DB2 goes SQL:2003 : Native SQL/XML-Unterstützung der IBM DB2-Datenbank

Gunnar Thies, Gottfried Vossen

Institut für Wirtschaftsinformatik

Universität Münster

Leonardo-Campus 3, 48161 Münster

Abstract

Die Verarbeitung von XML-Daten in objektrelationalen Datenbanken war lange Zeit nur unzureichend umgesetzt, obwohl die SQL/XML-Erweiterung bereits im SQL:2003-Standard definiert wurde. Mittlerweile ist in den aktuellen Releases vieler Datenbankhersteller diese Erweiterung umgesetzt. In diesem Bericht setzen wir uns mit der IBM DB2 Version 9 auseinander und zeigen, wie IBM's SQL/XML-Erweiterung den SQL:2003-Standard umsetzt. Dabei werden nicht nur XML-Inhalte am Beispiel eingelesen, bearbeitet und ausgegeben, sondern auch XML-Schemata zum Validieren eingesetzt und Indizes für XML-Daten erzeugt.

1 Einführung

Die *Extensible Markup Language* (XML) erlangt immer größere Beliebtheit in Wissenschaft und Wirtschaft. Dieses semi-strukturierte Format zur Speicherung von Daten findet sich vor allem in großen Teilen des World Wide Web: Viele Standards (beispielsweise des W3C) werden durch XML-Strukturen definiert und beschrieben. XML basiert auf der *Standard Generalized Markup Language* (SGML), die unter anderem von Charles Goldfarb entwickelt wurde und sich seit 1986 als ISO-8879 durchsetzte. XML 1.0 wurde 1996/1997[1] vom W3C vorgestellt.[2] XML basiert auf einer wohlgeformten Baumstruktur, die zunächst nur folgenden Regeln folgt:

- Es gibt genau ein Wurzelelement;
- jedes Tag beginnt mit einem Starttag und endet mit einem End-Tag (Ausnahmen sind leere Tags, diese enden mit />);
- Verschachtelungen innerhalb des Dokuments müssen streng geordnet sein und in umgekehrter Reihenfolge ihres Öffnens wieder geschlossen werden.

---

[1] Das genaue Datum ist schwer feststellbar, vgl. www.w3c.org oder [MI02], S. 26.

[2] Nähere Informationen zum 10-jährigen Bestehen des W3C unter: http://www.w3.org/2004/11/15-presskit-single.html

Die Struktur eines XML-Dokuments lässt sich über eine Grammatik beschreiben. Die beiden zur Verfügung stehenden Grammatik-Formalismen sind *Document Type Definition* (DTD) oder *XML-Schema*, anhand derer die Gültigkeit eines XML-Dokuments – zur im Grammatik-Dokument vorgeschriebenen Struktur – geprüft werden kann. Die aktuellste Empfehlung zu XML ist in der W3C Recommendation des 16. August 2006 zu XML 1.0 (Fourth Edition)[3] nachzulesen.

XML wird beispielsweise in der Prozessmodellierung für die *Business Process Execution Language* (BPEL), bei Web Services für die *Web Service Description Language* (WSDL, Version 2.0 seit Juni 2007), bei Vektorgrafiken für *Scalable Vector Graphics* (SVG), beim Aufbau von Webseiten für die *Extensible Hypertext Markup Language* (XHTML) sowie in vielen anderen Anwendungen verwendet. Um spezifische Daten aus einem XML-Dokument zu extrahieren, gibt es mehrere Möglichkeiten. Zum Einen ist hierbei die Sprache *XSLT* zu nennen, die es ermöglicht, aus einem XML-Dokument durch spezifische Mechanismen ein Subset (einen Teilbaum) als neue XML-Datei zu erhalten. Zum Anderen gibt es die Sprachen *XPath* und *XQuery*, die es ermöglichen, unterschiedliche Formen von Pfadausdrücken sowie Daten aus einem XML-Dokument nach komplexen Anfragen zu extrahieren.

Traditionell werden Daten in strukturierter Form heute überwiegend in relationalen Datenbanken gespeichert. Gängige Systeme sind IBM DB2, Oracle 10g, MySQL und PostgreSQL um nur einige zu nennen. Die Daten werden darin in Tabellen angeordnet und in Tupeln (Zeilen) abgespeichert. Um Daten aus einer Datenbank zu extrahieren, editieren und zu verarbeiten verwendet man SQL (und einige systemspezifische Dialekte bzw. Erweiterungen). Damit können weitgehend komplexe Anfragen an eine Datenbank formuliert werden.

Oft haben XML-Daten und relationale Datenbanken bisher nebeneinander existiert.[4] Das Abspeichern von XML-Dateien in relationalen Datenbanken als Datentyp Binary (oder BLOB) ist schon seit längerer Zeit möglich. Der große Nachteil daran war bisher aber, dass XML-Inhalte nur als Ganzes ausgelesen und bearbeitet werden konnten: bei DB2 beispielsweise mit dem XML Extender, obwohl die als SQL/XML bezeichnete SQL-XML-Erweiterung für relationale Datenbankmanagementsysteme bereits in dem SQL:2003 Standard definiert wurde. Diese native XML-Unterstützung (bei DB2 als „pureXML“ bezeichnet), die einen direkten Bezug zwischen dem Inhalt einer XML-Datei und den Inhalten einer relationalen Datenbank herstellen kann, wird mittlerweile in einigen aktuellen Versionen relationaler oder objektrelationaler Datenbanksysteme geboten.

Um einen generellen, herstellerunabhängigen Überblick über den in SQL:2003 definierten Befehlsumfang von SQL/XML zu erlangen, bietet sich Standardlektüre wie [TUE03] oder [ST05] an. Wir wollen in diesem Bericht nicht die Gesamtheit der SQL/XML Syntax von SQL:2003 vorstellen, vielmehr soll deren konkrete Umsetzung innerhalb der DB2 Version 9 exemplarisch gezeigt werden. Dazu gehen wir im Folgenden auf verschiedene Arten des Einfügens und des Bearbeitens von XML-Daten und darüber hinaus auf das Validieren anhand von XML-Schema-Definitionen und das Erstellen von Indizes für XML-Daten ein.

In Abschnitt 2 wird auf die Unterstützung von XML in der neuesten Version der relationalen Datenbank DB2 (Version 9.0, zuvor als Viper bezeichnet) von IBM eingegangen und untersucht,

---

[3] http://www.w3.org/TR/2006/REC-xml-20060816/

[4] Es existieren auch XML-Datenbanken, wie bspw. das Open Source Projekt *eXist* (http://exist.sourceforge.net), die XML-Daten in Datenbankmanier abspeichern und die Daten per XQuery und XUpdate (einer Möglichkeit der einfachen XML-Editierung) bearbeiten können.

welche Einsatzszenarien sich damit ergeben.[5] Abschnitt 3 widmet sich dann einer kurzen Vorstellung zweier möglicher Einsatzszenarien der Verknüpfung von XML und relationaler Datenbanken bzw. der Verwendung der nativen XML-Unterstützung. In Abschnitt 4 wird abschließend ein Fazit gezogen.

2 Nutzungsszenarien

Man kann hierbei zunächst zwischen zwei grundlegend Fälle unterschieden:

- Speichern von XML-Daten in der Datenbank
- Anfragen oder Bearbeiten von gespeicherten XML-Daten aus der Datenbank

Der erste Fall lässt lediglich zwei Unterscheidungen zu: die ungeprüfte Speicherung von XML und die Speicherung nach Validierung gegen ein XML-Schema (bzw. einer DTD). Der zweite Fall lässt sich in einige verschiedene Unterarten einteilen. So lassen sich bspw. Teile aus den XML-Daten extrahieren und editieren (per XQuery oder SQL), relationale Daten und XML-Daten in einer SQL-Abfrage mischen und sogar Views über XML-Dateien erstellen. Im Folgenden wird ein gängiges Beispielszenario aus den Tutorials[6] von IBM kurz vorgestellt, und es werden anhand derer die Möglichkeiten der DB2 erläutert. Im Anschluss daran wird ein kurzer Blick auf die Indexierungsmöglichkeiten der XML-Daten geworfen.

Für unser Beispiel werden zwei Tabellen in der Datenbank angelegt (siehe Tabelle 1):

- Tabelle Items (Produkte)
- Tabelle Client (Kunden)

Wichtig ist hierbei jeweils das letzte Attribut, das den Datentyp XML aufweist. Die Datei wird hier binär gespeichert und kann dann durch XQuery oder SQL/XML (SQL angereichert mit XML-spezifischen Teilen) durchsucht werden.

Tabelle: Items	
Name	**Datentyp**
id	int, PK
brandname	varchar(30)
itemname	varchar(30)
sku	int
srp	decimal(7,2)
comments	xml

Tabelle: Clients	
Name	**Datentyp**
id	int, PK
name	varchar(50)
status	varchar(10)
contactinfo	Xml

Tabelle 1: Datentabellen für das Beispiel.

[5] Dabei wird die Funktionsweise von XQuery und XPath nicht näher erklärt.

[6] Es existiert mittlerweile ein IBM Redbook, das sich mit der Thematik pureXML beschäftigt: http://www.redbooks.ibm.com/abstracts/SG247298.html?Open

Hat man nun XML-Dateien für Kundeninformationen und Produktkommentare vorliegen, werden diese entsprechend der schematischen Zeichnung in Abbildung 1 zu einem jeweiligen Datensatz gespeichert. Pro Datensatz kann hier jeweils eine komplette XML-Datei gespeichert werden.

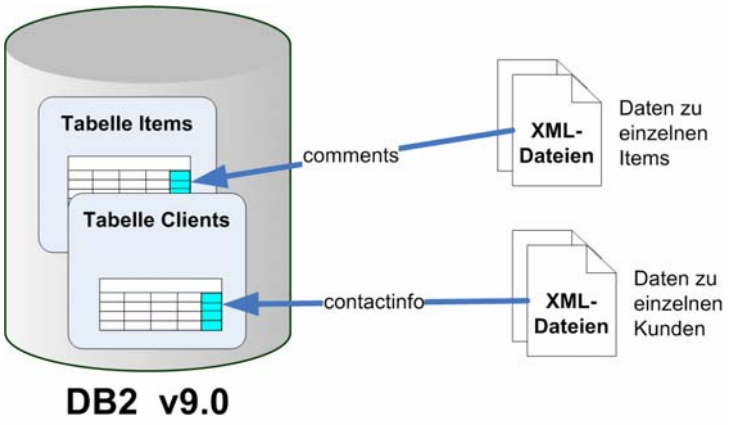

Abbildung 1: XML-Daten-Speicherung in der DB2.

Auf die Darstellung der SQL-Syntax zur Erstellung der Tabellen wird hier verzichtet.

2.1 Speichern von XML-Dateien

Nachdem die Datenbankstruktur angelegt ist, sollen die ersten Datensätze in den Tabellen gespeichert werden. Um schnelle Ergebnisse zu erlangen, schicken wir folgendes SQL-Statement direkt zur Datenbank:

```
insert into clients
values (1,'Max Müller','Gold',
'<addr>Musterstr. 1, Münster, Germany</addr>');
```

Listing 1: Einfaches Insert-Statement.

Somit wird ein Datensatz angelegt, der im Attribut *contactinfo* die Adressdaten in der angegebenen XML-Form speichert.

Es gibt eine weitere Möglichkeit, XML-Strukturen in die Datenbank zu speichern, die ein wenig komfortabler ist. Hierfür benötigt man die XML-Quelldateien und deren Namen in einem Verzeichnis (hier zur Vereinfachung C:/IBM/XMLdemo) und erstellt eine ASCII-Datei, die die einzutragenden relationalen Daten und einen Verweis auf die dazugehörigen XML-Dateien enthält (Listing 2). Diese Datei speichern wir unter dem Namen *clients.del*.

```
2, Ute Mayer, Gold, <XDS FIL='Client2.xml' />
3, Max Frisch, Gold, <XDS FIL='Client3.xml' />
4, Peter Müller, Silver, <XDS FIL='Client4.xml' />
```

Listing 2: ASCII-Datei zum Datenimport.

Der Inhalt einer ClientX.xml-Datei sieht hier beispielsweise aus wie in Listing 3:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<client>
   <address>
      <street>Musterstr. 4</street>
      <city>Münster</city>
      <state>Germany</state>
      <zip>48155</zip>
   </address>
   <phone>
      <work>0251/1234567890</work>
      <home>0251/1212121212</home>
      <cell></cell>
   </phone>
   <email>ute.mayer@uni-muenster.de</email>
</client>
```

Listing 3: XML-Datei eines Kunden.

Es muss nun für jede in Listing 2 aufgeführte Zeile eine entsprechende XML-Datei vorhanden sein; dann kann mit folgendem Statement der Datenimport gestartet werden.

```
import from C:/IBM/XMLdemo/clients.del of del xml from C:/IBM/XMLdemo
insert into clients;
```

Listing 4: SQL-Befehl zum Import.

Sollten die XML-Dateien gefunden werden, so sind anschließend alle drei Personen mit ihren zugehörigen XML-Daten in der Datenbanktabelle *clients* zu finden.

Der Vorteil von XML-Dateien gegenüber Plaintext oder CSV-Dateien ist die durch Grammatiken ermöglichte Gültigkeitsprüfung. So kann eine XML-Datei – bevor sie in der Datenbank gespeichert wird – auf Validität überprüft werden. Daher kann man im XML-Schemarepository (XSR) der DB2 XML-Schemas registrieren, die diese Prüfung übernehmen. Zunächst gehen wir von folgendem XML-Schema (Listing 5) aus.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="address">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="street" type="xsd:string"/>
            <xsd:element name="city" type="xsd:string" />
            <xsd:element name="state" type="xsd:string" />
            <xsd:element name="zip" type="xsd:integer"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="phone">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="work" type="xsd:string"/>
            <xsd:element name="home" type="xsd:string"/>
            <xsd:element name="cell" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="client">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="address"/>
            <xsd:element ref="phone"/>
            <xsd:element name="email" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Listing 5: XML-Schema für Kunden.

Dieses (rudimentäre) Schema entspricht exakt der XML-Struktur unsere ClientX.xml-Dateien und soll nun im XSR der DB2 registriert werden. Es ist unter C:/IBM/ClientSchema.xsd gespeichert und kann mit folgendem Statement als Schema „mySchema“ registiert werden.

```
register xmlschema 'http://placeholder.for.namespaces' from
         C:/IBM/XMLdemo/ClientSchema.xsd as mySchema complete;
```

Listing 6: Registrierung eines XML-Schemas.

Nach erfolgreicher Registrierung kann man XML-Daten beim Importieren per ASCII-Datei von diesem Schema validieren lassen. Wir löschen zunächst alle Daten aus der Tabelle *Clients* und fügen die Daten aus der ASCII-Datei (Listing 2) mit Hilfe des nachfolgenden Befehls wieder zur

Tabelle hinzu; die Besonderheit: diesmal wird das Schema „mySchema“ als Validierungsschema angegeben. Die XML-Dateien, die nicht gültig sind, werden auch nicht in die Tabelle gespeichert.

```
import from C:/IBM/XMLdemo/clients.del of del xml from C:/IBM/XMLdemo
xmlvalidate using xds default mySchema
insert into clients;
```

Listing 7: Hinzufügen von Daten mit XML-Schema-Validierung.

Auftretende Fehler beim Importieren der Daten werden nun von DB2 zurückgegeben und man hat die Möglichkeit, die XML-Dateien dementsprechend zu verbessern. DB2 bietet auch die Möglichkeit eine DTD zu hinterlegen und dann darauf die XML-Valididätsprüfung vorzunehmen, dies wird hier aber nicht weiter betrachtet.

Wir haben nun die wichtigsten Möglichkeiten besprochen, wie relationale Daten und XML-Dateien in eine Tabelle gespeichert werden. Es folgt eine Betrachtung der vielfältigen Ausgabemöglichkeiten per XQuery und SQL (bzw. SQL/XML).

2.2 Verarbeitung und Anfrage von XML-Dateien

Beim Anfragen von Daten gibt es im Falle der DB2 (Viper bzw. Version 9) zwei Möglichkeiten. Die erste realisiert Abfragen mit SQL/XML, die zweite stellt Anfragen in XQuery-Notation. Anhand der folgenden Beispiele soll exemplarisch gezeigt werden, wie gängige Anfragen an die Datenbank bzw. deren XML-Inhalte gestellt werden können. Die Ausgangsbasis für die hier besprochenen Anfragen bilden die oben angelegten Tabellen und deren Inhalte.

2.2.1 Anfragen mit SQL/XML

Kenntnisse über normale „Select From Where“-Statements und deren Aufbau werden hier vorausgesetzt. Um die XML-Daten aus der Datenbank anzusprechen, verwenden wir hier die spezielle XML-Syntax von IBM als Ergänzung zum Standard-SQL.

SQL/XML

Wir gehen von obiger XML-Kundenkontaktinformation aus (Auszug einer ClientX.xml-Datei):

```
<?xml version=”1.0”? encoding=”ISO-8859-1”>
<client>
   <address>
      <street>Musterstr. 4</street>
      <city>Muenster</city>
      <state>Germany</state>
      <zip>48155</zip>
   </address>
   …
```

Listing 8: Auszug aus der ClientX.xml.

Betrachten wir nun folgende Aufgabe: Wir suchen alle Zeilen der Tabelle *Clients*, die in Münster wohnen. Dazu benutzen wir folgendes SQL-Statement (Listing 9).

```
SELECT name FROM clients
WHERE xmlexists('$c/client/address[city="Münster"]'
                passing clients.contactinfo as "c")
```

Listing 9: SQL-XML-Statement.

Man sieht im Statement, dass das Attribut *contactinfo*, in dem die XML-Daten der Kunden liegen, durch den Befehl *passing clients.contactinfo as „c"*, an die Variable c gebunden wird. Anschließend kann durch die Traversierung am XML-Baum durch den XPath *$c/client/adress* ein Attribut auf der Stufe *address* – hier *city* – als WHERE-Bedingung angegeben werden. Somit erhalten wir alle Namen der Kunden aus der Tabelle *client*, in denen die XML-Datei den Ort der Person als „Münster" angibt. Wichtig ist hier die Notation mit den eckigen Klammern (…*address[city="Münster"]*), die nicht mit einer SQL-typischen Notation verwechselt werden darf (…*address/city = "Münster"*).

In obigem Fall bekamen wir „relational gespeicherte Werte" zurück: alle Namen, die der WHERE-Klausel entsprachen. Nun wollen wir XML-Informationen erhalten (z.B. die Straße, in der der Kunde wohnt) und verwenden dazu den Befehl *xmlquery* in der SELECT-Klausel (siehe Listing 10).

```
SELECT xmlquery('$c/client/address/street' passing cli-
ents.contactinfo as "c")
FROM clients
WHERE xmlexists('$c/client/address[city="Münster"]'
                passing clients.contactinfo as "c")
```

Listing 10: XML-Rückgabewerte per SQL/XML-Klausel.

Die Rückgabe ist in diesem Fall auf XML-Basis, d.h. es wird jeweils ein Teil der XML-Datei zurückgegeben.

```
<?xml version="1.0" encoding="UTF-16" ?><street>Musterstr. 4</street>
<?xml version="1.0" encoding="UTF-16" ?><street>Autorenstr.
1</street>
<?xml version="1.0" encoding="UTF-16" ?><street>Müllerstr.
40</street>
```

Listing 11: XML-Ausgabe.

Dies ist nur ein erstes Beispiel für die Ausgabemöglichkeiten von XML-Daten über die DB2. Es lassen sich auch in SQL-typischer Art Bedingungen und Ausgaben kombinieren, Views über XML-Daten erstellen etc.

Im folgenden Beispiel (Listing 12) verwenden wir den Befehl *xmltable,* um XML-Daten auf gleicher Ebene mit relationalen Daten auszugeben.

```
SELECT c.name, ct.city, ct.workphone
FROM clients as c,
xmltable('$c/client' passing c.contactinfo as "c"
          columns city varchar(100) path 'address/city',
                  workphone varchar(100) path 'phone/work') as ct
```

Listing 12: XML und SQL auf selber Ebene ausgeben.

Durch die Verwendung des Befehls *xmltable* wird aus der Telefonnummer am Arbeitsplatz (phone/work) und dem Wohnort (address/city) eine temporäre Tabelle erstellt, deren Attribute dann mit den „relationale" vorliegenden Attributen einheitlich ausgegeben werden kann. Als Ergebnis des obigen Statements erhält man eine Tabelle mit folgenden Informationen:

NAME	CITY	WORKPHONE
Ute Mayer	Münster	0251/1234567890
Max Frisch	Münster	0251/9878889
Peter Müller	Münster	0251/100200300

Tabelle 2: Ausgabe relationaler und XML-Daten.

Auf die gleiche Weise lassen sich auch View, Joins etc. über XML-Daten erstellen. Weitere Beispiele sind in [SA06a] zu finden.

2.2.2 Anfragen mit XQuery

Eine weitere Möglichkeit der Anfrage (neben den XPath-Ausdrücken) besteht in der Formulierung von XQuery-Statements. XQuery und die Syntax der FLWOR-Ausdrücke werden an dieser Stelle nicht weiter vertieft und es wird davon ausgegangen, dass die Syntax und die Funktionsweise bekannt sind.[7] Um mit einem einfachen Beispiel zu beginnen, wollen wir alle Straßennamen (genau wie oben mit dem SQL/XML-Statement) der Kunden ausgegeben bekommen.

```
xquery
for $y
in db2-fn:xmlcolumn('CLIENTS.CONTACTINFO')/client/address/street
return $y
```

Listing 13: Einfacher XQuery-Ausdruck.

Das Ergebnis dieser Anfrage ist eine XML-Struktur der extrahierten Daten und sieht folgendermaßen aus:

---

[7] Nähere Informationen zu XQuery unter http://www.w3.org/TR/xquery

```
<street>Musterstr. 4</street>
<street>Autorenstr. 1</street>
<street>Muellerstr. 40</street>
```

Listing 14: Ergebnis des einfachen X-Query-Ausdrucks.

Man kann durch die Verwendung des Befehls *text()* (also: *return $y/text()*) auch nur den Text ohne die XML-Tags erhalten; dies ist aber hier nicht gewünscht.

Als zweites Beispiel soll hier ein kompletter FLWOR-Ausdruck vorgestellt werden, der die Telefonnummer (der Arbeitsstelle) von jedem Kunden ausgibt, der entweder in Münster wohnt oder überhaupt eine Emailadresse angegeben hat (also irgendeine Form der Kundenkommunikation möglich ist).

```
xquery
for $y in db2-fn:xmlcolumn('CLIENTS.CONTACTINFO')/client
where $y/address/city="Münster" or $y/email!=""
order by $y/zip
return $y/phone/work
```

Listing 15: Kompletter FLWOR-Ausdruck.

Es lassen sich so auch alle weiteren Formen von XQuery-Ausdrücken auf der DB2-Datenbank ausführen: beispielsweise if-then-else-Statements, HTML-Ausgaben, let-Klauseln, etc.

Ein wichtiger Aspekt ist die Verknüpfung von XQuery mit SQL-Statements; Dies kann mit dem Befehl *xquery*, wie auch schon im vorherigen Abschnitt beschrieben, geschehen. Ein Beispiel für die Verwendung ist in Listing 16 zu sehen.

```
SELECT name, xmlquery('for $e in $c/client/email[1] return $e'
                      passing contactinfo as "c")
FROM clients
WHERE status = 'Gold'
```

Listing 16: Verknüpfung von XQuery und SQL.

Die Where-Klausel muss auf jeden Fall mit einfachen Anführungszeichen versehen werden!

2.2.3 Generierung von XML- oder HTML-Strukturen

Es ist ebenfalls möglich, direkt aus der Datenbank HTML oder XML-Ausgaben zu erstellen, auch aus relationalen Datenbeständen. Dies soll in zwei kurzen Beispielen verdeutlich werden.

Wenn wir direkt XML-Strukturen erstellen wollen, so müssen wir den Befehl *xmlelement* verwenden; dieser erlaubt das „Erstellen" von XML-Tags. Dies ist beispielsweise sinnvoll, um vorhandene XML-Strukturen umstrukturiert auszugeben oder vorhandene relationale Daten in XML-

Struktur auszugeben (oder auch relationale und XML-strukturierte Daten in ein XML-Dokument zu schreiben). Im folgenden Listing ist ein Beispiel wiedergegeben, wie Daten aus der Tabelle *Clients* (in gemischter Form) als XML-Dokument ausgegeben werden können.

```
SELECT xmlelement(name “Kunde”,
          xmlelement(name “id”, c.id),
          xmlelement(name “name”, c.name),
          xmlelement(name “stadt”,
                       xmlquery(‘$c/client/address/city/text()’
                                 passing c.contactinfo as “c”)))
FROM clients as c
WHERE c.id = 2
```

Listing 17: XML-Dokument erstellt aus relationalen und XML-Daten.

Aus dem obigen Listing ergibt sich dann folgendes XML-Dokument:

```
<Kunde>
  <id>2</id>
  <name>Ute Mayer</name>
  <stadt>Muenster</name>
</Kunde>
```

Listing 18: Aus DB2 erstelltes XML-Dokument.

Weitere Möglichkeiten ergeben sich hier durch die Verwendung des Befehls *xmlagg,* der es ermöglicht, ganze Sequenzen von XML-Daten auszugeben, Näheres dazu findet man in [CS06]).

HTML-Strukturen sind von Vorteil, wenn wir Daten aus der Datenbank z. B. direkt auf einer Webseite in „realtime“ anzeigen wollen. Dazu verwenden wir die im vorigen Abschnitt gezeigten FLWOR-Ausdrücke.

```
SELECT xmlquery(‘for $e in $c/client/email[1]/text()
                 return <p>{$e}</p>’
                 passing client.contactinfo as ”c“)
FROM clients
WHERE status = ‘Gold’
```

Listing 19: FLWOR-Ausdruck direkt in HTML.

Mit diesem Konstrukt erhalten wir alle Emailadressen der Kunden, die den Status „Gold“ haben, jedoch als HTML-Ausgabe.

2.2.4 Aktualisieren, Editieren und Löschen von XML-Daten

Um XML-Daten zu editieren, werden analog zu der Bearbeitung von relationalen Daten UPDATE- oder DELETE-Statements benutzt. Der einzige Unterschied in der Verwendung der SQL-Befehle besteht beim Aktualisieren von XML-Daten in der Datenbank. Es können hier **keine** partiellen Änderungen an den XML-Daten vorgenommen werden, es kann lediglich der **gesamte** Inhalt geändert werden.

2.3 Indexierung von XML-Daten

Hier soll ein kurzer Blick auf die Indexierungsmöglichkeiten der DB2-Datenbank geworfen werden. Die Indexierung erfolgt im Wesentlichen analog zu der von relationalen Daten. Auch dort kann man einen speziellen Index für ein Attribut einer Tabelle anlegen: so ist es also auch hier möglich einen Index auf einen speziellen Teil der XML-Struktur zu erstellen. Man kann hierbei zwei Arten unterschieden: den Index auf einen speziellen Wert oder den Volltextindex.

2.3.1 Wertindex

Eine weitere Neuerung bei XML-verarbeitenden Datenbanken ist die Möglichkeit, einen Index auf ein beliebiges Tag einer XML-Struktur zu erstellen. So lassen sich performante Suchoperationen durchführen. Hier wollen wir am Beispiel der Emailadresse der Kunden einen UNIQUE-Index auf ein XML-Tag erstellen.

```
CREATE UNIQUE INDEX idx1 ON clients(contactinfo)
GENERATE KEY USING XMLPATTERN '/client/email' AS SQL varchar(100)
```

Listing 20: Indexerstellung auf die Emailadresse der Kunden.

Durch diese einfache Syntax wird ein Index erstellt, der gewährleistet, dass keine doppelten Emailadressen in den XML-Daten der Tabelle *Clients* auftreten. Im XMLPATTERN-Teil wird der Pfad zum zu indizierenden Teil der XML-Struktur angegeben und durch AS SQL Type lässt sich der Datentyp des Index für das spezielle Datum angeben. Folgende Datentypen können verwendet werden: VARCHAR(n), VARCHAR HASHED,DOUBLE, DATE und TIMESTAMP.

2.3.2 Volltextindex

Der Volltextindex ermöglicht bessere Suchergebnisse auf dem kompletten Bestand der XML-Daten und sollte bei größeren Textsammlungen innerhalb der XML-Dateien verwendet werden. Ein Volltextindex macht am ehesten Sinn, wenn er das gesamte XML-Dokument erfasst: dies sollte mit folgendem Code (Listing 21) möglich sein[8].

[8] Leider konnte der Volltextindex nicht mit dieser Syntax – obwohl von zwei Autoren so erklärt – in der Datenbank angelegt werden. Möglicherweise ist die Syntax für die Endversion (DB2 9) gegenüber der Beta (DB2 Viper) doch noch geändert worden.

```
CREATE INDEX idx2 FOR TEXT ON clients(contactinfo) FORMAT XML
```

Listing 21: Volltextindex auf die XML-Daten erstellen.

3 Mögliche Nutzung von „pureXML“ am Beispiel

Die XML-Features der DB2 werden in der Presse und von IBM selbst mit dem Schlagwort „pureXML“ betitelt. Dies trifft auch durchaus zu, da man alle relevanten XPath und XQuery-Funktionalitäten verwenden kann. Im Folgenden werden zwei Beispielszenarien vorgestellt, in denen der Nutzen einer XML-verarbeitenden Datenbank deutlich zu sehen ist.

Die derzeitige Standardtechnik zur Umsetzung von Service-orientierten Architekturen sind Web Services, die eine lose Kopplung von Diensten in einem heterogenen Umfeld ermöglichen. Die technische Beschreibung eines solchen Dienstes wird dabei in einem XML-Dokument – dem *Web Service Description Language*-Dokument (WSDL) – vorgehalten. Darin werden einzelne Methoden, Input- und Outputparameter und weitere Daten beschrieben. Um einen passenden Web Service zu finden, werden z. B. UDDI-Verzeichnisse (Universal Description, Discovery and Integration) verwendet: Darin sind Web Services, deren Anbieter und Beschreibungen der Dienste hinterlegt. Einige große Unternehmen nutzen solche Verzeichnisse als eine Art „Gelbe Seiten“, um mögliche IT-Dienstleistungen zu finden.

Wir streben an, eine Art UDDI-Verzeichnis von Dienstleistungen zu konzipieren (siehe Abbildung 2), welches (vorerst nur interne) Web Services verwaltet. Natürlich kann hier auf UDDI direkt zurückgegriffen werden; möglicherweise ist aber die Komplexität der UDDI-Spezifikation (aktuell in Version 3.0.2) ein technischer Overkill, der vermieden werden kann. In Abbildung 2 ist ein solches Szenario für den Anwendungsfall einer Universität dargestellt. Hierbei ist eine Datenbank (DB2 9) mit einer Tabelle hinterlegt, die eine ID, eine Beschreibung sowie eine Metadaten- und WSDL-Datei im XML-Format speichern kann. Ein (hier nicht näher spezifiziertes) Web-Frontend lässt die einzelnen Benutzer nach passenden Services suchen. Mit einer Suche werden so leicht alle Daten (eben auch technische Details aus der WSDL-Datei) eines Services angesprochen.

Ein weiteres Szenario ist beispielsweise ein E-Learning-System (wie xLx[9]), bei dem XML-Kenntnisse gefördert werden und gefordert sind. Hier kann man die Validierung der zu speichernden XML-Daten anhand eines XML-Schemas so verwenden, dass gültige Lösungen zu einer Aufgabe direkt gespeichert (und damit auch automatisch korrigiert) werden. Ungültige Lösungen werden erkannt, da sie nicht validiert werden können. Ein möglicher Ansatz ist in Abbildung 3 zu sehen.

---

9 *eXtreme eLearning eXperience*: E-Learning-System am Institut für Wirtschaftsinformatik (Database Group) der Universität Münster.

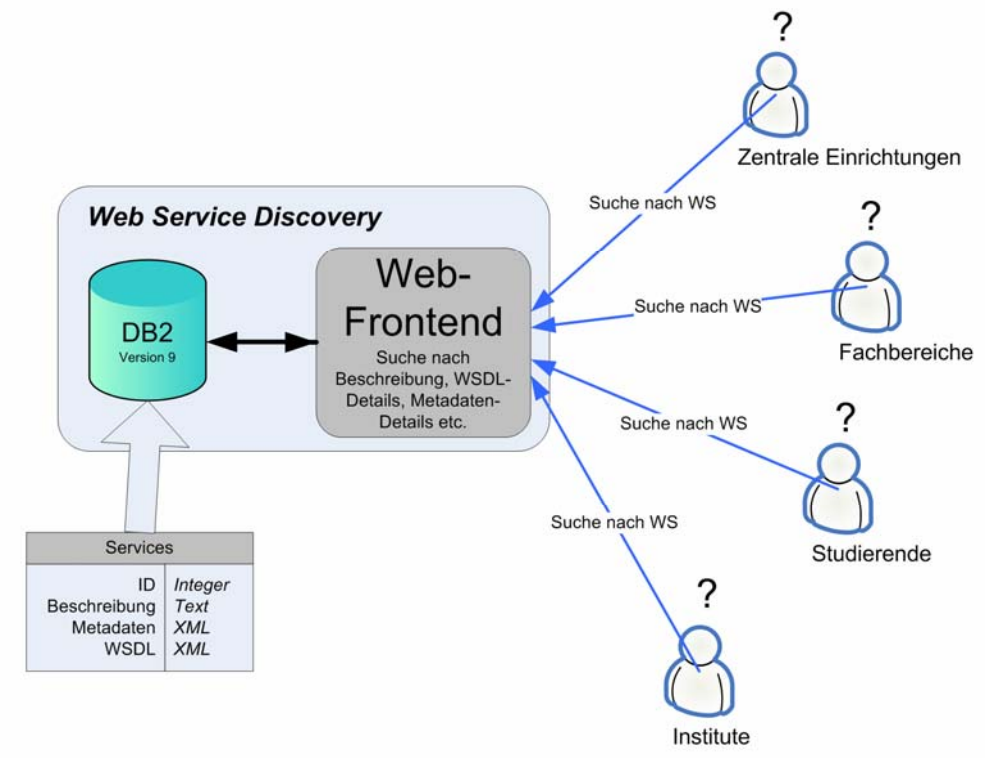

Abbildung 2: Einsatzszenario der DB2 als UDDI-Ersatz.

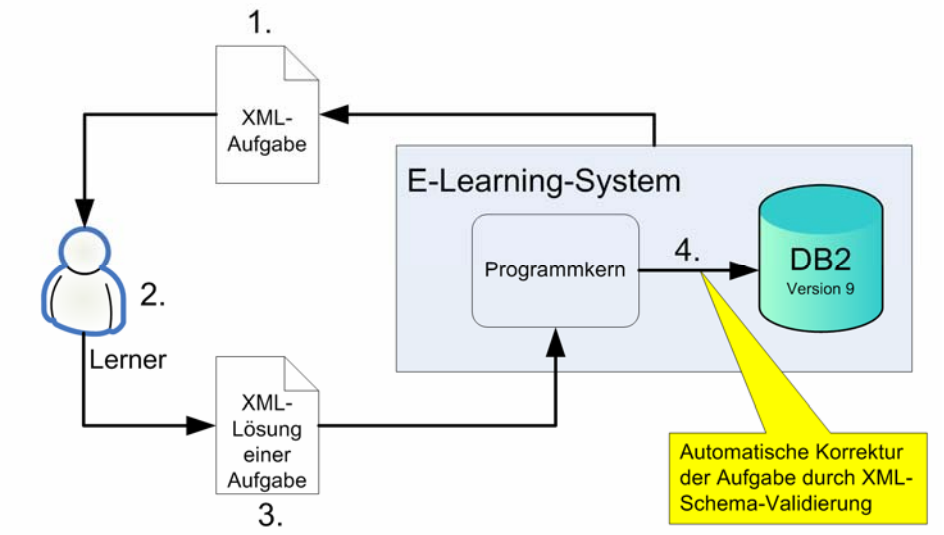

Abbildung 3: Einsatzszenario für automatische Korrektur von XML-Aufgaben.

Im 1. Schritt wird dem Lernenden eine Aufgabe (XML-Aufgabe) gestellt. Daraufhin erfolgt im zweiten Schritt die Bearbeitung der Aufgabe. Die Lösung des Lernenden wird in Teilschritt 3 an das E-Learning-System weitergegeben. Hier wird es zwischengespeichert werden, bis der Bearbeitungszeitraum der Aufgabe abläuft. Schließlich wird die abgegebene Lösung im 4. Schritt in

der DB2 (bspw. in der Tabelle Lösungen) unter Verwendung eines XML-Schemas gespeichert. Lösungen, die nicht validiert werden können, fallen automatisch heraus, da sie nicht gespeichert werden. So hat man die einfachste Form der automatischen Kontrolle.

4 Fazit

Der Einsatz neuer Datenbankfeatures für operationale Systeme in Unternehmen läuft meist sehr langsam an.[10] Der alte Wahlspruch der IT: „Never change a running system“ gilt wohl zu Recht auch hier. Es ist sehr schwer, verlässliche Informationen zur Verwendung von „pureXML“-Features in der Industrie und der Wirtschaft zu bekommen, um die Verbreitung dieser Technologie einzuschätzen. Dies liegt wohl auch daran, dass die Unterstützung von nativen XML-Funktionen erst seit kurzem in den aktuellen Releases der Datenbankgrößen zu finden sind. Klar ist, dass neben IBM auch Oracle in der neuesten Version ihrer Datenbank Oracle 10g die native XML-Verarbeitung eingebaut hat. Wahrscheinlich ist es nun nur eine Frage der Zeit, bis die ersten Unternehmen ernsthaft mit der Funktionalität arbeiten werden, die die neuen Versionen der größten kommerziellen Datenbanken bieten.

Die in Kapitel 3 angesprochenen Szenarien zeigen einige wenige Möglichkeiten der neuen XML-Features der DB2-Datenbank und insbesondere für den Bereich der Service Orientierten Architekturen und Web Services sollten diese Erweiterung mit Sicherheit eine Erleichterung für die Verarbeitung von XML-Daten schaffen, die es zu Nutzen gilt.

Literaturverzeichnis

[CS06] Chamberlin, D.; Saracco, C. M.: *Query DB2 XML data with XQuery*, IBM Developer Works (Website), IBM, April 2006.

[KM03] Klettke, M.; Meyer, H.: XML & Datenbanken, dpunkt.verlag, Heidelberg, 2003.

[MI02] Mintert, S.: XML & Co, Addison-Wesley, München, 2002.

[NL05] Nicola, M.; van der Linden, B.: *Native XML Support in DB2 Universal Database*, 31st International Conference on Very Large Data Bases 2005 (VLDB2005), Trondheim (Norway), September 2005.

[NO06] North, K.: *XML in DB2 9*, DB2 Magazine, 2006 (3rd Quarter).

[SA06a] Saracco, C. M.: Query *DB2 XML data with SQL*, IBM Developer Works (Website), IBM, März 2006.

[SA06b] Saracco, C. M.: *Get off to a fast start with DB2 Viper*, IBM Developer Works (Website), IBM, March 2006.

[SA06c] Saracco, C. M.: *Opening the door to XML*, DB2 Magazine, 2006 (3rd Quarter).

[SA06] Sanders, R. E.: *Distributed DBA*, DB2 Magazine, 2006 (3rd Quarter).

[ST05] Saake, G.; Türker C.: Objektrelationale Datenbanken. Ein Lehrbuch, dpunkt Verlag, 2005

[TUE03] Türker, C.: *SQL:1999 & SQL:2003*, dpunkt Verlag, 2003.

[WH06] Whitney, J.: *Eye on XQuery*, DB2 Magazine, 2006 (3rd Quarter).

[10] Dies ergaben u.a. Nachfragen bei ehemaligen Institutsmitarbeitern, die mittlerweile in Beratungsunternehmen der IT-Branche arbeiten.