RWTH Aachen
Wintersemester 2002 / 2003

Proseminar
„XML und semi-strukturierte Daten“

Lehrstuhl für Informatik V
Prof. Dr. M. Jarke

Ausarbeitung zum Thema
„XML und Datenbanken“

Mark Wiesemann

Betreuer: Dominik Lübbers

Inhaltsverzeichnis

1 Einleitung

XML-Datenbanken bieten Funktionalitäten wie Versions- und Transaktionskontrolle oder Zugriffssperren, die gleichzeitigen Schreibzugriff auf die gleichen Daten verhindern – also Funktionalitäten, die Benutzer bei der Arbeit mit XML-Dokumenten (wenn diese beispielsweise manuell im Dateisystem abgelegt werden) vermissen. Wenn von XML-Datenbanken gesprochen wird, muss zwischen drei Typen unterschieden werden (vgl. [TXI02]): native XML-Datenbanken (NXD), XML-erweiterte Datenbanken (XML-enabled databases, XED) und hybride XML-Datenbanken (HXD).

Native XML-Datenbanken wurden speziell für XML entwickelt. Der Begriff „native XML-Datenbank“ wurde von der Software AG geprägt – er tauchte erstmalig in der Werbekampagne für die von ihr entwickelte native XML-Datenbank „Tamino“ auf. Datenbanken dieses Typs verwenden XML als Datenmodell. Einfache Anfragen haben gegenüber anderen Datenbanksystemen Geschwindigkeitsvorteile, bei strukturfremden Anfragen ist die Performance nativer XML-Datenbanken meist schlechter. Je nach Anwendungszweck bietet die Spezialisierung auf XML-Daten aber einige Vorteile, auf die später genauer eingegangen wird.

XML-erweiterte Datenbanken sind bereits existierende Datenbanksysteme, die um XML-Funktionalitäten erweitert wurden. Hierbei handelt es sich vor allem um relationale Datenbanken, für die E. Codd 1970 das Modell vorgestellt hat. Relationale Datenbanken bestehen aus Tabellen (bzw. mathematischer ausgedrückt: Relationen), deren Spalten die Felder und deren Zeilen die Datensätze enthalten. Die Informationen werden meist auf mehrere Tabellen verteilt, um Redundanzen und eventuell daraus resultierende Probleme zu vermeiden. Der Primärschlüssel (primary key) ist eine Menge von Tabellenspalten, deren Werte jede Zeile eindeutig identifizieren, pro Tabelle gibt es nur einen Primärschlüssel. Der Fremdschlüssel (foreign key) ist eine Menge von Spalten, deren Werte auf den Primärschlüssel einer anderen Tabelle verweisen. In vielen relationalen Datenbanken können die gespeicherten Daten mit der Structured Query Language (SQL) abgefragt werden. Eine Verknüpfung mehrerer Tabellen wird als Join bezeichnet. Bekannte Datenbanksysteme, die SQL unterstützen, sind MySQL, Oracle oder Microsoft Access.

Hybride XML-Datenbanken sind eine Mischform aus NXD und XED, die genaue Einordnung hängt vor allem vom geplanten Einsatzzweck ab.

Kapitel 2 dieser Ausarbeitung zum Thema „XML und Datenbanken“ bietet Informationen über Strukturen und semi-strukturierte Daten sowie über XML-Datenmodelle, für deren Definition das XML Information Set verwendet wird. In Kapitel 3 folgt eine Definition der Begriffe „datenorientiertes Dokument“ und „textorientiertes Dokument“. Wie solche Dokumente gespeichert und ausgelesen werden können, wird in den Kapiteln 4 und 5 behandelt.

2 XML als Datenmodell

2.1 Strukturen

Wenn man Daten automatisiert verarbeiten möchte, muss man sie „strukturieren“, d.h. man muss sie in Einheiten unterteilen, Beziehungen zwischen diesen Einheiten finden und die Inhalte der Einheiten genau definieren.

Am Beispiel einer Rechnung wird dies leicht klar: Sie besteht aus vielen Einheiten wie z.B. Rechnungsnummer, Rechnungsdatum, Positionen, die wiederum aus Stückpreis, Anzahl und einer Bezeichnung bestehen, und dem Rechnungsempfänger, der sich aus dem Namen, der Straße, der Postleitzahl und dem Ort zusammensetzt. In einer Rechnung kann es mehrere Positionen geben, die anderen Einheiten dürfen jeweils nur einmal vorkommen. Die Inhalte der Rechnungseinheiten lassen sich ebenfalls leicht definieren: Ein Stückpreis ist beispielsweise eine positive Zahl mit zwei Nachkommastellen und das Rechnungsdatum muss ein gültiges Datum sein. Durch diese Struktur lässt sich eine Rechnung um Positionen erweitern, der Rechnungsbetrag lässt sich einfach errechnen und auch eine Suche nach bestimmten Rechnungen wird möglich.

Je regelmäßiger eine Struktur ist, desto leichter lässt sie sich automatisiert verarbeiten. Andererseits bedeutet eine regelmäßige Struktur gleichzeitig auch weniger Flexibilität. Auf das obige Rechnungsbeispiel übertragen bedeutet das, dass z.B. alle Positionen gleich aufgebaut sein müssen – Abweichungen bei einer Position sind nicht möglich.

Daten in traditionellen – also z.B. relationalen oder objekt-orientierten – Datenbanken sind stark strukturiert. Die Relationen in relationalen Datenbanken bestehen immer aus gleichartigen Tupeln mit genau definierten Datentypen. Im Gegensatz dazu stehen unstrukturierte Texte, die sich nicht in einzelne Einheiten aufteilen lassen. Semi-strukturierte Daten (siehe Kapitel 2.2.1), die einen „Kompromiss“ zwischen starker Strukturierung und Unstrukturiertheit bilden, spielen bei den XML-Standards eine wichtige Rolle [KST02].

2.2 Semi-strukturiertes Datenmodell

2.2.1 Semi-strukturierte Daten

[KST02] und [Abi97] benennen u.a. die folgenden Merkmale von semi-strukturierten Daten:

- Die Struktur kann varieren, d.h. Elemente dürfen fehlen oder es kann zusätzliche Elemente geben.
- Die Struktur kann manchmal erst nach (a posteriori) Vorliegen der Daten angegeben werden, statt wie bei strukturierten Daten vorher (a priori), sie ist also nur implizit vorhanden.

- Die Struktur kann partiell sein, d.h. dass nicht alle Daten strukturiert sein müssen.
- Die Schemata können sehr umfangreich sein – im Gegensatz zu relationalen Datenbanken, in denen die Schemata wesentlich kleiner als die Daten sind.

Die vielen HTML-Seiten im Internet sind gute Beispiele für semi-strukturierte Daten. So haben z.B. Restaurant- und Hotel-Führer zwar immer eine Struktur, aber bei den einzelnen Beschreibungen können Angaben fehlen oder es kann über ein Hotel zusätzliche Informationen geben, die nicht in die Struktur passen.

2.2.2 Datenmodell OEM

Stellvertretend für semi-strukturierte Datenmodelle wird hier das Object Exchange Model (OEM) vorgestellt. Daten im OEM können entweder als kanten- oder knotenbewerteter Graph dargestellt werden; im Zusammenhang mit XML spielt die knotenbewertete Variante die größere Rolle. Objekte werden dabei als Knoten dargestellt, der Bezeichner ist Bestandteil solcher Knoten. Jedes Objekt kann einen eindeutigen Identifikator (object identifier, oid) vom Typ `oid` besitzen, der aus dem `&`-Zeichen und einem Wert vom Typ `string` besteht. Diese Identifikatoren lassen sich als Anker für Referenzen benutzen, um auf Objekte verweisen zu können, die mehrfach verwendet werden sollen. Es gibt sowohl atomare Objekte, die Werte der Typen `integer`, `real`, `string` etc. enthalten, als auch komplexe Objekte vom Typ `set`, die Unterobjekte bzw. Verweise auf solche aufnehmen können. Alle Objekte haben den gleichen Aufbau: `<bezeichner typ wert>`.

Kazakos, Schmidt und Tomczyk [KST02] nennen für OEM das folgende Beispiel:

```
<katalog set {
  <&a1 artikel set {
    <name string "Bohrmaschine">
    <stueckpreis real 179.95>
    <hersteller string "Bosch">
  }>
  <&a2 artikel set {
    <name string "Fassendenfarbe">
    <stueckpreis real 25.50>
  }>
  <&d1 dienstleistung set {
    <name string "Anschluss Elektrogerät">
    <stundensatz real 45>
  }>
}>

<&b200101 bestellung set {
  <bestellposition set {
    &a1
```

```
20      <anzahl int 1>
21      <gesamtpreis real 179.95>
22    }>
23    <bestellposition set {
24      &d1
25      <anzahl int 2>
26      <gesamtpreis reak 51>
27    }>
28  }>
```

2.3 XML-Datenmodelle

Ursprünglich wurde kein formales XML-Datenmodell definiert – eine formelle Definition gab es erst später, als immer mehr Anwendungen XML zur Speicherung von Daten einsetzten. Zur Definition wird eine W3C-Empfehlung aus dem Oktober 2001 verwendet, das XML Information Set (Infoset, Informationsmenge) [CT01]. Für XML gibt es nicht nur ein Datenmodell, die Informationsmenge hängt vielmehr davon ab, ob und womit das XML-Dokument validiert wurde. Die Informationsmenge abstrahiert von konkreten Systemen und beschreibt, welche Information in einem XML-Dokument abgelegt werden kann und wie diese strukturiert werden kann.

Das Information Set wird aus einer Menge von Informations-Einträgen (information items) gebildet. Die Einträge entsprechen den Knoten der XML-Dokumente. Es gibt elf verschiedene Typen von Informations-Einträgen, u.a. Dokument, Element, Attribut, Zeichen, Kommentar, Namensraum. Jeder dieser Einträge besitzt mehrere Eigenschaften, so gibt es z.B. beim Dokument-Informations-Eintrag die Eigenschaft `children`, die eine geordnete Liste (in der Dokument-Reihenfolge) der Unter-Elemente enthält.

Cowan und Tobin [CT01] nennen im Anhang Ihrer W3C-Empfehlung das folgende Beispiel für ein XML-Dokument:

```
1 <?xml version="1.0"?>
2
3 <msg:message doc:date="19990421"
4             xmlns:doc="http://doc.example.org/namespaces/doc"
5             xmlns:msg="http://message.example.org/">
6 Phone home!</msg:message>
```

Das Infoset für dieses Dokument enthält die folgenden Einträge:

- einen Dokument-Informations-Eintrag
- einen Element-Informations-Eintrag mit dem Namensraum-Namen „`http://message.example.org/`“, mit dem lokalen Namen „`message`“ und dem Präfix „`msg`“
- einen Attribut-Informations-Eintrag mit dem Namensraum-Namen „`http://doc.example.org/namespaces/doc`“, mit dem lokalen Namen „`date`“, mit dem Präfix „`doc`“ und dem normalisierten Wert „`19990421`“

- drei Namensraum-Informations-Einträge für die Namensräume:
  - `http://www.w3.org/XML/1998/namespace`
  - `http://doc.example.org/namespaces/doc`
  - `http://message.example.org/`
- zwei Attribut-Informations-Einträge für die Namensraum-Attribute
- elf Zeichen-Informations-Einträge für die Zeichen-Daten

3 Gegenüberstellung: daten- und textorientierte Dokumente

3.1 Datenorientierte Dokumente

Datenorientierte Dokumente benutzen XML für den Daten-Transport. Beispiele für solche Dokumente sind Rechnungen, Flugpläne, wissenschaftliche Daten und Börsenkurse.

Datenorientierte Dokumente wurden für maschinelle Verarbeitung entworfen und haben eine sehr genau gegliederte Struktur, die in der untersten Ebene nur ein Element oder ein Attribut enthält.

Daten für diese Art von Dokumenten können sowohl aus einer (z.B. relationalen) Datenbank stammen, als auch aus anderen Quellen wie z.B. aus einer wissenschaftlichen Software, die Messergebnisse liefert und eine Konvertierung dieser Daten in XML ermöglicht.

Als Beispiel für weitere Dokumentarten, die datenorientiert sein können, nennt Ronald Bourret [Bou02a] eine Detail-Ansicht eines Buches im Internet-Shop Amazon. Die Struktur des Textes auf diesen Seiten ist sehr gleichmäßig, viele Elemente wiederholen sich auf den einzelnen Seiten und die Elemente sind auf eine bestimmte Größe beschränkt. Solche Internet-Seiten könnten aus einem datenorienierten XML-Dokument, das Informationen über ein Buch enthält, und einem XSL-Stylesheet, das die restliche Texte und Element enthält, erstellt werden. Allgemein ausgedrückt könnte man sogar alle dynamischen HTML-Seiten, die Templates mit Datenbank-Daten füllen, durch datenorientierte XML-Dokumente und ein oder mehrere XML-Stylesheets ersetzen und somit als datenorientiert betrachten.

Ein Beispiel für datenorientierte Dokumente gibt Bourret [Bou02a]:

```
<FlightInfo>
  <Airline>ABC Airways</Airline> provides <Count>three</Count>
  non-stop flights daily from <Origin>Dallas</Origin> to
  <Destination>Fort Worth</Destination>. Departure times are
  <Departure>09:15</Departure>, <Departure>11:15</Departure>,
  and <Departure>13:15</Departure>. Arrival times are minutes
  later.
</FlightInfo>
```

Dieses Dokument, das Flüge beschreibt, lässt sich aus dem folgenden XML-Dokument und einem einfachen XSL-Stylesheet erstellen.

```
<Flights>
  <Airline>ABC Airways</Airline>
```

```
  <Origin>Dallas</Origin>
  <Destination>Fort Worth</Destination>
  <Flight>
    <Departure>09:15</Departure>
    <Arrival>09:16</Arrival>
  </Flight>
  <Flight>
    <Departure>11:15</Departure>
    <Arrival>11:16</Arrival>
  </Flight>
  <Flight>
    <Departure>13:15</Departure>
    <Arrival>13:16</Arrival>
  </Flight>
</Flights>
```

3.2 Textorientierte Dokumente

Im Gegensatz zu datenorientierten Dokumenten sind textorientierte Dokumente so strukturiert, dass Menschen sie leicht verstehen können. Beispiele für solche Dokumente sind Bücher, Werbung, E-Mails und auch von Hand entworfene Internet-Seiten im XHTML-Format. Die Struktur ist bei textorientieren Dokumenten weniger regelmäßig als bei datenorienierte Dokumenten oder sogar unregelmäßig. Die Reihenfolge der Elemente spielt eine wichtige Rolle.

Textorientierte Dokumente sind typischerweise entweder handgeschriebenes XML oder Dokumente in anderen Standard-Formaten wie RTF, PDF oder SGML, die dann in XML konvertiert wurden. Daten aus Datenbanken sind im Normalfall nicht textorientiert, sondern datenorientiert (siehe Kapitel 3.1).

Eine Produktbeschreibung als Beispiel für ein textorientiertes Dokument findet sich bei Ronald Bourret [Bou02a]:

```
<Product>
  <Intro>
    The <ProductName>Turkey Wrench</ProductName> from
    <Developer>Full Fabrication Labs, Inc.</Developer> is
    <Summary>like a monkey wrench, but not as big.</Summary>
  </Intro>
  <Description>
    <Para>The turkey wrench, which comes in <i>both right- and
    left-handed versions (skyhook optional)</i>, is made of the
    <b>finest stainless steel</b>. The Readi-grip rubberized
    handle quickly adapts to your hands, even in the greasiest
    situations. Adjustment is possible through a variety of custom
    dials.</Para>
```

```
14      <Para>You can:</Para>
15      <List>
16        <Item><Link URL="Order.html">Order your own turkey
17          wrench</Link></Item>
18        <Item><Link URL="Wrenches.htm">Read more about
19          wrenches</Link></Item>
20        <Item><Link URL="Catalog.zip">Download the
21          catalog</Link></Item>
22      </List>
23      <Para>The turkey wrench costs <b>just $19.99</b> and, if you
24      order now, comes with a <b>hand-crafted shrimp hammer</b> as a
25      bonus gift.</Para>
26    </Description>
27  </Product>
```

3.3 Daten, Dokumente und Datenbanken

In der Praxis ist die Entscheidung, ob es sich um ein datenorientiertes oder um ein textorientiertes Dokument handelt, nicht immer einfach. So können datenorientierte Dokumente wie z.B. Rechnungen auch unstrukturierte Produktbeschreibungen enthalten. Und auch textorientierte Dokumente wie z.B. Handbücher können strukturierte Elemente wie die Autorennamen und das Erscheinungsdatum enthalten. Dies gilt auch für juristische und medizinische Texte, die zwar als komplette Dokumente gespeichert werden, aber auch viele strukturierte Elemente enthalten.

Als Faustregel gilt: datenorientierte Dokumente werden in traditionellen – z.B. relationalen oder objekt-orientierten – Datenbanken gespeichert. Dies geschieht entweder durch spezielle Zusatzprogramme oder durch die Möglichkeiten der Datenbank selbst (in diesem Fall bezeichnet man eine solche Datenbank als XML-erweitert). Textorientierte Dokumente werden grundsätzlich entweder in einer nativen XML-Datenbank oder in einem Content-Management-System gespeichert.

Es gibt aber auch Ausnahmen: Insbesondere semi-strukturierte Daten lassen sich sehr einfach in nativen XML-Datenbanken speichern. Andererseits kann man auch Dokumente in traditionellen Datenbanken speichern, wenn man nur wenige XML-Leistungsmerkmale benötigt. Die Grenzen zwischen traditionellen Datenbanken und nativen XML-Datenbanken sind fließend, da beide Typen inzwischen auch einige Merkmale und Funktionen des jeweils anderen Typs bieten.

4 Speichern und Auslesen von datenorientierten Dokumenten

4.1 Abbildung von Dokument-Schemata auf Datenbank-Schemata

In diesem Kapitel wird erläutert, wie datenorientierte Dokumente in einer Datenbank gespeichert werden können. Zunächst wird die Speicherung in relationalen Datenbanken betrachtet, für die Tabellen-basierte und Objekt-relationale Abbildungen benötigt werden. Beide Typen arbeiten nicht mit den kompletten Dokumenten, sondern mit den Daten, die diese Dokumente enthalten. Diese Abbildungen eignen sich daher gut für datenorientierte und eher schlecht für textorientierte Dokumente.

4.1.1 Tabellen-basierte Abbildungen

Tabellen-basierte Abbildungen werden von vielen Programmen benutzt, die Daten zwischen XML-Dokumenten und relationalen Datenbanken austauschen. XML-Dokumente werden entweder als einzelne Tabelle oder als mehrere Tabellen modelliert. Die Struktur des Dokuments muss daher wie folgt aussehen (`<database>` und `<table>` entfallen, wenn es nur eine Tabelle gibt):

```
<database>
  <table>
    <row>
      <column1>...</column1>
      <column2>...</column2>
      ...
    </row>
    <row>
      ...
    </row>
    ...
  </table>
  <table>
    ...
  </table>
```

```
16   ...
17 </database>
```

Tabellen-basierte Abbildungen eignen sich sehr gut, um Daten zwischen zwei relationalen Datenbanken auszutauschen. Nachteil dieser Abbildungsart ist, dass nur XML-Dokumente verarbeitet werden können, die das oben genannte Format haben. Außerdem gehen die physikalische Struktur und Kommentare verloren.

4.1.2 Objekt-relationale Abbildungen

Objekt-relationale Abbildungen, die auch als Objekt-basierte Abbildungen bezeichnet werden, werden von allen XML-erweiterten relationalen Datenbanken benutzt. Ein XML-Dokument wird dabei als Objekt-Baum modelliert, dessen Objekte von den Daten des Dokuments abhängen. Diese Objekte werden auf die Datenbank abgebildet.

Objekt-relationale Abbildungen werden in zwei Schritten durchgeführt. Zuerst wird ein XML-Schema (z.B. eine DTD) auf ein Objekt-Schema abgebildet (siehe Kapitel 4.1.2.1), anschließend wird das Objekt-Schema auf das Datenbank-Schema abgebildet (siehe Kapitel 4.1.2.2). In den meisten Programmen werden die beiden Schritte kombiniert, so dass eine direkte Abbildung einer DTD auf ein Datenbank-Schema möglich ist.

4.1.2.1 Abbildung von DTDs auf Objekt-Schemata

Element-Typen sind Daten-Typen; wenn sie nur PCDATA enthalten, bezeichnet man sie als einfache Element-Typen. Es handelt sich dabei also um Typen, die nur einen Daten-Wert enthalten und die man daher mit skalaren Daten-Typen in objekt-orientierten Programmiersprachen vergleichen kann. Element-Typen mit komplexem Inhaltsmodell bezeichnet man als komplexe Element-Typen. Solche Typen sind vergleichbar mit Klassen in objekt-orientierten Programmiersprachen.

Bei der Objekt-relationalen Abbildung werden zunächst einfache Element-Typen auf skalare Daten-Typen abgebildet. Zum Beispiel könnte der Element-Typ `Titel` auf einen String und der Typ `Preis` auf einen Float-Wert abgebildet werden. Anschließend werden komplexe Element-Typen auf Klassen abgebildet, die Element-Typen im Inhaltsmodell der komplexen Typen werden auf Eigenschaften der Klasse abgebildet. Der Daten-Typ jeder Eigenschaft entspricht dem Daten-Typ, auf den das Element abgebildet wird. Am Beispiel von `Titel` und `Preis` bedeutet das, dass ein Verweis auf das `Titel`-Element auf eine String-Eigenschaft und dass ein Verweis auf das `Preis`-Element auf eine Float-Eigenschaft abgebildet wird. Verweise auf komplexe Element-Typen werden auf Verweise zu einem Objekt der Klasse, auf das der komplexe Element-Typ abgebildet wird, abgebildet.

Im letzten Schritt bildet man Attribute auf Eigenschaften ab, der Daten-Typ der Eigenschaft entspricht dem Daten-Typ des Attributs. Attribute entsprechen Verweisen auf Element-Typen in Inhaltsmodellen, der einzige Unterschied ist, dass Attribut-Typen lokal definiert werden und nicht wie Element-Typen global in einer DTD.

Im folgenden Beispiel von Ronald Bourret [Bou01] werden die einfachen Element-Typen `B`, `D` und `E` sowie das Attribut `F` auf Strings abgebildet. Die komplexen Element-Typen `A` und `C`

werden auf die Klassen `A` und `C` abgebildet. Das Inhaltsmodell und die Attribute von `A` und `C` werden auf die Eigenschaften der Klassen `A` und `C` abgebildet. Die Verweise auf `B`, `D` und `E` in den Inhaltsmodellen von `A` und `C` und das Attribut `F` werden auf String-Eigenschaften abgebildet. Der Verweis auf `C` im Inhaltsmodell von `A` wird auf eine Eigenschaft mit dem Typ-Verweis auf ein Objekt der Klasse `C` abgebildet.

```
            DTD                          Classes
=========================           =============

<!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;
                                    }
```

4.1.2.2 Abbildung von Objekt- auf Datenbank-Schemata

Im zweiten Teil der Objekt-relationalen Abbildung werden Klassen auf Tabellen, skalare Eigenschaften auf Spalten und Verweis-Eigenschaften auf Primär- bzw. Fremdschlüssel abgebildet. Ronald Bourret [Bou01] nennt hierfür das folgende Beispiel:

```
  Classes                      Tables
============              ==============
class A {                 Table A:
   String b;                 Column b
   C      c;      ==>        Column c_fk
   String f;                 Column f
}

class C {                 Table C:
   String d;      ==>        Column d
   String e;                 Column e
}                            Column c_pk
```

Die Tabellen aus dem Beispiel werden über den Primärschlüssel `C.c_pk` und den Fremdschlüssel `A.c_fk` miteinander verbunden. Da die Beziehung zwischen den Tabellen 1:1 ist, kann sich der Primärschlüssel in einer der beiden Tabellen befinden. Wäre die Beziehung 1:n, müsste der Primärschlüssel auf der „1“-Seite der Beziehung sein.

Eine Primärschlüssel-Spalte kann entweder – wie die Spalte `c_pk` aus dem Bourret-Beispiel – während der Abbildung erstellt werden oder man kann eine bereits existierende Spalte als Primärschlüssel benutzen. Wenn die Primärschlüssel-Spalte während der Abbildung erstellt wird, muss die Datenbank oder das benutzte Programm für eindeutige Werte in dieser Spalte sorgen – dies führt zwar zu einem besseren Datenbank-Design, hat aber im Zusammenhang mit XML den Nachteil, dass der erzeugte Schlüssel nur innerhalb der Datenbank eine Bedeutung hat. Sobald man die Daten aus der Datenbank mit solch einem Primärschlüssel in ein XML-Dokument exportiert, hat man entweder einen Primärschlüssel ohne Bedeutung oder gar keinen Primärschlüssel (falls man ihn wegen der fehlenden Bedeutung nicht exportiert) in diesem Dokument. Im letzteren Fall kann es unmöglich sein, die Herkunft der Daten richtig zu ermitteln. Dies ist insbesondere dann ein großes Problem, wenn man die Daten nach einer Änderung wieder in die Datenbank importieren möchte.

4.2 Anfragesprachen

4.2.1 Template-basierte Anfragesprachen

Anfragesprachen, die XML aus relationalen Datenbanken zurückliefern, sind meistens Template-basiert. In diesen Sprachen gibt es keine vordefinierten Abbildungen zwischen Dokument und Datenbank.

Für Anfragen werden SELECT-Anweisungen in einem Template platziert. Bourret [Bou02a] nennt das folgende Beispiel, in dem ein Element namens `<SelectStmt>` die SELECT-Anweisung enthält und in dem die Variablen `$Spaltenname` durch die Ergebnisse der Anfrage ersetzt werden:

```
<?xml version="1.0"?>
<FlightInfo>
  <Introduction>The following flights have available
    seats:</Introduction>
  <SelectStmt>SELECT Airline, FltNumber, Depart, Arrive
    FROM Flights</SelectStmt>
    <Flight>
      <Airline>$Airline</Airline>
      <FltNumber>$FltNumber</FltNumber>
      <Depart>$Depart</Depart>
      <Arrive>$Arrive</Arrive>
    </Flight>
  <Conclusion>We hope one of these meets your needs</Conclusion>
</FlightInfo>
```

Das Ergebnis sieht dann so aus:

```
<?xml version="1.0"?>
<FlightInfo>
  <Introduction>The following flights have available
    seats:</Introduction>
  <Flights>
    <Flight>
      <Airline>ACME</Airline>
      <FltNumber>123</FltNumber>
      <Depart>Dec 12, 1998 13:43</Depart>
      <Arrive>Dec 13, 1998 01:21</Arrive>
    </Flight>
    ...
  </Flights>
  <Conclusion>We hope one of these meets your needs.</Conclusion>
</FlightInfo>
```

Template-basierte Anfragesprachen eignen sich hauptsächlich nur für den Daten-Transfer von einer relationalen Datenbank in ein XML-Dokument. Für den umgekehrten Weg gibt es zwar inzwischen auch schon Lösungen, die volle Funktionalität bieten aber nur Schemata-Abbildungen wie sie in Kapitel 4.1 beschrieben wurden.

4.2.2 SQL-basierte Anfragesprachen

SQL-basierte Anfragesprachen nutzen spezielle SELECT-Anweisungen. Die Ergebnisse der Anfragen werden anschließend in XML umgewandelt. Bei der einfachsten Variante benutzt man verschachtelte SQL-Anweisungen, die entsprechend der Abbildungsvorschriften direkt in verschachteltes XML umgewandelt werden.

4.2.3 XML-Anfragesprachen

Während sich Template-basierte (vgl. Kapitel 4.2.1) und SQL-basierte (4.2.2) Anfragesprachen nur für relationale Datenbanken eignen, können XML-Anfragesprachen für alle XML-Dokumente benutzt werden.

Christian Gross [Gro03], der für eine kurze Zeit Mitglied der XML Query Working Group war, gibt ein kurzes Beispiel für XML Query (oft auch als XQuery bezeichnet). Als Basis sei das folgende XML-Dokument gegeben:

```
<content>
  <tag>some value</tag>
  <tag>another value</tag>
</content>
```

Ein Beispiel für eine XQuery-Abfrage:

```
<results>
{
  for $b in document("http://example")/content/tag
  return
    <result>
      {b}
    </result>
}
</results>
```

Das Ergebnis sieht dann so aus:

```
<results>
  <result>
    <tag>some value</tag>
  </result>
  <result>
    <tag>another value</tag>
  </result>
</results>
```

Datenbanken, die Anfragesprachen wie XQuery unterstützen, müssen eine Schnittstelle bieten, die ein „virtuelles XML-Dokument“ zur Verfügung stellt. Auf diesem virtuellen Dokument wird der Abfrage-Ausdruck ausgewertet. Dieses Dokument kann entweder durch Tabellenbasierte Abbildung (also einzelne Tabellen, die ggf. von XQuery zusammengesetzt werden) oder durch Objekt-relationale Abbildung (also eine Zusammenfassung der einzelnen Tabellen zu einem Gesamtdokument) entstehen.

4.3 Speicherung von datenorientierten Dokumenten in nativen XML-Datenbanken

Es sprechen einige Gründe dafür, auch datenorientierte XML-Dokumente in einer nativen XML-Datenbank zu speichern. Gerade bei semi-strukturierten Daten, bei denen die Abbildung auf eine relationale Datenbank – wie in Kapitel 4.1 beschrieben – sehr viele NULL-Spalten und eine große Anzahl an Tabellen erzeugen kann, ist es effizienter, die Daten in einer nativen XML-Datenbank abzulegen.

Ein zweiter Grund ist die hohe Zugriffsgeschwindigkeit nativer XML-Datenbanken. Dieser Vorteil liegt daran, dass solche Datenbanken komplette Dokumente zusammen physikalisch speichern oder dass sie physikalische (und nicht logische wie bei relationalen Datenbanken)

Zeiger zwischen den Teilen der Dokumente benutzen. Joins lassen sich so entweder vermeiden bzw. es werden nur physikalische Joins benötigt, was gegenüber logischen Joins, die von relationalen Datenbanken benutzt werden, ein Geschwindigkeitsvorteil ist.

Am Beispiel einer Rechnung lässt sich dies leicht verdeutlichen: In einer relationalen Datenbank benötigt man vier Tabellen: eine für die Rechnungen, eine für die Posten, eine für die Teile und eine für die Kunden. Um eine Rechnung auszulesen, müssten mehrere Joins über diese Tabellen gemacht werden. Eine native XML-Datenbank könnte aber die gesamte Rechnung als ein Dokument gespeichert haben, der Zugriff darauf wäre wesentlich schneller.

Dieser Geschwindigkeitsvorteil ist gegeben, wenn man die Daten in der gleichen Struktur ausliest, in der man sie gespeichert hat. Möchte man beispielsweise eine Liste der Kunden zusammen mit den zugehörigen Rechnungen auslesen, so könnte eine relationale Datenbank das Ergebnis wahrscheinlich schneller liefern. Aufgrund dieses Problems ist das Speichern von Daten in nativen XML-Daten nur dann empfehlenswert, wenn man meistens mit demselben (einfachen) Anfragetyp arbeitet.

Ein großes Problem bei nativen XML-Datenbanken ist, dass die meisten nativen XML-Datenbanken nur XML-Daten liefern können. Da aber die meisten Programme die Daten in einem anderen Format benötigen, müssen zunächst die XML-Daten in das gewünschte Format konvertiert werden. Dies ist ein großer Nachteil gegenüber relationalen Datenbanken, da diese die Daten über Schnittstellen direkt im passenden Format ausgeben können.

5 Speichern und Auslesen von textorientierten Dokumenten

5.1 Textorientierte Dokumente im Dateisystem speichern

Die einfachste Möglichkeit, Dokumente zu speichern, ist, sie im Dateisystem abzulegen. Dieses Verfahren eignet sich z.B. für kleine Dokumentsammlungen. Auch wenn sich eine Volltextsuche (z.B. mit `grep`) für XML-Dokumente wegen der fehlenden Unterscheidung zwischen Markup und Text eigentlich nicht eignet, so kann man dieses Problem bei kleinen Datenmengen vernachlässigen. Eine einfache Variante einer Transaktions-Kontrolle lässt sich realisieren, wenn man die Dokumente in Versions-Kontroll-Systemen wie CVS oder RCS speichert.

5.2 Textorientierte Dokumente in relationalen Datenbanken speichern

Textorientierte Dokumente werden in relationalen Datenbanken gespeichert, indem das gesamte Dokument als BLOB oder CLOB (Binary / Character Large Object) in der Datenbank gespeichert wird. Eine relationale Datenbank bietet Funktionalitäten wie beispielsweise Sicherheit, Transaktions-Kontrolle oder Mehr-Benutzer-Zugriff. Auch eine Volltextsuche ist möglich – bei einigen Datenbanken sogar unter Berücksichtigung der XML-Syntax.

Effizientere Abfragen von Daten lassen sich durch ein Anlegen von Index-Tabellen, die als Look-Aside-Tabellen bezeichnet werden, erreichen. Dazu muss die Dokument-Tabelle einen eindeutigen Primärschlüssel und die Look-Aside-Tabelle muss Spalten für den indizierten Wert und den Verweis auf den Primärschlüssel-Wert der Dokument-Tabelle besitzen. Sofern die Look-Aside-Tabelle immer konsistent zur Dokument-Tabelle gehalten wird, sind schnelle Zugriffe auf bestimmte Elemente durch Nutzung der Index-Tabelle möglich.

5.3 Native XML-Datenbanken

Für viele Anwendungszwecke reichen die Speichermöglichkeiten, die in den Kapiteln 5.1 und 5.2 beschrieben wurden, nicht. Um XML-Dokumente zu speichern, eignen sich native XML-Datenbanken, die ähnliche Funktionalitäten wie andere Datenbanken bieten, gut. Zu diesen Funktionalitäten gehören Transaktionen, Sicherheit, Mehr-Benutzer-Zugriff, Anfragesprachen oder auch APIs zu verschiedenen Programmiersprachen (siehe auch Kapitel 5.3.2).

Im Gegensatz zu XML-erweiterten Datenbanken erhalten native XML-Datenbanken die Dokument-Reihenfolge, Kommentare, CDATA-Bereiche und Entities – sie eignen sich daher gut für textorientierte Dokumente. Durch die Unterstützung von XML-Anfragesprachen sind Anfragen wie „Gib mir alle Dokumente, in denen der dritte Absatz in einer Rubrik ein fett markiertes Wort enthält“ möglich, deren Formulierung mit SQL oder anderen Anfragesprachen wesentlich komplizierter wäre.

Bekannt wurde der Begriff „native XML-Datenbank“ durch die Werbekampagne der Software AG für die von ihr entwickelte native XML-Datenbank Tamino. Bedingt dadurch, dass es sich bei dem Begriff nur um einen Marketing-Begriff handelt, hat es nie eine formale Definition des Begriffs gegeben.

Mitglieder der XML:DB-Mailingliste[1] haben aber folgende Definition entworfen: „Eine native XML-Datenbank ist eine Datenbank, ...

- ... die ein (logisches) Modell für ein XML-Dokument – nicht für die Daten des Dokuments – definiert und die dem Modell entsprechend Dokumente speichert und zurückliefert. Das Modell muss mindestens Elemente, Attribute, PCDATA und eine Dokument-Reihenfolge enthalten. Beispiele für solche Modelle sind das XPath-Datenmodell, das XML-Infoset und die Modelle, die durch das DOM und die Ereignisse in SAX 1.0 impliziert werden.

- ... die als kleinste Daten-Einheit ein XML-Dokument hat. In relationalen Datenbanken ist die kleinste Daten-Einheit eine Zeile in einer Tabelle.

- ... der kein spezifisches physikalisches Speicher-Modell zugrunde liegen muss. Sie kann z.B. auf einer relationalen, hierarchischen oder objekt-orientierten Datenbank aufbauen oder ein proprietäres Speicherformat wie indizierte und komprimierte Dateien nutzen.“ (übersetzt aus [Bou02a])

5.3.1 Architekturen von nativen XML-Datenbanken

5.3.1.1 Text-basierte native XML-Datenbanken

Text-basierte native XML-Datenbanken speichern die XML-Daten als Text. Dieser Text kann z.B. im Dateisystem, in einer relationalen Datenbank oder einem anderen Format gespeichert werden. Indizes ermöglichen solchen nativen XML-Datenbanken schnelle Zugriffe auf die Daten, da bei Anfragen nach kompletten Dokumenten oder Dokumentteilen nur ein Index-Zugriff notwendig ist und das (der) gewünschte Dokument(teil) direkt ausgelesen werden kann. Bei komplexeren Anfragen haben Text-basierte native XML-Datenbanken aber einen Nachteil bezüglich der Geschwigkeit gegenüber relationalen Datenbanken (vgl. Kapitel 4.3).

5.3.1.2 Modell-basierte native XML-Datenbanken

Modell-basierte native XML-Datenbanken speichern im Gegensatz zu den Text-basierten nativen XML-Datenbanken nicht komplette Dokumente als Text, sondern analysieren das Dokument

[1] http://www.xmldb.org/

und erzeugen daraus ein Objektmodell. Dieses Modell wird von einigen Modell-basierten nativen XML-Datenbanken in relationalen oder objekt-orientierten Datenbanken gespeichert – um z.B. das Document Object Model (DOM) in einer relationalen Datenbank zu speichern, könnten die Tabellen `Elements`, `Attributes`, `PCDATA`, `Entities` und `EntityReferences` verwendet werden. Andere Modell-basierte Datenbanken nutzen proprietäre Formate, die auf das jeweilige Modell angepasst sind.

Die Zugriffsgeschwindigkeit von Modell-basierten nativen XML-Datenbanken, die auf relationalen oder objekt-orientierten Datenbanken aufbauen, hängt vor allem von diesen Datenbanken ab. Je nach Design der XML-Datenbank kann es aber notwendig sein, mehrere Anfragen zu machen, um die gewünschten Daten auszulesen. Wenn die Datenbank z.B. eine Objekt-relationale Abbildung des DOM nutzt, so sind mehrere Anfragen notwendig, um die Kind-Elemente aller Knoten zu bekommen.

Modell-basierte native XML-Datenbanken, die ein proprietäres Format nutzen, haben eine ähnliche Zugriffsgeschwindigkeit wie Text-basierte native XML-Datenbanken, wenn die Daten in der gleichen Struktur abfragt werden, in der sie gespeichert wurden (vgl. Kapitel 4.3).

5.3.2 Funktionalitäten von nativen XML-Datenbanken

Viele native XML-Datenbanken unterstützen Kollektionen, vergleichbar mit einer Tabelle in einer relationalen Datenbank oder einer Datei im Dateisystem. Wenn man z.B. Rechnungen in einer nativen XML-Datenbank gespeichert hat, könnte man bestimmte Rechnungen zu einer Kollektion zusammenfassen und dann Anfragen formulieren, die nur Dokumente aus dieser Kollektion liefern.

Die zur Zeit am häufigsten unterstützten Anfragesprachen in nativen XML-Datenbanken sind XPath und XQL, es gibt aber auch viele proprietäre Anfragesprachen. Die Datenbank Tamino arbeitet mit X-Query, einer Sprache, die auf XQL basiert und um proprietäre Funktionen erweitert wurde. Samar [Sam03] und Bourret [Bou02a] rechnen aber damit, dass sowohl Tamino als auch die meisten anderen nativen XML-Datenbanken in der Zukunft XQuery vom W3C unterstützen werden.

Zugriffssperren, die gleichzeitigen Schreibzugriff verschiedener Benutzer auf den selben Datensatz verhindern, (Locking) unterstützen die meisten nativen XML-Datenbanken nur auf Dokument-Ebene. Ob dies ein Nachteil ist, hängt vom Datenbank-Design ab. Wird z.B. ein Buch kapitelweise in XML-Dokumenten gespeichert, so wird dies kein Problem sein, da nur selten Autoren das gleiche Kapitel bearbeiten werden.

Application Programming Interfaces (APIs) werden von fast allen nativen XML-Datenbanken angeboten. Meistens handelt es sich dabei um ODBC-ähnliche Schnittstellen, die Funktionen für die Verbindung zur Datenbank, zum Ausführen von Anfragen und für das Zurückliefern von Ergebnissen bieten. Ergebnisse werden als XML-String, als DOM-Baum, als SAX-Parser oder als XMLReader zurückgegeben.

Eine wichtige Funktion nativer XML-Datenbanken ist, dass man in ihnen XML-Dokumente speichern kann und die gleichen Dokumente wieder zurückbekommen kann. Text-basierte native XML-Datenbanken können exakt die gleichen Dokumente zurückgeben, Modell-basierte native XML-Datenbanken geben die Dokumente entsprechend ihrem Dokument-Modell zurück.

6 Fazit

In den letzten Monaten und Jahren sind immer mehr native XML-Datenbanken entstanden und relationale Datenbanksysteme um XML-Funktionalitäten erweitert worden. So finden sich beispielsweise in Ronald Bourrets Produktliste [Bou02b] unter dem Punkt „XML-Enabled Databases“ bekannte relationale Datenbanken wie Microsoft Access, IBM DB2 oder Oracle.

Ein weiteres Beispiel für die Beliebtheit nativer XML-Datenbanken: Seit Ende Dezember 2002 ist in PEAR, dem PHP Extension und Application Repository, eine PHP-API für Tamino verfügbar. Eine solche API ist ebenfalls für die auf Java basierende native Open-Source-Datenbank Xindice verfügbar.

Während Robert Tolksdorf 1999 noch meinte, dass noch nicht abzusehen sei, welche Anfragesprache künftig dominieren werde [Tol99], geht Christian Gross, der für eine kurze Zeit Mitglied der XQuery Working Group war, 2003 davon aus, dass in Zukunft XQuery eine wichtige Rolle spielen wird – allerdings erst Ende 2004 / Anfang 2005 [Gro03]. Bis dahin seien wir weiter auf relationale Datenbanken angewiesen.

Laut Jost Enderle [End01] musste man bisher zur Verwaltung von semi-strukturierten Daten wie XML spezielle Systeme verwenden. XML-erweiterte Datenbanken wie z.B. relationale Datenbanken böten durch schon eingebaute und noch einzubauende XML-Funktionalitäten aber mittlerweile einen guten Ersatz für solche speziellen Systeme. Enderle empfiehlt nur noch bei besonderen Anforderungen wie Versions-Kontrolle die Nutzung von Spezialsystemen wie Content-Management-Systemen oder auch nativer XML-Datenbanken.

Literaturverzeichnis

[Abi97] ABITEBOUL, Serge: Querying Semi-Structured Data. In: ICDT, 1997. – http://citeseer.nj.nec.com/abiteboul97querying.html, S. 1–18

[Bou01] BOURRET, Ronald: Mapping DTDs to Databases. 2001. – http://www.xml.com/pub/a/2001/05/09/dtdtodbs.html, Kap. 1 - 3.1

[Bou02a] BOURRET, Ronald: XML and Databases. 2002. – http://www.rpbourret.com/xml/XMLAndDatabases.htm

[Bou02b] BOURRET, Ronald: XML Database Products. 2002. – http://www.rpbourret.com/xml/XMLDatabaseProds.htm

[CT01] COWAN, John ; TOBIN, Richard: XML Information Set. 2001. – http://www.w3.org/TR/2001/REC-xml-infoset-20011024

[End01] ENDERLE, Jost: XML in relationalen Datenbanken. In: Informatik-Spektrum 24. Dezember (2001), S. 357–368

[Gro03] GROSS, Christian: Gesucht, gefunden. Einführung in den Abfragestandard XQuery. In: PHP-Magazin 1 (2003), S. 83–87

[KST02] KAZAKOS, Wassilis ; SCHMIDT, Andreas ; TOMCZYK, Peter: Datenbanken und XML. Konzepte, Anwendungen, Systeme. Springer, 2002. – Kap. 3

[Sam03] SAMAR, Richard: Das X-Calibur der Datenbanken. Mit PHP auf die XML-Datenbank Tamino zugreifen. In: PHP-Magazin 1 (2003), S. 40–48

[Tol99] TOLKSDORF, Robert: XML und darauf basierende Standards: Die neuen Auszeichnungssprachen des Web. In: Informatik-Spektrum 22 (1999), S. 407–421

[TXI02] „THE XML:DB INITIATIVE“: Frequently Asked Questions About XML:DB. 2002. – http://www.xmldb.org/faqs.html