# Transformation von SQL in XQuery-Anfragen innerhalb föderierter Informationssysteme

Heiko Jahnkuhn, Ilvio Bruder, Ammar S. Balouch
Institut für Informatik, Universität Rostock
E-Mail: {hj016,ilr,ab006}@informatik.uni-rostock.de

In föderierten Informationssystemen steht man vor dem Problem, Daten und Informationen, die in verschiedenen Formaten und Systemen gespeichert sind, auswerten zu müssen. Die Daten können dabei in relationalen, objektrelationalen oder auch XML-Datenbanken vorliegen. Dazu werden Anfrage-Transformationen benötigt, aber auch die Transformation von Daten und Ergebnissen in andere Formate ist erforderlich. Diese Arbeit beschäftigt sich mit der Anfragetransformation von SQL in XQuery innerhalb föderierter Informationssysteme. Der in dieser Arbeit entwickelte allgemeingültige Algorithmus führt eine solche Transformation unter bestimmten Vorbedingungen automatisch durch.

## 1 Einführung

Ein grundlegendes Prinzip der föderierten Datenbank- und Informationssysteme besteht darin, dass eventuell mehrere unterschiedliche Datenmodelle in die Föderation eingehen. Somit besteht die Möglichkeit, dass Föderierungsdienst und Teilnehmer der Föderation dementsprechend heterogen sein können. Wenn eine Anfrage in einer konkreten Anfragesprache gestellt wird, aber auch von anderen Datenbanksystemen mit anderen Anfragesprachen ausgewertet werden soll, muss eine Transformation dieser Anfrage erfolgen. Dabei sind für diese Arbeit prinzipiell zwei Szenarien denkbar. Im ersten Fall nimmt ein objektrelationales Datenbanksystem an einer Föderation teil, dessen Datenmodell auf XML beruht. Somit ist die Anfragesprache des Föderierungsdienstes XQuery. Wenn nun der Teilnehmer eine Anfrage bezüglich seines lokalen Schemas stellt, welche aber von der Föderation ausgewertet werden soll, muss diese Anfrage derart transformiert werden, dass sie in XQuery vorliegt und sich auf das globale Schema bezieht. Im zweiten Fall, auf den sich der hier vorgestellte Algorithmus auch bezieht, nutzt der Föderierungsdienst ein objektrelationales Datenmodell, auf das Anfragen mittels SQL gestellt werden. Wird nun eine SQL-Anfrage bezüglich des globalen Schemas gestellt, muss diese für jeden Teilnehmer der Föderation transformiert werden. Für eine XML-Datenbank muss somit die Anfrage in XQuery transformiert werden, welche eine Anfrage auf ein XML-Dokument oder eine XML-Dokumentenkollektion beschreibt. Nutzt man für eine derartige Transformation beispielsweise ein regelbasiertes System, besteht der Nachteil darin, dass ein solches System für jeden XML-Teilnehmer neu entworfen werden muss. Nutzt dieser Teilnehmer nicht nur eine sondern eine Vielzahl unterschiedlicher Schemabeschreibungen, steigt der Entwurfsaufwand sogar linear mit der Anzahl der Schemabeschreibungen, die der Föderation bereitgestellt werden. Um diesem Effekt entgegenzuwirken wird hier ein Algorithmus vorgestellt, welcher eine solche Transformation automatisch auch bezüglich verschiedener Schemabeschreibungen durchführen kann. An dieser Stelle wird allerdings nur der grundlegende Algorithmus vorgestellt, der sich auf einfache SQL-Anfragen beschränkt. Einfach bedeutet an dieser Stelle, dass für die Where- und Having-Klausel lediglich elementare Vergleichsoperatoren wie <, > und = erlaubt sind, welche mit den logischen Operatoren AND, OR und NOT verknüpft werden können. In der Having-Klausel sind weiterhin Aggregatfunktionen möglich. Auf die Umsetzung weiterer Konstrukte, die SQL bietet, sowie weitere Phasen und auftretende Probleme der Anfrageverarbeitung in föderierten Informationssystemen wird in [Jah05] genauer eingegangen.

## 2 Anfragetransformation

An dieser Stelle wird die Anfragetransformation bezüglich einer bestimmten Schemabeschreibung (DTD oder XML Schema) durchgeführt. Das bedeutet, dass die resultierende XQuery-Anfrage an eine

Dokumentenkollektion gestellt wird, die bezüglich dieser Schemabeschreibung gültig ist. Auf den allgemeinen Fall, dass einer XML-Datenbank mehrere Schemabeschreibungen zugrunde liegen, wird ebenfalls in [JBB05] eingegangen. Es existiert somit eine injektive Abbildung der Attribute des globalen Schemas auf eine Menge von Pfadausdrücken bezüglich des lokalen Schemas, welches eine Schemabeschreibung des XML-Komponentendatenbanksystems. Der Algorithmus zur Transformation läuft in vier separaten Schritten ab, die nachfolgend erklärt werden sollen.

**Schritt 1**
Zu Beginn wird die globale Anfrage syntaktisch analysiert. Sie wird in die Fragmente der SELECT-, FROM, WHERE-, GROUP BY-, HAVING- und ORDER BY-Klausel aufgeteilt. Unter Nutzung einer Zuordnungstabelle von globalen Attributen auf lokale Pfadausdrücke wird eine Liste sämtlicher Pfadausdrücke erstellt, welche in der globalen SQL-Anfrage nach Substitution der globalen Attribute auftreten. Diese Liste beschreibt den Strukturbaum der zugrunde liegenden Anfrage, welcher einen Teilbaum der zugrunde liegenden Schemabeschreibung darstellt. Darauf hin wird die WHERE-Klausel derart analysiert, dass sie in eine Liste von Konjunktionstermen aufgespaltet wird. Besteht die WHERE-Klausel also aus dem Term A AND B, wobei A und B wiederum Terme sind, wird dieser Term aus der Liste gelöscht und A und B separat in diese Liste aufgenommen. Anschließend werden A und B auf dieselbe Art untersucht, so dass die Liste am Ende nur atomare Vergleiche oder Disjunktionen enthält. Anschließend wird für jedes Element dieser Liste der größte gemeinsame Pfad (gcp – greatest common path) ermittelt, der im Term $T_i$ nach Substitution der globalen Attribute enthalten ist. Somit ist jedem Term $T_i$ ein gcp($T_i$) zugeordnet. Im letzten Teil dieses Transformationsschrittes wird der ermittelte Strukturbaum der Anfrage mit drei Knotenarten markiert, den Where-Elementen, den Return-Elementen und den Splitting-Knoten. Die Where-Elemente repräsentieren dabei die ermittelten gcp($T_i$). Die Return-Elemente ergeben sich aus den Attributen der SELECT-, GROUP BY-, HAVING- und ORDER BY-Klausel. Die Splitting-Knoten lassen sich durch die konkreten Beziehungen zwischen den WHERE- und RETURN-Elementen ermitteln. Abbildung 1 zeigt dabei die möglichen Beziehungsklassen, aus denen sich Splitting-Knoten ermitteln lassen. XML-Attribute werden hierbei ebenfalls als XML-Elemente angesehen. Alle anderen Beziehungen lassen

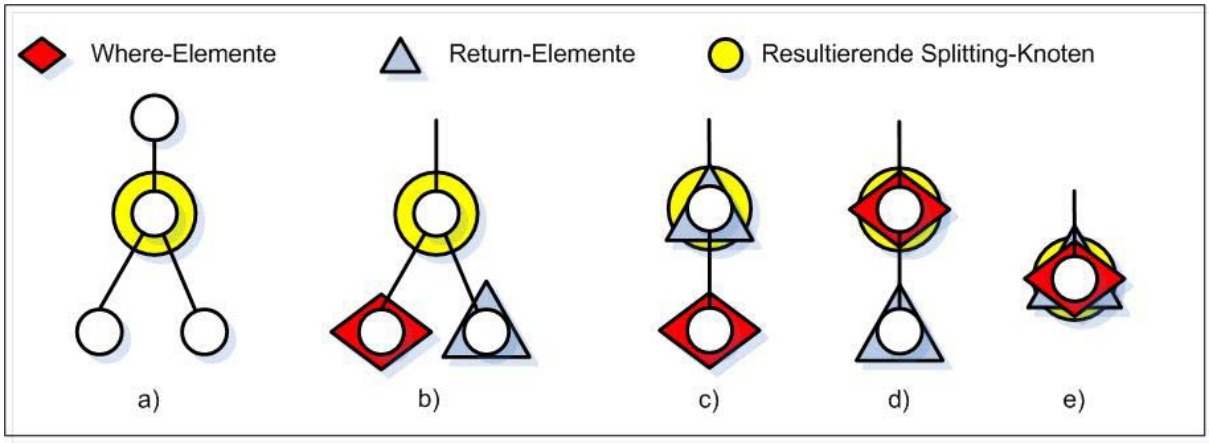


**Abb. 1:** Beziehungsklassen für Splitting-Knoten

sich durch diese Klassen darstellen. Die Linien repräsentieren dabei unbekannte Pfade, auf denen keine markierten Knoten liegen. Die Bedeutung der einzelnen Klassen lautet wie folgt, wobei SK für den resultierenden Splitting-Knoten steht:

a) SK ist der erste Knoten, der mehr als ein Kindelement hat.
b) SK hat sowohl Where- als auch Return-Elemente in verschiedenen descendent-Achsen.
c) SK ist ein Return-Element und hat ein Where-Element in einer descendent-Achse.
d) SK ist ein Where-Element und hat ein Return-Element in einer descendent-Achse.
e) SK ist sowohl Where- als auch Return-Element.

**Schritt 2**
Im zweiten Transformationsschritt wird aus dem markierten Strukturbaum ein XQuery-Ausdruck hergeleitet, welcher diesen repräsentiert. Jeder Splitting-Knoten des Strukturbaums entspricht dabei einem zu generierenden FLWR-Ausdruck, wobei die Hierarchie erhalten bleibt. Somit wird ein Top-Down-Verfahren eingesetzt, welches ausgehend von der Wurzel des Baumes den ersten Splitting-Knoten ermittelt. Für diesen Splitting-Knoten wird ein FLWR-Ausdruck erzeugt, der den Splitting-Knoten als Kontextknoten in der For-Klausel referenziert. Die Where-Klausel wird durch die Konjunktion sämtlicher Where-Elemente gebildet, die in der self-or-descendent-Achse des Kontextknotens liegen. Für die Erzeugung der Return-Klausel werden drei Fälle unterschieden:

1. Ist der Splitting-Knoten auch ein Return-Element wird der Kontextknoten direkt übernommen.
2. Für jeden Folgepfad, der ausschließlich Return-Elemente enthält, wird der Pfad dieser Return-Elemente übernommen.
3. Für jeden Folgepfad, der Splitting-Knoten enthält, wird dieses Verfahren rekursiv angewandt, wobei der jeweils neue Kontextknoten immer relativ zum übergeordneten Kontextknoten gesetzt wird.

Im ersten und zweiten Fall werden die Return-Elemente zusätzlich mit Tags umschlossen, deren Name sich aus dem korrespondierenden globalen Attributnamen des jeweiligen Pfadausdrucks ergibt. Außerdem wird der Inhalt der ersten Return-Klausel mit zwei tuple-Tags umschlossen, so dass das

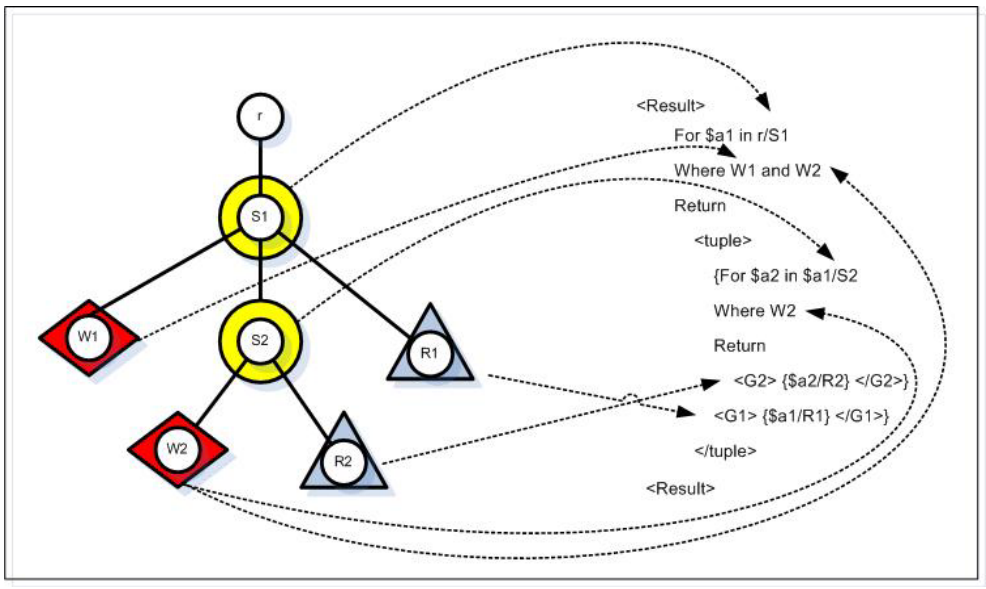


**Abb. 2:** Abbildung des Strukturbaumes auf XQuery-Anfrage

Gesamtergebnis als SET OF TUPLE OF dargestellt wird. In Abbildung 2 wird versucht diesen Schritt graphisch darzustellen. Die Ausgabe dieses zweiten Schrittes realisiert interessanterweise bereits eine relationale Darstellung des Strukturbaums, da die erzeugte Struktur gültig bezüglich folgender generischer DTD wäre:

<! ELEMENT Result (tuple*)>
<! ELEMENT tuple = any-node*>
(any-node bezeichnet hier ein beliebiges Element des Anfragebaums).

**Schritt 3**
Im dritten Schritt folgt die Realisierung der GROUP BY- und HAVING-Klausel, sofern diese in der globalen Anfrage verwendet wurden. Diese beiden Klauseln werden in diesem Schritt jedoch separat bearbeitet, da unter Umständen trotz GROUP BY- keine HAVING-Klausel existiert. Das Grundprinzip der Gruppierung basiert wiederum auf einem FLWR-Ausdruck. Die For-Klausel besteht dabei aus n Zuweisungen der Form $\$groupBy_i$ in result/tuple/$Att_i$, wobei n die Anzahl der Gruppierungsattribute und $Att_i$ ein bestimmtes Gruppierungsattribut darstellen. Dadurch wird ein n-

dimenstionaler Vektorraum aufgespannt, der durch die For-Klausel sequentiell durchlaufen wird. In der Let-Klausel wird nun jedes tuple-Element einem bestimmten Punkt in diesem Raum zugeordnet und in der Return-Klausel einem konkreten group-Element zugewiesen. Dieses group-Element enthält zum einen die n Gruppierungsattribute und zum anderen eine Menge von tuple-Elementen, in denen die jeweiligen Attribute mit den Gruppierungsattributen identisch sind. Die Ergebnisstruktur dieses Teils des dritten Schrittes ist somit gültig bezüglich folgender DTD:

<! ELEMENT result (group* | tuple*)>
<! ELEMENT group (any-node*, tuple*)>
<! ELEMENT tuple = any-node*>

Auf diese Struktur kann nun die Selektion bezüglich der Having-Klausel durchgeführt werden. Dies geschieht ebenfalls mittels einem FLWR-Ausdruck, in der die group-Elemente sequentiell durchlaufen werden. Die Where-Klausel ergibt sich dabei direkt aus der globalen HAVING-Klausel. Es muss dabei lediglich jedem globalen Attribut der Pfad /tuple/ vorangestellt werden, da die Attribute nun mit dem absoluten Pfad Result/group/tuple/Attribut erreichbar sind. In die Return-Klauseln werden die group-Elemente übernommen, die sich durch die HAVING-Klausel qualifiziert haben. Abbildung 3 stellt diesen Schritt graphisch dar.

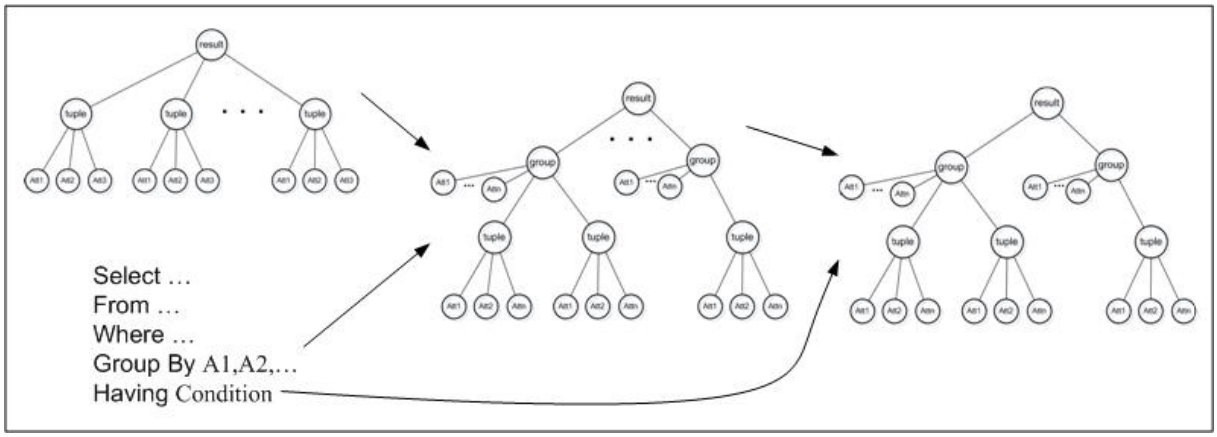


**Abb. 3:** Umsetzung der Group By- und Having-Klausel

**Schritt 4**

Im vierten und letzten Transformationsschritt werden die Attribute der globalen SELECT-Klausel berechnet und die eventuell spezifizierte ORDER BY-Klausel umgesetzt. Dies geschieht ebenfalls durch einen neuen FLWR-Ausdruck. Um festzustellen, worauf sich die Zuweisung in der For-Klausel bezieht, müssen prinzipiell drei Fälle unterschieden werden:

1. Die globale SELECT-Klausel enthält nur Aggregatfunktionen und keine Attribute. Falls eine Gruppierung existiert, beziehen sich die Aggregatfunktionen auf das group-Element des Ergebnisses, andernfalls auf das result-Element, also die ganze Ergebnismenge. In beiden Fällen jedoch auf das Vaterelement eines tuple-Elements.
2. Die globale SELECT-Klausel enthält nur Attribute und keine Aggregatfunktion. Falls eine Gruppierung existiert, müssen die Attribute in der GROUP BY-Klausel spezifiziert sein, somit beziehen sie sich auf das group-Element. Ohne GROUP BY-Klausel beziehen sich die Attribute auf jedes einzelne tuple-Element. In beiden Fällen erfolgt der Bezug jedoch auf alle direkten Kindelemente des result-Elements.
3. Die globale SELECT-Klausel enthält sowohl Attribute als auch Aggregatfunktionen. Daraus folgt, dass eine GROUP BY-Klausel spezifiziert wurde und zwar bezüglich der Attribute, die in der SELECT-Klausel als direkte Attribute vorkommen. Daraus folgt weiterhin, dass diese Attribute in allen tuple-Elementen der Gruppierung identisch sind. Somit ist dieser Fall analog zum zweiten Fall zu behandeln.

Die Order By-Klausel in diesem FLWR-Ausdruck entspricht der ORDER BY-Klausel der globalen Anfrage. Die textuelle Repräsentation wäre somit ORDER BY /{Sortierungsattribut(e)} ASCENDING | DESCENDING.

Die Return-Klausel besteht aus den Aggregatfunktionen und den Attributen der globalen SELECT-Klausel. Die Aggregatfunktionen haben dabei die Form AggFkt(/tuple/*Attribut*). Die Pfadnamen der Attribute lauten /*Attribut*. Diese werden jeweils mit zwei Tags umschlossen. Der Name dieser Tags ist entweder die Aggregatfunkion auf das Attribut angewandt, der globale Attributname oder der jeweilige, eventuell spezifizierte Aliasname. Der gesamte Inhalt der Return-Klausel wird ebenfalls wieder mit den tuple-Tags umschlossen.

Das Ergebnis des vierten Schrittes bildet den Abschluss der Anfragetransformation. Diese generierte Anfrage kann nun an das jeweilige XML-Datenbanksystem zusammen mit der Zugehörigen Schemabeschreibung, auf die sich diese Transformation bezog, geschickt werden. Die Ergebnismenge, die der Föderierungsdienst enthält liegt dann in einer XML-Form vor, die ebenfalls äquivalent zur SET OF TUPLE OF – Darstellung ist. Somit stellt eine Transformation der Ergebnismenge in das globale Datenmodell keine weitere Schwierigkeit mehr dar. Für weitere Schemabeschreibungen, die das XML-Komponentendatenbanksystem der Föderation eventuell bereitstellt, wird dieser Algorithmus jeweils separat wiederholt. Auf die darauf aufbauende Phase der Nachbearbeitung der Einzelergebnisse wird ebenfalls in [Jah05] genauer eingegangen.

# 3 Zusammenfassung und Ausblick

Das Ziel des hier vorgestellten Algorithmus ist eine Transformation einer globalen SQL-Anfrage in einen äquivalenten XQuery-Ausdruck, der bezüglich einer XML-Dokumentenkollektion an ein XML-Komponentendatenbanksystem übergeben werden kann. Vorraussetzung dafür ist lediglich eine semantikerhaltende Abbildung der lokalen Schemabeschreibung auf das globale Schema des Föderierungsdienstes. Ausgehend von dieser Abbildung und der aktuellen SQL-Anfrage wird der korrespondierende XQuery-Ausdruck abgeleitet. Der Algorithmus ist in der hier vorgestellten Form allerdings nur auf sehr einfach strukturierte Anfragen beschränkt. Die Umsetzung weiterer Konstrukte wie den Like-Operator oder verzahnt geschachtelte Anfragen wird in [Jah05] untersucht. Auch die Anwendung auf SQL-Erweiterungen ist denkbar. Eine in diesem Kontext interessante SQL-Erweiterung ist beispielsweise MM/Text. Dadurch ermöglichte Information Retrieval-Techniken, die Vergleiche mit Stammwortreduktion oder distanzbasierte Anfragen ermöglichen, könnten womöglich durch die Volltext-Erweiterung von XQuery ebenfalls durch diesen Algorithmus umgesetzt werden. Ein anderer Ansatzpunkt für diese Arbeit wäre beispielsweise ein mögliches Pipelining der Transformation zu untersuchen. Da die Einzelschritte völlig unabhängig vom vorangegangenen Schritt sind, sollte eine solche Parallelisierung verschiedener Anfragetransformationen bezüglich unterschiedlicher Schemabeschreibungen prinzipiell Möglich sein. Beispielsweise wäre der Einsatz von drei Prozessoren denkbar, wobei der dritte und der vierte Transformationsschritt von einem Prozessor bearbeitet werden muss. Da der dritte Schritt unter Umständen nicht durchgeführt wird, bestünde sonst die Gefahr, dass eine Transformation ohne GROUP BY-Klausel eine Transformation mit GROUP BY-Klausel "einholen" könnte.

# Bibliography


[Con97] Stefan Conrad. *Föderierte Datenbanksysteme*. Springer, 1997

[EEL04] Francisco J.C. Escobar, Enrique D. Espinosa, Rafael Lozano. *XML Information Retrieval Using SQL2Xquery*. Departmento do Computación, Tecnológico de Monterrey-Campus cd. De México

[ERS99] Ahmed Elmagarmid, Marek Rusinkiewicz, Amit Sheth. *Management of Heterogeneous and Autonomous Database Systems.* Morgen Kaufmann Publishers, Inc., 1999

[Jah05] Heiko Jahnkuhn. *Transformation zwischen SQL- und XQuery-Anfragen innerhalb föderierter Informationssysteme.* Studienarbeit, Universität Rostock, 2005

[KM03] Meike Klettke, Holger Meyer. *XML und Datenbanken*. dpunkt Verlag GmbH, 2003