Datenverarbeitung mit XML

Modelle und Sprachen

Jörn Clausen

joern@TechFak.Uni-Bielefeld.DE

Übersicht

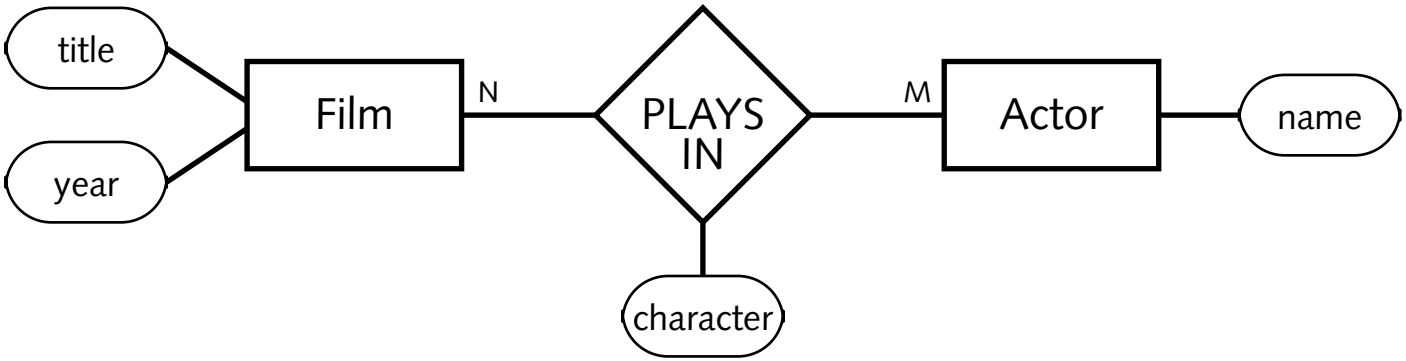
- Daten in Relationen
- Daten in spitzen Klammern
- XSLT vs. XQuery
- Beispiele

Relationales Datenmodell

- 1970: E.F. Codd
- vereinfacht: Tabellen und Operationen darauf
- mathematische Fundierung: Relationenalgebra, Relationenkalkül
- zahlreiche freie und kommerzielle Produkte

„At the time, Nixon was normalizing relations with China. I figured that if he could normalize relations, then so could I.“

Filmdatenbank



Film

fid	title	year
1	Lord of the Rings	2001
2	Star Wars II	2002
3	The Matrix	1999

Role

fid	aid	character
1	1	Elrond
1	2	Gandalf
3	1	Agent Smith
2	3	Count Dooku

Actor

aid	name
1	Hugo Weaving
2	Ian McKellen
3	Christopher Lee
4	Keanu Reeves

Fragen

- Datenbankanfragen:
  - alle Filme aus dem Jahr 2001
  - alle Schauspieler, die in „The Matrix“ mitgespielt haben
  - alle Filme, in denen Hugo Weaving mitgespielt hat
  - alle Schauspieler, die mit Hugo Weaving gearbeitet haben
- in diesem Modell nur kostspielig zu beantworten:
  - alle Filme, in denen die Rolle „Gandalf“ vorkommt
  - alle Schauspieler, die „Inspector Clouseau“ dargestellt haben

Sprache

„A first-order predicate calculus suffices if the collection of relations is in normal form. Such a language would provide a yardstick of linguistic power for all other proposed data languages [. . .].“

„The universality of the data sublanguage lies in its descriptive ability [. . .].“

E.F. Codd, 1970

- 1986: ISO 9075 – Structured Query Language (SQL)
- auch 1986: ISO 8879 – Standard Generalized Markup Language

SQL

```
SELECT title FROM Film
  WHERE year=2001

SELECT a.name, r.character FROM Actor a, Film f, Role r
  WHERE f.title='The Matrix' AND r.aid=a.aid AND r.fid=f.fid

SELECT DISTINCT a.name FROM Actor a, Film f, Role r
  WHERE f.fid IN (
    SELECT f.fid FROM Actor a, Film f, Role r
      WHERE a.name='Hugo Weaving'
        AND r.aid=a.aid AND r.fid=f.fid
    ) AND r.aid=a.aid AND r.fid=f.fid
```

noch'n Beispiel

Universitaet

uid	name
1	Bielefeld
2	Paderborn
3	Dortmund

Fakultaet

fid	name	uid
1	TechFak	1
2	LiLi	1
3	Physik	3

AG

name	fid
PI	1
NI	1
CoLi	2

```
<uni name="Bielefeld">
   <fak name="TechFak">
      <ag name="PI"/>
      <ag name="NI"/></fak>
   <fak name="LiLi">
      <ag name="CoLi"/></fak></uni>
```

Reinventing the Wheel

- in Datenbank-Lehrbüchern häufig überlesen:
  - (Netzwerk-Modell)
  - Hierarchisches Modell
- 1969: IMS/360 von IBM (Information Management System)
- 2003: IMS Version 9
- Behauptung: XML kann Basis eines HDBMS sein

Datenhaltung mit XML

- Vorteile:
  - variable Listen (one-to-many relations) + Reihenfolge
  - irreguläre Daten (semi-structured data)
  - Lokalität, referentielle Integrität
- XML-Datei(en):
  - Text-Editor und ein paar Zeilen Perl/Java/XSLT/...
  - Versionierung mit RCS/CVS/...
- aber:
  - Typisierung, Skalierbarkeit, locking, ...
  - many-to-many relations

semi-structured data

```
<manpage>
  <name>ls</name>
  <synopsis>ls [OPTION]... [FILE]...</synopsis>
  <description>
    <para>List information about the <opt>FILE</opt>s (the current
      directory by default). Sort entries alphabetically if none of
      <opt>-cftuSUX</opt> nor <opt>--sort</opt>.</para>
    <optionlist>
      <option short="a" long="all">do not hide ...</option>
      <option short="A" long="almost-all">do not list ...</option>
    </optionlist>
  </description>
  <seealso>
    <link ref="chmod" sec="1"/><link ref="environ" sec="5"/>
  </seealso>
</manpage>
```

FilmDB mit XML modellieren

- Möglichkeiten:
  - direkte Abbildung des relationalen Modells
  - streng hierarchisches Modell (mit Redundanzen)
  - gemischte Darstellung (Hierarchie + Verweise)

XML-Tabellen

films.xml:

```
<films>
  <film id="film1"> ...
  <film id="film2"> ...
  ...
</films>
```

actors.xml:

```
<actors>
  <actor id="actor1"> ...
  <actor id="actor2"> ...
  ...
</actors>
```

roles.xml:

```
<roles>
  <role fid="film1" aid="actor1"> ...
  <role fid="film1" aid="actor2"> ...
  ...
</roles>
```

Hierarchie + Verweise

```
<filmdb>
  <films>
    <film id="film3">
      <title>The Matrix</title>
      <year>2002</year>
      <role actorid="actor4">Neo/Thomas A. Anderson</role>
      <role actorid="actor1">Agent Smith</role>
      ...
    </film>
    ...
  </films>
  <actors>
    <actor id="actor1">Hugo Weaving</actor>
    <actor id="actor4">Keanu Reeves</actor>
    ...
  </actors>
</filmdb>
```

referentielle Integrität

- falls DTD vorliegt: Verweise durch ID/IDREF

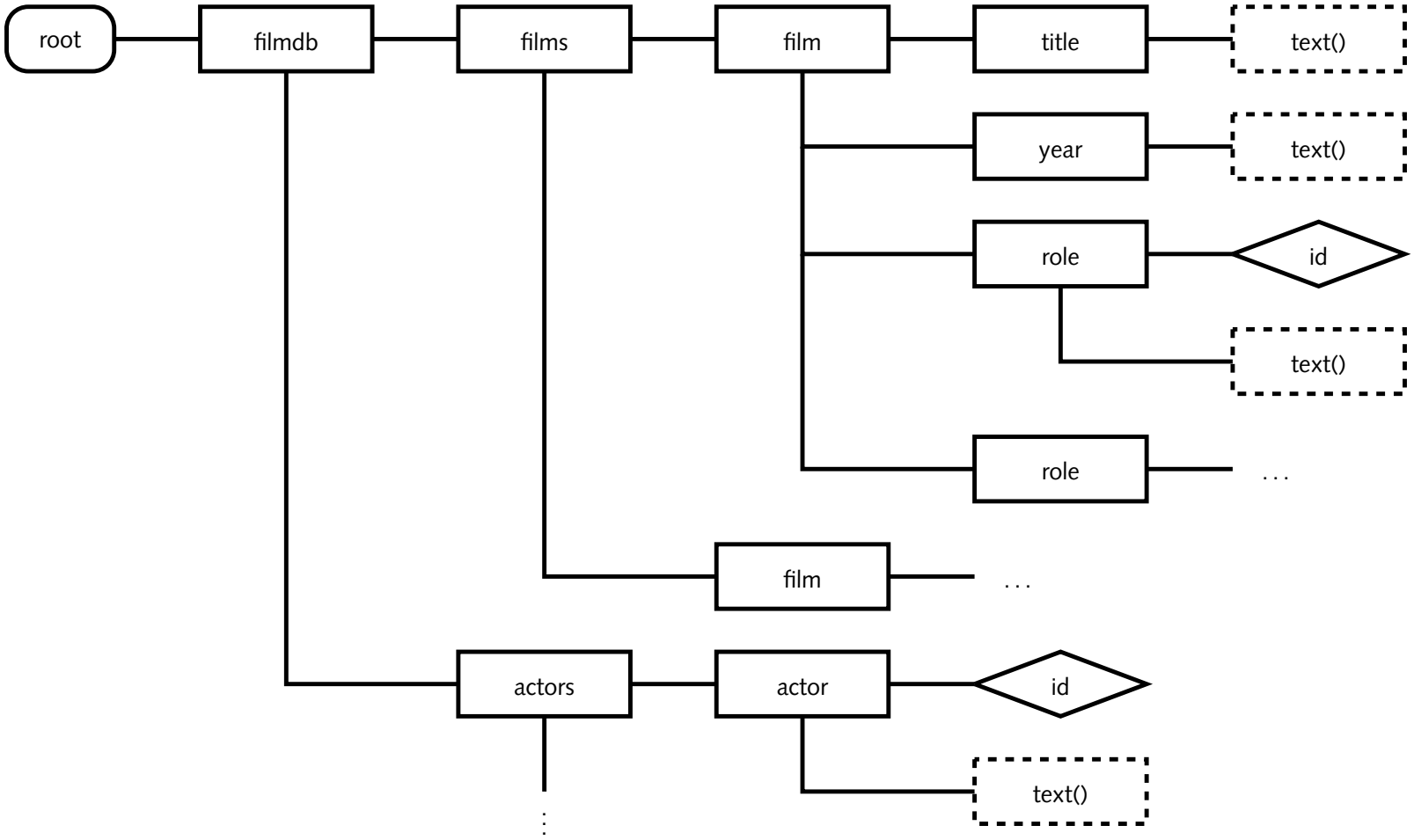
```
<!ELEMENT  actor              (#PCDATA)>
<!ATTLIST  actor
           id                 ID              #REQUIRED>

<!ELEMENT  role               (#PCDATA)>
<!ATTLIST  role
           actorid            IDREF           #REQUIRED>
```

- validierender XML-Parser findet
  - doppelt vergebene IDs
  - Verweise auf nicht-existierende IDs
- funktioniert nur innerhalb eines Dokuments

Sprachen für XML

- XPath:
  - Navigation innerhalb des „XML-Baums“
  - wie Pfade in einem Dateisystem
  - Grundlage von XSLT
- XQuery:
  - verwendet/erweitert XPath
  - FLWR expressions
- bisher nur Anfragen, keine Updates

Baumstruktur der FilmDB



XPath

- alle Filme:

```
/filmdb/films/film/title/text()
```

- alle Filme aus dem Jahr 2001:

```
/filmdb/films/film[year='2001']/title/text()
```

- alle Schauspieler, die in „The Matrix“ mitgespielt haben:

```
/filmdb/actors/actor[@id=/filmdb/films/
  film[title='The Matrix']/role/@actorid]/text()
```

- Abkürzung: `//` $\approx$ „rekursive Suche“
- alle Schauspieler, die mit Hugo Weaving gearbeitet haben:

```
//actor[@id=//film[role/@actorid=
  //actor[contains(.,'Hugo Weaving')]/@id]
  /role/@actorid]/text()
```

XSLT

```
<xsl:template match="film">
  <h1><xsl:value-of select="title"/>
      (<xsl:value-of select="year"/>)</h1>
  <ul>
    <xsl:apply-templates select="role"/>
  </ul>
</xsl:template>

<xsl:template match="role">
  <li>
    <xsl:value-of select="//actor[@id=current()/@actorid]"/>
    plays
    <xsl:value-of select="."/>
  </li>
</xsl:template>
```

Grenzen von XPath/XSLT

- Preisliste und Bestellung:

```
<warehouse>
  <item val="4.4">Food</item>
  <item val="6.4">Textiles</item>
  <item val="91.2">Luxuries</item>
  <item val="114.8">Narcotics</item>
</warehouse>
```

```
<order>
  <item quant="10">Food</item>
  <item quant="5">Narcotics</item>
  <item quant="5">Textiles</item>
</order>
```

- mit XSLT realisierbar: Rechnung mit Einzelpreisen und Summe
- nicht realisierbar: Rechnung nach Einzelpreisen sortieren
- Erweiterungen bzw. XSLT 2.0

XQuery

- funktionale Sprache
- Datentypen:
  - Literale, Atome
  - nodes
  - Sequenzen
- node sequences (XSLT 1.0: node sets)
- Konstruktoren für XML (Elemente und Attribute)
- Erzeugung von nicht-wohlgeformtem XML möglich

Ausdrücke

- Sequenz:

```
("Huey", "Dewie", "Louie")
(1 to 10)
```

- Ausdrücke und XML mischen:

```
<list>{(1 to 10)}</list>
```

- Variablen auswerten:

```
<name id="{$id}">{$name}</name>
```

- auf XML-Datei zugreifen:

```
<list>{doc('filmdb.xml')//film[year='2001']}</list>
```

- komplette `film`-Knoten werden kopiert

FLWR expressions

- Iteration über Sequenz:

```
for $n in (1 to 10)
return <li>{$n} zum Quadrat ist {$n*$n}</li>
```

- karthesisches Produkt:

```
for $n in (1 to 10),
    $m in (1 to 10)
return <li>{$n} mal {$m} ist {$n*$m}</li>
```

- Zuweisung:

```
for $n in (1 to 10)
let $m := (1 to $n)
return <seq max="{$n}">{$m}</seq>
```

FLWR expressions, cont.

- Bedingung:

```
for $n in (1 to 10)
let $m := (1 to $n)
where $n mod 2 = 0
return <seq max="{$n}">{$m}</seq>
```

- Iteration über XML-Knoten:

```
for $f in (doc('filmdb.xml')//film)
where $f/year='2001'
return $f
```

Mr. Anderson! Surprised to see me?

- SQL-Statement direkt umgesetzt

```
for $a in (doc('actors.xml')//actor),
    $f in (doc('films.xml')//film),
    $r in (doc('roles.xml')//role)
where $f/title='The Matrix'
      and $r/@actorid=$a/@id
      and $r/@filmid=$f/@id
return $a
```

- etwas kürzer (effizienter?)

```
let $f := doc('films.xml')//film[title='The Matrix']
let $r := doc('roles.xml')//role[@filmid=$f/@id]
for $a in (doc('actors.xml')//actor)
where $r/@actorid = $a/@id
return $a
```

Rechnung erstellen

- Idee: Erzeuge temporäres XML mit Einzelpreisen

```
for $i in (
  for $o in (doc('order.xml')//item)
  let $w := doc('wares.xml')//item[text() = $o/text()]
  return <item price="{$o/@quant*$w/@val}">{$o/text()}</item>
) order by ($i/@price) descending
return <tr><td>{$i/text()}</td><td>{$i/@price}</td></tr>
```

komplette Rechnung

```
let $l := (
  for $o in (doc('order.xml')//item)
  let $w := doc('wares.xml')//item[text() = $o/text()]
  return <item price="{$o/@quant * $w/@val}">{$o/text()}</item>
)
return <table>
  { for $i in $l
    order by ($i/@price) descending
    return <tr><td>{$i/text()}</td><td>{$i/@price}</td></tr> }
  <tr><td>Summe</td><td>{sum($l/@price)}</td></tr>
  </table>
```

Vergleich

XSLT	XQuery
transformiert Datei	Zugriff auf Datei(en)
templates	FLWR expressions
push/pull	pull
`node-set()`	Komposition
Parameter	Umgebungsinformation

- semi-Struktur, mixed content: push-Verarbeitung
- Struktur, container elements: pull-Verarbeitung
- Kombination aus XQuery und XSLT?