Anfragen an XML-Dokumente

Proseminar/Seminar SS2003 zum Themenbereich Logische Aspekte von XML

Björn Gehl <bjge@gmx.de>

Einführung

Semistrukturierte Daten die in einer Datenbank gespeichert sind, kann man mit ähnlichen Konstrukten abfragen, wie sie auch in relationalen oder objektorientierten Datenbanken verwendet werden. Da die Datenmodelle für semistrukturierte Daten auf Graphen basieren, müssen Anfragesprachen diese Graphen nach bestimmten Vorgaben durchsuchen können. Die Hauptaufgabe einer Anfragesprache (Query Language) für XML ist es aus einer XML-Datenbank bzw. mehreren XML-Datenbanken Informationen nach gewünschten Kriterien zu selektieren. Zusätzlich gibt es aber noch weitere Aufgaben, die man mittels einiger Anfragesprachen lösen kann. Dazu zählt u.a. das Verändern von Daten oder das Erstellen einer neuen semistrukturierten Datenbank in XML-Syntax. Merkmale sind somit komplexe Prädikate auf Daten anzuwenden, Daten aus verschiedenen Quellen zusammenzuführen und Daten neu zu strukturieren. Somit bilden Anfragen (Querys) die Grundlage um mit Datenbanken aktiv zu arbeiten.

Zuerst werden in dieser Ausarbeitung kurz die Grundlagen aufgeführt, welche wichtig für den weiteren Verlauf bezüglich Anfragen auf semistrukturierte Datenbanken sind. Dann wird näher auf diverse Query-Languages, wie XML-QL & Datalog eingegangen und einige spezielle Eigenschaften hervorgehoben. Letztlich spielt die Optimierung auch eine große Rolle um eine Query auf eine gegebene Datenbank zu verbessern. Hierbei bezieht sich in dem letzten Teil der Ausarbeitung die Optimierung auf die Datenbank selbst.

1] GRUNDLAGEN

1.1 Aussagelogik

Die logischen Ansätze bei Querys verlangen die Kenntnis der Aussagelogik. Hier kurz die Syntax:

$A_1, A_2, ..., A_n$ sind atomare Formeln.
Formeln der Aussagenlogik werden induktiv definiert:
1] Alle atomaren Formeln sind Formeln
2] Für alle Formeln F und G sind $(F \wedge G)$ und $(F \vee G)$ Formeln
3] Für jede Formel F ist $\neg F$ eine Formel

Mittels Aussagelogik wird eine Aussage über die Gültigkeit ermöglicht. Ob eine Formel erfüllbar, also „wahr“ ist, oder nicht erfüllbar und somit „falsch“ ist.

1.2 Graphentheorie

Da XML-Dokumente immer eine Graphstruktur beschreiben, werden hier Grundlagen der Graphentheorie kurz zusammengefasst.

Ein Graph(N,E) besteht aus einer Menge N von Knoten und einer Menge E von Kanten. Mit jeder Kante e aus E ist ein Paar von Knoten assoziiert. Bei gerichteten Graphen liegt ein geordnetes Paar mit Startknoten und Zielknoten vor. Ein Weg ist eine Folge von Kanten $e_1,e_2,\ldots$, die eine mögliche Verbindungskette beschreiben. Ein Knoten r ist ein Wurzelknoten für einen Graphen (N,E) wenn eine Weg von r nach n $\forall$ n$\in$N mit n$\neq$ r existiert. Ein Graph wird als Baum bezeichnet genau dann wenn es einen eindeutigen Pfad von der Wurzel r nach n $\forall$ n$\in$ N mit n $\neq$ r gibt. Ein Zyklus definiert einen Weg von einem Knoten zu sich selbst. Ein Graph ohne Zyklen heißt azyklisch.

2] ANFRAGESPRACHEN

Der Einsatzbereich von Anfragesprachen ist selektieren und transformieren von Daten aus Datenbanken beliebiger Größe. Hierbei wird von einer Datenbank ausgegangen, die man in einer Graph- oder Baumstruktur darstellen kann. In dieser Datenbank durchläuft die Anfrage die Pfade und führt die gewünschten Effekte aus. Zusätzlich haben komplexere Query Languages die Möglichkeit Daten von verschiedenen Datenbänken zusammenzuführen (Joins) und zu verarbeiten. Auch das Anwenden von (Daten-) Rekursion zählt zu einer Fähigkeit, die einige Anfragesprachen durchführen können.

2.1 Pfadausdrücke (Path Expressions)

Da Anfragesprachen von XML Datenbanken beliebige Tiefen in Datengraphen erreichen müssen, beschreiben alle eine Art von Pfadausdrücken, die den Weg zu einem Ergebnis in einer Baumstruktur beschreiben, bis hin zum zu selektierenden Knoten. Um Path Expressions zu beschreiben wird die Syntax regulärer Ausdrücke genutzt. Hier die allgemeine Syntax regulärer Ausdrücke:

e :: **(**	l	Labelname einer Kante
	e	leerer Ausdruck
	_	“wild card”, trifft auf jeden Knoten zu
	e.e	Komposition
	(e)	Klammerung
	e*	Kleen-Stern (null oder mehr Wiederholungen von e)
	e+	e+ : ein oder mehr Wiederholungen von e
	e?	e? : null oder ein Auftreten von e **)**

Ein regulärer Ausdruck e, der auf einen Datengraphen D angewendet werden soll, wird folgendermaßen berechnet. Zuerst erstellt man einen nichtdeterministischen Automaten A der äquivalent zu e ist. Sei $\{x_1,\ldots,x_n\}$ die Liste der Knoten in D mit x_1 = Wurzel. Und sei $\{s_1,\ldots,s_n\}$ die Liste der Zustände in A mit s_1 = Startzustand.
Dann initialisiere mit der Liste *Closure* = $\{(x_1,s_1)\}$ und wiederhole Folgendes solange bis Closure sich nicht mehr ändert:

Wähle Paar (x,s) $\in$ Closure & kontrolliere alle Kanten der Datenbank $x \xrightarrow{a} x'$ Bei allen Übergänge $s \xrightarrow{a} s'$ in A mit dem selben Label (a) füge (x’,s’) zu Closure hinzu. Nachdem Closure einen Fixpunkt erreicht hat ist das Resultat von e auf D die Liste der Knoten x ,die Closure in Paaren (x,s) beinhaltet, mit s = Fixpunkt in A. Diese

Prozedur ist immer endlich, und im worst-case Fall wird jeder Knoten x im Datengraph so oft wie A Zustände hat, besucht.

Pfadausdrücke sind aber nur ein besonderes Merkmal einer Anfragesprache, da noch einige Aufgaben unerfüllt bleiben. Dazu gehören folgende fehlende Möglichkeiten: Erstellen neuer Knoten, realisieren von Joins relationaler Datenbanken, testen von Werten die in der Datenbank gespeichert sind. Somit bilden Path Expressions eine Grundlage für umfangreichere Anfragesprachen.

2.2 First-Order Logic

Zu einer Klassifizierung von Anfragearten wird die Ordnung der Logik verwand. Bei der First-Order Logic (Logik erster Ordnung) werden die atomaren Formeln der Aussagelogik durch so genannte Prädikatensymbole (P_i^k) ersetzt. Hierbei beschreibt k die Stelligkeit und $i \in \mathbb{N}$ den Index. Zusätzlich gibt es Variablen x_i, Funktionssymbole $f_i(k)$ und Terme t_k. Für Terme gilt folgendes: Jede Variable ist ein Term und wenn f ein Funktionssymbol der Stelligkeit k ist, und $t_1,...,t_k$ Terme sind, so ist auch $f(t_1,...,t_k)$ ein Term.
Die Formeln der Prädikatenlogik werden induktiv folgendermaßen definiert:

1] Falls P ein Prädikatensymbol der Stelligkeit k ist, und falls $t_1,...,t_k$ Terme sind, dann ist $P(t_1,...,t_k)$ eine Formel

2] Für alle Formeln F und G sind $(F \wedge G)$ und $(F \vee G)$ Formeln

3] Für jede Formel F ist $\neg F$ eine Formel

4] Falls F eine Formel ist und x eine Variable, so sind auch $\forall xF$ und $\exists xF$ Formeln

In Abbildung 1 sieht man eine Datenbank in Graphdarstellung. Die von jedem Knoten ausgehenden Kanten sind knotenbezogen durchnummeriert. D.h., dass von jedem Knoten mit n Kanten kein Kantenlabel zweifach vorkommt. Jedoch sind die Kantenlabel des ganzen Datenbaumes nicht eindeutig. Jedoch die Knoten, die ihren Namen aus den Pfadlabels beziehen die durchlaufen werden müssen, um von der Wurzel ausgehend zu dem Knoten zu gelangen, sind eindeutig. Somit ergibt eine Anfrage über einen Pfad ein eindeutiges Ergebnis. Der Knotenbezeichner einer Anfrage ist „first-order“.

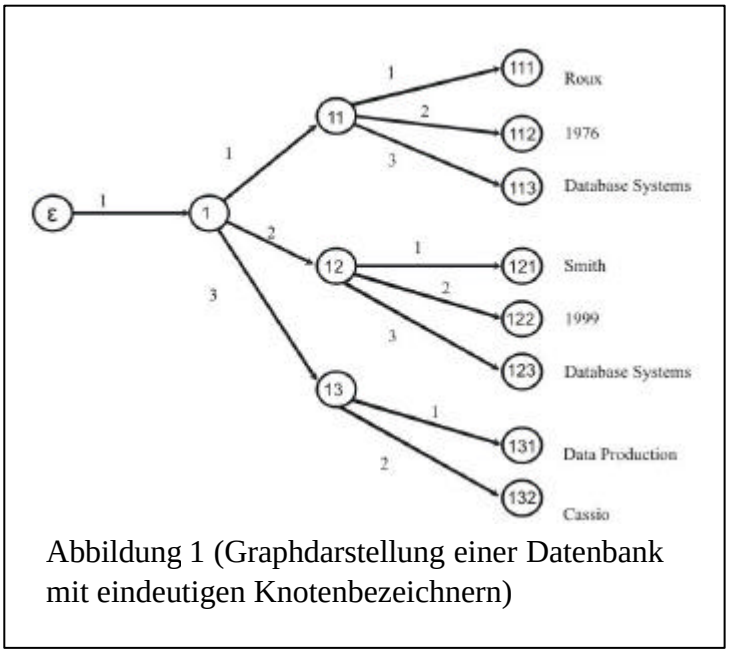

Abbildung 1 (Graphdarstellung einer Datenbank mit eindeutigen Knotenbezeichnern)

2.3 Second-Order Logic

Zu den Logik-Arten gehören auch Logiken höherer Ordnung (Higher-Order Logics), die aber alle als Varianten der Logik zweiter Ordnung (Second-Order Logic)

aufgefasst werden können. Im Vergleich zur Logik erster Ordnung ist die Logik zweiter Ordnung ausdrucksstärker. Als syntaktische Erweiterungen gibt es Prädikate höherer Ordnung, d.h. Prädikate von Prädikaten und Quantifizierung von Prädikaten, um die Ausdruckskraft zu erhöhen. Somit wird im Vergleich zu 2.2 ein Knoten durch eine Menge von Individuen aus dem Universum interpretiert, ist somit „second-order".

2.4 Lorel

Die Anfragesprache Lorel (Lightweight Object REpository Language) wurde an der Stanford University von S. Abiteoul, D. Quass, J. McHugh, J. Widom und J. Wiener entwickelt (http://www-db.stanford.edu/lore). Lorel ist eine benutzerfreundliche Sprache die das OEM-Datenmodell nutzt. Das Modell beschreibt einen gerichteten Graphen mit gelabelten Kanten. Die Knoten beschreiben die Objekte und die Kanten ihre Attribute. Lorel baut auf einer Anfragesprache namens OQL auf und dient als Erweiterung dieser Sprache. Bei der Bearbeitung der Anfrage in Lorel wird die Lorelsyntax in eine OQL-Syntax umgewandelt. Die Syntax einer Lorel-Anfrage setzt sich grundlegend aus drei Teilen zusammen:

```
select {select-Ausdruck}
[from {from-Ausdruck}]
[where {where-Ausdruck}]
```

Der from-Ausdruck beschreibt den Herkunftsort in der semistrukturellen Datenbank genauer. Legt somit also eine Menge fest. Im letzten Teil, dem where-Ausdruck, kann man die ausgewählten Informationen mit Bedingungen weiter einschränken und in der select-Klausel wird die gewünschte Information in der Datenbank ausgewählt. Somit minimiert man die Datenbank prinzipiell nach und nach auf den letztendlich gewünschten Teil.
Besondere Merkmale von Lorel sind die mächtigen und flexiblen Pfadausdrücke (2.1), die man im Ausdruck nutzen kann. Ein Pfad wird in Lorel durch eine von Punkten getrennte Aneinanderreihung von Bezeichnern angegeben, die Namen von Elementen und Attributen entsprächen. Hier ein einfaches Beispiel einer Lorelanfrage auf die in Abbildung 2 angegebene Datenbank:

```
select x
from biblio.book.y,
      y author x
where „1999" in x.date
```

Das Resultat dieser Anfrage ist der Autor Smith.

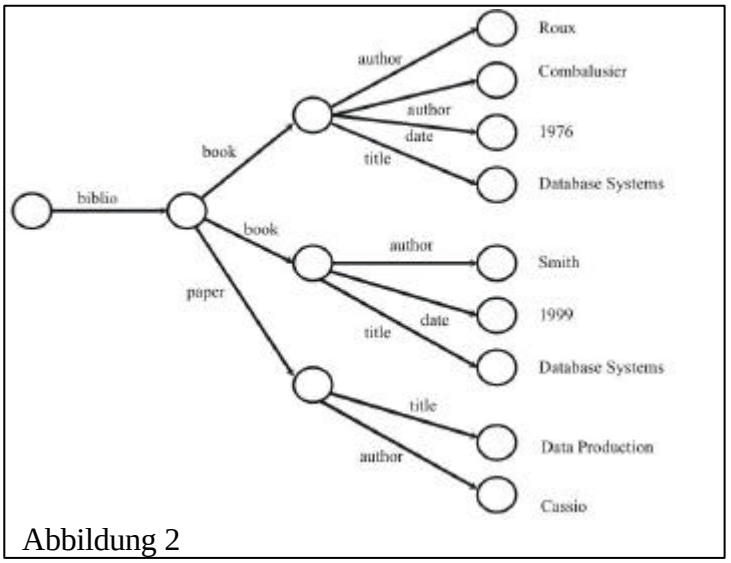

Abbildung 2

Mit Lorel besitzt man auch die Möglichkeit neue Knoten und neue Datenbankstrukturen zu erstellen.

```
select row: x
from biblio.book.year
```

Damit wird ein neuer Wurzelknoten erstellt. Von diesem wird jedes Resultat, welches mit dem Pfadausdruck (biblio.book.author) evaluiert, mit einer Kante verbunden. Die Kante hat das Label row. Der SSD-Ausdruck des Resultates der Anfrage sieht somit wie folgt aus: {row: „1976", row: „1999"}

Da von dem Nutzer das Schema der XML-Daten oftmals nicht gekannt wird, kann er in seinen Anfragen auch Pfadausdrücke verwenden. D.h. Muster für Pfade, Attribute und atomare Werte vorgeben, für die dann beim Durchlaufen des Graphen Entsprechungen gesucht werden. Muster für atomare Werte und für Attribute werden über Wildcards '_' angegeben. Bei den Pfaden werden reguläre Ausdrücke verwendet.
Zusätzlich kann man mittels Lorel-Querys auch allgemeine Bedingungen in der Where-Klausel angeben, um die Selektion einzugrenzen.

```
select x
from biblio.book y,
      y.author x
where x.date < 2000
```

Mit dieser Query kann man beispielsweise alle Bucheautoren auswählen lassen, die ein Buch vor dem Jahr 2000 veröffentlicht haben. Joins sind in Lorel auch vollständig möglich.

```
select  row: w
from biblio.book x,  x.refers-to y, y.author w, x.refers-to z
where NOT ( y = z )
   and w in z.author
   and matches ("*Database*", x.title)
```

In dieser anspruchsvolleren Query wird nach Autoren gesucht, die sich auf mindestens zwei Bücher beziehen und zusätzlich „Database" im Titel enthalten ist. Besonderheiten von Lorel wie ein negierter Vergleich NOT (y = z) und ein Textvergleich mit matches wurde hier genutzt um diese Anfrage zu ermöglichen.

2.5 UnQL

UnQL (Unstructured Query Language) ist eine Anfragesprache, die auf einem semistrukturierten Datenmodell ähnlich der SSD-Syntax basiert. Sie nutzt zusätzlich nur Pattern, um strukturnäher zu arbeiten und Variablen an Knoten zu binden. Beispielsweise „{biblio: **{**paper: X**}**} in db" statt „{biblio.paper: X} in db". Die from-Klausel wird in UnQL nicht benötigt, da das Binden der Variablen durch Pattern bereits erledigt wird. Die where-Klausel enthält zwei Arten syntaktischer Strukturen. Zum einen die *Generatoren*, die die Variablen einführen und auch binden, zum anderen die *Bedingungen*, die die Auswahl auf das Gewünschte einschränken.

```
select title: X
where  {biblio: {paper: {title: X, year: Y}}}
          in db,
         Y > 1999
```

In dieser UnQL-Anfrage ist „{biblio: {paper: {title: X, year: Y}}} in db" der Generator und "Y > 1999" ist die Bedingung. Auch Joins können mit Hilfe von Pattern realisiert werden.

2.6 XML-QL

XML-QL ist eine datenbankorientierte Anfragesprache, die demzufolge auch dieselben Aufgaben hat wie Lorel und UnQL. XML-QL wurde in den AT&T Labs von Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy und Dan Suciu als Teil des Strudel-Projektes entwickelt. Den Prototypen der Query Language kann man

unter http://www.research.att.com/sw/tools/xmlql finden. Die allgemeine Syntax sieht wie folgt aus:

where {Prädikat}
in {Dateiname}
construct {{Query}}

In der where-Klausel wird das Pattern angegeben nach dem gesucht werden soll. Durch „in" wird das Dokument bestimmt das durchsucht werden soll. XML-QL besitzt eine explizite construct-Klausel, die zur Erstellung eines neuen Dokumentes aus dem Ergebnis der Anfrage dient. Genutzt werden Pfadausdrücke (2.1) und wie bei UnQL (2.5) auch Pattern. XML-QL ist relational vollständig und das genutzte Datenmodell besitzt gelabelte Kanten
In folgendem Beispiel sind einige Möglichkeiten enthalten, die XML-QL bereitstellt.

```
where  <book>
           <title> Database Systems </title>
           <author> $A </author>
           <year> $Y </year>
       </book> in db.xml
construct $A
```

XML-QL ermöglicht den Umgang mit Variablen. Diese werden im Code mit $ markiert. In dieser Anfrage existieren zwei Variablen, die erste $A steht für die Autoren und $Y ist eine Variable für das Jahr der Buchveröffentlichung. <book> .. </book> ist in diesem Beispiel ein Pattern. Der Anfrageprozessor wird das Pattern in allen möglichen Wegen den Daten anpassen und die Variablen binden. Ein Pattern spezifiziert also nur was in den Daten enthalten sein soll, aber nicht was nicht existieren darf. Somit können in den Daten ohne weiteres zusätzliche Daten sein. In dem Beispiel entspricht ein Buch dem Pattern, solange es einen Autor, ein Erscheinungsjahr und ein Titel hat. Wenn ein Buch also zwei Autoren hätte, würde es auch auf die Anfrage passen.
Es ist auch mit XML-QL möglich Joins auszudrücken. Dazu nutzt man dieselbe Variable in zwei verschiedenen Tags:

```
where  <book>
           <author> $A </>
           content_as $B1 in "db.xml",
       </>
       <book>
           <author> $A </>
           content_as $B2 in "db.xml",
       </> in db.xml
       B1!=B2
construct <result> $A </>
```

Hier werden alle Autoren ausgegeben, die mindestens 2 Bücher herausgegeben haben. In dem Beispiel sind auch Endtags enthalten, die nicht betitelt sind (</>). Diese schließen den zuletzt geöffneten Tag. Sie dienen also lediglich zur Kürzung des Codes. Die Anfrage konstruiert für jede Bindung der Variable $A ein Element der Form: <result><author> … </author></result>.
Das Konstrukt content_as bindet eine Variable an den Inhalt eines Elementes. Zählt also als „syntaktischer Zucker" genau ebenso wie element_as, womit man Bindungen an das Element selbst realisieren kann.
Eine Negation von einem Prädikat über eine Menge von Objekten ist erfüllt, wenn kein Objekt das Prädikat erfüllt. In Lorel wird z. B. ein Prädikat durch den Befehl NOT

negiert (3.4). Das ist bei XML-QL nicht realisiert. Es unterstützt aber bei einfachen Prädikatsausdrücken den Ungleichheitstest, genau wie im Beispiel mittels B1!=B2. Als Mengenoperationen sind die Vereinigung und der Schnitt möglich jedoch keine Differenz.

3] DATALOG

Datalog ist ein Teil der logischen Programmiersprache Prolog, die 1972 an der Universität Aix-Marseille entwickelt wurde. Diese Teilsprache kann man nutzen um Anragen auf relationale Datenbänke zu realisieren. Vereinfacht beschrieben, bestehen relationale Datenbanken aus Tabellen, in denen Daten gespeichert sind. Üblicherweise stehen die Tabellen in Beziehungen zueinander, d. h. manche Einträge sind Verweise auf andere Einträge in weiteren Tabellen. Relationale Datenbanken basieren also wie Tabellenkalkulationen auf der Anordnung von Daten in einer Struktur aus Zeilen und Spalten. Die mehrfache Speicherung von Daten (Redundanz) wird aber durch die Verknüpfung von Daten über Schlüsselbeziehungen minimiert.
Die theoretische Basis von Datalog ist die first-order Prädikatenlogik (2.2). Die Syntax von Datalog basiert auf der Syntax der Hornklauseln ohne Funktionen.

Definition Hornklausel:
- Eine Hornklausel ist eine Klausel mit maximal einem positiven Literal (Atom).
- Eine definite Klausel ist eine Klausel mit genau einem positiven Literal.

Somit ist die Klausel der Form: $P_0 \leftarrow \neg P_1 \vee \ldots \vee \neg P_m$

In der logischen Programmierung in Datalog wird nachfolgende Notation verwendet:
P_0 :- $P_1, \ldots, P_m$.

Hierbei nennt man P_0 den Kopf und $\{P_1, \ldots, P_m\}$ den Rumpf der Klausel. Hornklauseln sind also Einschränkungen in der Syntax und Komplexität der Prädikatenlogik.
Die relationale Struktur der Datenbank in Abbildung 3a entspricht folgendem Aufbau. Die Information der Datenbank kann man in zwei Tabellen darstellen. Eine beschreibt die Kantenmenge und eine beschreibt das Label der Knoten (Blätter).

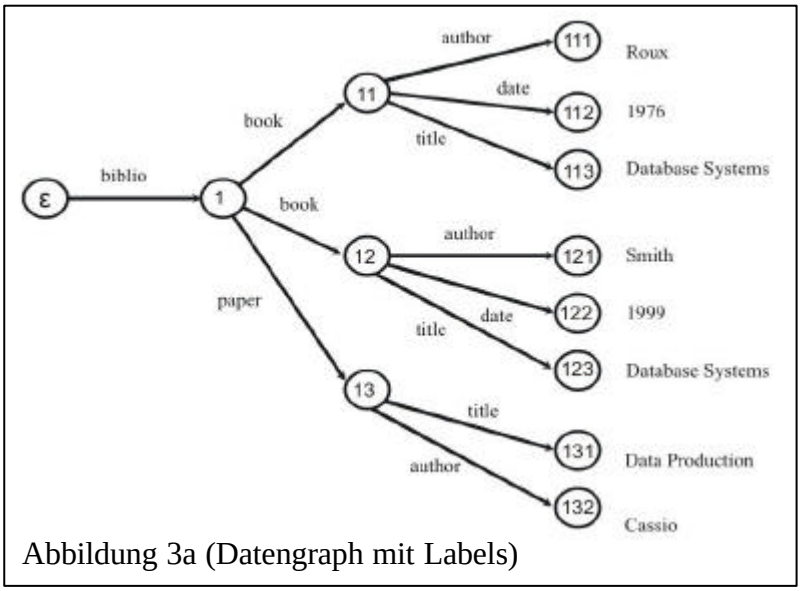

Abbildung 3a (Datengraph mit Labels)

Abbildung 3b zeigt einen Ausschnitt aus den Tabellen:
Die Tabelle „ref“ beschreibt die Kantenmenge mit den drei Spalten Start-, Zielknoten und Kantenlabel. Die Tabelle „val“ beinhaltet die Informationen der Blattlabels mit der Knoten-ID und dem dazugehörigen Label.

ref:

Start	Kantenlabel	Ziel
e	biblio	1
1	book	11
1	book	12
1	paper	13
11	author	111

usw.

val:

Knoten	Knotenlabel
111	Roux
112	1976
113	Database System
121	Smith
122	1999

usw.

Abbildung 3b (relationale Struktur)

In Datalog werden Anfragen auf eine Datenbank über die relationalen Daten gemacht, also über Tupel (val) und Tripel (ref).
Folgende Beispielsanfrage in Datalog auf die Datenbank in Abbildung 3a ist im Ergebnis in Abbildung 4 dargestellt.

ref(ans,"author",x) ← ref(e,"biblio",w),
ref(w,"book",y),
ref(y,"author",x),
ref(y,"title",v),
val(v,"Database Systems")

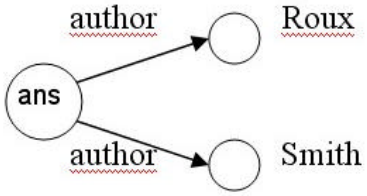

Abbildung 4

Diese Query entspricht bis auf das Knotenlabel „ans" dieser Lorelanfrage:

```
select author: x
from biblio.book.y
      y.author x
where „Database Systems" in y.titel
```

Anfragen, die rekursiv definierte Prädikate betreffen sind für Datalog auch möglich. Hier die transitive Hülle über einen Graphen G=(x,y) als ein rekursives Beispiel:

Def. transitive Hülle:
$T(x,y) \leftarrow G(x,y)$
$T(x,y) \leftarrow G(x,z), T(z,y)$

$\Leftrightarrow$

in Datalog:
$\forall x,y\ (T(x,y) \leftarrow G(x,y))$
$\forall x,y,z\ (T(x,y) \leftarrow (G(x,z) \wedge T(z,y)))$

4] DATENBANKOPTIMIERUNG FÜR ANFRAGESPRACHEN

4.1 Duplikatenentfernung

Das Graphenmodell das im Folgenden verwendet wird, entspricht grundsätzlich dem OEM Datenmodel jedoch haben die Knoten keine Objektbezeichner (Objectidentifier). Dadurch wird das Teilen oder Vereinigen von Knoten erlaubt ohne eine Änderung der Datensemantik vorauszusetzen. Dieses Datenmodell wird auch in UnQL (2.5) verwendet.

Um Duplikate zu entfernen (duplicate elimination) geht man stufenweise vor. Es wird von den Blättern aus begonnen und über die Unterbäume bis hin zur Wurzel vorgegangen (siehe Abbildung 5).

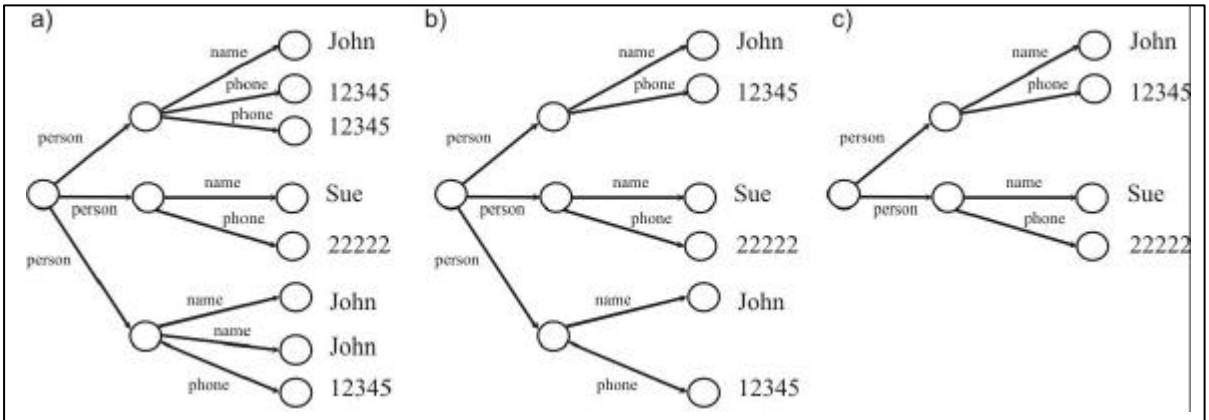

Abbildung 5 (Beispiel einer „duplicate elimination“)

Die Duplikateneliminierung wird immer auf den jeweiligen Unterbaum bezogen, da sonst jedes doppelte Auftreten gelöscht würde ohne den Bezug zu beachten. Beim Nichtbeachten des Baumbezugs wäre beispielsweise Folgendes möglich:
Angenommen im Ursprung a) der Datenbank in Abbildung 3 wäre die Telefonnummer von Sue dieselbe Nummer wie die von John. Dann würde bei einem der Beiden die Telefonnummer als Doppeltauftreten interpretiert. Bei Löschung einer Aussage fehle aber die Information über die Nummer der anderen Person. Also läge somit kein echtes Duplikat vor.

4.3 Auffalten

Das Problem in der zuvor beschriebenen Möglichkeit Duplikate zu eliminieren stellen Graphen da, die Zyklen beinhalten. Die Rekursion würde da unendlich weiterlaufen. Ein Lösungsansatz ist das Auffalten. Dort gälten (unformal) zwei Objekte als gleich, wenn zu beweisen ist, dass die unendlich aufgefalteten Bäume gleich sind. Ein Beispiel hierfür ist in Abbildung 6a und 6b zu finden.

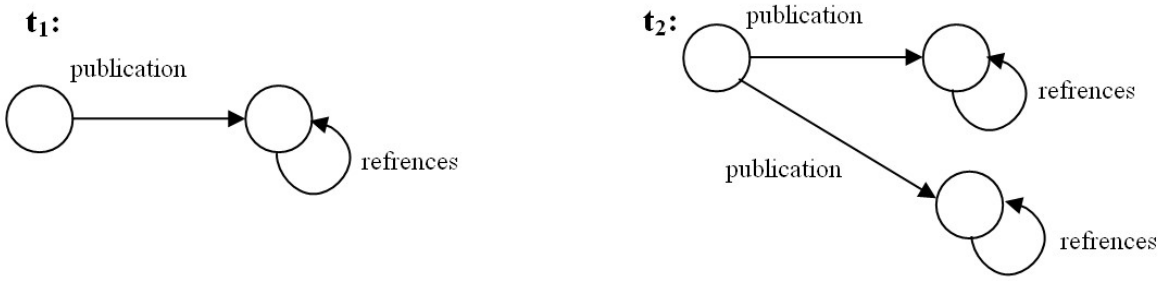

Abbildung 6a (Beispiel-Datengraphen, die in 4b Aufgefaltet dargestellt werden)

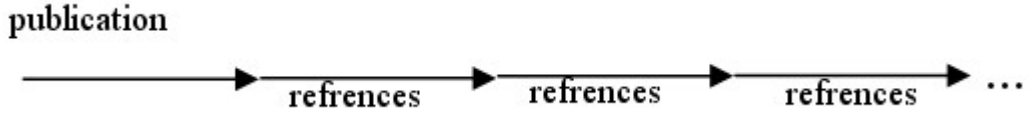

Abbildung 6b (aufgefalteter unendlicher Graph)

4.3 Bisimulation

Ein effizienter Algorithmus um ohne Entfalten mit zyklischen Datenbanken zu arbeiten ist die Bisimulation. Dabei handelt es sich um eine binäre Relation zwischen den Knoten zweier Graphen. Die Bisimulation eine strukturelle Rekursion, die es ermöglicht gegebene zyklische Daten endlich zu berechnen. Dabei wird folgendermaßen vorgegangen:

Seien zwei Graphen G_1 und G_2 gegeben. Wobei x,x',y,y' Knoten sind mit $x,x' \in G_1$ und $y,y' \in G_2$. Zusätzlich beschreiben x und y jeweils die Wurzeln des Graphen.

Die Binäre Relation zwischen diesen gegebenen zwei Graphen G_1 und G_2 muss folgende vier Eigenschaften erfüllen:

1) wenn x,y mit die Wurzeln von G_1 und G_2 sind, so $x \sim y$
2) wenn $x \sim y$ und einer von x,y ist Wurzel, so ist auch der andere Wurzel
3) wenn $x \sim y$ und (x,l,x') eine Kante in G_1, dann existiert eine Kante (y,l,y') in G_2 mit derselben Bezeichnung, so daß $x' \sim y'$. (Ebenso umgekehrt)
4) Wenn $x \sim y$ und x ist ein Blatt mit Wert v in G_1, dann ist y auch ein Blatt mit demselben Wert v in G_2. (Ebenso umgekehrt)

Zwei Graphen sind genau dann gleich, wenn eine Bisimulation zwischen ihnen existiert. Den Ablauf eine Bisimulation nachzuweisen wird in folgendem einfachen Beispiel gezeigt:

Seien zwei Datenbänke g_1 und g_2 gegeben (Abbildung 7).

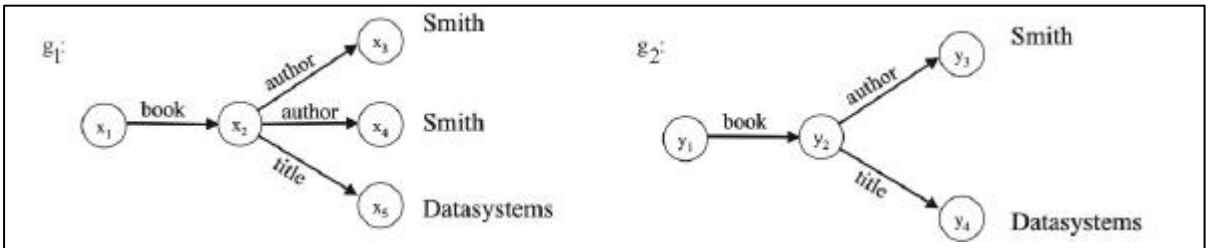

Abbildung 7

Hier der Nachweis der Binären Relation zwischen den Knoten:
$x_1 \sim y_1$, $x_2 \sim y_2$, $x_3 \sim y_3$, $x_4 \sim y_3$, $x_5 \sim y_4$

→ es existiert eine Bisimulation zwischen g_1 und g_2

Somit ist die Gleichheit der beiden Graphen gezeigt.

Referenzen:

[1] Serge Abiteboul, Peter Buneman and Dan Suciu. Data on the Web, Kapitel 2-3, Morgan Kaufmann, 2000.

[2] Serge Abiteboul, Semistructured Data: from Practice to Theory, LICS 2001.

[3] http://www-db.standford.edu/lore

[4] http://www.w3.org

[5] http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/

[6] http://www.w3.org/TandS/QL/QL98/pp.html