WS 99/2000
Seminar: “XML-basierte Datenbanken”
“Speichern von XML-Daten”

Roger Dostert

23.11.1999

Inhaltsverzeichnis

1 Einleitung

Der erste Teil (Abschnitte 2 bis 5) dieses Dokuments beschreibt die Möglichkeit, XML-Dokumente als Bäume abzuspeichern. Diese Methode hat den Vorteil, daß die Dokumente durch einfache Baumtraversierung wiederhergestellt werden können. Als Quelle wurde hier [1] verwendet

Im zweiten Teil (Abschnitt 6) gehe ich auf einige Aspekte der XML-Unterstützung von Oracle 8i ein. Herangezogen wurde hier [2]

2 Begriffe

In diesem Abschnitt fasse ich die Begriffe zusammen, die ich im folgenden für das Verständnis für wichtig halte.

2.1 Logisches Modell

Das logische Modell entspricht der intuitiven Art, XML-Dokumente als Baum darzustellen. Die inneren deren Knoten sind mit den Tags und die Blattknoten mit den Daten zwischen den Tags des Dokuments beschriftet.

2.2 Physisches Modell

Beim physischen Modell handelt es sich um eine Erweiterung des logischen Modells. Dabei werden in den Baum zusätzliche Knoten eingefügt, die als Pointer dienen.

2.3 Knotenklassifikation

Je nach Kriterium lassen sich die Knoten der Modellbäume anderen Gruppen zuordnen. Hier unterscheide ich nach

- Inhalt:
  - Aggregations-Knoten (engl.: aggregate nodes): innere Knoten des Baums. Die Bezeichnung kommt daher, weil sie andere Knoten - ihre Kindknoten - zusammenfassen
  - Literal-Knoten (engl.: literal nodes) : alle Blattknoten. Sie enthalten uninterpretierte Bytestreams.
  - Proxy-Knoten (engl.: proxy nodes): sie verweisen auf andere Knoten und es gibt sie nur im physischen Modell. Ihr Gegenstück sind die Hilfsknoten (engl.: helper nodes)
- Modell :
  - Facette-Knoten (engl.: facete nodes): dies sind alle Knoten, die bereits im logischen Modell vorhanden sind.
  - Gerüst-Knoten (engl.: scaffolding nodes): die Knoten, die bei der Erweiterung auf das physische Modell eingefügt wurden

- Position im Baum :
  - Alleinstehende Knoten (engl.: standalone nodes): nur der Wurzelknoten des Baums
  - Eingebette Knoten (engl.: embedded nodes): die restlichen Knoten

2.4 Beispiel

Zur Verdeutlichung obiger Begriffe soll folgende Abbildung dienen :

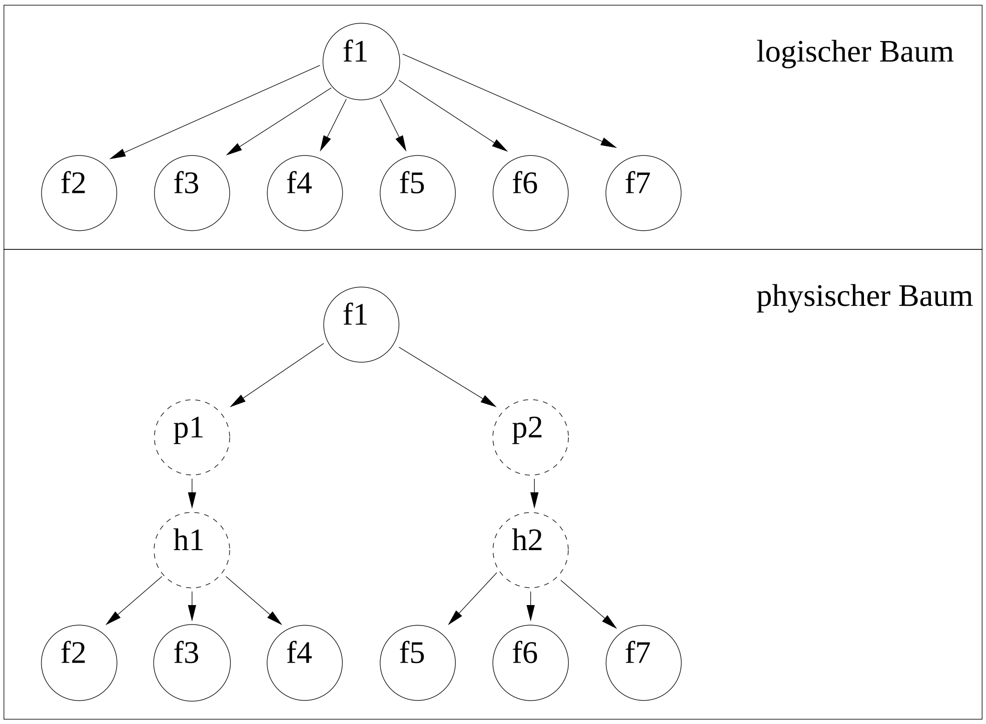

Abbildung 1: Baummodelle

Knoten f1 ist demnach ein alleinstehender Knoten, der Rest eingebette Knoten; oder: f1 bis f1 sind Facette-Knoten, p1, p2, h1 und h2 Gerüst-Knoten; oder: f1 ist ein Aggregations-Knoten, f2 bis f7 sind Literal-Knoten, bei p1 und p2 handelt es sich um Proxy-Knoten und h1 und h2 sind Hilfs-Knoten.

3 Clustering-Problem

Das Clustering-Problem für XML-Dokumente besteht darin, daß die Doklument-Darstellung im logischen Modell in irgendeiner Weise auf Festplattenseiten verteilt werden muß. Dabei kann die Größe des Baums die einer Plattenseite übersteigen.

Ein Lösungsansatz besteht darin, das logischen Modell auf das physische Modell zu erweitern, den Baum an den Proxy-Knoten in Teilbäume zu zerschneiden und schließlich diese Teilbäume zu speichern. Dabei möchte man in jeder Seite genau einen solchen Teilbaum speichern. Man sollte auch vermeiden, daß diese Teilbäume zu klein werden. Solche Seiten enthielten im Extremfall außer einem Facette-Knoten nur Gerüst-Knoten. Als Ergebnis entsteht erneut eine Baumstruktur, deren Knoten jetzt aber die Festplattenseiten sind, welche ihrerseits wiederum die Teilbäume des XML-Dokuments enthalten. Dies verdeutlicht die Abbildung 2 (welche eine Fortführung des letzten Beispiels darstellt)

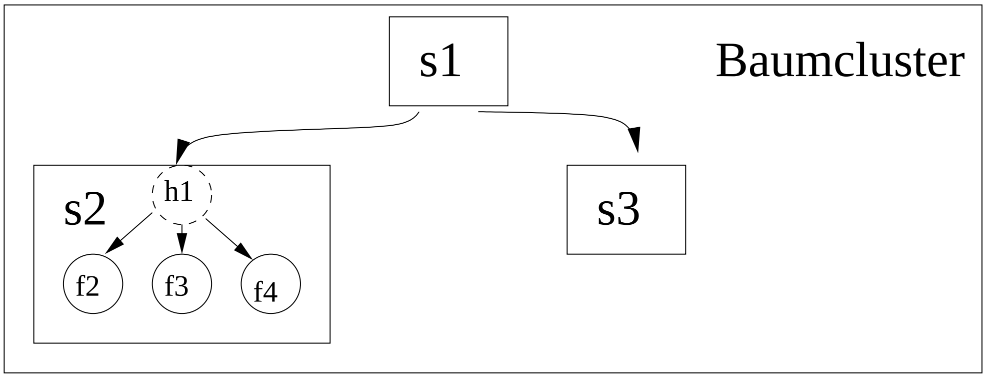

Abbildung 2: Baumcluster

4 Einfügealgorithmus und Splitmatrix

Nachdem sowohl das Problem als auch ein Lösungsansatz bekannt sind, ist es jetzt an der Zeit, diesen Ansatz in einem Algorithmus zusammenzufassen.

4.1 Einfügealgorithmus

Zunächst stelle ich den Einfügealgorithmus vor:

1. Bestimme die Seite s, in die der Knoten d eingefügt werden soll.
2. Falls auf s nicht genug Platz für d ist, splitte s:
   (a) Bestimme rekursiv den Separator in s´s Teilbaum.
   (b) Verteile die sich ergebenden Partitionen auf Seiten.
   (c) Füge den Separator in seine Elternseite ein und führe den Algorithmus aus.
3. Füge d ein.

Anmerkungen zum Algorithmus: (Die Nummerierung entspricht den Algorithmusschritten)

1. Potentiell gibt es so viele Möglichkeiten zum Einfügen eines Knotens wie der bisher gespeicherte Baum Seiten belegt. Dieses Problem kann durch die Splitmatrix (s. u.) behoben werden.

2. Ist die Einfügestelle gefunden, kann es passieren, daß auf der gewählten Seite nicht mehr genügend Platz ist, um einen zusätzlichen Knoten aufzunehmen. Daher muß die Seite gesplittet werden

   (a) Der Separator entsteht durch Abschneiden eines kleinen Teilbaums von der Wurzel des in s gespeicherten Teilbaums. Die Schnittstelle wird durch einen Knoten d' bestimmt. Auf diese Weise zerfällt der Baum in einen linken Teil L (= alle Knoten links von d´), den Separator S (= alle Knoten oberhalb d´) und einen rechten Teil R (= alle Knoten rechts und unterhalb von d´). Die Abbildung zeigt die Seite direkt vor der Durchführung des Splittens. d' ist das Ergebnis eines weiteren Parameters, der das Größenverhältnis zwischen L und R bestimmt. Abbildung 3 zeigt eine Seite, in der bereits diese Aufteilung vorgenommen wurde.

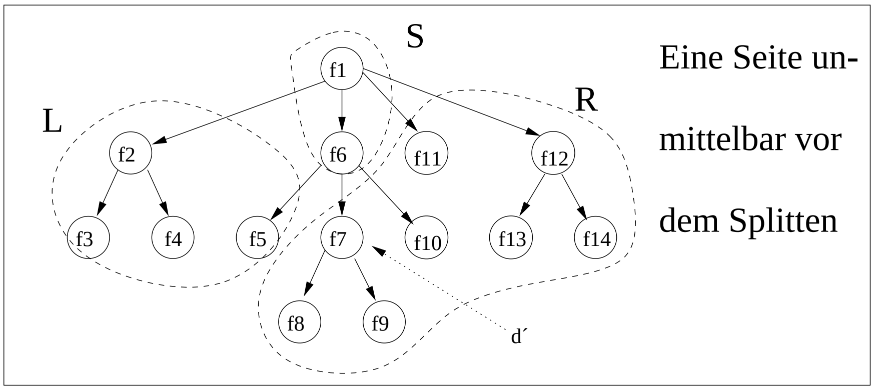

Abbildung 3: eine Seite vor dem Splitten

   (b) Nachdem die Zerlegung des Baums festgelegt ist, muß der verbliebene Wald von Teilbäumen wieder zusammengefaßt und auf Seiten verteilt werden: Wurzelknoten, die ursprünglich Geschwisterknoten waren, werden durch Hilfsknoten zu größeren Teilbäumen verbunden und auf einer eigenen Seite abgelegt. Der Separator wird jetzt noch um entsprechende Proyknoten erweitert.

   (c) Da das Einfügen des Separators in die Elternseite der gesplitteten Seite ein Einfügen ist, wird der gesamte Algorithmus für den Separator ausgeführt. Dies kann sich bis zur Wurzel des Dokumentenbaums fortsetzen. Die Seite aus Schritt 2a ist jetzt über 4 neue Seiten verteilt, wie die folgende Abbildung zeigt.

3. Als abschließender Schritt kann der Knoten jetzt in die Seite eingefügt werden. Alle Berechnungen wurden aber so angestellt, als sei der Knoten bereits im Baum enthalten, um den benötigten Speicherplatz zu schaffen.

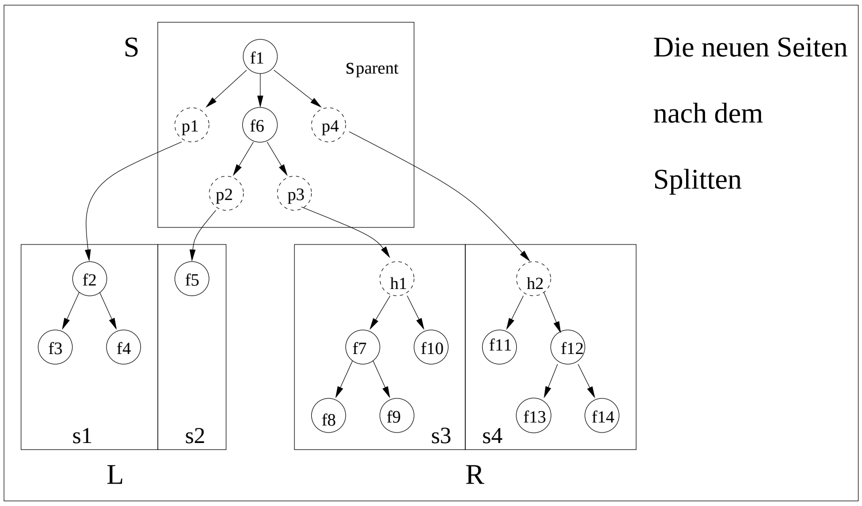

Abbildung 4: Seite nach dem Splitten

4.2 Die Splitmatrix

In der Splitmatrix ist für jedes Knotenpaar des Dokumenten-Baums festgelegt, wie es gespeichert werden soll bzw. an welcher Stelle im Baum es eingefügt werden soll. Dadurch bekommt die Matrix eine Funktionalität ähnlich dem Füllgrad bei Datenbanksystemen. zum anderen kann das Speicherverhalten anderer Systeme (die durchaus auf anderen Ansätzen beruhen als dem Native-Mode) simuliert werden. Die formale Definition der Splitmatrix lautet:

$$s_{i,j} = \begin{cases} 0 & x \text{ wird in eigenem Record gespeichert} \\ \infty & x \text{ und } y \text{ werden so lange wie möglich in einem Record gespeichert} \\ other & \text{der Algorithmus entscheidet} \end{cases}$$

x_j und y_i bilden hierbei ein solches Knotenpaar.

Der Einfügealgorithmus verwendet die Matrix beim Auffinden der Einfügestelle wie folgt: Wenn x_j als Kind von y_i eingefügt werden soll und wenn $s_{ij} = \infty$ ist, dann wird x_j auf der gleichen Seite wie y_i gepseichert. Wenn aber $s_{ij} = \text{other}$ ist, das wird x_j auf der gleichen Seite wie seine potentiellen Geschwister gespeichert.

5 Experimente

5.1 Konfiguration

Datenbasis für die Experimente waren Shakespears Dramen, denen ein Baum-Modell mit ca. 320.000 Knoten entsprach. Die Testreihen wurden eimal mit einer reinen 0-Matrix und einmal mit einer reinen other-Matrix durchgeführt (Die letztere Konfiguration des Testsystems wird auch als Native-Mode bezeichnet). Außerdem wurde die Seitengröße variiert.

5.2 Die Testreihen

Die Testreihen selbst lassen sich wie folgt unterteilen:

- Inserts / Updates
- Baumtraversierung
- Queries
  1. alle Blattknoten eines Teilbaums besuchen (Alle Sprecher im 3.Akt, 2.Szener aller Stücke)
  2. Wiederherstellung eines Dokument-Stücks (alle ersten Sprechpassagen aller Szenen aller Stücke)
  3. Pathquery (die Eröffnungs-Reden aller Stücke)

5.3 Testergebnisse

Inserts und Updates absolvierte der Algorithmus am Schnellsten im Native-Mode mit großen Seiten. Das gleiche gilt auch für die Baumtraversierung.

Bei den Queries zeigt sich eine Schwäche des Algorithmus' : Die Performance ist hier stark abhäng von der Einfügereihenfolge. Im ungünstigsten Fall degeneriert hier der Baum zu einer Liste.

6 Oracle 8i

In diesem Abschnitt sollen einige Utillities und Mechanismen vorgestellt werden, die das objekt-relationale Datenbanksystem Oracle 8i zur Unterstützung von XML-Dokumenten verwendet.

6.1 interMedia

Hinter diesem klangvollen Namen verstecken sich verschiedene Speichermechanismen für XML-Dokumente. Diese sind :

- XML-Dokumente als Dokumente speichern, sprich die Dokumente werden innerhalb des Systems als BLOB (=Binary Large OBject) gespeichert.
- XML-Dokumente als objekt-relationale Datenbank-Tabellen speichern.
- XML-Dokumente und Daten können gemischt und als Views verwaltet werden.

6.2 XML-Parser

XML-Parser sind ein Hilfsmittel, um programmiertechnisch auf XML-Dokumente zugreifen zukönnen. Für jede der von Oracle unterstützen Programmiersprachen (Java, CORBA, PL/SQL, C, C++) existiert ein Parser im System.

6.3 XML-Klassengenerator

Der Klassengenerator erzeugt aus einer DTD Java-Klassen, wobei jedes Objekt der DTD zu einer eigenen Klasse wird. Die generierten Klassen können dann in einem Programm gültige XML-Dokumente zu erzeugen, die auf der Ausgangs-DTD basieren (siehe folgende Abbildung).

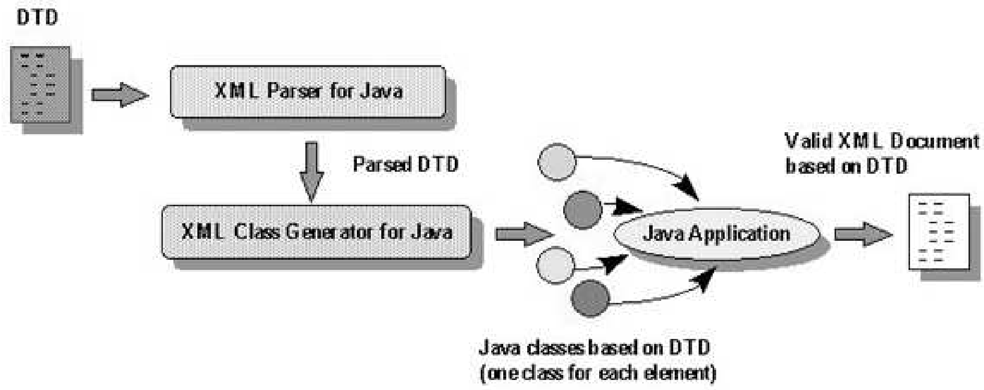

Abbildung 5: Klassengenerator

6.4 XML-SQL

Dieses Dienstprogramm deckt folgende Aspekte ab :

- Das Ergebnis einer SQL-Query als XML-Dokument zurückliefern. Dabei basiert die Struktur des zurückgegebenen XML-Dokuments auf der Struktur der Datenbank, auf der die Query ausgeführt wurde. Skalare Werte werden zu reinen Textelementen, während Objekt-Typen zu Elementen mit Attributen und Sub-Elementen werden.

- Schreiben eines XML-Dokument in eine Datenbank: Dabei bleibt die Dokumentstruktur erhalten. Tags werden zu Spaltennamen, reiner Textinhalt zu Skalarspalten und Elemente, die Subelemente enthalten, werden auf Objekttypen abgebildet.

6.5 XSQL Servlet

Beim XSQL Servlet handelt es sich um einen Mechanismus, um z.B. über das Internet SQL-Queries an eine Datenbank stellen zu können.

Im Einzelnen werden dabei folgende Stationen durchlaufen:

1. Der Benutzer gibt über einen Webbrowser eine URL ein, die dann vom Web-Server intepretiert und an das XSQL-Servlet weitergeleitet wird. Die URL enthält den Namen einer XSQL-Datei (<name>.xsql) und optional Parameter wie z.B. Werte oder den Namen eines XSL-Stylesheets.

2. Das Servlet gibt die XSQL-Datei an einen XML-Parser weiter, welcher ein API für den Zugriff auf den Dokument-Inhalt generiert.

3. Die Seitenprocessor-Komponente des Servlets benutzt dieses API, um XML-Parameter und SQL-Anweisungen, die zwischen <query> und </query> stehen, an das XML-SQL Diensprogramm weiterzugeben. Weiterhin gibt der Seitenprocessor XSL Anweisungen an den XSLT Processor weiter.

4. Das XML-SQL Dienstprogramm schickt die SQL-Anweisungen an das Datenbanksystem, welches die Ergebnisse an das XML SQL Utility zurückgibt

5. Das XML-SQL Dienstprogramm übergibt die XML-formatierten Ergebisse an den XSLT-Processor. Die Ergebnisse befinden sich an der gleichen Stelle im Dokument wie zuvor die <query> - Tags

6. Falls erwünscht, werden die Ergebnisse noch vom XSLT-Processor noch gemäß eines Stylesheets formatiert.

7. In einem letzten Schritt wird das fertige Dokument an den Benutzer zurückgegeben, der es auf seinem Webbrowswe bewundern kann.

Folgende Abbildung verdeutlich den Vorgang als Ganzes noch einmal:

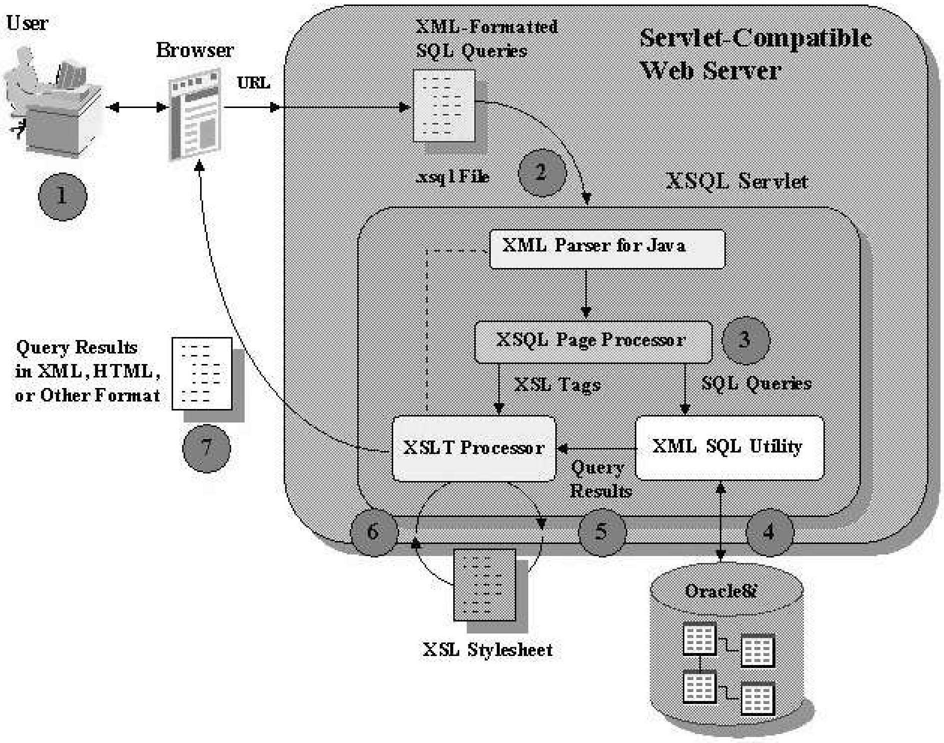

Abbildung 6: XSQL Servlet

Literatur

[1] Carl-Christian Kanne + Guido Moerkotte: "Efficient storage of XML data", Accepted for publication in: International Conference on Data Engineering, 16.Juni 1999 http://www-dbs.cs.uni-sb.de/public_html/lehre/semli99lit.html

[2] B.Wait :"Using XML in Oracle Database Applications: Part 2: About Oracle XML Products." November 1999, http://technet.oracle.com/tech/xml/info/htdocs/otnwp/about_oracle_xml_product.htm, Zugriff : 17.11.1999.