

Kommerzielle Anbieter von XML-Datenbanksystemen

Rupert Späth

info@rupert-spaeth.de

Abstract

Diese Arbeit beschäftigt mit den aktuellen Implementierungen von XML-Funktionalität in IBM DB2 UDB, MS SQL Server 2005 und Oracle Database. Dabei werden die wesentlichen Technologien näher untersucht, um XML-Daten zu schreiben, teilweise oder vollständig zu lesen und darin zu suchen. In diesem Zusammenhang spielen Performance-Optimierungen wie z. B. durch Indizes eine große Rolle. Weiterhin beschäftigt sich diese Arbeit mit der Änderung von XML-Dokumenten, wie sie in dem aktuellen Oracle-System implementiert ist. Dies geschieht anhand der Arbeit von Liu et al. „Effective and Efficient Update of XML in RDBMS“. Auch hier geht es nicht um die Sicht von außen (also der zur Verfügung gestellten Funktionalität) sondern, wie das Datenbanksystem intern funktioniert.

1 Einleitung

Auf den großen Datenbank-Konferenzen der letzten Jahre wurden von den kommerziellen Datenbanken hauptsächlich IBM DB2 UDB, MS SQL Server und Oracle Database behandelt. Zu MySQL konnte kein einziges wissenschaftliches Paper im Bereich XML (Extensible Markup Language) gefunden werden. Außerdem gibt es derzeit bei MySQL keine konkreten Planungen für einen XML-Datentyp [MySQL07].

In dieser Ausarbeitung werden die oben genannten drei Datenbanksysteme näher betrachtet. Dabei werden bei jedem dieser Systeme jeweils der XML-Datentyp, die Speicherung der XML-Daten, XML-Indizes und der lesende Zugriff auf die XML-Daten im Kapitel 2 näher betrachtet. Besonders XML-Indizes sind sehr wichtig, da mit Ihnen nicht nur ermittelt werden kann, welche Zeilen einer relationalen Tabelle mit einer XML-Spalte bearbeitet werden müssen sondern auch, wo innerhalb eines XML-Dokuments gesucht werden muss [MS04].

Andere wichtige Themen wie die XML-Schema-Validierung werden nicht behandelt, da sie den Umfang dieser Arbeit sprengen würden. Auch auf das wichtige Thema Änderung von XML-Daten wird im ersten Teil nicht näher eingegangen, da dies Gegenstand des zweiten Teils dieser Arbeit ist und in Kapitel 3 näher betrachtet wird.

2 Überblick über derzeit implementierte Funktionalität

In diesem Kapitel werden zunächst grundsätzliche Dinge über XML in Datenbanken angesprochen, bevor nacheinander die Systeme von IBM, Microsoft und Oracle näher untersucht werden. Alle vier Unterkapitel haben denselben Aufbau: Der XML-Datentyp, Speicherung von XML-Daten, XML Indizes und lesender Zugriff auf XML-Daten.

2.1 Allgemeines

2.1.1 Der XML-Datentyp

Die einfachste Art XML-Daten in einem relationalen Datenbanksystem zu speichern ist, die Daten in einem Text-Datentyp zu speichern [Rys05]. Dies hat den Vorteil, dass absolute Texttreue gewahrt wird, hat aber gleichzeitig den Nachteil, dass die Strukturinformationen des XML-Dokuments nicht genutzt werden können und dass es keine spezialisierten Funktionen gibt um Abfragen darauf anzuwenden. Deswegen wird dieser Ansatz nachfolgend nicht mehr näher betrachtet, sondern es werden nur noch Lösungen untersucht, die einen speziellen XML-Datentyp verwenden.

2.1.2 Speicherung von XML-Daten

Eine weit verbreitete Methode XML-Daten zu speichern ist das so genannte *shredding* bei dem XML-Dokumente, für die ein Schema vorliegt, auf mehrere Spalten aufgeteilt werden [Rys05]. Auch wenn keine Schemainformationen vorliegen ist es denkbar ein shredding vorzunehmen, wenngleich es weniger effizient ist. Der Prozess des shreddings ist auch umkehrbar. Inwieweit die einzelnen Datenbanksysteme auf diese Technik zugreifen wird in den entsprechenden Kapiteln näher dargestellt.

2.1.3 XML Indizes

Um Abfragen performant durchführen zu können werden auf die gespeicherten Daten Indizes erstellt [Rys05]. Allerdings ist die Erstellung und das Aktualhalten von XML-Indizes komplexer als bei relationalen Indizes. In den nachfolgenden Unterkapiteln wird noch gezeigt werden, dass sich die Systeme der unterschiedlichen Datenbankanbieter in diesem Bereich erheblich unterscheiden.

2.1.4 Lesender Zugriff auf XML-Daten

Wie bereits in Kapitel 2.1.1 erwähnt, wird die zusätzliche XML-Funktionalität im wesentlichen zur Verfügung gestellt, um effizient Daten zu lesen und zu schreiben. Während sich **XQuery** (XML Query Language, näheres in [W3C07a]) als Hauptabfragesprache inzwischen etabliert hat, ist die wissenschaftliche Diskussion um die Durchführung von Änderungen an XML-Daten noch nicht abgeschlossen. Dieses Thema wird ausführlich in Kapitel 3 diskutiert.

2.2 IBM DB2 UDB

2.2.1 Der XML-Datentyp

Der XML-Datentyp bildet den Kern der nativen XML-Unterstützung der DB2 [IBM05]. Dieser ist ein gleichberechtigter SQL-Datentyp. Dementsprechend können eine oder mehrere Spalten einer relationalen Tabelle vom Typ XML sein. Eine Zelle kann entweder ein well-formed XML Dokument (also dem W3C Standard entsprechend, siehe [W3C06]) oder NULL enthalten, wobei NULL anzeigt, dass kein XML-Dokument vorhanden ist. Relationale und XML-Daten werden in unterschiedlichen Formaten gespeichert. Die XML-Daten werden in einer hierarchischen Struktur gespeichert. Eine XML-Spalte kann sowohl schemalose Dokumente als auch schemagebundene XML-Dokumente enthalten. (Ein Schema ist z. B. durch DTD (Dokumenttypdefinition) oder XML-Schema definiert.) Der XML-Datentyp hat keine Länge, die mit ihm assoziiert ist und ist in der Länge unbegrenzt. Allerdings begrenzt das Client-Server-Kommunikationsprotokoll die Größe auf 2 GByte pro Dokument.

XML wird intern in keiner stringähnlichen Art verarbeitet, kann aber mittels der Funktion XMLSERIALIZE in eine Zeichenkette umgewandelt werden. Der umgekehrte Prozess kann mittels der Funktion XMLPARSE durchgeführt werden. Der XML-Datentyp kann nicht nur in Spalten, aber auch in Variablen, Parametern und Rückgabewerten in gespeicherten Prozeduren und benutzerdefinierten Funktionen verwendet werden.

2.2.2 Speicherung von XML-Daten

Eingehende XML-Daten werden mittels eines **SAX-Parsers** (Simple API for XML) auf Wohlgeformtheit (XML well formed) überprüft und es wird gegebenenfalls eine Schemavalidierung durchgeführt [IBM05]. Weitere Informationen über SAX können auf der offiziellen Seite von SAX: <http://www.saxproject.org> gefunden werden. Die SAX-Events werden in eine hierarchische Repräsentation des XML-Dokuments umgewandelt.

Betrachten wir dazu folgendes Beispiel:

```
<Gebäude>
  <Raum id=123>
    <Bezeichnung>Alpensaal</Bezeichnung>
    <Fläche>33</Fläche>
  </Raum>
  <Raum id=124>
    <Bezeichnung>Nordseekammer</Bezeichnung>
    <Fläche>8</Fläche>
  </Raum>
</Gebäude>
```

Die hierarchische Repräsentation schaut dann ähnlich aus wie der Baum in Abbildung 2.1. Während des Einfügens werden alle Bezeichner und Namespace URIs (Uniform Resource Identifier) des Baums in Ganzzahlen umgewandelt. Das Ergebnis wird in Abbildung 2.2 dargestellt.

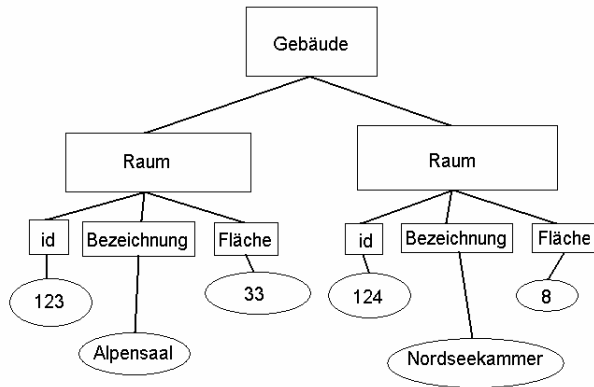


Abbildung 2.1 XML-Baum

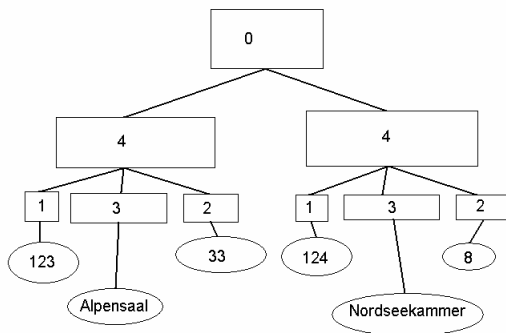


Abbildung 2.2 XML-Baum nach Ersetzung der Bezeichner.

Die bijektive Abbildungsvorschrift wird in der Tabelle SYSXMLSTRINGS gespeichert. Wie gezeigt gibt es für eine Textkette somit nur eine Zeile auch wenn diese in unterschiedlichen XML-Dokumenten verwendet wird. Für unser Beispiel würde der Tabellenausschnitt wie in Tabelle 2.1 dargestellt aussehen:

Gebäude	0
id	1
Raum	4
Bezeichnung	3
Fläche	2

Tabelle 2.1 Übersetzungen Textbezeichner-Ganzzahl-IDs

Beim ersten auftreten eines neuen Bezeichners wird dieser in die Tabelle SYSXMLSTRINGS eingefügt. Da die Tabelle üblicherweise relativ klein ist (hunderte oder tausende Einträge) und über einen speziellen Caching-Mechanismus verfügt, ist eine hohe Lese- und Schreib-Performance auf die Tabelle gewährleistet. Die tatsächliche Speicherung des XML-Dokuments auf der Festplatte ähnelt der Darstellung in Abbildung 2.2, enthält aber noch zusätzliche Informationen.

Ist ein Dokumentenbaum zu groß für eine Datenbankseite wird dieser in Regionen zerteilt und auf mehreren Datenbankseiten gespeichert. Da diese nicht notwendigerweise hintereinander liegen, werden die Regionen mit einem speziellen **Regionenindex** verkettet. Nähere Informationen zum Regionenindex finden sich in [IBM05].

2.2.3 XML Indizes

Die DB2 unterstützt zwei unterschiedliche Typen von XML-Indizes [IBM05]: **XML-Wert-Indizes** und **XML-Volltextindizes**.

XML-Wert-Indizes werden explizit angelegt. Dabei muss ein XML-Muster angegeben werden. Dieses XML-Muster stellt eine Teilmenge der gültigen XPath-Ausdrücke dar. (XPath ist eine XML-Abfragesprache, siehe dazu: [W3C07b]) Für jeden Pfad, der dem XML-Muster entspricht, wird ein Eintrag im XML-Index generiert. Um Speicherplatz zu sparen, werden nicht die ganzen Pfade gespeichert, sondern es werden anstatt der Bezeichner ganzzahlige Werte gespeichert. Das Prinzip funktioniert ähnlich dem Konzept, wie es in Kapitel 2.2.2 vorgestellt wurde. Die bestehende Volltextindexfunktionalität der DB2 wurde derart erweitert, dass sie nun auch den XML-Datentyp unterstützt. Dies ist insbesondere interessant, wenn es sich um inhaltsorientierte Dokumente handelt. Dabei kann entweder die ganze Spalte indiziert werden oder es kann nur ein Teilbereich vorgegeben werden, wenn bekannt ist, dass nur in diesem gesucht wird. Weiterhin ist Standardtextsuche-Funktionalität wie Gewichtung der Suchergebnisse verfügbar. Um die Performance zu verbessern wird der Volltextindex wie bei Textspalten asynchron aktualisiert, wobei eine Aktualisierung aber auch erzwungen werden kann.

2.2.4 Lesender Zugriff auf XML-Daten

XQuery ist eine primäre Abfragesprache der DB2 [IBM05]. Eine Übersetzung von XQuery nach SQL findet nicht statt [IBM06]. Abfragen werden in einer der beiden Sprachen an das Datenbanksystem gestellt und in einen Abfrageplan übersetzt. Ab da wird keine Unterscheidung mehr durchgeführt.

In einer XQuery wird der Zugriff auf relationale Spalten durch die beiden Eingabefunktionen db2-fn:xmlcolumn und db2-fn:sqlquery zur Verfügung gestellt. Dabei wird der Funktion db2-fn:xmlcolumn als Eingabeparameter ein Spaltenname übergeben, während db2-fn:sqlquery einen SQL-Ausdruck erwartet.

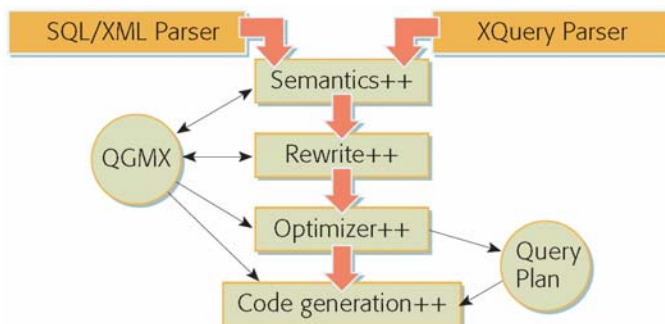


Abbildung 2.3 Der hybride SQL/XQuery Compiler der DB2 [IBM06]

In Abbildung 2.3 wird ein Überblick über den Abfrageübersetzer der DB2 vermittelt. Es wurde ein neuer **XQuery Parser** der bestehenden Funktionalität hinzugefügt und alle anderen Komponenten des Übersetzungsprozesses dahingehend erweitert dass XQuery unterstützt werden kann. Das Vorgehen im Einzelnen ist folgendes: Zuerst wird ein SQL-Befehl (mittels des **SQL/XML Parser**) oder ein XQuery-Ausdruck (mittels des **XQuery Parser**) in einen internen Datenflussgraphen übersetzt. Dabei wird eine semantische Analyse (**Semantics++**) gemacht. Als nächstes wird eine Umschreibetransformation (**Rewrite++**) durchgeführt um den Datenfluss zu normalisieren, zu vereinfachen und zu optimieren. Der Optimierer (**Optimizer++**) benutzt dieses Graph um einen physikalischen Plan (**Query Plan**) zu generieren, der durch den Codegenerator (**Code generation++**) in einen ausführbaren Code übersetzt. Im Falle von XQuery wird der Datenflussgraph durch ein internes Graphmodell (**QGMX: Query Graph Model XML**) dargestellt.

2.3 Microsoft SQL Server 2005

2.3.1 Der XML-Datentyp

Der SQL Server 2005 bietet native Unterstützung das XML Datenmodell mittels des Datentyps **xml** [MS04]. Dieser kann als Typ einer Spalte einer Tabelle oder einer Sicht sowie als Variable oder Parameter in der Funktion oder gespeicherten Prozedur verwendet werden.

Daten die im Typ xml gespeichert werden, können entweder vollständige Dokumente, aber auch ein Fragment sein. Eine xml-Spalte kann an eine Sammlung von XML-Schemas gebunden werden.

2.3.2 Speicherung von XML-Daten

Im SQL Server 2005 ist es nicht nur möglich XML in CLOB (Character Large Object) oder BLOB (Binary Large Object) (im SQL Server 2005: **nvarchar(max)** und **varbinary(max)**) zu speichern, sondern auch in dem XML-Datentyp [MS05a]. Dabei ist es wahlweise möglich eine XML-Tabelle anzulegen oder eine relationale Tabelle mit einer XML-Spalte.

Der SQL Server 2005 unterstützt das Parsen aus einer SQL Zeichenfolge oder einem binären Datentyp. Dabei gibt es unterschiedliche Konversionsoptionen die auch den Erhalt von Whitespaces und die Validierung gegen ein DTD ermöglichen.

Das Parsen des XML überprüft nicht nur, ob es well-formed ist, sondern erzeugt auch eine interne Repräsentierung auf physikalischer Ebene. Ähnlich wie relationale Datenbanken das logische Konzept von Reihen und Spalten auf eine byte-Anordnung abbilden, werden auch die Attribute und Elemente entsprechend binär abgebildet. Dieses interne Format kann durch XQuery-Befehle leichter ausgewertet werden und verringert die Größe gegenüber der vollen textuellen Darstellung in UTF-16 Kodierung (Unicode Transformation Format) um ca. 20-30%. Das interne Format kann auch an Clients übergeben werden, sofern sie dieses binäre XML-Format unterstützen. Da für die Speicherung die bestehende BLOB-Infrastruktur verwendet wird, dürfen diese binären XML-Daten jeweils nicht größer als 2 Gigabyte sein.

2.3.3 XML Indizes

Der SQL Server kann einen so genannten **primären Index** und 3 verschiedene Typen von sekundären Indizes erzeugen [MS05a]. Der primäre Index ist ein B⁺-Baum, dessen wichtigste Spalten sind: ORDPATH, TAG, NODE_TYPE, VALUE und PATH_ID [MS04]. Die ORDPATH-Spalte enthält für jeden Knoten, Attribut und Wert einen eindeutigen Wert, der vergleichbar einer Gliederungsnummerierung erstellt wird. In der Spalte TAG wird die Textauszeichnung (markup) gespeichert, wobei dieser nicht als Text gespeichert wird, sondern als Ganzzahl. Zwischen Text und Ganzzahl besteht eine bijektive Beziehung. Die NODE_TYPE-Spalte gibt an, ob es sich um ein Element, Attribut oder Wert handelt. Die VALUE-Spalte speichert den Wert, sofern es ihn gibt (sonst NULL), wobei die Werte in die entsprechenden SQL-Datentypen überführt werden, wenn der XML-Wert typisiert ist. Die PATH_ID-Spalte speichert einen Pfad ähnlich eines XPath-Ausdrucks, allerdings in umgekehrter Reihenfolge und unter Verwendung der Abbildungsvorschrift, wie sie für die Spalte TAG verwendet wird. Somit ist diese Spalte nicht eindeutig. Der primäre XML-Index enthält Redundanzen und ist größer als das XML-Dokument selbst. Deshalb wird das vorgestellte Verfahren noch optimiert und es wird mit Kompression gearbeitet.

Als sekundäre Indizes gibt es den **Pfadindex**, der einfache Arten von Pfadausdrücken unterstützt, den **Eigenschaftenindex**, der gängige Szenarios von Vergleichen von Eigenschaften unterstützt, und den **Wertindex**, der zu Hilfe genommen wird, wenn in der Abfrage Wildcards verwendet werden [MS05a]. An dieser Stelle wird nur vereinfacht skizziert, wie die sekundären XML-Indizes funktionieren, da eine vollständige Abhandlung den Rahmen dieser Arbeit sprengen würde. Eine detaillierte Darstellung findet sich bei [MS04]. Vereinfacht gesagt handelt es sich bei dem Pfadindex um einen Index auf die Spalten ORDPATH und PATH_ID des primären Indexes und beim Wertindex um einen Index auf die Spalte Value des primären XML-Index. Der Eigenschaftenindex wird verwendet, wenn einzelne Eigenschaften eines Objekts bei gegebenem Pfad abgefragt werden sollen. Darüber hinaus gibt es noch Inhaltindizes, die ähnlich einem SQL-volltextindex funktionieren und die XML-whitespaces mit beachten.

2.3.4 Lesender Zugriff auf XML-Daten

An dieser Stelle soll die Implementierung der XQuery-Funktionalität kurz skizziert werden.

Ein XQuery-Ausdruck wird analysiert und in eine interne Struktur namens XML Algebrabaum übersetzt [MS05b]. Auf diese wird eine regelbasierte Optimierung angewendet. Nach der Optimierung wird der XML Algebrabaum in einen relationalen Ablaufbaum transformiert.

Die XQuery-Übersetzung erzeugt einen Abfragebaum, der relationale Operatoren verwendet soweit primäre XML-Indizes verfügbar sind. Für nicht indizierte XML-Spalten enthält der Abfrageplan Operatoren um jedes XML-BLOB zu analysieren, die Knoten zu finden, die dem Pfadausdruck entsprechen und Zeilen zu generieren, die einem XML-Index ähneln. Ab diesem Punkt ist das weitere vorgehen im Wesentlichen dasselbe.

Der übersetzte Abfrageplan wird mit den bekannten relationalen Optimierungstechniken wie Kostenfunktionen und Histogrammen der Datenverteilung optimiert.

Als Ergebnis wird ein einziger Abfrageplan erzeugt, der sowohl den relationalen Teil als auch den XML-Teil umfasst.

2.4 Oracle Database

Die in diesem Kapitel dargestellten Technologien wurden in früheren Versionen der Oracle XML DB eingesetzt. Als Quellen wurden die neuesten wissenschaftlichen Paper herangezogen, die verfügbar waren. Allerdings wurde die XML-Funktionalität inzwischen erweitert [Ora07b]. Es kann deswegen keine Aussage darüber gemacht werden, in welchem Maße dies noch aktuell gültig ist.

2.4.1 Der XML-Datentyp

Die Oracle XML DB basiert auf dem Datentyp **XMLType** [Ora04]. Dieser native Datentyp ist ein Datentyp der vergleichbar zu anderen relationalen Datentypen ist. Dabei stehen XML-bezogene Funktionen zur Verfügung, die diesen Datentyp unterstützen. Diese können unter anderem dazu verwendet werden, XML Daten in relationale und objektrelationale Daten zu überführen (shredding) und umgekehrt.

2.4.2 Speicherung von XML-Daten

Die Oracle XML DB unterstützt verschiedene Möglichkeiten der Speicherung von XML-Daten [Ora07a]: Als Text in einer CLOB-Spalte (Character Large Object), als Objekte in objektrelationalen Tabellen, binärer in einer BLOB-Spalte (Binary Large Object) oder in einer hybriden Form, in der gewisse strukturierte Teile Objektrelational gespeichert werden und nicht-strukturierte Teile in LOBs (Large Objects) gespeichert werden. Die binäre Speicherung hat den Vorteil, dass die Daten komprimiert abgelegt werden können.

Beim **objektrelationalen Mapping** geht das genaue XML-Dokument verloren, da z. B. „white-spaces“ verloren gehen. Die Oracle XML DB hat eine Option die XML-Treue zu wahren. Dabei werden intern binäre Spalten (so genannte Positionsdeskriptoren) definiert, die Prozessierungsanweisungen, Kommentare, Namespaces und insbesondere leere Elemente (zur Unterscheidung von fehlenden Elementen) speichern. Diese Spalten werden während der internen SQL-Ausführung mit aktualisiert. Wird ein Element mit Reihenfolgentreue gespeichert, so wird eine Reihenfolgenindexspalte angelegt. Dadurch, dass Kommazahlen als Werte zugelassen sind, müssen die Elemente beim löschen oder einfügen nicht neu durchnummeriert werden, sondern es wird einfach die entsprechende Nummer gelöscht oder eine entsprechender Zwischenwert eingefügt.

2.4.3 XML Indizes

Die bevorzugte Speicherung von XML in der Oracle XML DB ist relational bzw. objektrelational [Ora04]. Da es dafür ausgefeilte Indexierungsfunktionalität gibt, werden diese intensiv verwendet. Die relationale Indexierung wird in diesem Papier nicht näher betrachtet, da der Fokus auf spezieller XML-Funktionalität liegt. In der Oracle XML DB gibt es zwei Arten spezieller XML-Indizes: Funktionale Indizes und Text Indizes [Ora03]. B-Baum Indizes können auf beliebige Ausdrücke einschließlich dem Ergebnis von Funktionen erstellt werden. Diese werden **funktionale Indizes** genannt. Der Abfrageoptimierer verwendet einen entsprechenden Index wenn eine entsprechender Ausdruck verwendet wird. So kann

z. B. auf einen XPath-Ausdruck ein Index gelegt werden. Wird ein XML-Dokument in einem CLOB gespeichert, so kann ein **Textindex** auf dieser Spalte errichtet werden. Bei einer entsprechenden Abfrage wird dieser verwendet.

Weiterhin gab es bei Oracle im Jahre 2005 Planungen für einen neuen Indextyp namens XMLIndex [Ora05b]. Dieser logische Index besteht aus einer physikalischen Pfadtable, die Zeilen entsprechenden der Knoten enthält. Sekundäre Indizes werden auf dieser Tabelle erzeugt. Das Konzept erinnert an die XML-Index-Funktionalität, wie sie beim MS SQL Server 2005 in Kapitel 2.3.3 beschrieben wurde. Inzwischen gibt es in der Oracle XML DB einen XMLIndex [Ora07b]. Inwieweit die angedachte Funktionalität so implementiert wurde, konnte nicht ermittelt werden.

Weiterhin gibt es bei Oracle neuere Forschungen, die die hier vorgestellten Indizierungsmethoden in Frage stellen [Ora07c]. So wird nun ein Konzept namens XMLTable Index propagiert. Dabei werden in mehreren Spalten nebeneinander Daten aus dem XML-Dokument gespeichert. Der wesentliche Vorteil dieses Konzepts ist, dass es unabhängig von der physikalischen Speicherung der XML-Daten ist. Eine genaue Behandlung des Konzepts würde nicht nur den Rahmen dieser Ausarbeitung sprengen sondern ist auch deshalb nicht zielführend, da es derzeit noch nicht implementiert ist.

2.4.4 Lesender Zugriff auf XML-Daten

In der Oracle XML DB wird ein XQuery Ausdruck in einen SQL-Ausdruck übersetzt und dieser wird ausgeführt [Ora05a]. Nachfolgend wird dargestellt, wie dieser Prozess funktioniert.

Die Abarbeitung von XMLQuery() und XMLTable()-Funktionen geschieht während der Übersetzung von SQL-Abfragen. Nach dem Analysieren des SQLs wird XMLTable in eine XMLQuery()-Funktion überführt. Nach der semantischen Analyse und der Typüberprüfung werden die einzelnen XMLQuery()-Funktionen bearbeitet. Der native XQuery Übersetzungstreiber analysiert den statischen XQuery-Text, macht die statische Analyse und Typüberprüfung und übersetzt es in SQL-Datenstrukturen. Dabei wird als Zwischenschritt XQueryX erzeugt. Die statische Typüberprüfung wird gleichzeitig dazu verwendet, um XQuery-Optimierungen durchzuführen. Falls der XQuery-Ausdruck nicht in SQL bzw. SQL/XML übersetzt werden kann, wird die XMLQuery()-Funktion so belassen, wie sie ist. Dies wird als der **hybride Ansatz** bezeichnet. Ab da wird die generierte SQL-Struktur wie gewöhnlich weiter verarbeitet. Kann keine SQL-Struktur erzeugt werden, so wird der XQuery()-Ausdruck mittels eines Koprozessors verarbeitet. Dabei wird das ganze XML-Dokument an einen XQuery-Prozessor übergeben und mit dem Ergebnis der Verarbeitung weiter gearbeitet. Dies ist allerdings wesentlich Imperformeranter als wenn nur Teile des XML-Dokuments mit dem umgeschriebenen SQL verarbeitet werden.

3 Effektives und effizientes Ändern von XML in relationalen Datenbanksystemen

Nach der Vorstellung der Systeme von IBM, Microsoft und Oracle wird nun im zweiten Teil dieser Arbeit das Paper von Liu et al. „Effective and Efficient Update of

XML in RDBMS“ [Ora07a] näher vorgestellt, da es nicht nur eine sehr aktuelle Veröffentlichung ist, sondern auch, weil in diesem Bereich aktuell noch geforscht wird. So wird klar im Abstract formuliert, dass dieses Paper den Zweck hat, die Standardisierung der Update-Funktionalität in SQL/XML zu beeinflussen.

3.1 Einführung

Derzeit werden unterschiedliche Mechanismen zur Manipulation von XML-Daten diskutiert. In Oracle Database 10g wurden Funktionen der Art insertXML(), updateXML() und deleteXML() als Erweiterungen von SQL/XML implementiert. Ein größerer Teil der Arbeit von Liu et al. beschäftigt sich mit dem Vergleich der von Oracle bereitgestellten Funktionalität und anderen wissenschaftlich diskutierten Ansätzen wie „XQuery!“ vorgestellt von Ghelli, Ré und Siméon sowie „XQueryP“ vorgestellt von Chamberlin et al.. Diese Vergleiche werden hier nicht näher beachtet, da das Ziel dieses Kapitels ist, einen vertieften Einblick in tatsächlich implementierter Funktionalität zu gewinnen.

3.2 Beispiel

In den nachfolgenden Unterkapiteln werden die genannten Beispieldaten verwendet:
Erzeugen einer XML-Tabelle:

```
Create table Bestellung of XMLType;
```

Erzeugen einer relationalen Tabelle mit 2 Spalten von Typ XML:

```
Create table Bestellung2 (Spalte1 XMLType, Spalte2 XMLType);
```

Als Daten wird folgendes XML-Dokument verwendet:

```
<Bestellung>
  <Bestellnummer>2008/01</Bestellnummer>
  <Kunde>Prof. Specht</Kunde>
  <Bestellzeile Nummer="1">
    <Artikelname>Datenbanksysteme: Eine
    Einführung</Artikelname>
    <Beschreibung>Dieses Buch vermittelt eine
    systematische und umfassende Einführung in moderne
    Datenbanksysteme.</Beschreibung>
    <Menge>1</Menge>
    <PreisProStueck>39,80</PreisProStueck>
    <Teile/>
  </Bestellzeile>
</Bestellung>
```

3.3 Erweiterungen von SQL/XML

Oracle unterstützt folgende XML Update Funktionen als Erweiterungen des SQL/XML-Standards: insertXMLBefore(), insertChildXML(), appendChildXML(), updateXML() und deleteXML(). Ein rename-Ausdruck mit dem XML-Knoten umbenannt werden können ist derzeit nicht verfügbar. Alle oben genannten Funktionen besitzen als ersten Eingabeparameter eine XML-Instanz und geben eine

veränderte Kopie dieser XML-Instanz als Rückgabewert zurück. Der zweite Parameter ist ein XPath-Ausdruck der die Zielknoten bestimmt. Bis auf deleteXML() haben alle Funktionen als dritten Parameter den zu schreibenden Wert. Als optionalen letzten Parameter besitzen alle Funktionen die Möglichkeit einen Namespace anzugeben. Eine detaillierte Beschreibung dieser Funktionen und Verwendungsbeispiele finden sich in [Ora07a], sind aber für das weitere Verständnis dieser Arbeit nicht erforderlich. Weiterhin wird in der Arbeit von Liu et al. eine Möglichkeit vorgestellt, wie automatisch Updatestatements generiert werden können, die ein gegebenes XML-Dokument in ein anderes überführen können. Auch dies wird hier nicht weiter betrachtet.

3.4 Das XML Update Rewrite-Konzept

Bisher wurde gezeigt, welche Funktionalität von Oracle XML DB zur Verfügung gestellt wird. Nun wird direkt auf das implementierte Konzept eingegangen. Die verschiedenen Speichermöglichkeiten wurden bereits in Kapitel 2.4.2 dargestellt und werden deswegen hier nicht mehr näher erläutert. Da in XML sowohl strukturierte Daten als auch unstrukturierte Daten gespeichert werden können gibt es nicht die eine optimale Lösung.

Die **funktionale Auswertung** einer XML-Update-Funktion erzeugt aus dem gespeicherten XML einen DOM-Baum (Document Object Model) und führt darauf die gewünschten Änderungen aus. Der entsprechende SQL-Befehl führt daraufhin die entsprechende Änderung in der Datenbank durch.

Beispiel:

```
Update Bestellung2 set  
Spalte1=deleteXML('/Bestellung/Kunde', Spalte1)
```

Der Update-Befehl ändert die Spalte während der deleteXML-Befehl eine veränderte Kopie erzeugt.

Dieses Vorgehen wird derzeit verwendet, wenn das XML in einer CLOB-Spalte gespeichert wird. Für andere Speichermodelle kann das vorgehen dahingehend optimiert werden, dass nur die Änderungen direkt durchgeführt werden ohne dass das ganze XML-Dokument ausgetauscht wird. Dies wird als die XML update rewrite-Technik bezeichnet, die sehr effizientes Ändern von XML ermöglicht. Die Herausforderung dabei ist, es für die unterschiedlichen Speicher und Index-Methoden entsprechende XML update rewrite-Lösungen zur Verfügung zu stellen.

3.5 Das XML update rewrite für Objekt-Relationale Speicherung

3.5.1 Der Algorithmus

Für den Fall, dass ein XML-Schema mit dem XML-Inhalt assoziiert ist, erlaubt die Oracle XML DB, dass der XML-Inhalt objektrelational gespeichert wird. In diesem Fall ist die darunter liegende objektrelationale-Speichertabellen-Metadaten-Information während der Kompilierzeit der SQL Befehle bekannt. Diese Information kann dazu verwendet werden, um die angewiesene Änderung derart umzuschreiben,

dass lediglich in den Tabellen Änderungen durchgeführt werden wo dies tatsächlich notwendig ist.

Betrachten wir folgendes Beispiel:

```
update Bestellung Best
set value(Best)=deleteXML(value(Best),
'/Bestellung/Bestellzeile[Nummer="1"]')
where XMLExists(' $doc/Bestellung[Bestellnummer = "2008-001"] '
passing by value value(Best) as "doc")
```

Konzeptuell wird dieses Beispiel in zwei Schritten ausgeführt:

Zuerst wird der where-Befehl der äußeren Abfrage ausgewertet um zu ermitteln, welche Zeilen geändert werden müssen. Diese Abfrage kann effizient ausgewertet werden, wie in [Ora04, Ora05a] gezeigt wird.

Dann wird für jede der oben identifizierten Zeilen in der Tabelle Bestellung rekursiv ein anderer DML-Befehl (Data Manipulation Language) ausgeführt, der tatsächlich die darunter liegenden Speichertabellen die zu dem betreffenden XML-Dokument gehören.

In SQL umgeschrieben lautet der erste Schritt somit:

```
select Best.pkeyCol from Bestellung Best where
Best.Bestellnummer = '2008-001'
```

Anmerkung: In der Quelle ist in dem entsprechenden Beispiel (Seite 931, rechte Spalte unteres Drittel) an dieser Stelle ein Fehler: purchaseOrder ist mit po veraliast. Somit ist purchaseOrder.pkeyCol kein gültiger Spaltenbezeichner. Dieser Fehler wird in der hier vorliegenden Ausarbeitung nicht gemacht.

Jeden Wert von pkeyCol aus der äußeren Abfrage wird als Bindevariable an den nachfolgenden DML-Befehl übergeben, der die tatsächlichen Änderungen an der darunter liegenden Speichertabelle BestellzeileTab durchführt.

```
delete from BestellzeileTab bzt
where bzt.Nummer='1'
and bzt.fkeyCol = :Best.pkeyCol
```

Im Originaldokument von Liu et al. wird der Prozess des Umschreibens von XML-Manipulationsausdrücken an einem aufwändigeren Beispiel erklärt. In dieser Arbeit wird zugunsten der Verständlichkeit ein einfacheres Beispiel verwendet, was allerdings den Nachteil hat, dass die Schwierigkeiten, die das umschreiben mit sich bringt nicht ersichtlich sind. In der Arbeit von Liu et al. wird gezeigt, dass auch XPath-Ausdrücke wie /s1/c1[pred1]/c2[pred2]/c3[pred3] in SQL umgeschrieben werden können, wobei mit geschachtelten Unterabfragen gearbeitet wird, die in umgekehrter Reihenfolge wie die entsprechenden XPath-Teile aufgeschrieben werden müssen.

Eine wesentliche Beschleunigung der Ausführung findet nicht nur dadurch statt, dass die generierte Befehlsfolge durch den Abfrageoptimierer verarbeitet wird sondern auch, dass interne APIs (Aplication Programing Interface) verwendet werden.

3.5.2 Unterstützung rekursiver Schemata

Rekursive Schemata werden durch die Verwendung des Konzepts Objekt-ID und Objektverweis des objektrelationalen SQLs unterstützt. Dies wird an folgendem Beispiel erläutert. Gegeben sei das rekursive Dokument:

```
<Abschnitt>
  <Name>N1</Name>
  <Abschnitt>
    <Name>N2</Name>
  </Abschnitt>
  <Abschnitt>
    <Name>N3</Name>
    <Abschnitt>
      <Name>N4</Name>
    </Abschnitt>
  </Abschnitt>
</Abschnitt>
```

Es werden zwei objektrelationale Tabellen erzeugt, um dieses XML-Dokument zu speichern.

Eine XML-Tabelle die alle Abschnittselemente egal welcher Schachtelungstiefe speichert und die Verschachtelungstabelle, die die Eltern-Kind-Beziehungen abbildet. Diese sind in den Abbildungen 3.1 und 3.2 dargestellt.

Objekt-Id	Name	Primärschlüssel
oid1	N1	P1
oid2	N2	P2
oid3	N3	P3
oid4	N4	P4

Abbildung 3.1 – XML-Tabelle zum speichern aller Abschnittselemente

Fremdschlüssel	XML-Verweis
P1	oid2
P1	oid3
P3	oid4

Abbildung 3.2 – Verschachtelungstabelle zur Speicherung der XML-Verweise

Es ist leicht zu sehen, dass der Abschnitt N1 kein Elternelement hat, dass die Abschnitte N2 und N3 als Elternelement N1 haben und dass N3 das Elternelement von N4 ist.

Aufbauend auf diesen Tabellen kann der selbe XML update rewrite-Algorithmus verwendet werden.

3.5.3 Schlussfolgerungen und Ausblick

In dieser Ausarbeitung wurde kurz ein Einblick in das XML update rewrite gegeben. Das Dokument von Liu et al. steigt wesentlich tiefer ein. So wird z. B. gezeigt, dass dieser Ansatz auch verwendet werden kann, wenn nur Teile eines XML-Dokuments einem Schema gehorchen. In unserem Beispiel könnte zum Beispiel die Beschreibung ein XHTML-Dokument (eXtensible HyperText Markup Language:

Seitenbeschreibungssprache, die XML-wellformed ist.) sein, das keinem Schema zugeordnet ist. Gezeigt wurde, dass mittels dieses Algorithmus ein effizientes Ändern von XML-Dokumenten möglich ist.

3.6 XML update rewrite für binär gespeicherte XML-Dokumente

Traditionell werden Änderungen an binär gespeicherten Daten derart vorgenommen, dass der ganze Wert der BLOB-Zelle ausgetauscht wird, was natürlich aufwändig ist. Deswegen wird eine neue BLOB-Infrastruktur bereitgestellt, die effizientes, stückweises Ändern von BLOB-Zellen ermöglicht: Wird an einer Stelle etwas eingefügt, so wird nicht an dieser Stelle Daten eingefügt, weil dann nämlich der Rest verschoben werden muss, sondern die Daten werden hinten an das BLOB **angefügt**. Somit wird der Offset zum **logischen Offset** und ist kein physikalischer Offset. Somit wird das BLOB wie verkettete Datenfragmente organisiert. Bei Bedarf kann das BLOB reorganisiert werden um die interne Fragmentierung zu reduzieren. Diese Strategie macht das Aktualisieren skalierbar und effizient. Wieweit dieses Vorgehen die Abfrageleistung reduziert wird von Liu et. al nicht angegeben.

Auf binären XML-Daten können XML-Indizes erstellt werden. Der logische XML-Index ist physikalisch als Pfadtabelle implementiert, die in jeder Zeile Information über einen XML-Knoten speichert. In einer Zeile werden folgende Informationen eines Knotens verwaltet: Dokument-Id, Positionsinformation über den XML-Knoten und der Pfad vom Wurzel-Element bis zu diesem Knoten. Außerdem wird der Wert des Knotens gespeichert, wenn es sich um ein Blatt-Element oder einen Attributs-Knoten handelt. Wenn es sich um einen komplexen Knoten handelt, beinhaltet die Spalte der Pfadtabelle den darunter liegenden BLOB-Locator, der dazu benutzt werden kann, diesen Knoten effizient innerhalb des BLOBs zu laden. Wenn XML-Daten geändert werden, muss auch der XML-Index geändert werden. Da aber der BLOB-Locator das logische Offset besitzt wird, der Aufwand während der Index-Aktualisierung drastisch reduziert.

Der XML-Änderungsprozess hat immer noch eine äußere Abfrage, die die Zeilen ermittelt, die geändert werden müssen. Die `XMLExists()`-Funktion in der Where-Bedingung kann effizient durch die Verwendung des XML-Index ausgewertet werden. Sobald die XML-Dokumente identifiziert sind, werden die Knoten ermittelt, auf die Änderungen angewendet werden sollen. Jede einzelne XML-Änderungsfunktion erzeugt eine Menge von stückweisen BLOB-Änderungsoperationen, die jeweils die alte Stelle (Offset) und Länge sowie den neuen Wert mit Längenangabe enthalten. Wie bereits oben dargestellt, können in der neuen BLOB-Infrastruktur diese Änderungen durchgeführt werden, ohne dass das ganze original-BLOB getauscht werden muss. Nachdem die XML-Daten geändert wurden muss auch der XML-Index aktualisiert werden, damit dieser konsistent zu den Daten ist.

3.7 Ausblick

In diesem Kapitel wurde gezeigt, dass effizientes ändern von XML-Daten durch Umschreiben auf SQL-Befehle möglich ist. Dabei wurden nicht alle Aspekte der

Arbeit von Liu et al. dargestellt. Zum Beispiel wurde auf die tatsächliche Performance-Messung nicht eingegangen. Die Diskussion, wie die Syntax der XML-Änderungsanweisungen lauten soll ist noch nicht abgeschlossen. Somit wird auch in den nächsten Jahren im Bereich effektives und effizientes Ändern von XML-Daten geforscht werden.

4 Zusammenfassung

In dieser Ausarbeitung wurde ein kleiner Einblick in die interne XML-Technologie der Datenbanksysteme von IBM, Microsoft und Oracle gegeben. Dabei wurde gezeigt, dass zwar alle Systeme einen **XML-Datentyp** besitzen, auf diesen **Indizes** erstellen können und die Daten mittels **XQuery** abgefragt werden können, aber dass die Systeme intern sehr unterschiedlich funktionieren. Weiterhin wurde am Beispiel Oracle gezeigt, dass die XML-Funktionalität noch nicht vollständig ist und daran aktuell geforscht wird. Man darf gespannt sein, wie die fortschreitende Standardisierung von XML-Funktionalität die weitere Entwicklung beeinflussen wird.

Quellen:

[IBM05] Nicola, van der Linden
Native XML Support in DB2 Universal Database
31st VLDB Conference, Trondheim, Norway, 2005
<http://www.vldb2005.org/program/paper/thu/p1164-nicola.pdf>
Abgerufen am: 30.05.2008

[IBM06] Beyer et al.
DB2 goes hybrid: Integrating native XML and XQuery with relational data and SQL
IBM SYSTEMS JOURNAL, VOL 45, NO 2, 2006, S. 271-298
<https://www.research.ibm.com/journal/sj/452/beyer.pdf>
Abgerufen am: 31.05.2008

[MS04] Pal et al.
Indexing XML Data Stored in a Relational Database
30th VLDB Conference, Toronto, Canada, 2004
<http://www.vldb.org/conf/2004/IND5P2.PDF>
Abgerufen am: 07.06.2008

[MS05a] Michael Rys
XML and Relational Database Management Systems: Inside Microsoft® SQL
Server™ 2005
SIGMOD 2005, June 14–16, 2005, Baltimore, Maryland, USA.
<http://www.cs.uiuc.edu/class/fa05/cs591han/sigmodpods05/sigmod/pdf/p958-rys.pdf>
Abgerufen am: 01.06.2008

[MS05b] Pal et al.
XQuery Implementation in a Relational Database System
31st VLDB Conference, Trondheim, Norway, 2005
<http://www.vldb2005.org/program/paper/thu/p1175-pal.pdf>
Abgerufen am: 01.06.2008

[MySQL07] Jon Stephens
MySQL Forums :: XML :: Any plans for Xml data type?, 2007
<http://forums.mysql.com/read.php?44,155525,157883#msg-157883>
Stockholm, Sweden, 2007
Abgerufen am: 18.05.2008

[Rys05] Rys, Chamberlin, Florescu
XML and Relational Database Management Systems: the Inside Story
SIGMOD'05, June 14–16, 2005, Baltimore, Maryland, USA.
<http://www.cs.uiuc.edu/class/fa05/cs591han/sigmodpods05/sigmod/pdf/p945-rys.pdf>
Abgerufen am: 31.05.2008

[Ora03] Murthy, Banerjee
XML Schemas in Oracle XML DB

29th VLDB Conference, Berlin, Germany, 2003
<http://www.oracle.com/technology/tech/xml/xmlldb/Misc/vldbschemas4.pdf>
Abgerufen am: 01.06.2008

[Ora04] Krishnaprasad et al.
Query Rewrite for XML in Oracle XML DB
30th VLDB Conference, Toronto, Canada, 2004
<http://www.isys.ucl.ac.be/vldb04/eProceedings/contents/pdf/IND5P1.PDF>
Abgerufen am: 01.06.2008

[Ora05a] Liu, Krishnaprasad, Arora
Native XQuery Processing in Oracle XMLDB
SIGMOD2005, June 14-16, 2005, Baltimore, Maryland, USA
<http://www.cs.uiuc.edu/class/fa05/cs591han/sigmodpods05/sigmod/pdf/p828-liu.pdf>
Abgerufen am: 01.06.2008

[Ora05b] Murthy et al.
Towards An Enterprise XML Architecture
SIGMOD2005, June 14-16, 2005, Baltimore, Maryland, USA
http://www.oracle.com/technology/tech/xml/xquery/pdf/oraclexmltutorialfinal_otn.pdf
Abgerufen am 11.06.2008

[Ora07a] Liu et al.
Effective and Efficient Update of XML in RDBMS
SIGMOD'07, June 12–14, 2007, Beijing, China
<http://wwwcs.upb.de/cs/boettcher/lehre/WS07/sem-ws07/liu-Effective-and-Efficient-Update-of-XML-in-RDBMS-sigmod2007.pdf>
Abgerufen am: 31.05.2008

[Ora07b] Drake et al.
New Features in Oracle XML DB for Oracle Database 11g Release 1
Oracle Corporation, World Headquarters, Redwood Shores, California, USA
<http://www.oracle.com/technology/products/database/oracle11g/pdf/xml-db-11g-whitepaper.pdf>
Abgerufen am 11.06.2008

[Ora07c] Liu et al.
XMLTable Index - An Efficient Way of Indexing and Querying XML Property Data
2007 IEEE 23rd International Conference on Data Engineering, 15-20 April 2007, Istanbul, Türkei
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4221768
Abgerufen am 11.06.2008

[W3C06] Bray et. al.
Extensible Markup Language (XML) 1.0 (Fourth Edition), 2006,
<http://www.w3.org/TR/xml/>

Abgerufen am 18.05.2008

[W3C07a] Boag et al.
XQuery 1.0: An XML Query Language
W3C Recommendation 23 January 2007
<http://www.w3.org/TR/xquery/>
Abgerufen am 11.06.2008

[W3C07b] Berglund et al.
XML Path Language (XPath) 2.0
W3C Recommendation 23 January 2007
<http://www.w3.org/TR/xpath20/>
Abgerufen am 11.06.2008