データベースシステム 119−41
（１９９９．７．２２）

XML問い合わせ処理システム(xQues)のデータ格納管理

金政 泰彦 kanemasa@flab.fujitsu.co.jp
久保田 和己 kubotak@flab.fujitsu.co.jp
石川 博 hiro@flab.fujitsu.co.jp

株式会社 富士通研究所
〒211-8588 神奈川県川崎市中原区上小田中 4-1-1

あらまし 近年、ECやEDIなどにおけるデータ交換の標準フォーマットとして、XMLが広く使われるようになってきている。さらに大規模なXMLデータが出現するにつれて、それをデータベースに格納し、問い合わせを行なうことが求められてきている。本稿では、XMLデータをデータベースに格納する際に、特に検討を必要とする問題点について言及し、それに対する一つの解として我々のXML問い合わせ処理システムxQuesの格納構造を紹介する。xQuesのXMLデータ格納構造では特に、汎用にあらゆるXMLデータを格納することと、XMLの木構造を辿る問い合わせを高速実行することに焦点が当てられている。

キーワード XML, 格納構造, インデックス

Storage Management of a XML Query Processing System *xQues*

Yasuhiko KANEMASA kanemasa@flab.fujitsu.co.jp
Kazumi Kubota kubotak@flab.fujitsu.co.jp
Hiroshi Ishikawa hiro@flab.fujitsu.co.jp

FUJITSU LABORATORIES LTD.
1–1, Kamikodanaka 4–Chome, Nakahara–ku, Kawasaki 211–8588, Japan

Abstract Recently, XML has been widely used as a standard format of data interchange in EC or EDI. As large-scale XML data appear, it is necessary to store XML data in databases and retrieve them. In this paper, we mention the problems on storing XML data in databases, and present a storage structure of our XML Query Processing System *xQues* as one answer of the problems. As our XML data storage structure of *xQues*, we store every XML data for general purpose and process queries which traverse the tree structure of XML at high speed.

key words XML, Storage Structure, Index

1 はじめに

近年、新しいWebページ記述言語としてXML (Extensible Markup Language) が注目されている。XMLは、HTMLにSGMLの構造定義機能を加えたようなマークアップ言語で、1996年に発表されて以来、その高い構造記述能力と、構造定義や表示記述の独立化などの設計の良さによって、有望視されている。またXMLは、単にWebページ記述に用いられるだけではなく、テキストベースであることと構造記述能力の高さを利用して、ECやEDIなどにおけるデータ交換の標準フォーマットとしても用いられるようになってきている。

そのようなXMLで記述されたデータが増えるにしたがって、それらのデータに対して検索を行ないたいという要望が出てきた。特に大規模なXMLデータについては、それをデータベースに格納して、それに対して問い合わせを行なうことが求められてきている。

そのような流れを受けて、現在W3Cでは、XMLの為の問い合わせ言語の標準化の作業が進められている。その中で我々も、去年12月に行なわれたワークショップで独自の言語仕様を提案しており [1]、標準化へ向けて活動している。最終的にW3Cで標準化される問い合わせ言語がどのような構文のものになるかは分からないが、機能的には我々の提案しているものと同等の問い合わせ機能を持つものになると考えている。そこで、現在我々はW3Cの標準化に先駆けて、我々の言語仕様を基にして、そのような問い合わせ機能を実行できるデータ格納検索システムについて研究している。以下では、そのシステムの中における、XMLデータをデータベースに格納するための格納構造について説明する。まず最初に節2で、XMLデータをデータベースに格納する際に特に検討を必要とする問題点について言及し、それから節3で、それを解決するための我々のアプローチを紹介する。

2 XMLデータの格納

2.1 XMLデータの格納手法の分類

現在、XMLデータを格納するのに用いられている手法は、大まかに次の3つのタイプに分類することができる (図1)。

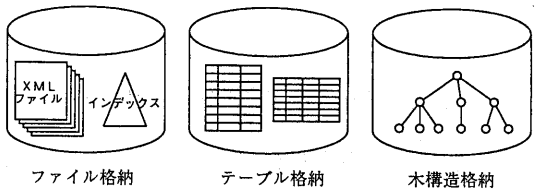

図 1: XMLデータを格納する3つの方法

ファイル格納 XML文書をファイル形式のまま格納する手法 [2][5]。この手法は、オリジナルのXMLファイルの全体あるいは一部をそのまま利用することを目的としていて、その為にXML文書をファイル形式のまま格納する。それだけでは、ファイルの数が増えたときに目的とするファイルを見つけ出すことが困難になるので、目的とするファイルを検索する為のインデックスも用意しておく必要がある。

テーブル格納 XMLをRDBのテーブルにマッピングして格納する手法 [2]。この手法ではXML文書を構造化データと見なし、データベースに格納することによって高速な検索を行なうことを目的としている。その為にこの手法では、各エレメントをRDBのテーブル[1]の各カラムにマッピングして格納する。XMLデータをテーブルにマッピングする為には、XMLの各エレメントをテーブルの各カラムにどのようにマッピングするかというマッピング規則が必要である。このマッピング規則は、現在はユーザに指定させるのが一般的であるが [2]、将来的にはDTDの情報から自動的に生成することも考えられる。

木構造格納 XMLを木構造を保持したままデータベースに格納する手法 [3][4]。この手法は、XMLを木構造の情報を保持したままデータベースに格納することによって、そのXMLデータに対して木構造を辿るような問い合わせを行なえるようにすることが目的である。この手法では、XMLの木構造をノード単位で分割してデータベースに格納する。そしてそれとともに、各ノード間の関係を保持する為のリンク情報も一緒に格納する。この格納法では、XMLの木構造をそのまま格納するので、元のXML文書の構造に関わらずあらゆるXMLデータを格納できるというメリットがある。格納に用いられるデータベースとしては、ODBが使われることが多いが [3]、RDBでも実装可能である。また、データベースを専用設計して高性能化することも行なわれている [4]。

ただし、これら3つの分類は厳密ではない。これら3つの分類に当てはまり難い格納法も考えられるし、3

[1] マッピングされるテーブルは、XMLデータによって1つの場合もあるし複数の場合もある

つの境界線上にある格納法も考えられる。特にファイル格納に関しては、XMLデータをファイル形式で格納するものを一括りにしてこう分類しているが、検索の為のインデックスの形式は色々な方法が考えられ、その中には木構造の情報を持っているものも存在する[5]。また、ファイル格納のインデックスとしてRDBのテーブルを利用することも考えられる。

2.2 XMLデータの格納の問題点

XMLデータを格納する際に一番問題となるのは、そのデータ構造が一意に定まっていないという事である。DTDのないXMLデータでは、どこにどのようなタグが出現するか分からず、データ構造は全く分からない。DTDのあるXMLデータでさえも、DTDの中でタグの繰り返しやタグの選択、タグの再帰的な宣言が許されているので、データ構造が一意に定まらない(このようなデータを半構造データと呼ぶ[6])。このようなデータ構造の定まっていないXMLデータを格納しようとすると、格納スキーマの設計が問題となる。

例えば、図2に示されるサンプルXMLデータをテーブル格納で格納した場合を考える(図3)。このサンプルXMLデータは、2冊の本の情報を含む書籍目録のデータである。図3のテーブルでは、1タプルが本1冊分の情報に相当していて、列にはXMLデータ中で出現する可能性のある全てのタグがとられている。これを見ると、一見サンプルデータが問題なく格納されているかのように見える。しかし、サンプルデータのDTDに書かれた定義には著者数の制限が無いのに、図3のテーブルでは著者を格納するスペースは最大2人分しか用意されていない。もしXMLデータの中に著者がそれ以上存在したら、そのデータは格納できないか、格納しても情報が一部欠損することになる。

このように、テーブル格納では、XMLのDTDで記述される繰り返しタグを格納することができない。これは、テーブル格納ではあらかじめ格納する要素を列として指定しておく必要があるので、最大数が未定の繰り返し要素を表現できないからである[2]。また、同じ理由で再帰的に定義されているタグも格納できない。さらに、そもそもXMLデータにDTDが存在しなくて、どのようなタグが出現するか分かっていないときには、テーブルの構造を決められず、全く対応できない。

一方、ファイル格納は、XMLデータをファイル形式のまま格納するので、DTDの無いXMLデータであろうと半構造のXMLデータであろうと、格納できないXMLデータは存在しない。しかし、それだけでは大量に格納されたデータの中から自分の求める情報だけを検索することができないので、検索用のインデックスが必要となる。インデックスの構成は目的に応じて色々と考えられ、簡単なものではタグ名と文字列の組をキーにして、そのタグに囲まれてその文字列が出現しているようなXML文書を検索してくるというものがある。しかし、そのような簡単なインデックスでは、タグの階層構造を考慮した検索は行なえない。タグの階層構造の情報を持つようにインデックスを工夫することも考えられるが、それでもなお次のことが問題として残る。

- インデックスがXMLの木構造の全ての情報を持っていないので、XMLデータの全情報を使った検

[2] 厳密に言えば格納できないわけではない。繰り返し部分を別テーブルに独立させれば格納できる。しかし、この場合はテーブルが分割されることによる性能面の問題や、分割されたテーブルへの問い合わせの複雑さの問題が生じる

[DTD]

```
<!ELEMENT book (author+, title, publisher)>
<!ATTLIST book year CDATA>
<!ELEMENT article (author+, title, year?, (shortversion | longversion))>
<!ATTLIST article type CDATA>
<!ELEMENT publisher (name, address?)>
<!ELEMENT author (firstname?, lastname)>
```

[XMLデータ]

```
<bib>
    <book year="1995">
        <title> An Introductory to Database System </title>
        <author> <lastname> Date </lastname> </author>
        <publisher> <name> Addison-Wesley </name> </publisher>
    </book>

    <book year="1998">
        <title> Foundation for Object/Relational Database </title>
        <author> <lastname> Date </lastname> </author>
        <author> <lastname> Darwen </lastname> </author>
        <publisher> <name> Addison-Wesley </name> </publisher>
    </book>
</bib>
```

図 2: XMLデータの例

bookのテーブル

ID	title	author1 firstname	author1 lastname
1	An Introductory to Database System		Date
2	Foundation for Object/Relational Database		Date
⋮	⋮	⋮	⋮

author2 firstname	author2 lastname	publisher name	publisher address	year
		Addison-Wesley		1995
	Darwen	Addison-Wesley		1998
⋮	⋮	⋮	⋮	⋮

図 3: テーブル格納による半構造データの格納

索ができない。

- インデックスが木構造を辿ることに最適化されていないので、そのような検索を行なった場合は検索速度が遅い。

2.3 我々のアプローチ

前節で述べたように、XMLデータの格納には、

- いかにして、DTD無しのXMLデータや半構造のXMLデータを格納するか
- 格納されたXMLデータに対して、いかにして木構造を辿るような複雑な問い合わせを高速に実行できるようにするか

という問題がある。これらの問題を解決するために、我々のXML問い合わせ処理システムxQuesでは、格納法として木構造格納を選択した。木構造格納は、XMLの木構造をそのまま格納するので、DTD無しのXMLデータや半構造のXMLデータも格納できる。また、木構造格納では、XMLの木構造を全てデータベース上に格納しているので、木構造の全ての情報を検索に利用することができる。

我々の木構造格納は、富士通版XQLで問い合わせを行なことを目的としているので、木構造を辿る検索を高速に実行できる必要がある。木構造格納はRDB上にもODB上にも構築することができるのだが、我々は速度性能の観点からRDB上に構築することを選択した。また、木構造を辿る検索を高速に実行するためには、木のリンク(枝)を辿る際にインデックスを効果的に使用できる必要がある。我々の格納構造は、この点を特に考慮して設計してある。

3 xQuesのXMLデータ格納構造

3.1 木構造格納によるXMLデータの格納

前節で述べたように、xQuesではXMLデータをRDB上に木構造格納で格納している。木構造格納はXMLデータの木構造をノード単位に分割して格納するのであるが、我々はその分割されたノードを格納するのにRDBのテーブルを使用している。

XMLデータを木構造で表現する方法はいくつかあるが、我々は図4のような表現方法を用いた。図4は、図2のXMLデータを木構造で表現したものである。この木構造表現では、丸い中間ノードはエレメントを表していて、ノードの親子関係がエレメントの包含関係を表している。ノードの丸の中の数字はノードIDを表している。ノードとノードを結ぶリンク(枝)はタグを表していて、リンクの横に書かれている文字列はタグ名を表している。三角の葉ノードはエレメントの値を表し、四角い葉ノードはタグに付けられた属性(Attribute)を表している。値を持つのはこの2つの葉ノードだけである。

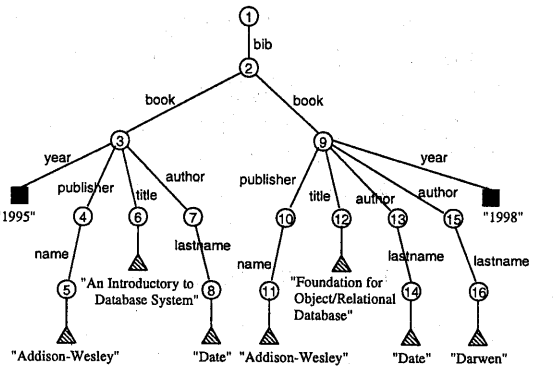

図4: XMLデータの木構造表現

ノードを分割してデータベースに格納するときに、ノードの情報だけをデータベースのテーブルに格納したのでは、木構造のノード間の繋がり、つまりリンクの情報が欠落してしまう。そこで、リンクの情報はリンクの情報としてそれを格納する専用のテーブルを用意する。またノードも、中間ノードとエレメント値の葉ノード、属性の葉ノードは格納構造が異なるので別々のテーブルに格納する必要がある。

xQuesの木構造格納で使用するテーブルは、全部で次の6つである。

中間ノードテーブル これは中間ノードの情報を格納するテーブルである。ノードID (id)の他に、そのノードが含まれている文書の文書ID (docid)、そのノードまでのルートからのフルパスのID (pathid)をカラムとして持っている。

リンクテーブル これはノード間のリンクを格納するテーブルである。ノードID (id)、リンクのラベル(タグ名)のID (labelid)、子ノードのノードID (child)、その子ノードの全兄弟ノード中での出現順序(tord: total order)、その子ノードの同ラベルを持つ兄弟ノード中での出現順序(pord: partial order)をカラムとして持っている。

葉ノードテーブル これはエレメント値の葉ノードを格納するテーブルである。そのエレメントにあたる中間ノードのノードID (id)の他に、エレメン

トの値 (value) と、そのエレメント中でその値が出現した順序 (order) をカラムとして持っている。

属性ノードテーブル これはタグにつけられた属性を格納するテーブルである。そのタグが含まれるエレメントにあたる中間ノードのノード ID (id) の他に、属性名の ID (labelid)、属性値 (attvalue) をカラムとして持つ。

我々の木構造表現では XML のタグが木構造のリンクに相当するので、XML のタグに付けられる属性は、本来ならばリンクに付くべきである。しかし我々の方法では、属性はリンクに対してではなく、その下のノードに付いている。これは、検索時のテーブル参照の回数を少なくするためである。

パス ID テーブル これはパス ID とパスの文字列の対応表である。パスの文字列を中間ノードテーブルの中に直接書き込まないでこのように別に持っているのは、スペースの節約の為もあるが、パス名の文字列マッチングを含む検索が行なわれたときに、検索対象が少なくてすみ、検索が高速化できるからである。

ラベル ID テーブル これはラベル ID とラベルの文字列の対応表である。

xQues の木構造格納の格納状態の例として、図 2 のサンプル XML データを上記のテーブル群で格納した様子を図 5 に示す。

3.2 インデックスの構成

我々の木構造格納では、本来連結されていたはずの木構造のノードが、1 つ 1 つに分割されて RDB のテーブルに格納されている。これによって、木構造を辿る問い合わせが行なわれた場合、問い合わせで辿る部分のリンクを連結し直すためにジョイン操作が行なわれる。このジョイン操作の速度は全体の検索速度に大きく影響するので、ジョイン操作を高速に行なえるようにインデックスを効果的に張っておく必要がある。

また、問い合わせが行なわれる場合、検索条件として指定されるのは、エレメントの値、属性、パス、出現順序などである。それらの検索も高速に行なう必要があるので、そこにもインデックスを用意しておく必要がある。

我々が図 5 のテーブルに張ったインデックスの一覧を表 1 に示す。このインデックスは B-tree で張ってあり、キーが複数の属性の組からなるインデックスは、その組の先頭からの部分的な属性の組で検索に用いることもできる。

中間ノードテーブル

id	docid	pathid
5	1	1
4	1	2
6	1	3
8	1	4
7	1	5
3	1	6
11	1	1
10	1	2
12	1	3
14	1	4
13	1	5
16	1	4
15	1	5
9	1	6
2	1	7
1	1	8

リンクテーブル

id	labelid	tord	pord	child
4	5	0	0	5
7	8	0	0	8
3	4	0	0	4
3	6	1	0	6
3	7	2	0	7
10	5	0	0	11
13	8	0	0	14
15	8	0	0	16
9	4	0	0	10
9	6	1	0	12
9	7	2	0	13
9	7	3	1	15
2	2	0	0	3
2	2	1	1	9
1	1	0	0	2

葉ノードテーブル

id	order	value
5	0	Addison-Wesley
6	0	An Introductory to Database System
8	0	Date
11	0	Addison-Wesley
12	0	Foundation for Object/Relational Database
14	0	Date
16	0	Darwen

属性ノードテーブル

id	labelid	attvalue
3	3	1995
9	3	1998

パスIDテーブル

pathid	path
1	bib.book.publisher.name
2	bib.book.publisher
3	bib.book.title
4	bib.book.author.lastname
5	bib.book.author
6	bib.book
7	bib
8	/

ラベルIDテーブル

labelid	label
1	bib
2	book
3	year
4	publisher
5	name
6	title
7	author
8	lastname

図 5: テーブル構成

表 1: インデックス一覧

テーブル名	キー
中間ノード	id
中間ノード	(docid, id)
中間ノード	(pathid, id)
リンク	(id, labelid, child)
リンク	(child, labelid, id)
葉ノード	id
葉ノード	value
属性ノード	id
属性ノード	attvalue
パス ID	path
ラベル ID	label

なお中間ノードテーブルに張ってあるインデックスでキーが (pathid, id) のものは、あるパスに該当する全てのノードを検索してくるときに使用するものである。このインデックスのキーは、一見 pathid 単独で構わないように思われるかもしれない。しかしキーを pathid だけにすると、同じキー値を持つエントリが多量に発生して、B-tree インデックスが機能しなくなる。

3.3 問い合わせの実行手順

最後に、xQues で木構造格納された XML データに対する問い合わせ処理が、どのように行なわれるかを示す。例として、図 2 のサンプル XML データを xQues の木構造格納で格納した図 5 の状態に、「著者が Darwen である本のタイトルを求めよ」という問い合わせを行なった場合を考える。この場合に生じるテーブル検索は、次のようになる。

1. 葉ノードテーブルを検索して、値が "Darwen" であるノードのノード ID を得る
2. パス ID テーブルを検索して、パス "bib.book.author.lastname" のパス ID を得る
3. 中間ノードテーブルを 1. で得られたノード ID で検索して、得られたパス ID が 2. で得られたパス ID と一致することを確認する
4. ラベル ID テーブルを検索して、ラベル "lastname" のラベル ID を得る
5. リンクテーブルを検索して、3. のノード ID と 4. で得られたラベル ID から、親ノードのノード ID を得る
6. ラベル ID テーブルを検索して、ラベル "author" のラベル ID を得る
7. リンクテーブルを検索して、5. で得られたノード ID と 6. で得られたラベル ID から、親ノードのノード ID を得る
8. ラベル ID テーブルを検索して、ラベル "title" のラベル ID を得る
9. リンクテーブルを検索して、7. で得られたノード ID と 8. で得られたラベル ID から、子ノードのノード ID を得る
10. 葉ノードテーブルを検索して、9. で得られたノード ID から、そのノードの値を得る

4 おわりに

本論文では、XML データの格納に関して、特に検討を必要とする問題点について言及し、それに対する一つの解として、我々の XML 問い合わせ処理システム xQues の格納構造を紹介した。我々は、この格納構造を現在 Oracle8 上に実装しており、性能評価を行なっているところである。

今後の課題としては、XML データの更新に関する検討が挙げられる。XML データの更新については、どの単位 (ファイル単位 or エレメント単位) で一貫性を保証するかとか、どのようなロック戦略を採れば性能低下を防げるかなど、まだまだ検討すべき課題が残っている。今後は、それらの検討を進めていく予定である。

参考文献

[1] H. Ishikawa, K. Kubota, Y. Kanemasa, "XQL: A Query Language for XML Data", In Position papers for W3C Query Language Workshop, Dec. 1998, http://www.w3.org/TandS/QL/QL98/pp/flab.txt.

[2] Oracle Corporation, "XML Support in Oracle8*i* and Beyond", An Oracle Technical Whitepaper, http://www.oracle.com/xml/documents/xml_twp/, (1998).

[3] Object Design Inc., "eXcelon", http://www.odi.com/excelon/main.htm.

[4] Software AG Corporation, "Tamino? White Paper", http://www.softwareag.com/tamino/Download/tamino.pdf, (1999).

[5] 志村 壮是, 吉川 正俊, "オブジェクト関係データベースを用いた XML 文書の格納と検索", DEWS'99 電子情報通信学会 (1999).

[6] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, J. Widom, "Lore: A Database Management System for Semistructured Data", SIGMOD Record, Vol.26, No.3, September 1997.