

# FACHHOCHSCHULE WEDEL

## Seminararbeit

In der Fachrichtung

Wirtschaftsinformatik

Thema:

XML und Datenbanken

Eingereicht von: Simon Temp (Mat.-Nr. 4684)  
Guipavasring 17  
22885 Barsbüttel  
Telefon (040) 67081092 Mobil: 0176-23561086  
E-Mail: simon.temp@temp.de

Erarbeitet im: 10. Semester

Abgegeben am: 07.12.2005

Referent: Prof. Dr. Sebastian Iwanowski  
Fachhochschule Wedel  
Feldstraße 143  
22880 Wedel  
Telefon (04103) 8048 63  
E-Mail: iw@fh-wedel.de

## INHALTSVERZEICHNIS

1	PROBLEMSTELLUNG - VORWORT .....	- 4 -
2	VERGLEICH VON XML UND RELATIONALEN DATENBANKEN .....	- 5 -
3	INTEGRATIONSARCHITEKTUR FÜR XML UND RELATIONALER DATENBANKEN .....	- 7 -
4	INTEGRATIONSSTRATEGIE VON XML MIT RELATIONALEN DATENBANKEN .....	- 14 -
5	„MAPPING“ VON XML UND RELATIONALEN DATEN.....	- 16 -
6	HERSTELLERUNTERSTÜTZUNG VON XML IN DATENBANKEN .	- 24 -
7	NATIVE XML-DATENBANKEN.....	- 26 -
8	SCHLUSSWORT .....	- 27 -
9	LITERATURVERZEICHNIS .....	- 28 -

## ABBILDUNGSVERZEICHNIS

Abbildung 1	XML-Dokumente gespeichert in RDBMS .....	- 8 -
Abbildung 2	XML-Dokumente und Schemata gespeichert in RDBMS .....	- 8 -
Abbildung 3	XML-Dokumente fragmentiert gespeichert in RDBMS .....	- 9 -
Abbildung 4	XML-Dokumente und Schemata fragmentiert gespeichert in RDBMS .....	- 10 -
Abbildung 5	DB Daten werden als XML-Dokumente geliefert.....	- 11 -
Abbildung 6	Relationale Daten werden gemappt .....	- 12 -
Abbildung 7	XML-Daten sind im Speicher zwischengespeichert.....	- 13 -
Abbildung 8	„Bear Sigthings“ Datenbank .....	- 16 -
Abbildung 9	Elementzentrierte Darstellung .....	- 17 -
Abbildung 10	Attributzentrierte Darstellung .....	- 17 -
Abbildung 11	XML-Dokument mit ID and IDREF .....	- 20 -
Abbildung 12	XML-Dokument mit ID and IDREF .....	- 20 -
Abbildung 13	Vordefinierte Typen in XSD .....	- 22 -
Abbildung 14	XML-Tools von Datenbankherstellern.....	- 24 -
Abbildung 15	Unterschiede in der XML-Unterstützung der Datenbankherstellern.....	- 25 -

# 1 Problemstellung - Vorwort

XML (Extensible Markup Language) hat in den letzten Jahren immer mehr an Bedeutung gewonnen und ist in der heutigen Zeit als universelles Austauschformat nicht mehr zu vernachlässigen. XML ist zu einem Standardformat für die Repräsentation von strukturierten, als auch unstrukturierten Daten, geworden.

Daher stellt sich die Frage, ob nicht auch relationale Datenbanken verwendet werden können, um XML zu speichern.

Ziel dieser Seminararbeit ist es nicht, dem Leser zur Implementierung eines XML-Speichersystems zu befähigen, sondern es werden Integrationsbeispiele gezeigt und Modelle vorgestellt, um XML-Daten auf Datenbanken abzubilden und um bereits bestehende Datenbankinhalte mit XML zu repräsentieren.

Die Seminararbeit hält sich zum größten Teil an Thomas Erl, der in seinem Buch „Service-Oriented Architecture – A Field Guide to Integrating XML and Web Services“ im Kapitel sieben sich mit diesem Thema beschäftigt. Sollten andere Behauptungen aufgestellt werden, sind entsprechende Quellenangaben vorhanden.

## 2 Vergleich von XML und relationalen Datenbanken

Die XML- und relationale Datenbank-Technologien unterscheiden sich wesentlich voneinander. Um XML besser in eine relationale Datenbank zu integrieren, werden diese unterschiedlichen Merkmale zuvor kurz erläutert.

Während Datenbanken, die Daten in einer kontrollierten Speicherumgebung sichern, erfolgt die Speicherung bei der XML-Technologie ausschließlich als reine Textdatei. Dadurch muss die Sicherheit beim XML auf Datei und Verzeichnisebene durch die eigene Anwendung implementiert werden.

Bedeutende Unterschiede liegen in der Repräsentation der Daten. Das Datenmodell einer Datenbank beruht auf tabellarischen Instanzen mit Zeilen und Spalten. Ein XML-Dokument ist dagegen hierarchisch aufgebaut und beinhaltet eine Dokumentenstruktur mit Elementen und Attributknoten. Die umfangreiche Typendefinition einer Datenbank kann XML nur mit Hilfe von XSD (XML-Schema-Definition) liefern.

Die Beziehungen zwischen den Elementen können bei XML durch deren Struktur selber vorliegen, oder durch ein Schema definiert werden. Die Datenbank stellt die Beziehungen durch die DDL (Date Definition Language) da, die innerhalb von Tabellen oder zwischen den Tabellen bestehen können.

Eine der grundlegenden Eigenschaft einer Datenbank ist die Plausibilität und Integrität der Daten. Die Integrität der Daten wird in der Datenbank durch Beziehungsbedingungen, sowie die Fähigkeit von Änderungen, die in Beziehung stehen, über kaskadierende Löschungen oder Updates sichergestellt. Die Plausibilität der Daten können durch gespeicherte Prozeduren und „Trigger“ bei Datenbanken unterstützt werden. XML kann diese Plausibilität z.B. nur mit einem XSD Schema realisieren. Bei der Integritätsprüfung müssen derartige Schemata ebenfalls eingesetzt werden und können meist nur simuliert werden. Es entspricht nicht dem Umfang einer Datenbank.

Die Suche von Daten in der Datenbank wird durch die Indexierung unterstützt, die anpassbar auf das Spaltenniveau ist und auch für bestimmte

Sucheigenschaften optimiert werden kann. Als Anfragesprache hat sich in der relationalen Datenbankwelt der industrielle Standard „SQL“ durchgesetzt, der von Datenbankherstellern teilweise proprietär erweitert wird. Eine Indexierung wird von XML selber nicht angeboten und muss, wenn benötigt, von der eigenen Anwendung erzeugt und gewartet werden. Bei der Abfragesprache gibt es eine Menge an Auswahlmöglichkeiten. XQuery für XML ist mit SQL zum Vergleich.

### 3 Integrationsarchitektur für XML und relationaler Datenbanken

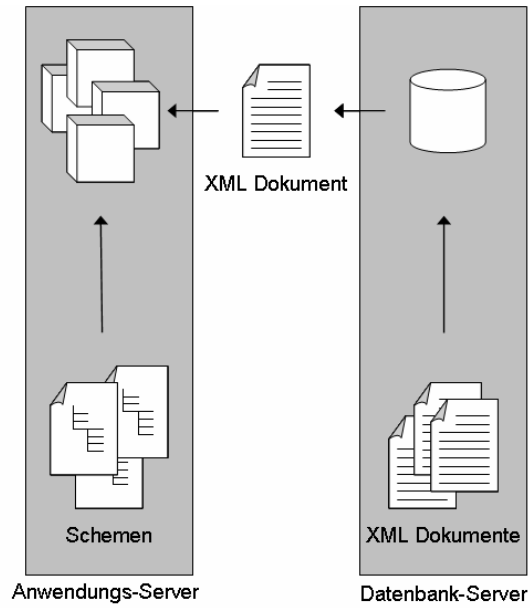
Für die Integrationsarchitektur von XML und Datenbanken gibt es keinen Standard, an den sich gehalten werden kann. Bevor nach einer Möglichkeit gesucht wird, müssen einige Punkte nach Thomas Erl beantwortet werden. Dazu gehört das klare Verständnis, warum und wie XML in die Anwendung eingebunden ist oder werden soll, falls noch keine Anwendung vorhanden ist. Weiterhin muss dargestellt werden, wie und wo die XML-Daten manipuliert werden und wie diese Daten in der Datenbank vorhanden sind. Erst nachdem diese Fragen beantwortet sind, ist die Wahl einer Integrationsstrategie möglich und sinnvoll.

Thomas Erl stellt in seinem Buch Möglichkeiten für die Integration vor, die keinen Anspruch haben, jegliche Eventualitäten abzudecken und gegebenenfalls angepasst werden müssen. Sie unterscheiden sich in Ihrer Art der Speicherung der Daten und dem Zeitpunkt der Erstellung der XML-Dokumente, sowie dem Speicherort der entsprechenden XML-Schemata.

Am wenigsten wird das Datenmodell einer bereits existierenden, relationalen Datenbank beeinflusst, wenn die XML-Dokumente in separaten Tabellen abgespeichert werden, die keine Beziehungen zwischen den XML-Daten und den bestehenden Daten des Modells aufweisen. Dafür reichen einspaltige Tabellen, die in einem allgemeinen Datentyp mit langer Zeichenkette, z.B. CLOB (DB2) oder VARCHAR, das gesamte XML-Dokument abspeichern (siehe Abbildung 1). Dies bezeichnet man auch als opake Speicherung<sup>1</sup>. Aus diesem Grund entfällt auch das „Mapping“ zwischen dem XML-Dokument und der Datenbank, was später im Kapitel 5 erläutert wird.

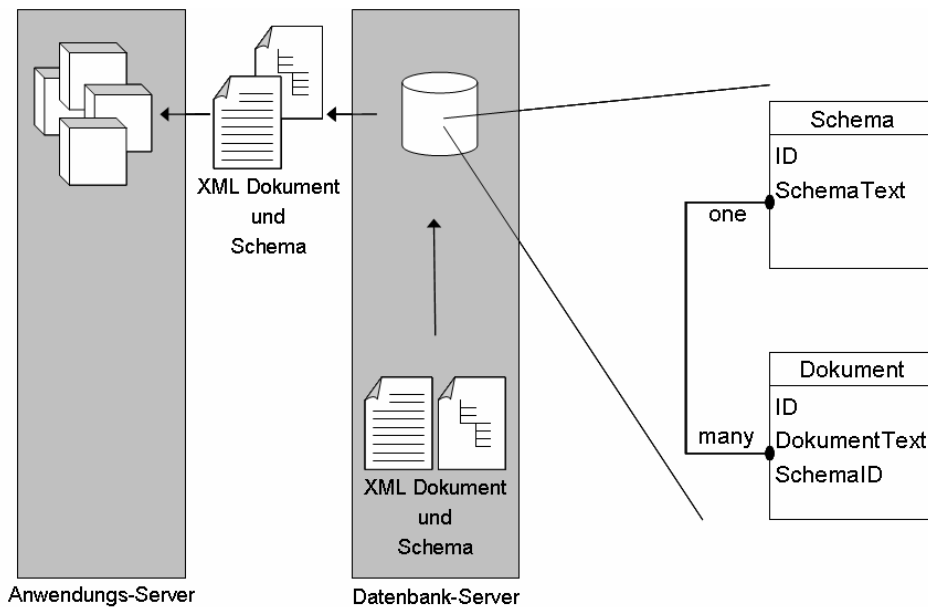
---

<sup>1</sup> Schöning, H.: „XML und Datenbanken“; München: Carl Hanser Verlag, 2003, S. 187



**Abbildung 1** XML-Dokumente gespeichert in RDBMS

Es besteht ebenfalls die Möglichkeit, die Daten nicht in einem einzelnen XML-Dokument abzuspeichern, sondern in kleine Fragmentteile, die dynamisch zusammengestellt werden können (siehe Abbildung 3).



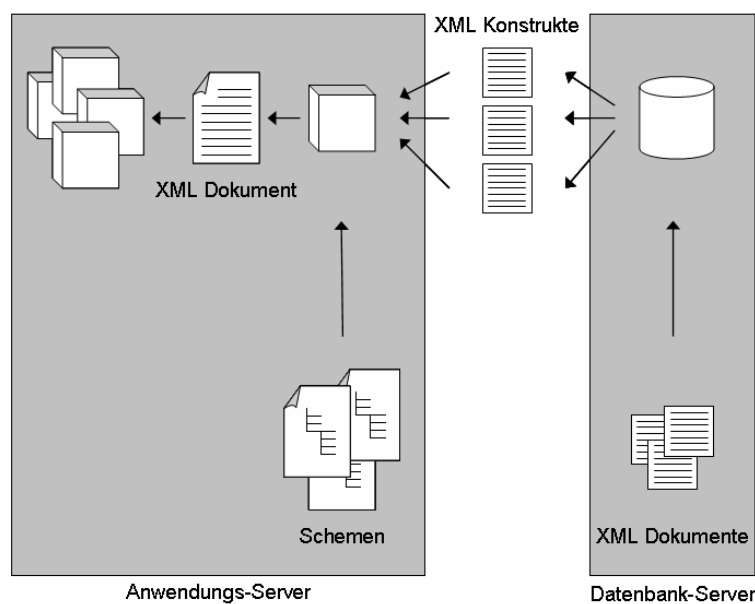
**Abbildung 2** XML-Dokumente und Schemata gespeichert in RDBMS

Die Daten sind durch diese Techniken redundant in der Datenbank abgespeichert, und es muss für eine Synchronisation zwischen den Datenbankdaten und dem XML-Dokument gesorgt werden.

Änderungsoperationen betreffen das gesamte Dokument, das dafür ebenfalls komplett ausgetauscht werden muss<sup>2</sup>.

Probleme ergeben sich bei einer Anfrage, die sich auf die XML-Daten beziehen, da hier nur eine Volltextsuche auf dem XML-Dokument oder Fragmentteil möglich ist, die langsam und meistens nicht zwischen XML-Daten und Markierungen unterscheiden kann. Ebenfalls lassen sich werteorientierte und strukturierte Anfragen nicht beantworten<sup>3</sup>. Für die Sicherheit, bezüglich des Zugriffsrechtes auf die Daten, ergeben sich auch Einschränkungen bei diesem Verfahren. Bei der opaken Speicherung gibt es nur die Möglichkeit, die Zugriffsrechte auf die Daten der Dokumentenebene vom RDBMS zu überwachen, nicht aber auf einer kleineren Ebene, wie knotenspezifische Zugriffsrechte<sup>4</sup>. Bei diesem Ansatz der opaken Speicherung weist das aktuelle Datenmodell keine Datengleichheit mehr auf.

Das zugehörige Schema kann entweder auf der Datenbankseite zum entsprechenden XML-Dokument abgespeichert (siehe Abbildung 4 und 2), oder auf der Anwenderseite zur Laufzeit eingebunden werden.

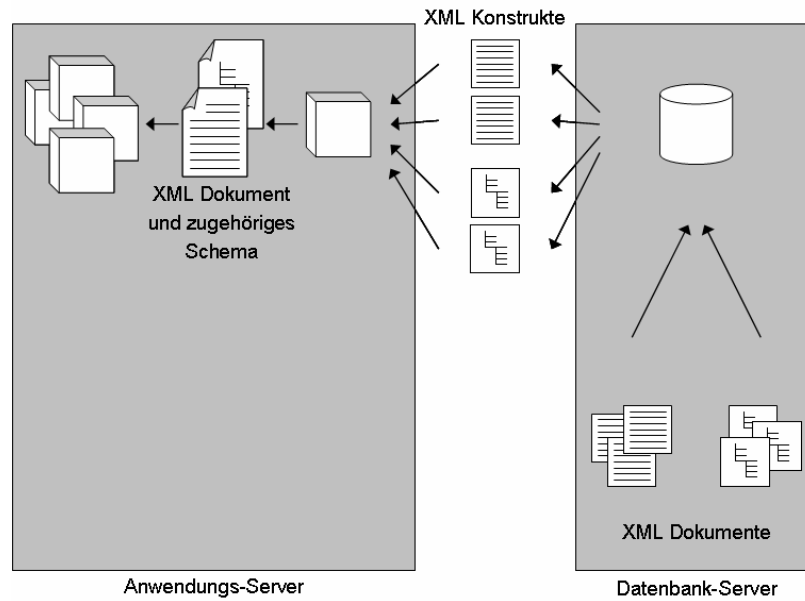


**Abbildung 3** XML-Dokumente fragmentiert gespeichert in RDBMS

<sup>2</sup> Schöning, (FN 1), S. 188

<sup>3</sup> Schöning, (FN 1), S. 188

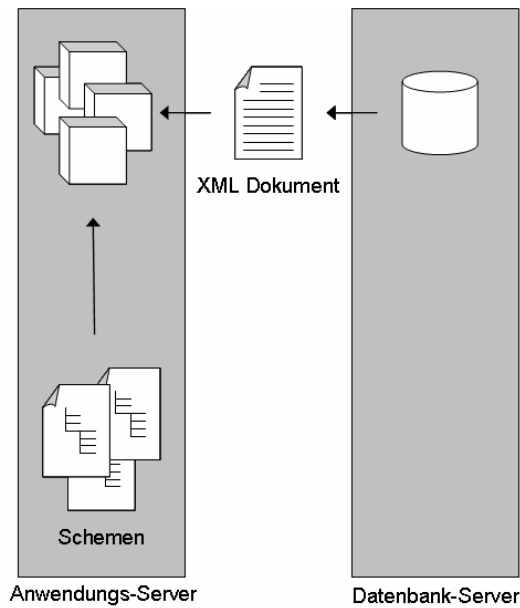
<sup>4</sup> Schöning, (FN 1), S. 191



**Abbildung 4** XML-Dokumente und Schemata fragmentiert gespeichert in RDBMS

Einige RDBMS Hersteller bieten proprietäre XML-Erweiterungen an, mit denen die bestehenden Datenbankdaten in ein XML-Format gebracht werden können. Dieses bezeichnet man auch als XML „publishing“ oder „wrapping“, da die Datenbankdaten in XML-Markierungen verpackt werden<sup>5</sup>. Dadurch ist es möglich, aus Anfragesichten dynamisch ein XML-Dokument zu erstellen, das auf der Anwenderseite durch ein entsprechendes Schema noch validiert werden kann (siehe Abbildung 5).

<sup>5</sup> Schöning, (FN 1), S. 167



**Abbildung 5** DB Daten werden als XML-Dokumente geliefert

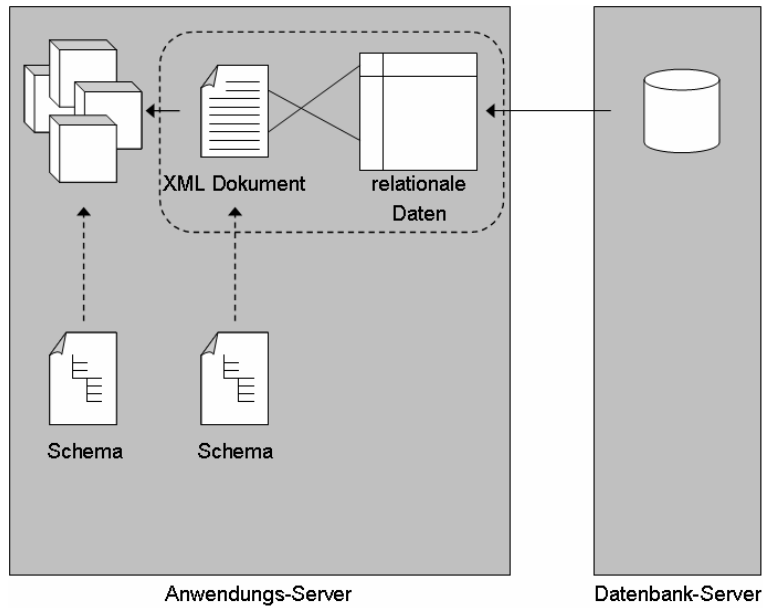
Durch die Unterstützung, die die Datenbankhersteller zur Verfügung stellen, ist dieser Ansatz leicht zu implementieren und beeinträchtigt das vorhandene Datenbankschema ebenfalls nicht.

Komplexer ist der Ansatz der inhaltsorientierten Zerlegung, indem die relationale Datenbankstruktur in eine XML-Struktur umgesetzt wird und umgekehrt. Diese Technik wird oft als „shredding“ bezeichnet<sup>6</sup>. Diese Integrationsart erfordert ein „Mapping“ der Struktur (siehe Kapitel 5 und Abbildung 6), auf dem auch der Fokus liegt. Als XML-Struktur eignet sich jedoch nur eine datenorientierte Struktur, deren Schema im Vorfeld bekannt ist, da sich ein gemischter Inhalt eines XML-Dokumentes nicht abbilden lässt<sup>7</sup>. Die originalgetreue Struktur kann bei diesem Verfahren nicht garantiert werden<sup>8</sup>.

<sup>6</sup> Schöning, (FN 1), S. 167

<sup>7</sup> Schöning, (FN 1), S. 183

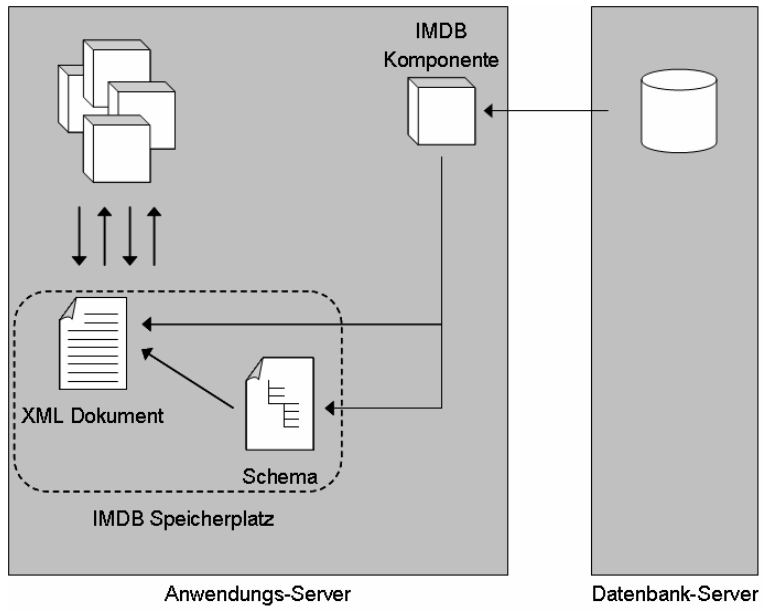
<sup>8</sup> Schöning, (FN 1), S. 183



**Abbildung 6** Relationale Daten werden gemappt

Es wird eine hohe Abstraktion von der Datenbank erreicht. Es ermöglicht das Erstellen von XML-Dokumenten, die nur Teilausschnitte der vorhandenen Daten repräsentieren. Die Architektur gestattet das Einfügen und Aktualisieren von Daten in die Datenbank durch XML-Dokumente.

XML kann die Performance der Anwendung steigern, indem die benötigten relationalen Daten im XML-Format zwischengespeichert werden. Dies gelingt jedoch nur bei sehr statischen Inhalten. Für diese „Caching-Strategie“ wird eine IMDB Komponente (in-memory database) auf der Anwenderseite integriert (siehe Abbildung 7). Da die Daten jedoch nur im flüchtigen Speicher zwischengespeichert sind, verliert diese Architektur an Robustheit. Dieser Ansatz einer IMDB Komponente kann mit den anderen Architekturmöglichkeiten kombiniert werden.



**Abbildung 7** XML-Daten sind im Speicher zwischengespeichert

## 4 Integrationsstrategie von XML mit relationalen Datenbanken

Da das Senden von Informationen Bandbreite in Anspruch nimmt und das Verpacken der Daten in XML-Markierungen Rechenzeit benötigt, ist der wichtigste Aspekt einer Integrationsstrategie, dass nur benötigte Daten betrachtet werden. Somit sollte das Datenmodell auf der XML-Seite möglichst klein gehalten werden.

Es empfiehlt sich ebenfalls auf der Datenbankseite möglichst mit virtuellen Sichten zu arbeiten, die die erforderlichen Daten entsprechend zur Verfügung stellen, anstatt die Informationen immer über die Tabellenbeziehungen neu zu generieren. Dies ist nur möglich, wenn immer die gleiche Zusammenstellung der Informationen benötigt wird. Durch diese virtuellen Sichten lassen sich zusätzlich auch bei einem bestehenden Datenbankmodell neue Spaltennamen für die Sicht vergeben. Dadurch werden die erstellten XML-Dokumente selbstbeschreibender und die XML-Markierungen werden von einigen Datenbankerverweiterungen „on demand“ optimiert. Die virtuellen Sichten unterstützen in den meisten Fällen nur lesenden Zugriff, so dass ein Einfügen oder Aktualisieren nicht realisierbar ist.

Sollte noch kein Datenbankmodell vorhanden sein, kann ein XML „freundliches“ Modell geplant werden. Hierzu muss folgendes beachtet werden.

Im Datenbankmodell sollte möglichst auf zu zahlreiche Beziehungen verzichtet werden. Das bedeutet jedoch nicht, dass das Datenbankmodell dadurch eingeschränkt wird. Es ist jedoch wesentlich einfacher, Datenbank und XML-Daten zu „mappen“, wenn das Modell weniger Beziehungen beinhaltet. Zusätzlich sollte eine Tabelle im Modell angelegt werden, in dem ganze Dokumente gespeichert werden können (z.B. CLOB, VARCHAR). Diese Daten sind zwar redundant, die Datenbank stellt möglicherweise eine automatisierte Funktion zum Synchronisieren durch Speicherprozeduren, „Trigger“ oder anderen Techniken zur Verfügung.

Es sollte vermieden werden, zusammengesetzte Schlüssel im Datenbankmodell zu verwenden. Die Schematechnik XSD unterstützt

allerdings derartige Schlüsselbildung. Beim Einsatz der DTD Technik ist diese nicht möglich.

Gültigkeitsüberprüfungen sollten nicht fest in der Anwendung implementiert werden. Dies passiert häufig, wenn die Verweis-Integrität sichergestellt werden soll. Dies ist allerdings die Aufgabe des Schemas und sollte es auch bleiben. Die XSD Schematechnik unterstützt ein dynamisches Einbinden von Prozessregeln durch den „appinfo“ Typen, der zur Verfügung gestellt wird und eine Art Kommentar ist.

Diese Technik der Gültigkeitsprüfung kann komplett in ein eigenes XSD Schema ausgelagert werden und muss nicht im entsprechenden Schema stehen. Somit lassen sich dynamisch zur Laufzeit die entsprechenden Regeln zusammenstellen, da ein XSD Schema selbst ein XML-Dokument ist und somit den gleichen Regeln unterliegt. Dies bedeutet, dass Anfragesprachen wie XQuery ebenfalls bei XSD greifen. Dies gilt natürlich nicht nur für das XSD Schema mit dem Inhalt des „appinfo“, sondern ebenfalls für das eigentliche Schema.

XML Dokumente zur Laufzeit zu generieren ist eine prozessorintensive Aufgabe und kann durch Caching-Strategien reduziert werden.

In Kapitel 2 wurde eine IMDB Komponente vorgestellt, die in betracht gezogen werden sollte, oder zumindest über andere Caching-Verfahren. Dies ist jedoch nur bei statischen Daten sinnvoll, wie z.B. Statusreporte oder Sitzungsinformationen.

Da die Suchanfragen in XML-Dokumenten wesentlich langsamer sind, gegenüber Datenbankabfragen, sollten die Daten vor dem „wrapping“ entsprechend gefiltert werden. Sollte im Vorfeld nicht klar sein, welche Daten benötigt werden, muss darauf geachtet werden, die XML-Dokumente möglichst klein zu modellieren.

## 5 „Mapping“ von XML und relationalen Daten

Wie schon in Kapitel 2 beschrieben, gehört die Integrationstechnik zu den aufwendigsten und wird in diesem Kapitel genauer erläutert.

Es wird vom folgenden Datenmodell ausgegangen, in dem Camp-Informationen abgespeichert werden und Bären, die in diesen Camps gesichtet wurden. Es besteht eine 1:n Beziehung.

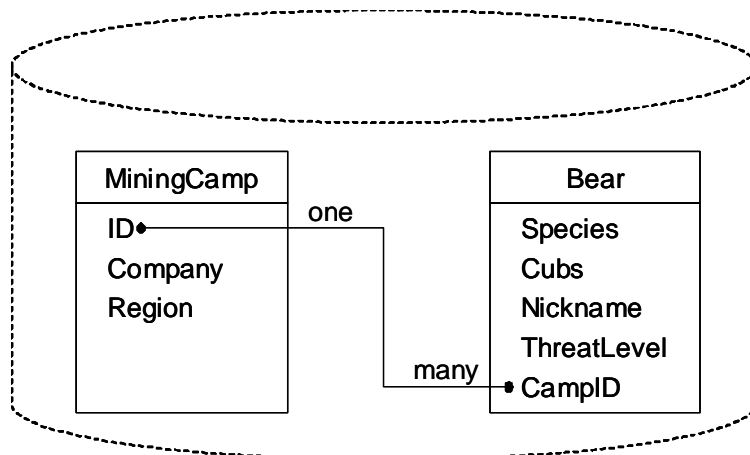
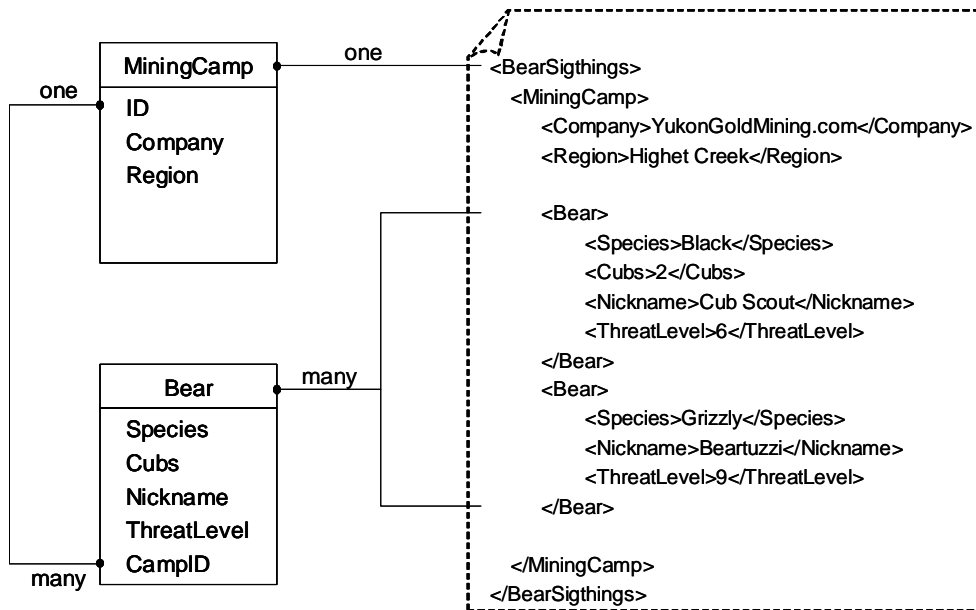


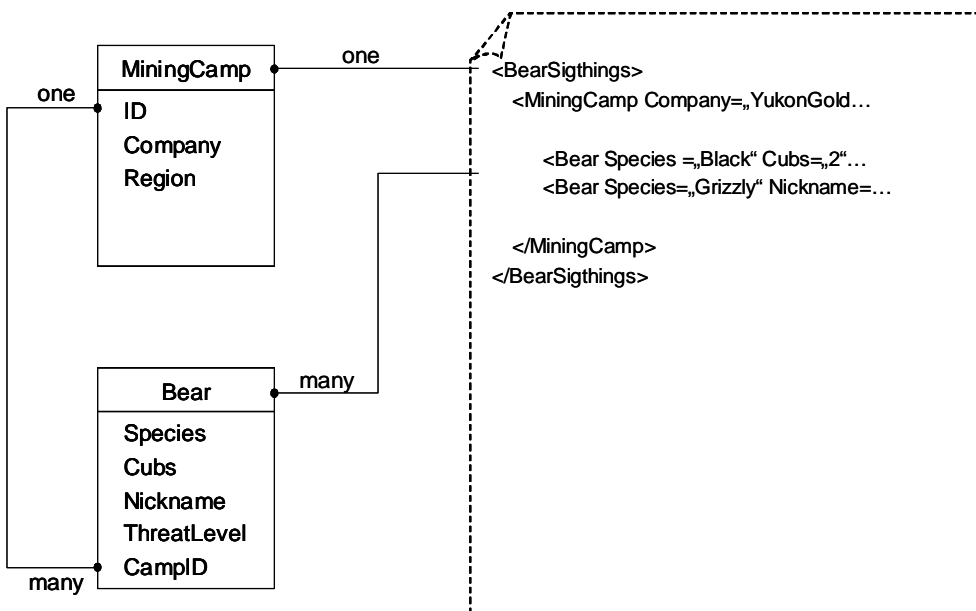
Abbildung 8 „Bear Sightings“ Datenbank

Der hierarchische Aufbau eines XML-Dokumentes erlaubt das einfache Abbilden einer 1:1 oder 1:n Beziehung, durch das Anordnen von Kindelementen innerhalb eines Elternelementes.



**Abbildung 9** Elementzentrierte Darstellung

Die Abbildung 9 zeigt dies entsprechend als elementzentrierte Darstellung, in der Tabellenzeilen durch Kindelemente repräsentiert werden. Ein XML-Dokument kann die Daten aber auch in einer attributzentrierten Darstellung wiedergeben, wie das folgende Beispiel zeigt (siehe Abbildung 10).



**Abbildung 10** Attributzentrierte Darstellung

Diese Darstellungsarten scheitern jedoch, sobald ein Kindelement nicht nur zu einem Elternelement eine Beziehung hat, sondern zu mehreren.

Für derartige Darstellungen benötigt man z.B. ein DTD oder XSD Schemata, die in der Lage sind, mit Zeigern zu operieren. Ein solches XML-Schema beschreibt schematisch die Struktur eines XML-Dokumentes, ähnlich dem eines Datenbankschemas. Es beschreibt somit den schematischen Aufbau eines XML-Dokumentes und kann eingesetzt werden, um deren Struktur zu erzwingen.

Zuerst soll das DTD Schema betrachtet werden. Es kann den Wertebereich, den die Daten annehmen dürfen, beschreiben und festlegen. Das DTD stellt hierfür lediglich drei Typen zur Verfügung. Dazu gehört der Typ ANY, PCDATA und EMPTY, sodass die Typinformation der Datenbank verloren geht und nicht validiert werden kann. Eine Möglichkeit, diese Restriktion zu umgehen, ist das Einfügen der Typinformation als ein Attribut.

```
<ThreatLevel DataType=„integer“>9</ThreatLevel>
```

Dadurch würde die Information nicht verloren gehen, das „ThreatLevel“ vom Typ „Integer“ ist. Da dieses jedoch selbst definiert wurde und außerhalb der eigenen Umgebung nicht bekannt gemacht werden kann und wofür diese Information steht, wird davon abgeraten. Diese Lösung ist kein Standardverfahren. Ein DTD Schema kennt auch nicht den Typen „NULL“, welcher von Datenbanken unterstützt wird. Hierfür kann zum Beispiel ein Schlüsselwort definiert werden, wie es das nächste Beispiel zeigt.

```
<ThreatLevel>NULL</ThreatLevel>
```

Es kann aber auch wie folgt gelöst werden: Sollte ein Element oder Attribut nicht im XML-Dokument vorkommen, besitzt es den Wert NULL. Beide Lösungen sind hier denkbar.

Primärschlüssel können in DTD durch das „ID“ Attribut dargestellt werden, die durch die XML-Spezifikation unterstützt wird. Es ist somit möglich, ein bestimmtes Element im XML-Dokument zu identifizieren. Als Beispiel wird ein ID Attribut namens „id“ in die MiningCamp Definition eingefügt.

```
<!ELEMENT MiningCamp (Company, Region, Bear*)>
<!ATTLIST MiningCamp id ID #REQUIRED>
```

Es ergeben sich jedoch für die Wahl eines Primärschlüssels folgende Restriktionen, die in einer Datenbank nicht gegeben sind. Der Schlüssel darf nicht mit einer Zahl beginnen. Das bedeutet, wenn der Schlüssel in der Datenbank als Zahl vorliegt, dieser ebenso konvertiert werden muss. Das Voranstellen des Tabellennamen oder ein Teil dessen, eignet sich hierfür.

z.B:

```
<MiningCamp ID=„Camp1“>
```

Dadurch umgeht man auch die zweite Restriktion, dass ein Schlüssel nur einmal im XML-Dokument auftreten darf. Dies ist der Fall, wenn zwei oder mehrere Tabellen im XML-Dokument mit identischen Schlüsseln abgebildet werden.

Für die Fremdschlüssel gilt dies ebenso. Sie werden durch das „IDREF“ oder „IDREFS“ Attribut abgebildet. Diese Attribute stellen die Beziehung zu einem oder mehreren „ID“ Attributen her. Da bei „IDREFS“ nicht zwingend ein Mehrfachverweis besteht, bietet es sich an, auf das „IDREF“ Attribut vollständig zu verzichten, da sich somit das Erweitern zu einem späteren Zeitpunkt einfacher gestaltet.

```
<!ELEMENT Bear (Species, Cubs?, Nickname, ThreatLevel)>
<!ATTLIST Bear CampID IDREF #REQUIRED>
```

Wird dieses auf das Modellbeispiel angewendet, ergibt sich daraus folgendes XML-Dokument (siehe Abbildung 11).

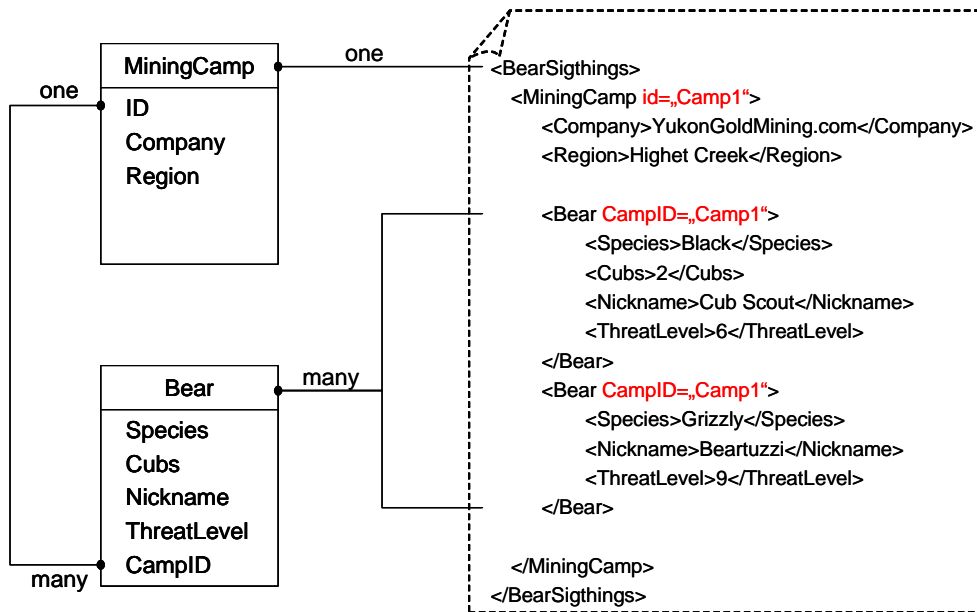


Abbildung 11 XML-Dokument mit ID and IDREF

Die Beziehungen zwischen Kind- und Elternelementen werden durch Zeiger, welche die Datenbankschlüssel repräsentieren, dargestellt. In diesem Fall wird dies zusätzlich noch durch die Verschachtelung der Elemente selber abgebildet, was nun nicht mehr erforderlich ist. Dadurch lässt sich das XML-Dokument in der Struktur folgendermaßen darstellen (siehe Abbildung 12).

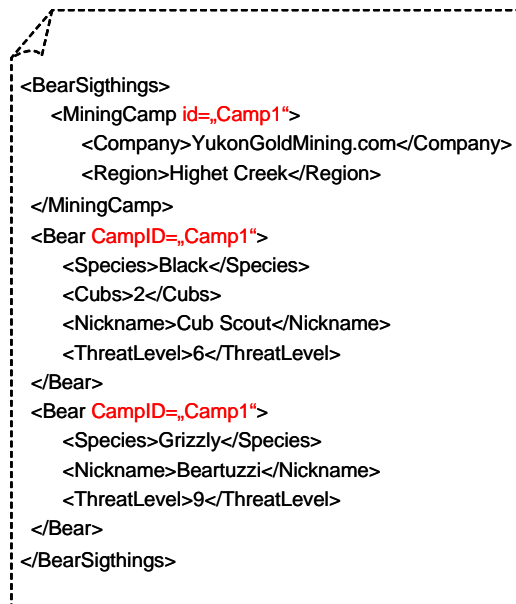


Abbildung 12 XML-Dokument mit ID and IDREF

Das ehemalige Kindelement „Bear“ steht jetzt in der gleichen hierarchischen Ebene, wie das „MiningCamp“ Element, und die hierarchische Ebene stellt nicht mehr die Beziehung her. Das vollständige DTD Schema hierzu:

```
<!ELEMENT BearSightings
  (MiningCamp+, Bear*)>
<!ELEMENT MiningCamp
  (Company, Region)>
<ATTLIST MiniCamp id ID #REQUIRED>
<!ELEMENT Company (#PCDATA)>
<!ELEMENT Region (#PCDATA)>
<!ELEMENT Bear
  (Species, Cubs?, Nickname, ThreatLevel)>
<!ATTLIST Bear CampID IDREFS #REQUIRED>
<!ELEMENT Species (#PCDATA)>
<!ELEMENT Cubs (#PCDATA)>
<!ELEMENT Nickname (#PCDATA)>
<!ELEMENT ThreatLevel (#PCDATA)>
```

Wie zu erkennen ist, entspricht die Syntax nicht dem eines XML-Dokumentes und stellt auch keines dar. Entsprechende XML-Suchanfragen, wie XQuery lassen sich nicht auf ein DTD Schema anwenden.

Dies sieht bei der Schematechnik XSD anders aus. Es ist selber ein XML-Dokument und bietet darüber hinaus noch weitere Vorteile. Es ist nicht wie DTD, auf drei Typen beschränkt, so dass die Typinformationen der Datenbank nicht verloren gehen, wie in Abbildung 13 zu sehen ist.

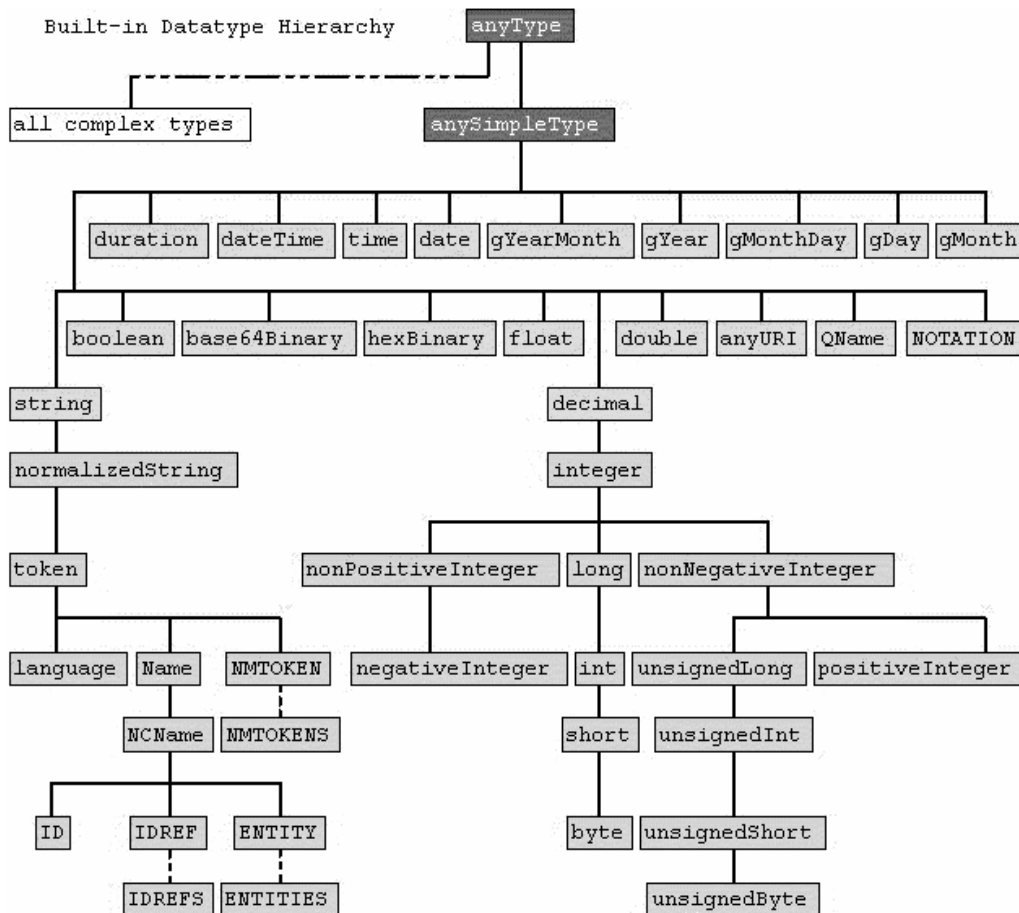


Abbildung 13 Vordefinierte Typen in XSD<sup>9</sup>

Zusätzlich erlaubt es XSD eigene Typen zu definieren. Diese ermöglicht es, eine Tabelle des relationalen Modells als eigene Typen zu definieren. Die „Bear“ Tabelle lässt sich somit wie folgt abbilden:

```

<element name=„Bear“>
  <complexType>
    <attribut name=„Species“ type=„string“ />
    <attribut name=„Nickname“ type=„string“ />
    <attribut name=„ThreatLevel“ type=„integer“ />
  </complexType>
</element>
  
```

Das Problem des „NULL“ Wertes besteht jedoch genauso wie beim DTD Schema und muss entsprechend behandelt werden.

<sup>9</sup> Erl, T.: „Service-Oriented Architecture – A Field Guide to Integrating XML and Web Services“; New Jersey: Prentice Hall PTR, 2004

Referenzbeziehungen sind in XSD weiterhin zum Nachbilden von Primärschlüssel- und Fremdschlüsselbeziehungen eines relationalen Datenbankmodells möglich. Primärschlüssel werden durch „key“ definiert, sodass sich für die „MiningCamp“ Tabelle folgende Definition ergibt.

```
<key name=„MiningCampPrimaryKey“>
  <selector xpath=„./MiningCamp“ />
  <field xpath=„PrimaryKey“ />
</key>
```

Das ID Attribut ist auch in XSD bekannt, doch bietet „key“ Vorteile. Es kann nicht nur ein Attribut als Schlüssel auftreten, sondern auch Elemente oder auch Kombinationen davon. Dies ist bei DTD nicht möglich<sup>10</sup>.

Zusätzlich kann mit den Xpath-Pfadausdrücken gearbeitet werden, um anzugeben, für welchen Pfad der Schlüssel eindeutig sein soll.

Bei den Fremdschlüsseln, kann im Gegensatz zur IDREF Technik angegeben werden, auf welchen Primärschlüssel er verweist. Definiert werden Fremdschlüssel mit „keyref“, was am Beispiel der „Bear“ Tabelle gezeigt wird.

```
<keyref name=„MiningCampForeignKey“ refer=„x:MiningCampPrimaryKey“>
  <selector xpath=„./Bear“ />
  <field xpath=„ForeignKey“ />
</keyref>
```

Beziehungen können in XSD anzahlmäßig mit „minOccurs“ und „maxOccurs“ festgelegt werden. Somit lassen sich alle Beziehungskombinationen wie 1:1, 1:n, n:1, m:n oder sogar Beziehungen wie 4:n abbilden.

Zusammenfassend lässt sich sagen, dass XSD zum Modellieren besser geeignet ist als DTD. Es besitzt ein umfangreiches Typsystem mit der Möglichkeit, eigene Deklarationen vorzunehmen. Das Konzept der Referenzbeziehungen entspricht eher dem einer Datenbank.

---

<sup>10</sup> Schöning, (FN 1), S. 46

## 6 Herstellerunterstützung von XML in Datenbanken

In diesem Kapitel wird nicht auf die einzelnen Tools (siehe Abbildung 14), die die Hersteller anbieten eingegangen, sondern allgemeiner auf die Bedeutung der proprietären Erweiterungen für die Architektur selbst.

	Oracle	SQL Server	IBM DB2
XML aus Relationen lesen	XSU und XSQL Servlet	DAD über SQL- und RDB- Zuordnung. XML-Objektgruppen	FOR XML
XML in Relationen schreiben	XSU und XSQL Servlet, iFS	DAD über RDB- Zuordnung. XML-Objektgruppen	Update Grams
XML Sichten über Relationen		DAD über RDB- Zuordnung. XML-Objektgruppen	XDR-, XMLSchema-Annotationen
Relationale Sicht über XML			OpenXML
Anfragen in XML-Spalten	SQL Constraints, WITHIN	XPath	XPath
Schreiben als Ganzes (CLOB oder XML-Typ)	InterMedia Text, Indizierung möglich, als CLOB, XMLType und extern	DAD, XML-Spalte, Indizierung über Seitentabelle, Update einzelner Elemente möglich, CLOB, XML-Typ	CLOB und extern

Abbildung 14 XML-Tools von Datenbankherstellern<sup>11</sup>

Die einzelnen Hersteller gehen ihre eigenen Wege, sodass eine Standardisierung nicht vorhanden ist. Mit der Wahl des RDBMS wird gleichzeitig eine bestimmte Zugriffsmethode gewählt.

SQL wird in einigen Fällen proprietär erweitert, um angeben zu können, ob die Anfrage als Ergebnis, ein XML-Dokument liefern soll. Auf der einen Seite wird die Anwendung stärker an das proprietäre SQL gebunden, gewinnt dadurch aber eine höhere Abstraktion zum Datenbankzugriff selbst. Durch eine Auslagerung in eine eigene Komponentenebene gelingt es der Anwendung trotzdem unabhängig vom RDBMS-Hersteller zu bleiben.

Oft werden Tools oder API-Schnittstellen, die für das Transferieren von vorhanden relationalen Daten zu und nach XML-Dokumenten, angeboten. Die Vorschriften für das „Mappen“ liegen oft in herstellereigenen Dateien vor und nicht im unabhängigen XSD oder XSLT Format, sodass eine Umstellung

<sup>11</sup> Schöning, (FN 1), S. 266

in andere Systeme sich als schwerer gestaltet. Die Anwendung selber wird durch die API-Schnittstelle ebenfalls an den Hersteller gebunden.

Hauptproblem ist die schlechte Unterstützung von Suchanfragen auf abgespeicherten XML-Dokumenten. Es werden nicht alle Anfragesprachen für XML unterstützt. Bei der Volltextsuche auf ein gespeichertes XML-Dokument haben Datenbanken die Schwierigkeiten, Daten von Markierungen zu unterscheiden.

Spezifische Herstellerunterschiede der drei meist verbreiteten Systeme ist in Abbildung 15 kurz tabellarisch dargestellt.

	Oracle	SQL Server	IBM DB2
Standardkonformität (Namensräume, XML Schema)	Ja	Eingeschränkt	Nein
Inhaltsorientierte Zerlegung generisch	Ja	Ja	Nein
Inhaltsorientierte Zerlegung definitorisch	Ja	Ja	Ja
Opake Speicherung	Ja	Nein	Ja
Originaltreue	Ja	Nein	Ja
Speicherung beliebiger XML Dokumente (Speicherung von rekursiven Elementen, gemischtem Inhalt, Kommentaren, Verarbeitungsanweis.)	Ja	Nein	Ja
Kodierungen	XML-Generierung: nur UTF-8 und UCS-2	Kann für Ausgabe via HTTP gewählt werden	inkonsistent
Validierung (DTD oder XML Schema)	Ja	Nein	Ja
Schema-Evolution möglich	Nein	Ja	Nein
Offenes Inhaltsmodell möglich	Nein	Ja	Nein
Speicherung schemaloser Dokumente möglich	Ja	Nein	Ja
Textsuche	(nur wenn ein Textindex definiert ist)	Nein	(nur wenn einer der Text-Extender vorhanden ist und ein Textindex definiert ist)
Direkte Web-Anbindung (reine URL-Adressierung)	unvollständig	gut	Nein
Erweiterbarkeit	Ja	Nein	Ja
Speicherungsstrukturunabhängigkeit	Nein	(es gibt nur eine Speicherungsform)	Nein

**Abbildung 15** Unterschiede in der XML-Unterstützung der Datenbankherstellern<sup>12</sup>

Es empfiehlt sich eine möglichst lose Verbindung zwischen der Anwendung und der herstellerspezifischen Datenbank zu entwerfen.

<sup>12</sup> Schöning, (FN 1), S. 266

## 7 Native XML-Datenbanken

XML-Fähigkeit wird von den relationalen Datenbanken nur mit Einschränkungen angeboten. Native XML-Datenbanken (NXDB) oder auch „Reine XML-Datenbanken“ genannt, sollen die Einschränkungen aufheben und bieten volle XML-Unterstützung an, ohne auf Datenbankfunktionalitäten zu verzichten.

Oft werden derartige Datenbanksysteme nicht in Unternehmen eingesetzt, obwohl deren Umgebung XML orientiert ist. Stattdessen wird weiterhin auf die bereits bestehenden RDBMS gesetzt. Es gibt jedoch Integrationsmöglichkeiten von NXDB in die bestehende Architektur. Sie können die Performance und Sicherung der Datenintegrität in der Architektur steigern.

NXDB können die XML-Dokumentenstruktur, unabhängig von ihrem Inhalt, speichern und verwalten. An diesem Punkt scheitern RDBMS mit XML-Unterstützung, da sie nicht in der Lage sind, Daten und Markierungen zu unterscheiden.

Weiterer Vorteil einer NXDB ist, dass das XML-Schema (DTD, XSD, etc.) integriert wird. In RDBMS können diese auch abgespeichert (siehe Kapitel 3), die aber wie Textdaten hinterlegt werden, da unbekannt ist, dass es sich um Schemainhalte handelt. NXDB können dadurch einen Index um die Schemastruktur selbst herum aufbauen. Es ergeben sich dadurch Leistungssteigerungen bei Abfragen und Datenaufrufen.

Die Schnittstelle ist in einem NXDB System XML-spezifisch, die jedoch nicht standardisiert ist. Die XML:DB-Initiative versucht hierfür einen Standard zu definieren.<sup>13</sup> Auch die NXDB Systeme selber unterscheiden sich untereinander sehr.

---

<sup>13</sup> Schöning, (FN 1), S. 310

## **8 Schlusswort**

Es wurde gezeigt, dass XML und relationale Datenbanken zusammen arbeiten können. XML kann als Austauschformat zwischen Anwendung und Datenbank fungieren und wird vor allem bei Web-Services verwendet. Die relationalen Datenbankhersteller haben sich dem Thema schon angenommen und bauen die Unterstützung von Version zu Version weiter aus.

## 9 Literaturverzeichnis

Erl, T.: „Service-Oriented Architecture – A Field Guide to Integrating XML and Web Services“; New Jersey: Prentice Hall PTR, 2004

Schöning, H.: „XML und Datenbanken“; München: Carl Hanser Verlag, 2003

Malhotra, A., V. Biron, P.: „XML Schema Part 2: Datatypes“, W3C Proposed Recommendation 30 March 2001 <http://www.w3.org/TR/2001/PR-xmlschema-2-20010330/> (Visit: 25.11.2005)