

Relational Storage of XML Data
XML: Anfrage- und Transformationssprachen
Seminar
Sommersemester 2001

Carsten Halle
halle@rz.uni-karlsruhe.de

Claudius Link
linkclau@uni-freiburg.de

7. August 2001

Zusammenfassung

In diesem Text soll ein Überblick über verschieden Techniken zur Konvertieren und Speicherung von XML Dokumenten in relationalen Datenbanken gegeben werden. Es werden Methoden zur Transformation von XML Daten in relationale Schematas und der Suche darauf, vorgestellt.

1 Kurzer Überblick

XML, die *eXtensible Markup Language* war vom W3C Konsortium zunächst als eine Methode zum medienunabhängigen Publizieren gedacht. Es sollte SGML vereinfachen um die Anfangshemmschwelle zu senken und damit den Einstieg zu erleichtern, ohne dabei aber die Flexibilität von SGML zu verlieren. Damit sollte den Ideen, Konzepte und Vorzügen von Markup-Sprachen endlich zum Durchbruch geholfen werden.

Seit der ersten Empfehlung des W3C zu XML 1.0 1996 wurden diese Ziele jedoch immer wieder erweitert. Die (semi-)strukturierte Art Daten in XML darzustellen erwies sich für viel Anwendungsgebiete als sehr nützlich. Da XML nicht nur Daten, sondern auch Metadaten enthalten kann, ermöglicht es die Zusammenführung der Daten und ihrer Formatbeschreibung. XML

wurde schon von Anfang an konzipiert um sowohl automatisiert verarbeitet zu werden, als auch immer noch für Menschen lesbar und verständlich zu sein. Durch die frühe Übernahme der Standardisierung durch das W3C, hat sich XML zu einer, von keiner Firma dominierten aber trotzdem standardisiert Technologie entwickelt, für die keinerlei Nutzungsgebühren zu entrichten sind.

Damit wurde XML die Basis für verschiedenste Protokolle zur Kommunikation oder Datenaustausch aber auch zur Datenbeschreibung. Zum einen wird XML in den ursprünglich vorgesehenen Gebieten als XHTML, MathML, SVG (Scalable Vector Graphics), SMIL (Synchronized Multimedia Integration Language) eingesetzt. Aber mit SOAP (Simple Object Access Protocol), XML-RPC (XML-Remote Procedure Call) hat XML auch in der *klassischen* Maschine-Maschine Kommunikation Fuß gefaßt.

Der XML Einsatz im Gesundheits- und Finanzwesen oder im *eGovernment* und vielen anderen Gebieten, ist sehr stark expandierend. In diesen Einsatzgebiet fallen große Mengen an Daten an, die effizient und sicher gespeichert werden müssen. Weiter Ansprüchen an das Speichersystem sind eine effiziente Speicherung, hohe Datensicherheit nicht nur auf referentielle Integrität bezogen, sondern auch im Hinblick auf Mehrbenutzerbetrieb auf dem Datenbestand.

Schon XML selbst bietet mit den zugehörigen Erweiterungen und Werkzeugen (wie XQuery, XML Schema, XLink, XBase und XPointer) eine DBMS ähnliche Schnittstelle, ohne sich auf eine Implementation festzulegen. Viele Anwendungen für die sich XML anbietet, wie zu Beispiel Kataloge, arbeiten außerdem schon Datenbank gestützt.

Deshalb bietet es sich an, XML mit eine Datenbank zu verknüpfen, beziehungsweise die XML Daten in einer Datenbank zu speichern.

2 Warum Relational Datenbanken

Speziell im Bereich des Datenaustausches und der Datenbeschreibung werde immer größer Mengen an Daten anfallen. Diese müssen effektiv und sicher gespeichert werden. Such- und Indizierungsmöglichkeiten innerhalb der Datenstrukturen sind erwünscht. Mehrere Benutzer sollten gleichzeitig an dem Dokument arbeiten können ohne sich zu beeinflussen. Diese Anforderungen werden von Datenbank Management Systeme (DBMS) erfüllt.

2.1 XML native Datenbanken

Zur Speicherung von Daten die in XML vorliegen, würde sich eine Datenbank die direkt mit XML umgehen kann anbieten. Da es sich bei XML um eine noch relative jungen Technologie handelt, sind auch die DBMS, die direkt mit XML Daten umgehen können noch sehr neu oder sogar erst in Entwicklung. Es fehlt ihnen damit an Ausgereiftheit, Stabilität und Skalierbarkeit, um sie für die Speicherung unternehmenskritischer Daten oder großer Datenmengen zu nutzen. Außerdem gibt es verschiedene Ansätze, wie XML-native DBMS implementierbar sind, von denen keiner standardisiert ist, Es ist noch gar nicht abzusehen welcher der Ansätze sich durchsetzen wird.

2.2 Objekt Orientierte Datenbanken

Auch Objekt Orientiert DBMS (OODBMS) würden sich zur Speicherung anbieten, da damit leicht die dem XML Dokument zu Grunde liegende Struktur abgebildet werden kann. Element-Unterelement- oder Eltern-Kind-Beziehungen können zu Beispiel direkt übernommen werden. Trotz ihrer längeren Entwicklungszeit haben sie allerdings nicht die nötige Stabilität, Ausgereiftheit und Leistungsfähigkeit erreicht um Relationale DBMS nur annähernd hinfällig zu machen und zu ersetzen.

2.3 Relationale Datenbanken

Relationale DBMS (RDBMS) sind aber nicht nur eine Notlösung. Auch wenn die relational Lösung nicht für alle XML Dokumente gleichermaßen gut geeignet ist, bieten sie durchaus auch Vorteile. Referenzen, Constraints und Trigger können zu Abbildung der logischen Struktur des XML Dokumentes auf die Datenbank verwendet werden¹. RDBMS sind durch 30 jährige Forschung zu einer sehr gut verstanden, hoch optimierten und stabilen Technologie geworden. Da sie sich zum Standard bei Datenbanken entwickelt, und in der Verbreitung alle anderen DBMS Modelle weit hinter sich gelassen haben, sind sie überall im Einsatz. Wenn also beim Speicher von XML Dokumente auf RDBMS zurückgegriffen werden kann, werden Investitionen weiter genutzt und gesichert, da die Anschaffung eines neuen DBMS unnötig wird. Bereits vorhandene (Legacy-)Daten und Werkzeuge können weiter genutzt und sogar integriert werden.

¹Was bei den hier vorgestellten Modellen aber nicht der Fall ist

3 Transformations Methoden

Generell lassen sich folgende Ansprüche an die Methoden zur Transformation stellen;

- Die semi-strukturierten XML Daten werden in ein strukturiertes relationales Datenbank Schema konvertiert.
- Die Konvertierung sollte möglichst verlustlos sein. Das heißt das XML Dokument sollte aus der Datenbank wiederherstellbar sein ohne das dessen Struktur zerstört wird.
- Es muß eine intuitive Anfragesprache existieren, die XML Anfragen mit SQL verbindet beziehungsweise darauf zurückführt ².
- Die Ergebnisse von Anfragen müssen wieder *vollständige* XML Dokumente sein.

Es müssen nicht alle dieser Anforderung komplett erfüllt werden, sondern es wird auf das jeweilige Anwendungsgebiet ankommt, welche Anforderungen Priorität haben.

Wie bereits erwähnt eignen sich nicht alle XML Dokumente gleichermaßen zur Speicherung in relationalen Datenbanken. Man unterscheidet deshalb zwischen daten- und dokumentenzentrischen XML Dokumenten.

Datenzentrisch sind XML Dokumente die eine reguläre, regelmäßige und feinkörnige Struktur aufweisen, wie Beispiel 1.

```
<bibliography>
  <book isbn="0672320541">
    <author>Benoit Marchal</author>
    <title>Applied XML Solutions</title>
  </book>
  <book isbn="0596000588">
    <author>Elliotte Rusty Harold</author>
    <author>W. Scott Means</author>
    <title>XML in a Nutshell</title>
  </book>
  <book isbn="0735710201">
    <author>Steven Holzner</author>
    <title>Inside XML</title>
  </book>
</bibliography>
```

²In Zukunft wird wohl XQL diese Lücke füllen

Beispiel 1: Ein Buch Katalog

Da die Relationen eine feste Struktur darstellen lassen sich XML Dokumente die sehr strukturiert sind relativ einfach zu Relationen umwandeln.

Dokumentenzentrisch heißt im Gegensatz dazu, das die Daten sehr viel irreguläre strukturiert sind und daß sie in größere Blöcken auftreten. Beispiele hierfür sind Handbücher (Beispiel 2 auf Seite 6) oder die meisten XHTML Dokumente.

```
<front>
  <docTitle>
    <titlePart type="main">xmltex: A non validating
      (and not 100% conforming) namespace aware XML
      parser implemented in &TeX;
    </titlePart>
  </docTitle>
  <docDate>Date: 2000-02-02</docDate>
  <docAuthor rend="email">davidc@nag.co.uk</docAuthor>
  <docAuthor>David Carlisle</docAuthor>
</front>
<body>
  <div id="intro">
    <head>Introduction</head>
    <p>
      xmltex implements a non validating parser for documents
      matching the W3C XML Namespaces Recommendation.

      The system may just be used to parse the file (expanding
      entity references and normalising namespace declarations)
      in which case it records a trace of the parse on the
      terminal. Normally however the information from the parse
      is used to trigger &TeX; typesetting code. Declarations
      (in &TeX; syntax) are provided as part of xmltex to
      associate &TeX; code with the start and end of each XML
      element, attributes, processing instructions, and with
      unicode character data.
    </p>
  </div>
```

Beispiel 2: Auszug aus der Dokumentation zu *xmltex*[5]

Zur Speicherung dieser Art von Dokumenten ist ein Content Management System wesentlich besser geeignet. Die Unterscheidung zwischen den verschiedenen Arten von Dokumenten wird aber nicht immer eindeutig möglich sein. Zum Beispiel kann technische Dokumentation sehr wohl eine feinkörnige reguläre Struktur aufweisen und es kann durchaus sinnvoll sein sie in einer Datenbank zu speichern.

Die Unterscheidung hängt also letztlich von dem geplanten Einsatzgebiet ab und muß vom Benutzer getroffen werden.

Für datenzentrische XML Dokumente gibt es jetzt verschiedene Methoden um ein relationales Schema zu erstellen und damit die Daten in einem RDBMS speichern zu können. Siehe Beispiel 1 auf Seite 5.

3.1 Manuelle Konvertierung

Es ist natürlich möglich zu jedem XML Dokument passende Relationen zu entwerfen und damit die Datenbank zu erzeugen. Allerdings erfordert dieses Vorgehen einen hohen Wissensstand sowohl im Anwendungsgebiet, als auch was XML und das RDBMS betrifft. Der große Aufwand der zum manuellen Konvertieren nötig ist, weil ja auch die Werkzeuge und Anfragen geschrieben werden müssen, ist zudem nur schlecht wiederverwendbar. Wenn sich die XML Datenstrukturen ändern ist unter Umständen der ganze Prozeß zu wiederholen und zusätzlich muß noch Aufwand betrieben werden um die Migration der alten Daten zu ermöglichen.

Einer Analyse des XML Dokumentes ist nicht zu vermeiden. Es bietet sich also an die Datenbank direkt aus den XML Daten zu erzeugen.

3.2 Anfragenanalyse

Eine Anfragen- und Aufwandsanalyse ist wie auch die manuelle Konvertierung aufwendig und benötigt auch einen sehr hohen Wissensstand. Um die Analyse zu automatisieren werden Beispiel Daten und Anfragen benötigt. Diese müssen also vorhanden sein oder zumindest für die Analyse erzeugt werden.

Auch hier ist die Analyse des XML Dokumentes ist nicht zu vermeiden. Es bietet sich also wieder an die Datenbank direkt aus den XML Daten zu erzeugen.

3.3 Analyse der DTD

Da die DTD die erlaubte Struktur der XML Daten beschreibt, scheint es auch angebracht diese zur Erzeugung der Relationen zu benutzen. Allerdings ist die DTD wesentlich allgemeingültiger und flexibler als die Daten. Zum Beispiel sind in der DTD Wertebereiche von Elementen und Attributen nicht nicht begrenzt, in den Daten werden aber gewisse Grenzen nicht überschritten. Diese Komplexität erfordert einen hohen Zusatzaufwand bei der Verwaltung der Daten wenn diese Flexibilität der DTD zu berücksichtigt werden soll. Die Folgen sind dann eine größere Datenbank und länger Laufzeiten, beides ist nicht erwünscht. Um diesen Aufwand einzuschränken müssen aber Daten analysiert werden. Es bietet sich wieder an, für die Transformation nur die XML Daten zu nutzen.

3.4 Analyse des XML Dokumentes

Da es sich herausgestellt hat, daß eine Analyse des XML Daten kaum zu vermeiden ist, werden im folgenden verschiedene Methoden vorgestellt, wie die Umsetzung bewerkstelligt werden kann.

Die vorgestellten Modelle sind:

1. Kanten Ansatz
2. Binärer Ansatz
3. Universelle Tabelle
4. Normalisierte Universelle Tabelle
5. Monet Modell

Die Methoden haben alle gemeinsam, daß das XML Dokument als Graph interpretiert wird, und dieser auf Relationen Abgebildet wird.

Als Beispielen zugrundeliegendes XML Dokument wird im Folgendem immer Beispiel 1 auf Seite 5 verwendet.

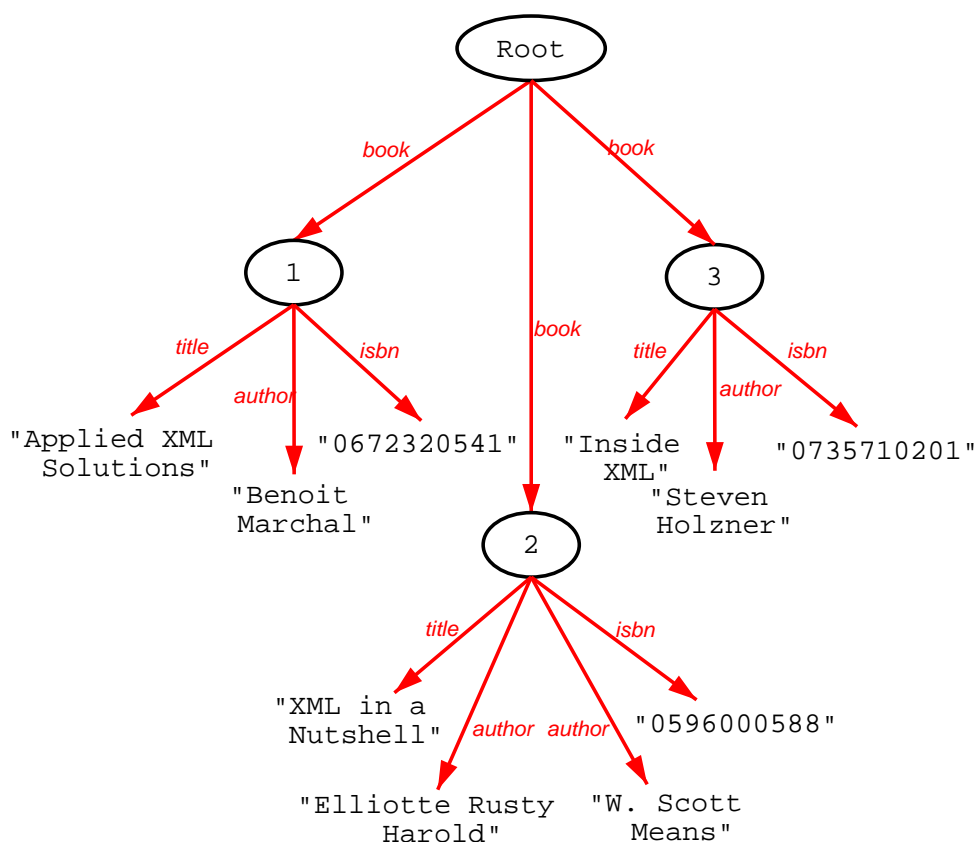


Abbildung 1: Baumgraph

4 Mappingverfahren ohne Verwendung von DTDs

Ausgangspunkt für alle Mappingverfahren ist die Herausbildung eines Graphen. Der Graph wird direkt aus der Ordnung des XML-Dokumentes abgeleitet, so dass sich eine Baumstruktur herausbildet.

Vereinfacht ausgedrückt sind in diesem Fall die Kanten Attribute, die inneren Knoten die Elemente, die Blätter des Graphenbaumes die Werte. Der Kantename ist immer der Attributname. Darüber hinaus bekommt jeder innere Knoten eine *unique identifier*, womit später Objekte mit gleichem Namen unterschieden werden können. Referenzen werden durch querverweisende Bögen dargestellt.

Der Baumgraph aus dem Beispiel sieht dann vereinfacht so aus:

Die einfachste Art eines Schemas ist das Speichern aller Attribute in einer Tabelle. Dazu werden die Kanten in folgender Weise in der Tabelle aufgenom-

Quellnr.	Ordnung	Name_Att.	Int_Kan_Fl	Zielobjekt
1	1	ISBN	0	06272320541
1	2	Author	0	Benoit Marchal
1	3	Title	0	Applied XML Solutions
2	4	ISBN	0	0596000588
2	5	Author	0	Elliotte Rusty
2	6	Author	0	W. Scott Means
2	7	Title	0	XML in a Nutshell
3

Tabelle 1: Kantentabelle für das XML-Dokument

men:

Kante(Quellnummer, Ordnung, Name_Attribut, interne_Kante_Flag, Zielobjekt)

Quellnummer beinhaltet die Nummer des in der Hierarchie oben liegenden Objektes. *Ordnung* gibt die Position der Kante wieder, um die Struktur des Dokumentes rekonstruieren zu können. Der Name des Attributes wird in *Name_Attribut* gespeichert. Das Flag *interne_Kante_Flag* repräsentiert den Unterschied, ob die Kante auf ein weiteres Objekt oder einen Wert zeigt. *Zielobjekt* beinhaltet entweder wieder eine Objektnummer, oder den Wert des jeweiligen Attributes. Der Schlüssel der Tabelle wird durch $\{ Quellnummer, Ordnung \}$ erzeugt.

Zum Speichern der Werte gibt es zwei Möglichkeiten. Entweder werden die Werte zusammen mit den Attributen oder in separaten Wertetabellen gespeichert.

5 Der Attributansatz

Ein geringfügig geänderter Ansatz gegenüber dem Kantenansatz wird hier vorgestellt. Es werden alle Attribute mit dem gleichen Namen in einer eigenen Tabelle gespeichert. Die Gliederung der Attributtabelle gestaltet sich derart:

Attributname(Quellnummer, Ordnung, interne_Kante_Flag, Zielobjekt)

Die einzelnen Bezeichnungen sind in der Bedeutung dem vorangegangenen Ansatz gleich. Der Schlüssel wird ebenfalls wieder durch $\{ Quellnummer, Ordnung \}$ gebildet.

Quellnr.	Ordnung	Name_Att.	Int_Kan_Fl	Zielobjekt
1	1	ISBN	0	06272320541
1	2	Author	0	Benoit Marchal
1	3	Title	0	Applied XML Solutions
2	4	ISBN	0	0596000588
2	5	Author	0	Elliotte Rusty
2	6	Author	0	W. Scott Means
2	7	Title	0	XML in a Nutshell
3

6 Ansatz mit einer universellen Tabelle

Vom Kantenansatz und Attributansatz ausgehend kann man noch eine universelle Tabelle kreieren. Es werden wieder alle Attribute eines XML-Dokumentes in einer Tabelle gespeichert. Die Besonderheit besteht darin, dass diese Tabelle aus einem outer join, einer äußeren Vereinigung, aller Attributtabelle resultiert. Die Kopfzeile der universellen Tabelle wird in folgender Weise gebildet:

*Universal(Quellnummer, Ordnung_{n1}, interne_Kante_Flag_{n1},
Zielobjekt_{n1}, Ordnung_{n2}, interne_Kante_Flag_{n2},
Zielobjekt_{n2}, ..., Ordnung_{nk}, interne_Kante_Flag_{nk},
Zielobjekt_{nk})*

n1 bis *nk* sind Namen aller enthaltenen Attribute.

Wie deutlich zu ersehen, ist diese Art der Speicherung nicht optimal. Die Null-Werte müssen in der Datenbank ebenfalls mit gehalten werden, von der Mehrfachspeicherung der Werte einmal abgesehen.

7 Ansatz mit einer normalisierten universellen Tabelle

Die normalisierte Variante baut auf der universellen Tabelle auf. Der Unterschied besteht in der gesonderten Behandlung von Vorgabelisten der Attribute. Diese Attribute werden in zusätzlichen Tabellen, den Überlauftabellen, gespeichert. Die Art, wie diese Tabellen erstellt werden, kann dem Attributansatz entnommen werden.

Die Tabellen werden wie folgt erstellt:

```
UniNorm( Quellnummer, Ordnungn1, interne_Kante_Flagn1,  
Zielobjektn1, Ordnungn2, interne_Kante_Flagn2,  
Zielobjektn2, ..., Ordnungnk,  
interne_Kante_Flagnk, Zielobjektnk)
```

```
Überlaufn1(Quellnummer, Ordnung, interne_Kante_Flag, Zielobjekt)
```

```
...
```

```
Überlaufnk(Quellnummer, Ordnung, interne_Kante_Flag, Zielobjekt)
```

n1 bis *nk* sind die Namen aller enthaltenen Attribute.

8 Das Monet Modell

Beim Monet Modell wird das XML Dokument³ als Syntax Baum dargestellt (siehe Abbildung 2).

Im Gegensatz zu den bisher vorgestellten Beispielen werden alle Elemente nicht in einer einzigen Tabelle gespeichert, sondern für jeden (Teil-)Pfad des Baumes wird eine Relation erzeugt.

Man unterscheidet dabei drei Arten von Relationen:

1. Verbindungen zwischen Knoten
2. Knoten und Attributpaare
3. Knoten und Rang

Beispiel 1 (Seite 5) erzeugt dann die Relationen

³Zusätzlich zu den XML Daten wird noch die Ordnung der Elemente mit abgespeichert um das Dokument wieder aus der Datenbank erzeugen zu können

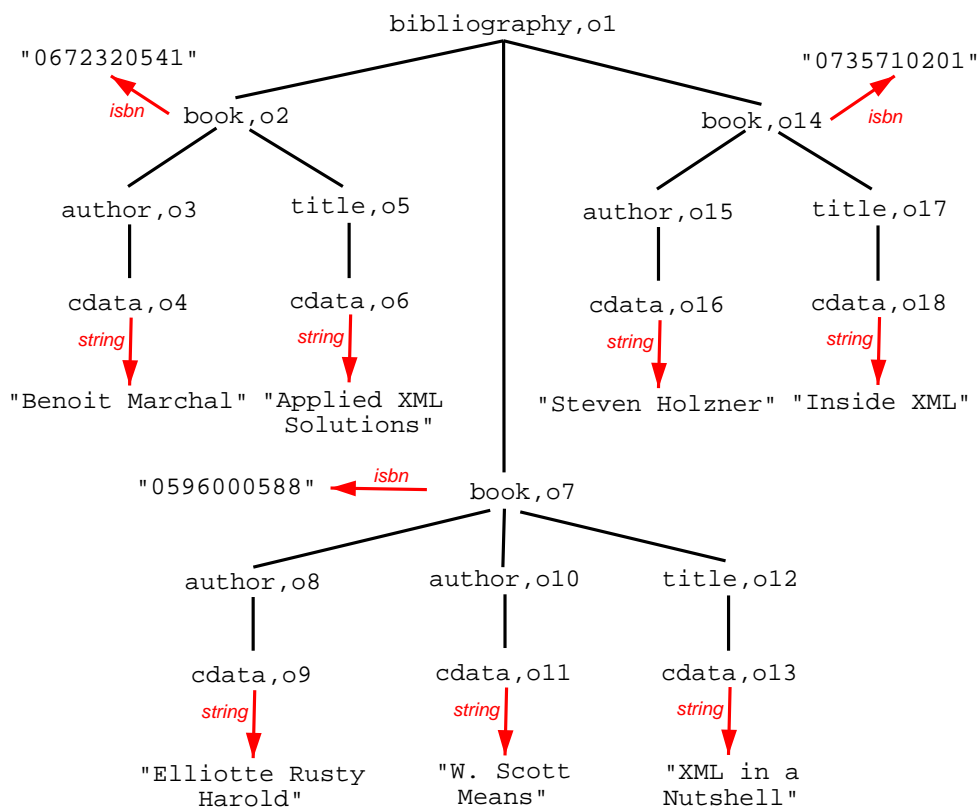


Abbildung 2: Syntax Baum

```

bibliography→book == {(o1, o2), (o1, o7), (o1,o14)}
bibliography→book⇒isbn == {(o2, '0672320541'), (o7, '0596000588'),
(o14, '0735710201')}
bibliography→book→author == {(o2, o3), (o7, o8), (o7, o10), (o14, o15)}
bibliography→book→author→cdata == {(o3, o4), (o8, o9), (o10, o11),
(o15, o16)}
bibliography→book→author→cdata ⇒string == {
(o4, 'Benoit Marchal'), (o9, 'Elliotte Rusty Harold'),
(o11, 'W. Scott Means'), (o16, 'Steven Holzner')}
bibliography→book→title == {(o2, o5), (o7, o12), (o14, o17)}
bibliography→book→title→cdata == {(o5, o6), (o12, o13), (o17, o18)}
bibliography→book→title→cdata⇒string == {
(o6, 'Applied XML Solutions'), (o13, 'XML in a Nutshell'),
(o18, 'Inside XML')}

```

Beispiel 3: Relationen im Monet Modell

Dieser Ansatz ermöglicht einerseits eine automatische Umwandlung, da die Tabellen eindeutig durch Pfade gegeben werden, andererseits vermeidet er weitgehend die Speicherung von Nullwerte und zusätzlichen Verwaltungsflags in der Datenbank. Da XML Tags, Element- und Attributnamen nicht innerhalb der Tabellen gespeichert werden, sondern in den Namen der Tabelle kodiert sind, wird die mehrfache Speicherung von redundanten Daten minimiert. Dadurch läßt sich die Datenbankgröße beschränkt halten. Bei konkreten Daten heißt das, die Datenbankgröße ist linear von der Größe der zugrundeliegenden XML Dokumentes abhängig. Sie kann in Grenzfällen aber sogar kleiner sein.

Durch die Aufspaltung in binäre Relationen ist es auch sehr einfach, nachträglich strukturelle Änderungen am XML Dokument in die Datenbank aufzunehmen, ohne sie komplett neu erzeugen zu müssen. Die einzigen Operationen die dazu nötig sind, ist das Umbenennen von Tabellen (Element- oder Attributnamen wurden geändert), das neu Erzeugen von Tabellen (es kamen neue Element hinzu) oder das Löschen von Tabellen (Element wurden entfernt).

Wie Anfragen möglich sind und umgeformt werden, zeigt sich am besten anhand eines einfachen Beispiels:

```

SELECT p
FROM bibliography->book p,
      p->author->cdata a,
      p->title->cdata t
WHERE a="Benoit Marchal" AND t LIKE "XML";

```

Es sollen also alle Artikel, daß heißt ihre Attribute und Unterelemente ausgegeben werden, die von *Benoit Marchal* geschrieben wurden und in deren Titel *XML* vorkommt. Beim Ausführen der Anfrage müssen zunächst zwei verschiedene Typen von Variablen unterschieden werden. Zum Einem Variablen, die Mengen enthalten (in unserem Beispiel *p*) und zum Anderem Variablen die Verbindungen ausdrücken (*a* und *t*). Die für das Ergebnis nötigen Pfade erhält man, falls sie nicht schon in der Datenbank erhalten sind, indem die binären Relationen entlang der Pfade über Joins verbunden werden. Damit wird eine Relation zwischen dem ersten und dem letzten Element des neuen Pfades erzeugt, die Einschränkungen aus der *WHERE* Klausel können dann einfach auf die entsprechenden Pfade angewendet werden. Diese Vorgehensweise ermöglicht es auch, die Anfragen mit einer Einschränkung der Pfade (als z.B Reguläre Ausdrücke) zu verbinden.

Zunächst scheint diese Methode durch die vielen nötigen Joins Probleme mit der Laufzeit von Anfragen zu haben. Untersuchungen[3] mit bis zu 1GB Datenbankgröße haben allerdings gezeigt, daß diese Befürchtungen unbegründet sind, da die Joins immer nur auf kleinen Datenmengen ausgeführt werden.

Ein Nachteil des Monet Modells, ist daß Rekursion und Verlinkung nicht explizit behandelt werden.

Beliebig tiefe Rekursion läßt sich aber relativ einfach hinzufügen indem man auch Teilpfade die ihren Ursprung nicht in der Wurzel des Baumes haben, zuläßt. Dadurch wird aber Zusatzaufwand nötig, wenn das XML Dokument aus der Datenbank rekonstruierbar bleiben soll. Außerdem kann es dabei zu Problemen kommen wenn Anfragen nur Teile des XML Dokumentes als Antwort liefern. Referenzen innerhalb dieser Menge auf Elemente die in der Antwortmenge nicht vorkommen dürfen nicht auftreten. Sobald ein Element aber mehrmals in der Antwort vorkommt, sollen natürlich alle weiteren Vorkommen als Referenzen realisiert werden um die logische Struktur des ursprünglichen XML Dokumentes nicht zu zerstören.

9 Fazit

Die verschieden hier vorgestellten Ansätze bieten alle die Möglichkeit XML Daten in einem RDBMS zu speichern. Viele der Ansprüche an ein System zur Speicherung von XML Daten, werden von ihnen erfüllt. Allerdings bleibt die Einschränkung, daß nicht alle XML Dokumente gleichermaßen für diese Art von Speicherung geeignet sind. Es zum Teil durchaus Vorteile haben (oder sogar nötig sein), die relationale Speicherung bei der Festlegung der Struktur des XML Dokumentes im Hinterkopf zu behalten um Probleme zu vermeiden oder Zugriffsgeschwindigkeiten zu optimieren.

Bis auf weiteres sind die beschriebenen Ansätze, gerade auch wegen der einfachen Integration von bereits vorhanden Daten, interessante Lösungen zur Speicherung von XML. Ob andere Systeme die Ansprüche in Stabilität, Ausgereiftheit und Skalierbarkeit erfüllen können wird sich in Zukunft zeigen müssen.

Literatur

- [1] D. Florescu, D. Kossmann *Storing and Querying XML Data using an RDBMS*. Data Engineering Bulletin, 22(3), 1999.
- [2] D. Florescu, D. Kossmann *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database*. 1999
- [3] A. Schmidt, M. Kersten, M. Windhouwer, F. Waas. *Efficient relational storage and retrieval of xml documents* . Proceedings of WebDB, Dallas, TX, May 2000.
- [4] J. Shanmugasundaram, K. Tufté, G. H. C. Zhang, D. DeWitt, J. Naughton *Relational Databases for Querying XML Documents: Limitations and Opportunities*. Proceedings of 25th International Conference on Very Large Data Bases, Edinburgh, Scotland, pages 302-304, 1999.
- [5] D. Carlisle *xmltex: A non validating (and not 100% conforming) namespace aware XML parser implemented in T_EX*. 2000-02-02