Referat zum Seminar
"XML-basierte Datenbanken"

Thema: Query-Optimierung
Eine Zusammenfassung

von Karsten Klein
Seminarleitung: Prof. Dr.-Ing. Gerhard Weikum und Dipl.Inform. Anja Theobald

21. Dezember 1999

Inhaltsverzeichnis

1 Einleitung

Viele Daten im World-Wide Web sind semistrukturiert, also unregelmäßig oder unvollständig, wobei die Struktur häufig und unvorhersehbar wechseln kann. Ein neues Format zum Datenaustausch und zur Datenrepräsentation im Web, XML, unterstützt diese Flexibilität. Um die Anfragen an ein System, das mit semistrukturierten Daten arbeitet, effizient ausführen zu können, muß der Mechanismus der Query-Optimierung an diese Voraussetzungen angepaßt werden. Im folgenden wird der Ansatz des *LORE(Lightweight Object Repository)*-Systems mit dem darin implementierten kostenbasierten Query-Optimierer beschrieben.

2 Grundlagen

2.1 Das LORE-System

Das LORE-DBMS wurde schon vor der XML-Spezifikation für semistrukturierte Daten basierend auf dem *OEM(Object Exchange Model)*-Datenmodell an der Universität Stanford entwickelt und mittlerweile an das XML-Format angepaßt. Alle vorgestellten Konzepte sind in LORE implementiert. Die Anfragesprache *Lorel* ist eine OQL-basierte Sprache zur deklarativen Navigation und Veränderung in Datenbanken. Die semistrukturierten Daten müssen keinem Schema entsprechen, statt dessen wird eine dynamisch generierte Strukturübersicht der Datenbank, der *Dataguide*, angelegt, zu deren Erstellung eine eventuell vorhandene *Document Type Definition(DTD)* allerdings genutzt werden kann.

2.2 Das Datenmodell

Beim OEM-Datenmodell handelt es sich um ein graphbasiertes Modell mit einer Datenrepräsentation durch einen gerichteten, markierten Graph, dessen Knoten Objekten entsprechen und dessen markierte Kanten eine Subobjektbeziehung darstellen. Knoten ohne Subobjekte entsprechen atomaren Objekten, d.h. den Werten der Datenbank. Verschiebt man die Markierungsinformation in die Knoten, so entsprechen diese Objekte gerade den XML-Elementen.
Beispielgraph:

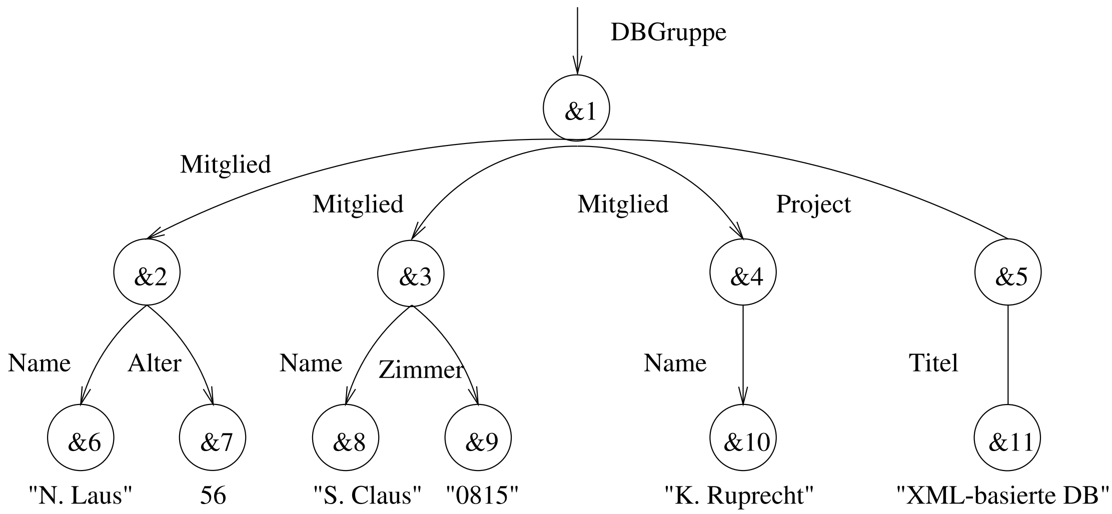

Abbildung 1: OEM - Beispielgraph

2.3 Pfadausdrücke

Für die Navigation durch die Daten werden *Pfadausdrücke* verwendet, sie sind auch ein grundlegendes Prinzip der Anfragesprache Lorel des LORE-Systems.
Ein Pfadausdruck besteht aus einer Liste von *Schritten*, wobei jeder Schritt einer einzelnen Kantennavigation in der DB der Form x.l y mit Markierung l, Quellvariable x und Zielvariable y entspricht, wodurch y über alle l-markierten Subobjekte des x zugewiesenen Objektes läuft, z.B.

DBGruppe.Mitglied.Alter y

weist Variable y alle über diesen Pfad erreichbaren Objekte zu (Die einzelnen Markierungen sind hier in einer Kurzschreibweise unter Auslassung der Variablen mit einem Punkt verknüpft). Ein kompletter Pfadausdruck beginnt mit einem Namen, d.h. mit einem eindeutigen Einstiegspunkt in die Datenbank, im Beispielgraph DBGruppe. Pfadausdrücke können auch verzweigen, z.B.

DBGruppe.Mitglied m, m.Alter m, m.Name n.

Auch reguläre Ausdrücke können als Teil eines Pfadausdrucks verwendet werden, z.B.

<x, (.Bauteil)*,y> bindet an y alle Nachfolger von x im Datengraph, die über eine beliebig lange Kantenfolge aus "*Bauteil*"-Markierungen erreicht werden können.
Beispiele für unterstützte Operatoren sind Abschluß (*,+), Alternation (|), Optional (?) und beliebige Teilpfade (#). Die Information kann auch quantifiziert werden durch die Quantifizierer > (Markierung bezieht sich nur auf Subelemente) und @ (Markierung bezieht sich nur auf Attribute) sowie die Bereichseinschränkung [i-j] (z.B. x.Markierung[1-2] bezieht sich auf die ersten beiden Markierung-Subobjekte jedes DB-Objektes).

3 Ziele, Möglichkeiten und Probleme der Optimierung

Das Ziel der Query-Optimierung ist es, Anfragen an das System mittels Auswahl eines guten Ausführungsplans und passenden Zugriffsstrategien (Indeximplementierung) in vertretbarem Zeitaufwand zu beschleunigen. Für den hier vorliegenden Fall semistrukturierter Daten in einem graphbasierten System mit kostenbasiertem Optimierer ist der grundlegende Aufbau einer Query-Verarbeitung zunächst nicht viel anders als in klassischen Systemen:

- Vorverarbeitung (parsen, Ersetzung von "syntaktischem Zucker", Erkennung häufiger Muster)
- Erstellung eines *logischen Plans* als Repräsentation der notwendigen Schritte auf abstrakter Ebene ("high-level")
- Aufzählung *physikalischer Pläne* als Implementierung innerhalb des Laufzeitsystems in Form von Operatorbäumen ("low-level")
- Auswahl des besten Plans über Kostenschätzung

Für die Möglichkeit der Beschleunigung und die Qualität des Ergebnisses spielen drei Faktoren eine wichtige Rolle:

- die Indexerzeugung für einen schnellen Datenzugriff (effiziente Graphdurchlaufstrategie)
- ein gutes Kostenmodell für eine vernünftige Planbewertung

- eine sinnvolle Suchraumeinschränkung für eine schnellere Planauswahl/Optimierung

Es ergeben sich allerdings auch Probleme durch die Datenrepräsentation:

- neue Indexarten durch die Graphstruktur
- eine komplexere Statistik über dieser Struktur
- es ist kein verwertbares Schema vorhanden
- ein exponentieller Suchraum für die Ausführungspläne

4 Typen von Indizes

Jeder Schritt in einem Pfadausdruck braucht eine Zugriffsmethode, um für seine Variablen eine Objektbindung zu erzeugen. Um diesen Datenzugriff zu beschleunigen, kann ein Index erzeugt werden. Im Gegensatz z.B. zu relationalen Systemen, in denen ein Wertindex über einem Attribut erzeugt wird, um Tupel mit bestimmten Werten schnell zu finden, muß hier die Pfadinformation zur Navigation im Graph mit berücksichtigt werden. Zur schnellen Graphtraversierung gibt es deshalb pfadbasierte Indizes z.B. zum Finden von Nachbarobjekten uber Label, Subobjekte und Pfade. Es stehen u.a. folgende Konzepte zur Verfügung:

- ES (Extent Scan): ES(Label l) gibt alle Objektpaare (x,y) im Graph zurück, die über l verbunden sind
- FS (Forward Scan): FS(Objekt x, Label l) gibt alle von x aus über Label l erreichbaren Subobjekte y zurück, implementiert auf der physikalischen Speicherebene ("pointer chase"). Hierfür muß x bereits gebunden sein.
- BS (Backward Scan): BS(Objekt x, Label l) gibt alle von x aus über Label l erreichbaren Elternobjekte y zurück, implementiert durch extendible hashing ("reverse pointer chase"). Hierfür muß x bereits gebunden sein.
- Vindex (Value): Vindex(Label l, Prädikat P) gibt alle atomaren Objekte (Blätter) im Graph zurück, die Eingangslabel l haben, und P erfüllen
- Pindex (Path): Pindex(Pfadausdruck p) gibt alle über den Pfad erreichbaren Objekte zurück, unterstützt durch den *Dataguide*

5 Die Statistik

Für die Bewertung der Kosten verschiedener Pläne werden über den Daten der DB Statistiken geführt, die im vorliegenden Fall eines graphbasierten Modells die Formaspekte des Graphs und auch Wertebereiche atomarer Objekte beinhalten sollen. In LORE werden Statistiken für Teilpfade des Graphs bis zu einer Länge k gespeichert, wobei k ein Tuningparameter ist. Dabei werden nur lineare Teilpfade berücksichtigt und für Verzweigungen an den Verzweigungspunkten diese Teile verknüpft. Die ermittelten Werte sind damit optimal für Pfadlänge $\leq k + 1$, für längere Pfade erhält man durch Zerlegung und Kombination der Werte für kürzere Teilpfade annähernde Werte. Für Pfade bis zur Länge k und alle Label l werden gespeichert:

- Anzahl Vorkommen des Pfades, $\mid p \mid$
- Anzahl unterschiedlicher, über p erreichbarer Objekte, p_d
- Anzahl über l erreichbarer Subobjekte dieser Objekte, p_l
- Anzahl nach p eingehender, l-gelabelter Kanten, p^l
- Anzahl atomarer Objekte pro Typ, die über p erreichbar sind
- Minimum und Maximum dieser Objeke pro Typ

6 Die Planerstellung

Ziel der Ausführung eines Plans ist die iterative Aufstellung vollständiger Objektbindungen aller Query-Variablen, um aus diesen Listen die Ergebnismenge zu konstruieren. Für den Aufbau eines Ausführungsplans zur Auswertung einer Anfrage können abhängig von der Anfrage und der Datenbank ganz unterschiedliche Konzepte sinnvoll sein.
Grundlegende Typen für die Auswertung sind z.B.

- top-down-Strategie: Man läuft die Pfade vom Namen aus ab, dies entspricht einer Tiefensuche über Pfadausdruck-Kanten. Dies ist vorteilhaft bei nur wenigen passenden Pfaden in der DB.
- bottom-up-Strategie: Umgekehrt zu top-down läuft man von atomaren Objekten zum Namen im Graph nach oben, was bei nur wenigen passenden atomaren Objekten, aber vielen passenden Pfaden von Vorteil ist.
- hybride Strategie: Verknüpfe beide Konzepte, indem ein Teil eines Pfadausdruckes top-down und ein Teil bottom-up ausgewertet wird, und durch Join eine Verknüpfung erzeugt wird.

(Bei komplexen Anfragen gibt es natürlich noch mehr Möglichkeiten.)

Ein einfaches Beispiel:

Sei folgende Anfrage über unserer Beispieldatenbank gegeben:

SELECT x	Resultat:
FROM DBGruppe.Mitglied x	⟨ Mitglied ⟩
WHERE EXISTS y IN x.Alter : y $<$ 60	⟨ Name⟩ N. Laus ⟨/Name⟩
	⟨Alter⟩ 56 ⟨/Alter⟩
	⟨/Mitglied⟩

Die top-down Strategie würde hier die From-Klausel vor der Where-Klausel behandeln, bottom-up umgekehrt.

Man möchte einen logischen Plan erstellen, der die nötigen Schritte auf abstrakter Ebene gut repräsentiert, und trotzdem flexibel in der Umsetzung in physikalische Pläne ist. Für jede der folgenden Datenbanken wäre z.B. eine andere Strategie von Vorteil:

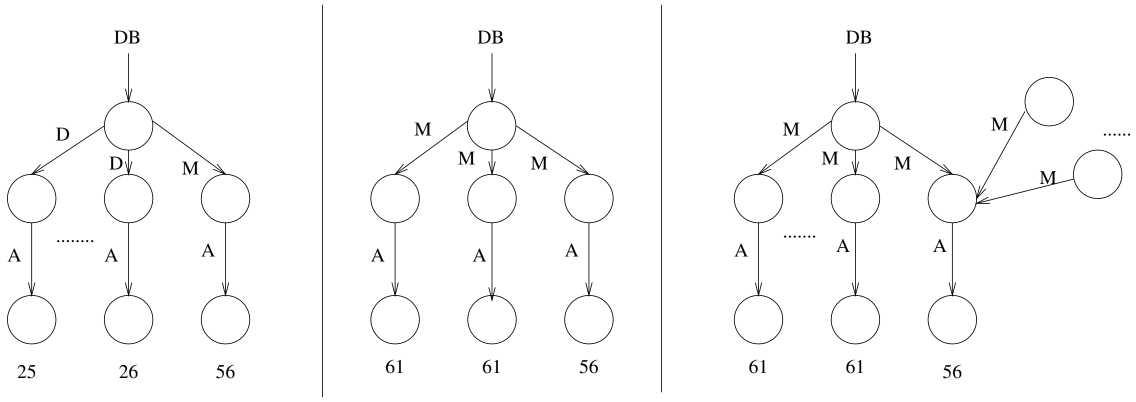

Abbildung 2: 3 DB-Beispiele

Beim ersten Beispiel wäre wegen nur eines passenden Pfades top-down zu bevorzugen, beim zweiten wegen nur eines atomaren Objektes, welches das Prädikat erfüllt, bottom-up, und beim dritten wegen der starken Verzweigung in beide Richtungen die hybride Variante.

7 Logische Pläne

Zur Aufstellung eines logischen Planes wird die Anfrage in unabhängige Komponenten zerlegt, deren Ausführungsreihenfolge nicht festgelegt ist. Der Plan wird durch einen Operatorbaum repräsentiert. An der Verbindung dieser Komponenten, einem sogenannten Rotationspunkt, bewirkt eine Drehung die Erzeugung eines anderen physikalischen Plans. Eine Möglichkeit der Erzeugung physikalischer Pläne wäre dann z.B., daß jeder logische Operator bzw. der zu ihm gehörende Knoten im logischen Plan, seinen optimalen Teilplan erzeugt, indem er basierend auf schon vorhandenen Bindungen und den Ergebnissen seiner Kinderknoten eine lokal optimale Entscheidung trifft.
Beispiel:

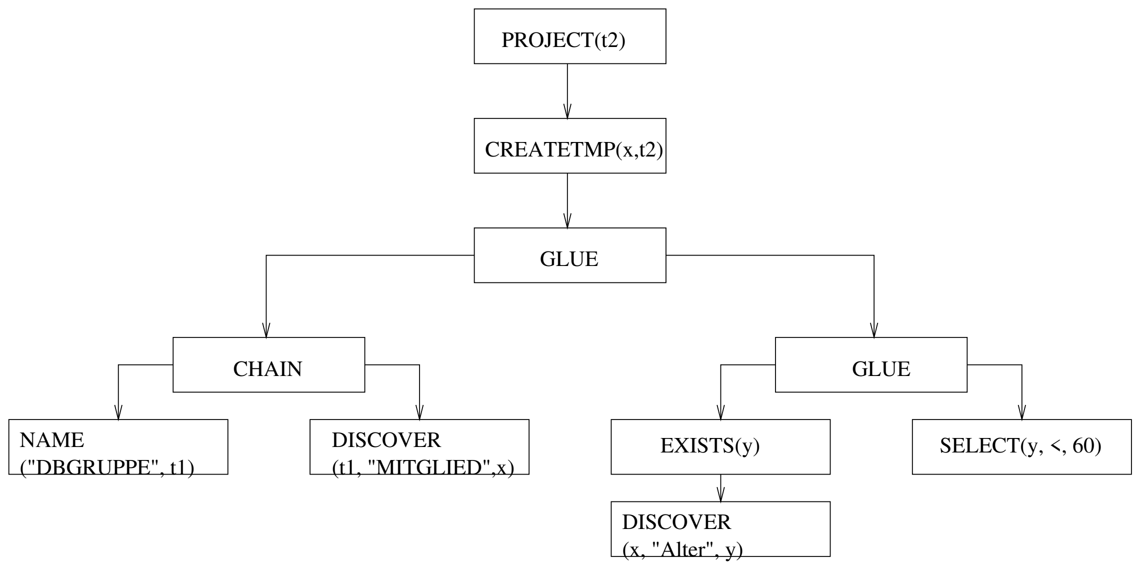

Abbildung 3: Kompletter logischer Plan

chain verbindet hierbei zwei Komponenten eines Pfadausdrucks, der Knoten bekommt von seinen Kindern den jeweils besten Teilplan, zunächst für die Links-vor-Rechts- und dann für die Rechts-vor-Links-Reihenfolge, bestimmt dafür die optimale Join-Methode, vergleicht beide mit dem Pindex für den gesamten Teilausdruck, der an diesem Knoten bearbeitet wird, und gibt den günstigsten Plan zurück.
discover steht für einen Datenzugriff. Ist die Quellvariable gebunden, so wird ein ForwardScan betrachtet, sonst ein BackwardScan, und das Resultat mit einem ExtentScan verglichen
glue verbindet unabhängige Teilpläne, die in beliebiger Reihenfolge durchgeführt werden können (hier das Where-Prädikat und den Existenztest), an einem Rotationspunkt, indem die jeweils optimalen Teilpläne der Kinderknoten für die Links-vor-Rechts- und die Rechts-vor-Links-Reihenfolge mittels des besten Join verknüpft werden und der günstigere Plan zurückgegeben wird.

8 Physikalische Pläne

Das Ziel der Ausführung physikalischer Pläne ist die Erzeugung von Vektoren von Variablenbindungen für die Query-Variablen.

Zur Aufstellung eines physikalischen Planes weist man jedem Schritt eine Zugriffsmethode (phys. Operator des Laufzeitsystems) zu, verknüpft Teilpläne durch Join, und legt eine Reihenfolge fest. Alle möglichen Kombinationen bilden zusammen den Plansuchraum, allerdings sind nicht alle Kombinationen auch gültige Pläne, da z.B. einige Zugriffsmethoden eine Variablenbindung benötigen, die vorher im Plan erzeugt werden muß. Auch hier wird der Plan durch einen Operatorbaum repräsentiert. Das Ergebnis wird dann iterativ über die Operatoren in den Knoten des Baumes erzeugt.

- Operatoren, die Dateninformation bereitstellen:
  - Scan(x, l, y): entspricht Forward Scan
  - Lindex(x, l, y): entspricht Backward Scan
  - Pindex(Pfadausdruck, x): entspricht Pfadindex, unterstützt durch Dataguide
  - Bindex(l, x, y): entspricht Extent Scan
  - Name(x, n): Prüft, ob x ein Name-Objekt ist
  - Vindex(Op, Wert, l, x): entspricht dem Wertindex

- Weiterhin gibt es Operatoren für Join, Mengenoperatoren usw.

Beispiel: Für den Pfadausdruck A.B x, x.C y sind der logische Plan und einige physikalische Pläne angegeben.

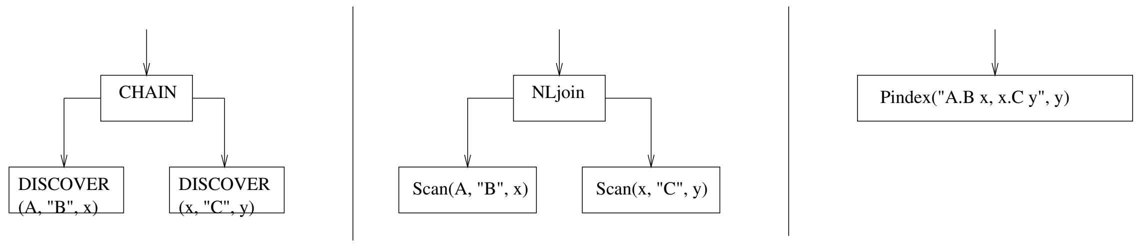

Abbildung 4: 2 Umsetzungen eines logischen Plans

9 Das Kostenmodell

Die Kosten der Ausführung jedes betrachteten Plans müssen zur Auswahl des vermutlich besten Plans möglichst genau abgeschätzt werden. Das hierfür notwendige Kostenmodell berücksichtigt in LORE die Ausführungskosten für Teilpläne und deren Kombination durch einen simplen rekursiven Ansatz: Für jeden Planknoten werden seine Ausführungskosten und die von den Kinderknoten erhaltenen Kosten addiert. Die Kosten für die Knotenausführung orientieren sich an der Zahl der erwarteten Objektzugriffe, d.h. der Anzahl der bei gegebener Variablenbindung bis zu einem Knoten im nächsten Schritt gebundenen Variablen. Dies wird mit I/O-Kosten bezeichnet. Im Fall gleicher I/O-Kosten für mehrere Pläne werden die sogenannten CPU-Kosten zum Vergleich hinzugezogen, die die Anzahl von CPU-Operationen bei der Ausführung angeben (nur grob, um eine Unterscheidung herbeizuführen).

Beispiel:

- Objektzugriffe: Gegeben Variablenbindung bis zu bestimmtem Knoten, schätze Anzahl Objekte, die im nächsten Schritt gebunden werden.
  Für A.B x, x.C y , x gebunden, entspricht dies gerade dem Fanout für C-markierte Subelemente

- Kostenzuweisung für BS(x, l, y) (Implementierung durch extendible hashing):
  - I/O: 2 + Fanin(Pfad(y), l)
  - CPU: CPUHashCost + Fanin(Pfad(y), l)*CPUCompCost
- Kostenzuweisung für Pindex(p):
  - I/O: Länge(p) + $\mid p \mid$
  - CPU: Länge(p)

10 Algorithmen zur Suchraumbegrenzung

Für fast alle Anfragen wird das komplette Durchsuchen des Plansuchraumes wegen der exponentiellen Größe nicht möglich sein. Deshalb stellt LORE verschiedene Heuristiken zur Verfügung, um den Suchraum einzuschränken. Ein Plan wird hauptsächlich charakterisiert durch drei Punkte:

- die Reihenfolge der Schritte
- der Zuweisung einer Zugriffsstrategie für jeden Schritt
- der Zuweisung von join-Methoden, die die Zugriffsmethoden verbinden

Ziel ist es, den Plansuchraum möglichst stark einzuschränken, ohne gute Ergebnisse auszuschließen. Alle Algorithmen erhalten als Eingabe einen Pfadausdruck als Liste von Schritten. Bei einer Ausdrucklänge n, m möglichen Zugriffsmethoden und j Joinmethoden ergibt sich ein Suchraum der Größe $n!m^n j^{(n-1)}$, wobei $n!$ für die Permutationen der Reihenfolge, m^n für die Zuweisung der Zugriffsmethoden für alle n Schritte, und $j^{(n-1)}$ für die Verknüpfung der Schritte durch ein Join steht. Die vollständige Suche kann nur für kleine Anfragen als Vergleichsmaßstab für die Qualität der Planauswahl der anderen Algorithmen herangezogen werden. Da nicht alle Pläne auch gültig sind (z.B. wegen nötiger Variablenbindungen), wird die Gültigkeit der erzeugten Pläne von den Algorithmen geprüft. Folgende Algorithmen sind in LORE implementiert (in zunehmendem Maß der Einschränkung):

0 VOLLSTÄNDIG: Durchsucht den gesamten Suchraum,
Komplexität $n!m^n j^{(n-1)}$

1 SEMIVOLLSTÄNDIG: Betrachtet weiterhin alle Schrittpermutationen und Kombinationen von Joins, weist aber für jede solche Kombination die möglichen Zugriffsmethoden nach einer lokal optimalen Entscheidung (*greedy*) zu, indem in einem linearen Durchlauf durch die Schritte jeder Schrittpermutation die zu diesem Zeitpunkt günstigste Methode in Abhängigkeit von den davor bereits erzeugten Variablenbindungen gewählt wird, wodurch der Faktor m^n für die Zugriffskombinationen entfällt, die Komplexität ist
$n!j^{(n-1)}$

2 EXPONENTIELL: Betrachtet nur eine Teilmenge der Schrittreihenfolgen, indem für jede übergebene Schrittliste nur zwei Planreihenfolgen berücksichtigt werden: Im ersten Plan wird rekursiv der beste Plan für die ersten n-1 Schritte ausgewählt und danach die beste Wahl für den n-ten Schritt und die Kombination der beiden gewählt, im zweiten Plan wird zuerst der letzte Schritt und dann die ersten n-1 Schritte betrachtet. Dadurch entfällt der $n!$-Term für die Schrittpermutationen, und durch die lokal optimale Entscheidung für Zugriffe und Joins sinkt die Komplexität auf die Anzahl der betrachteten Reihenfolgen, $2^{(n-1)}$

3 POLYNOMIELL: Hier erfolgt sowohl die Zuweisung der Zugriffsmethoden und der Joins, als auch die Festlegung der Schrittreihenfolge durch lokal optimale Entscheidungen (*komplett greedy*), wobei ausgehend von einem leeren Plan und der Schrittliste in einer Schleife

in jedem Durchlauf ein Schritt hinzugefügt wird ($\Rightarrow$ n Durchläufe) indem für die noch nicht betrachtetenen Schritte ($\Rightarrow$ jeweils i Durchläufe, i von n bis 1) geprüft wird, welcher bei Hinzunahme mit den lokal optimalen Methoden den günstigsten Plan erzeugt, bis der gesamte Plan komplett ist. Durch die beiden Schleifen ist die Komplexität die Summe der Durchläufe der inneren Schleife,
$O(n^2)$

4 ES-START: Da der ExtentScan keine Variablenbindung benötigt, sind hierfür beliebige Einstiegspunkte in den Pfadausdruck möglich. Nach Wahl dieser Einstiegspunkte werden benachbarte Punkte jeweils entweder komplett mit ForwardScan oder mit BackwardScan verbunden, abhängig von den günstigeren Kosten (Entscheidung in Linearzeit). Die Einstiegspunkte, deren Qualität über die Performance entscheidet, werden gewählt, indem man die Pfadausdruckschritte in aufsteigender Zahl ihrer Vorkommen sortiert (damit stehen Namen vorne, obwohl man für sie eigentlich kein ES benötigt), und dann in dieser Reihenfolge solange Schritte als Startpunkte hinzufügt, bis das Verhältnis zur letzten Zahl der Vorkommen einen bestimmten Grenzwert überschreitet. Durch die Sortierung ergibt sich eine Komplexität von $O(nlogn)$

5 BRANCHES: Zerlegt den Pfadausdruck in die linearen Teilausdrücke, optimiert jeden Zweig einzeln, und kombiniert optimale Teilpläne in der Reihenfolge ihrer Kosten mit der jeweils besten Methode. Die Komplexität hängt von dem zur Teilplanerzeugung benutzten Algorithmus ab (in LORE wird Algorithmmus Polynomiell benutzt). Ein Nachteil dieser Methode ist, daß für jeden Zweig außer dem, der den Namen enthält, mindestens einmal ES benutzt werden muß, um eine erste Bindung zu erzeugen.

6 SIMPLE: Betrachtet den kleinsten Suchraum, indem die Schrittreihenfolge einfach durch eine Schrittsortierung nach Anzahl ihrer Vorkommen festgelegt wird, und Zugriffs- und Join-Methoden lokal optimal nach den jeweils vorhandenen Bindungen zugeordnet werden.
Die Sortierung dominiert damit die Komplexität $O(nlogn)$

11 Postoptimierungen

Viele der ausgewählten Pläne können u.U. dadurch verbessert werden, daß die Reihenfolge der Schritte geändert wird. LORE's Postoptimerungen erzeugen deshalb aus vollständigen Plänen äquivalente Pläne mit nichtgrößeren Kosten durch Umordnung der Zugriffsmethoden an günstigere Stellen. Es gibt zwei Strategien, nämlich das Verschieben ganzer Teilpfade und das Verschieben einzelner Schritte:

1. Ganze Teilpfade (Anzahl l) verschieben:

   - Verfahren A: Schätze Teilplankosten und füge den günstigsten, der nicht von noch nicht eingefügten Schritten abhängig ist, mit optimaler Zugriffsmethode zum neuen Plan hinzu, bis der Plan komplett ist. Man hat also l Durchläufe mit jeweils i Schritten (i von l bis 1) und damit Komplexität $O(l^2)$
   - Verfahren B: Betrachte alle Umordnungen, die erlaubt sind, durch die Permutationsanzahl ergibt sich die Komplexität $O(l!)$
     (aber l ist meistens relativ klein im Vergleich zu n, und viele Anordnungen sind wegen der Abhängigkeiten nicht erlaubt)

2. Einzelschritte vertauschen:

   - Verfahren C: entspricht A auf Schrittebene, Komplexität $O(n^2)$

- Verfahren D: entspricht B auf Schrittebene, Komplexität $O(n!)$
(aber durch die Abhängigkeiten der bereits zugewiesenen Zugriffsmethoen muß man viele Kombinationen nicht betrachten)

12 Einige Ergebnisse

Um die Leistunsgfähigkeit der vorgestellten Konzepte zu testen, wurde eine Datenbank über die Kinofilme eines Jahres aufgebaut, mit über 12.000 Einträgen für die Filme und weiteren Informationen über ihre Schauspieler, den Handlungsort und den Verleih.
Für einen relativen Vergleich der Algorithmen und deren Kombination mit Postoptimierungen wurden 8 verzweigende Pfadausdrücke benutzt (siehe Anhang).

Experiment 1:

Algorithmus	1	2	2A	2B	2C	2D	3	3A	3B	3C	3D	4	5	6
Ausführung	11	14	9	10	3	1	6	8	7	5	3	2	13	12
Gesamtzeit	14	13	9	11	5	11	2	4	6	2	9	1	7	8

Es zeigt sich, daß Algorithmus 4 ohne Postoptimierungen insgesamt gut abschneidet, während Algorithmus 1 nicht nur in der Optimierung, sondern auch in der Ausführung sehr langsam ist. Algorithmus 2 ist zwar in der Ausführung am langsamsten, diese kann allerdings mittels Postoptimierung D schneller als alle anderen gemacht werden.
Für einen festen Ausdruck (siehe Anhang) werden auch einige absolute Werte angegeben:

Experiment 2:

Rang	Algorithmus	Optimierung	Ausführung	Gesamt
1	6	0.0741	0.0729	0.147
2	4	0.104	0.127	0.231
3	3	0.1108	0.136	0.247
4	5	0.085	1.241	1.326
5	2	0.26	1.996	2.256
6	1	174.749	1.38	176.129

Hier schneidet der einfachste Algorithmus sowohl in der Optimierung als auch in der Ausführung am besten ab, was bei dem kleinen durchsuchten Planraum eher überraschend ist.

Ein typisches Problem bei starker Benutzung von ES, z.B. im Branches-Algorithmus, ist, daß etwa bei einem Ausdruck, der für Verleihläden ihren Ort abfragt, also s.Location enthält (ungefähr 256 Einträge in der Datenbank), auch alle Einträge für Film-Orte (etwa 12.000 Einträge) berücksichtigt werden.
Experiment 3:

Algorithmus	Optimierung	Ausführung
3	0.071	0.312
2	0.144	0.312
5	0.111	7.122

Für die gegebenen Experimente noch der Laufzeitfaktor der Algorithmen im Vergleich zum vollständigen Algorithmus (=1.0) im Durchschnitt.

1	2A	2B	2C	2D	3A	3B	3C	3D	4	5	6
1.23	1.35	1.12	2.38	1.08	2.19	1.26	2.30	1.29	2.05	2.25	2.60

13 Fazit

Zur Optimierung von Queries aus Pfadausdrücken über semistrukturierten Daten wurden ein neues Statistikkonzept, neue Arten graphbasierter Indizes sowie Heuristiken zur Suchraumbegrenzung vorgestellt. Alle vorgestellten Ideen sind in LORE implementiert. Die Algorithmen sind als repräsentative Beispiele bzw. als Ansatz zu einer weiteren Entwicklung zu verstehen. Ihre Effizienz schwankt sehr stark abhängig von Datenbankstruktur und Pfadausdruck. Hier kann durch gute oder schlechte Auswahl das eine oder andere Extrem erreicht werden. Über die Implementierung der Statistiken wurde leider keine Aussage gemacht. Insgesamt scheint die Entwicklung noch sehr stark in Bewegung und vieles noch nicht ausreichend untersucht, zumal XML-Anwendungen und Datenbanken in der Praxis noch eher selten sind und deshalb nur wenige Werte zur Verfügung stehen, was z.B. die Praxisnähe der Beipieldatenbank und der Testausdrücke angeht.

14 Anhang

14.1 Grundlagen der Experimente

Die acht Pfadausdrücke für Experiment 1:

- DB.Movies x, x.Movie m, m.Actor a, m.AvailableAt t
- DB.People x, x.Person o, p.Name n, p.Phone z, p.Likes l, l.Thing t
- DB.People x, x.Person p, p.Likes l, l.Thing t2, p.Dislikes d, d.Thing t1
- DB.Stores x, x.Store s, s.Name n, s.Location l, l.City c
- DB.Movies x, x.Movie m, m.Sequel s, s.AvailableAt a, a.OwnedBy o, o.Affiliated f, f.Name n
- DB.Movies x, x.Movie m, m.Sequel s, s.AvailableAt a, a.OwnedBy o, o.Affiliated f, f.Phone p
- DB.Movies x, x.Movie m, m.Actor a, a.Likes l, l.Thing t, a.Adress d, m.Title z
- DB.Companies x, x.Company c, c.Affiliated a, x.Name n

Der Pfadausdruck für Experiment 2:
DB.People x, x.Person p, p.Name n, p.Phone h, p.likes l, l.Thing t

Der Pfadausdruck für Experiment 3:
DB.Stores x, x.Store s, s.Name n, s.Location l, l.City c

14.2 Quellen

Die maßgeblichen zugrundeliegenden Paper sind:

- IEEE Computer Society, Bulletin of the Technical Committee on Data Engineering Sep.99 Vol.22
- J. McHugh, J. Widom: Query Optimization for XML (+ Appendix). Very Large Databases Conference (VLDB) 1999, pp. 315 - 326.
- J. McHugh, J. Widom: Optimizing Branching Path Expressions. Technical Report 06/1999, Stanford University.
- J. McHugh, J. Widom: Compile-Time Path Expansion in Lore. Technical Report 1999, Stanford University.
- J. McHugh, J. Widom: From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. Technical Report 1999, Stanford University.

Informationen über das Lore - Projekt unter http://www-db.stanford.edu/lore