Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Datenbanksysteme I

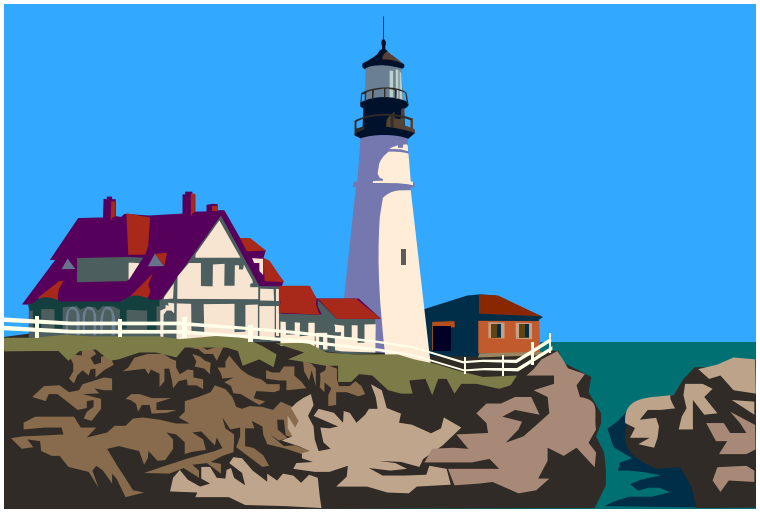
XML & Datenbanken

30.1.2008

Felix Naumann

Überblick

- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
- Speicherung von XML

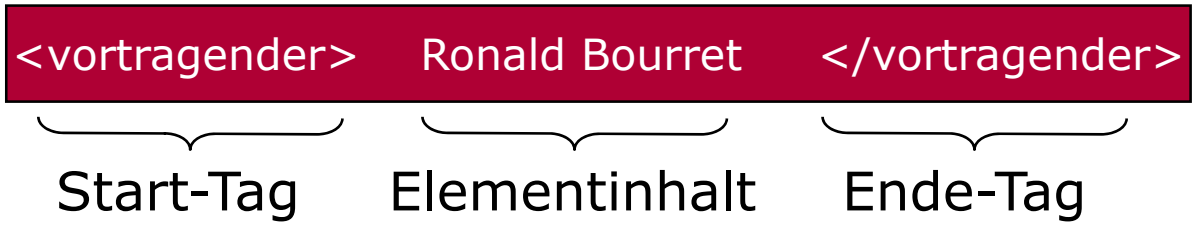


Motivation

- XML - EXtensible Markup Language
- mark up – ursprünglich aus dem Verlagswesen, Anweisungen an den Setzer
- Daten und Informationen über die Daten (Struktur/Metadaten) im gleichen Dokument
- Durch das World Wide Web Consortium (W3C) entwickelt
- Gut lesbar (human readable)
- Häufig eingesetztes Austauschformat

Syntax von Elementen

- „Grundbausteine“ eines XML-Dokumentes
- Ein Element besteht aus:
  - Start-Tag
  - Ende-Tag und
  - Elementinhalt
- Beispiel:



Syntax von Elementen

Leere Elemente:

```
<koordinaten/>
<koordinaten></koordinaten>
```

Schachtelung:

```
<vortragender>
        <name>Bourret</name>
        <vorname>Ronald</vorname>
</vortragender>
```

Start-Tag

Elementinhalt

Ende-Tag

Klassifikation von XML-Dokumenten

- Datenzentrierte Dokumente
  - strukturiert, regulär
  - Beispiele: Produktkataloge, Bestellungen, Rechnungen
- Dokumentzentrierte Dokumente
  - unstrukturiert, irregulär
  - Beispiele: wissenschaftliche Artikel, Bücher, E-Mails, Webseiten
- Semistrukturierte Dokumente
  - datenzentrierte und dokumentzentrierte Anteile
  - Beispiele: Veröffentlichungen, Amazon

```
<order>
  <customer>Meyer</customer>
  <position>
    <isbn>1-234-56789-0</isbn>
    <number>2</number>
    <price currency=„Euro“>30.00</price>
  </position>
</order>
```

```
<content>
XML builds on the principles of two existing
 languages, <emph>HTML</emph> and
<emph>SGML</emph> to create a simple
mechanism ..
The generalized markup concept ..
</content>
```

```
<book>
  <author>Neil Bradley</author>
  <title>XML companion</title>
  <isbn>1-234-56789-0</isbn>
  <content>
    XML builds on the principles of two existing
    languages, <emph>HTML</emph> and ..
  </content>
</book>
```

Datenzentriertes XML-Dokument


```
<?xml version="1.0" encoding="UTF-8"?>
<rechnung kundennummer="k333063143">
    <monatspreis>0,00</monatspreis>
    <einzelverbindungsnachweis>
        <verbindung>
            <datum>26.2.</datum>
            <zeit>19:47</zeit>
            <nummer>200xxxx</nummer>
            <einzelpreis waehrung="Euro">0,66</einzelpreis>
        </verbindung>
        <verbindung>
            <datum>27.2.</datum>
            <zeit>19:06</zeit>
            <nummer>200xxxx</nummer>
            <einzelpreis waehrung="Euro">0.46</einzelpreis>
        </verbindung>
        <verbindungskosten_gesamt waehrung="Euro">2.19</verbindungskosten_gesamt>
    </einzelverbindungsnachweis>
</rechnung>
```

XML-Dokument, Eigenschaften

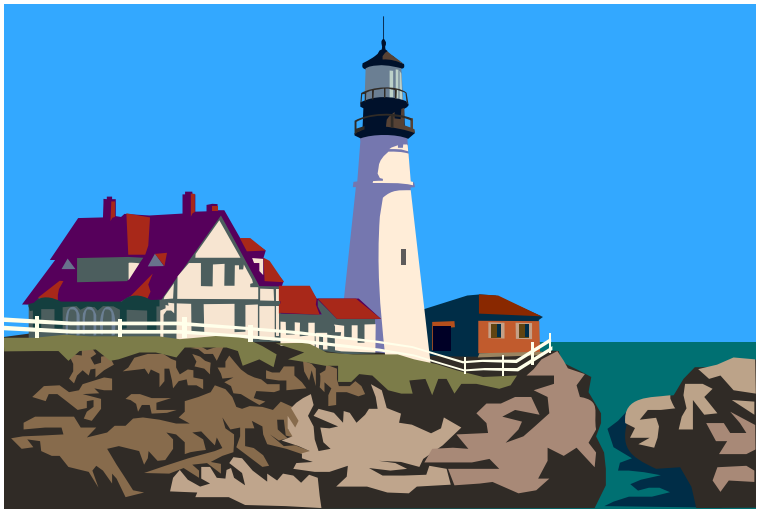
- Selbstbeschreibend
  - XML-Dokumente enthalten Daten und Struktur über die Daten in einem Dokument.
- Irregulär (semistrukturiert)
  - Alle Dokumente sind unterschiedlich strukturiert
- Ungetypt
  - Informationen im XML-Dokument haben keinen oder einen wechselnden Datentyp
- Dokumentzentriert
  - XML-Dokumente enthalten große Anteile von Volltext

XML vs. Datenbanken

- XML & Datenbanken
  - Welche Speicherungstechniken für XML-Dokumente?
  - Welche Indizierung für gespeicherte Dokumente?
  - Welche Anfragesprache und Updatesprache?
  - Abbildung von XML zum relationalen Modell
- Datenbanken & XML
  - Neue Anforderungen an DBMS
  - XML Erweiterungen in kommerziellen DBMS
  - Abbildung von Relationen auf das XML Datenmodell

Überblick

- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
- Speicherung von XML



XML-Prozessoren

- Inhalt eines XML-Dokumentes wird für eine Anwendung verfügbar.
- Standardisierte Schnittstellen für zahlreiche Programmiersprachen
  - Java, Python, C, C++, ...
- Zwei prominente Vertreter:
  - Ereignis-orientiert: SAX
  - Baum-orientiert: DOM

SAX - Simple API for XML

- Ereignisorientierte Verarbeitung
- Vorgehensweise:
  - XML-Engine liest sequentiell den Eingabestrom (das Dokument) und
  - ruft Callback-Methoden bei Eintreten von Ereignissen auf.
    - z.B. wenn ein Begin oder End-Tag abgearbeitet wird
  - Anwendung kann auf diese Ereignisse reagieren oder sie ignorieren.
  - Anwendungsprogrammierer muss "Event-Handler" für die Callback-Methoden implementieren, an deren Ereignissen er interessiert ist.
  - SAX ist zustandslos.

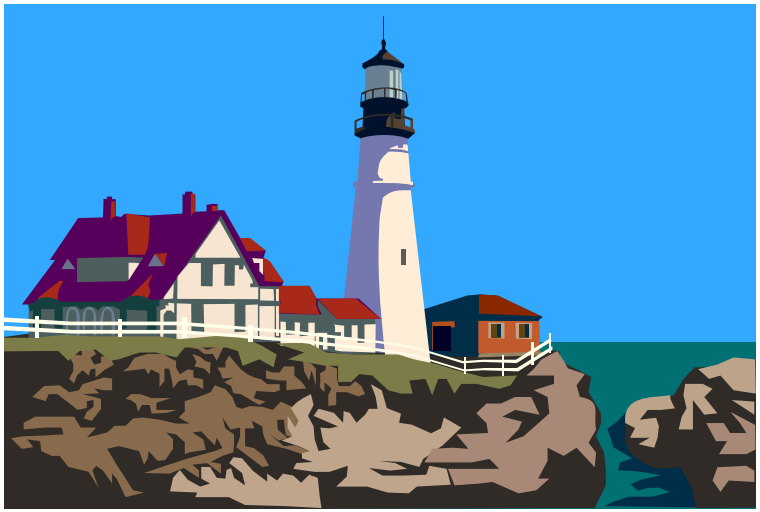
DOM – Document Object Model

- DOM
  - Beschreibt Schnittstellen (APIs) zum Zugriff auf XML-Dokumente und zur Veränderung von Struktur und Inhalten
  - Definiert nicht die zugrunde liegende Implementierung und Speicherung der XML-Dokumente
- Objektorientierte Sicht auf XML-Dokumente
  - XML-Dokumente intern als Bäume repräsentiert
  - Unterschiedliche Knotentypen: Element, Attribut, etc.
  - Methoden zum Traversieren und Manipulieren der Baumstruktur
- Es gibt zahlreiche DOM-Implementierungen, z.B.
  - Java (+ XML-Parser)
  - JavaScript und Web-Browser
  - C++-Bibliotheken

Quelle: Can Türker

Überblick

- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
- Speicherung von XML



DTDs


```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hotel SYSTEM "hotel_dt.dtd">
<hotel id="id001"
     url="http://www.hotel-huebner.de">
   <name>Strand Hotel Huebner</name>
   <adresse>
      <plz>18119</plz>
      <ort>Rostock-Warnemuende</ort>
      ...
   </adresse>
   <hausbeschreibung> Direkt an der
Promenade von Warnemuende befindet sich das
Strand-Hotel Huebner mit Blick auf Leuchtturm,
Hafeneinfahrt und Strand.
   </hausbeschreibung>
   <preise waehrung=„Euro">
      <einzelzimmer>ab 78,-</></>
...
</hotel>
```

```
<!-- Hotel DTD-->
<!ELEMENT hotel (name, kategorie?,
adresse, hausbeschreibung, preise*>
<!ATTLIST hotel id ID #REQUIRED
      url CDATA #REQUIRED>
<!ELEMENT  name (#PCDATA)>
<!ELEMENT  kategorie (#PCDATA)>
<!ELEMENT  adresse (plz, ort, strasse,
hausnummer, telefon, fax?, e-mail?)>
<!ELEMENT  plz (#PCDATA)>
...
<!ELEMENT hausbeschreibung (#PCDATA)>
<!ELEMENT preise (#PCDATA | einzelzimmer
|  doppelzimmer | appartment)*>
<!ATTLIST preise waehrung CDATA
#REQUIRED>
<!ELEMENT einzelzimmer (#PCDATA)>
...
```

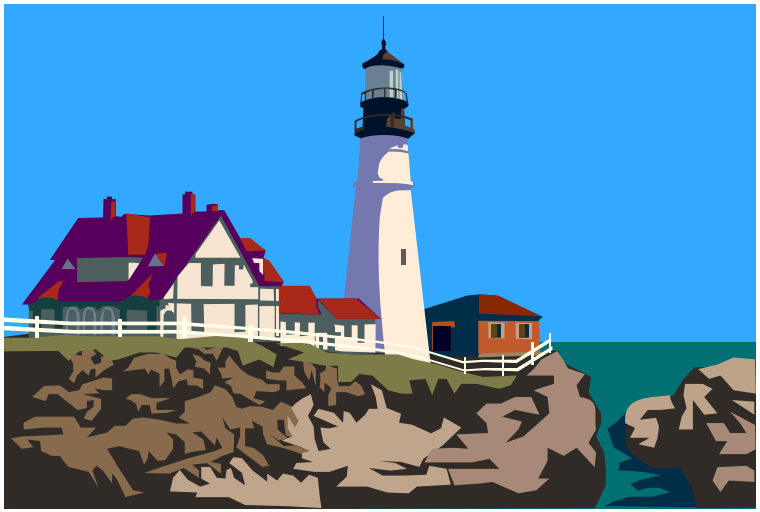
XML Schema

Wesentlich umfangreichere Darstellungsmöglichkeiten als DTDs

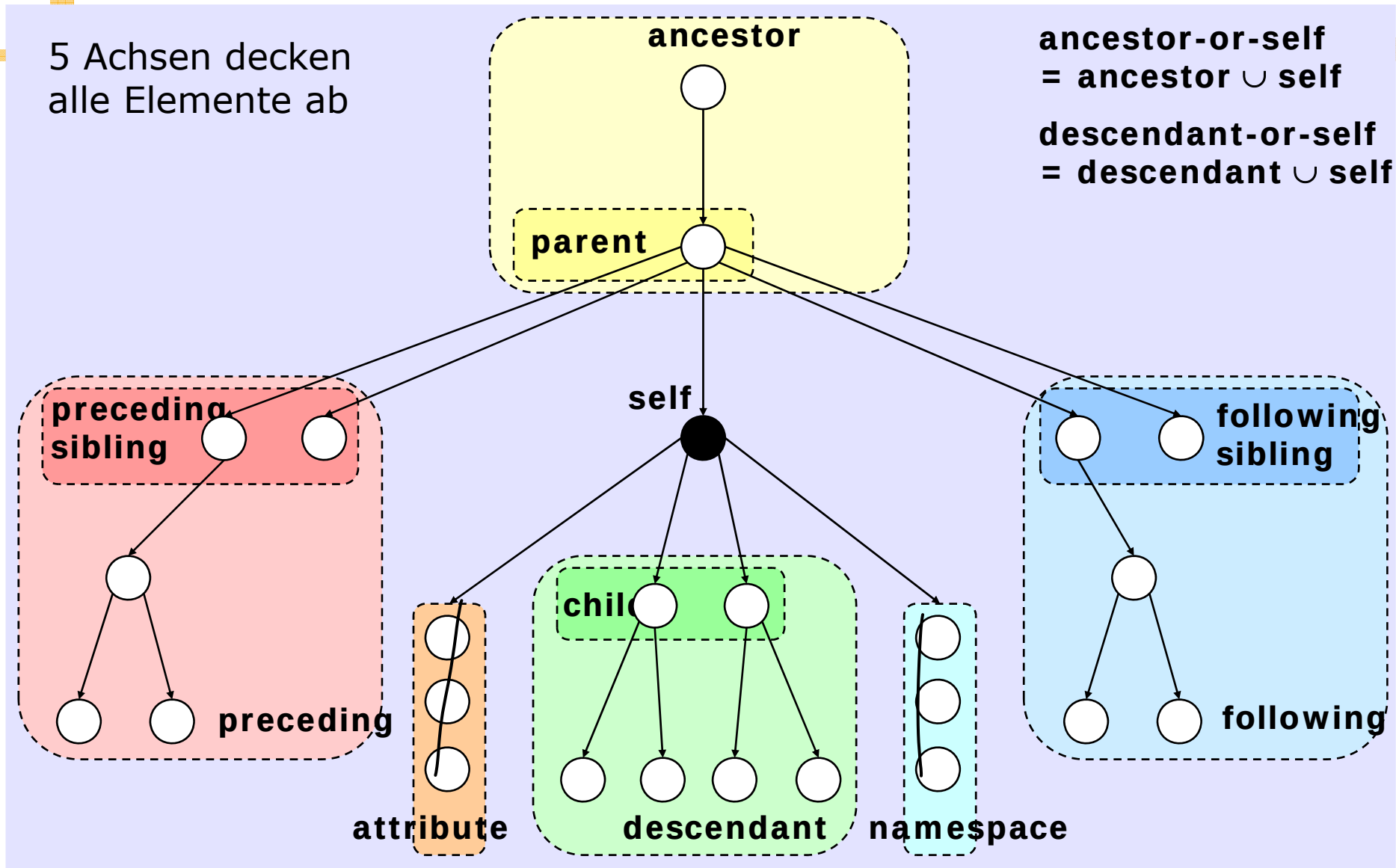
- Vielfältige vordefinierte Datentypen
- Möglichkeit zur Definition eigener Datentypen
- Umfangreiche Darstellungsmöglichkeiten
- Integritätsbedingungen
  - Unique, Key, Foreign key
- Dargestellt in XML-Syntax
  - Leichter parsebar

Überblick

- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
- Speicherung von XML



XPath – Navigationsachsen



Beispiele für XPath-Anfragen

- **`/bookstore/book/@genre`**
  - gibt das Attribut 'Genre' aller Bücher aus
- **`/bookstore/book[author/name='Plato']`**
  - gibt alle Bücher aus, die vom Autor 'Plato' stammen
- **`//author[first-name='Herman']/last-name`**
  - gibt den Nachnamen aller Autoren aus, deren Vorname 'Herman' ist
- **`/bookstore/book[author/first-name='Benjamin']/price`**
  - liefert den Preis für alle Bücher, die mind. einen Autor mit dem Vornamen 'Benjamin' haben
- **`//book[contains(title, 'XML')]/title`**
  - selektiert alle Bücher, die den Begriff 'XML' im Titel enthalten

XQuery

- Basiert auf XPath
- Ähnlichkeit zu SQL
- Basiskonstrukt: FLWR-Ausdruck,
  - **`for/let`**: geordnete Liste von Elementen
  - **`where`**: eingeschränkte Liste von Elementen
  - **`return`**: Ergebniskonstruktion, Instanzen des XML Query data model
- Ausdrücke werden aus anderen Ausdrücken zusammengesetzt.
- Datenmodell ist geordneter Wald
  - Flach
  - Geordnet
  - Nicht duplikatfrei

Beispiel


```
<hotel name="Hotel Neptun">
      <zimmertyp typ="EZ" preis="180" waehrung="DM"/>
      <foto href="neptun01.jpeg"/>
</hotel>
<hotel name="Hotel Huebner">
      <zimmertyp typ="EZ" preis="150" waehrung="DM"/>
      <zimmertyp typ="DZ" preis="180" waehrung="DM"/>
</hotel>
<hotel name="Pension Draeger">
      <foto href="bild-pd01.jpeg"/>
      <foto href="bild-pd02.jpeg"/>
</hotel>
```

```
for $hotel in //hotel
return $hotel/foto
```

```
<foto href="neptun01.jpeg"/>
<foto href="bild-pd01.jpeg"/>
<foto href="bild-pd02.jpeg"/>
```

Auswertung von **`return`** pro hotel, aber nicht immer mit Ergebnis.

Beispiel


```
<billighotels> {
       for $h in //hotel
              for $z in $h/zimmertyp
              where $z/@preis <= 100
              return <hotel>
                            <name>{ $h/@name }</name>
                            <preis>{ $z/@preis }</preis>
                     </hotel> }
</billighotels>
```

{ } kennzeichnet einen auszuwertenden Ausdruck

```
<billighotels>
       <hotel>
              <name>...</name>
              <preis>...</preis>
       </hotel>
       ...
</billighotels>
```

Ein **hotel**-Element pro **zimmertyp**

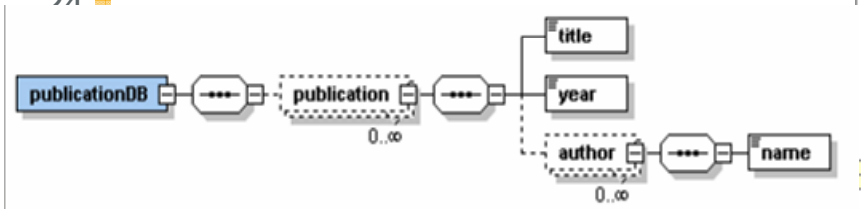
Beispiele


```
<result>
 {
   for $b in fn:doc("bib.xml")/bib/book
   where $b/publisher = "Addison-Wesley" and $b/@year > "1991"
   return <book year="{ $b/@year }">
            { $b/title }
          </book>
 }
</result>
```

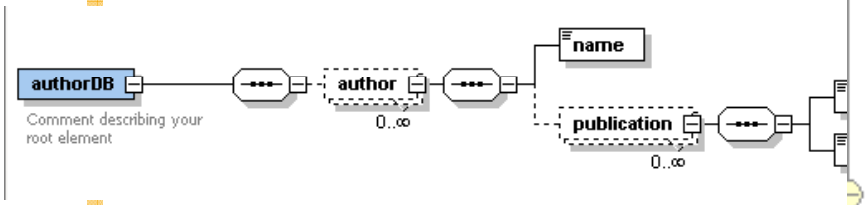
Welche Bücher sind von Addison-Wesley nach 1991 publiziert worden?

```
<result>
   <book year="1994">
      <title>TCP/IP Illustrated</title>
   </book>
   <book year="1992">
      <title>Advanced Programming in the Unix environment</title>
   </book>
</result>
```

Beispiel



Pivotisierung



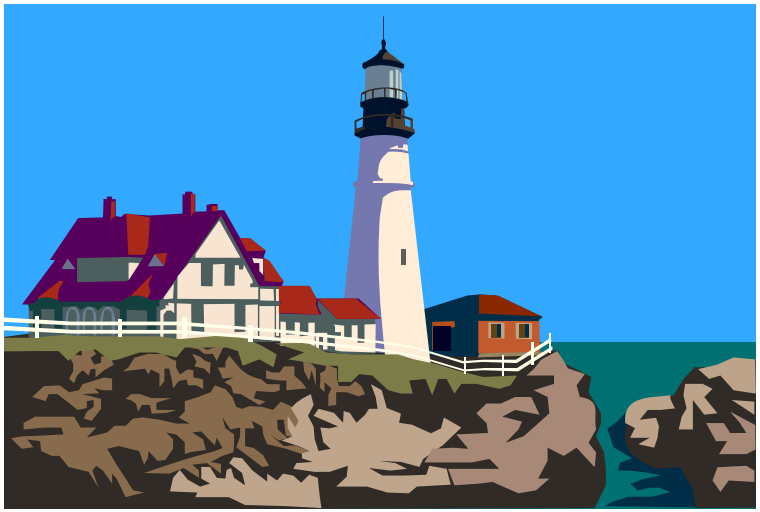

```
LET $doc0 := document("input XML file goes here")
RETURN
<authorDB>
  {
  distinct-values (
  FOR
   $x0 IN $doc0/publicationDB/publication,
   $x1 IN $x0/author
  RETURN
    <author>
      <name> { $x1/name/text() } </name>
      {
      distinct-values (
      FOR
       $x0L1 IN $doc0/publicationDB/publication,
       $x1L1 IN $x0L1/author
      WHERE
        $x1/name/text() = $x1L1/name/text()
      RETURN
        <publication>
          <title> { $x0L1/title/text() } </title>
          <year> { $x0L1/year/text() } </year>
        </publication> )
      }
    </author> )
  }
</authorDB>
```

Vergleich XQuery / SQL

XQuery	SQL
for **$k** in **/bookstore/book** return **$k**	SELECT *** FROM **bookstore**
for **$k** in **//book** return **$k**	SELECT *** FROM **bookstore**
for **$k** in **//book/title** return **$k**	SELECT **title** FROM **bookstore**
for **$k** in **//book** return **$k/title**	SELECT **title** FROM **bookstore**
for **$k** in **//book/author** return **$k/last-name**	SELECT **last-name** FROM **bookstore**
for **$k** in **/bookstore/book** where **$k/title='XML und Datenbanken'** order by **$k/author/last-name** return **$k/author**	SELECT **author** FROM **bookstore** WHERE **title='XML und Datenbanken'** ORDER BY **author.last-name**
for **$k** in **/bookstore/book** where count**($k/author) > 2** return **$k/title**	SELECT **title** FROM **bookstore** GROUP BY **title** HAVING COUNT**(author) > 2**

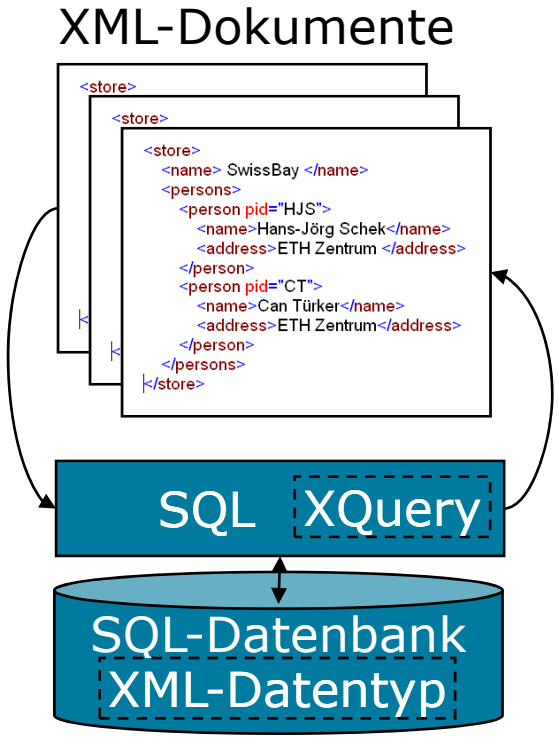
Überblick

- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
- Speicherung von XML



SQL / XML – Grundidee

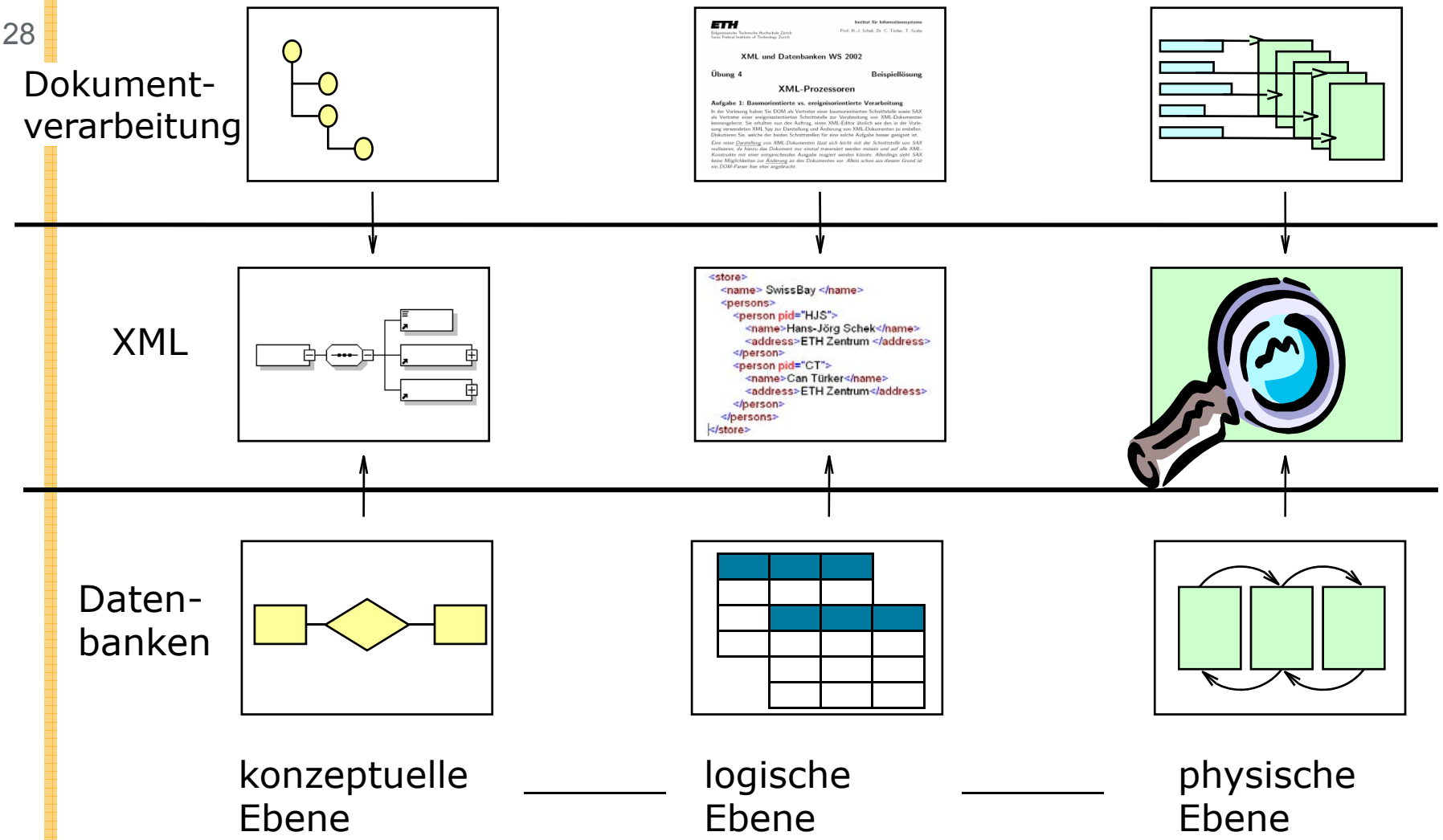
- Stellt neuen Datentyp XML mit darauf operierenden Funktionen bereit
- Definiert Abbildungen zwischen SQL und XML



Speicherung von XML-Dokumenten in der Datenbank als Wert des XML-Datentyps

Generierung von XML-Dokumenten mittels SQL/XML-Funktionen

Architektur



Beispiel


```
<hotel id=„H0001"
      url=http://www.hotel-huebner.de
      erstellt-am=„09/15/2002“
      autor=„Hans Huebner“>
    <hotelname>Hotel Huebner</hotelname>
    <kategorie>4</kategorie>
    <adresse>
        <plz>18119</plz>
        <ort>Warnemuende</ort>
        <strasse>Seestrasse</strasse>
        <nummer>12</nummer>
    </adresse>
    <telefon>0381 / 5434-0</telefon>
    <fax>0381 / 5434-444</fax>
</hotel>
```

Elemente

DocID	Elementname	ID	Vorgaenger	Ordnung	Wert
H00001	Hotel	101	⊥	1	⊥
H00001	Hotelname	102	101	1	Hotel Huebner
H00001	Kategorie	103	101	2	4
H00001	Adresse	104	101	3	⊥
H00001	PLZ	105	104	1	18119
H00001	Ort	106	104	2	Warnemuende
H00001	Strasse	107	104	3	Seestrasse
H00001	Nummer	108	104	4	12
H00001	Telefon	109	101	4	0381 / 5434-0
H00001	Fax	110	101	5	0381 / 5434-444

Abbildung von XML auf relationale Datenbanken


```
<Hotel>
        <HotelID>H0001</HotelID>
        <Name>Hotel Hübner</Name>
        <Adresse>
            <PLZ>18119</PLZ>
            <Ort>Warnemünde</Ort>
            <Strasse>Seestrasse</Strasse>
            <Nr>12</Nr>
        </Adresse>
        <Preise>
                <Einzelzimmer> 198 </Einzelzimmer>
                <Doppelzimmer> 299 </Doppelzimmer>
        </Preise>
    </Hotel>
```

Hotel:

HotelID	Hotelname	Adresse	Preise
H0001	Hotel Hübner	A0001	P0001

Adresse:

AdresseID	PLZ	Ort	Strasse	Nr
A0001	18119	Warnemünde	Seestraße	12

Preise:

PreiseID	Einzelzimmer	Doppelzimmer
P0001	198	299

- DTD/XML-Schema ist typischerweise erforderlich.
- Anfragen verwenden SQL-Funktionalität.
- RDBMS-Datentypen werden eingesetzt
- Abbildung von Kollektionstypen durch Aufteilung auf zusätzliche Relationen