Seminar:
XML und Datenbanken

Speicherung von XML in (objekt-) relationalen Datenbanken

Burkhard Schäfer
WS 2002/2003

12. Februar 2003

Inhaltsverzeichnis

Teil I

Nutzung von Datenbanken für XML-Anwendungen

1 Motivation

Durch Einsatz eines Datenbanksystems (DBS) ist es möglich, einige Eigenschaften eines DBS ohne großen Aufwand für eine XML-Anwendung nutzbar zu machen. So kann auf bewährte Funktionalität zurückgegriffen und der Entwicklungsprozess verkürzt werden.

Unter anderem kann eine XML-Anwendung auf diese Weise Transaktionen und Integritätsregeln nutzen. Weiterhin kann auf ein großes Angebot bestehender Werkzeuge zurückgegriffen werden.

1.1 Transaktionsmanagement

Es ist wünschenswert, Operationen auf XML-Dokumenten zu Transaktionen zusammenfassen zu können. Dies ist sinnvoll, da ein Dokument in der Regel auf zahlreiche Tupel verteilt ist, die nicht alle zeitgleich geändert werden können.

Im Folgenden werden die aus der Datenbankwelt bekannten *ACID-Eigenschaften* im Hinblick auf die Verarbeitung von XML-Dokumenten näher untersucht.

Atomicity. Ein XML-Dokument wird bei Abbildung auf ein relationales Schema gewöhnlich auf mehrere Tupel verteilt. Entsprechend sind von Updates auf ein Dokument meistens auch mehrere Tupel betroffen. Dabei sollten entweder *alle* oder *keine* der Modifikationen durchgeführt werden („all or nothing").

Consistency. Die Datenbank soll über definierte Integritätsregeln (siehe 1.2) sicherstellen, dass spätestens bei Abschluss einer Transaktion nur gültige Dokumente in der Datenbank abgelegt sind.

Isolation. Ziel ist es, die Operationen eines Benutzers vor anderen, parallel arbeitenden Benutzern zu verbergen. Diese Isolation ist aufgrund der Baumstruktur allerdings nicht trivial zu gewährleisten und führt bei Abbildungen von XML-Dokumenten auf Tabellenstrukturen zu starker Einschränkung von Parallelität.

Durability. Bei erfolgreichem Abschluss einer Transaktion (COMMIT) sollen alle Änderungen endgültig und dauerhaft auf den Datenträger geschrieben werden. Selbst bei Ausfällen stellt das System die Integrität der Daten sicher.

1.2 Integritätsregeln

Mit Techniken wie *Document Type Definitions* (DTD) und *XML Schema* ist es möglich, die Struktur und das Vokabular von XML-Dokumenten festzulegen. Wenn solche Dokumente in Datenbanksystemen abgelegt werden, ist es wünschenswert, die Einhaltung dieser Strukturvorgaben auf Datenbankebene durchsetzen zu können. Dies kann durch Einsatz von Integritätsregeln geschehen.

1.3 Verfügbarkeit von Werkzeugen

Heute steht eine breite Palette von Werkzeugen zur Wartung und Pflege (objekt-) relationaler Datenbanksysteme zur Verfügung. Diese können nun auch für XML-Anwendungen nutzbar gemacht werden.

Zu nennen wären hier Komponenten zur Leistungs-Optimierung (Analyzer, Optimizer) sowie Backup- und Query-Werkzeuge und standardisierte Schnittstellen wie ODBC und JDBC. Es stehen standardisierte und akzeptierte Anfragesprachen zur Verfügung, die allerdings eher nicht für die Manipulation von Dokumenten geeignet sind.

Weiterhin ergeben sich Vorteile bei der Administration, da ein Systemverwalter keinen zusätzlichen Einarbeitungsaufwand auf sich nehmen muss.

2 Anforderungen

XML-Dokumente können aufgrund ihrer Struktur nicht direkt in einem (objekt-) relationalen Datenbanksystem ((O)RDBS) abgelegt werden. Hierzu muss eine Abbildung von XML-Strukturen in ein (objekt-) relationales Schema definiert werden, anhand der eine Transformation von Dokumenten in Datenbank-Tupel vorgenommen werden kann.

Darüber hinaus ist es nötig, die Anfragen, die an das System gerichtet werden sollen, ebenfalls zu transformieren, damit sie von einem (O)RDBS ausgewertet werden können.

2.1 Transformation von Dokumenten

Um XML-Dokumente in einer relationalen Datenbank abzulegen, müssen diese in geeigneter Form transformiert werden. In der Regel zerlegt man sie in Fragmente, die dann in Tupeln abgelegt werden.

Bei der Wahl der Abbildung ist es wichtig, dass ein Dokument wieder vollständig und ohne Änderung seiner Bedeutung aus der Datenbank zurückgewonnen werden kann. Es muss also im Besonderen der Dokumentbaum rekonstruiert werden können.

Dabei ist zu berücksichtigen, dass in einem relationalen Schema das Konzept der Ordnung bzw. die Reihenfolge von Tupeln keine Rolle spielt, während die Dokumentordnung in XML semantisch bedeutsam ist.

Es kann sinnvoll sein, bei der Festlegung einer Abbildungsstrategie, die DTD oder das Schema der abzulegenden Dokumente zu kennen. So ist es möglich, eine Abbildung spezifisch auf die Zielanwendung hin zu optimieren.

2.2 Transformation von Anfragen

Sind Dokumente in der Datenbank abgelegt, so sollen diese über Anfragen selektiert und abgerufen werden können. Dazu müssen Anfragen, die in Sprachen wie XQuery oder XML-QL formuliert werden, in SQL umgesetzt werden, um sie so für die Datenbank aufzubereiten.

Genauso sind die Ergebnisse einer Anfrage wieder in XML zurück zu transformieren, um diese in der Client-Anwendung weiter verarbeiten zu können.

Kommt eine dynamische Abbildungsstrategie zum Einsatz, so ist es zusätzlich erforderlich, die Abbildung selbst zu beschreiben und geeignet zu hinterlegen.

Teil II

Ansätze

Es ist möglich, XML-Dokumente trivial als Inhalt eines BLOB-Feldes in Datenbanken abzulegen. Dies ist jedoch häufig nicht sinnvoll, da auf diese Weise Anfragen auf die abgelegten Dokumente nur schwer durchführbar sind.

Aus diesem Grund sind zahlreiche Ansätze entstanden, die ein XML-Dokument in verschiedene Abschnitte aufteilen („shredding") und diese in geeigneter Form in (objekt-)relationalen Tabellen ablegen.

Grundsätzlich kann zwischen zwei Klassen von Abbildungen unterschieden werden, den modellorientierten und den strukturorientierten. Bei modellorientierten Ansätzen ist die Transformation an das XML-Dokumentmodell angelehnt, in dem sich jedes Element und Attribut mit einem eindeutigen Pfad adressieren lässt.

Strukturorientierte Ansätze bilden hingegen die logischen Zusammenhänge eines Dokuments im Relationenmodell ab.

3 Modellorientierte Ansätze

Die Tatsache, dass XML-Dokumente hierarchisch organisiert sind, kann ausgenutzt werden, um eine Transformation in ein relationales Datenbankschema zu modellieren.

Innerhalb eines Dokuments kann man jedes Element und jedes Attribut mit einem eindeutigen Pfad durch den Dokumentenbaum adressieren. Hierzu können Techniken wie XPath herangezogen werden.

Speichert man den Inhalt eines Elements/Attributs zusammen mit diesem Pfad (und einigen Zusatzinformationen), kann ein XML-Dokument recht einfach in einem relationalen Datenbankschema abgelegt werden.

Ein Vorteil eines solchen Ansatzes ist, dass abzulegende XML-Dokumente nicht einer bestimmten DTD oder einem bestimmten Schema genügen müssen. So sind für unterschiedliche Dokumenttypen weniger Tabellen nötig.

3.1 XRel: ein pfadorientierter Ansatz

In [1] stellen Yoshikawa und Amagasa einen Ansatz namens *XRel* vor, der XML-Dokumente entlang ihrer Baumstruktur zerlegt und auf mehrere relationale Tupel verteilt. Dabei wird ein aus vier Tabellen bestehendes SQL92-Schema zu Grunde gelegt.

Innerhalb dieses Schemas kann jedes beliebige wohlgeformte XML-Dokument abgelegt werden. Eine DTD oder ein Schema, worin die Struktur des Dokuments beschrieben wird, ist nicht erforderlich.

Abbildung 1 zeigt exemplarisch ein XML-Dokument, anhand dessen der XRel-Ansatz veranschaulicht werden soll.

```
<buch>
   <titel>Professional XML</titel>
   <isbn>1-861005-05-9</isbn>
   <preis waehrung="EUR">60,44</preis>
   <autoren>
      <autor geschlecht="m">
          <vorname>Mark</vorname>
          <nachname>Birbeck</nachname>
      </autor>
      <autor geschlecht="m">
          <vorname>Jon</vorname>
          <nachname>Duckett</nachname>
      </autor>
   </autoren>
</buch>
```

Abbildung 1: Beispieldokument

3.1.1 Baumdarstellung eines Dokuments

Jedes XML-Dokument lässt sich als Baum darstellen, in dem die hierarchischen Zusammenhänge zwischen Dokumentknoten dargestellt werden. Ein solcher Baum besitzt dabei Knoten der folgenden Typen:

Wurzelknoten. Der Wurzelknoten repräsentiert ein komplettes Dokument. Da der Wurzelknoten immer genau ein Element aggregiert, kann das Dokument vereinfacht auch über das Wurzel*element* beschrieben werden. Insofern werden Wurzelknoten im weiteren Verlauf nicht mehr betrachtet.

Elementknoten. Für jedes Element im Dokument wird im Baum ein Elementknoten angelegt. Es ist möglich, Elemente zu schachteln. Damit kann ein Elementknoten sowohl Textknoten als auch andere Elementknoten als Kinder besitzen. (Attribute werden im XRel-Ansatz aus Effizienzgründen nicht als Kinder von Elementen angesehen.)

Attributknoten. Elemente werden durch Attribute genauer beschrieben. Daher hat ein Attributknoten stets genau einen Elementknoten als Vater, besitzt jedoch keine Kindknoten.

Textknoten. In Textknoten werden Zeichenketten abgelegt, die den Inhalt eines Dokuments ausmachen. Textknoten sind im Baum stets unterhalb von Elementknoten angeordnet.

Die XML-Spezifikation sieht zusätzlich noch Namespace-Knoten, Verarbeitungsanweisungen und Kommentare vor, die bei XRel jedoch nicht berücksichtigt werden.

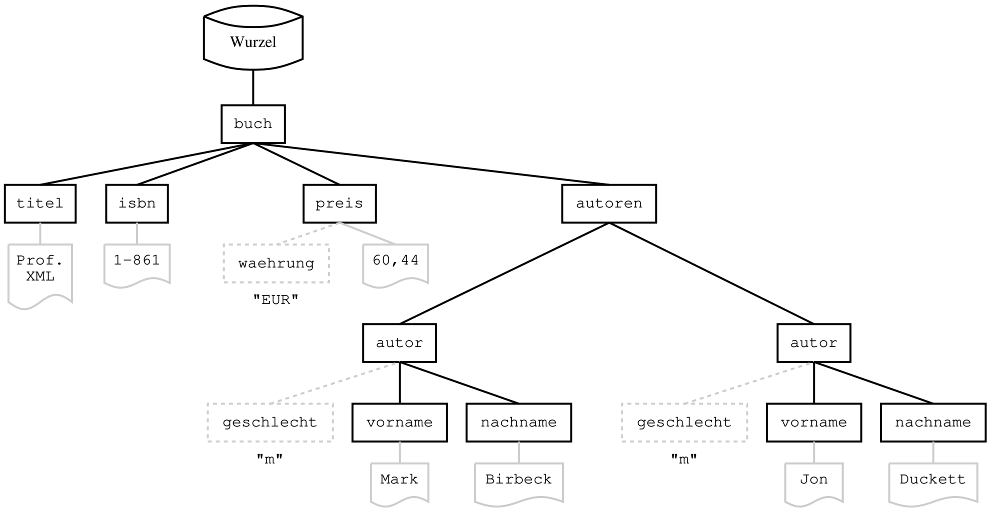

Abbildung 2: Baumdarstellung des Beispieldokuments

3.1.2 Pfadausdrücke

Ziel des XRel-Ansatzes ist, die Knoten des Dokumentenbaums relational darzustellen. Dazu wird zunächst eine Nomenklatur eingeführt, mit der die Lage von Knoten im Baum beschrieben werden kann: *Pfadausdrücke*.

Pfadausdrücke sind eng an die Syntax von XPath [2] angelehnt. Sie folgen der in Abbildung 3 dargestellten Grammatik.

```
SimplePathExpr  ::=  '#/' Step
                     | SimplePathExpr '#/' Step
Step            ::=  NameTest
                     | '@' NameTest
NameTest        ::=  QName
```

Abbildung 3: Syntax für Pfadausdrücke

Um den Pfadausdruck für einen Knoten zu bestimmen, wird der Baum von der Wurzel an durchlaufen und an jedem Elementknoten, der traversiert wird, die Zeichenfolge `#/` sowie der Name des Elements an den Pfad angehängt. Trifft man auf ein Attribut, wird die Zeichenfolge `#/@` sowie der Attributname verwendet. Die genannten Trennzeichen wurden gewählt, um eine einfache Verarbeitung von XPath-Anfragen zu ermöglichen (siehe 3.1.6). Textknoten erben den Pfad des übergeordneten Elementknotens.

Beispiel 1: Das Attribut `waehrung` besitzt den Pfadausdruck

```
#/buch#/preis#/@waehrung
```

Beispiel 2: Das Element `vorname`, das den Autor „Mark Birbeck" näher beschreibt, hat den Pfad

```
#/buch#/autoren#/autor#/vorname
```

Hier wird deutlich, dass Pfadausdrücke nicht eindeutig sind: Das Element `vorname` des rechten `Autor`-Zweigs wäre durch den gleichen Pfadausdruck auffindbar.

3.1.3 Knotenbeschreibung

Um nun einen Knoten eindeutig beschreiben zu können, wird zusätzlich zu dem zugehörigen Pfadausdruck auch die *Dokumentregion* gespeichert, in der sich dieser befindet. Dabei handelt es sich um die (ab Dokumentanfang in Byte gemessene) absolute Start- und Endposition der zum Knoten gehörenden Zeichenkette.

Es wäre nun denkbar, für jeden betrachteten Knotentyp eine relationale Tabelle einzuführen, etwa so:

```
Element (docID, path, start, end)
Attribute (docID, path, start, end, value)
Text (docID, path, start, end, value)
```

Bei dem Feld `docID` handelt es sich stets um einen eindeutigen Bezeichner für ein Dokument. Dies ist nötig, da mehrere Dokumente in dem Schema abgelegt werden sollen.

Für jeden Knoten ist im Feld `path` der korrespondierende Pfadausdruck abgelegt. Die Felder `start` und `end` enthalten die absoluten Positionen im Dokument.

Zusätzlich enthalten die Tabellen `Attribute` und `Text` noch ein `value`-Feld zum Abspeichern des Werts. Dieses ist für Elemente nicht nötig, da deren Inhalte bereits durch Attribute und Textfelder beschrieben sind.

Es ist nicht notwendig, den Namen der Attribute oder Elemente explizit abzulegen, da dieser bereits im Pfadausdruck enthalten ist.

Da zu erwarten ist, dass Pfadausdrücke, die im Feld path abgelegt werden, mehrfach vorkommen werden, bietet es sich an, diese in eine zusätzliche Tabelle auszulagern und mit einem Fremdschlüssel auf sie zu verweisen. Dies ist besonders dann sinnvoll, wenn mehrere Dokumente des gleichen Typs verarbeitet werden.

Wie bereits in 2.1 erwähnt, muss bei XML-Dokumenten die Reihenfolge eingehalten werden. Diese lässt sich zwar über die `start` und `end`-Felder rekonstruieren, allerdings ist dies nicht sehr elegant. Daher sieht XRel in der Element-Tabelle zusätzlich die Felder index und reindex vor, über die die Reihenfolge von Elementen, die sich auf der gleichen Ebene befinden, festlegen.

Aus den genannten Gründen verwendet die XRel-Methode das folgende relationale Datenbankschema:

```
Element (docID, pathID, start, end, index, reindex)
Attribute (docID, pathID, start, end, value)
Text (docID, pathID, start, end, value)
Path (pathID, pathexp)
```

3.1.4 Repräsentation eines Dokuments

Um ein Dokument in das XRel-Schema einzufügen, wird zunächst die Path-Tabelle generiert. Dazu wird das Dokument zunächst in eine Baumdarstellung überführt und für jeden Knoten der Pfadausdruck erzeugt.

Wird dieser Schritt auf das Beispieldokument in Abbildung 1 angewandt, so ergibt sich folgende Belegung der `Path`-Tabelle:

pathID	**pathexp**
1	`#/buch`
2	`#/buch#/titel`
3	`#/buch#/isbn`
4	`#/buch#/preis`
5	`#/buch#/preis#/@waehrung`
6	`#/buch#/autoren`
7	`#/buch#/autoren#/autor`
8	`#/buch#/autoren#/autor#/@geschlecht`
9	`#/buch#/autoren#/autor#/vorname`
10	`#/buch#/autoren#/autor#/nachname`

Abbildung 4: Beispielbelegung der `Path`-Tabelle

Daraufhin wird das Dokument sequenziell durchlaufen und für jeden Element-, Attribut- und Textknoten ein Eintrag in der korrespondierenden Tabellen vorgenommen. Bei Attributen wird als `start`- und `end`-Wert der um 1 inkrementierte `start`-Wert des zugehörigen Elements verwendet.

docID	**pathID**	**start**	**end**	**index**	**reindex**	*dargestelltes Element*
1	1	0	390	1	1	*buch*
1	2	11	41	1	1	*titel*
1	3	47	72	1	1	*isbn*
1	4	78	111	1	1	*preis*
1	6	117	381	1	1	*autoren*
1	7	134	246	1	2	*autor (1)*
1	9	168	190	1	1	*vorname (1)*
1	10	203	230	1	1	*nachname (1)*
1	7	255	366	2	1	*autor (2)*
1	9	289	310	1	1	*vorname (2)*
1	10	323	350	1	1	*nachname (2)*

Abbildung 5: Beispielbelegung der `Element`-Tabelle

docID	**pathID**	**start**	**end**	**value**	*dargestelltes Attribut*
1	5	79	79	`EUR`	*waehrung*
1	8	134	134	`m`	*geschlecht*
1	8	255	255	`m`	*geschlecht*

Abbildung 6: Beispielbelegung der `Attribute`-Tabelle

docID	**pathID**	**start**	**end**	**value**
1	2	18	33	`Professional XML`
1	3	53	65	`1-861005-05-9`
1	4	99	103	`60,44`
1	9	177	180	`Mark`
1	10	213	219	`Birbeck`
1	9	298	300	`Jon`
1	10	333	339	`Duckett`

Abbildung 7: Beispielbelegung der `Text`-Tabelle

Somit ist das Beispieldokument vollständig in der Datenbank abgelegt.

3.1.5 Rekonstruktion von Dokumenten

Soll ein Dokument aus der Datenbank wieder in seine ursprüngliche XML-Darstellung überführt werden, kann die folgende Methode zum Einsatz kommen:

1. Selektieren des Root-Elements eines Dokuments

   `SELECT * FROM Element WHERE start = 0 AND docID=`*DokumentNr*

2. Allokieren eines Arrays zur Aufnahme des Dokuments. Als Größe wird der `end`-Wert des gerade selektierten Datensatzes verwendet. Das Array wird mit Leerzeichen initialisiert.

3. Selektieren aller Text-Knoten eines Dokuments und Einfügen an die durch `start` und `end` gegebenen Stellen des Dokument-Arrays

   `SELECT * FROM Text WHERE docID=`*DokumentNr*

4. Selektieren aller Elementknoten und Einfügen an die entsprechenden Stellen im Dokument-Array. Für jedes Element wird überprüft, ob es Attribute besitzt. Ist dies der Fall, werden diese ebenfalls übernommen.

   `SELECT * FROM Element WHERE docID=`*DokumentNr*
   `SELECT * FROM Attribut WHERE docID=`*DokumentNr* `AND start=`*elementStart* `+ 1`

Nach Abarbeitung des Algorithmus enthält das Array das gewünschte XML-Dokument.

3.1.6 Transformation von Anfragen

Da die Pfadausdrücke stark an die Syntax von XPath angelehnt sind, können XPath-Anfragen recht einfach durch eine Variante des String-Matching realisiert werden.

In einem XPath-Ausdruck ersetzt man die Zeichenkette '/' durch '#/' und die Zeichenkette '//' durch '#%/'. Der resultierende Ausdruck kann direkt für eine LIKE-Bedingung in einem SQL-Query auf die `Path`-Tabelle verwendet werden.

Dies soll an zwei Beispielen gezeigt werden:

Beispiel 1: Der XPath-Ausdruck `/buch` liefert alle Kinder des Knotens `<buch>...</buch>` zurück. Daraus resultiert folgende SQL-Anfrage:

```
SELECT   e.docID, e.start, e.end
FROM     Element e, Path p
WHERE    e.pathID = p.pathID
AND      p.pathexp LIKE '#/buch'
ORDER BY e.docID, e.start, e.end
```

Beispiel 2: Um alle Autoren zu erhalten, kann beispielsweise die XPath-Anfrage `/buch/*/autor` verwendet werden. Dem entspricht diese SQL-Anfrage:

```
SELECT   e.docID, e.start, e.end
FROM     Element e, Path p
WHERE    e.pathID = p.pathID
AND      p.pathexp LIKE '#/buch#%/autor'
ORDER BY e.docID, e.start, e.end
```

Nach Durchführung der genannten SQL-Queries müssen - ähnlich wie in 3.1.5 beschrieben - entsprechende Schritte eingeleitet werden, um untergeordnete Elemente, Attribute und Textknoten zu rekonstruieren. Für komplexere Anfragen wird in [1] ein Algorithmus angegeben.

Es fällt auf, dass mit XRel bei der Verarbeitung von Anfragen die zu Grunde liegende Dokumentstruktur keine Rolle spielt.

3.1.7 Bewertung des Verfahrens

Das XRel-Verfahren ist gut einsetzbar für Aufgaben, die das Speichern von zahlreichen unterschiedlich strukturierten XML-Dokumenten erfordern, beispielsweise für digitale Bibliotheken. Auch für Anwendungen, bei denen die DTDs bzw. Schemata für Dokumente nicht bekannt sind, bietet sich XRel an.

XPath-Anfragen sind recht einfach auszuführen. Allerdings ist das notwendige String-Matching bei großen und heterogenen Dokumentbeständen langsam und kann mittels Indexstrukturen durch die Datenbank nicht optimal unterstützt werden.

Wenn eine Anwendung die Speicherung von zahlreichen Dokumenten erfordert, die der gleichen Strukturbeschreibung genügen, empfiehlt sich die Wahl eines strukturorientierten Ansatzes.

3.2 Edge Tables

Während das XRel-Verfahren die Knoten des Dokumentenbaums betrachtet, existiert mit *Edge Tables* ein Ansatz, der sich an den Kanten orientiert.

3.2.1 Datenstruktur

Florescu und Kossmann beschreiben in [5] ein Modell zur Darstellung semistrukturierter Daten. Dieses wird für Edge Tables und einige andere Ansätze verwendet.

Das XML-Dokument wird dabei als gerichteter, beschrifteter Graph beschrieben. Als Element gegebene Datenobjekte werden durch Knoten repräsentiert, während Kanten Attribute darstellen. Dabei werden die Kanten mit den Attributnamen beschriftet. Referenzen werden ebenfalls als Kanten dargestellt. Attributwerte und Textknoten werden in Form von Blättern abgelegt. Jedes Objekt erhält zusätzlich einen eindeutigen numerischen Bezeichner (ID), mit der die Knoten beschriftet werden.

Zur Vereinfachung differenziert das Modell nicht zwischen Attributen und attributiv verwendeten Kind-Elementen.

Um die Reihenfolge innerhalb eines Dokuments zu erhalten, wird auf der Menge der Kanten eine totale Ordnung definiert.

Wird das Beispieldokument (Abb. 1) in dieser Struktur repräsentiert, so ergibt sich der in Abbildung 8 dargestellte Graph.

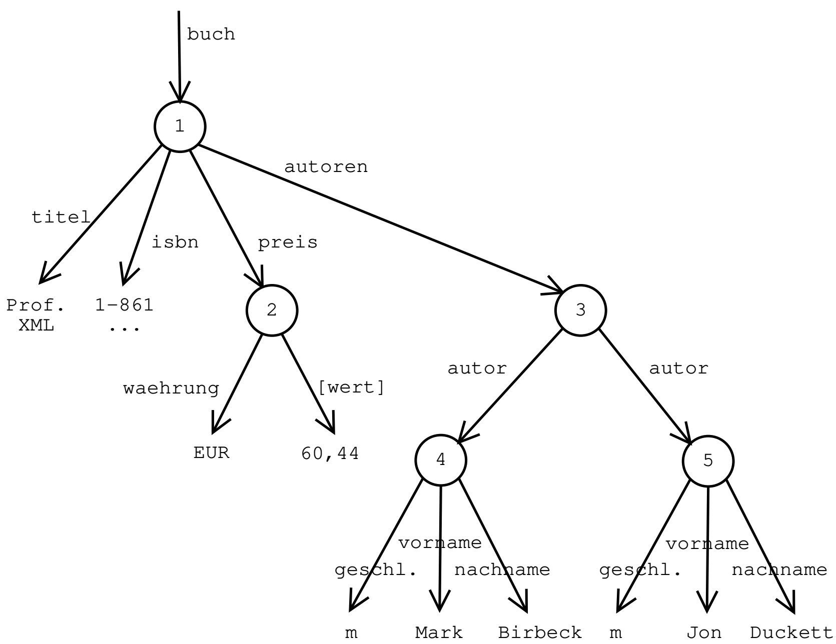

Abbildung 8: Graphdarstellung des Beispieldokuments

3.2.2 Darstellung von Dokumenten

Bei dem vorliegenden Ansatz wird lediglich eine relationale Tabelle benötigt, um die Dokumentstruktur abzubilden. Jede Zeile der Tabelle beschreibt eine Kante des Graphs. Die Tabelle ist wie folgt aufgebaut:

Edge (source, ordinal, name, flag, *target*)

Im `source`-Feld wird die Objekt-ID des Elements (Objekts) abgelegt, von dem die Kante abgeht. Das `ordinal`-Feld enthält eine Ordnungszahl, die die Dokumentreihenfolge von Attributen auf der gleichen Ebene festlegt. Der Name des Attributs, das durch die Kante dargestellt wird, steht im `name`-Feld.

Da eine Kante sowohl ein Wert eines Attributs als auch eine Referenz beschreiben kann, ist es nötig, im Feld `flag` den Kantentyp festzulegen. Bei Referenzen trägt man die Zeichenkette `ref` ein, während man bei Wertangaben den Datentyp des Werts (z.B. `int` oder `string`) hinterlegt.

Zur Darstellung von Werte schlagen Florescu und Kossmann zwei alternative Verfahren vor.

Separate Wertetabellen. Für jeden Datentyp wird eine eigene Tabelle erzeugt, die dann alle Werte des entsprechenden Typs aufnimmt.

$\texttt{V}_{int}$ (vid, value)
$\texttt{V}_{float}$ (vid, value)
$\texttt{V}_{string}$ (vid, value)

Die dort angelegten Tupel werden dann in der `target`-Spalte der `Edge`-Tabelle referenziert. Handelt es sich bei der beschriebenen Kante um eine Referenz, so wird in der `target`-Spalte die Objekt-ID des Zielelements eingetragen.

In den Abbildung 9 bis 11 wird das Verfahren anhand des Beispieldokuments veranschaulicht.

Inlining. Bei Inlining werden sowohl Kanten als auch Werte in der gleichen Tabelle dargestellt. Dazu wird die `Edge`-Tabelle um ein Feld pro Datentyp erweitert, das die jeweiligen Werte aufnehmen kann. Diese Struktur kann auch durch einen Verbund der Edge-Tabelle mit den oben vorgestellten Wertetabellen erzielt werden.

Es ergibt sich also folgendes Datenbankschema:

```
Edge (source, ordinal, name, flag,
      intValue, floatValue, stringValue, ..., target)
```

In die nicht benötigten Wert-Felder wird `NULL` eingetragen. Die target-Spalte wird nur im Fall von Referenzen benutzt, um auf die ID des Zielobjekts zu verweisen. Besitzt das Attribut in diesem Fall einen Wert, so wird das target-Feld mit NULL belegt.

source	**ordinal**	**name**	**flag**	**target**
0	1	`buch`	`ref`	1
1	1	`titel`	`string`	*v1*
1	2	`isbn`	`string`	*v2*
1	3	`preis`	`ref`	2
1	4	`autoren`	`ref`	3
2	1	`waehrung`	`string`	*v3*
2	2	`[wert]`	`float`	*v4*
3	1	`autor`	`ref`	4
3	2	`autor`	`ref`	5
4	1	`geschlecht`	`string`	*v5*
4	2	`vorname`	`string`	*v6*
4	3	`nachname`	`string`	*v7*
5	1	`geschlecht`	`string`	*v8*
5	2	`vorname`	`string`	*v9*
5	3	`nachname`	`string`	*v10*

Abbildung 9: Beispielbelegung der Tabelle Edge (separate Wertetabellen)

vid	**stringValue**
1	`Professional XML`
2	`1-861...`
3	`EUR`
5	`m`
6	`Mark`
7	`Birbeck`
8	`m`
9	`Jon`
10	`Duckett`

Abbildung 10: Beispielbelegung der Tabelle V_{string} (separate Wertetabellen)

vid	**floatValue**
4	`60,44`

Abbildung 11: Beispielbelegung der Tabelle V_{float} (separate Wertetabellen)

source	ordinal	name	flag	stringValue	floatValue	target
0	1	buch	ref	*null*	*null*	1
1	1	titel	string	Professional XML	*null*	*null*
1	2	isbn	string	1-861-...	*null*	*null*
1	3	preis	ref	*null*	*null*	2
1	4	autoren	ref	*null*	*null*	3
2	1	waehrung	string	EUR	*null*	*null*
2	2	[wert]	float	*null*	60,44	*null*
3	1	autor	ref	*null*	*null*	4
3	2	autor	ref	*null*	*null*	5
4	1	geschlecht	string	m	*null*	*null*
4	2	vorname	string	Mark	*null*	*null*
4	3	nachname	string	Birbeck	*null*	*null*
5	1	geschlecht	string	m	*null*	*null*
5	2	vorname	string	Jon	*null*	*null*
5	3	nachname	string	Duckett	*null*	*null*

Abbildung 12: Beispielbelegung der Edge-Tabelle (Inline-Verfahren)

3.2.3 Verarbeitung von Anfragen

Der vorgestellte Transformationsansatz erlaubt die Formulierung effizienter Anfragen auf Attributwerte. Hierzu wird der entsprechende Attributname in den `WHERE`-Teil einer SQL-Anfrage aufgenommen.

Anfragen, die die Dokumentstruktur betreffen, können über Verbunde und rekursive SQL-Konstrukte dargestellt werden. Durch Einführung von Objekt-IDs sind solche Anfragen effizient durchführbar. Mit Indices auf die Spalten {`source`, `ordinal`} sowie `target` lässt sich die Verarbeitungsgeschwindigkeit weiter steigern.

3.3 Attributtabellen

Als Alternative zu dem in 3.2 vorgestellten Edge-Ansatz wird in [5] ein Ansatz beschrieben, der alle gleichnamigen Attribute in jeweils eine eigene relationale Tabelle gruppiert.

Für jeden Attributnamen entsteht also eine Tabelle nach dem folgenden Muster:

$\mathtt{A}_{attrName}$ `(source, ordinal, flag,` *`target`*`)`

Die Wertedarstellung kann wie in 3.2.2 beschrieben mittels Inlining oder separaten Wertetabellen erfolgen.

3.4 Weitere Ansätze

Florescu und Kossmann beschreiben in [5] einige weitere interessante Verfahren: Durch Bildung eines äußeren Verbunds über alle Attributtabellen entsteht eine (denormalisierte) Struktur, in der sämtliche Dokumentdaten in nur einer Tabelle, der sog. *Universal Table* abgelegt werden. Diese Tabelle ist wie folgt aufgebaut:

$$
\begin{array}{ll}
\texttt{Universal} & \texttt{(source,} \\
& \texttt{ordinal}_{n1}\texttt{, flag}_{n1}\texttt{, target}_{n1}\texttt{,} \\
& \texttt{ordinal}_{n2}\texttt{, flag}_{n2}\texttt{, target}_{n2}\texttt{,} \\
& \texttt{...} \\
& \texttt{ordinal}_{nm}\texttt{, flag}_{nm}\texttt{, target}_{nm}\texttt{)}
\end{array}
$$

Naturgemäß enthalten hierbei zahlreiche Felder lediglich den Wert `NULL`. Außerdem entsteht gerade bei mehrwertigen Attributen Redundanz. Um dies zu vermeiden, schlägt [5] vor, mehrwertige Attribute in separaten Überlauftabellen abzulegen. Ähnlich wie im Attributansatz (siehe 3.3) wird für jeden Attributnamen, der im XML-Dokument auftritt, eine eigene Überlauftabelle angelegt.

Es entsteht also folgendes relationale Schema:

$$
\begin{array}{ll}
\texttt{UnivNorm} & \texttt{(source,} \\
& \texttt{ordinal}_{n1}\texttt{, flag}_{n1}\texttt{, target}_{n1}\texttt{,} \\
& \texttt{ordinal}_{n2}\texttt{, flag}_{n2}\texttt{, target}_{n2}\texttt{,} \\
& \texttt{...} \\
& \texttt{ordinal}_{nm}\texttt{, flag}_{nm}\texttt{, target}_{nm}\texttt{)} \\
\texttt{Overflow}_{n1} & \texttt{(source, ordinal, flag, target)} \\
\texttt{Overflow}_{n2} & \texttt{(source, ordinal, flag, target)} \\
\texttt{...} & \\
\texttt{Overflow}_{nm} & \texttt{(source, ordinal, flag, target)}
\end{array}
$$

Diese Methode ähnelt der Vorgehensweise objekt-relationaler Datenbanken (z.B. Oracle 8) bei der Verwaltung mehrwertiger Attribute.

4 Strukturorientierte Ansätze

Anders als modellorientierte Ansätze bilden die nun diskutierten Verfahren die tatsächliche logische Dokumenthierarchie ab. Solche strukturorientierten Ansätze analysieren in der Regel die in einer Beschreibungssprache hinterlegte Grammatik eines XML-Dokuments und bilden die darin festgelegten Objekttypen auf ein geeignetes (objekt-)relationales Konstrukt ab.

Für jede (durch eine DTD oder ein XML Schema gegebene) Klasse von XML-Dokumenten wird ein eigenes Schema erzeugt, in dem keine klassenfremden Dokumente abgelegt werden können.

4.1 X-Database: Relationale Abbildung

Ein Vertreter der strukturorientierten Ansätze wird von Varlamis und Vazirgiannis in [4] am Beispiel eines Videoarchivs beschrieben. Nach dem *X-Database*-Verfahren wird ein XML-Schema eingelesen und daraus ein relationales Datenbankschema erzeugt. In dieser Tabellenstruktur können dann beliebige Dokumente, die gültig im Sinne des gegebenen XML-Schemas sind, abgelegt werden.

Grundsätzlich wird für jeden Elementtyp eine eigene Relation erzeugt. Da in einem XML Schema komplexe Konstrukte von Elementen und Attributen definiert werden können, die so nicht direkt von einem relationalen Datenbanksystem angeboten werden, wird im Folgenden auf einige Spezialfälle eingegangen.

4.1.1 Abbildung von Attributen und Attributgruppen

Attribute eines Elements werden als Felder der zugehörigen Elementtabelle repräsentiert. Da Attributgruppen lediglich mehrere einfache Attrbute aggregieren, werden sie in der Elementtabelle in Form mehrerer Felder dargestellt.

So würden die Elemente `buch`, `autoren` und `autor` aus dem Beispieldokument (Abb. 1) in folgenden Tabellen abgelegt:

```
buch    (id)
autoren (id, parentBuchId)
autor   (id, parentAutorenId, geschlecht)
```

Jedes Element verweist per Fremdschlüsselbeziehung auf das ihm übergeordnete. Bei dem Wurzelelement ist das nicht nötig. Hier entspricht die Element-ID der Dokumenten-ID.

4.1.2 Abbildung einfacher Typen

Durch einfache Typen (*simpleTypes*) ist es möglich, die erlaubten Werte eines Attributs mittels einer Menge von Strings festzulegen. Die Feldbeschreibung im Datenbankschema wird hierfür mit einem geeigneten `CONSTRAINT` erweitert.

```
CONSTRAINT CHECK (Feldname IN ('Alternative1', ... ,
                              'AlternativeN'))
```

Das `geschlecht`-Attribut des `autor`-Elements würde also folgenden `CONSTRAINT` erhalten:

```
CONSTRAINT CHECK (geschlecht IN ('m', 'w'))
```

Somit kann das Datenbanksystem bereits einen Teil der durch das XML Schema gegebenen Regeln durchsetzen.

4.1.3 Sonderfälle

Jeder im XML Schema definierte komplexe Typ wird mit einer separaten Tabelle dargestellt, die per Fremdschlüsselbeziehung aus der Elementtabelle referenziert wird.

Da XML Schemata das Konzept der Vererbung unterstützen, sieht X-Database einen entsprechenden Abbildungsmechanismus vor. Dieser ist an das einschlägig bekannte Hausklassenmodell angelehnt.

4.1.4 Anfrageverarbeitung

X-Database bietet lediglich proprietäre Schnittstellen zur Anfrageverarbeitung. Anfragen werden als XML-Dokumente formuliert, die einem bestimmten Schema genügen müssen. Diese werden an ein Microsoft COM-Objekt übergeben, welches die Interaktionen mit einer Oracle-Datenbank und die Dokumenttransformationen durchführt.

Die genauen Mechanismen der Anfrageverarbeitung sind nicht dokumentiert. Aus der Quelle geht jedoch hervor, dass Unterstützung für die üblichen Operationen (Select, Insert, Update, Delete) besteht.

4.2 Abbildung über benutzerdefinierte Datentypen

Die bis jetzt betrachteten Ansätze basieren auf rein relationalen Datenbankschemata. Die im Folgenden diskutierte Methode macht sich die in SQL1999 enthaltenen objektorientierten Konzepte zunutze, um eine natürlichere Darstellung von XML-Dokumenten in Datenbanken zu erreichen.

Kudrass und Conrad schlagen in [6] vor, die in einer DTD deklarierten XML-Konstrukte mittels benutzerdefinierter Datentypen (UDTs) auszudrücken. Für jedes Element wird ein eigener UDT eingeführt. Eventuell vorhandene Attribute werden dabei als Felder übernommen. Es wird eine Tabelle angelegt, der der so entstandene UDT als Tupeltyp zugrunde liegt.

Subelemente, die lediglich Zeichenketten enthalten (`#PCDATA`), werden wie Attribute behandelt und erscheinen im UDT des übergeordneten Elements als Stringfeld. Falls ein Subelement selbst geschachtelte Elemente enthält, wird für dieses ein eigener UDT angelegt. Dieser wird von dem übergeordneten Element referenziert.

Da ein Dokumentenbaum nun vollständig über geschachtelte UDTs dargestellt wird, kann ein XML-Dokument mit nur einem INSERT-Befehl in die Datenbank eingefügt werden. Auch Anfragen auf Teilbäume sind durch die hierarchische UDT Querysyntax in SQL1999 einfach zu formulieren.

4.3 Bewertung von Abbildungsstrategien

In der vorliegenden Ausarbeitung wurden einige Abbildungsstrategien dargestellt, die auf unterschiedlichen Kernideen beruhen. Für einen Benutzer stellt sich nun die Frage, welcher Ansatz bei einem gegebenen Anforderungsprofil die höchste Leistung bietet.

Mit LegoDB wird in [3] ein Ansatz vorgestellt, der aus einem gegebenen XML-Schema eine Reihe möglicher Abbildungen erzeugt und bewertet sowie die günstigste auswählt. Um diese Aufgabe zu erfüllen, benötigt das Verfahren neben dem XML Schema einige statistische Angaben über die zu verarbeitenden XML-Daten sowie eine typische XQuery-Workload.

Der Ansatz macht sich die Tatsache zunutze, dass es mehrere XML Schemata geben kann, die dieselbe Klasse von XML-Dokumenten beschreiben. Dies liegt daran, dass z.B. identische Kardinalitätsangaben mittels unterschiedlicher regulärer Ausdrücke beschrieben werden können.

Auf dem gegebenen XML Schema führt LegoDB eine Reihe von Äquivalenztransformationen durch und spannt so einen Suchraum äquivalenter XML Schemata auf. Für jedes der so erzeugten XML Schemata wird unter Anwendung einer festen, strukturorientierten Abbildungsstrategie ein relationales Datenbankschema erzeugt. Zusätzlich werden die XQuery-Anweisungen in die zum jeweiligen Datenbankschema passenden SQL-DML-Befehle überführt.

Nun wird ein relationaler Optimizer verwendet, um die Ausführungszeiten des Lastprofils für die verschiedenen Datenbankschemata abzuschätzen. Auf diese Weise kann die für die Anwendung günstigste Abbildung bestimmt werden.

5 Zusammenfassung und Ausblick

Es kann festgestellt werden, dass sich (objekt-)relationale Datenbanksysteme gut für die Nutzung in XML-Anwendungen eignen. Sie stellen für die Datenhaltung zahlreiche wesentliche Funktionen zur Verfügung, die auch in der XML-Welt benötigt werden.

Das (objekt-)relationale Datenmodell ist jedoch nicht direkt für die Speicherung von XML-Dokumenten geeignet. Aus diesem Grund wurde in dieser Ausarbeitung auf eine Reihe unterschiedlicher Abbildungsstrategien eingegangen. Die Vergleichbarkeit dieser Strategien ist nicht unmittelbar gegeben; daher bieten sich Ansätze an, die unterschiedliche Abbildungen bewerten und vergleichen können.

Für die Zukunft wäre wünschenswert, Konzepte wie LegoDB (siehe 4.3) weiter zu verallgemeinern. Anwendungsentwickler benötigen Methoden, für ihre Domänen optimale Abbildungen zu finden. Wünschenswert ist daher ein Werkzeug, das selbst fundamental unterschiedliche Ansätze mittels realistischer Lastprofile miteinander vergleichen kann. In dieser Richtung sind sicher noch interessante Entwicklungen zu erwarten.

Literatur

[1] Yoshikawa, M., Amagasa, T.: *XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases*, ACM Transactions on Internet Technology (TOIT), Volume 1, Issue 1 (August 2001), pp. 110-141

[2] XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath

[3] Bohannon, P., Freire, J., Roy, R., Siméon, J.: *From XML Schema to Relations: A Cost-Based Approach to XML Storage*, Proc. Intl. Conference on Data Engineering (ICDE), 2002

[4] Varlamis, I., Vazirgiannis, M.: *Bridging XML-Schema and relational databases. A system for generating and manipulating relational databases using valid XML documents*, Proc. ACM Symposium on Document Engineering, 2001

[5] Florescu, D., Kossmann, D.: *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database*, Proc. VLDB'99, 1999

[6] Kudrass, T., Conrad, M.: *Management of XML Documents in Object-Relational Databases*, EDBT 2002 Workshops, LNCS 2490, pp. 210-227, 2002