schema (Default=TRUE)
    - DD-View: USER\_XML\_TABLES
- XML Schema Annotationen für eine benutzerdefinierte Abbildung!



# XMLType-Tabellen

## Speicheroptionen: Schema Annotationen – Beispiel

```
<xs:schema targetNamespace="http://xmlns.oracle.com/xdb/documentation/purchaseOrder"
 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xdb="http://xmlns.oracle.com/xdb"
 xmlns:po="http://xmlns.oracle.com/xdb/documentation/purchaseOrder" version="1.0"
 xdb:storeVarrayAsTable="true">
<xs:element name="PurchaseOrder" type="po:PurchaseOrderType<,
 xdb:defaultTable="PURCHASEORDER"/>
<xs:complexType name="PurchaseOrderType"
 xdb:SQLType="PURCHASEORDER_T"xdb:SQLName="REFERENCE"/>
 <xs:element name="Actions" type="po:ActionsType"
 xdb:SQLName="ACTION_COLLECTION"/>
 <xs:element name="Reject" type="po:RejectionType" minOccurs="0"/>
 <xs:element name="Requestor" type="po:RequestorType"/>
 <xs:element name="User" type="po:UserType"
 xdb:SQLName="EMAIL"/>
 <xs:element name="CostCenter" type="po:CostCenterType"/>
 <xs:element name="ShippingInstructions" type="po:ShippingInstructionsType"/>
 <xs:element name="SpecialInstructions" type="po:SpecialInstructionsType"/>
 <xs:element name="LineItems" type="po:LineItemsType<,
 xdb:SQLName="LINEITEM_COLLECTION"/>
 <xs:element name="Notes" type="po:NotesType" xdb:SQLType="CLOB"/>
 </xs:sequence>
</xs:complexType>
...

```

# XML in Oracle

## Speicheroption I: Strukturierte und Schema-basierte Speicherung

- XMLType-Tabelle

```
CREATE TABLE purchaseorder_as_table OF XMLType
XMLSCHEMA "http://localhost:8080/po.xsd"
ELEMENT "PurchaseOrder";
```

Wurzelement muss  
deklariert werden!

Oracle-interner eindeutiger  
Bezeichner für das  
registrierte Schema

- XMLType-Spalte

```
CREATE TABLE purchaseorder_as_column
 (id NUMBER, xml_doc XMLType)
XMLTYPE COLUMN xml_doc
ELEMENT "http://localhost:8080/po.xsd#PurchaseOrder";
```

# XML in Oracle

## Speicheroption II: Unstrukturierte und Schema-basierte Speicherung

- Eigenschaften der CLOB-Speicherung werden mit der STORE AS-Klausel angegeben
- XMLType-Tabelle:

```
CREATE TABLE purchaseorder_as_table OF XMLType
 XMLTYPE STORE AS CLOB
 XMLSCHEMA "http://localhost:8080/po.xsd"
 ELEMENT "PurchaseOrder";
```

- XMLType-Spalte:

```
CREATE TABLE purchaseorder_as_column
 (id NUMBER,
 xml_doc XMLType)
 XMLTYPE COLUMN xml_doc STORE AS CLOB
 XMLSCHEMA "http://localhost:8080/po.xsd"
 ELEMENT "PurchaseOrder";
```

# XML in Oracle

Speicheroption II: **Unstrukturierte** und nicht Schema-basierte Speicherung

- STORE AS CLOB-Klausel ist optional
- XMLType-Tabelle (ohne CLOB-Klausel):

```
CREATE TABLE purchaseorder_as_table OF XMLType;
```

- XMLType-Spalte:

```
CREATE TABLE purchaseorder_as_column
 (id NUMBER,
 xml_doc XMLType)
XMLType COLUMN xml_doc STORE AS CLOB;
```

# XML in Oracle

## Speicheroptionen: Laden von XML-Daten 1/3

- 3 Möglichkeiten
  - Mit einer `INSERT`-Anweisung und den `XMLType()`-Konstruktoren, indem man `VARCHAR2`, `CLOB`, `BLOB` oder `BFILE` verwendet
  - Oracle Export/Import und `SQL*Loader`
  - Oracle XML/SQL Utility für Java, FTP und WebDAV
- Beispiel: Nicht Schema-basierte Speicherung mit `BFILE`

```
CREATE DIRECTORY XML_FILES AS 'd:\user\haslinger\data';
INSERT INTO purchaseorder_as_table VALUES (
 XMLType(BFILENAME('XML_FILES', 'purchaseorder.xml'),
 NLS_CHARSET_ID('AL32UTF8')));
```

- Im Fall einer Schema-basierten Speicherung
  - XML-Daten müssen explizit mit XML Schema assoziiert werden
  - XML Schema kann wie folgt registriert werden ...

# XML in Oracle

## Speicheroptionen: Laden von XML-Daten 2/3

- (1) CreateSchemaBasedXML () -Methode
  - XML Schema wird als Parameter übergeben
  - Vorteil:
    - ◆ XML-Dokument benötigt keine Information über sein Schema

```
INSERT INTO purchaseorder_as_table VALUES(
 XMLType ('<PURCHASEORDER>
 <REQUESTOR>Steve Jones</REQUESTOR>
 <COSTCENTER>S30</COSTCENTER>
 ...
 </PURCHASEORDER>').CreateSchemaBasedXML(
 'http://localhost:8080/po.xsd'));
```

Oracle-interner eindeutiger  
Bezeichner für das  
registrierte Schema

# XML in Oracle

## Speicheroptionen: Laden von XML-Daten 3/3

- (2) Angabe der Schema-Information im XML-Dokument
  - Im Wurzelement mit XMLSchema-instance-Mechanismus
- Beispiel: Schema ohne Zielnamensraum (`targetNamespace`)

```
INSERT INTO purchaseOrderTab VALUES (
 XMLType ('<PURCHASEORDER
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation=
 "http://localhost:8080/po.xsd">
 <REQUESTOR>Steve Jones</REQUESTOR>
 <COSTCENTER>S30</COSTCENTER>
 ...
 </PURCHASEORDER>'));
```

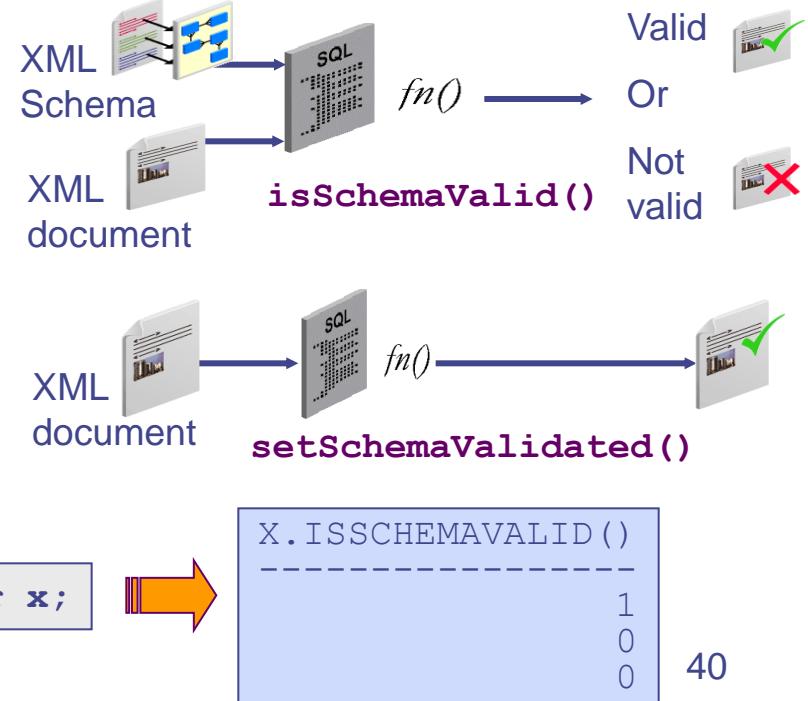
- Beispiel: Schema mit Zielnamensraum

```
xsi:schemaLocation= "http://www.example.com/po
 http://localhost:8080/po.xsd"
```

# XML in Oracle

## Speicheroptionen: Validierung von XMLType-Instanzen

- Wenn eine XMLType-Instanz erstellt, aktualisiert oder abgefragt wird, dann
  - führt Oracle eine Prüfung auf Wohlgeformtheit und
  - eine grundsätzliche Validierungsprüfung, z.B. ob alle Elemente und Attribute vorhanden sind, durch.
- Vollständige Validierung mit vordefinierten Funktionen/ Prozeduren, z.B. in Triggern
- Funktionen
  - XMLisValid()
  - isSchemaValid()
  - isSchemaValidated()
- Prozeduren
  - setSchemaValidated()
  - schemaValidate()



```
SELECT x.isSchemaValid() FROM customer x;
```

# Inhalt

---

- Motivation ✓
- Speichertechniken ✓
- Abfragetechniken ✓
- XML in Oracle
  - Speicheroptionen ✓
  - Abfragen von XML-Daten
  - Manipulation von XML-Daten
  - XMLType-Sichten

# XML in Oracle

## Abfragen von XML-Daten

### (1) Standard-SQL

- XML → XML (1:1): Rückgabe als XMLType
- `XMLType`-Spalten selektieren

### (2) XMLType-Funktionen

- XML → XML (1:1) – Typkonvertierung (CLOB, String, etc.)
- `getClobVal()`, `getStringVal()` etc.

### (3) SQL/XML-Generierungsfunktionen

- Relational → XML

Abfrage

### (4) XQuery

- XML → XML (Transformation möglich!)
  - ◆ `XQuery`
- XML → Relational
  - ◆ `XMLTable`

# XML in Oracle

## (1) Abfragen von XML-Daten: Standard-SQL

- XMLType-Spalte

```
SELECT resume
FROM emp_resumes
WHERE employee_id
= 100;
```

```
CREATE TABLE emp_resumes (
 employee_id NUMBER(6)
 PRIMARY KEY,
 resume XMLType);
```

```
RESUME

<?xml version="1.0"?>
<RESUME>
 <FULL_NAME>Steven King</FULL_NAME>
 <JOB_HISTORY>
 <JOB_ID>AD_PRES</JOB_ID>
 </JOB_HISTORY>
</RESUME>
```

- XMLType-Tabelle

```
SELECT object_value
FROM purchaseOrder_as_Table;
```

```
OBJECT_VALUE

<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://localhost:8080/source/schemas
...>
```

# XML in Oracle

## (2) Abfragen von XML-Daten: XMLType-Funktionen

- Können verwendet werden, wenn es sich bei einem Spalten- oder Funktionsrückgabewert um eine XMLType-Instanz handelt.

XMLType-Funktionen	Beschreibung
<code>getBlobVal()</code>	Liefert die XMLType-Inhalte als CLOB-Wert zurück
<code>getStringVal()</code>	Liefert den XMLType-Wert als Zeichenkette zurück
<code>getNumberVal()</code>	Liefert den XMLType-Knoten als num. Wert zurück
<code>isFragment()</code>	Liefert den Wert 1 zurück, wenn das Dokument ein Fragment ist. Gibt andernfalls 0 zurück (wenn die XML-Deklaration vorhanden ist).

```
SELECT e.resume.getBlobVal()
FROM emp_resumes e
WHERE employee_id = 100;
```

Korrelationsvariable

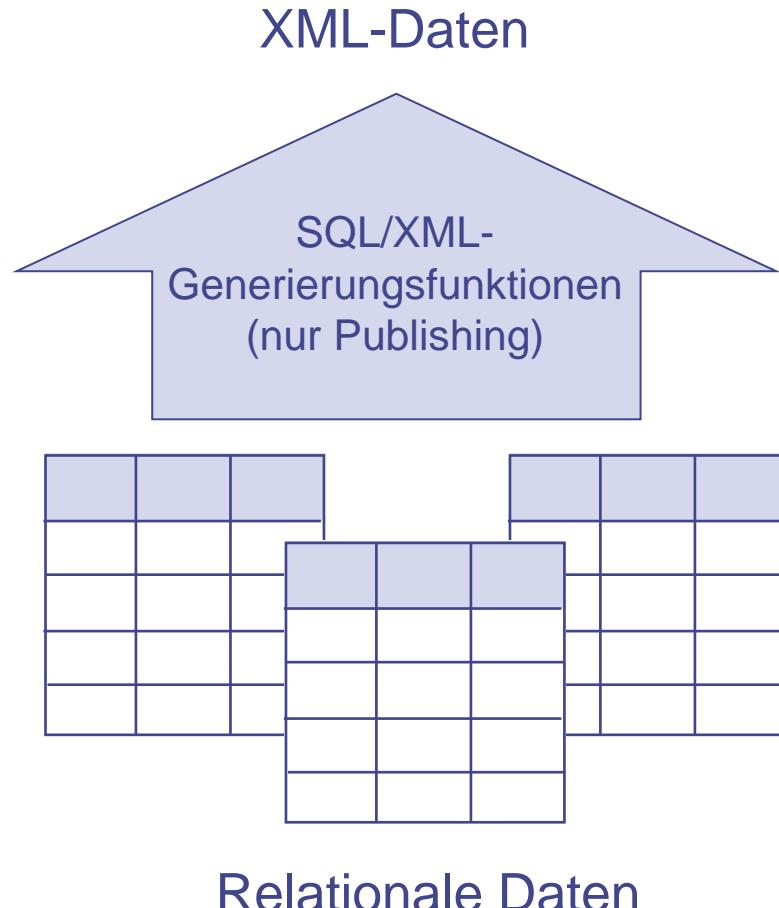


```
RESUME

<?xml version="1.0"?>
<RESUME>
 <FULL_NAME>Steven King</FULL_NAME>
 <JOB_HISTORY>
 <JOB_ID>AD_PRES</JOB_ID>
 </JOB_HISTORY>
</RESUME>
```

# XML in Oracle

## (3) Abfragen von XML-Daten: SQL/XML-Generierungsfunktionen



# XML in Oracle

## SQL/XML-Generierungsfunktion: **XMLElement()**

### ■ Syntax:

```
XMLElement(
 [NAME] identifier
 {, XML_attributes_clause}
 {, value_expression})
```

Name des zu erstellenden XML-Elements  
(doppelte Anführungszeichen behält Groß/Kleinschreibung bei)

**XMLAttributes()** ... nächste Folie!

Optionen für die Definition des Elementinhalts:

- Spaltenname
- Spalten-Ausdruck
- Instanz eines Objekttyps
- Literalwert
- SQL/XML-Generierungsfunktion
- SQL-Subqueries

} Alias für Element-name notwendig!

Liefert eine XMLType-Instanz zurück

### ■ Beispiel:

```
SELECT e.employee_id,
 XMLElement("Emp",
 e.first_name || ' ' || e.last_name) AS result
 FROM employees e
 WHERE employee_id > 200;
```

EMPLOYEES		
EMPLOYEE_ID	FIRST_NAME	LAST_NAME
EMPLOYEE_ID RESULT		
-----		
201	<Emp>Michael Hartstein</Emp>	
202	<Emp>Pat Fay</Emp>	
203	<Emp>Susan Mavris</Emp>	

# XML in Oracle

SQL/XML-Generierungsfunktion: **XMLElement ()** Alias – doppelte Anführungszeichen

- Verwenden Sie im zweiten Argument der Funktion XMLElement () die Funktion XMLAttributes () mit

- einem Spaltennamen:

```
SELECT XMLElement("Emp",
 XMLAttributes(employee_id,
 first_name||' '||last_name) as result
 FROM employees e
 WHERE employee_id = 201;
```

RESULT  
-----  
<Emp EMPLOYEE\_ID="201">Michael Hart</Emp>

- einem Literalwert oder einem Spaltenausdruck mit Aliasnamen:

```
SELECT XMLElement("Emp",
 XMLAttributes(employee_id AS "id", '10' AS "dept",
 first_name||' '||last_name) as result
 FROM employees e
 WHERE department_id = 10;
```

RESULT  
-----  
<Emp id="200" dept="10">Jennifer Whalen</Emp>

implizit definiert  
durch Spalten-  
namen

explizit definiert  
durch Alias

# XML in Oracle

SQL/XML-Generierungsfunktion: **XMLForest()**

- Syntax: **XMLForest(value\_expression [AS alias]  
[, value\_expression [AS alias]] ...)**
- Verkettet n Wertausdrücke (z.B. DB-Spalten), um daraus ein XML-Fragment zu generieren
  - Optionen für die Definition des Elementsinhalts wie bei `XMLElement()`
  - Elementname kann implizit (Spaltenname = Elementname) oder explizit definiert werden (wie Attributnamen bei `XMLAttributes()`)
- Anwendungsmöglichkeiten
  - standalone
  - in sich geschachtelt
  - innerhalb von `XMLElement()`, um Wurzel zu erzeugen
- Beispiel:

```
SELECT employee_id,
 XMLForest(first_name,last_name) result
 FROM employees WHERE department_id = 10;
```

```
EMPLOYEE_ID RESULT

200 <FIRST_NAME>Michael</FIRST_NAME>
 <LAST_NAME>Hart</LAST_NAME>

201 <FIRST_NAME>Pat</FIRST_NAME>
 <LAST_NAME>Fay</LAST_NAME>
```

# XML in Oracle

## **XMLForest()** vs. **XMLElement()** – Schachtelung

- Verwenden Sie verschachtelte XMLElement () -Funktionen:

```
SELECT employee_id, XMLElement("Emp",
 XMLElement("name",
 e.first_name||' '||e.last_name),
 XMLElement("hire_date", e.hire_date)) result
FROM employees e
WHERE employee_id > 200;
```

EMPLOYEE\_ID RESULT

```

201 <Emp><name>Michael Hart</name><hire_date>17-FEB-96</hire_date></Emp>
202 <Emp><name>Pat Fay</name><hire_date>17-AUG-97</hire_date></Emp>
203 <Emp><name>Susan Mavris</name><hire_date>07-JUN-94</hire_date></Emp>
```

- Verwenden Sie die Funktion XMLForest ():

```
SELECT employee_id, XMLElement("Emp",
 XMLForest(first_name||' '||last_name "name",
 hire_date AS "hire_date")) result
FROM employees
WHERE employee_id > 200;
```

# XML in Oracle

SQL/XML-Generierungsfunktion: **XMLConcat()**

- Verkettet n XML-Fragmente zu einem XML-Fragment
  - Pendant zu XMLForest für XML-Elemente
  - Falls ein Wert `NULL` ist, wird er vom Ergebnis ausgeschlossen
  - Falls alle Werte `NULL` sind, ist auch das Ergebnis `NULL`
- Syntax:

```
XMLConcat(
 {XMLType_instance | XMLSequenceType})
```

```
SELECT XMLConcat(
 XMLElement("NAME", first_name),
 XMLElement("EURO", 12*salary))
 AS salary
FROM employees;
```

SALARY

-----  
`<NAME>Joe</NAME><EURO>24000</EURO><NAME>Jim</NAME><EURO>42000</EURO>`

# XML in Oracle

SQL/XML-Generierungsfunktion: **XMLAgg()**

- Aggregatfunktion, die aus n Spalten von n Tupel ein XML-Fragment generiert
  - Verkettet XMLType-Instanzen über mehrere Zeilen
  - = Unterschied zu XMLConcat()
- Syntax:  

```
XMLAgg (XMLType_instance
[ORDER BY sort_list])
```

- Beispiel:

```
SELECT XMLElement(department,
 XMLAgg (XMLElement(employee,
 XMLForest(last_name, salary)))) result
FROM employees WHERE department_id = 20;
```

1 Tupel!

RESULT

```
<DEPARTMENT>
 <EMPLOYEE>
 <LAST_NAME>Hartstein</LAST_NAME>
 <SALARY>13000</SALARY>
 </EMPLOYEE>
 <EMPLOYEE>
 <LAST_NAME>Fay</LAST_NAME>
 <SALARY>6000</SALARY>
 </EMPLOYEE>
</DEPARTMENT>
```

# XML in Oracle

SQL/XML-Generierungsfunktion: Master-Detail-Inhalte generieren 1/2

- Beispiel:

```
SELECT department_id, XMLElement("department",
 XMLAttributes(e.department_id "dept_id"),
 XMLAgg(XMLElement("emp", e.last_name)))
) result
FROM employees e
WHERE department_id in (10, 20)
GROUP BY department_id;
```

Detail  
Master

Mehrere Tupel!

DEPARTMENT_ID	RESULT
10	<department dept_id="10"> <emp>Whalen</emp> </department>
20	<department dept_id="20"> <emp>Hartstein</emp> <emp>Fay</emp> </department>

# XML in Oracle

## SQL/XML-Generierungsfunktion: Master-Detail-Inhalte generieren 2/2

- Beispiel:

```
DEPARTMENT_ID RESULT

10 <Department name="Administration">
 <emp NAME="Whalen">
 <jobs job="AD_ASST"></jobs>
 <jobs job="AC_ACCOUNT"></jobs>
 </emp>
</Department>
```

```
SELECT department_id, XMLElement("Department",
 XMLAttributes(d.department_name "name"),
 (SELECT XMLAgg(XMLElement("emp",
 XMLAttributes(e.last_name name),
 (SELECT XMLAgg(XMLElement("jobs",
 XMLAttributes(j.job_id "job"))))
 FROM job_history j
 WHERE j.employee_id=e.employee_id)))
 FROM employees e
 WHERE e.department_id=d.department_id)
 AS result
FROM departments d
WHERE department_id < 40;
```

# XML in Oracle

SQL/XML-Generierungsfunktion: **XMLSequence()** 1/2

- Inverse Funktion zu XMLConcat()
- "Shreddert" Eingabe in mehrere Zeilen
- Beispiel:

```
SELECT e.*
FROM TABLE(XMLSequence(
 CURSOR(SELECT first_name,
 last_name
 FROM employees
 WHERE department_id = 20))) e;
```

COLUMN\_VALUE

```
<ROW>
 <FIRST_NAME>Michael</FIRST_NAME>
 <LAST_NAME>Hart</LAST_NAME>
</ROW>
<ROW>
 <FIRST_NAME>Pat</FIRST_NAME>
 <LAST_NAME>Fay</LAST_NAME>
</ROW>
```

```
SELECT e.*
FROM TABLE(XMLSequence(
 CURSOR(SELECT first_name,
 last_name
 FROM employees
 WHERE department_id = 20),
 XMLFormat('EMPLOYEE')))) e;
```

COLUMN\_VALUE

```
<EMPLOYEE>
 <FIRST_NAME>Michael</FIRST_NAME>
 <LAST_NAME>Hart</LAST_NAME>
</EMPLOYEE>
<EMPLOYEE>
 ...
```

# XML in Oracle

## SQL/XML-Generierungsfunktion: **XMLSequence()** 2/2

- Syntax:

```
XMLSequence(
 {XMLType_instance | SYS_REF_CURSOR_instance}
 [, XMLFormat_instance])
```

**XMLFormat()**

**CURSOR**-statement

- Zwei Formen

- Zerlegt Ergebnis aus beliebiger SQL-Anfrage in mehrere Zeilen
  - ◆ Input: **SYS\_REF\_CURSOR**-Argument mit optionalen **XMLFormat**-Objekt.
  - ◆ Output: **XMLType**-Instanz für jede Zeile des Cursor
- Zerlegt XML-Fragmente in mehrere Zeilen
  - ◆ Input: **XMLType**-Instanz
  - ◆ Output: **VARRAY** von Knoten der obersten Ebene
- Liefert einen **XMLSequenceType** = **VARRAY of XMLType-Instanzen** (**TABLE**-Funktion notwendig)

# XML in Oracle:

SQL/XML-Generierungsfunktion: **XMLColAttVal()**

- Ähnlich zu XMLAttributes(), aber
  - verwendet eine Standardabbildung von DB-Spalten
    - ◆ 1 XML-Element "column" für den DB-Spaltenwert
    - ◆ 1 XML-Attribut "name" für den DB-Spaltennamen
- Syntax: **XMLColAttVal(value\_expression [AS alias]  
[, value\_expression [AS alias]]...)**

```
SELECT XMLElement("Emp",
 XMLAttributes(
 e.first_name||' '||e.last_name AS "name") ,
 XMLColAttVal(e.hire_date, e.department_id
 AS "department"))
) result
FROM employees e
WHERE e.department_id = 20;
```

RESULT

```
<Emp name="Michael Hartstein">
 <column name = "HIRE_DATE">17-FEB 96</column>
 <column name = "department">20</column>
</Emp>
<Emp name="Pat Fay">
 <column name = "HIRE_DATE">17-AUG-97</column>
 <column name = "department">20</column>
</Emp>
```

# XML in Oracle

SQL/XML-Generierungsfunktion: **SYS\_XMLGEN()**

- Ähnlich zu XMLElement(), aber
  - akzeptiert nur ein einzelnes Argument
    - ◆ Skalaren Wert, Objekttyp oder XMLType-Instanz zusammen mit einem optionalen XMLFormat()-Objekt
  - liefert ein wohlgeformtes XML-Dokument mit einem Prolog zurück
- Beispiel:

```
SELECT SYS_XMLGEN(last_name) result
FROM employees WHERE last_name LIKE 'K%';
```

```
SELECT SYS_XMLGEN(XMLForest(last_name, salary),
 XMLFormat('EMPLOYEE')) result
FROM employees WHERE department_id = 30;
```

RESULT

```
<?xml version="1.0"?>
<LAST_NAME>Kaufling</LAST_NAME>
...

```

RESULT

```
<?xml version="1.0"?>
<EMPLOYEE>
<LAST_NAME>Raphaely</LAST_NAME>
<SALARY>11000</SALARY>
</EMPLOYEE>
...

```

# XML in Oracle

SQL/XML-Generierungsfunktion: **SYS\_XMLAGG()**

- Ähnlich zu XMLAgg(), aber
  - liefert ein wohlgeformtes XML-Dokument mit einem Prolog zurück
  - akzeptiert XMLFormat() -Objekt für die Definition des Wurzelelements
- Beispiel:

```
SELECT SYS_XMLAGG(SYS_XMLGEN(last_name),
 XMLFormat('EmployeeGroup')) result
 FROM employees
 WHERE department_id < 30
 GROUP BY department_id;
```

RESULT

```
<?xml version="1.0"?>
<EmployeeGroup>
 <LAST_NAME>Whalen</LAST_NAME>
</EmployeeGroup>

-----<?xml version="1.0"?>
<EmployeeGroup>
 <LAST_NAME>Hartstein</LAST_NAME>
 <LAST_NAME>Fay</LAST_NAME>
</EmployeeGroup>
```

# XML in Oracle

## (3) Abfragen von XML-Daten: SQL/XML-Generierungsfunktionen

	Eingabe	Ausgabe
<b>XMLElement()</b>	1 DB Attribut	1 XML Element
<b>XMLAttributes()</b>	1 DB Attribut	1 XML Attribut + zu einem Element hinzugefügt
<b>XMLForest()</b>	n DB Attribute	1 XML Fragment
<b>XMLConcat()</b>	n XML Elemente	1 XML Fragment
<b>XMLAgg()</b>	n DB Attribute von n Tupel	1 XML Fragment
<b>XMLSequence()</b>	n DB Attribute (SQL Anfrage) oder XML Elemente	1 XML Fragment (Wurzel "row") per Tupel oder 1 VARRAY von Top-Level XML Elementen
<b>XMLColAttrVal()</b>	1 DB Attribut	1 XML Element (mit Namen "column") + 1 XML Attribut (mit Namen "name")
<b>SYS_XMLGen()</b>	1 DB Attribut oder 1 XMLType Instanz	1 XML Dokument
<b>SYS_XMLAgg()</b>	n DB Attribute von n Tupel	1 XML Dokument (Wurzel "rowset")

# XML in Oracle

## Abfragen von XML-Daten

### (1) Standard-SQL

- XML → XML (1:1): Rückgabe als XMLType
- XMLType-Spalten selektieren

### (2) XMLType-Funktionen

- XML → XML (1:1) – Typkonvertierung (CLOB, String, etc.)
- getClobVal(), getStringVal() etc.

### (3) SQL/XML-Generierungsfunktionen

- Relational → XML

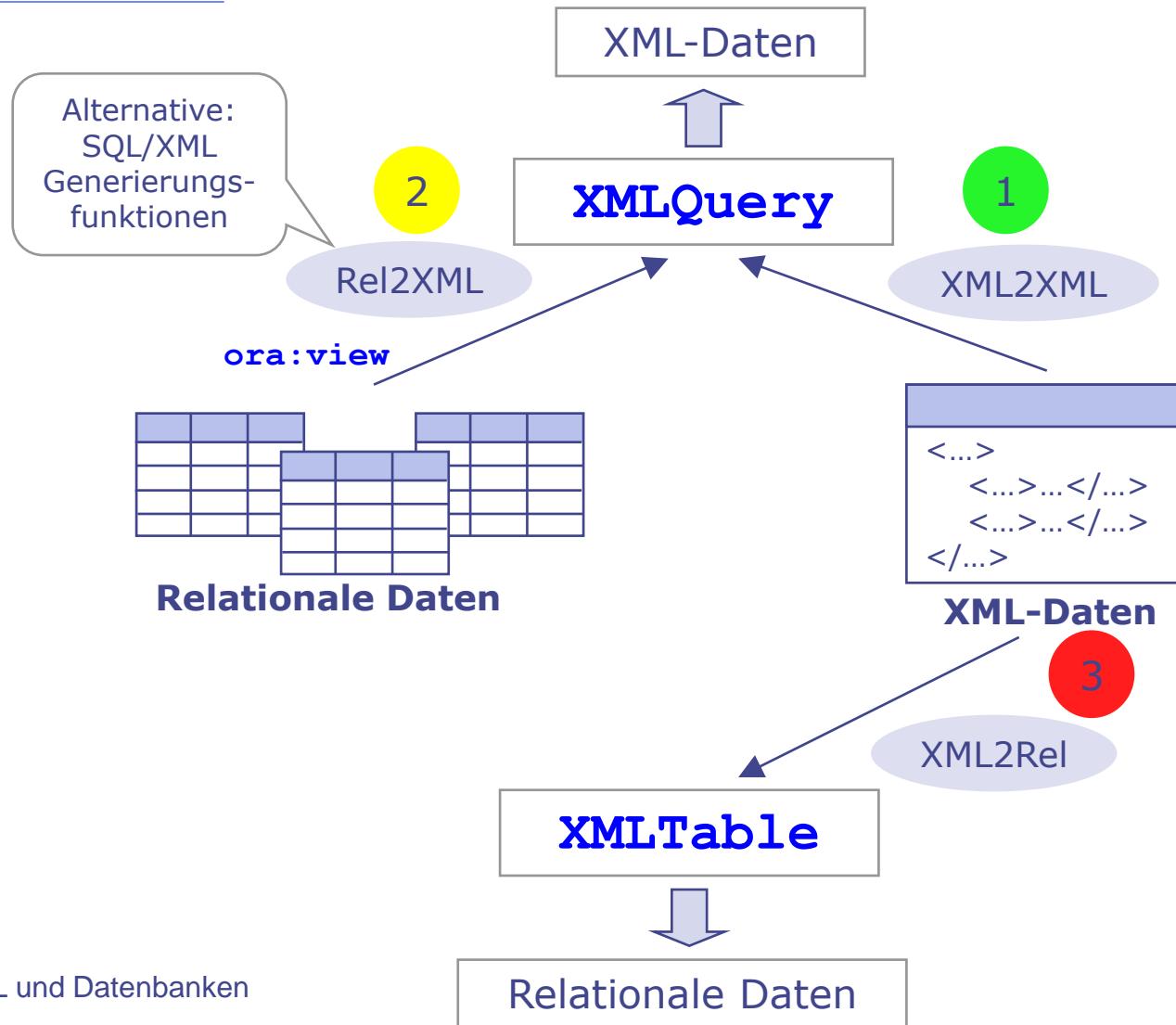
### (4) XQuery

- XML → XML (Transformation möglich!)
  - ◆ **XMLQuery**
- XML → Relational
  - ◆ **XMLTable**

Abfrage

# XML in Oracle

## Abfragen von XML-Daten: XQuery - Überblick



# XML in Oracle

## Abfragen von XML-Daten: XQuery – **XMLQuery**

- Erzeugen und abfragen von XML-Daten

```
XMLQuery(XQuery_string,
 [XML_passing_clause]
 RETURNING CONTENT
 [NULL ON EMPTY]
)
```

```
PASSING [BY VALUE]
 expr [AS identifier]
 [, expr [AS identifier]
] ...
```

# XML in Oracle

## Abfragen von XML-Daten: XQuery – **XMLQuery** XML2XML 1/2

1

```
<PurchaseOrder ...>
 <Reference>MILLER-20021009123336151PDT</Reference>
 <Requestor>JAMES MILLER</Requestor>
 <User>MILLER</User>
 <CostCenter>C10</CostCenter>
 <ShippingInstructions>
 <name>James Miller</name>
 <address>400 Oracle Parkway Redwood Shores CA 94065 USA </address>
 <telephone>650 506 7400</telephone>
 </ShippingInstructions>
 <SpecialInstructions>Air Mail</SpecialInstructions>
 <LineItems>
 <LineItem ItemNumber="1">
 <Description>A Night to Remember</Description>
 <Part Id="715515009058" UnitPrice="39.95" Quantity="2"/>
 </LineItem>
 <LineItem ItemNumber="2">
 <Description>The Unbearable Lightness Of Being</Description>
 <Part Id="37429140222" UnitPrice="29.95" Quantity="2"/>
 </LineItem>
 </LineItems>
 ...
</PurchaseOrder>
```

Selektiere alle Bestellungen  
von Kunde „Miller“

# XML in Oracle

Abfragen von XML-Daten: XQuery – **XMLQuery** XML2XML 2/2

- Beispiel: Abfrage der **XMLType**-Tabelle **purchaseorder**

1

```
SELECT XMLQuery(
 'for $i in /PurchaseOrder
 where $i/User eq "MILLER"
 return <po id="{$i/Reference}" />'
 PASSING OBJECT_VALUE
 RETURNING CONTENT) Result
FROM purchaseorder;
```



Result

```
<po id="MILLER-20021009123336151PDT"></po>
<po id="MILLER-20021009123336341PDT"></po>
<po id="MILLER-20021009123337173PDT"></po>
<po id="MILLER-20021009123335681PDT"></po>
<po id="MILLER-20021009123335470PDT"></po>
<po id="MILLER-20021009123336972PDT"></po>
```

# XML in Oracle

## Abfragen von XML-Daten: XQuery – **XMLQuery Rel2XML** 1/3

2

Countries

Country_Id	Country_Name	Region_Id
AU	Australia	3
...	...	...

Regions

Region_Id	Region_Name
3	Asia
...	...

ASIAN\_COUNTRIES

```
<!--
<ROW>
 <COUNTRY_ID>AU</COUNTRYID>
 <COUNTRY_NAME>Australia</COUNTRY_NAME>
 <REGION_ID>3</REGION_ID>
</ROW>
<ROW>
 <COUNTRY_ID>CN</COUNTRY_ID>
 <COUNTRY_NAME>China</COUNTRY_NAME>
 <REGION_ID>3</REGION_ID>
</ROW>
```

Selektiere alle Länder der Region Asia und gibt das Ergebnis als XML zurück

# XML in Oracle

## Abfragen von XML-Daten: XQuery – **XMLQuery** Rel2XML 2/3

2

```
SELECT XMLQuery(
 'for $i in ora:view("REGIONS") , $j in ora:view("COUNTRIES")
 where $i/ROW/REGION_ID = $j/ROW/REGION_ID
 and $i/ROW/REGION_NAME = "Asia"
 return $j'
 RETURNING CONTENT)
AS asian_countries FROM DUAL;
```

ora:view

SQL-Funktion zur Umwandlung  
relationaler Tabellen in XML



```
ASIAN_COUNTRIES

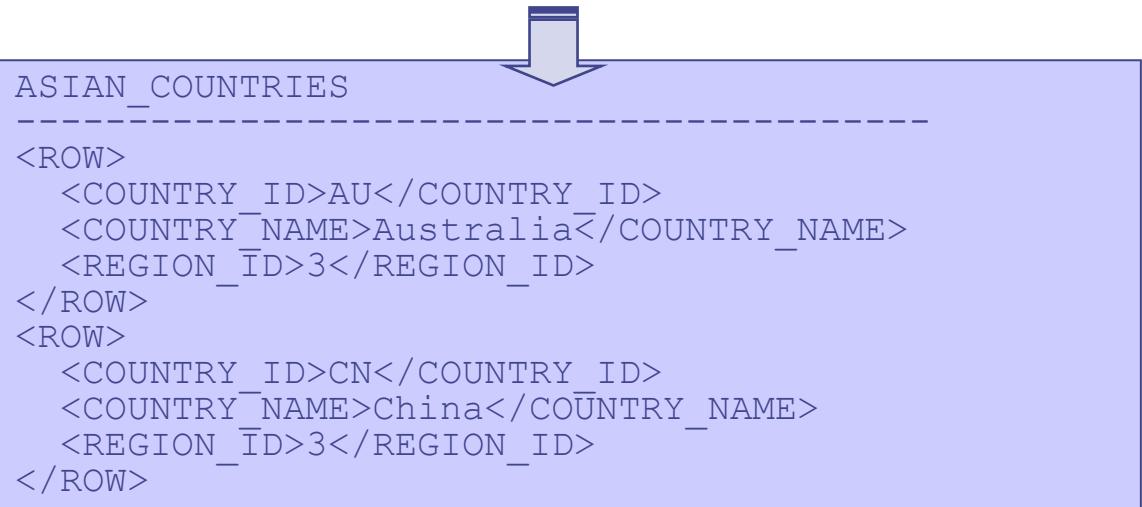
<ROW>
 <COUNTRY_ID>AU</COUNTRYID>
 <COUNTRY_NAME>Australia</COUNTRY_NAME>
 <REGION_ID>3</REGION_ID>
</ROW>
<ROW>
 <COUNTRY_ID>CN</COUNTRY_ID>
 <COUNTRY_NAME>China</COUNTRY_NAME>
 <REGION_ID>3</REGION_ID>
</ROW>
```

# XML in Oracle

## Abfragen von XML-Daten: Rel2XML 3/3

- Alternative: SQL/XML-Generierungsfunktionen

```
SELECT XMLElement("ROW",
 XMLElement("COUNTRY_ID", country_id),
 XMLElement("COUNTRY_NAME", country_name),
 XMLElement("REGION_ID", c.region_id)
) Asian_Countries
FROM REGIONS r, COUNTRIES c
WHERE r.region_id = c.region_id and
 r.region_name = 'Asia';
```



# XML in Oracle

## Abfragen von XML-Daten: XQuery – **XMLTable**

- Zerlegt („shredding“) das Ergebnis eines XQuery-Ausdrucks in Zeilen und Spalten einer neuen, virtuellen Tabelle

```
XMLTable([XML_namespace-clause,]
 XQuery_string
 XMLTABLE_options
)
```

```
XMLNAMESPACES ([string AS identifier]
 [[, string AS identifier]]
 [...]
 [DEFAULT string])
```

```
[XML_passing_clause] [COLUMNS XML_table_column
 [, XML_table_column]
 [...]
]
```

# XML in Oracle

## Abfragen von XML-Daten: XQuery – **XMLTable** XML2Rel 1/3

3

maxOccurs="1"

maxOccurs="unbounded"

```
<PurchaseOrder ...>
 <Reference>MILLER-20021009123336151PDT</Reference>
 <Requestor>JAMES MILLER</Requestor>
 <User>MILLER</User>
 <CostCenter>C10</CostCenter>
 <ShippingInstructions>
 <name>James Miller</name>
 <address>400 Oracle Parkway Redwood Sh...
 <telephone>650 506 7400</telephone>
 </ShippingInstructions>
 <SpecialInstructions>Air Mail</SpecialIn...
 <LineItems>
 <LineItem ItemNumber="1">
 <Description>A Night to Remember</Description>
 <Part Id="715515009058" UnitPrice="39.95" Quantity="2"/>
 </LineItem>
 <LineItem ItemNumber="2">
 <Description>The Unbearable Lightness Of Being</Description>
 <Part Id="37429140222" UnitPrice="29.95" Quantity="2"/>
 </LineItem>
 </LineItems>
 ...
</PurchaseOrder>
```

POREF	PRIORITY	CONTACT
SKING-20021009123336	Fastest	Steven A. King
SMCCAIN-200210091233	Regular	Samuel B. McCain
SMCCAIN-200210091233	Fastest	Samuel B. McCain
JCHEN-20021009123337	Fastest	John Z. Chen
JCHEN-20021009123337	Regular	John Z. Chen
SKING-20021009123337	Regular	Steven A. King
SMCCAIN-200210091233	Regular	Samuel B. McCain
JCHEN-20021009123338	Regular	John Z. Chen
SMCCAIN-200210091233	Regular	Samuel B. McCain
SKING-20021009123335	Regular	Steven X. King

Erzeuge relationale Daten aus XML-Daten

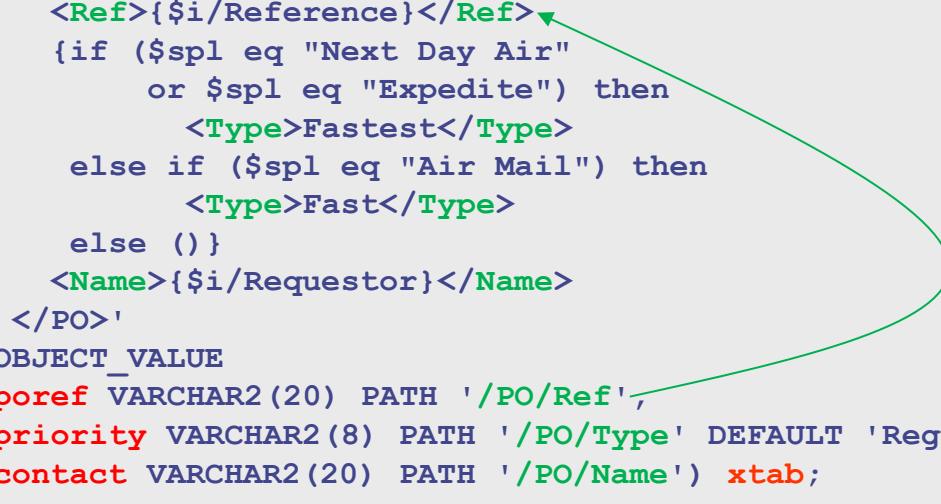
Air Mail = Fast  
Next Day Air = Expedite

# XML in Oracle

## Abfragen von XML-Daten: XQuery – **XMLTable** XML2Rel 2/3

3

```
SELECT xtab.poref, xtab.priority, xtab.contact
FROM purchaseorder,
 XMLTable('for $i in /PurchaseOrder
 let $spl := $i/SpecialInstructions
 where $i/CostCenter eq "A10"
 return <PO>
 <Ref>{$i/Reference}</Ref>' ◀
 {if ($spl eq "Next Day Air"
 or $spl eq "Expedite") then
 <Type>Fastest</Type>
 else if ($spl eq "Air Mail") then
 <Type>Fast</Type>
 else ())
 <Name>{$i/Requestor}</Name>
 </PO>')
PASSED OBJECT_VALUE
COLUMNS poref VARCHAR2(20) PATH '/PO/Ref',
 priority VARCHAR2(8) PATH '/PO/Type' DEFAULT 'Regular',
 contact VARCHAR2(20) PATH '/PO/Name') xtab;
```



POREF	PRIORITY	CONTACT
SKING-20021009123336	Fastest	Steven A. King
SMCCAIN-200210091233	Regular	Samuel B. McCain
SMCCAIN-200210091233	Fastest	Samuel B. McCain
JCHEN-20021009123337	Fastest	John Z. Chen
JCHEN-20021009123337	Regular	John Z. Chen
SKING-20021009123337	Regular	Steven A. King
SMCCAIN-200210091233	Regular	Samuel B. McCain
JCHEN-20021009123338	Regular	John Z. Chen
SMCCAIN-200210091233	Regular	Samuel B. McCain
SKING-20021009123335	Regular	Steven X. King

# Inhalt

---

- Motivation ✓
- Speichertechniken ✓
- Abfragetechniken ✓
- XML in Oracle
  - Speicheroptionen ✓
  - Abfragen von XML-Daten ✓
  - Manipulation von XML-Daten
  - XMLType-Sichten

# XML in Oracle

## Manipulation von XML-Daten

**(1) Aktualisieren** (SQL-UPDATE und `updateXML()`-Funktion)

**(2) Löschen** (SQL-DELETE)

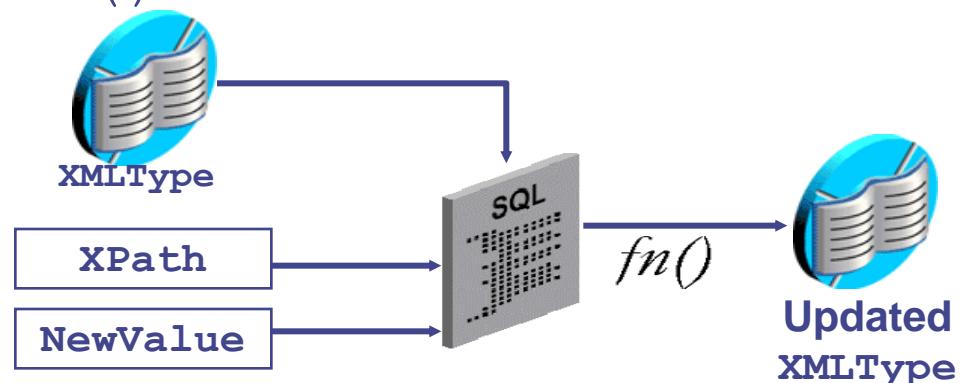
**(3) Transformieren** auf Basis von XSL-Stylesheets

Manipulation

# XML in Oracle

## Manipulation von XML-Daten: updateXML () 1/3

- XML-Dokumente, die in einer XMLType-Spalte oder -Tabelle in der Datenbank gespeichert sind, können mit der Funktion updateXML () aktualisiert werden.
  - Attributwert
  - Textknoten
  - Element
  - Knotenbaum
- Syntax:



```
updateXML (XMLType,
 XPath-expr1,
 value-expr1
 [, XPath-expr2, value-expr2,...])
```

- Bei strukturierter Speicherung → „piecewise update“
- Bei unstrukturierter Speicherung → gesamtes CLOB wird geschrieben

# XML in Oracle

## Manipulation von XML-Daten: updateXML() 2/3

- Beispiel: Aktualisieren des Textknotens last\_name
- XML-Daten:

```
SELECT xmldoc FROM xmldocuments WHERE id = 1;

<employees>
 <employee>
 <employee_id>100</employee_id>
 <first_name>Steve</first_name>
 <last_name>King</last_name>
 </employee> ...
</employees>
```

- SQL mit der Funktion updateXML():

```
UPDATE xmldocuments
SET xmldoc = updateXML(xmldoc,
 '/employees/employee/last_name/text()' , 'Kingsley')
WHERE existsNode(xmldoc,
 '/employees/employee[employee_id="100"]') = 1;
```

# XML in Oracle

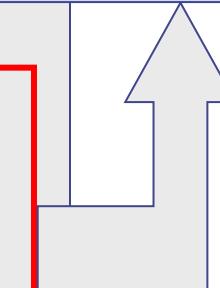
## Manipulation von XML-Daten: updateXML () 3/3

- Beispiel: Aktualisieren des Elementknotens `employee` im XML-Baum, der in der XMLType-Spalte gespeichert ist

```
UPDATE xmldocuments
SET xmldoc = updateXML(
 xmldoc,
 '/employees/employee[2]',
 XMLType ('<employee>
 <employee_id>102</employee_id>
 <first_name>Lex</first_name>
 <last_name>De Haan</last_name>
 <salary>17000</salary>
 </employee>'))
WHERE id = 1;
```

XMLEDOC

```
<?xml version="1.0"?>
<employees>
 <employee>
 <employee_id>100</employee_id>
 <first_name>Steve</first_name>
 <last_name>Kingsley</last_name>
 </employee>
 <employee>
 <employee_id>101</employee_id>
 <first_name>Neena</first_name>
 <last_name>Kingsley</last_name>
 </employee>
</employees>
```



# XML in Oracle

## Manipulation von XML-Daten: Löschen

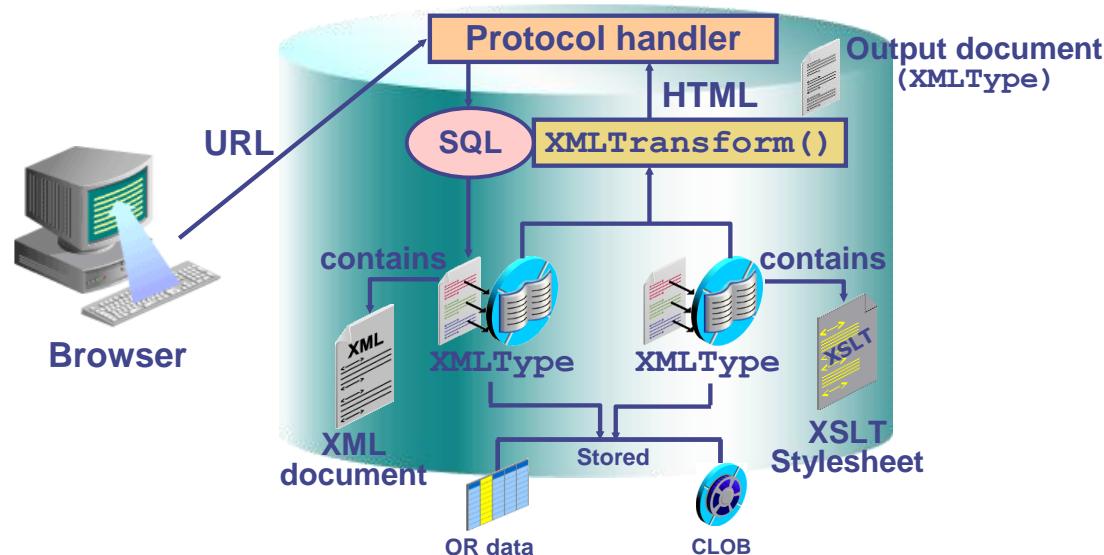
- Verwendung der `existsNode()`-Funktion in der WHERE-Klausel

```
DELETE FROM xmldocuments
WHERE existsNode(xmldoc,
 '//employee[last_name="Kingsley"]') = 1;
```

# XML in Oracle

## Manipulation von XML-Daten: Transformation 1/2

- In Oracle XML DB können XML-Daten, die in XMLType-Tabellen, -Spalten oder -Views gespeichert sind, mit Hilfe von XSL-Stylesheets transformiert werden  
→ bessere Performance!
  - XMLTransform (XMLType, XMLType)
  - XMLType.Transform (XMLType, XMLType)



# XML in Oracle

## Manipulation von XML-Daten: Transformation 2/2

- XSL-Stylesheet in XMLType-Spalte speichern:

```
CREATE TABLE stylesheets (
 id NUMBER(4), xsldoc XMLType);
INSERT INTO stylesheets
VALUES (1, XMLType('<xsl:stylesheet ...'));
```

- XML-Dokument in XMLType-Spalte speichern:

```
CREATE TABLE xmldocuments (
 id NUMBER(4), xmldoc XMLType);
INSERT INTO xmldocuments
VALUES (1, XMLType('<employees>...</employees>'));
```

- XMLType-Spalten für die Transformation verwenden:

```
SELECT XMLTransform(xmldoc,xsldoc) re
FROM xmldocuments d, stylesheets s
WHERE d.id = s.id
```

RE

```
<html>
<body>

Steve,King
Neena,Kocchar

</body>
</html>
```

# Inhalt

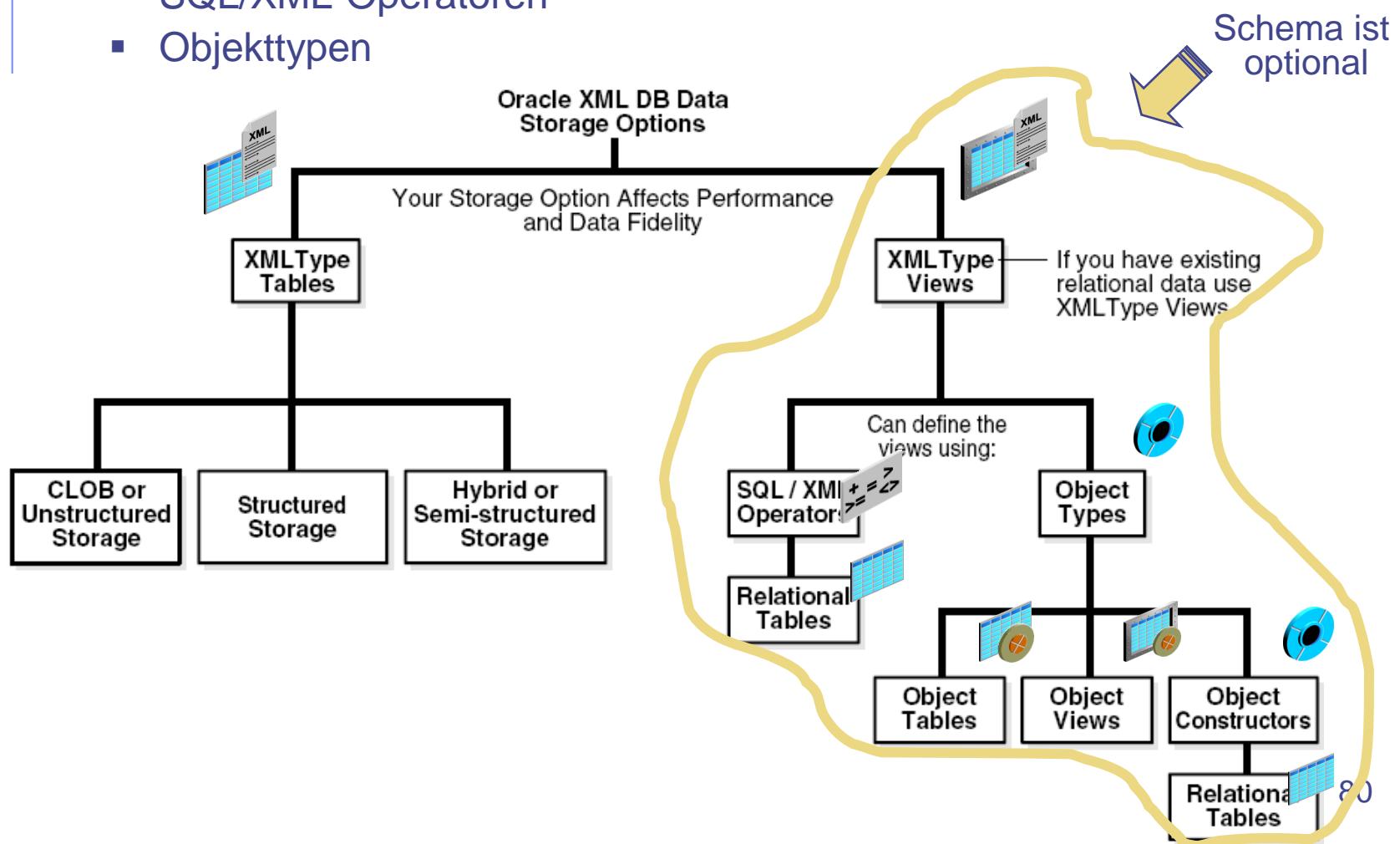
---

- Motivation ✓
- Speichertechniken ✓
- Abfragetechniken ✓
- XML in Oracle
  - Speicheroptionen ✓
  - Abfragen von XML-Daten ✓
  - Manipulation von XML-Daten ✓
  - XMLType-Sichten

# XML in Oracle

## XMLType-Sichten

- Erstellen von XMLType-Sichten mit Hilfe von:
  - SQL/XML-Operatoren
  - Objekttypen



# XML in Oracle

XMLType-Sichten: Nicht Schema-basiert vs. Schema-basiert

## ■ Nicht Schema-basiert

```
CREATE OR REPLACE VIEW view_name OF XMLType
WITH OBJECT ID {DEFAULT|(expression)}
AS sql_query;
```

- OBJECT\_ID wird mit  
`extract(OBJECT_VALUE, 'xpath-expr').getNumberVal()` eingestellt.

## ■ Schema-basiert

```
CREATE OR REPLACE VIEW view_name OF XMLType
XMLSCHEMA "url" ELEMENT "element-name"
WITH OBJECT ID {DEFAULT|(expression)}
AS sql_query;
```

# XML in Oracle

## XMLType-Sichten: Nicht Schema-basierte XMLType-Sicht mit SQL/XML

```
CREATE OR REPLACE VIEW emp_view OF XMLType
 WITH OBJECT ID (extract(
 OBJECT_VALUE , '/employee/@empno').getNumberVal())
AS SELECT XMLElement("employee",
 XMLAttributes(employee_id AS "empno"),
 XMLForest(e.last_name AS "last_name",
 e.hire_date AS "hiredate",
 e.salary as "salary")) AS "result"
FROM employees e
WHERE salary > 15000;
```

OBJECT\_VALUE

```
<employee empno="100"><last_name>King</last_name><
hiredate>17-JUN-87</hiredate><salary>24000</salary>
</employee>
```

```
SELECT * FROM emp_view;
```

```
<employee empno="101"><last_name>Kochhar</last_nam
e><hiredate>21-SEP-89</hiredate><salary>17000</sal
ary></employee>
```

```
<employee empno="102"><last_name>De Haan</last_nam
e><hiredate>13-JAN-93</hiredate><salary>17000</sal
ary></employee>
```

# XML in Oracle

XMLType-Sichten: Nicht Schema-basierte XMLType-Sicht mit Objekttypen und SYS\_XMLGEN()

- Objekttyp erstellen:

```
CREATE TYPE employee_t AS OBJECT (
 "@empno" NUMBER(6),
 name VARCHAR2(30),
 paycheck NUMBER(8,2));
/
```

```
SELECT * from empobj_view;
```

OBJECT\_VALUE  
-----  
<?xml version="1.0"?>  
<EMPLOYEE empno="100">  
 <NAME>King</NAME>  
 <PAYCHECK>24000</PAYCHECK>  
</EMPLOYEE>  
  
<?xml version="1.0"?>  
<EMPLOYEE empno="101">  
 <NAME>Kochhar</NAME>  
 <PAYCHECK>17000</PAYCHECK>  
</EMPLOYEE>  
...

- Sicht mit Hilfe von SYS\_XMLGEN() erstellen:

```
CREATE OR REPLACE VIEW
 empobj_view OF XMLType
 WITH OBJECT ID (extract(
 OBJECT_VALUE, '/employee/@empno').getNumberVal()))
AS SELECT SYS_XMLGEN(
 employee_t(e.employee_id, e.last_name, e.salary),
 XMLFormat('EMPLOYEE'))
FROM employees e
XM WHERE salary > 15000;
```

# XML in Oracle

XMLType-Sichten: Schema-basierte XMLType-Sicht mit SQL/XML – kein TargetNS

```
CREATE OR REPLACE VIEW region_xmlv OF XMLType
 XMLSCHEMA "region.xsd" ELEMENT "region"
 WITH OBJECT ID (extract(OBJECT_VALUE,
 '/region/region_id').getNumberVal())
AS SELECT XMLElement("region",
 XMLAttributes(
 'http://www.w3.org/2001/XMLSchema-instance' AS
 "xmlns:xsi",
 'region.xsd' AS
 "xsi:noNamespaceSchemaLocation"),
 XMLForest(region_id AS "region_id",
 region_name AS "region_name"))
FROM regions;
```

```
SELECT *
FROM region_xmlv
WHERE ROWNUM < 3;
```

```
OBJECT_VALUE

<region
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="region.xsd"><region_id>1
 </region_id><region_name>Europe</region_name></region>
<region
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="region.xsd"><region_id>2
 </region_id><region_name>Americas</region_name></region>
>
```

# XML in Oracle

XMLType-Sichten: Schema-basierte XMLType-Sicht mit SQL/XML – mit TargetNS

```
CREATE OR REPLACE VIEW region_ns_xmlv OF XMLType
 XMLSCHEMA "region_ns.xsd" ELEMENT "region"
 WITH OBJECT ID (extract(OBJECT_VALUE,
 '/region/region_id').getNumberVal()) AS
 SELECT XMLElement("rgn:region",
 XMLAttributes(
 'http://www.w3.org/2001/XMLSchema-instance' AS "xmlns:xsi",
 'http://www.hr.com/regions' AS "xmlns:rgn",
 'http://www.hr.com/regions region_ns.xsd' AS
 "xsi:schemaLocation"),
 XMLForest(region_id AS "rgn:region_id",
 region_name AS "rgn:region_name"))
 FROM regions;
```

```
SELECT *
 FROM region_ns_xmlv
 WHERE ROWNUM = 1;
```

```
<rgn:region
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:rgn="http://www.hr.com/regions"
 xsi:schemaLocation="http://www.hr.com/regions
 region_ns.xsd"><rgn:region_id>1</rgn:region_id><rgn:region_name>Europe</rgn:region_name></rgn:region>
```

# XML in Oracle

XMLType-Sichten: Schema-basierte XMLType-Sicht aus XMLType-Tabelle

- Erstellen und füllen der XMLType-Tabelle:

```
CREATE TABLE region_xmltab OF XMLType
 XMLSchema "region.xsd" ELEMENT "region";
```

```
INSERT INTO region_xmltab
VALUES (XMLType(
'<region xmlns:xsi=
 "http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="region.xsd">
 <region_id>5</region_id>
 <region_name>Japan</region_name>
</region>'));
```

- Erstellen der XMLType-Sicht:

```
CREATE OR REPLACE VIEW region_xmltabv OF XMLType
 XMLSchema "region.xsd" ELEMENT "region"
AS SELECT value(r) FROM region_xmltab r;
```

# XML in Oracle

## XMLType-Sichten: Aktualisierbare XMLType-Sichten 1/2

- XMLType-Sichten weisen dieselben Einschränkungen bei Aktualisierungen auf wie herkömmliche Sichten
  - INSTEAD OF-Trigger können eingesetzt werden
- Beispiel:

- (1) Objekttyp erstellen:

```
CREATE TYPE region_t AS OBJECT(
 "id" NUMBER,
 "name" VARCHAR2(25)) ;
/
```

- (2) XML Schema generieren und registrieren

```
BEGIN
 DBMS_XMLSHEMA.registerSchema(
 'http://www.hr.com/region_t.xsd',
 DBMS_XMLSHEMA.generateSchema(
 USER, 'REGION_T', 'region'),
 TRUE, FALSE, FALSE);
END;
/
```

# XML in Oracle

## XMLType-Sichten: Aktualisierbare XMLType-Sichten 2/2

### (3) XMLType-Sicht erstellen

```
CREATE OR REPLACE VIEW region_upd_xmlv OF XMLType
 XMLSCHEMA "http://www.hr.com/region_t.xsd"
 ELEMENT "region" WITH OBJECT ID
 (object_value.extract(
 '/region/id').getNumberVal())
AS SELECT region_t(r.region_id, r.region_name)
 FROM regions r;
```

### (4) DML-Operation ausführen

```
INSERT INTO region_upd_xmlv
 VALUES (XMLType(
'<region xmlns:xsi=
 "http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation=
 "http://www.hr.com/region_t.xsd">
<id>5</id>
<name>Pacific</name>
</region>'));
```

# Inhalt

---

- Motivation ✓
- Speichertechniken ✓
- Abfragetechniken ✓
- XML in Oracle
  - Speicheroptionen ✓
  - Abfragen von XML-Daten ✓
  - Manipulation von XML-Daten ✓
  - XMLType-Sichten ✓

## Modul 11

# JSON

## JavaScript Object Notation

```
{
 "Lehrveranstaltung": {
 "#typ": "Vorlesung",
 "titel" : "Semistrukturierte Datenmodellierung und XML",
 "kapitel" : "JSON",
 "leiter": "Julian-Paul Haslinger",
 "email": "julian@haslinger.io",
 "andereLVAs": ["CLC", "DAE"]
 }
}
```

Julian Haslinger

Der vorliegende Foliensatz basiert vorwiegend auf:

- Standard ECMA-404, „The JSON Data Interchange Format“, 2<sup>nd</sup> ed., Ecma International, Dec 2017
- Michael Inden, „Der Java-Profi: Persistenzlösungen und REST-Services“, dpunkt, Mai 2016



# Inhalt

## ■ JSON

- Einführung / Grammatik
- Anwendungsbeispiele
- Vor- und Nachteile von JSON / XML
- JSON-Verarbeitung
- Aktuelle Themen im JSON-Umfeld
- Ausblick „JSON und XML“
- JSON und Datenbanken
- Literatur / Ressourcen

# JSON – JavaScript Object Notation

## Einführung 1/2

- JSON (/dʒeɪ·sən/) ist ein leichtgewichtiges, einfach aufgebautes, text-basiertes, sprach-unabhängiges Datenformat
  - Unterstützt hierarchische Strukturen
  - Beruht auf Schlüssel-Wert-Definitionen
- JSON basiert auf einigen Standarddatentypen wie
  - Zahlen (Ziffern-Sequenz; ganzzahlig und Gleitkomma)
  - Text (Unicode-Zeichen-Sequenz)
  - Boolesche Werte
  - Leerer Wert `null`
- JSON unterstützt
  - Name/Werte-Paare (`record`, `struct`, `dict`, `map`, `hash`...)
  - Listen (`array`, `vector`, `list`...)
- JSON unterstützt nicht
  - Zyklische Graphen
  - Binärdaten

# JSON – JavaScript Object Notation

## Einführung 2/2

- JSON ist durch den *ECMA-404*-Standard standardisiert
  - 1<sup>st</sup> edition: 2013
  - 2<sup>nd</sup> edition: 2017
    - Umfang: ca. 5 Seiten
- 2001 auf json.org erstmals präsentiert
- Einfache Datenmodellierungssprache
  - → zukunftssicher, da sich die Definition vermutlich nie ändert

# JSON – Struktur

## ■ 6 Struktur-Token

- Eckige Klammern: [ , ]
- Geschwungene Klammern: { , }
- Doppelpunkt: :
- Komma: ,

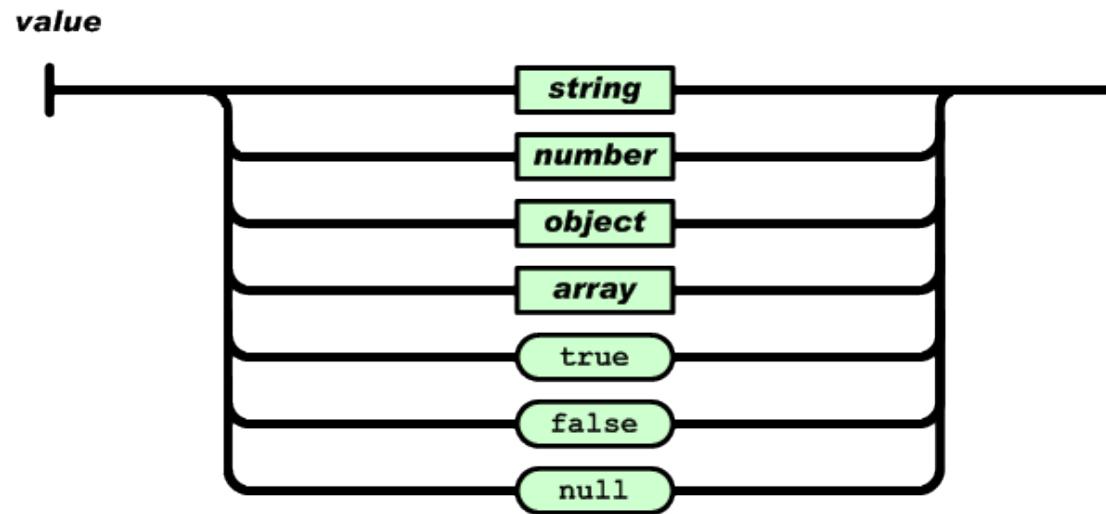
## ■ 3 Literal-Token

- true
- false
- null

# JSON

## Grammatik 1/5

- Werte ([values](#))
  - Kommentare sind nicht erlaubt

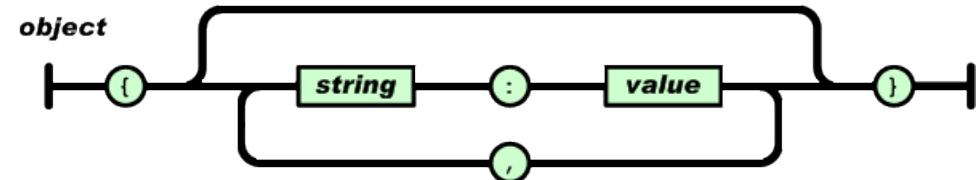


# JSON

## Grammatik 2/5

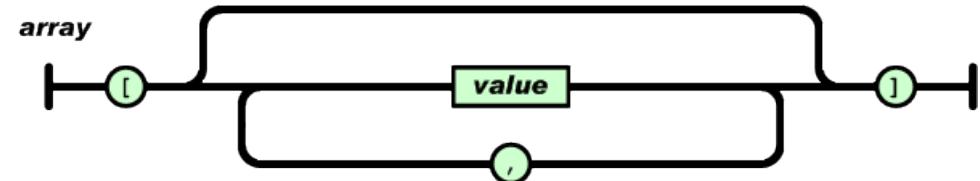
### ■ Objekt (object)

- Menge von Key-Value-Paaren
- Key ist immer ein String (Duplikate nicht erlaubt)
- Value ist ein beliebiger Typ
- Key-Value-Paare (Doppelpunkt zwischen Key und Value) werden in geschwungenen Klammern gesetzt und durch Beistriche getrennt
- Beispiel eines komplexen Objekts:
  - `{"lva-leiter": { "vorname": "Julian", "nachname": "Haslinger", "personalnr": "P22080" }}`



# JSON

## Grammatik 3/5



### ■ Array (array)

- Geordnete Liste von Werten
- Elemente der Liste können beliebige Typen aufnehmen
- Werte werden in eckige Klammern gesetzt und durch Beistriche getrennt
- Beispiel:
  - Einfaches Array:  
`{"models": ["iPhone 13", "iPhone 14"]}`
  - Verschachteltes Array:  
`{"models": ["iPhone 14", ["Pro Max", 2, 3]]}`

# JSON

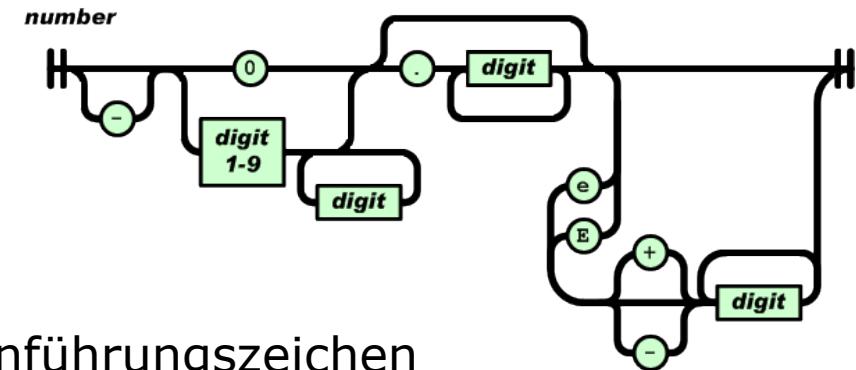
## Grammatik 4/5

### ■ Zahl (`number`)

- Folge von Ziffern (0-9)
- Vorzeichen (+, -), Dezimalpunkt (.), Exponent (e, E)
- Beispiel:

`{"zahl":1}`

`{"zahl":-10e+13}`

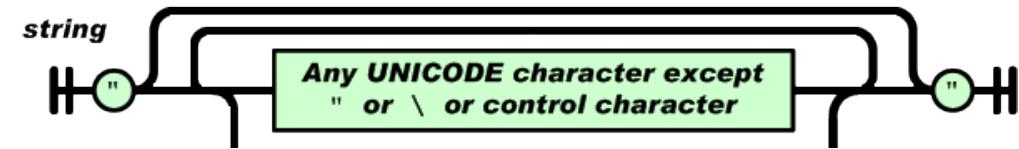


### ■ Zeichenkette (`string`)

- Unicode-Zeichen-Sequenz in Anführungszeichen
- Beispiel:

`{"matrikelnummer":"S180307000"}`

`{"string":"newline \n"}`

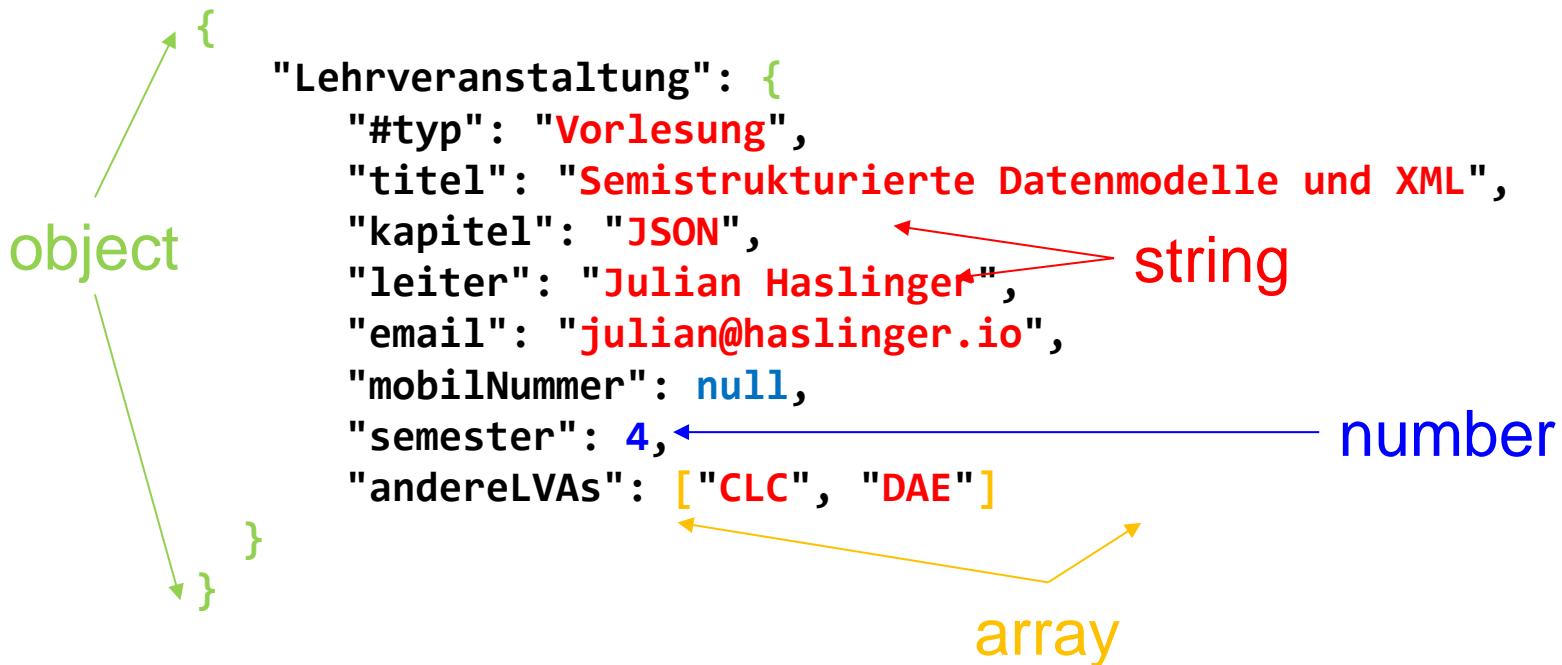


# JSON

## Grammatik 5/5

### ■ Objekt (object)

- beginnt mit { und endet mit }
- enthält eine durch Komma geteilte, ungeordnete Liste von Eigenschaften (Schlüssel/Werte-Paare)
- Objekte ohne Eigenschaften (leere Objekte) sind zulässig



# JSON

## Anwendungsbeispiele

- JSON wird vor allem zum Datenaustausch im Web verwendet.
- JSON kann einfach in JavaScript-Objekte geparsst werden
  - Einfachere Anwendung in Web-Applikationen
  - Einfachere Realisierung von AJAX-Anwendungen
  - JavaScript
    - `JSON.parse(...)` - JSON-Zeichenkette → JS-Objekt
    - `JSON.stringify(...)` - JS-Objekt → JSON-Zeichenkette
- JSON wird als Datenspeicherungsformat in NoSQL-Systemen verwendet (z.B. MongoDB, CouchDB)
- JSON steht als Datentyp in relationalen Datenbanken zur Verfügung (z.B. Oracle, DB2, MS SQL Server, PostgreSQL)

# JSON

## JSON vs. XML

- XML ist eine Sprache – JSON ein Datenformat
- JSON wurde nicht als Markup-Sprache entwickelt
  - JSON-Syntax ist kompakt und für den effizienten Datenaustausch optimiert.
  - JSON-Daten können einfach in JavaScript (Browser!) verwendet werden.
  - Mithilfe von XML können neue Sprachen zum standardisierten Austausch von Daten und zum Beschreiben von Dokumenten erstellt werden.
- JSON: eher Daten-orientiert
- XML: eher Dokumenten-orientiert

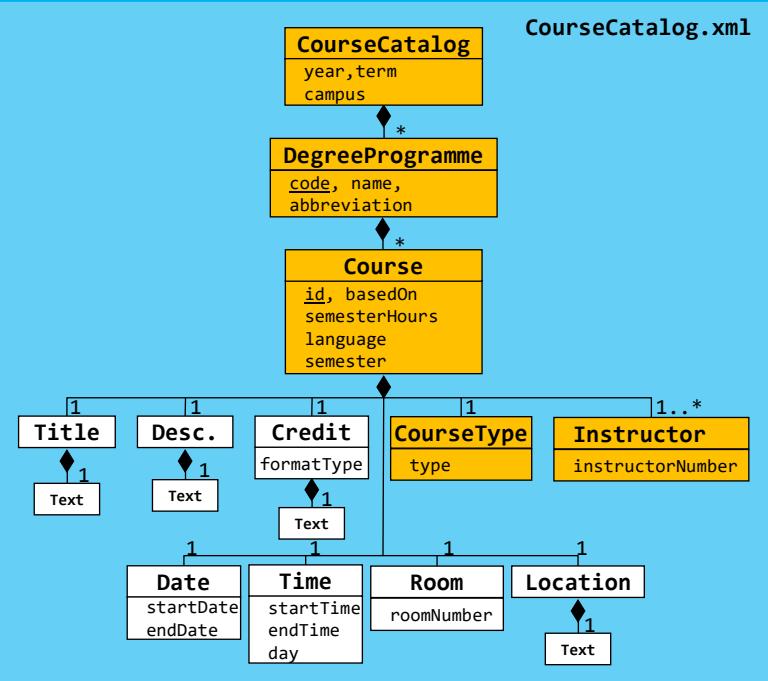
JSON: The Fat-Free Alternative to XML [[www.json.org/xml.html](http://www.json.org/xml.html)]

# Eigenschaften von XML

- **XML** ist eine Meta-Sprache, mit der neue Vokabulare definiert werden können.
  - Elemente und Attribute zur Definition von Daten und Metadaten
- **XSLT** ist eine (funktionale) Sprache, mit der XML-Dokumente außerhalb von bekannten Programmiersprachen verarbeitet/transformiert werden können.
- Mit **XPath** können Teile von Dokumenten abgefragt werden.
- Mit **XQuery** gibt es eine Möglichkeit, wie man Daten aus XML-Datenbanken selektiert.

# Beispiel

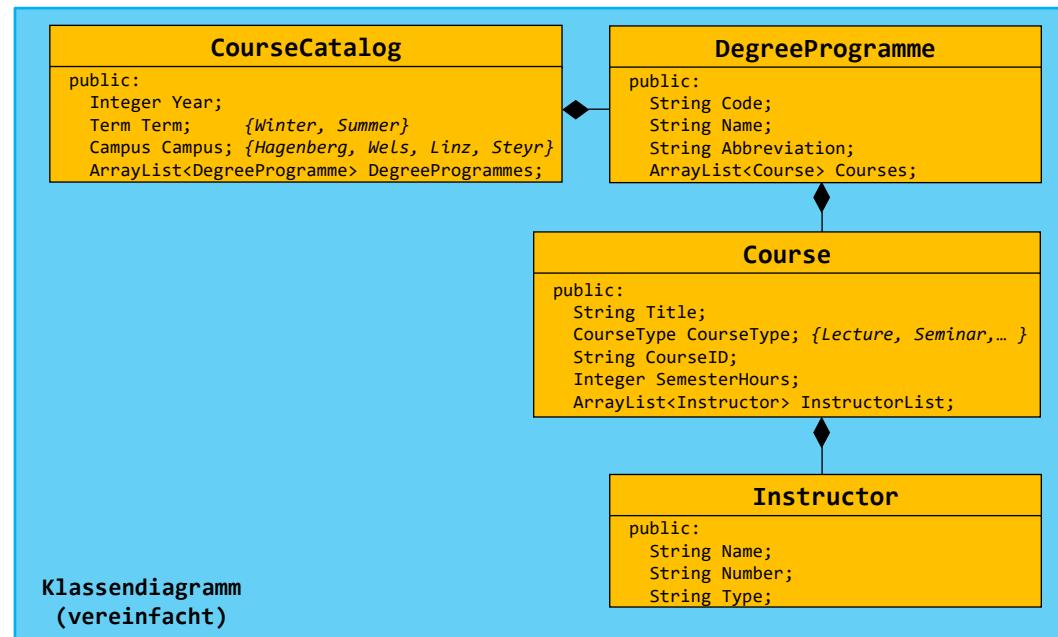
## Kurskatalog



```

<CourseCatalog year="2019" term="summer" campus="Hagenberg">
 <DegreeProgramme code="0307" name="Software Engineering" abbreviation="SE">
 <Course id="cID_8314" semesterHours="1" language="en" semester="4">
 <Title>Introduction to semi-structured data models and XML</Title>
 <Description>Introduction of skills related to XML.<Content>Includes DTD, Schema, XPath, XQuery, XSLT, JSON</Content> <Exam>Final Exam required.</Exam>Participation without any previous knowledge.</Description>
 <Credit formatType="ECTS">1</Credit>
 <CourseType type="Lecture"/>
 <Date startDate="--02-28" endDate="--05-03"/>
 <Time startTime="08:00:00" endTime="10:25:00" day="THU"/>
 <Room roomNumber="1.004"/>
 <Instructor instructorNumber="p22080" type="NBL">Julian Haslinger</Instructor>
 </Course>
 <!!-- ... -->

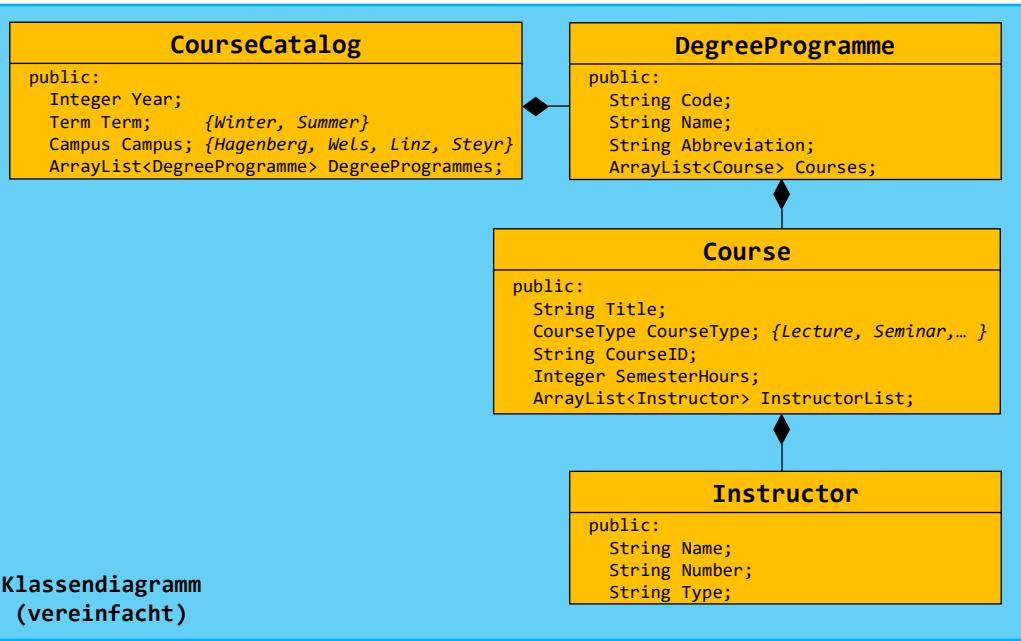
```



# JSON

## JSON-Verarbeitung - Beispiel

- CourseCatalog (Auszug)
  - Attribute zu CourseCatalog
  - Kindelemente von
    - CourseCatalog
    - » DegreeProgramme
    - Course



```
{
 "Year": 2019,
 "Term": "Summer",
 "Campus": "Hagenberg",
 "Degree Programmes": [
 {
 "Code": "0307",
 "Name": "Software Engineering",
 "Abbreviation": "SE",
 "Courses": [
 {
 "courseID": "cID_8314",
 "courseType": "LabSession",
 "semesterHours": 2,
 "CourseTitle": "Introduction to XML",
 "InstructorList": [
 {
 "Instructor Number": "P22080",
 "name": "Julian Haslinger",
 "Type": "NBL"
 }
]
 }
]
 }
]
}
```

# JSON

## JSON-Verarbeitung - Grundlagen

- JSON-Bibliotheken ermöglichen das Serialisieren und Deserialisieren von JSON bzw. Objekte/Objektgraphen
- JSON wird in vielen Programmiersprachen unterstützt
  - Diverse Libraries
  - In die Sprache eingebaut bzw. standardmäßig über die unterschiedlichen Datentypen unterstützt
- JSON wird auch von aktuelleren Standards im XML-Umfeld unterstützt
  - XPath 3.1
  - XQuery 3.1
  - XSLT 3.0

# JSON

## JSON-Verarbeitung im XML-Umfeld

- Aktuelle Versionen von XPath und XQuery unterstützen die Verarbeitung von JSON
  - „XPath and XQuery Functions and Operators 3.1“ (<https://www.w3.org/TR/xpath-functions-31/#json>)
  - 2 Möglichkeiten
    - JSON über **maps** und **arrays** darstellen
      - » JSON objects → XDM **maps**
      - » JSON arrays → XDM **arrays**
      - » 2 Funktionen: **fn:parse-json(...)** und **fn:serialize(...)**
    - JSON über XML darstellen
      - » JSON wird als Kombination aus (XDM) Elemente und Attribute dargestellt
      - » **fn:json-to-xml(...)**
      - » **fn:xml-to-json(...)**
      - » **fn:json-doc(...)**
      - » **fn:parse-json(...)**

# JSON

## JSON-Verarbeitung im XML-Umfeld

CourseCatalog.json

```
{
 "Year": 2019,
 "Term": "Summer",
 "Campus": "Hagenberg",
 "Degree Programmes": [
 {
 "Code": "0307",
 "Name": "Software Engineering",
 "Abbreviation": "SE",
 "Courses": [
 {
 "courseID": "cID_8314",
 "courseType": "LabSession",
 "semesterHours": 2,
 "CourseTitle": "Introduction to XML",
 "InstructorList": [
 {
 "Instructor Number": "P22080",
 "name": "Julian Haslinger",
 "Type": "NBL"
 }
]
 }
]
 }
]
}
```

json-to-xml(...)

```
<?xml version="1.0" encoding="UTF-8"?>
<map xmlns="http://www.w3.org/2005/xpath-functions">
 <number key="Year">2019</number>
 <string key="Term">Summer</string>
 <string key="Campus">Hagenberg</string>
 <array key="Degree Programmes">
 <map>
 <string key="Code">0307</string>
 <string key="Name">Software Engineering</string>
 <string key="Abbreviation">SE</string>
 <array key="Courses">
 <map>
 <string key="courseID">cID_8314</string>
 <string key="courseType">LabSession</string>
 <number key="semesterHours">2</number>
 <string key="CourseTitle">Introduction to XML</string>
 <array key="InstructorList">
 <map>
 <string key="Instructor Number">P22080</string>
 <string key="name">Julian Haslinger</string>
 <string key="Type">NBL</string>
 </map>
 </array>
 </map>
 </array>
 </map>
 </array>
</map>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="3.0">
 <xsl:output method="xml" indent="yes"/>
 <xsl:variable name="jsondoc" select="unparsed-text('./courseCatalogFiles/CourseCatalog.json')"/>
 <xsl:template match="/">
 <xsl:copy-of select="json-to-xml($jsondoc)"/>
 </xsl:template>
</xsl:stylesheet>
```

json-to-xml.xslt

# JSON

## JSON-Verarbeitung in JAVA (mit Jackson) - Grundlagen

- Jackson besteht aus mehreren JAR-Dateien
  - *jackson-core* [<https://github.com/FasterXML/jackson-core>]
  - *jackson-annotation* [<https://github.com/FasterXML/jackson-annotations>]
    - z.B. *Definition der Serialisierungs-Reihenfolge von Datenkomponenten (@JsonPropertyOrder)* und *Neu-Benennung der Datenkomponenten (@JsonProperty)*
  - *jackson-databind* [<https://github.com/FasterXML/jackson-databind>]
    - z.B. Klasse `ObjectMapper`
- Klasse `ObjectMapper` wird u.a. zum Lesen und Schreiben von JSON verwendet („Mappen“ von JSON-Daten auf Objekte)
  - Lesen
    - `readValue(...)`
  - Schreiben
    - `writeValue(...)`
    - `writeValueAsString(...)`

# JSON

## JSON-Verarbeitung in JAVA (mit Jackson) - Grundlagen

- Jackson bietet mehrere Möglichkeiten, um JSON-Dateien zu verarbeiten
  - Data-Binding (siehe vorige Folie)
    - Mapping JSON-Dokument  $\Leftrightarrow$  Objekte (Klassen)
  - Baumstruktur (tree model)
    - Einlesen einer Baum-Repräsentation
    - Navigation innerhalb des Baumes (`<context>.path(...)`) – Rückgabe: `JsonNode`
    - Verarbeiten der einzelnen `JsonNode`-Knoten
    - Manipulationen innerhalb des Baumes möglich
    - vgl. DOM
  - Streaming-Variante
    - Iteration über das Dokument und schrittweises Verarbeiten der einzelnen JSON-Dokumentteile

# JSON

## JSON-Verarbeitung in JAVA (mit Jackson) – Data-Binding

```
@JsonPropertyOrder({ "Year", "Term", "Campus", "DegreeProgrammes" })
public class CourseCatalog {
```

```
 @JsonProperty("Year")
 public Integer Year;
```

```
 @JsonProperty("Term")
 public Term Term;
```

```
 @JsonProperty("Campus")
 public Campus Campus;
```

```
 @JsonProperty("Degree Programmes")
 public ArrayList<DegreeProgramme> DegreeProgrammes;
// ...
```

```
 @JsonPropertyOrder({ "CourseID", "CourseType",
 "SemesterHours", "Title", "InstructorList" })
 public class Course {
```

```
 @JsonProperty("CourseTitle")
 private String Title;
```

```
// ...
```

```
 @JsonProperty("Instructor List")
 public ArrayList<Instructor> InstructorList;
// ...
```

```
 @JsonPropertyOrder({ "Number", "Name", "Type" })
 public class Instructor {
// ...
}
```

```
 @JsonPropertyOrder({ "Code", "Name", "Abbreviation", "Courses" })
 public class DegreeProgramme {
 public String Code;
 public String Name;
 public String Abbreviation;
 public ArrayList<Course> Courses;
// ...
```

Name im Ergebnisdokument

Reihenfolge im Ergebnisdokument

De- / Serialisierung

CourseCatalog.json

```
{
 "Year": 2019,
 "Term": "Summer",
 "Campus": "Hagenberg",
 "Degree Programmes": [
 {
 "Code": "0307",
 "Name": "Software Engineering",
 "Abbreviation": "SE",
 "Courses": [
 {
 "courseID": "cID_8314",
 "courseType": "LabSession",
 "semesterHours": 2,
 "CourseTitle": "Introduction to XML",
 "Instructor List": [
 {
 "Instructor Number": "P22080",
 "name": "Julian Haslinger",
 "Type": "NBL"
 }
]
 }
]
 }
]
}
```

# JSON

## JSON-Verarbeitung in JAVA (mit Jackson) – Data-Binding

### ■ Schreiben (Objekt → JSON, Serialisierung)

```
ObjectMapper mapper = new ObjectMapper();

// Create a new course catalog ...
CourseCatalog catalog = new CourseCatalog(2019, Term.Summer, Campus.Hagenberg);
// other properties of the course catalog

mapper.writeValue(new File("CourseCatalog0307.json"), catalog);
// ...
```

### ■ Lesen (JSON → Objekt, Deserialisierung)

*Objekt vom Typ ObjectMapper kann wiederverwendet werden.*

```
ObjectMapper mapper = new ObjectMapper();

CourseCatalog courseCatalog =
 mapper.readValue(new File("CourseCatalog0307.json"), CourseCatalog.class);

System.out.println(
 mapper.writerWithDefaultPrettyPrinter().writeValueAsString(courseCatalog));
```

# JSON

## Technologien rund um JSON - Auswahl

- Bemühungen, um bestimmte XML-Technologien auch im JSON-Umfeld anzusiedeln, z.B.
  - JSON Schema (dzt. Version 2020-12)
    - <http://json-schema.org/>
    - <https://json-schema.org/learn/>
    - <https://json-schema.org/implementations.html>
  - JSONPath
    - <https://github.com/json-path/JsonPath>
    - Online Evaluator: <http://jsonpath.com/>
- Zusätzlich gibt es eine Vielzahl an Tools und Technologien, die auf JSON aufbauen
- Oft noch keine etablierten Standards, aber eine Reihe von Kandidaten
- Implementierungsgrad der unterschiedlichen Spezifikationen sehr abhängig von den jeweils verwendeten Tools
  - [Beispiel](#): JSON Schema Unterstützung von XMLSpy: draft-04, draft-06, draft-07

# JSON

## Ausblick „JSON und XML“

- Wichtig: Interoperabilität der beiden Formate
  - Unterschiedliche Datenquellen mit unterschiedlichen Datenformaten müssen gleichzeitig abgerufen und verarbeitet werden können
- Anwendungsgebiete für beide Technologien klar definieren – Vorteile der Technologien ausnutzen
  - XML vorwiegend als Dokumentenformat
  - JSON vorwiegend als Datenaustauschformat (mit einfacher Integration in Programmiersprachen)

# JSON

## JSON und Datenbanken

- Speicherung
  - meist als **(N)VARCHAR**, **CLOB**, **BLOB**
- SQL Standard
  - SQL:2016 standardisiert JSON-Support
  - [ISO/IEC TR 19075-6:2017: SQL support for JavaScript Object Notation \(JSON\)](#)
- Abfragesprache
  - JSON APIs für verschiedene Programmiersprachen
  - Noch keine standardisierte JSON Query Language
    - proprietäre JSON Query Language in NoSQL-DBMS
- Datenbanken mit JSON-Unterstützung
  - *RDBMS*: [SQL Server](#), [PostgreSQL](#), IBM DB2, Oracle etc.
  - *NoSQL*: MongoDB, CouchDB, Couchbase, MarkLogic, BaseX etc.

# JSON

## JSON und Oracle



- Speicherung als `VARCHAR2`, `CLOB` oder `BLOB`
- Validierung (`IS JSON` bzw. `IS NOT JSON`) – kann als `CHECK`-Constraint oder in Abfragen verwendet werden
- Anfragen in starker Anlehnung an XML-Integration
  - Abfragen durch Funktionen `JSON_VALUE`, `JSON_EXISTS`, `JSON_QUERY` und `JSON_TABLE` integriert in SQL (starke Analogie zu SQL/XML!)
  - REST-API für nativen JSON-Zugriff (ohne SQL)
  - Funktionen für Generierung von JSON aus relationalen Daten
    - z.B. `JSON_OBJECT` und `JSON_ARRAY`



JSON

[Oracle, JSON Developer's Guide

<https://docs.oracle.com/en/database/oracle/oracle-database/12.2/adjsn/json-developers-guide.pdf>

# Literatur / Ressourcen

- Offizielle Homepage
  - <http://json.org/>
- RFC 4627, Juli 2006
  - <https://tools.ietf.org/html/rfc4627>
  - (aktuell: RFC 8259, Dezember 2017,  
<https://tools.ietf.org/html/rfc8259>)
- ECMA-404 – „The JSON Data Interchange Format“, 2<sup>nd</sup> edition, 2017
  - <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>
- MSDN – „Comparing JSON to XML“
  - <https://msdn.microsoft.com/en-us/library/bb299886.aspx>

## Modul 12

# {✓} X JSON Schema

Julian Haslinger

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://example.org/schema/courseCatalog",
 "title": "Course Catalog",
 "description": "Schema for degree programmes and courses.",
 "type": "object",
 "properties": {
 "year": { "type": "integer" },
 "term": { "enum": ["Winter", "Summer"] },
 "degreeProgrammes": {
 "type": "array",
 "items": { "$ref": "#/$defs/degreeProgramme" },
 "minItems": 1
 },
 "required": ["year", "term", "degreeProgrammes"],
 ...
 }
}
```

**Der vorliegende Foliensatz basiert vorwiegend auf:**

json-schema.org, „JSON Schema“ [[json-schema.org/](https://json-schema.org/)]

json-schema.org, „Understanding JSON Schema“ [[json-schema.org/understanding-json-schema/](https://json-schema.org/understanding-json-schema/)]



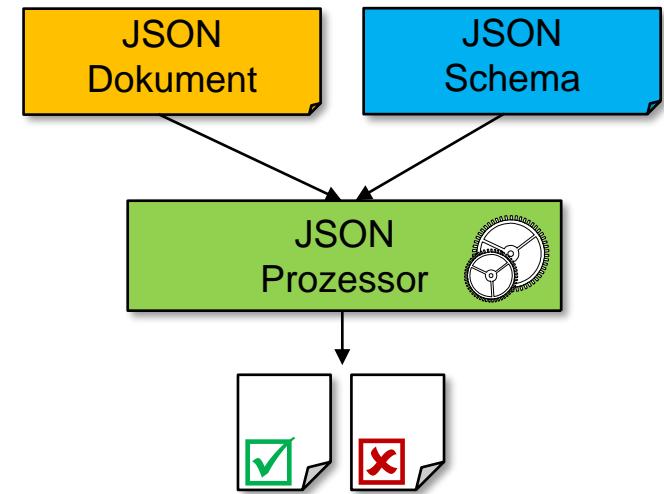
# Inhalt

- **Einführung**
- **Grundlagen**
  - Schema Core and Validation
  - Schema
  - Annotationen
  - Validierung
- **Modellierungskonzepte**
  - Basisdatentypen
  - Schemakomposition
- **Zusammenfassung**
  
- Anhang I: JSON Schema versus XML Schema
- Anhang II: Entwurfsrichtlinien

# Einführung 1/2

## Umfang

- JSON Schema definiert Struktur und Aufbau von JSON Instanzdokumenten
- JSON Schema unterstützt
  - die Beschreibung und
  - die Validierung sowie
  - den Datenaustausch und
  - die Code-Generierung.
- JSON Schema unterstützt keine
  - Namensräume
  - benutzerdefinierten Datentypen und Vererbung
  - strikte Reihenfolge von Objekteigenschaften
  - Schlüssel und Schlüsselreferenzen



# Einführung 2/2

## Standardisierung

- Internet Engineering Task Force (IETF)
- JSON Schema ist mit JSON definiert
  - Media type [application/schema+json](#)
  - derzeit als Entwurfsversion [Draft 2020-12](#) verfügbar
- JSON Schema<sup>1</sup> besteht aus
  - JSON Schema Core<sup>2</sup>
  - JSON Schema Validation<sup>3</sup>
  - JSON Pointer<sup>4</sup>

<sup>1</sup> json-schema.org, „JSON Schema“ [[json-schema.org](https://json-schema.org)]

<sup>2</sup> IETF, „draft-bhutton-json-schema-00 (core)“ [<https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-00>]

<sup>3</sup> IETF, „draft-bhutton-json-schema-validation-00“ [<https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-validation-00>]

<sup>4</sup> IETF, „draft-bhutton-relative-json-pointer-00“ [<https://datatracker.ietf.org/doc/html/draft-bhutton-relative-json-pointer-00>]

# Inhalt

- Einführung
  - **Grundlagen**
    - Schema Core and Validation
    - Schema
    - Annotationen
    - Validierung
  - Modellierungskonzepte
    - Basisdatentypen
    - Schemakomposition
  - Zusammenfassung
- 
- Anhang I: JSON Schema versus XML Schema
  - Anhang II: Entwurfsrichtlinien

# Grundlagen 1/5

## JSON Schema Core and Validation Draft 2020-12

### Schema

- **\$schema** sets the meta-schema
- **\$id** defines the base-URI for this schema, used in schema references
- **\$defs** document section dedicated for sub-schemata
- **\$ref** used to reference other schemata
- **\$anchor** defines custom referable schema links

### Annotation

- **title** and **description** provide a short or longer description for a schema
- **default** default value for a schema
- **deprecated** if true defines that a schema should not be used anymore
- **readOnly** and **writeOnly** declares if values should only be read- or writable
- **examples** array of example values for a schema
- **\$comment** developer comments

### General Validation

- **type** restrict the data type to `object`, `array`, `number`, `integer`, `null`, `boolean`, `string`
- **enum** defines an array of permitted values
- **const** requires a specific value

### Numeric Validation

- **(exclusive) maximum** restrict the value to a range
- **(exclusive) minimum** restrict the value to a range
- **multipleOf** multiple of this number

### String Validation

- **maxLength** restrict string length
- **minLength** restrict string length
- **pattern** matches string with a regular expression
- **format** matches string with a predefined format  
ex. date, time, email, ect.

### Array Validation

- **items** or **prefixItems** defines the schemata to validate array elements with
- **maxItems** and **minItems** restrict the array size
- **uniqueItems**: if true checks array items for uniqueness
- **contains**, **maxContains** and **minContains** defines which schema has to be included in an array. Optionally restrict the number of occurrences.
- **items** schema for additional items in a prefixItems context

### Object Validation

- **properties** defines names and schemata of properties
- **maxProperties** and **minProperties** restrict the amount of properties
- **propertyNames** and **patternProperties** defines which schema or pattern the property names.should match
- **required** array including names of properties that are required
- **additionalProperties** and **unevaluatedProperties** defines which schema is used for additional or unevaluated properties
- **dependentRequired** and **dependentSchema** may require or apply a schema depending on the presence of a property

# Grundlagen 2/5

## Schema

### Schema

- **\$schema** sets the meta-schema
- **\$id** defines the base-URI for this schema, used in schema references
- **\$defs** document section dedicated for sub-schemata
- **\$ref** used to reference other schemata
- **\$anchor** defines custom referable schema links

## ■ Schema Keywords identifizieren, strukturieren und referenzieren JSON Schemata und Sub Schemata

CourseCatalog\_Schema.json

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema", ← Meta Schema (JSON Schema Standard)
 "$id": "https://www.fh-ooe.at/courseCatalog-Schema.json", ← Root Schema Base URI (for schema references)
 "title": "Course Catalog",
 "description": "Schema for degree programmes and courses.",
 "type": "object",
 "properties": {
 ...
 "degreeProgrammes": {
 "type": "array",
 "items": { "$ref": "#/$defs/degreeProgramme" },
 ...
 },
 ...
 },
 "$defs": {
 "degreeProgramme": {
 "type": "object", ...
 }
 }
}
```

Document section dedicated for **Sub Schema**

**Sub Schema** Reference

Structuring a complex schema [<https://json-schema.org/understanding-json-schema/structuring.html>]

# Grundlagen 3/5

## Annotation

### Annotation

- `title` and `description` provide a short or longer description for a schema
- `default` default value for a schema
- `deprecated` if true defines that a schema should not be used anymore
- `readOnly` and `writeOnly` declares if values should only be read- or writable
- `examples` array of example values for a schema
- `$comment` developer comments

### ■ Annotation Keywords dokumentieren JSON Schemata

- Keywords sind optional und werden nicht für die Validierung verwendet

CourseCatalog\_Schema.json

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://www.fh-ooe.at/courseCatalog-Schema.json",
 "title": "Course Catalog",
 "description": "Schema for degree programmes and courses.",
 "comment": "TBD: Version 2.0 separate in sub schemas",
 "type": "object",
 "properties": {
 "institution": { "type": "string", "examples": ["FH OOE", "JKU"]},
 "year": { "type": "integer", "default": "2022"},
 "term": { "enum": ["Winter", "Summer"], "readOnly": "true"},
 "campus": { "enum": ["Linz", "Wels"], "$comment": "May be extended if needed." },
 "degreeProgrammes": {
 ...
 }
 },
 ...
}
```

} Informationen für Werkzeuge  
} Entwicklungsdocumentation

# Grundlagen 4/5

## Validierung

### General Validation

- **type** restrict the data type to `object`, `array`, `number`, `integer`, `null`, `boolean`, `string`
- **enum** defines an array of permitted values
- **const** requires a specific value

- JSON Schema unterstützt alle JSON Basisdatentypen
  - `string`, `number`, `object`, `array`, `boolean`, `null` und `integer`
  - abhängig vom Basisdatentyp können mit **Validation Keywords** Einschränkungen definiert werden

CourseCatalog\_Schema.json

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://www.fh-ooe.at/courseCatalog-Schema.json",
 "title": "Course Catalog",
 "description": "Schema for degree programmes and courses.",
 "comment": "TBD: Version 2.0 separate in sub schemas",
 "type": "object",
 "properties": {
 ...
 },
 ...
}
```

} ← Datentypspezifikation des Schemas

# Grundlagen 5/5

## Validierung

### General Validation

- **type** restrict the data type to `object`, `array`, `number`, `integer`, `null`, `boolean`, `string`
- **enum** defines an array of permitted values
- **const** requires a specific value

### ■ `enum`: Aufzählungstyp (mind. ein Element)

```
...
"creditType": {
 "enum": ["ECTS", "CP", null]
}
```

{ "creditType": "ECTS" }	<input checked="" type="checkbox"/>
{ "creditType": "CU" }	<input type="checkbox"/>
{ "creditType": null }	<input checked="" type="checkbox"/>

### ■ `const`: Konstantendeklaration

```
...
"isInternational": {
 "const": true
}
```

{ "isInternational": true }	<input checked="" type="checkbox"/>
-----------------------------	-------------------------------------

# Inhalt

- Einführung
- Grundlagen
  - Schema Core and Validation
  - Schema
  - Annotationen
  - Validierung
- Modellierungskonzepte
  - Basisdatentypen
    - String Validierung
    - Numeric Validierung
    - Object Validierung
    - Array Validierung
  - Schemakomposition
- Zusammenfassung
- Anhang I: JSON Schema versus XML Schema
- Anhang II: Entwurfsrichtlinien

## String Validation

- **maxLength** restrict string length
- **minLength** restrict string length
- **pattern** matches string with a regular expression
- **format** matches string with a predefined format  
ex. date, time, email, ect.

```
"courseId": { "type": "string", "pattern": "^cID[0-9]{4}$" }
```

"courseId": "cID1234"	<input checked="" type="checkbox"/>
"courseId": "c12345"	<input type="checkbox"/>

### ■ **format** unterstützt vordefinierte Formatspezifikationen

- Dates and times: "date-time", "time", "date", "duration"
- Email addresses: "email", "idn-email"
- Hostnames: "hostname", "idn-hostname"
- IP Addresses: "ipv4", "ipv6"
- etc.

```
"startTime": { "type": "string", "format": "date-time" }
"length": { "type": "string", "format": "duration" }
```

"startTime": "2022-04-01T20:20:39+00:00"	<input checked="" type="checkbox"/>
"length": "PT1H30M"	<input checked="" type="checkbox"/>

-- "PT1H30M": Duration of 1 hour 30 minutes

Regular Expressions [[https://json-schema.org/understanding-json-schema/reference/regular\\_expressions.html#regular-expressions](https://json-schema.org/understanding-json-schema/reference/regular_expressions.html#regular-expressions)]

Built-in formats [<https://json-schema.org/understanding-json-schema/reference/string.html#id8>]

- `(exclusive) maximum` restrict the value to a range
- `(exclusive) minimum` restrict the value to a range
- `multipleOf` multiple of this number

# Number Validation

- JSON Schema unterstützt `integer` und `number`
  - Validation Keywords gelten für beide numerischen Datentypen

```
"year": { "type": "integer" }
"semesterHours": { "type": "number" }
"ectsAmount": { "type": "number", "exclusiveMinimum": 0, "maximum": 30 }
"tutorContractHours": { "type": "integer", "multipleOf": 14 }
```

"year": 2022	✓
"year": 2022.22	✗
"semesterHours": 1.5	✓
"semesterHours": "4.236e8"	✗
	-- Numbers as strings are rejected
"ectsAmount": 0	✗
"tutorContractHours": 28	✓
"tutorContractHours": 0	✓
	-- Numbers restricted to a multiple of a given number

# Object Validation 1/4

## Object Validation

- `properties` defines names and schemata of properties
- `maxProperties` and `minProperties` restrict the amount of properties
- `propertyNames` and `patternProperties` defines which schema or pattern the property names.should match
- `required` array including names of properties that are required
- `additionalProperties` and `unevaluatedProperties` defines which schema is used for additional or unevaluated properties
- `dependentRequired` and `dependentSchema` may require or apply a schema depending on the presence of a property

- **Object Keywords** definieren Eingenschaften von JSON Objekten
  - `properties` definiert Schlüssel/Werte-Paare
  - `required` definiert obligatorische Schlüssel/Werte-Paare (`default = false`)
  - `additionalProperties` erlaubt/verbietet nicht definierte Schlüssel/Werte-Paare (`default = true`)

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://www.fh-ooe.at/courseCatalog-Schema.json",
 ...
 "type": "object",
 "properties": {
 "institution": { "type": "string" },
 "year": { "type": "integer" },
 "term": { "enum": ["Winter", "Summer"] },
 "campus": { "enum": ["Linz", "Hagenberg", "Wels", "Steyr"] },
 "degreeProgrammes": { ... }
 },
 "required": ["institution", "year", "term", "campus", "degreeProgrammes"],
 "additionalProperties": false,
 ...
}
```

# Object Validation 2/4

## Object Validation

- `properties` defines names and schemata of properties
- `maxProperties` and `minProperties` restrict the amount of properties
- `propertyNames` and `patternProperties` defines which schema or pattern the property names should match
- `required` array including names of properties that are required
- `additionalProperties` and `unevaluatedProperties` defines which schema is used for additional or unevaluated properties
- `dependentRequired` and `dependentSchema` may require or apply a schema depending on the presence of a property

## ■ Validierung von Schlüsselnamen - Property Names

- `patternProperties`: Regulärer Ausdruck definiert Schlüsselnamen
- `propertyNames`: Regulärer Ausdruck definiert Schlüsselnamen, Datentyp für Wert ist undefined

```
{
 ...
 "type": "object",
 "patternProperties": {
 "^S_": { "type": "string" },
 "^I_": { "type": "integer" }
 }
}
```

"S_25": "This is a string"	<input checked="" type="checkbox"/>
"I_0": 42	<input checked="" type="checkbox"/>
"S_0": 42	<input checked="" type="checkbox"/>
-- Property name with prefix S_ must be strings	
"I_25": "This is a string"	<input checked="" type="checkbox"/>
-- Property name with prefix I_ must be integer	

```
{
 ...
 "type": "object",
 "propertyNames": {
 "pattern": "^[A-Za-z_][A-Za-z0-9_]*$"
 }
}
```

"_a_proper_token_001": "value"	<input checked="" type="checkbox"/>
"001 invalid": "value"	<input checked="" type="checkbox"/>
-- Property name must be strings	

# Object Validation 3/4

## Object Validation

- `properties` defines names and schemata of properties
- `maxProperties` and `minProperties` restrict the amount of properties
- `propertyNames` and `patternProperties` defines which schema or pattern the property names should match
- `required` array including names of properties that are required
- `additionalProperties` and `unevaluatedProperties` defines which schema is used for additional or unevaluated properties
- `dependentRequired` and `dependentSchema` may require or apply a schema depending on the presence of a property

## ■ Instanzabhängige Eigenschaften (Properties)

- `dependentRequired`

```
{
 ...
 "type": "object",
 "properties": {
 "id": { "type": "string", "pattern": "^\w{4}$" }
 "title": { "type": "string" },
 ...
 "language": { "type": "string" },
 "languageCertificate": { "type": "string" },
 },
 "required": ["id", "title"],
 "dependentRequired": { "languageCertificate": ["language"] }
}
```

"id": "cID1234",  
"title": "Java Programming",  
"language": "en",  
"languageCertificate": "TOEFL"



"id": "cID1234",  
"title": "Java Programming",  
"languageCertificate": "TOEFL"  
-- language is missing



"id": "cID1234",  
"title": "Java Programming"



"id": "cID1234",  
"title": "Java Programming",  
"language": "en"



# Object Validation 4/4

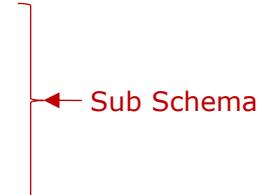
## Object Validation

- `properties` defines names and schemata of properties
- `maxProperties` and `minProperties` restrict the amount of properties
- `propertyNames` and `patternProperties` defines which schema or pattern the property names.should match
- `required` array including names of properties that are required
- `additionalProperties` and `unevaluatedProperties` defines which schema is used for additional or unevaluated properties
- `dependentRequired` and `dependentSchema` may require or apply a schema depending on the presence of a property

## ■ Instanzabhängige Sub-Schemata

- `dependentSchema`

```
{
 "type": "object",
 "properties": {
 "id": { "type": "string", "pattern": "^\w{4}\d{3}$" },
 "title": { "type": "string" },
 "languageCertificate": { "type": "string" },
 },
 "required": ["id", "title"],
 "dependentSchemas": {
 "languageCertificate": {
 "properties": {
 "language": { "type": "string" }
 },
 "required": ["language"]
 }
 }
}
```



<pre>"id": "cID1234", "title": "Java Programming", "language": "en", "languageCertificate": "TOEFL"</pre>	<input checked="" type="checkbox"/>
-----------------------------------------------------------------------------------------------------------------------	-------------------------------------

<pre>"id": "cID1234", "title": "Java Programming", "languageCertificate": "TOEFL" -- language is missing</pre>	<input type="checkbox"/>
----------------------------------------------------------------------------------------------------------------------------	--------------------------

# Array Validation 1/3

## Array Validation

- `items` or `prefixItems` defines the schemata to validate array elements with
- `maxItems` and `minItems` restrict the array size
- `uniqueItems`: if true checks array items for uniqueness
- `contains`, `maxContains` and `minContains` defines which schema has to be included in an array. Optionally restrict the number of occurrences.
- `items` schema for additional items in a `prefixItems` context

## ■ JSON Arrays unterstützen

- **List Validation:** Arrays mit variabler Größe und Elementen mit gleichen Schemata
- **Tuple Validation:** Arrays mit fixer Größe und Elementen mit unterschiedlichen Schemata

### List Validation

```
{
 "numberArray": {
 "type": "array",
 "items": {
 "type": "number"
 }
 }
}
```

### Tuple Validation

```
{
 "address": {
 "type": "array",
 "prefixItems": [
 { "type": "number" },
 { "type": "string" },
 { "enum": ["Street", "Avenue"] },
 { "enum": ["NW", "NE", "SW", "SE"] }
]
 }
}
```

[1, 2, 3, 4, 5, 6, 7]	<input checked="" type="checkbox"/>
[1, 2, "3", 4, 5, 6, 7]	<input type="checkbox"/>
[]	<input checked="" type="checkbox"/>

<input checked="" type="checkbox"/>
<input type="checkbox"/>
<input checked="" type="checkbox"/>

[1600, "Pennsylvania", "Avenue", "NW"]	<input checked="" type="checkbox"/>
[24, "Sussex", "Drive"]	<input type="checkbox"/>
["Palais de l'Élysée"]	<input type="checkbox"/>
[10, "Downing", "Street"]	<input checked="" type="checkbox"/>
[1600, "Oak", "Avenue", "NW", "Washington"]	<input checked="" type="checkbox"/>

# Array Validation 2/3

## Array Validation

- `items` or `prefixItems` defines the schemata to validate array elements with
- `maxItems` and `minItems` restrict the array size
- `uniqueItems`: if true checks array items for uniqueness
- `contains`, `maxContains` and `minContains` defines which schema has to be included in an array. Optionally restrict the number of occurrences.
- `items` schema for additional items in a `prefixItems` context

## ■ Tuple Validation mit/ohne zusätzlichen Elementen

```
{
 "address": {
 "type": "array",
 "prefixItems": [
 { "type": "number" },
 { "type": "string" },
 { "enum": ["Street", "Avenue"] },
 { "enum": ["NW", "NE", "SW", "SE"] }
],
 "items": false
 }
}
```

```
{
 "address": {
 "type": "array",
 "prefixItems": [
 { "type": "number" },
 { "type": "string" },
 { "enum": ["Street", "Avenue"] },
 { "enum": ["NW", "NE", "SW", "SE"] }
],
 "items": { "type": "string" }
 }
}
```

[1600, "Pennsylvania", "Avenue", "NW"]	<input checked="" type="checkbox"/>
[1600, "Pennsylvania", "Avenue" ]	<input checked="" type="checkbox"/>
[1600, "Oak", "Avenue", "NW", "Washington"]	<input type="checkbox"/>

[1600, "Oak", "Avenue", "NW", "Washington"]	<input checked="" type="checkbox"/>
[1600, "Oak", "Avenue", "NW", 20500]	<input type="checkbox"/>

# Array Validation 3/3

## Array Validation

- `items` or `prefixItems` defines the schemata to validate array elements with
- `maxItems` and `minItems` restrict the array size
- `uniqueItems`: if true checks array items for uniqueness
- `contains`, `maxContains` and `minContains` defines which schema has to be included in an array. Optionally restrict the number of occurrences.
- `items` schema for additional items in a `prefixItems` context

- `maxItems` und `minItems` definiert die Array-Größe
- `uniqueItems` überprüft Duplikate

```
{
 ...
 "degreeProgrammes": {
 "type": "array",
 "items": { "$ref": "#/$defs/degreeProgramme" },
 "comment": "by default minItems is 0",
 "minItems": 1,
 "maxItems": 12,
 "uniqueItems": true,
 "comment": "uniqueness validated for nested objects too"
 }
}
```

# boolean and null Validation

```
"isInternational": { "type": "boolean" }
"phone": { "type": ["string", "integer", "null"] }
```

"isInternational": false	<input checked="" type="checkbox"/>
"isInternational": "false"	<input type="checkbox"/>
"phone": null	<input checked="" type="checkbox"/>
"phone": 0	<input checked="" type="checkbox"/>

# Inhalt

- Einführung
- Grundlagen
  - Schema Core and Validation
  - Schema
  - Annotationen
  - Validierung
- Modellierungskonzepte
  - Basisdatentypen
  - Schemakomposition
- Zusammenfassung
- Anhang I: JSON Schema versus XML Schema
- Anhang II: Entwurfsrichtlinien

# Schemakomposition

## Schema Composition

- **allOf** (AND) Must be valid against all of the subschemas
- **anyOf** (OR) Must be valid against any of the subschemas
- **oneOf** (XOR) Must be valid against exactly one of the subschemas
- **not** (NOT) Must not be valid against the given schema

```
{ ...
 "appointment": {
 "type": "object",
 "properties": {
 "startTime": { "type": "string", "format": "date-time" },
 "length": { "type": "string", "format": "duration" },
 "room": { "$ref": "#/$defs/room" },
 "address": { "$ref": "#/$defs/address" }
 },
 "required": ["startTime", "length"],
 "oneOf": [
 { "required": ["room"] },
 { "required": ["address"] }
]
 }
}
```

```
{
 "startTime": "2021-11-14T14:30:00+00:00",
 "length": "PT45M",
 "room": {
 "roomNumber": "3.108",
 "building": "FH3",
 "description": "XORTEX LBS3"
 }
}
```

```
{
 "startTime": "2022-01-02T12:30:00+00:00",
 "length": "PT60M",
 "address": {
 "street": "White House Str.",
 "houseNumber": "1",
 "zip": "1000",
 "country": "United States of America"
 }
}
```

# Schemakomposition

## if-then-else

if	then	else	whole schema
T	T	n/a	T
T	F	n/a	F
F	n/a	T	T
F	n/a	F	F
n/a	n/a	n/a	T

```
{
 "type": "object",
 "properties": {
 "street_address": { "type": "string" },
 "country": { "default": "United States of America",
 "enum": ["United States of America", "Canada"] }
 },
 "if": {
 "properties": { "country": { "const": "United States of America" } } },
 "then": {
 "properties": { "postal_code": { "pattern": "[0-9]{5}(-[0-9]{4})?" } } },
 "else": {
 "properties": { "postal_code": { "pattern": "[A-Z][0-9][A-Z] [0-9][A-Z][0-9]" } } }
}
}
```

```
{
 "street_address": "1600 Oak Avenue NW",
 "country": "United States of America",
 "postal_code": "20500"
}
```



```
{
 "street_address": "24 Sussex Drive",
 "country": "Canada",
 "postal_code": "10000"
}
```



```
{
 "street_address": "24 Sussex Drive",
 "country": "Canada",
 "postal_code": "K1M 1M4"
}
```



```
{
 "street_address": "1600 Oak Avenue NW",
 "postal_code": "20500"
}
-- because "country" is not a required property
-- it will apply the then schema
```



# Inhalt

- Einführung
- Grundlagen
  - Schema Core and Validation
  - Schema
  - Annotationen
  - Validierung
- Modellierungskonzepte
  - Basisdatentypen
  - Schemakomposition
- Zusammenfassung
- Anhang I: JSON Schema versus XML Schema
- Anhang II: Entwurfsrichtlinien

# Zusammenfassung

- JSON Schema nutzt JSON Syntax
- JSON Schema basiert auf XML Schema Konzepten
  - unterscheidet sich jedoch in wesentlichen Punkten
- Validatoren stehen mit unterschiedlichen Konformitätsstufen zur Verfügung
  - Standard liegt als Entwurfsversion vor

## Anhang I

---

# **JSON Schema versus XML Schema**

---

Gegenüberstellung

---

# XML Schema versus JSON Schema

	XML Schema	JSON Schema
Standardisierungsprozess	W3C	Internet Engineering Task Force Arbeitsgruppe in Form einer Community
Reifegrad Standard	Recommendation XSD 1.1	Entwurfsversion Draft 2020-12
Verbreitung	weit verbreitet große Werkzeugunterstützung	nicht weit verbreitet geringere Werkzeugunterstützung
Syntax	benutzt XML Syntax	benutzt JSON Syntax
Komplexität (Syntax, Sprachumfang, Modellkomplexität)	schwergewichtig (?)	leichtgewichtig (?)
Namensräume	✓	✗
Vordefinierte Datentypen	zahlreiche Datentypen	nur wenige Datentypen
Benutzerdefinierte Datentypen	✓	✗ (~ Sub Schemata)
Vererbung	✓	✗
Wiederverwendung	✓	✓ (nur durch Erweiterung)
strikte Reihenfolge von Element- bzw. Objekteigenschaften	✓	✗ (nur in Arrays möglich)
Schlüssel, Schlüsselreferenzen	✓	✗

## Anhang II

---

# Entwurfsrichtlinien

# Entwurfsrichtlinie 1/6

- Definiere in `$schema` immer die Version des Meta Schemas lt. JSON Schema Standard
- Dokumentiere JSON Schema mit **Annotation Keywords**
- Vermeide strukturelle Komplexität
  - Vermeide tiefe Schachtelung von Schema-Definitionen
  - Verwende `$def`-Bereiche und/oder mehrere Schema-Dateien für Sub-Schema-Definitionen und referenziere diese mit `$ref`.

```
{ "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://www.fh-ooe.at/courseCatalog-Schema.json",
 "title": "Course Catalog",
 "description": "Schema for degree programmes and courses.",
 "comment": "TBD: Version 2.0 separate in sub schemas",
 "type": "object",
 "properties": {
 "appointments": {
 "type": "array",
 "items": { "$ref": "#/$defs/appointment" },
 "minItems": 1
 }
 }
}
```

# Entwurfsrichtlinie 2/6

## Property Groups

- Gruppiere zusammengehörige Eigenschaften (Properties) in einem Schema-Objekt

```
...
"$defs": {
 "credit": {
 "title": "Course Credit",
 "type": "object",
 "properties": {
 "amount": { "type": "number", "exclusiveMinimum": 0, "maximum": 30 },
 "creditType": {
 "enum": ["ECTS", "CP"]
 }
 },
 "required": ["amount", "creditType"],
 "additionalProperties": false
 }
}
```

# Entwurfsrichtlinie 3/6

## Generalisierung und Erweiterung

- Faktorisiere gemeinsame Eigenschaften zu einem *Base Schema*

Person\_Schema.json

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://www.fh-ooe.at/person-Schema.json",
 "title": "Person",
 "description": "Base Schema for Person",
 "type": "object",
 "properties": {
 "firstName": { "type": "string" },
 "lastName": { "type": "string" },
 "email": { "type": "string", "format": "email" },
 "phone": { "$ref": "#/$defs/phone" }
 },
 "required": ["firstName", "lastName", "email"]
}
```

- Benutzerdefinierten Datentypen,
- Vererbung und somit auch kein
- Überschreiben!

# Entwurfsrichtlinie 4/6

## Generalisierung und Erweiterung

### ■ Erweitere Base Schema

Instructor\_Schema.json

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://www.fh-ooe.at/instructor-Schema.json",
 "title": "Instructor",
 "description": "Extended Schema for Instructor based on Person",
 "type": "object",
 "properties": {
 "instructor": {
 "description": "An instructor is a person.",
 "$ref": "https://www.fh-ooe.at/person-Schema.json", } ← Person
 "properties": { Base Schema
 Instructor → ["instructorNr": { "type": "string", "pattern": "^p[1-4]{1}[0-9]{4}$" },
 "type": { "enum": ["HBL", "NBL"] }
 },
 "required": ["instructorNr"],
 "comment": "unevaluatedProperties disallows further properties in this context",
 "unevaluatedProperties": false
 }
 }
}
```

# Entwurfsrichtlinie 5/6

## Generalisierung und Erweiterung

```
{
 "instructor": {
 "instructorNr": "p22080",
 "firstName": "Juulian",
 "lastName": "Haslinger",
 "email": "julian.haslinger@fh-hagenberg.at",
 "phone": "0664 123123",
 "type": "NBL"
 }
}
```



```
{
 "instructor": {
 "instructorNr": "p20650",
 "firstName": "Norbert",
 "lastName": "Niklas",
 "email": "nobert.niklas@fh-hagenberg.at"
 }
}
```



```
{
 "instructor": {
 "instructorNr": "p20700",
 "lastName": "Traxler",
 "email": "barbara.traxler@fh-hagenberg.at"
 }
}
```



# Entwurfsrichtlinie 6/6

## Gemischter Inhalt

- Modelliere gemischten Inhalt alternativ mit `oneof`

```
"description": {
 "type": "array",
 "items": { "$ref": "#/$defs/descriptionItem" },
 "$comment": "if a description exists there has to be at least 1 element",
 "minItems": 1
}
```

```
"descriptionItem": {
 "description": "Mixed content: strings, objects (tool, exam)",
 "oneOf": [
 { "type": "string" },
 { "type": "object",
 "properties": {
 "tool": { "type": "string" },
 "exam": { "type": "string" },
 },
 "$comment": " '(min|max)Properties: 1' here has the same meaning as
 'oneOf' for all properties",
 "minProperties": 1,
 "maxProperties": 1,
 "additionalProperties": false
 }
]
}
```

[“Hello”, {"tool": “JSON Schema”}, “is being used!”]

Modul 13

---

# Rück- und Ausblick

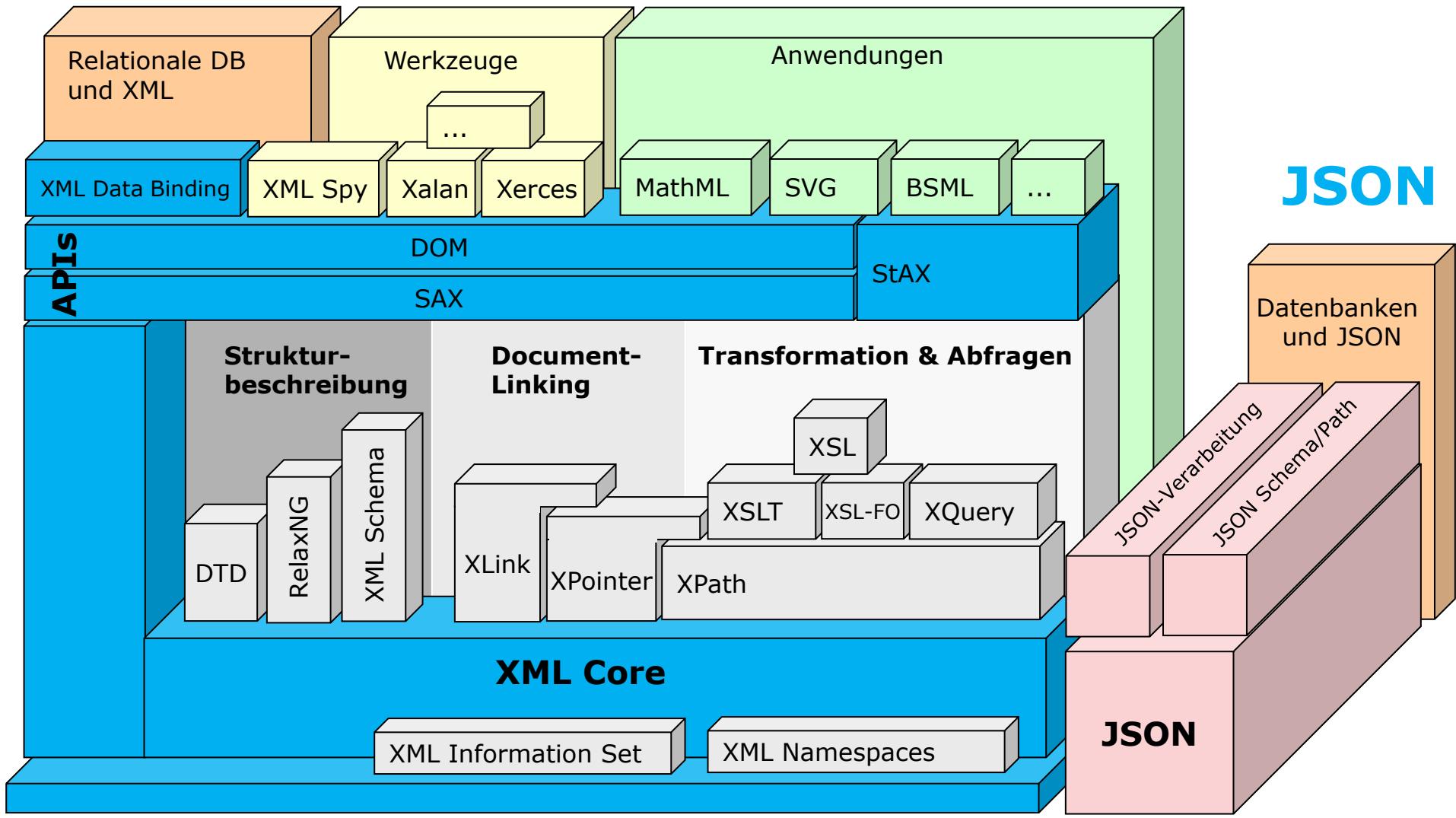
Julian Haslinger

<?xml?>

---

- Was ist eine **generische Auszeichnungssprache**?
- Grundbausteine und **Syntaxregeln** von XML anwenden
- **Wohlgeformtheit** und **Gültigkeit** von XML-Dokumenten unterscheiden
- **Namensräume** verstehen und anwenden
- **Dokumentenmodell** definieren:
  - DTD und XML Schema lesen und erstellen
  - Vorteile von XML Schema gegenüber DTD erklären
- XML-Dokumente verarbeiten:
  - **Anfragen** erstellen – XPath, XQuery
  - **Transformationen** definieren - XSLT (XSLT-Prozessoren)
  - **Verarbeitung** programmieren - XML-Parser (SAX, DOM, XML Data Binding)
- XML-Dokumente in **relationale Datenbanken** speichern
- **JSON** als alternatives Format

# XML-Technologiefamilie



# Was sollten Sie mitnehmen?

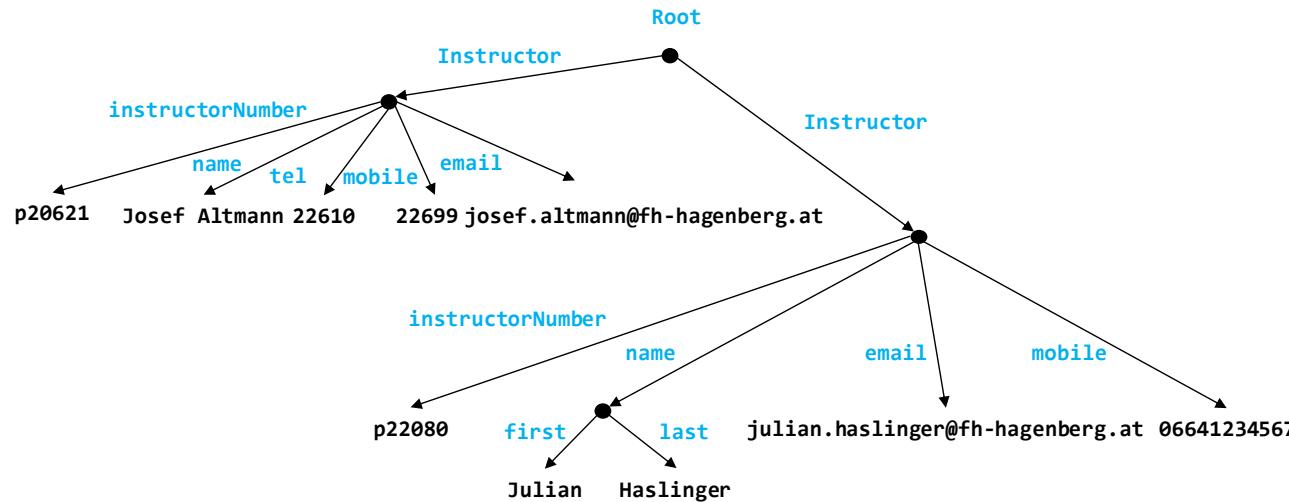
Unstrukturiert  
(Text)

Semistrukturiert  
(formatierter Text)

(stark) Strukturiert  
(Datenbanken)

- Wie können semistrukturierte Daten beschrieben/modelliert werden?

```
p20621, Josef Altmann, 22610, 22699, josef.altmann@fh-hagenberg.at
p22080, Julian, Haslinger, julian.Haslinger@fh-hagenberg.at, 06641234567
```



# Vorlesungsklausur

- 60 Minuten
- schriftlich
- ohne Unterlagen
  - (Ausgedruckte) Quick References sind jedoch erlaubt

Alles Gute für die Prüfung!

# Merkmal – Wohlgeformtheit

## ■ Wohlgeformtheit \*

- Es existiert genau ein Wurzelement
- Jedes Start-Tag muss ein dazugehöriges End-Tag besitzen
- Tags dürfen einander nicht überschneiden
- Attributwerte müssen in Anführungszeichen stehen (paarweise "... " oder '...')
- Element- und Attributbezeichner müssen XML-Namen sein (Namenskonvention)
- XML ist case-sensitive (SE != se)
- Ein Element darf nicht zwei Attribute mit gleichem Namen besitzen
- Kommentare dürfen nicht innerhalb Tags stehen
- Reservierte Zeichen < und & dürfen nicht innerhalb von Elementinhalten oder Attributwerten auftreten (müssen ggf. maskiert werden)
- ... es gibt noch mehr

```
<CourseCatalog year="2018" term="winter" campus="Hagenberg">
 <DegreeProgramme name="Software Engineering" abbreviation="SE">
 <Course>
 <Title>Introduction to semi-structured data models and XML</Title>
 <CourseType type="Lecture"/>
 </Course>
 <Course>
 <Title>Intercultural Communications</Title>
 <CourseType type="Training"/>
 </Course>
 </DegreeProgramme>
</CourseCatalog>
```

\* grundlegende syntaktische Korrektheit,  
unabhängig vom anwendungsspezifischen Einsatz

# Element vs. Attribut

## Entwurfsentscheidungen

- keine allgemeingültige Antwort, aber Anhaltspunkte
  - Element muss verwendet werden, wenn
    - Inhalt weiter strukturiert werden soll
    - Reihenfolge relevant ist (bei Attributen beliebig!)
    - Elemente mehrmals vorkommen sollen (Attribut kann pro Element nur einmal vorkommen!)
  - Attribut muss verwendet werden, wenn man
    - Aufzählungstyp, Vorgabewert, fixer Wert, ID/IDREF einsetzen möchte
- Faustregel
  - Attribute für einfache, unstrukturierte Zusatz- bzw. Metainformationen für Elemente geeignet
  - Alternative Bedingungen sollten durch Attributwerte repräsentiert werden und nicht durch die An- bzw. Abwesenheit von Elementen
  - Elemente sollen für die eigentlichen Daten genutzt werden oder als "künstliches" Gruppierungselement
  - Einheitliche Darstellung mit Elementen eleganter (aber speicherintensiver), Darstellung mit Attributen kompakter

# Element vs. Attribut

## Beispiele

```
<Name id="1234" creation-date="21.02.2020">
 <First>John</First>
 <Middle>Fitzgerald Johansen</Middle>
 <Last>Doe</Last>
</Name>
```

- Schlüssel vom Datensatz (**id**) und Erstellungsdatum (**creation-date**) sind Zusatz-/Meta-Information
- Nachteil: Datum unstrukturiert

```
<Price currency="EUR">
 <Amount>100</Amount>
</Price>
```

Sichtbarkeit?

```
<Price>
 <Currency>EUR</Currency>
 <Amount>100</Amount>
</Price>
```

- Sichtbarkeit von Informationen nach der Verarbeitung (bspw. Transformation) als Anhaltspunkt

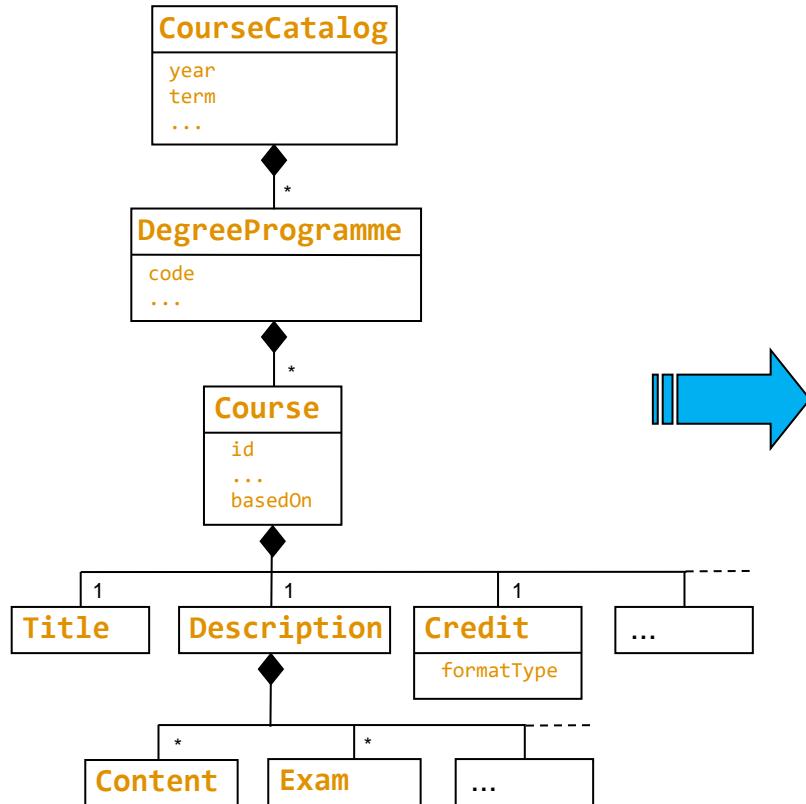
# Merkmal – Gültigkeit

- Gültigkeit (Validität)
  - XML-Dokument ist wohlgeformt und
  - entspricht einem formalen Dokumentenmodell
- Formales Dokumentenmodell definiert
  - die ihr bekannten bzw. von ihr akzeptierte Elemente (**Vokabular**) sowie
  - die Dokumentenstruktur (**Grammatik**)
- Formales Dokumentenmodell kann mit Hilfe von Schemata definiert werden, z.B. mit
  - einer sog. **Document Type Declaration** (DTD, beschränkte Möglichkeiten) oder
  - einem **XML Schema** (aktuell)

# DTD

## Beispiel: CourseCatalog.dtd

CourseCatalog.uml



CourseCatalog.dtd

```

<!-- CourseCatalog.dtd Version 1.0 -->
<!ELEMENT CourseCatalog (DegreeProgramme*)>
<!ATTLIST CourseCatalog year CDATA #REQUIRED>
<!ATTLIST CourseCatalog term (summer|winter) #REQUIRED>
...
<!ELEMENT DegreeProgramme (Course*)>
<!ATTLIST DegreeProgramme code CDATA #REQUIRED>
...
<!ELEMENT Course (Title, Description, Credit, ...)>
<!ATTLIST Course id ID #REQUIRED>
<!ATTLIST Course basedOn IDREFS #IMPLIED>
...
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Description (#PCDATA|Content|Exam)*>
<!ELEMENT Content (#PCDATA)>
<!ELEMENT Exam (#PCDATA)>
...
<!ELEMENT Credit (#PCDATA)>
<!ATTLIST Credit formatType (ECTS|CP1) "ECTS">
...

```

The corresponding DTD definition follows the UML structure. It defines the CourseCatalog element as containing multiple DegreeProgramme elements. It specifies attributes for year (required) and term (either summer or winter, required). It defines the DegreeProgramme element as containing multiple Course elements. It specifies the Course element as containing Title, Description, and Credit elements. The Description element can contain Content or Exam elements. The Credit element contains a required attribute formatType with values ECTS or CP<sup>1</sup>, defaulting to ECTS.

### Legende:

<b>XML Element</b>	: genau eines
1..*	: ein oder mehrere
*	: null oder mehrere
◆	: besteht aus

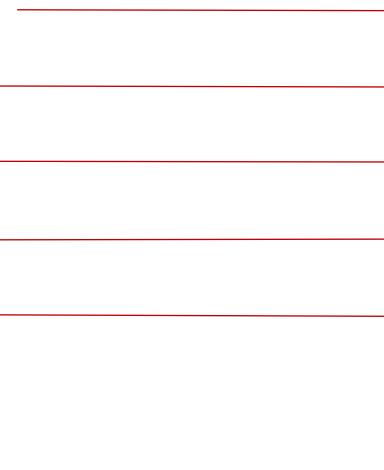
<sup>1</sup> CP ... Credit Points

# DTD

## Inhaltsmodelle von Elementen

### ■ Inhaltsmodelle:

1. Unstrukturierter Inhalt
2. Strukturierter Inhalt
3. Gemischter Inhalt
4. Leerer Inhalt
5. Beliebiger Inhalt



```
<!ELEMENT Elementname (Inhaltsmodell)>
```

Elementtypdeklaration

**Inhaltsmodell** legt den erlaubten Inhalt eines Elements fest.

# Entities

## Überblick

- Referenzierbare, mit Namen versehene Textteile
  - eines XML-Instanzdokuments
  - oder einer DTD
- Zweck: Zeichen-/Textersetzung, Modularisierung
- Verarbeitung: Referenzen werden beim Parse-Vorgang expandiert

