Kapitel 5 – Werkzeuge: XML und XML-Datenbanken

Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Daten- und Dokumentmodelle

- Datenbankmodelle: Stärke sind strukturierte Daten, Metadaten
- Dokumentmodelle: Stärken sind unstrukturierte Daten
- Modelle für semistrukturierte Daten:
  - Markup languages: GML, Scribe, LaTeX, troff, SGML, ...
  - Hypertext: HTML; Hypermedia: HyTime, HyperWave, ...
- W3C: XML

XML – Herausforderungen

- Ursprünge im WWW
- Integration der WWW Infrastruktur mit „echten" Anwendungen
- Zunehmende Professionalisierung
- Weiterverwendbarkeit der übertragenen Information auf Benutzerseite
- Hohe Anforderungen an das Layout
- Metadaten, Zusammenhänge, Sichten
- Auffindbarkeit von Information
- Verschmelzung mit anderen Medien

XML – Entwurfskriterien

- Unterstützung eines breiten Spektrums von Anwendungen
- Modellierung beliebiger Datentypen
- Aufwärtskompatibel zu SGML
- Nicht voll abwärtskompatibel zu HTML
- Einfach genug für Massennutzung
- Vereinfachtes Parsing durch neues Konzept der „Wohlgeformtheit"
- Minimale Zahl von optionalen Merkmalen
- Keine semantischen Anteile
- Unabhängigkeit von logischer und physischer Struktur („Entitäten")

Anwendungsbereiche von XML

- **Dokument-zentriert:** tief strukturierte Dokument-Hierarchie für viele Dokumentinstanzen
  - grob-granulare Daten
  - Reihenfolge signifikant
  - Beispiele: Bücher, Email, Werbung, ...
  - Nutzung durch Personen
- **Daten-zentriert:** große Bestände in Datenbanksystemen von dort aus in ein XML-Format umwandeln
  - fein-granulare Daten
  - Reihenfolge nicht signifikant
  - Beispiele: Flugpläne, Speisekarten, ...
  - maschinelle Verarbeitung
- [**EDI**] (Electronic Data Interchange): Datenformat zur Kommunikation (Austausch von Informationen) zwischen Anwendungen

XML - Processing

- Client-basiert:
  - Quelldaten an Client
  - Processing Instructions spezifiziert Style Sheet
  - Client übernimmt Bearbeitung
  - Nachteil: Client muß XML-Kenntnisse besitzen
- Server-basiert:
  - Client beherrscht XML nicht
  - Anfrage, welche Formate Client verarbeiten kann
  - Server schickt Daten z.B. als HTML
  - Nachteil: Last auf Server-Seite

XML-Dokumente

- Beispiel

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE Memo SYSTEM "http://www.fck.de/DTDs/memo.dtd">

<Memo>
        <Von>
                <Name>Andi Brehme</Name>
                <Email>Andi Brehme@fck.de</Email>
        </Von>
        <An>
                <Name>Vorstand</Name>
                <Email>Vorstand@fck.de</Email>
        </An>
                <Betreff Dringlichkeit="hoch">Spielereinkauf</Betreff>
        <Inhalt>
                <Absatz>Vorstand, ich glaube wir <Betonung>muessen
                </Betonung> unbedingt noch einige neue Spieler
                kaufen, sonst wird das wohl nichts mit der Meisterschaft.
                </Absatz>
        </Inhalt>
</Memo>
```


XML-Dokumente - Aufbau

- Prolog
  - XML-Deklaration
    - Versionsangabe
    - Kodierungsdeklaration (optional)
    - standalone-Angabe (optional)
  - Dokumenttyp-Deklaration
    - Angabe der verwendeten DTD
- XML-Dokument besteht aus Markup-Elementen
- Markup-Element
  - besteht aus Tag-Paar (Begin-Tag, End-Tag)
    - Bsp: <Name>Harry Koch</Name>
      <LeeresElement/>
  - kann leer sein
  - Schachtelung möglich
  - enthält Zeichendaten
  - Elementnamen werden bzgl. Gross-/Kleinschreibung unterschieden
  - können näher durch Attribute beschrieben werden

XML-Dokumente - Aufbau

- Attribute
  - haben Namen (innerhalb des Elements eindeutig) und Wert
  - 3 Typen
    - Zeichendaten
    - Aufzählungen
    - über Schlüsselwörter, wie ID oder IDREF, ausgezeichnete Zeichendaten
- es ergeben sich verschiedene Möglichkeiten der Darstellung derselben Information
  - Unterelemente:
```
<Spieler>
    <Nummer>24</Nummer>
    <Tore>5</Tore>
    <Name>Harry Koch</Name>
</Spieler>
```
  - Attribute: `<Spieler Nummer="24" Tore =„5" Name="Harry Koch"/>`
  - Gemischt: `<Spieler Nummer="24" Tore="5">Harry Koch</Spieler>`

XML-Dokument

- logische Struktur
  - wird durch DTD festgelegt
  - Deklarationen, Elemente, Attribute, Kommentare ...
- physische Struktur
  - Entities (Makro-, Datei-Include-Mechanismus)
  - auch nicht-XML-Daten, z.B. GIF-, JPEG-Bilder
  - wiederverwendbar

XML - Entities

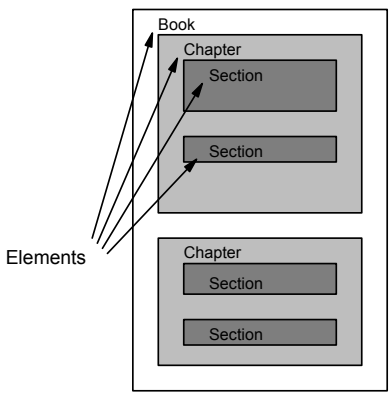
- Entities
  - bilden physische Dokumentstruktur
  - Name
  - Inhalt: parsed, unparsed
  - Unterscheidung: extern, intern, allgemein, Dokument, ...
  - Entity-Referenzen
    - können auch (nicht-rekursiv) in Entity-Deklarationen auftreten
  - Beispiele:

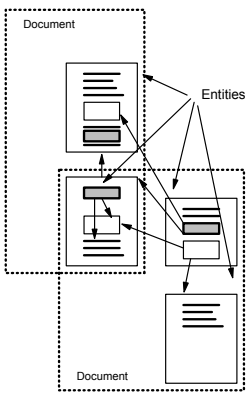
```
<!ENTITY FckLogo
SYSTEM http://www.fck.de/wapppen.gif
NDATA GIF>

<!ENTITY FCK "1. FC Kaiserslautern">

<Text>Andi Brehme ist Trainer des &FCK;.</Text>
```


Logische vs. physische Struktur





logische Struktur

physische Struktur

DTD – Document Type Definition

- dient struktureller Validierung
  - **well-formed**: konform zu XML-Standard
  - **valid**: konform zu angegebener DTD
- Beispiel

```
<!DOCTYPE Memo[
<!ELEMENT Memo (Von, An, Betreff, Inhalt)>
<!ELEMENT Von (Name, Email)>
<!ELEMENT An (Name, Email)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Spieler ANY>
<!ATTLIST Spieler
          Dringlichkeit (niedrig| mittel| hoch)
          "niedrig"
<!ELEMENT Inhalt (Absatz+)>
<!ELEMENT Absatz (#PCDATA|Betonung)*>
<!ELEMENT Betonung (#PCDATA)>
]>
```


DTD - Syntax

- kann extern

```
<!DOCTYPE Memo
SYSTEM "http://www.fck.de/DTDs/memo.dtd">
```

  oder intern deklariert sein
- Inhaltsspezifikationen für Elemente
  - Text (parsed character data): <!ELEMENT Email (#PCDATA)>
  - leerer Inhalt: <!ELEMENT Email EMPTY>
  - Beliebiger Inhalt: <!ELEMENT Email ANY>
  - Kombinierter Inhalt:
    <!ELEMENT Email (#PCDATA | Element1 | Element2)*>
  - Elementinhalt: <!ELEMENT Email (Element1, (Element2, Element3)+, Element4?)>
    (Indikatoren für das Auftreten: ?, *, +)

Deklaration von Attributlisten

```
<!ATTLIST element attr_name attr_typ default_wert ... >
```

- Definition von Attributen für gegebenen Elementtyp
- Typbeschränkungen für Attribute
- Vorgabewerte für Attribute
- Informationen, ob Attribut vorkommen muß oder wie XML-Prozessor bei fehlendem Attribut reagieren soll
- `#REQUIRED`, notwendig, Attribut muß immer angegeben werden
- `#IMPLIED`, impliziert, es gibt keinen Vorgabewert, Prozessor zeigt aber definiertes Verhalten (z.B. in Abhängigkeit von Nutzereinstellungen)
- `#FIXED`, fest, alle Instanzen müssen diesen Vorgabe-Wert haben

XML-Attribut-Typen

- Zeichenkettentyp (CDATA)
- Aufzählungstyp (enumerated), Liste von Werten, von denen einer ausgewählt werden muß
- Menge von Token-Typen z.B.
  - ID/IDREF, eindeutige Identifikation eines Element
  - ENTITY, in DTD deklarierte Entity kann benutzt werden
  - NOTATION, Elementinhalt wird abhängig von diesem Attribut interpretiert, muß zuvor mit <!NOTATION> deklariert sein
- Beispiele:

```
<Spieler Nummer="24" Tore="5">Harry Koch</Spieler>
<!ATTLIST Spieler
          Nummer CDATA    #REQUIRED
          Tore   CDATA    "0">

<!ATTLIST image format NOTATION ( tex | jpeg )>
<image format="tex"> TeX-Markup ... </image>
```

XML-Schema

- Erweiterung der Möglichkeiten zur Beschreibung von Dokumenttypen im Vergleich zu DTD
  - Beschreibung von Dokumenten-Modellen
    - Kardinalitätsrestriktionen
    - Vererbung
  - Datentypen
    - vielfältige vordefinierte Datentypen, z. B. Integer, Date
    - benutzerdefinierte, komplexe Datentypen
    - Constraints
  - Referenzen
    - bisher 'nur' Wertübereinstimmung von ID- und IDREF(S)-Attributen
    - nun getypte Referenzen auf bestimmte Elemente/Attribute

XML-DTD: Beispiel

```
<!ELEMENT description
((content, date*)+, author?)>
<!ELEMENT content (#PCDATA)>
<!ATTLIST content info CDATA #REQUIRED>
<!ELEMENT date (#PCDATA)>
<!ELEMENT author (#PCDATA)>
```

XML-Schema: Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
    <schema xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="description" type="mycontent"/>
    <complexType name="mycontent">
      <sequence>
            <sequence maxOccurs="unbounded">`
                  <element name="content" type="mytype"/>
                  <element name="date" type="date" minOccurs="0"
                        maxOccurs="unbounded"/>`
            </sequence>
        <element name="author" type="string" minOccurs="0"/>
      </sequence>
    </complexType>
    <complexType name="mytype">
      <simpleContent>
            <extension base="string">
                  <attribute name="info" type="string" use="required"/>
            </extension>
      </simpleContent>
    </complexType>
</schema>
```


XLink

- XLink: XML Linking Language
  - Verweise in XML-Dokumenten
  - Attribute von XLink-Elementen: **type (simple, extended, ...), href** (URI-Referenz oder XPointer)**, role, title, show (new, replace, embed, undefined), actuate (onLoad, onRequest, undefined), from, to**
  - Arten von Verweisen
    - outbound: lokale Start- , entfernte End-Ressource
    - inbound: entfernte Start- , lokale End-Ressource
    - third-party: entfernte Start- , entfernte End-Ressource
    - es können mehrere Ressourcen beteiligt sein
    - multi-direktionale Verweise
    - Verweise auf ausschließlich lesbare Ressourcen

XLink - Beispiel

- in der DTD:

```
<!ELEMENT Spieler ANY>
<!ATTLIST Spieler
                xlink:type  (simple)   #FIXED     "simple"
                xlink:href  CDATA   #REQUIRED
                xlink:role   CDATA   #IMPLIED
                xlink:title   CDATA    #IMPLIED
                xlink:show    (new|embed|replace)   "replace"
                xlink:actuate (onLoad|onRequest)   "onRequest"
        >
```

- in der Dokumentinstanz:

```
<Spieler xlink:href="http://www.fck.de/Spielerliste.xml"
                xlink:role="playerlist"
                xlink:title="Liste aller FCK-Spieler"
                xlink:show="new"
                xlink:actuate="onRequest">
                Hier geht es zu einer Liste aller FCK-Spieler.
</Spieler>
```


XSL

- **XSL**: *Extensible Stylesheet Language*
  - darstellungsunabhängiges Markup von XML
  - medienunabhängige Präsentation
  - **XSLT**: *XSL Transformation Language*
    - Formatvorlage (stylesheet)
    - Transformationsregeln
      - jeweils bestehend aus einem Pattern und einem Template
    - Ursprungsbaum
    - Ergebnisbaum

XSL - Beispielanwendung

- ein Fragment eines XML-Dokuments:

```
<Inhalt>
<Absatz>XSLT <Fremdwort> (XSL Transformation Language) </Fremdwort>
ist eine <Betonung> phantastische</Betonung> Sprache um XML-Dokumente in
XHTML <Fremdwort> (Extensible HTML) </Fremdwort> zu konvertieren.
</Absatz>
</Inhalt>
```

- das gewünschte Ergebnis-Dokument:

```
<html>
    <head>
        <title>Ein XSLT-Beispiel</title>
    </head>
    <body>
        XSLT <i>(XSL Transformation Language)</i> ist eine
        <b>phantastische</b> Sprache um XML-Dokumente in XHTML
        <i>(Extensible HTML)</i> zu konvertieren.
    </body>
</html>
```

- Präsentation:
  XSLT *(XSL Transformation Language)* ist eine **phantastische** Sprache um XML-Dokumente in XHTML *(Extensible HTML)* zu konvertieren.

XSL - Beispiel

- XSLT-Stylesheet:

```
<xsl:stylesheet     xmlns:xsl=http://w3.org/XSL/Transform/1.0
                    xmlns=http://w3.org/TR/xhtml1
                    indent-rules="yes">
<!-- Regel 1 -->    <xsl:template match="/">
                        <html>
                        <head>
                            <title>Ein XSLT-Beispiel</title>
                        </head>
                        <body>
                            <xsl:apply-templates/>
                        </body>
                        </html>
                        </xsl:template>
<!-- Regel 2 -->    <xsl:template match="Absatz">
                        <xsl:apply-templates/>
                    </xsl:template>
<!-- Regel 3 -->    <xsl:template match="Betonung">
                        <b><xsl:apply-templates/></b>
                    </xsl:template>
<!-- Regel 4 -->    <xsl:template match="Fremdwort">
                        <i><xsl:apply-templates/></i>
                    </xsl:template>
```

XPATH

- **XPath**: *XML Path Language*
  - Beschreibung der *Pattern* in XSLT
  - Angabe der Position eines Knotens
    - absolut bzgl. Wurzelknoten
    - relativ bzgl. eines Kontextknotens
  - Pfadausdruck
    - *steps* (durch "/" getrennt)*: axis::node test[predicate]*
      - *basis*
        - *axis (descendant, child, following, following-sibling, ancestor, parent, preciding, preciding-sibling, self, descendant-or-self, ancestor-or-self)*
      - *node test*
    - optionale Liste von predicates
  - Beispiel:
    id("Mannschaft")/descendant::Spieler/
    child::Name[position()=last()]
  - abgekürzte Notation:
    id("Mannschaft")//Spieler/
    Name[position()=last()]

XQuery

- Motivation
  - Anfragen über XML
    - physisches XML-Dokument (z.B. als Datei gespeichert)
    - logische XML-Sicht (z.B. über relationaler Datenbank)
  - soll sowohl Daten- als auch Dokument-zentrierte Sicht unterstützen
- Grundlagen
  - funktionale Sprache
    - basiert auf Komposition von funktionalen Ausdrücken
  - Erweiterung von XPath
    - XPath 2.0 wird von XQuery-, XSLT-WGs weiterentwickelt
  - XQuery-Datenmodell erweitert XPath 1.0 DM
    - collections of documents
    - XQuery ist abgeschlossen bzgl. DM

XQuery – Grundlegende Konzepte

- Wichtige Erweiterungen
  - FLWR-expressions (For-Let-Where-Returns)
    - Iteration
    - Binden von Variablen
  - Konstruktoren
    - XML Syntax
- Beispiel 1:

```
FOR $b IN document("bib.xml")//book
WHERE $b/publisher = "Morgan Kaufmann"
  AND $b/year = "1998 "
RETURN $b/title
```


XQuery – Weitere Beispiele

- Beispiel 2:

```
FOR $p IN distinct(document("bib.xml")//publisher)
LET $a := avg(document("bib.xml")
          /book[publisher = $p]/price)
RETURN
  <publisher>
          <name> { $p/text() } </name>
          <avgprice> { $a } </avgprice>
  </publisher>
```

- Beispiel 3:

```
<author_list>
  { FOR $a IN distinct(document("bib.xml")//author)
  RETURN
          <author>
                  <name> { $a/text() } </name>
                      { FOR $b IN document("bib.xml")//
                            book[author = $a]
                            RETURN $b/title }
          </author> }
</author_list>
```

Anwendungsprogrammierung mit XML

- Anwendungsschnittstellen zum Verarbeiten von XML Daten/Dokumenten
  - **Parsen** von XML um relevante Informationen zu extrahieren
  - Erzeugen von XML
    - als Zeichenkette
    - Aufbau einer internen XML Dokumentrepräsentation, anschließendes **Serialisieren**
  - *Simple API for XML (SAX)*
    - "Push"-parser (ereignisbasiert)
    - Parser benachrichtigt die Anwendung bei Erreichen/Erkennen von Dokumentbestandteilen (Elemente/Fragmente, etc.)
    - Benachrichtigungen erfolgen in der durch das serlialisierte Dokument vorgegebenen Reihenfolge
    - Haupspeichereffizientes Verfahren, vorteilhaft für große Dokumente
  - *Document Object Model (DOM)*
    - Parser erzeugt Haupspeicherdarstellung des Dokuments (*parse tree*)
    - DOM spezifiziert die versch. Arten von Objekten im parse tree, Eigenschaften, Operationen
      - Programmiersprachenunabhängig
    - Anbindung an spez. Programmiersprachen definiert (z.B., Java)

DOM

- DOM-Struktur ist eine Baumstruktur mit untersch. Knotentypen

Node type	Contains
Document	Element (maximum of one), ProcessingInstruction, Comment, DocumentType
DocumentFragment	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
DocumentType	no children
EntityReference	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
Element	Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
Attr	Text, EntityReference
ProcessingInstruction	no children
Comment	no children
Text	no children
CDATASection	no children
Entity	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
Notation	no children

Speicherung von XML

- Anforderungen an die Speicherung
- Motivation der Existenz verschiedener Varianten
- Vorstellung verschiedener Speicherungsverfahren
  - Speicherung und Indizierung
  - Speicherung der Graphstruktur
  - strukturierte Speicherung in Datenbanken
- Hybride Verfahren

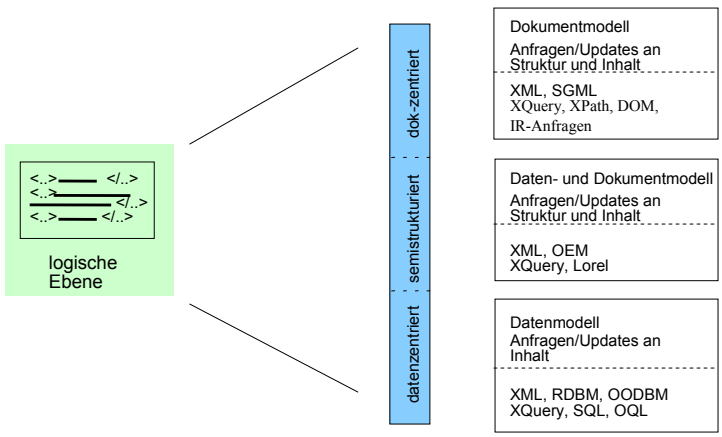
Anforderungen an die Speicherung

- Effektive Speicherung
- Effizienter Zugriff auf XML-Dokumente oder Teile davon
  - Transaktionsverwaltung
  - Unterstützung von XPath und XQuery
  - Unterstützung von SAX und DOM für Anwendungen
- Wiederherstellbarkeit der Dokumente (oder der Informationen aus den Dokumenten)

Bedeutung des Dokumentcharakters

- XML-Dokumente können die ganze Bandbreite von Daten bis zu Volltextdokumenten einnehmen
  - (dokumentzentriert, semistrukturiert, datenzentriert)
- Entsprechend eignen sich auch Speicherungsverfahren von der *Dokumentenverarbeitung (Information Retrieval)* bis zur *Datenbanktechnologie*
- Weiterhin: Neuentwicklung von Methoden
- Keine optimale Lösung für alle Anwendungen, Dokumentcharakter spielt entscheidende Rolle

Logische Ebene



Realisierungen für die logische Ebene

Dokumentzentrierte XML-Dokumente:

- Darstellung: **XML**, SGML
- Anfragen: **XQuery**, XPath, DOM, IR-Anfragen

Semistrukturierte XML-Dokumente:

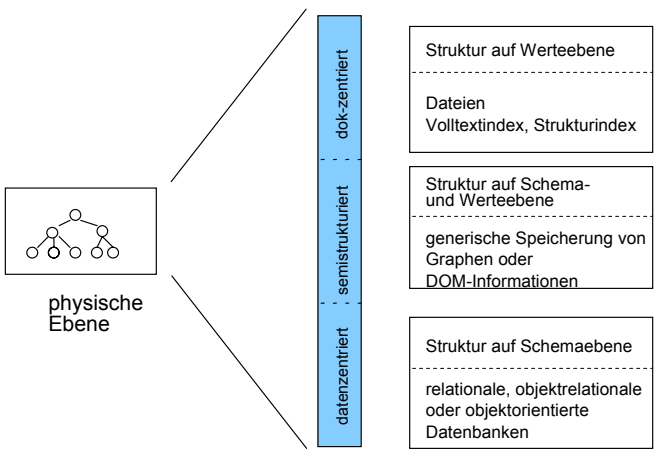
- Darstellung: **XML**, OEM
- Anfragen: **XQuery**, Lorel

Datenzentrierte XML-Dokumente:

- Darstellung: **XML**, rel. DM, oo. DM
- Anfragen: **XQuery**, SQL, OQL

große Band-breite

Architektur: physische Ebene

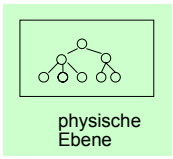


Realisierungen für die physische Ebene

Speicherung der XML-Dokumente als Ganzes und Indizierung (textbasiert native Verfahren)
- Volltextindex
- Volltext- und Strukturindex

Speicherung der Graphenstruktur (modellbasierte native Verfahren)
- generische Graphspeicherung
- Speicherung der DOM-Informationen

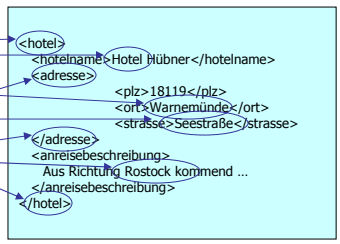
strukturierte Abbildung auf Datenbanken
- relationale Datenbanken
- objekt-orientierte und objekt-relationale Datenbanken
- Einsatz von benutzerdefinierten Mappingverfahren



Speicherung von XML Dokumenten als Ganzes

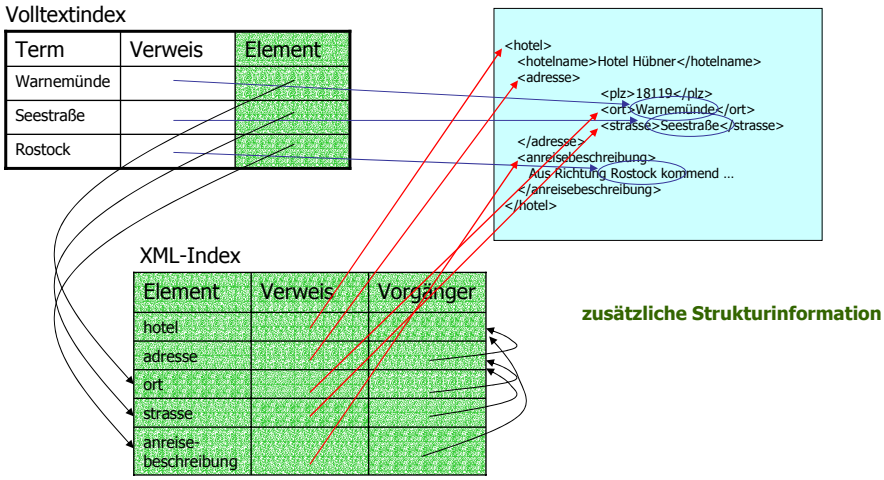
Volltextindex

Term	Verweis
Hotel	
Warnemünde	
Seestraße	
Rostock	
adresse	



- XML Dokument wird unverändert gespeichert
- zusätzlich Volltextindex (z.B. invertierte Wortliste)
  - Volltextretrieval
  - keine Unterscheidung von Markup (Elementnamen) und Textinhalt
  - keine Strukturinformation

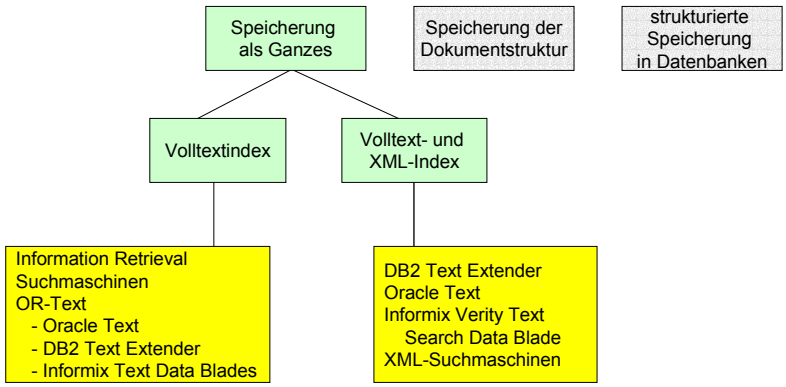
Speicherung von XML Dokumenten als Ganzes



Eigenschaften

Schemabeschreibung	nicht erforderlich
Dokumentrekonstruktion	Dokumente bleiben im Original erhalten
Anfragen	Anfragen des Information Retrieval Auswertung des Markup in den Anfragen XML-Anfragen möglich
Weitere Besonderheiten	Volltextfunktionen (SQL-MM)
Einsatz	für dokumentzentrierte und semistrukturierte Anwendungen

Speicherung als Ganzes



Dekomposition und Generische Speicherung

- Speicherung der Graphstruktur
  - Knoten: Elemente und Attribute
  - Kanten: Beziehung zu Subelement/Attribut
  - oft RDBMS als Basis, generisches Schema
    - Tabelle **ELEMENTE** mit Spalten DOCID, ELEMENTNAME, ID, VORGÄNGER, POSITION, WERT
    - Tabelle **ATTRIBUTE** mit Spalten DOCID, ATTRIBUTNAME, ELEMENTID, WERT

Elemente

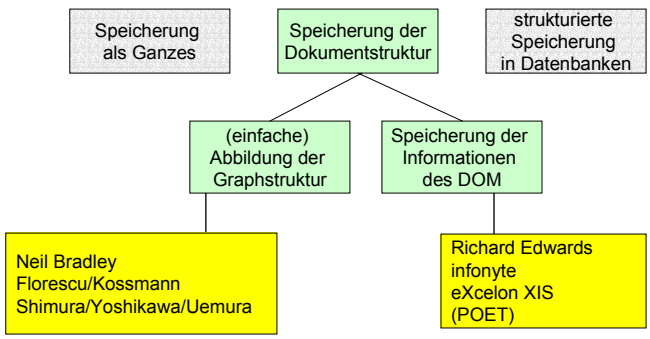
docID	Elementname	ID	Vorgänger	Position	Wert
H0001	hotel	001		1	
H0001	hotelname	002	001	1	Hotel Hübner
H0001	adresse	003	001	1	

- Speicherung der Informationen des Document Object Model
  - generisches Speicherungsschema orientiert sich an den DOM Node Types
    - Document, Element, Attribute, ...
  - auch hier RDBMS als Basis möglich

Eigenschaften

Schemabeschreibung	zur Speicherung nicht erforderlich
Dokumentrekonstruktion	möglich, aber sehr aufwändig
Anfragen	XML-Anfragen möglich angepasste DB-Anfrage
Weitere Besonderheiten	Anfragen über vielen Elementen/Attributen sind aufwändig DOM: standardisierte und allgemein akzeptierte Schnittstelle
Einsatz	daten-, dokumentzentrierte und semistrukturierte XML-Dokumente

Generische Speicherung



Systeme

Infonyte-DB

- verwendet zur Speicherung von XML-Dokumenten ein persistentes Document Object Model (PDOM)
- baut aber nicht auf existierende Datenbanken auf, sondern entwickelt Komponenten zur physischen Speicherung, die an die XML-Dokumente optimal angepasst sind
- Anfragesprache: Richtung XQuery

Tamino

- modellbasierte Speicherung von XML-Dokumenten
- XML-Anfragen auf den gespeicherten Dokumenten sind durch die Verwendung von XPath realisierbar
- Entwicklung geht Richtung erweitertes XPath, das den Namen Tamino XQuery trägt

eXcelon

- verwendet zur Speicherung von XML-Dokumenten das Document Object Model
- Alle Informationen, die in dieser API enthalten sind, werden in einer objektorientierten Datenbank objectstore adäquat gespeichert.
- Anfragen: OQL

Strukturierte Speicherung in Datenbanken

- XML Dokumentstruktur auf Schemaebene repräsentiert
  - Voraussetzung: explizites Schema der XML-Dokumente
- Abbildung auf ein (anwendungsspezifisches) DB-Schema
  - vom System per default festgelegt
  - benutzerdefinierbar
- Abbildungsvorschriften
  - Transformation von Anfrageresultaten
  - DB-Schemagenerierung
- Beispiel (objekt-relationales DBMS)

Hotel

id	hotelname	adresse			...
		plz	ort	strasse	
0001	Hotel Hübner	18119	Warnemünde	Seestraße 12	

Vor- und Nachteile

Vorteile: bei der Speicherung strukturierter Daten

- Anfragen, Datentypen, Aggregatfunktionen, Sichten
- Integration in andere Datenbanken

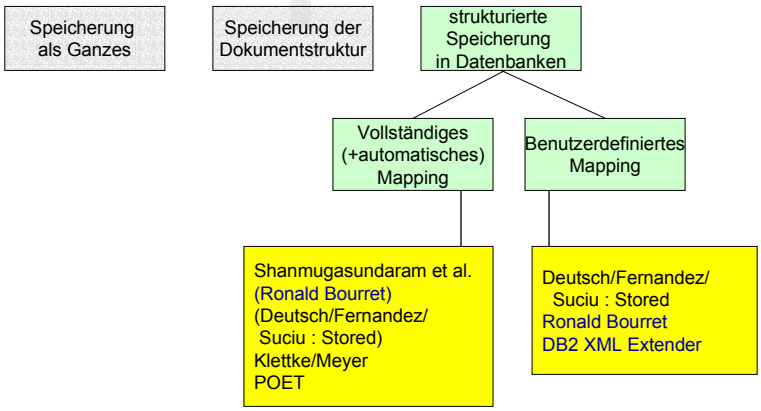
Nachteile: bei der Speicherung semi- und unstrukturierter Daten

- großes Schema, schwach gefüllte Datenbanken, viele Nullwerte
- Keine flexiblen Datentypen, Speicherung von Alternativen problematisch
- Fehlende Information Retrieval Anfragen, keine Volltextoperationen möglich

Eigenschaften

Schemabeschreibung	Zur Speicherung erforderlich
Dokumentrekonstruktion	Meist nicht möglich (Voraussetzung: Protokollierung des Abbildungsprozesses, vollständige Abb.)
Anfragen	- Datenbankanfragen - XML-Anfragen möglich
Weitere Besonderheiten	- Integration in bestehende Datenbanken möglich - XML-Dokumente und DB voneinander unabhängig
Einsatz	für datenzentrierte XML-Anwendungen

Strukturierte Speicherung in Datenbanken



Systeme: Systemdefinierte Abbildung

POET

- zu jedem Element der DTD oder der XML-Schema-Beschreibung wird eine korrespondierende Java-Klasse erzeugt
- Diese wird innerhalb des Systems gespeichert, indem dazu eine Relation generiert wird
- Ist kein Schema für eine Dokumentkollektion vorhanden, so wird hier eine Datenbank mit festem Schema eingesetzt. Diese Schema entspricht den Informationen des Document Object Model. (siehe vorn)

Oracle 8i / Oracle 9i

- Zum Datenbanksystem Oracle werden seit der Version 8i Tools angeboten, die die XML-Speicherung unterstützen.
- Bestandteil des Oracle XML Developer's Kit (XDK)
- Verfügbar sind zum Beispiel XML-Parser und XSL-Transformator

Systeme: Benutzerdefinierte Abbildung

IBM DB2 XML-Extender

- Speicherung von XML-Dokumenten im Datenbanksystem DB2
- über eine Mappingvorschrift (DAD - Data Access Definition) wird angegeben, wie die Zuordnung von Elementen und Attributen eines Dokuments auf die Attribute der Datenbank erfolgen soll.
- Die Syntax der DAD-Dateien ist XML

Oracle

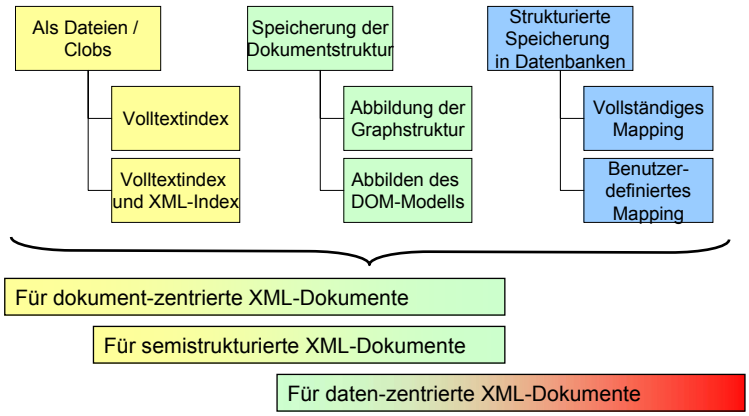
- objektrelationale Speicherung des XMLType
- Art der Speicherung wird durch ein annotiertes XML-Schema beschrieben, dieses enthält eine Zuordnung von Datenbankinformationen zu den XML-Bestandteilen
- Dadurch benutzerdefinierte Speicherung
- Zugriff auf die so gespeicherten XML-Dokumente mit XPath oder SQL

Systeme: Benutzerdefinierte Abbildung

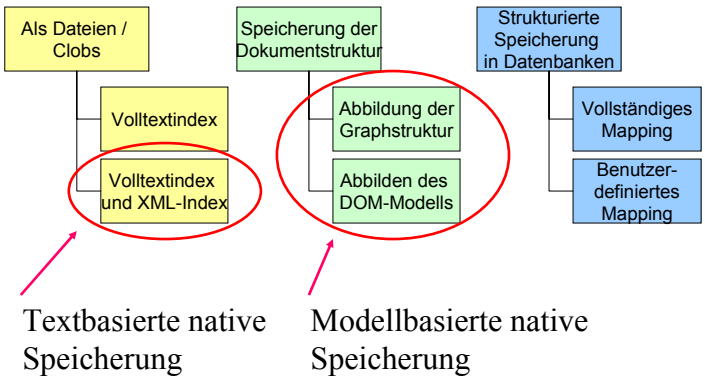
Microsoft SQL-Server

- speichert XML-Dokumente in Datenbanken
- anwenderdefiniertes Mapping wird verwendet, das durch ein annotiertes XDR-Schema festgelegt wird.
- Annotationen beschreiben die Zuordnung zwischen XML-Dokument und relationaler Datenbank, zum Beispiel: sql:relation und sql:field, bestimmen die Zuordnung zwischen Elementen und Attributen zu Datenbankrelationen und Datenbankattributen.

Speicherung von XML-Dokumenten



Was ist native Speicherung?



Systeme

- Hybride Speicherungsverfahren werden durch Datenbanksysteme unterstützt, die mehrere Speicherungsmöglichkeiten für XML-Dokumente anbieten.
- Oracle
  - Speicherung eines XMLType als CLOB oder objektrelational, Kombination der Varianten kann durch entsprechende Angaben im annotierten XML Schema erfolgen
- DB2
  - Speicherung von XML-Dokumenten als xmlcolumn und xmlcollection
    - Kombination derzeit nicht möglich!
  - Gemeinsame Nutzung von XML- und Text-Extender

XML and CM/DL

- XML can be used to represent documents and data
- but XML is a text-oriented language
  - not really suitable for multimedia content
- Multi-media content can be referenced in XML documents
  - URI, XLink, XPointer, XPath
- Multi-media content can be encoded in a text-based format
  - Scalable Vector Graphics (SVG)
  - Synchronized Multimedia Integration Language (SMIL)
- XML for meta-data representation
  - Meta-data standards (e.g., Dublin Core)
  - Representation in XML
    - RDF

Synchronized Multimedia Integration Language

- Creations of multi-media presentations, declarative description of
  - presentation layout
  - objects involved
  - timing aspects
- SMIL is **not** a container format
  - mulit-media objects (pictures, audio, video, …) are not included, only referenced via a URI
    - same object can be included in multiple presentations
    - integration of "remote" objects (e.g., weather chart)
- Object alternatives in SMIL
  - same object in different resolutions, dynamically selected based on available bandwidth
  - text or audio stored in different languages, selected based on user language preferences
- Interactive capabilities (limited)
  - follow presentation links

SMIL Documents

- XML documents following the SMIL DTD or Schema
  - document header
    - general presentation properties, such as layout, position of regions, …
    - based on a layout language close to cascading style sheets (CSS)
  - document body
    - actual definition of the presentation (incl. timing aspects)
    - number of XML element tags for different types of MM-objects
      - <audio>, <img>, <video>, <textstream>
      - Attributes for representing object URI, presentation region, MIME-type, presentation duration
    - objects can be "hyperlinked" to other resources on the web
    - timing aspects
      - parallel presentation (i.e., at the same time)
      - sequential presentation
      - duration attribute

Example:

```
<seq>
   <img id="a" dur="6s" begin="0s" src="..." />
   <img id="b" dur="4s" begin="0s" src="..." />
   <img id="c" dur="5s" begin="2s" src="..." />
</seq>
```

Resource Description Framework (RDF)

- Language for representing information (e.g., meta data) about resources on the web
  - identify something on the web using Uniform Resource Identifier (URI)
  - describe it using simple property/value pairs
- RDF statement can be represented using a graph
  - Example (from the RDF spec): "there is a Person identified by http://www.w3.org/People/EM/contact#me, whose name is Eric Miller, whose email address is em@w3.org, and whose title is Dr."
- RDF heritage: knowledge representation
  - semantic networks

RDF/XML

- XML-based syntax for encoding and exchanging RDF statements
  - Example

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf=…>
<contact:Person
  rdf:about=
  "http://www.w3.org/People/EM/contact#me">
   <contact:fullName>
              Eric Miller
   </contact:fullName>
   <contact:mailbox
    rdf:resource="mailto:em@w3.org"/>
  <contact:personalTitle>
              Dr.
  </contact:personalTitle>
</contact:Person>
</rdf:RDF>
```

Dublin Core

- Meta-data standard for describing networked resources
  - established by international, cross-disciplinary group of professionals from librarianship, computer science, text encoding, the museum community, and other related fields
  - major goal: help improve resource discovery
  - also used in closed environments, for other purposes(e.g., meta-data exchange)
- Meta-data description
  - uses a set of common meta data elements (nouns) and qualifiers (adjectives)
    - can be embedded in the resource (e.g., as HTML meta tags)
    - can be contained in a separate record/description of a resource (e.g., in a meta-data catalog file or database)
- Dublin Core defines
  - a vocabulary for meta data
    - simple to use, based on commonly used semantics, international, extensible,
  - best practices of how to use the language in various formats
    - HTML, XML, RDF

Dublin Core Elements

- Elements can be broadly grouped into three categories
  - Content
    - **Title**: A name given to the resource.
    - **Subject**: The topic of the content of the resource.
    - **Description**: An account of the content of the resource.
    - **Type**: The nature or genre of the content of the resource.
    - **Source**: A reference to a resource from which the present resource is derived.
    - **Relation**: A reference to a related resource.
    - **Coverage**: The extent or scope of the content of the resource.
  - Intellectual Property
    - **Creator**: An entity primarily responsible for making the content of the resource.
    - **Contributor**: An entity responsible for making contributions to the resource content.
    - **Publisher**: An entity responsible for making the resource available
    - **Rights**: Information about rights held in and over the resource.
  - Instantiation
    - **Date**: A date associated with an event in the life cycle of the resource.
    - **Format**: The physical or digital manifestation of the resource.
    - **Identifier**: An unambiguous reference to the resource within a given context.
    - **Language**: A language of the intellectual content of the resource.

DC Elements (cont.)

- Each element is optional and repeatable
- There is no defined order of elements
- Definition of controlled vocabularies possible (i.e., permitted values for elements)
  - uses concept of qualifiers

Qualifiers

- Two broad classes
  - Element Refinement: make the meaning of an element narrower or more specific
  - Encoding Scheme: identify schemes that aid in the interpretation of an element value
- Example: Element **Date**
  - Refinements
    - Created, Valid, Available, Issued, Modified, Date Copyrighted, Date Submitted
  - Encoding Schemes
    - DCMI Period, W3C-DTF
- Example: Element **Relation**
  - Refinements
    - Is Version Of, Has Version, Is Replaced By, Replaces, Is Required By, Requires, Is Part Of, Has Part, Is Referenced By, References, Is Format Of, Has Format, Conforms To
  - Encoding Scheme
    - URI

DC XML Implementation Guidelines

- Each DC element is represented as a separate XML element
  - refinements become elements of their own
  - encoding schemes are represented using xsi:type
- Example

```
<metadata xmlns=…>
  <dc:title> UKOLN </dc:title>
  <dc:subject> national centre, network information support</ dc:subject>
  <dc:identifier xsi:type="dcterms:URI"> http://www.ukoln.ac.uk/ </dc:identifier>
  <dcterms:modified xsi:type="dcterms:W3CDTF">
          2001-07-18
  </dcterms:modified>
  …
</metadata>
```


DC RDF Implementation Guidelines

- Each resource is described in an RDF description element
  - most appropriate URI to be used for rdf:about attribute
- Example

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:dc ="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://dublincore.org/">
    <dc:title>Dublin Core Metadata Initiative - Home Page</dc:title>
    <dc:description>
          The Dublin Core Metadata Initiative Web site.
    </dc:description>
    <dc:date>2001-01-16</dc:date>
    <dc:format>text/html</dc:format>
    <dc:language>en</dc:language>
    <dc:contributor>The Dublin Core Metadata Initiative</dc:contributor>
  </rdf:Description>
</rdf:RDF>
```

Zusätzliche Literatur

- Rahm, E., Vossen, G. (Herausgeber):
  Web & Datenbanken – Konzepte, Architekturen, Anwendungen
  dpunkt Verlag, 2003