XML und Datenbanken

Tobias Lohmann
Michael Mussil
Mark Wiesemann

4. Februar 2003

Gliederung

Einführung

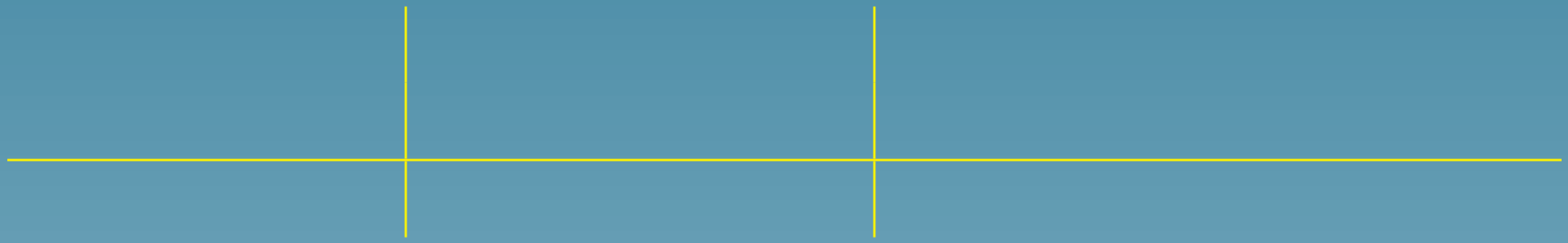
- großer Umfang an XML-Dokumenten verlangt strukturierte Organisation
- Funktionalitäten wie Versionskontrolle, Mehrbenutzerzugriff werden benötigt
- daher: Verwendung von DB-Technologie zur Ablage von XML-Dokumenten
- 3 Typen von „XML-Datenbanken“:
  - ⋆ native XML-Datenbanken (NXD)
  - ⋆ XML-erweiterte (relationale) Datenbanken (XED)
  - ⋆ hybride XML-Datenbanken (HXD)

Gliederung

Strukturiertheit

regelmäßige Struktur:

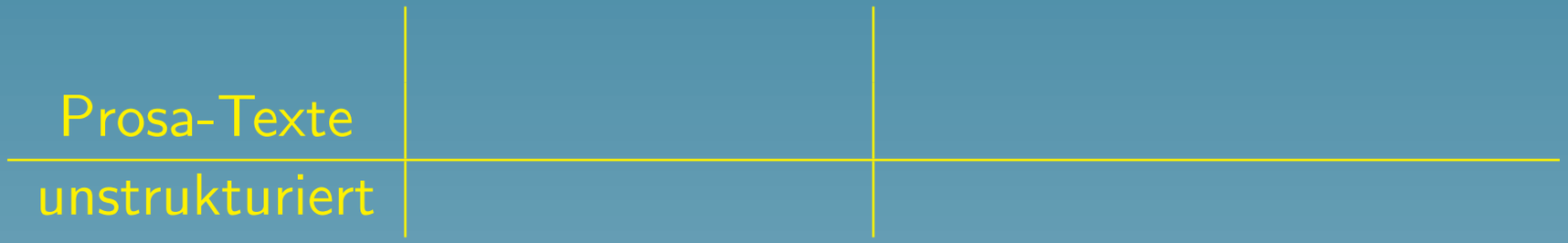
- leichter automatisiert zu verarbeiten
- weniger Flexibilität



Strukturiertheit

regelmäßige Struktur:

- leichter automatisiert zu verarbeiten
- weniger Flexibilität



Strukturiertheit

regelmäßige Struktur:

- leichter automatisiert zu verarbeiten
- weniger Flexibilität

Prosa-Texte		Tabellenschemata in relationalen Datenbanken
unstrukturiert		stark strukturiert

Strukturiertheit

regelmäßige Struktur:

- leichter automatisiert zu verarbeiten
- weniger Flexibilität

Prosa-Texte	Kompromiss	Tabellenschemata in relationalen Datenbanken
unstrukturiert	semi-strukturiert	stark strukturiert

Semi-strukturierte Daten

Merkmale:

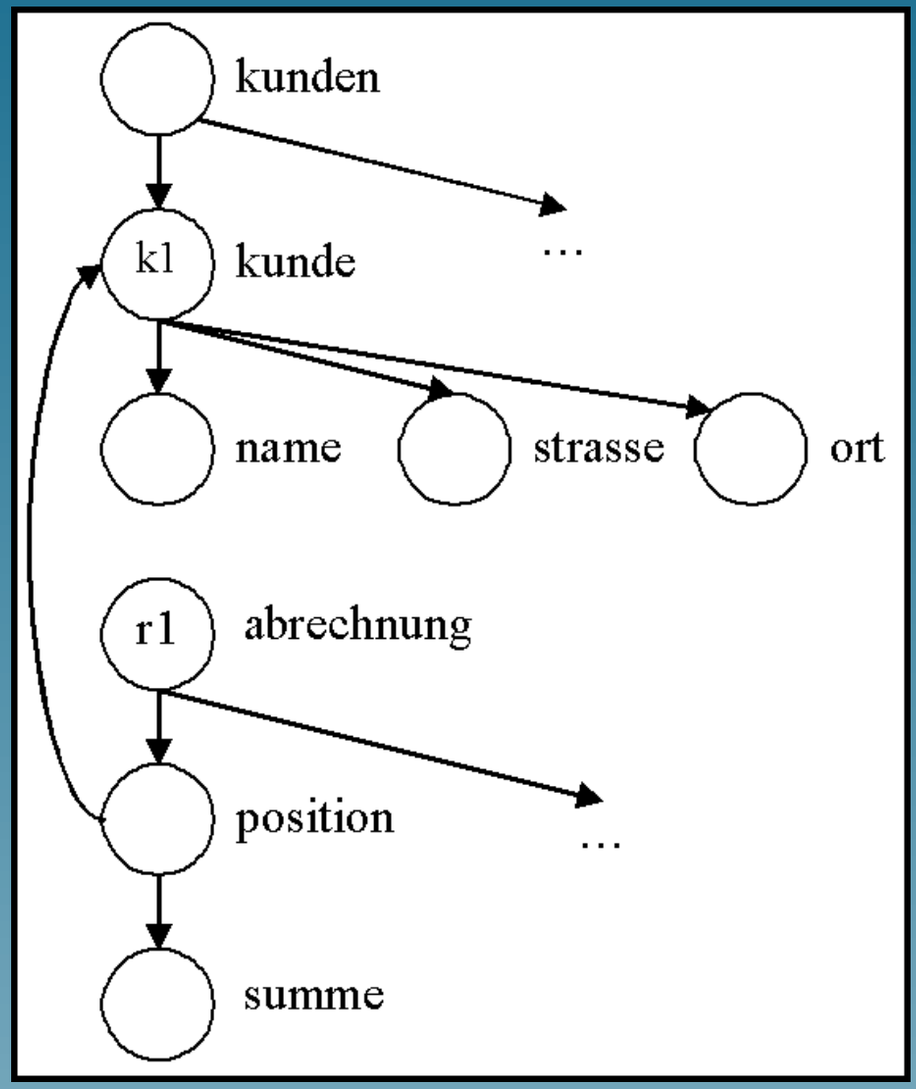
- Kompromiss zwischen starker Strukturierung und Unstrukturiertheit
- unregelmäßige Struktur
- implizite Struktur
- partielle Struktur

OEM-Datenmodell

- OEM = Object Exchange Model
- Objekte können eindeutigen Identifikator (oid) besitzen: besteht aus &-Zeichen und String, kann für Referenzen genutzt werden
- Aufbau: `<[oid ]bezeichner typ wert>`
- atomare Objekte vom Typ `string`, `integer`, `real` etc.
- komplexe Objekte vom Typ `set`, die Unterobjekte oder Verweise enthalten
- Darstellung durch knotenbewerteten Graph (Objekte als Knoten)

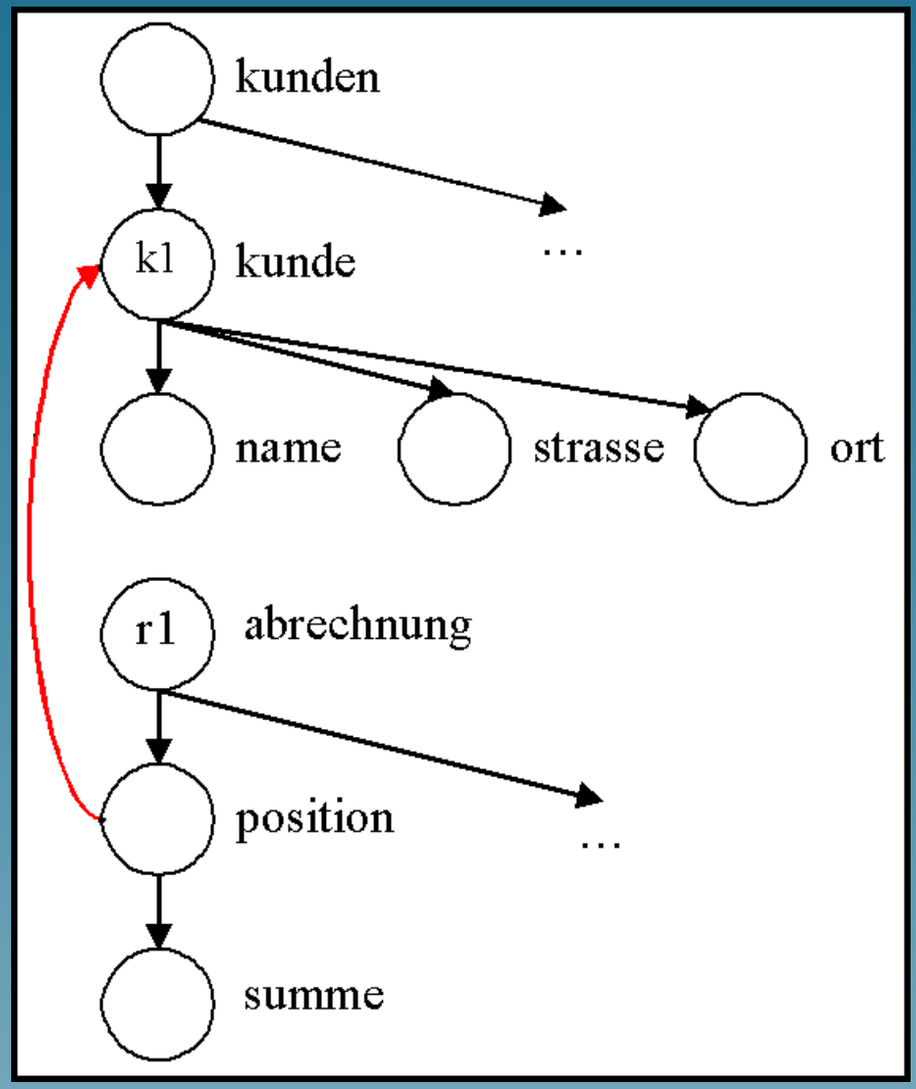
OEM-Datenmodell: Beispiel

```
<kunden set {
    <&k1 kunde set {
        <name string "Herr Müller">
        <strasse string "Bergweg 5">
        <ort string "Köln">
    }>
}>

<&r1 abrechnung set {
    <position set {
        &k1
        <summe real 113.11>
    }>
}>
```



OEM-Datenmodell: Beispiel

```
<kunden set {
    <&k1 kunde set {
        <name string "Herr Müller">
        <strasse string "Bergweg 5">
        <ort string "Köln">
    }>
}>

<&r1 abrechnung set {
    <position set {
        &k1
        <summe real 113.11>
    }>
}>
```



XML-Datenmodell

- ursprünglich nicht formell definiert
- W3C Recommendation (Oktober 2001):
  XML Information Set, kurz: Infoset
  - ⋆ abhängig von der Validierungsart
  - ⋆ unabhängig von konkreter Syntax
  - ⋆ beschreibt, welche Informationen abgelegt und wie diese strukturiert werden können
- 11 verschiedene „information items“ mit unterschiedlichen Eigenschaften (z.B. document, element, comment, attribute, namespace)

Gliederung

Datenorientierte Dokumente

- Merkmale:
  - ⋆ stark strukturiert
  - ⋆ für maschinelle Verarbeitung gut geeignet
  - ⋆ Reihenfolge der Kindelemente meist unwichtig
- Beispiele:
  - ⋆ Börsenkurse
  - ⋆ wissenschaftliche Daten (Messreihen)

Datenorientierte Dokumente: Beispiel

```
<Fluege>
  <Airline>i5 Airways</Airline>
  <Start>Aachen</Start>
  <Ziel>New York</Ziel>
  <Flug>
    <Abflug>09:15</Abflug>
    <Ankunft>10:15</Ankunft>
  </Flug>
  <Flug>
    <Abflug>11:15</Abflug>
    <Ankunft>12:15</Ankunft>
  </Flug>
  <Flug>
    <Abflug>13:15</Abflug>
    <Ankunft>14:15</Ankunft>
  </Flug>
  ...
</Fluege>
```

Textorientierte Dokumente

- Merkmale:
  - ⋆ schwach strukturiert
  - ⋆ für Menschen leicht verständlich
  - ⋆ Reihenfolge der Kindelemente wichtig
- Beispiele:
  - ⋆ Bücher
  - ⋆ E-Mails
  - ⋆ handgeschriebenes XML

Textorientierte Dokumente: Beispiel

```
<Produkt>
  <Einleitung>
    <ProduktName>i5-Pizza</ProduktName> von der <Hersteller>RWTH
    Food GmbH</Hersteller> ist <Zusammenfassung>die neueste
    Pizza-Kreation für Informatiker</Zusammenfassung>.
  </Einleitung>
  <Beschreibung>
    <Absatz>Die neue Pizza sorgt zusammen mit der bereits seit letztem
    Jahr verfügbaren <b>i5-Cola</b> für eine <i>optimale Ernährung</i>
    aller Informatiker an der RWTH.</Absatz>
    <Absatz>Sie können:</Absatz>
    <Liste>
      <Eintrag><Link URL="i.html">mehr Infos erhalten</Link></Eintrag>
      <Eintrag><Link URL="b.html">bestellen</Link></Eintrag>
    </Liste>
    <Absatz>Die Pizza ist ab 2010 am Lehrstuhl erhältlich und
    kostet <b>nur 99 Cent</b>.</Absatz>
  </Beschreibung>
</Produkt>
```

Gegenüberstellung: daten- und textorientierte Dokumente

- Entscheidung, ob Dokument daten- oder textorientiert ist, nicht einfach, z.B.:
  - ⋆ Rechnungen können unstrukturierte Beschreibungen enthalten
  - ⋆ Handbücher können strukturierte Elemente enthalten
- Faustregel zur Speicherung:
  - ⋆ datenorientierte Dokumente: in relationalen Datenbanken
  - ⋆ textorientierte Dokumente: in nativen XML-Datenbanken

Gliederung

Ablage und Abfrage von datenorientierten Dokumenten

- Übersetzung des Schema-Formalismus (DTD) in Datenbank-Schema-Formalismus
- zwei Methoden:
  - ⋆ Tabellen-basierte Abbildungen
  - ⋆ Objekt-relationale Abbildungen

Tabellen-basierte Abbildung: Beispiel

```
<A>
  <B>
    <C>ccc</C>
    <D>ddd</D>
    <E>eee</E>
  </B>
  <B>
    <C>fff</C>
    <D>ggg</D>
    <E>hhh</E>
  </B>
</A>
```

<=>

Tabelle A

C	D	E
...	...	...
ccc	ddd	eee
fff	ggg	hhh
...	...	...

Tabellen-basierte Abbildung

- geeignet für Austausch von Daten zwischen zwei relationalen Datenbanken
- Nachteil 1: XML-Dokumente müssen bestimmtes Format haben
- Nachteil 2: Verlust von physikalischer Struktur und Kommentaren

Objekt-relationale Abbildung

- wird von XML-erweiterten relationalen Datenbanken benutzt
- Modellierung des XML-Dokuments als Objektbaum
- zwei Schritte:
  - ⋆ DTD => Objekt-Schema
  - ⋆ Objekt-Schema => Datenbank-Schema
- viele Programme kombinieren beide Schritte

Abbildung von DTD auf Objekt-Schema

- einfache Element-Typen werden auf skalare Daten-Typen abgebildet
- komplexe Element-Typen werden auf Klassen abgebildet
- Attribute werden auf Eigenschaften abgebildet

```
            DTD                                  Klassen
=========================                  ============
<!ELEMENT A (B, C)>                        class A {
<!ELEMENT B (#PCDATA)>                        String b;
<!ATTLIST A                          ==>      C      c;
       F CDATA #REQUIRED>                     String f;
                                           }
<!ELEMENT C (D, E)>                        class C {
<!ELEMENT D (#PCDATA)>               ==>      String d;
<!ELEMENT E (#PCDATA)>                        String e;
                                           }
```

Abbildung von Objekt- auf Datenbank-Schema

- Klassen werden auf Tabellen abgebildet
- skalare Eigenschaften werden auf Spalten abgebildet
- Verweis-Eigenschaften werden auf Primär- / Fremdschlüssel abgebildet

```
  Klassen                      Tabellen
============                 ==============
class A {                    Tabelle A:
   String b;                    Spalte b
   C      c;      ==>           Spalte c_fs
   String f;                    Spalte f
}
class C {                    Tabelle C:
   String d;      ==>           Spalte d
   String e;                    Spalte e
}                               Spalte c_ps
```

Anfragesprachen

(1) Template-basierte Anfragesprachen:

- Einbettung von SQL-Anfragen in ein XML-Dokument

(2) SQL-basierte Anfragesprachen:

- spezielle SELECT-Anweisungen
- Ergebnisse werden in XML zurückgegeben

(3) XML-Anfragesprachen:

- Datenbank stellt virtuelles XML-Dokument zur Verfügung
- Beispiel: XML Query bzw. XQuery

Anfragesprachen: Beispiel

```
<FlugInfo>
  <Einleitung>Verfügbare Flüge:</Einleitung>
  <SelectStmt>SELECT Airline, FlugNr, Abflug FROM Fluege</SelectStmt>
    <Flug>
      <Airline>$Airline</Airline>
      <FlugNr>$FlugNr</FlugNr>
      <Abflug>$Abflug</Abflug>
    </Flug>
</FlugInfo>
```

```
<FlugInfo>
  <Einleitung>Verfügbare Flüge:</Einleitung>
  <Fluege>
    <Flug>
      <Airline>i5 Airways</Airline>
      <FlugNr>123</FlugNr>
      <Abflug>2003-02-04 16:00</Abflug>
    </Flug>
    ...
  </Fluege>
</FlugInfo>
```

Speicherung von datenorientierten Dokumenten in nativen XML-Datenbanken

- semi-strukturierte Daten führen zu großem Datenbank-Schema mit vielen Tabellen und vielen NULL-Spalten
- schneller Zugriff auf die Daten, wenn Abfrage in ähnlicher Struktur wie Speicherung
- Nachteil bei strukturfremden Abfragen
- XML-Anfragesprachen nutzbar

Gliederung

Ablage und Abfrage von textorientierten Dokumenten (1/2)

- ... im Dateisystem:
  - ⋆ geeignet für kleine Dokumentkollektionen
  - ⋆ Volltextsuche z.B. mit `grep` (keine Unterscheidung zwischen Text und Markup)
  - ⋆ Versionskontrolle mit CVS möglich
- ... in nativen XML-Datenbanken
  - ⋆ Dokument-Reihenfolge, Kommentare, CDATA-Bereiche und Entities bleiben erhalten
  - ⋆ Funktionalitäten wie Versionskontrolle, Mehrbenutzerzugriff sind vorhanden

Ablage und Abfrage von textorientierten Dokumenten (2/2)

- ... in relationalen Datenbanken
  - ⋆ Speicherung in einem Feld vom Typ **B**inary / **C**haracter **L**arge **OB**ject (BLOB / CLOB)
  - ⋆ Funktionalitäten: Sicherheit, Transaktionskontrolle, Mehrbenutzerzugriff
  - ⋆ effizienter durch Nutzung von Look-Aside-Tabellen
  - ⋆ Look-Aside-Tabelle enthält indizierten Wert und Verweis auf Primärschlüssel-Wert der Dokument-Tabelle
  - ⋆ schnelle Zugriffe, aber Tabellen müssen konsistent gehalten werden

Beispiel: Benutzung einer Look-Aside-Tabelle

PS	Dokument
1	„Flugplan-Beispiel“
2	„i5-Pizza-Beispiel“

FS	Schlüsselwort
1	Aachen
1	i5
2	i5
1	New York
2	Pizza
2	Informatiker
	...

Gliederung

Native XML-Datenbanken

- definieren ein logisches Modell eines XML-Dokuments
- speichern Dokumente und rufen sie gemäß dem Modell wieder ab
- Modell muss mindestens aus Elementen, Attributen, PCDATA und einer festgelegten Reihenfolge bestehen
- kleinste Dateneinheit: XML-Dokument (bei rel. DB: Tabellenzeile)
- an keiner besonderen physikalischen Speicherung festgemacht
- Begriff von der Software AG in der Tamino-Werbekampagne geprägt

Architekturen von nativen XML-Datenbanken

- Text-basierte native XML-Datenbanken:
  - ⋆ Speicherung der Daten als Text
  - ⋆ schneller Zugriff über Indizes auf ganze Dokumente bzw. Dokumentfragmente
  - ⋆ schlechte Performanz bei strukturfremden Abfragen
- Modell-basierte native XML-Datenbanken:
  - ⋆ Speicherung in relationalen Datenbanken durch Analyse des Dokuments und Aufbau eines Objektmodells
  - ⋆ Zusammensetzung der fragmentierten Informationen bei Anfragebeantwortung erforderlich
  - ⋆ Performanz für alle Anfragetypen ungefähr gleich

Gliederung

Zusammenfassung

- Semi-strukturierte Daten
- Datenorientierte vs. textorientierte Dokumente
- Ablage und Abfrage von datenorientierten Dokumenten
  - ⋆ Abbildungen
  - ⋆ Anfragesprachen
- Ablage und Abfrage von textorientierten Dokumenten
  - ⋆ im Dateisystem
  - ⋆ in nativen XML-Datenbanken
  - ⋆ in relationalen Datenbanken
- Native XML-Datenbanken

Fragen?