# Generierung und Anfragebearbeitung von hierarchischen XML-Sichten auf relationale Datenbanken

*Sascha Klopp, Udo W. Lipeck*
*Institut für Informationssysteme, Universität Hannover*
*Lange Laube 22, 30159 Hannover*
{skl|ul}@dbs.uni-hannover.de

**Abstract:** In diesem Beitrag wird beschrieben, wie aus einem relationalen Datenbankschema äquivalente XML-Dokumentbeschreibungen in XMLSchema generiert werden können, die implizite hierarchische Strukturen aus der Datenbank möglichst explizit darstellen. Dabei wird ausgenutzt, dass Primär- und Fremdschlüssel sowie Eindeutigkeits- und Nullwert-Bedingungen in XMLSchema übernommen und dass Datentypen des Datenbankschemas auf entsprechende XML-Datentypen abgebildet werden können.

Die Schemaerzeugung geschieht in zwei Schritten: Nach der Überführung jedes Relationenschemas in eine flache XML-Struktur werden mit Hilfe der Fremdschlüssel Hierarchien zwischen den Relationen erkannt bzw. selektiert und ein baumartiges XML-Schema erzeugt.

Die generierten XML-Schemata werden als virtuelle Sichten auf die Datenbank aufgefasst, aus der mit Hilfe von XQuery-Anfragen Auszüge geholt werden können. Dazu wird ein neuer Ansatz vorgestellt, (eingeschränkte) XQuery-Anfragen durch Spezialisierung einer geschachtelten Gesamtanfrage in SQL zu übersetzen.

## 1 Motivation

Frühere Arbeiten, die die Erzeugung vom XML-formatierten Daten aus einer relationalen Datenbank zum Thema haben, benutzen zur Darstellung des XML-Formats meist noch die alten DTDs (Document Type Definitions). Dies hatte zur Folge, dass wegen der mangelhaften Ausdrucksfähigkeit von DTDs nicht viel Wert auf eine verlustfreie Schemaübersetzung gelegt werden konnte. Ein Ziel dieses Beitrags ist es, das vorhandene Datenbankschema so verlustlos wie möglich in ein XML-Schema zu übersetzen, was durch die neuen Möglichkeiten von XMLSchema stark unterstützt wird. Es wird ausgenutzt, dass Primär- und Fremdschlüssel sowie Eindeutigkeits- und Nullwert-Bedingungen übernommen und Datentypen des Datenbankschemas auf entsprechende XML-Datentypen abgebildet werden können.

Außerdem sollen die in XML möglichen und erwünschten hierarchischen Strukturierungen ausgenutzt werden, um hierarchische Strukturen, die in einer relationalen Datenbank nur implizit repräsentiert sind, möglichst explizit darzustellen. Das XML-Schema wird

weitgehend automatisch generiert: Nach der Überführung jedes Relationenschemas in eine flache XML-Struktur werden mit Hilfe der Fremdschlüssel Beziehungen zwischen den Relationen dem Benutzer als Graph angeboten, durch den Benutzer auf eine oder mehrere Hierarchien (Bäume) reduziert und daraus ein hierarchisches XML-Schema erzeugt.

Das generierte XML-Schema kann als virtuelle Sicht auf die Datenbank aufgefasst werden, so dass sich die intern relational organisierte Datenbank nach aussen wie eine XML-Datenbank darbietet. Aus dieser können mit Hilfe von XQuery-Anfragen Auszüge geholt werden können. Für die dazu nötige Übersetzung von (eingeschränkten) XQuery- in SQL-Anfragen wird ein neuer Ansatz vorgestellt, der eine geschachtelte Gesamtanfrage schrittweise entlang der Anfragepfade spezialisiert.

Abb. 1 zeigt einen Überblick über die Struktur unseres Systems. Die Gesamtstruktur folgt

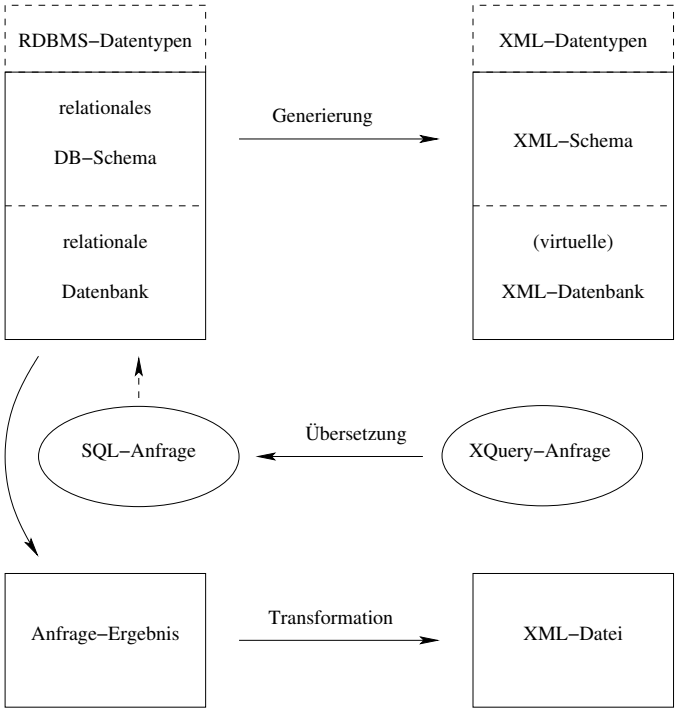


Abbildung 1: Überblick

vor allem dem System SilkRoute [FTS00], [FMST01], das dabei hilft, relationale Datenbankinhalte nach XML zu übersetzen. Dazu wird eine sogenannte *View Query* erstellt, die eine XML-Sicht definiert. An diese *View Query* kann der Benutzer Anfragen, *User Queries*, stellen. Diese beiden Anfragen werden kombiniert und in ein SQL-Anfrageprogramm übersetzt. Nach der Ausführung dieser Anfragen konvertiert ein XML-Tagger das Ergebnis in XML-Daten.

SilkRoute enthält keine Ansätze, das XML-Schema automatisch aus dem Datenbanksche-

ma zu erzeugen, dafür ist es sehr flexibel darin, die View Queries an vorhandene DTDs anzupassen.

In [FMS01] wurde genauer untersucht, auf welche Art am effektivsten das Anfrageergebnis erhalten werden kann. Es wurde zwischen den Extremen unterschieden, die Anfrage in eine einzige große SQL-Anfrage zu übersetzen oder für jedes verschiedene XML-Element, für das Daten aus der Datenbank eingelesen werden müssen, eine eigene Anfrage zu generieren und die Ergebnisse extern zu verknüpfen, und schließlich ein Mittelweg empfohlen.

Unsere Übersetzung nutzt neue Schachtelungsmöglichkeiten für SQL-Abfragen aus, wie sie von objekt-relationalen DBMS angeboten werden. Der Ansatz vereinfacht die Übersetzung von XQuery-Anfragen nach SQL, es wird allerdings ein Performance-Verlust in Kauf genommen.

[MFK01] behandelt die Übersetzung von *Quilt* [RCF00], einem Vorläufer von XQuery, in SQL-Anfragen in drei Phasen: Zunächst wird die Anfrage *normalisiert*, d. h. die vielen möglichen XML-Anfrage-Konstruktionen werden auf einen Satz einfacher Konstrukte reduziert. Auf diesen einfachen Konstrukten ist im zweiten Schritt eine Übersetzung in ein SQL-Anfrageprogramm möglich. Dieses Anfrageprogramm operiert auf einem generischen relationalen Schema, das aus der XML-Anfrage abgeleitet wird. Im letzten Schritt wird das Anfrageprogramm umgeschrieben auf das tatsächlich vorhandene Datenbankschema.

In unserem System wird die XML-Anfrage direkt in eine Anfrage an die Datenbank übersetzt, was durch die vorangehende automatische Generierung des XML-Schemas aus dem Datenbankschema ermöglicht wird.

Zur Abbildung von DB-Schemata in XML-Schemata sind in [KKRSR00] die Strukturelemente von XML-DTDs und relationalen Datenbankschemata verglichen worden und es wurde eine übergeordnete Schemabeschreibung vorgestellt. Der wesentliche Unterschied des XML-Datenmodells zu dem von RDBSen ist, dass XML-Elemente wiederum XML-Elemente enthalten können, also ein Strukturelement in einem gleichartigen Strukturelement liegen kann. Dies ist in RDBSen nicht möglich. Solche ist-Teil-von-Beziehungen sind hier nur mit Hilfe von Fremdschlüsseln möglich, also mit Hilfe der Schemabeschreibung. Allerdings wird in [KKRSR00] stark auf DTDs Bezug genommen. Die Beschreibungsmöglichkeiten von XMLSchema erlauben andere, genauere Abbildungen der Strukturelemente. Es ist jetzt möglich, ein nach den vorliegenden Regeln generiertes XMLSchema wieder in des DB-Schema zurückzuverwandeln.

Eine automatische Erzeugung von DTDs aus ER-Schemata wird in [KL01] thematisiert; dort finden sich auch Hinweise auf andere Arbeiten zur XML-Generierung aus semantischen Datenmodellen. Es wird ein Verfahren vorgestellt, um sämtliche Konzepte des ER-Modells in die DTD abzubilden: Ebenso wie die vorgenannte Arbeit baut dieses Verfahren auf DTDs auf. Ein Problem bezüglich der Integrität ist, dass `IDREF`-Attribute prinzipiell auf beliebige Elemente verweisen können, eine Typprüfung findet nicht statt. Hier ist ebenfalls nicht gewährleistet, dass die erzeugte DTD in eindeutiger Weise ein DB-Schema beschreibt.

Im vorliegenden Ansatz werden die Beschreibungsmöglichkeiten von XMLSchema [W3C01a] ausgenutzt (`key`, `keyref`), um diese Integritätsbedingungen im XML-

Dokument abzubilden.

Dieser Text behandelt entsprechend den zwei Hauptfunktionen des Systems im nächsten Abschnitt die Generierung von XML-Sichten (notiert in XMLSchema) aus relationalen Datenbanken und danach die Übersetzung von XQuery-Anfragen auf diesen generierten Sichten in die relationale Anfragesprache SQL.

## 2 XML-Generierung

Für die automatische Generierung eines XML-Schemas werden sämtliche Datenbankschema-Informationen aus dem Data Dictionary des DBMS geholt. Daher ist eine vollständige Definition der Schlüsselattribute und Fremdschlüsselbeziehungen unumgänglich. Außerdem werden die Eindeutigkeits- und Nullwert-Bedingungen berücksichtigt.

Fremdschlüssel dürfen nur auf Primärschlüssel verweisen, nicht auf andere eindeutige Spalten(gruppen), wie es z. B. in Oracle 9i möglich ist. Diese Einschränkung ergibt sich aus der entsprechenden Beschränkung von XMLSchema, wo `keyref`-Constraints sich nicht auf `unique`-Constraints beziehen können.

In allen generierten XML-Schemata wird ein sogenanntes Datenbankelement eingeführt, dass als einziges Wurzelelement auftritt. Dadurch wird gewährleistet, dass ein vollständiger Auszug ein gültiges XML-Dokument wird. Es enthält dann nämlich garantiert nur ein Element.

### 2.1 Generierung von flachen XML-Schemata

Mit den Funktionen, die XMLSchema bereitstellt, ist es möglich, beinahe ohne Informationsverlust (Datentypenumwandlung, s. u.) das Datenbankschema in ein XML-Schema umzuwandeln.

Im Folgenden wird folgendes relationales Beispielschema betrachtet:

department(dnumber, mgrssn→employee, ...)
dept_location(dlocation, dnumber→department, ...)
employee(ssn, dnumber→department, mgrssn→employee, ...)
project(pnumber, dnumber→department, ...)
works_on(essn→employee, pno→project, ...)

Zunächst wird für die gesamte Datenbank ein XML-Element (das „Datenbankelement“) erzeugt, das für jede Relation Unterelemente enthalten kann:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="company">
  <xsd:complexType>
```

```
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" name="department"
                                     type="departmentType"/>
  <xsd:element maxOccurs="unbounded" name="dept_location"
                                     type="dept_locationType"/>
  <xsd:element maxOccurs="unbounded" name="employee"
                                     type="employeeType"/>
  <xsd:element maxOccurs="unbounded" name="project"
                                     type="projectType"/>
  <xsd:element maxOccurs="unbounded" name="works_on"
                                     type="works_onType"/>
 </xsd:sequence>
</xsd:complexType>
```

Sämtliche Primärschlüssel, Fremdschlüssel und Eindeutigkeitsbedingungen werden in `key-`, `keyref-` bzw. `unique`-Constraints abgebildet. Z.B. findet sich die Referenz von employee→department wieder als[1]

```
<xsd:key name="sys_c002897">
 <xsd:selector xpath="department"/>
 <xsd:field xpath="dnumber"/>
</xsd:key>
<xsd:keyref name="sys_c002906" refer="sys_c002897">
 <xsd:selector xpath="employee"/>
 <xsd:field xpath="dnumber"/>
</xsd:keyref>
```

Für jede Relation wird für dessen Tupel ein Datentyp generiert, der für jede Spalte der Relation ein XML-Element mit einem Datentyp, der dem Typ der Spalte entspricht, enthält. Falls Nullwerte in der Spalte zugelassen sind, muss das XML-Attribut `nillable` auf `true` gesetzt werden. Diese Spalten werden als Sequenz gruppiert, auch wenn die Reihenfolge der Werte irrelevant ist. [2] Hier gehen auch die Nullwert-Bedingungen ein. Z.B.:

```
<xsd:complexType name="departmentType">
 <xsd:sequence>
  <xsd:element name="dname" type="string_15"/>
  <xsd:element name="dnumber" type="integer_1"/>
  <xsd:element name="mgrssn" type="integer_9"/>
  <xsd:element name="mgrstartdate" nillable="true"
               type="xsd:dateTime"/>
 </xsd:sequence>
</xsd:complexType>
```

[1] Die Schlüsseldefinitionen, auf die Fremdschlüssel zugreifen, müssen für diese jeweils sichtbar sein, daher passieren der Einfachheit halber alle diese Definitionen auf der Ebene des Datenbankelements.

[2] Als content model könnten die Datentypen `all` erhalten, denn ein Tupel besteht aus Spaltenwerten, die jeweils genau einmal vorkommen. Wenn allerdings später im hierarchischen Schema Unterelemente eingefügt werden, können diese mehrfach auftreten, was die `all`-Gruppe nicht erlaubt; also wird hier gleich `sequence` gewählt.

Schließlich wird für jeden vorkommenden SQL-Datentyp gemäß folgender Tabelle ein (möglichst) äquivalenter XML-Datentyp definiert:

| SQL-Datentyp: | XML-Datentyp: |
|---|---|
| VARCHAR2(n) | string(maxLength=n) |
| CHAR(n) | string(length=n) |
| NUMBER(p,s) | decimal(totalDigits=p,fractionDigits=s) |
| FLOAT(p) | decimal(totalDigits=p/$\log_2(10)$) |
| NUMBER(p) | integer(totalDigits=p) |
| DATE | dateTime |

### 2.2 Generierung von hierarchischen XML-Schemata

Durch die Fremdschlüsselbedingungen im Datenbankschema ist es danach möglich, dem XML-Schema eine Hierarchie zu geben, die die Semantik des Schemas meist besser ausdrückt. Eine baumartige Struktur nutzt die Möglichkeiten von XML aus, die Informationen zu strukturieren. Überflüssig werdende Fremdschlüssel machen zudem das XML-Schema etwas kleiner.

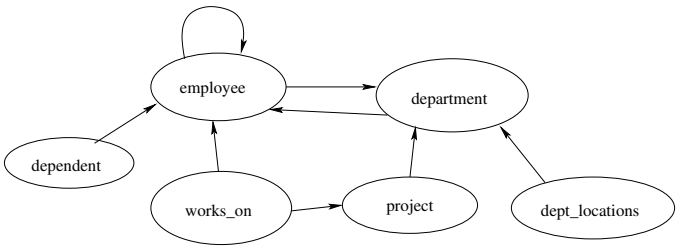


Abbildung 2: Beispiel-DB-Schema

Der in Abb. 2 abgebildete Graph stellt zum obigen Beispielschema die Fremdschlüsselbeziehungen zwischen den Relationen dar (eine Kante hat die Bedeutung „hat Fremdschlüssel auf“).

Um das Datenbankschema in ein hierarchisches XML-Schema zu übersetzen, muss es eine Baumstruktur haben. Man sieht allerdings, dass in dem Beispielschema zwei Zyklen enthalten sind. Außerdem lässt sich die DEPARTMENT-Relation auf zwei Wegen von der WORKS_ON-Relation aus erreichen. Daher können nicht alle Fremdschlüssel durch eine Baumstruktur im XML-Schema abgebildet werden. Ein Beispiel, welche der Beziehungen aufgelöst werden könnten, ist in Abb. 3 gezeigt.

Im zweiten Schritt können Elemente, die einen Fremdschlüssel auf ein anderes Element besitzen, unterhalb dieses Elements angeordnet werden. Dadurch entsteht eine Baumstruktur, s. Abb. 3. Die entsprechenden `keyref`-Definitionen aus dem XML-Schema fallen dann weg. Außerdem müssen die Pfadangaben in dem übrigen `key`-, `keyref`- und

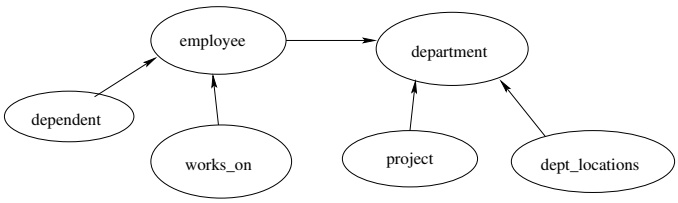


Abbildung 3: Diese Fremdschlüssel werden im Beispiel aufgelöst

`unique`-Definitionen aktualisiert werden.

Allgemein lässt sich ein Element sinnvoll einem anderen unterordnen, wenn es zum Beispiel einen Fremdschlüssel auf dieses besitzt. Allerdings kommen nur solche Fremdschlüssel für eine Hierarchiebildung in Frage, bei denen die referenzierenden Spalten nicht NULL werden können (durch NOT NULL oder weil sie Primärschlüssel sind); sonst kann es passieren, dass sich ein Tupel auf kein Tupel in der Oberrelation bezieht und infolgedessen in der hierarchischen XML-Sicht gar nicht auftaucht[3].

Falls ein Element nur einen Fremdschlüssel hat, spricht nichts dagegen, es entsprechend einzuordnen. Im vorliegenden Beispiel ist es aber möglich, das `works_on`-Element sowohl als Unterelement von `employee` als auch von `project` zu definieren. Dabei würden allerdings Redundanzen entstehen, denn ein bestimmtes `works_on`-Element tauchte dann sowohl unter seinem `project`- als auch unter seinem `employee`-Element auf. Das Schema soll allerdings später auch dazu dienen, neue XML-Daten, die in die Datenbank eingefügt werden sollen, zu überprüfen. Der Test, ob die redundanten Daten widerspruchsfrei sind, würde das XML-Schema unverhältnismässig stark vergrößern, denn es müssten weitere XML-Schlüssel- und Fremdschlüsselbeziehungen eingeführt werden.

Man muss sich in solchen Situationen also entscheiden, ob und an welcher Stelle man eine Hierarchie einführen will. Semantisch handelt es sich bei Relationen, die Fremdschlüssel auf verschiedene andere Relationen enthalten, meist um n:m-Beziehungen, die z. B. in [KL01] als Top-Level-Elemente definiert werden. Sie beinhalten nur Referenzen auf die entsprechenden Elemente, um gleichzeitig Vollständigkeit und Redundanzfreiheit zu erhalten. Im vorliegenden Ansatz kann zusätzlich eines der beteiligten Elemente direkt untergeordnet werden.

Es ist nicht sinnvoll, die Fremdschlüsselbeziehung in der anderen Richtung auszunutzen, denn dann kann zum einen ein Unterelement an mehreren Stellen auftauchen. Im Beispiel heißt das, dass unter verschiedenen Oberelementen `works_on` das gleiche `project`-Element erscheint. Zum anderen ist wiederum nicht garantiert, dass sämtliche Tupel der Unterrelation referenziert werden.

Außerdem können durch die Fremdschlüsselbeziehungen Zyklen auftreten. Das XML-Schema könnte hier wie oben beschrieben definiert werden. Wenn jedoch später die XML-Sicht materialisiert wird, könnte es passieren, dass ein Element in verschiede-

[3] Mögliche Alternativen wären, dieses in Kauf zu nehmen, oder die entsprechenden Elemente unter künstlich erzeugten Oberobjekten einzuordnen. Solche Ansätze wurden hier nicht untersucht.

nen Hierarchiestufen auftritt, so dass es ein Nachfolger von sich selbst ist. In diesem Fall würde das Generieren der Sicht nicht zu einem Ende kommen. Zur Behandlung dieses Problems gibt es mindestens zwei Ansätze: Entweder wird garantiert, dass die Datenbank eine strikte Hierarchie darstellt, oder der Zyklus wird an einer Stelle aufgetrennt, also an dieser Stelle der Fremdschlüssel nicht in eine Unterelementbeziehung umgewandelt.

Wird die erste Möglichkeit gewählt, muss noch eine Relation bestimmt werden, die die Hierarchie beginnt. Diese Relation erscheint dann als Wurzelelement des für diese Hierarchie zuständigen Teilbaums. Für diese Relation muss vermerkt werden, wie Tupel ausgewählt werden, die die Hierarchie beginnen. Bei einer Rangfolge in einer Firma zum Beispiel der Chef. Er würde dann als einziges Element als oberstes in der XML-Instanz auftreten. Alle weiteren Mitarbeiter ordneten sich dann entsprechend ihrem Rang darunter ein. Dieser Ansatz wurde hier nicht gewählt, da nur schwer zu ermitteln ist, ob alle Tupel in der XML-Instanz erscheinen werden, und auch jedes nur einmal, insbesondere, wenn an dem Zyklus mehr als eine Relation beteiligt ist.

Bei der zweiten Möglichkeit wird in Kauf genommen, dass der inhaltliche Zusammenhang der beteiligten Relationen verloren geht, bzw. nur noch im erzeugten XML-Schema erkennbar bleibt. In der XML-Instanz würden die Elemente völlig unabhängig erscheinen. Lediglich über die Fremdschlüsselspalten können die Elemente zum Beispiel in einer Anfrage zusammengeführt werden.

In sämtlichen Unterelementen können die Datenfelder gelöscht werden, die die Referenz auf ihr Oberelement beinhalten (die Fremdschlüsselspalten). Im Hinblick auf Redundanzfreiheit (wichtig für den Import) ist dies sogar notwendig, denn ihre Inhalte sind innerhalb eines Oberelementes gleich und entsprechen den Schlüsselspalten des Oberelementes.

Diese Überlegungen führen zu dem nun folgenden Algorithmus:

1. Bilde einen gerichteten Graph. Jede Relation stellt einen Knoten dar. Für jeden Fremdschlüssel, dessen Spalten sämtlich mit NOT NULL-Bedingungen ausgestattet sind, erzeuge eine Kante von der referenzierten Relation zur referenzierenden Relation.

2. Für jeden Knoten, auf den mehr als eine Kante zeigt, frage den Benutzer, welche von ihnen gelöscht werden soll, bis nur eine von ihnen übrig bleibt.

3. Markiere alle Knoten, auf die keine Kante zeigt. Jeder dieser markierten Knoten gehört zu einer anderen Zusammenhangskomponente des Graphen. Jede dieser Komponente mit einem markierten Knoten ist ein Baum, die Wurzel ist der markierte Knoten. Es kann jedoch noch Komponenten wie in Abb. 4 geben, die also einen Zyklus enthalten. Eine Komponente kann nur einen Zyklus enthalten, da auf jeden Knoten nur höchstens eine Kante zeigt. Verschiedene Zyklen gehören also zu verschiedenen Zusammenhangskomponeneten.

4. Aus der Menge der Kanten, die nicht zu einem der soeben erzeugten Bäume gehören, also zu Komponenten mit einem Zyklus, entferne nach Auswahl durch den Benutzer eine Kante, die zu eben diesem Zyklus gehört. Damit ist der Zyklus aufgelöst und der Knoten, auf den die soeben entfernte Kante gezeigt hat, kann ebenfalls als Wurzelknoten gekennzeichnet werden.

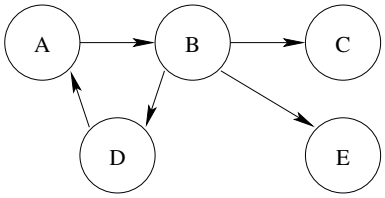


Abbildung 4: A, B und D sind potentielle Wurzeln

Führe diesen Schritt aus, bis alle Relationen in Bäumen enthalten sind, es also keine Zyklen mehr gibt.

5. In diesem Schritt werden die Unterelemente in ihre Oberelemente eingefügt. Beginnend mit den untersten Elementen wird folgendes getan: In das zugehörige Oberelement wird eine Elementdeklaration für das aktuelle Element eingefügt. Aus dem aktuellen Element werden die Fremdschlüsselspalten entfernt, die sich auf das Oberelement beziehen, da sie stets mit den entsprechenden Spalten des Oberelements übereinstimmen. Zum Schluss kann das Element aus dem Baum entfernt werden.

   Übrig bleiben die Elemente, die kein Oberelement haben, also die Wurzeln der Bäume. Nur diese dürfen als Elemente im Datenbankelement des XML-Schemas erhalten bleiben.

6. Entferne alle `keyref`-Bedingungen, die zu Fremdschlüsselbedingungen gehören, deren Kanten nicht gelöscht wurden.

7. Passe die Pfadangaben in den restlichen `keyref`- und den `key`- und `unique`-Bedingungen an. Im `selector`-Element muss der vollständige Pfad zu dem Element erscheinen. Falls im vorletzten Schritt ein Spaltenelement gelöscht wurde, das in einer der `field`-Felder der Bedingungen noch referenziert wird, muss diese Pfadangabe unter Benutzung der `../`-Konstruktion durch eine Referenz auf das zugehörige, weiter oben liegende Element ersetzt werden.

Im Beispiel wird dann u.a. erzeugt:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="company">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element maxOccurs="unbounded" name="department"
                                       type="departmentType"/>
   </xsd:sequence>
  </xsd:complexType>
  ......
 </xsd:element>
 <xsd:complexType name="departmentType">
```

```
  <xsd:sequence>
   <xsd:element name="dnumber" type="integer_1"/>
   ......
   <xsd:element maxOccurs="unbounded" name="employee"
                                      type="employeeType"/>
   <xsd:element maxOccurs="unbounded" name="dept_locations"
                                      type="dept_locType"/>
   <xsd:element maxOccurs="unbounded" name="project"
                                      type="projectType"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="employeeType">
  <xsd:sequence>
   ......
   <xsd:element maxOccurs="unbounded" name="works_on"
                                      type="works_onType"/>
   <xsd:element maxOccurs="unbounded" name="dependent"
                                      type="dependentType"/>
  </xsd:sequence>
 </xsd:complexType>
 ......
</xsd:schema>
```

Das so entstandene hierarchische XML-Schema und das vorherige flache Schema beschreiben Mengen von Instanzen, die sich bijektiv einander zuordnen lassen (und auch auseinander rekonstruieren lassen); die Schemata sind also äquivalent. Das letztere Schema nutzt aber die Baumstruktur von XML aus.

## 3 Anfragebearbeitung

Das erzeugte XML-Schema liefert eine hierarchische Sicht auf die Datenbank. Damit sollte dieses Schema auch als Ausgangspunkt für Anfragen dienen können, etwa in der aktuellen XML-Anfragesprache XQuery.

Angenommen, man möchte für jeden Angestellten sämtliche Daten derjenigen Projekte, an denen er mitarbeitet, ausgeben. Dazu kann man folgende XQuery-Anfrage formulieren:

```
<employees>{
 for $e in dbview('company.xsd')/company/department/employee
 let $p := dbview('company.xsd')/company/department/
                                project[pnumber=$e/works_on/pno]
 return
  <emp_projects>{
   $e/name,$p
  }</emp_projects>
}</employees>
```

Diese Anfrage muss in eine SQL-Anfrage übersetzt werden, um die Inhalte aus der Datenbank zu holen. Alle Informationen über die Struktur der XML-Sicht sind im XML-Schema enthalten, daher wird zur Übersetzung auch nur dieses benötigt.

Schließlich soll das Anfrageergebnis XML-formatiert ausgegeben werden:

```
<employees>
 <emp_projects>
  <name>Johnny Walker</name>
  <project>
   <pnumber>42</pnumber>
   <budget>12345</budget>
   <pname>Reorganisation</pname>
   <plocation>Houston</plocation>
  </project>
  <project>
   ....
  </project>
 </emp_projects>
 <emp_projects>
  ....
 </emp_projects>
</employees>
```

Der Prozess erfolgt also in drei getrennten Schritten: Die Erzeugung der SQL-Anfrage, die Auswertung mit Hilfe des DBMS und die Herstellung der XML-Datei.

Eine XQuery-Anfrage wird hier in eine einzige SQL-Anfrage übersetzt. Dies hat den Vorteil, das Zwischenwerte nicht zwischen Programm und DBMS ausgetauscht werden müssen. Dies wäre der Fall, wollte man Joins vom externen Programm ausführen lassen. Die Anfrage unterscheidet sich allerdings von den in [FMS01] erwähnten Anfragen, die auf sortierten Outer Joins aufbauen, denn das Anfrageergebnis enthält keine redundanten Daten, sondern besteht aus verschachtelten Tabellen.

Als Startpunkt für die Übersetzung dient eine SQL-Anfrage, die die gesamte XML-Sicht materialisieren könnte. Pfadausdrücke in der XQuery-Anfrage reduzieren diese Anfrage auf die notwendigen Teile, und Konstruktoren und FLWR-Ausdrücke gruppieren diese Teile neu, so dass das gewünschte Ergebnis erhalten wird.

Die Gesamtauszugs-Anfrage für das Beispielschema sieht so aus:

```
select cursor(
  select cursor(
    select
      to_char(DNAME) dname_out_,
      to_char(DNUMBER) dnumber_out_,
      [...]
      cursor(select to_char(NAME) name_out_,
                    to_char(SSN) ssn_out_,
                    [...]
```

```
                    to_char(SUPERSSN) superssn_out_,
                    cursor(select to_char(PNO) pno_out_,
                                  to_char(HOURS) hours_out_
                           from WORKS_ON works_on
                           where (ESSN) in (employee.SSN))works_on_out_,
             from EMPLOYEE employee
             where (DNUMBER) in (department.DNUMBER))employee_out_,
      cursor(select to_char(DLOCATION) dlocation_out_
             from DEPT_LOCATIONS dept_locations
             where (DNUMBER) in (department.DNUMBER))dept_locations_out_,
      cursor(select to_char(PNAME) pname_out_,
                    to_char(PNUMBER) pnumber_out_,
                    [...]
             from PROJECT project
             where (DNUMBER) in (department.DNUMBER))project_out_
  from DEPARTMENT department) department_out_
from dual  )company_out_ from dual
```

Zur Erzeugung der Untertabellen im Anfrageergebnis wird reger Gebrauch von der CURSOR-Funktion gemacht, die eine spezielle Konstruktion von Oracle ist. Für jedes einzelne Ausgabetupel, in dem ein Cursor auftritt, wird die entsprechende Anfrage ausgewertet. Außerhalb definierte Aliase sind dabei sichtbar, so dass diese wie Parameter der Anfrage wirken.

Das Verfahren, das jetzt aus dieser Anfrage und der XQuery-Anfrage die passende SQL-Anfrage erzeugt, ist in [Klo01] genauer beschrieben. Dort sind für eine gegenüber [W3C01b] eingeschränkte Menge von XQuery-Konstrukten Regeln angegeben.

Die Übersetzung der Beispielanfrage sieht so aus:

```
select cursor(
  select cursor(
    select cursor(
      select cursor(
        select cursor(
          select cursor(
            select cursor(
              select to_char(NAME) name_out_ from dual),
                  cursor(
                    select cursor(
                       select to_char(PNAME) pname_out_,
                              to_char(PNUMBER) pnumber_out_,
                              [...]
                       from PROJECT project
                       where (DNUMBER) in (department.DNUMBER)
                             and (pnumber IN (select to_char(PNO) pno_out_
                                              from SKL.WORKS_ON works_on
                                              where (ESSN) in (e.SSN))))project_out_
                       from DEPARTMENT department) from dual)
             from dual) emp_projects_out_ from dual)
       from (select *
             from EMPLOYEE employee,
                  (select *
                   from DEPARTMENT department) department
             where (employee.DNUMBER) in (department.DNUMBER)) e) from dual)
  from dual)employees_out_ from dual
```

Die starke Verschachtelung ist für eine einfache Umwandlung des Ergebnisses in das XML-Dokument notwendig. Dafür werden Spaltenbezeichnungen, die auf `_out_` enden,

bei der Ausgabe in das XML-Ergebnis berücksichtigt.

## 4 Ausblick

Im vorgestellten Ansatz werden bisher im Wesentlichen die skalaren Standard-SQL-Datentypen unterstützt. Weitere Entwicklungsarbeit ist notwendig, um auch strukturierte Datentypen (Arrays, Nested Tables, benutzerdefiniert), wie sie durch objekt-relationale Datenbanksysteme unterstützt werden, im XML-Ergebnis zu erhalten.

Eine weitere Anforderung an einen effektiven XML-basierten Datenaustausch sind Updates der Datenbank mit Hilfe von XML-Dokumenten. Die Möglichkeiten von XMLSchema, Schlüsselbedingungen, etc. zu definieren, werden hier eine große Erleichterung sein, denn dadurch kann ein großer Teil der Integritätssicherung durch Standard-Parser erledigt werden.

Schließlich bleibt für eine effiziente Bearbeitung von Anfragen an generierte XML-Sichten zu untersuchen, welche Optimierungen in der unterliegenden relationalen Datenbank möglich sind, bzw. ob mit vertretbarem Aufwand eine Übersetzung der XQuery-Anfragen möglich ist, die besser von bestehenden DBMSen optimiert wird.

## Literatur


[FMS01] M. Fernández, A. Morishima, D. Suciu: Efficient Evaluation of XML Middle-ware Queries. In T. Sellis, S. Mehrotra (eds.), *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Dallas, May 21-24, 2001*, SIGMOD Record 2, ACM Press, New York, 2001, 103–114.

[FMST01] M. Fernández, A. Morishima, D. Suciu, W.-C. Tan: Publishing Relational Data in XML: the SilkRoute Approach. *Data Engineering 24:2 (2001)*, 12–19.

[FTS00] M. Fernández, W. C. Tan, D. Suciu: SilkRoute: Trading between Relations and XML. *Computer Networks 33:1-6 (2000)*, 723–745. Proceedings of the 9th International World Wide Web Conference.

[KKRSR00] G. Kappel, E. Kapsammer, S. Rausch-Schott, W. Retschitzegger: X-Ray - Towards Integrating XML and Relational Database Systems. In A. Laender, S. Liddle, V. Storey (eds.), *Conceptual Modeling - ER 2000 - 19th Int. Conference on Conceptual Modeling, Salt Lake City, Utah, USA, October 9-12, 2000*, LNCS 1920, Springer-Verlag, Berlin, 2000, 339–353.

[KL01] C. Kleiner, U. W. Lipeck: Automatic Generation of XML DTDs from Conceptual Database Schemas. In K. Bauknecht, W. Brauer, T. Mück (eds.), *Informatik 2001 – Wirtschaft und Wissenschaft in der Network Economy - Visionen und Wirklichkeit / GI/OCG-Jahrestagung Sept. 2001, Universität Wien - Band I*, Oesterreichische Computer Gesellschaft, Wien, 2001, 396–405.

[Klo01] S. Klopp: Ableitung von XML-formatierten Daten aus objekt-relationalen Datenbanken. Master's thesis, Institut für Informatik, Universität Hannover, Hannover, 2001.

[MFK01] I. Manolescu, D. Florescu, D. Kossman: Pushing XML Queries inside Relational Databases. Technical report, Unité de recherche INRIA Rocquencourt, 2001.

[RCF00] J. Robie, D. Chamberlin, D. Florescu. Quilt: An XML Query Language. 2000. verfügbar im WWW:
http://www.almaden.ibm.com/cs/people/chamberlin/quilt_euro.html

[W3C01a] W3Consortium. XMLSchema Part 0: Primer. World Wide Web Consortium, 2001. W3C Recommendation 2-May-2001.

[W3C01b] W3Consortium. XQuery 1.0: An XML Query Language. World Wide Web Consortium, 2001. W3C Working Draft 20-Dec-2001.