Universität des Saarlandes, SS 2003

Seminar „Informationsorganisation und –suche mit XML“

XML Systems & Benchmarks

01.07.2003

Christoph Staudt Peter Chiv

Dozent
Ralf Schenkel

Betreuer
Sergej Sizov

Inhaltsverzeichnis

Seminar Teil I: XML Datenbanksysteme

1. Motivation

Wie kommen die Konzepte von XML und Datenbanken zusammen?
XML setzt sich zunehmend als „die“ Sprache zum Datenaustausch zwischen Systemen durch. Daher entstehen immer grössere Datensammlungen die Effizient verwaltet, durchsucht und gepflegt werden können müssen
Im vergleich zu anderen Dokumenten, haben XML Dokumente Graph-Format. D.h. es gibt eine hierarchische Ordnung der einzelnen Elemente innerhalb eines Dokuments. Um den Datenretrieval Anforderungen aus XML Dokumenten heraus unter der Verwendung der Semistrukturellen Eigenschaften dieser gerecht zu werden wurde XQuery vom W3C Working Draft, am 7 Juni 2001 in der Version 1.0 standardisiert

Was die Herangehensweise an die Kombinierung der beiden Konzepte angeht gibt es zwei Konzepte.
Eine Variante ist es XML Daten so zu verändern, dass sie in relationalen Datenbanken abgespeichert werden können.
Auf der anderen Seite stehen die nativen XML Datenbanksysteme.

2. Zusammenfassung des Konzeptes relationaler DBS:

Sie sind flache Datenstrukturen als eine Menge von Tupeln zusammengesetzt aus einer Menge von primitiven Datentypen.
Arbeiten auf primitiven Datentypen heisst, dass alles andere Bildungen aus diesen sind. Somit können komplexe Datentypen wie XML Bäume nicht direkt ohne Modifikation abgebildet werden.

Dokumente im XML Format sind semistrukturierte mit tags versehene Textfiles.
Was bedeutet semistrukuturiert in diesem Zusammenhang?
- Die Daten sind unregelmässig und enthalten Schema Informationen verschlüsselt mit den eigentlichen Daten, da die Tags und der Dateninhalt nicht voneinander getrennt werden.
- Es kann passieren, dass ein Attribut entweder gar nicht, oder zusätzlich auftritt.
- Wie in unserem Beispielbaum (Abb.1) zu sehen ist, können einzelne Attribute eines Elementes auch mehrfach auftreten (Ein faculty member kann mehrere research assistants haben)

3. Native XML Datenbanken im Detail

Nativ in diesem Zusammenhang bedeutet dass man auf Relationen verzichtet und XML Dokumente als ganzes abspeichert. Als Marktführer im Kommerziellen Umfeld ist das System Tamino zu nennen. Wegen seines kommerziellen Charakters wenig über Hintergründe und Implementierung bekannt
Es gibt eine Vielzahl unterschiedlicher Systeme, z.B. Oracle, Caché oder prototypische. Einige sind Open Source Projekte, andere kommerziell.

Wir können unsere Ausführungen hier nur anhand eines Open Source Systems vornehmen aufgrund der Einblicke in die Implementierungshintergründe. Im Folgenden werden wir uns daher auf das System Timber beziehen.

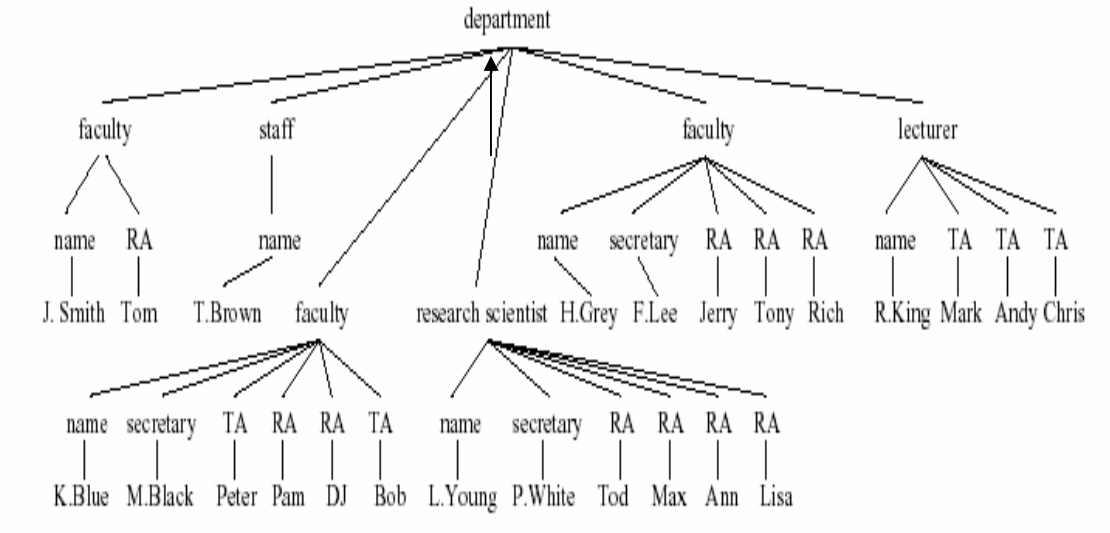


Abb.1

4. Architektur des Timber Systems

Wir werden jetzt auf einzelne Punkte der Systemarchitektur genauer eingehen, und Zusammenhänge erläutern
Die Grafik in Abb. 2 zeigt die Gesamtarchtektur des Timber Systems.
Da er bei allen Vorgängen eine wichtige Rolle spielt werden wir uns als erstes den Storage Manager genauer ansehen.
Danach werden wir 2 Prozeduren im Datenbanksystem genauer betrachten.
Als ersten Vorgang werden wir uns das Laden der XML Datenbank ansehen und somit die Funktionen des Data Parsers bzw. Data Managers betrachten.
Danach werden wir eine XQuery Anfrage durch das System verfolgen.

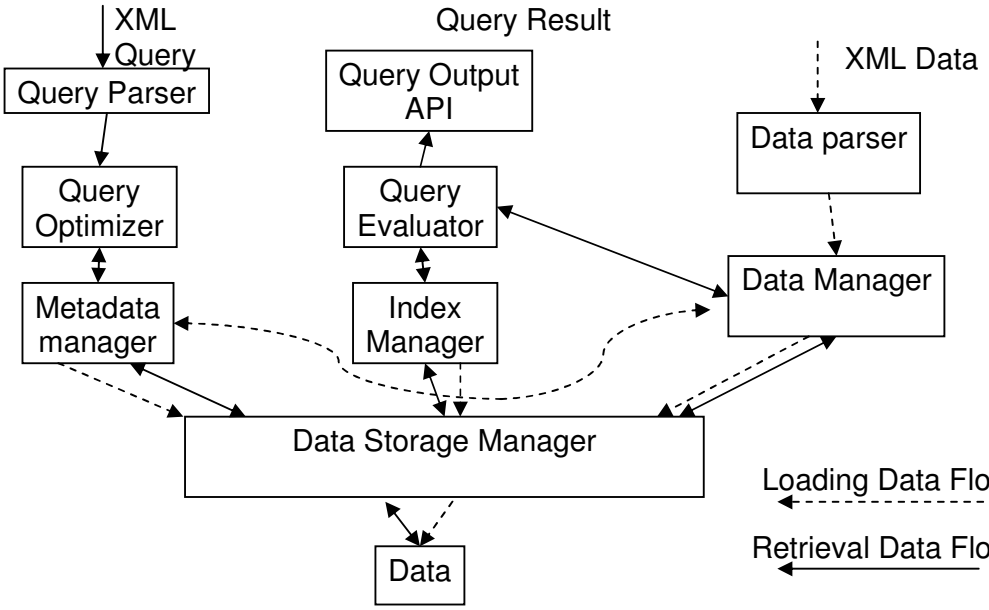


Abb.2

4.1 Data Storage Manager:

Timber nutzt als Backend für die Speicherverwaltung Shore. Shore ist eine sehr generische gehaltene Sammlung von Algorithmen die als Prototyp für Datenbanken genutzt werden können. Sie sind generisch genug um den Entwickler nicht in das relationale Schema zu zwingen. Shore ist ein Open Source Projekt

4.2 Füllen der Datenbank:

Beim Füllen der Datenbank nimmt der „Data Parser“ ein XML Dokument als Input und generiert daraus einen Parse Tree. Der „Data Manager“ nimmt jeden Knoten dieses Parse Trees so wie er ihn bekommt, transformiert ihn Schritt für Schritt in die interne Repräsentation und legt ihn dann in Shore als atomare Speichereinheit ab.
Was genau bedeutet interne Repräsentation?

- Es wird ein Knoten pro Element angelegt
- Es wird ein Kindknoten für jedes Subelement angelegt
- Alle Attribute des Elementes werden zu einem Kindknoten zusammengefasst
- Der Content eines Elementes wird ebenfalls in einen Kindknoten ausgelagert

Es bleibt zu bemerken, dass Verarbeitungsanweisungen, Kommentare und ähnliches einfach ignoriert werden. Für zukünftige Versionen ist geplant diese ebenfalls zu speichern.

Wir haben jetzt gesehen wie die Daten konkret gespeichert werden schauen wir uns nun an wie in den gespeicherten Daten weiterhin sinnvoll navigiert werden kann.
Das Feststellen von Parent-Child und Ancestor-Descendant Beziehungen spielt eine absolut entscheidende Rolle. Dies werden wir insbesondere später noch bei der Besprechung des structural Joins sehen. Wenn man die Labels der einzelnen Knoten geschickt wählt kann dies ohne grossen Aufwand errechnet werden. Es ist möglich jedem Knoten ein numerisches Start und Endlabel zu geben so dass jeder nachfolgende Knoten mit seinem Intervall komplett innerhalb des Intervalls seines Vorgängers liegt. Also im Prinzip eine Trichterform

Wenn man dann noch die Level Information aufnimmt, also die Tiefe des Knotens im Baum, dann können auch Parent-Child Beziehungen einfach überprüft werden.

A-D Relationship: (S1,E1,L1) is Ancest. Of (S2,E2,L2) iff (S1<S2 AND E1 > E2.
P-C Relationship: (S1,E1,L1) ist Par of (S2,E2,L2) iff S1 < S2 AND E1> E2 AND L1=L2 -1

Updates können dafür sorgen, dass beim Einfügen von Knoten unter Umständen der komplette Baum mit neuen Labels versehen werden muss um konsistent zu bleiben. Eine Möglichkeit diesem Problem zumindest teilweise vorzubeugen ist, indem man Abstände zwischen aufeinander folgenden Labels lässt.
Ein weiterer Positiver Seiteneffekt dieser Labeling Strategie ist, dass das Tupel wunderbar als ein Ersatz für das fungiert was bei relationalen Systemen die eindeutigen record identifier sind.

Index Abfragen geben in relationalen Datenbanksystemen Listen von RIDs als Ergebnisse zurück. Dementsprechend geben sie Listen von start, end und level Labels in einer XML Datenbank zurück
Der index ist ein B* Baum der genau so sortiert ist wie in einer relationalen Datenbank. Es soll schneller Zugriff auf vorgegebene Bereiche ermöglicht werden.

Es ist hier noch der Umgang des Timber Systems mit DTD oder Schema Informationen zu erwähnen. Das System ist darauf ausgelegt auch ohne die Anwesenheit solcher Informationen gute Performance zu liefern. Aktuell werden solche Informationen – falls vorhanden – ignoriert. Für zukünftige Versionen ist geplant, dass falls DTD oder Schema Informationen vorliegen dies zur Verbesserung der Performance genutzt werden.

5. Verarbeitung einer Query in Timber

Als erstes werden die XML Queries die in der Sprache XQuery vorliegen vom Parser in einen Algebraoperatoren Baum übersetzt.Die dazu verwendete Baumalgebra stellen wir sofort vor nachdem wir den groben weiteren Weg der Query durch das Timber System betrachtet haben:

Der Query Optimizer reorganisiert auf basis von Regelsatz, metadata infos, und macht die benötigten umwandlungen von logischen in physikalische operatoren.
Der sich daraus ergebende Ausführungsplan wird vom Query Evaluator ausgewertet. Dieser bedient sich dabei der Funktionen des Data- Indexmanager, diese können nun Operationen auf Mengen von Bäumen ausführen und rufen ihrerseits Shore – das bereits erwähnte Speicherbackend auf. Was genau das für Operationen sind, werden wir uns jetzt sofort ansehen.

5.1 Tree Algebra

Wie wir bereits gesehen haben haben XML Dokumente im weitesten Sinne eine Baumstruktur haben. Zumindest beschäftigt sich Timber mit XML Dokumenten als Bäume und lässt XPointer (Links) ausser Acht. (Hier wird vereinfachend angenommen das XML Dokumente immer Bäume sind.)
Eine Kernanforderung an die Algebra ist dass sie auf Mengen von geordneten Bäumen arbeiten kann.
Timber benutzt für diese Aufgabe die Algebra TAX, welche die standardoperationen wie Selektion, Projektion, kartesisches Produkt, etc. Enthält.
Wir werden im Folgenden auf den Umganb der Algebran mit den heterogenen Daten eines XML Dokuments näher betrachten sowie die Operation der Selektion und Projektion:

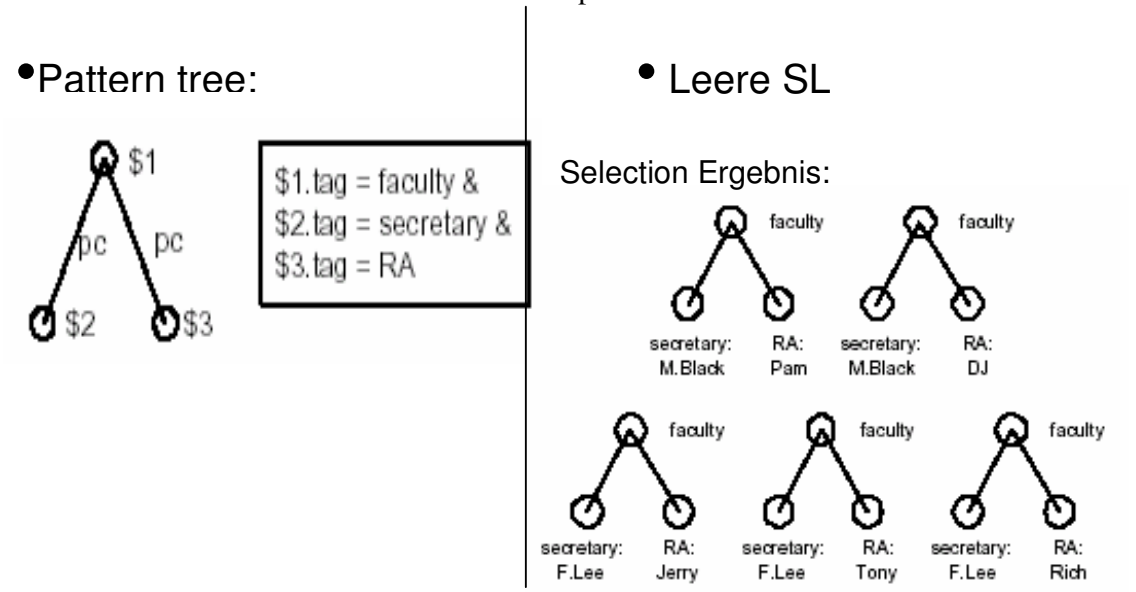
5.1.1 Umgang mit Heterogenität von XML Dokumenten

In einer Relation hat jedes Tupel die gleiche Struktur. Man kann also aus einer Menge von Tupeln jeden einzelnen Teil eines Tupels eindeutig über seine Position ansprechen. Bäume hingegen haben eine kpmlexere Struktur, in XML können Subelemente weiterhin mehrfach oder gar nicht auftreten.
Das bedeutet man kann Knoten nicht eindeutig über ihre Adresse im Baum ansprechen

Um dieses Problem in den Griff zu bekommen arbeitet TIMBER mit Pattern Trees. Es wird der Gesamte XML Baum nach einem gewissen Muster durchsucht. Für jeden Fundort wird ein Witness Tree zurück gegeben. Der Name leuchtet ein, da jeder dieser Witness Trees Zeuge eines Vorkommens des Musters aus dem Pattern Tree ist.
Muster die der Pattern Tree erkennt sind alle Homogen. Somit können Knoten in den Witness Trees eindeutig angesprochen werden. Es wird also von der gesamten Heterogenen Struktur auf eine homogene Teilstruktur abstrahiert auf der gearbeitet werden kann.

5.1.2 Selektion:

Wie auch in relationalen Datenbanken ist die Selektion dazu da, aus einer Sammlung von Bäumen diejenigen zurückzugeben, die eine bestimmte Bedingung erfüllen.
Als Argumente nimmt die Selektion eine Sammlung von Bäumen, einen Pattern tree P sowie eine Selection List SL. Die SL kann Wildcards enthalten. Die Funktionsweise der Selektion wird am besten anhand eines Beispiels klar:



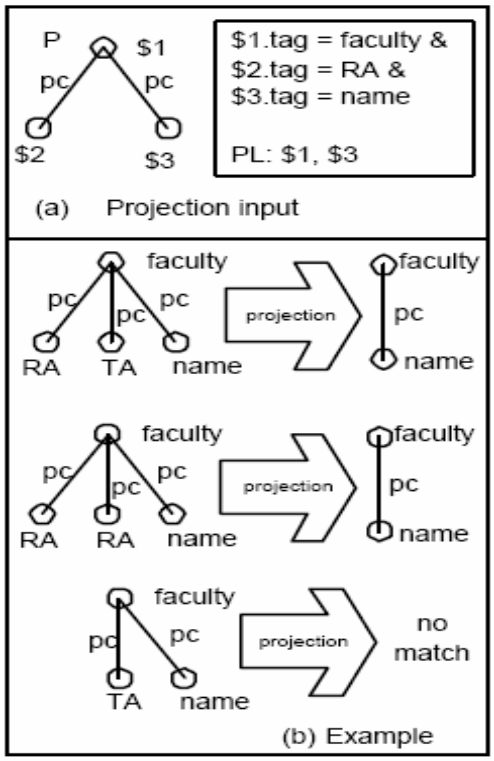
An diesem Beispiel lässt sich auch gut einer der Hauptunterschiede zwischen der Selektion im relationalen Sinne und der im Sinne unserer XML Datenbank darstellen.
Was man mit relationalem Hintergrund als Ergebnis erwarten könnte wäre, dass das Ergebnis die beiden betreffenden faculty Mitglieder (K.Blue, H.Grey) und evtl. deren Subbäume wären. In der tree Algebra wird nicht einfach wie in der relatonalen Algebra der Input bei einem select gefiltert. In einer Tree Algebra werden die matchenden Teile des Inputbaums identifiziert. Sollten mehrere Matches auftreten, werden diese alle aufgelistet.

5.1.3 Projektion

Wenn man Bäume betrachtet kann man sich Projektion vorstellen als die Beseitigung von Knoten die nicht gewünscht sind.
Als Argumente werden ihr übergeben: Eine Sammlung von Bäumen auf die die Projektion angewendet werden soll, wie auch schon bei der Selektion ein Pattern tree und eine Projection list. Diese besteht auf einer Auflistung von Knotenlabels die in P vorkommen, unter Umständen ergänzt durch *. D.h. das von dem Betreffenden Knoten

auch der Unterbaum im Output erscheinen wird. Bemerkenswert ist, dass ein einziger Baum im Output zu keinem, einen oder mehreren Output Bäumen führen kann.

Zu dem Vergleich mit der relationalen Projektion: Im relationalen Sinne sind Projektion und Selektion offensichtlich und zu 100% orthogonale Operationen, da man sie sich sogar bildlich mit hilfe von Spalten und Zeilen als auf Senkrecht zueinander stehenden Datenbeständen operierend vorstellen kann. Im Baumumfeld ist dies nicht mehr so klar zu sehen. Dennoch bleiben Projektion und Slektion 100% eigenständige und grundverschiedend Operationen.



In Bild a sieht man die Projektionsbedingungen.
In b sieht man die Bedingungen aus a angewendet auf 3 verschiedene Bäme.
Der erste faculty member hat einen RA und einen TA und einen Namen. Das Ergebnis ist wie man es erwarten würde ein Witnesstree.
Der 2. faculty member hat 2 Ras, und deshalb würde er 2 Witnesstrees produzieren, die den spezifizierten patterntree matchen würden. Die Projezierten Elemente in den Witness Trees sind jeweils die gleichen. Duplicate Elimination sorgt bei der Projektion dafür, dass nur ein tree im Output erscheint.

5.2 Query Optimierung

Wenn man mit Joins arbeitet so wächst die Anzahl der möglichkeiten zur Ausführung dieser was die Reihenfolge angeht in exponentieller geschwindigkeit der Argumente.
Die Kosten der einzelnen Pläne können dabei sehr unterschiedlich sein. Aus diesem Grund listet der Query Optimizer erst alle möglichkeiten für Ausführungspläne auf, schätzt dann die Kosten jedes einzelnen dieser Pläne ab, und wählt dann denjenigen mit den niedrigsten geschätzten kosten. Dem Paper zufolge sind solche Kostenschätzungen bisher aber noch deutlich verbesserungswürdig und unzuverlässig.

Result Size estimation. Warum ist dies wichtig? Nun um den Sinnvollsten Query Plan auswählen zu können ist es unabdingbar dass man weis wie gross, das Endergebnis einer Query ist, aber auch schon wie gross Zwischenergebnisse sind. Eine mögliche Anwendung ist auch im Internetbereich. Man kann dem user bereits vor der eigentlichen Ausführung sagen wie gross sein Ergebnis ca. Werden wird.
Schauen wir uns ein Beispiel an: Wir suchen Faculties und TAs die in Vater-Kind Beziehung zueinander stehen. Wie wir aus unserer Grafik sehen können gibt es 3 faculties und 5 TA Knoten. Da uns hier kein Schema vorliegt müssen wir davon ausgehen dass jede Kombination aus Faculties und TAs möglich ist. Das ergäbe hier 3 x 5 also 15 Möglichkeiten.

Verfeinerungen lassen sich durch den Gebrauch von **Position Histogrammen** ermitteln

0	1
2	

0	3
3	

Links-oben: Knoten, die sich von der ersten in die zweite Hälfte erstrecken.
Recht-unten: Immer leer, da es nicht sein kann, dass sich Knoten aus der zweiten in die erste Hälfte erstrecken.

Die 55 Knoten aus dem Beispielbaum werden entsprechend auf die beiden Felder verteilt. Es kann nicht sein, dass TAs aus der zweiten Hälfte für facultys aus der ersten Hälfte arbeiten und andersrum. Von daher ergibt sich eine Obergrenze von (2x2 + 1x3) = 7 anstatt der Obergrenze von 15, wenn man keine Strukturinformationen heranzieht.

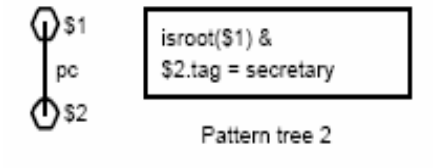
Mittels Verfeinerung der Histogramme lassen sich genauere Ergebnisse erzielen.

5.3 Query Auswertung

Wie bereits bei der Behandlung der logischen Algebraoperationen gesehen spielt das Pattern tree matching eine Zentrale Rolle.
Um es nochmal in Erinnerung zu rufen: Mit Ihrer Hilfe werden die Knoten eines Baums identifiziert die für die jeweilige Operation von Interesse sind. Insbesondere bei algebraischen Operationen kann man beobachten, dass mehrere Operatoren den gleichen Pattern tree benutzen. Diesen jedes Mal neu auszuwerten wäre sinnlos.

5.3.1 Pattern tree Reuse

Die wiederverwendung von Matchergebnissen erreicht man indem man knoten innrhalb des Baums sog. PIDWIDs anhängt. Wenn nun ein match eines Pattern trees ausgeführt wird werden die betreffenden Knoten mit der nummer des Pattern trees (PID) und der nummer des Knotens innerhalb des Pattern Trees (WID) gekennzeichnet.
Dann kann zu beispiel eine Selectionsoperation auf Knoten $3 des Pattern Trees 2 ausgeführt werden in dem man sie auf alle Knoten im Input anwendet die die PIDWID(2,3) haben.



An dieser Stelle wollen wir auf ein Problem der PIDWIDs eingehen, da Operationen ja auch Datenstrukturen ändern.
Wenn wir den hier angegebenen Pattern tree auf die Ursprüngliche Datenbank anwenden ist das Ergibnis leer.
Wenn wir aber erst eine Selektion ausführen welche alle Faculties und ihre Kindkonten zurücliefert wird das Ergebnis der Anwendung des abgebildeten Pattern Trees jeder Sekretär in der Datenbank sein.
Auf dieses Problem wird in den und zu verfügung stehenden Papers nicht eingegangen.

5.3.2 Node Materialization

Wie auch in relationalen Datenbanksystemen können in nativen XML Systemen viele Operationen ausgeführt werden ohne dass man dafür die eigentlichen Daten benötigt. Sie können direkt auf Indexstrukturen ausgeführt werden.Im Fall von XML kann man die Baumstruktur so kodieren dasss recht komplexe Operationen ausgeführt werden können ohne die eigentlichen Daten anzusehen. Auf der anderen Seite ist es in Relationalen Datenbanken aufgrund der klaren Datenstruktur von Tupeln recht billig die eigentlichen Daten anzusehen. Anders in XML! Daher hat die Physikalisxche Algebra in XML einen eigenen Materialization Operator. Knoten mit denen man ohne die eigentlichen Daten anzuschauen operieren kann heissen unmaterialized
Nun aber was heisst es genau einen Knoten zu Materialisieren? Man könnte nur den Wert eines Attributes benötigen um z.B. zu gruppieren. Oder wir könnten die Daten eines Child subelements benötigen…
Optimal ist also wenn eine Teilweise materialisierung möglich ist.
Beispiel: Selection then projecttion with $4 as a Proj list. Nur Name muss materialisiert werden!

5.3.3 Structural Join Algorithmus

Was genau ist hier das Ziel?
Ich habe 2 Punkte und suche nach allen Teilpfaden die z.B. Einen best. Ziel- oder Startpunkt haben.
Die Wichtigkeit von pattern trees haben wir bereits gesehen. Jetzt sehen wird, dass jede Kante in einem pattern tree eine strukturelle Beziehung repräsentiert.
Der einfachste Weg Matches zu finden ist die komplette Datenbank zu durchsuchen. Allerdings ist ein kompletter Datenbankscan nicht das, was für eine simple Anfrage gewünscht wird.
Eine bessere Lösung ist es, Indices zu benutzen. Dabei wird mittels einem Index nach einem Knoten gesucht. Anschließend werden dessen Kindknoten nach den relevanten Matches durchsucht. Diese Methode kann trotzdem unter Umständen sehr teuer sein.
So kommen wir zum Struktural Join. Hier wird in 2 Phasen vorgegangen:

1. Es werden mit Hilfe aller zur Verfügung stehender Indizes Kandidatenknoten für Witness-Trees gesucht.
2. Im zweiten Schritt werden mit Hilfe des Structural Join Algorithmus Knoten zusammengebracht. Der Algorithmus arbeitet dabei mit 2 geordneten Listen und einem Stack über den die beiden Listen zusammengeführt werden.

Schauen wir uns auch dies anhand eines Beispiels an:

Wir suchen nun in unserer Datenbank faculties die einen sekretär haben der für sie arbeitet.
Das herangehen mit Hilfe eines navigational plan würde so aussehen dass man einen match für einen Knoten des Patterns versucht zu finden und von dort aus weiter „navigiert“ und versucht weitere Knoten des Musters zu errreichen. Wenn man nun also einen faculty node findet und von dort zu einem Child node secretary gelangt kann man diese beiden Knoten als witness tree zurückgeben.
Man kann sich leicht vorstellen dass dieses Vorgehen bei komplexeren Strukturen bzw. grossen Datenmengen leicht ausufern kann.
Das herangehen mit Hilfe eines Strukturellen Joins wäre wie folgt:
Mittels mehrerer verfügbarer Indices unabhängig alle möglichen Kandidaten zu suchen. D.h. jeder Knoten der eine Bedingung des Pattern Tree erfüllt wird in die engere Auswahl genommen. Danach werden mit einem structural Join mögliche Elter-Kind oder Vorgänger-Nachfolger Beziehungen ausfindig gemacht und so die Witness Trees „zusammengebaut“

Es werden listen mit matches für jeden der Knoten innerhalb des Pattern trees erstellt. Also hier eine Liste von 3 faculty nodes und eine liste von 3 secretary nodes. Dann würde der structural Join algorithmus versuchen diese zusammenzubringen.

5. Zusammenfassung:

In diesem Teil unseres Vortrages haben wir die zwei Herangehensweisen betrachtet die sich bieten wenn man XML-Dokumente in Datenbanken speichern will.
Wir sind dann tiefer in das native XML-DBS Timber eingestiegen und haben dort insbesondere Kernfeatures wie Pattern-Tree-Match oder den Strukurellen Join betrachtet.

Seminar Teil II: Benchmark

7. Motivation

- **Was bedeutet „Benchmark“?**

  Benchmarks dienen hauptsächlich als eine Skala für das Bewerten und Vergleichen von neuen Techniken und Systemkomponenten.

  Simpler und kürzer gesagt, vergleichen Benchmarks die Vor- und Nachteile unterschiedlicher Systeme.

- **Wieso sind „XML Benchmarks“ notwendig?**

  XML Datenbank Management Systeme (kurz DBMS) sind sowohl in ihrer Komplexität als auch in ihrer Kapazität gewachsen. Um Vergleiche in der Leistungsfähigkeit zwischen diesen DBMS zu ziehen, werden Benchmarks benötigt.

  Mittlerweile gibt es viele Vorschläge über die Art und Weise, wie man XML Daten physikalisch speichern kann. Z.B. könnte ein System das Konzept der sogenannten OIDs (IDs, die Informationen über die Struktur des XML Dokumentes speichern) benutzen, ein anderes wiederum verzichtet auf dieses Konzept. Jeder Vorschlag hat somit seine Vor- und Nachteile, welche sich in unterschiedliche Queryeigenschaften der Daten äußern. Diese können mithilfe von Benchmarks untersucht werden.

  Ein weiterer Grund, wenn nicht der wichtigste Grund überhaupt, für die Nachfrage nach XML Benchmarks kommt aus der Wirtschaft. In letzter Zeit haben immer mehr die physikalische Art und Weise XML Daten zu speichern und die Query Performance den Erfolg oder Misserfolg eines XML Datenbanksystems bestimmt. Mithilfe von Benchmarks lassen sich Erfolg und Misserfolg besser kalkulieren.

- **Wer profitiert von „Benchmarks“?**

  Auf der einen Seite haben wir die Datenbank Vertreiber. Sie benötigen Benchmarks zum Vergleich ihres Produktes mit anderen Produkten, um somit ihre Query Prozessoren verifizieren und verfeinern zu können.

  Aus Sicht des Kunden bzw. Anwenders lässt sich folgende wichtige Frage beantworten: Welches Produkt soll ich wählen?
  Diese Frage lässt sich wiederum verfeinern.
  - Wie hoch sind die Anschaffungskosten und wie hoch sind die Kosten für die Integration des Systems in meine Anwendungsumgebung?
  - Welches Produkt ist für mich und meine Ansprüche am besten geeignet?

  Benchmarks unterstützen den Anwender in der Beantwortung dieser Fragen.

  Auch Wissenschaftler von XML Datenbanksystemen können Nutzen aus Benchmarks ziehen. Die Informationen aus einem Benchmark können z.B. das Verfeinern oder

sogar Neudesign von Algorithmen anstoßen. Ein weiterer Punkt ist, dass mittels Benchmarks untersucht werden kann, inwiefern bestehende Technologien aus der relationalen Datenbankwelt in die von XML übernommen werden können.

- **Ziel dieses Ausarbeitungsteils**

Zur Zeit ist in der XML Welt vieles im Aufschwung und im Standardisierungsverfahren bzw. –suche. Bis jetzt gibt es noch keine einheitliche und endgültige Methodologie für das Bewerten der Unterschiede verschiedener XML Datenbanksysteme und somit übertragend auch für den Bereich der Benchmarks.

Von daher werden in diesem Paper nur allgemeine Punkte aufgezeigt, die XML Benchmarks beachten und abdecken sollten.

8. Anforderungen an den Benchmark

- **Vorüberlegungen**

In der Entwicklung eines XML Datenbank Benchmarks muss man sich anfangs mit der essentiellen Frage auseinandersetzen, welche Operationen auf XML Dokumente überhaupt vorstellbar sind. Im Weiteren stellt man sich die Frage, welche Operationen sinnvoll und geeignet sind, um mit diesen die Leistungsfähigkeit eines XML Datenbanksystems bewerten zu können. Die Operationen, die sich aus der Beantwortung der Frage ergeben, werden dann im Benchmark getestet.

Mit den oben genannten Fragen haben sich auch die Entwickler des XML Benchmarks XMark beschäftigt. Im Folgenden werden ihre Ergebnisse vorgestellt.

Im ersten Schritt resultierte eine Sammlung von 10 Anforderungen, die eine Grundlage für eine umfassende Analyse darstellen sollen, die alle Performance-kritischen Aspekte des XML Processings abdeckt.

Anhand der 10 Anforderungen wurde letztendlich der XML Benchmark XMark entwickelt. In XMark wurden diese 10 Anforderungen auf ein 14-Query-Konzept erweitert, welche in diesem Paper vorgestellt werden sollen.

Die oben genannten Performance-kritischen Aspekte des XML Processings lassen sich zu Ordnung, Typproblem, hierarchische Ordnung und das „lose“ Schema von XML zusammenfassen. Von daher beziehen sich die 14-Query Konzepte von XMark auf diese vier Gruppierungen.

9. XMark Benchmark: Einführung

Einführung

Der Benchmark XMark wurde am „National Research Institute for Mathematics and Computer Science (CWI)“ der Niederlande entwickelt.

Er bewertet die Retrieval Performance von XML Datenbanksystemen und Query Prozessoren. XMark stellt ein Rahmenwerk für die Einschätzung der Fähigkeiten einer XML Datenbank dar, mit vielen verschiedenen Querytypen (14-Query Konzepte) umgehen zu können.

Daten

Das Design eines XML Benchmarks erfordert eine modellierte Beispielsdatenbank, um das Verhalten von Queries vorhersehbar zu machen.

XMark modelliert dazu ein Internet Auktionshaus, wobei der Benchmark alle Daten innerhalb eines Dokumentes verwaltet. Die Datenbank besteht also aus nur einem Dokument, welches sich im Größenrahmen von 10 MB bis 10 GB skalieren lässt.

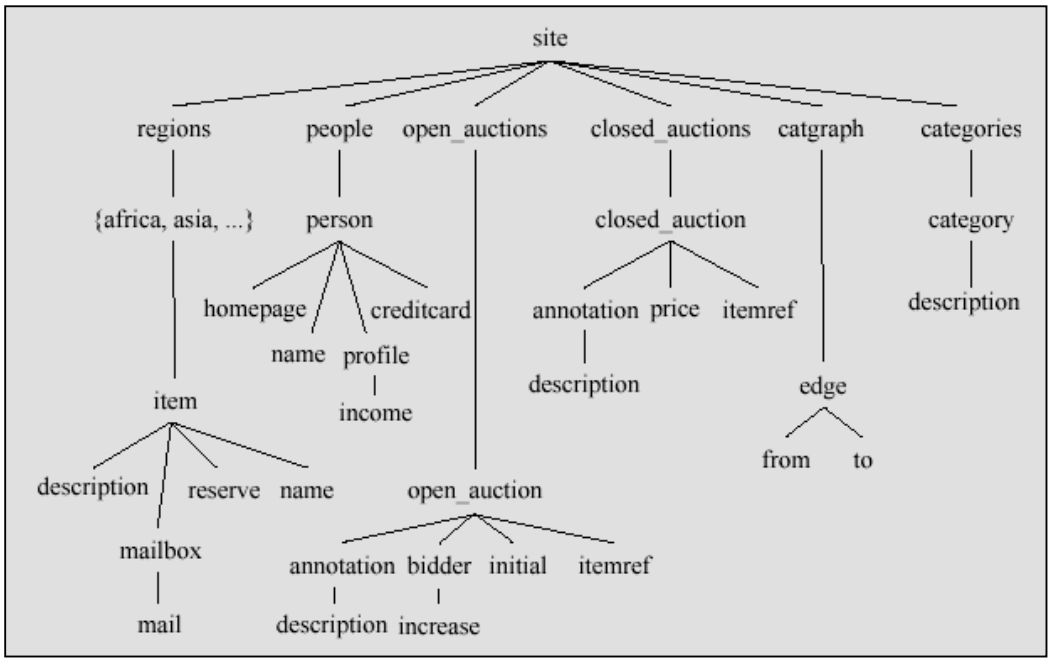


Bild 1: XMark Datenbank

Das Dokument weist Referenzen zwischen item, person, open_auctions und closed_auctions auf. Diese sind hier nicht eingezeichnet.

10. XMark Benchmark: 14-Query Konzepte

Im Folgenden werden die oben erwähnten 14 Konzepte vorgestellt, die alle Performance-kritischen Aspekte des XML Processings abdecken.

Wir werden hier nur genauer auf sechs von den 14 Konzepten näher eingehen. Die restlichen acht werden nur kurz vorgestellt.

- **Exact Match**

Für das Konzept des „Exact Match“ kommt folgende Query zum Einsatz:

Return the name of the person with ID ‚person0’.

Diese Query testet die Fähigkeit des Datenbanksystems eine einfache String Suche mit angegebenem Pfad zu bearbeiten.

Diese simple Query hat ein bestimmtes Hauptziel. Mit dem Resultat dieser Query soll nämlich eine Performance Basis für Bewertung der weiteren Konzepte gebildet werden.

- **Reconstruction aka Round Tripping**

Rekonstruktion bedeutet, dass das Datenbanksystem die zur physikalischen Speicherung herunter gebrochenen XML Daten wieder zum ganzen bzw. zu einem Fragment des Original XML Dokument zusammenfügt.

In der XML Welt ist Rekonstruktion eine häufige Operation. Gerade XML bietet dem User an, die Daten immer wieder zu verwenden. Denn mit Hilfe von XML Daten kann der Anwender ein und denselben Inhalt in verschiedene Layouts wie HTML Seiten, Brochüren etc. einbinden.

List the the names of items registered in Australia along with their descriptions.

Die Query verlangt also in diesem Fall nicht eine komplette Rekonstruktion des original Dokumentes, sondern (s. Bild 1) nur die Rekonstruktion des Unterbaums Item, die in der Region Australien registriert sind.

Wie kann es zu Unterschieden in der Queryauswertung kommen?

Im Falle eines relationalen Datenbanksystems wird die Operation Reconstruction sehr teuer sein, weil die XML Daten in Relationen bzw. Tupeln transformiert wurden. Bei der Rekonstruktion werden von daher viele teure Joins benötigt, um das Original XML Dokument wiederherzustellen.
Ein XML Datenbanksystems allerdings, das XML Dokumente nativ speichert, kann auf solche teuren Joins verzichten.

- **Path Traversals**

  Für das Konzept der Pfad Traversierung werden zwei Querys herangezogen.

  Print the keywords in emphasis in annotations of closed auctions.

  Weiter auf diese Query aufbauend ist die zweite Query.

  Return the IDs of the sellers of those auctions that have one or more keywords in emphasis.

  Was man also anhand Bild 1 sehen kann, ist dass beide Queries an Informationen interessiert sind, die in großer Tiefe im Baum zu finden sind. Von dem Datenbanksystem wird also eine lange Pfadtraversierung gefordert.

  Die beiden Queries geben Aufschluss darüber, wie hoch die Kosten für lange Pfadtraversierungen sind.

  Wie kann es hier zu Unterschieden in der Queryauswertung kommen?

  Ein XML Datenbanksystem, das XML Daten nativ speichert, müsste – ohne geeignete Methoden - sich durch das ganze Dokument arbeiten, um die Suche zu vollenden.
  Relationale Datenbanksysteme hingegen umgehen diese Schwierigkeit, da sie das XML Dokument in kleinere Teile (Relationen) aufteilen müssen, so dass diese kleinen Teile des XML Dokuments einzeln uns separat schneller durchsucht werden können.

- **Missing Elements**

  Die Query, die für das Konzept „fehlende Elemente“ ausgewählt wurde, lautet

  Which persons don’t have a homepage?

  Was diese Query bringen soll, ist offensichtlich. Da nicht jede Person eine Homepage besitzen muss, stellt sich also hier die Frage, wie das XML Datenbanksystem mit semi-strukturierten XML Daten umgehen kann.

  Nicht existierende Attribute werden in der Datenbank Welt beim physikalischen Abspeichern mit NULL Werten gekennzeichnet. Was hier also getestet wird, ist, wie effizient die Methoden des Datenbanksystems sind, die auf NULL Werte testen.

  NULL Werte sind häufig – wie in diesem Anwenderszenario – von großem Interesse.

- **Casting**

  Zum Thema „Casting“, also Typumwandlung, muss man anmerken, dass String der generische Datentyp in XML Dokumenten ist.

  How many sold items cost more than 40?

  Was vom Datenbanksystem bei der Querybearbeitung gefordert wird, ist ein Preisvergleich bzw. eine Suche nach Preisen größer als 40. Da die Preise im String

Format vorliegen, müssen diese dafür z.B. in den int Typen umgewandelt werden, um einen Vergleich möglich zu machen.

Diese Query testet also, wie effizient eine Typumwandlung ausgeführt wird.

- **Chasing References**

Referenzen sind ein integraler Teil von XML, mit denen man feinere Beziehungen als eine reine Hierarchie aufbauen kann.

Was die Queries aus diesem Konzept erfordern, ist den XML Baum nicht nur in die Tiefe zu traversieren, sondern auch entlang der Referenzen in der Horizontale.

1. Print the keywords in emphasis in annotations of closed auctions.

2. Return the IDs of the sellers of those auctions that have one or more keywords in emphasis.

Zieht man Bild 1 zur Hilfe sieht man, dass die erste Query also Verbindungen zwischen persons und closed_auction (Referenz zwischen persons und closed_auctions) beachten muss. Die zweite Query muss noch weitergehen, indem sie zusätzlich zu den Verbindungen aus der ersten Query noch Verbindungen zu items beachten muss.

Durch Referenzen in einer XML Datenbank ergeben sich zusätzliche zu bearbeitende Pfade. Wie effizient das Datenbanksystem dies erledigen kann, sollen die Ausführungen dieser beiden Queries zeigen.

Folgende Konzepte werden nur kurz vorgestellt. Für mehr Informationen verweisen wir auf „*XMark: A Benchmark for XML Data Management*." (siehe Literaturangaben).

- **Full Text**

Volltextsuche ist eine elementare Operation bei XML Anfragen. In diesem Konzept wird getestet, wie eine Volltextsuche in Form von Schlüsselwörtern bearbeitet wird.

- **Ordered Access**

Die Ordnung ist eines der wichtigsten Features in XML. Die Auswertung der Queries gibt einen Einblick darüber, wie das Datenbanksystem mit der Ordnung der XML Dokumente zurecht kommt und wie effizient es mit Queries mit Ordnungsbedingungen umgeht.

Der Query Optimizer sollte/muss den Aspekt der Ordnung beachten und wenn Ordnung keine Rolle spielt, diese vernachlässigen können.

- **Regular Path Expressions**

  Regular Path Expressions sind fundamental für fast alle Query Languages für XML Daten. Durch Queries aus diesem Konzept sieht man, wie der Query Prozessor Path Expressions optimiert und irrelevante Teilbäume außer Acht lässt.

- **‚Construction of Complex Results' und 'Joins on Values'**

  Diese beiden Konzepte beschäftigen sich mit der Fähigkeit des Datenbanksystems mit komplexen und großen Zwischen- und Endergebnisse umgehen zu können. Dies ist ein wichtiger Punkt, weil viele Applikation fordern, dass mit großen Datenvolumen umgegangen werden muss. U.a. zeigen die Queries auch, wie gut der Query Optimizer arbeitet, der den günstigsten Evaluierungsplan auswählen muss.

  Folgende Zahlen sollen einen Einblick in die Größendimension geben:
  - Bei der Query für das Konzept der komplexen Resultate erhält man beispielsweise bei einem 100 MB großen Dokument ein Resultat von 10 MB.
  - Die Query aus dem Konzept „Joins on Values" erfordert bei einem 100 MB großen Dokument zwischenzeitlich ein Join mit einer Größe von 12 Millionen Tupeln.

- **Function Application**

  Convert the currency of the reserves of all open auctions to another currency.

  Diese Query testet, inwiefern das Datenbanksystem mit benutzerdefinierten Funktionen zurechtkommt.

- **Sorting und Aggregation**

  Schließlich gibt es noch die beiden Konzepte Sortierung und Aggregation. Diese sind bereits wichtige bekannte Operationen aus der relationalen Datenbankwelt und werden ebenfalls in diesem XML Benchmark getestet.

11. Zusammenfassung: Teil II

Zusatzinformation

In dieser Ausarbeitung haben wir uns nur auf die 14-Query Konzepte des XMark Benchmarks konzentriert.

Zu Ausführungseinsätzen von XMark kann man folgende Aussage machen. XMark wurde (laut Paper *XMark: A Benchmark for XML Data Management*., siehe Literaturangaben) auf mehreren unterschiedlichen XML Datenbanksystemen ausgeführt. Darunter waren beispielsweise relationale und Hauptspeicher Datenbanksysteme. Nach Ende Experimente konnte man feststellen, dass es kein eindeutiges XML Datenbanksystem gab, das alle anderen mit seiner Leistungsfähigkeit überbieten konnte. Vielmehr waren die Top-Ergebnisse aller 14-Query Konzepte auf alle Testsysteme „gleichmäßig" verteilt.

Zusammenfassung

Im zweiten Teil dieser Ausarbeitung haben wir gesehen, was Benchmarks sind und wozu Benchmarks in der XML Welt benötig werden.

Die Frage, was ein Benchmark alles abdecken und testen sollte, führte uns zu anfänglich zu den allgemeinen 10 Anforderungen, die eine Grundlage für eine umfassende Analyse darstellen sollen, die alle Performance-kritischen Aspekte des XML Processings abdecken.

Im Weiteren wurde der Benchmark XMark vorgestellt und seine 14-Query Konzepte, welche die oben genannten 10 Anforderungen erweitern.

Anhang

Literatur

- H.V. Jagadish et al.: *Timber: A Native XML Database*. Technical report, University of Michigan, April 2002.
- Albrecht Schmidt et al.: *Why And How To Benchmark XML Databases*. In: SIGMOD Record 30(3), 2001.
- Albrecht Schmidt et al.: *XMark: A Benchmark for XML Data Management*. In: Proceedings of the 28th International Conference on Very Large Databases (VLDB), Hongkong, China, 2002.