XMLデータベース技術概説

天笠　俊之，吉川　正俊

大量のXMLデータを格納し，高速な検索，更新機能を提供するXMLデータベースを実現する手法は，世界中の大学やベンダで非常に活発な研究が行われている分野であり，数多くの提案がなされている．XMLデータベースシステムを実現する手法は，XML専用に設計したネイティブXMLデータベースを構築する方法と関係データベースにXMLデータを分解して格納する方法に大別することができる．XMLデータの論理構造は木であるため，木のノードのラベリング手法は，XMLデータベースのための重要な基盤技術である．

キーワード：ネイティブXMLデータベース，関係XMLデータベース，ノードラベリング

1. XML入門

1.1 XMLの構文とデータモデル

図1にXMLデータの例を示す．XMLデータは，階層構造を持った要素（element）によって構成される．ある要素は開始タグ<要素名>とそれに対応する終了タグ</要素名>によって規定され，開始タグの中には属性名＝“値”のようにいくつかの属性を持つことができる．要素は，その内容として文字列や複数の要素を持つことができ，このような要素の再帰的な構造でデータを記述する．このため，開始タグと終了タグは完全な入れ子になっている必要がある．すべてのタグが正しく入れ子になっているXMLデータを整形式（wellformed）という．

```
<?xml version="1.0"  encoding="utf-8"?>
<!DOCTYPE bibliography SYSTEM "bib.dtd">
<bibliography>
  <book year="2003">
    <title>XQuery from the Experts</title>
    <authors>
      <author>Chamberlin</author>
      <author>Draper</author>
    </authors>
    <publisher>Addison Wesley</publisher>
    <price>30</price>
    <comment>An introduction to XQuery.</comment>
  </book>
</bibliography>
```

図1　XML文書の例

あまがさ　としゆき
筑波大学 計算科学研究センター
〒305-8573 つくば市天王台1-1-1
よしかわ　まさとし
名古屋大学 情報連携基盤センター
〒464-8601 名古屋市千種区不老町

要素が階層構造を成していることから，XMLデータは本質的に木構造を表現していると考えることができる．XMLデータモデルは，要素，属性，文字列等をノードとして木表現したものである．例えば，図1のXMLデータを構文解析することによって，図2に示す木構造を得ることができる．XMLデータモデルにおいて，根ノードは実体を持たない仮想的なノードであり，その子ノードにトップレベルの要素が配置される．ある要素ノードはその子ノードとして，要素，属性，テキストノードなどを持つことができる．

1.2 DTD

XMLは，要素名や属性名などのマークアップ語彙を自由に与え，用途に応じた言語を定義することのできるメタ言語である．マークアップ語彙はスキーマ言語によって与える．文書型定義（DTD；Document Type Definition）は，XMLの前身であるSGML（Standard Generalized Markup Language）に由来するスキーマ言語である．DTDの例を図3に示す．DTDでは基本的に，要素型宣言によってタグ名とそ

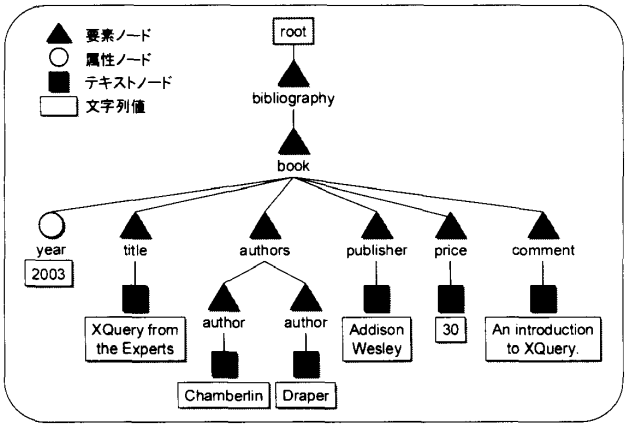

図2　XMLデータの木構造

```
<!ELEMENT bibliography (book*)>
<!ELEMENT book (title, authors, publisher?, price?, comment?)>
<!ATTLIST book year CDATA #IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT authors (author+)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT author (#PCDATA)>
```

図3　DTDの例

の内容モデルを与える．例えば，

〈!ELEMENT bibliography(book＊)〉

では，bibliography要素は，子要素として0個以上のbook要素を持つことをbook＊で表している．同様に，1回以上の出現は＋，0回または1回の出現は？で表す．ある要素に付随する属性は，属性リスト宣言によって記述する．

〈!ATTLIST book year CDATA #IMPLIED〉

では，book要素がyearを属性として持ち，その値は文字列であって，その属性自体は必須でない（属性を記述しなくても良い）ということを表している．これらを組み合わせて，マークアップしたい言語の形式を与えることができる．あるXMLデータがスキーマ定義に従うとき，そのデータを妥当（valid）であるという．すべての妥当なXMLデータは整形式のXMLデータである．

DTDは古い仕様であるため，1)記法が難解である，2)要素の内容モデルを詳細に記述することができない，3)整数や浮動小数点などのデータ型を持たない，4)名前空間[1][13]に対応していないなどの問題が指摘されている．このため，XML SchemaやRELAX NGなどの新たなスキーマ言語が提案されている．

1.3　XPath

すでに述べたように，XMLデータは要素が階層を成した木構造を有する．このため，XMLデータを利用する際には，特定の要素や属性などを指定する方法が必要である．XPath（XML Path Language）は，XMLデータ内の特定の部分を指定するための汎用言語である．XPathはXSLT（XSL Transformations）やその他の多くの規格でも使われている．より正確な文法や意味論は，田島による解説［16］を参照されたい．XPathでは，XMLのデータモデルに基づき，経路式で要素や属性を指定する．例えば，XPath式/bibliography/bookは，bibliography要素の子要素であるbook要素を示している．属性ノードへは@isbnのように，属性名に@をつけて表現する．また，XPathに特徴的な表現として，1)//によって子孫ノードを記述することができる（/bibliography//authorはbibliography要素から辿れる任意の深さにあるauthor要素を示す），2)[]によって述語を記述することができる（//book[year＝"2003"]は，子要素yearの値が"2003"であるようなbook要素を示す）等が挙げられる．

```
for $b in doc("bib.xml")//book
let $a := $b//author
where count($a) >= 2
return
    <result>{ $b/title }</result>
```

図4　XQueryの例

1.4　XQuery

XQueryはW3Cで標準化が進んでいる，XMLのための問合せ言語である．XQueryは一言で言うと，リレーショナルデータベースにおけるSQLのように，XMLデータベースに対して柔軟で強力な問合せを可能にする言語である．

XQueryにおいて，最も強力で一般的な表現はFLWOR式である．これはfor, let, where, order by, returnの頭字語であり，SQLにおけるSELECT-FROM-WHEREと似ている．基本的にfor句およびlet句で変数に値を束縛し，タプル列を生成する．where句では，条件を与えタプルをフィルタリングする．order by句でフィルタされたタプルの整列を行い，return句で出力の整形を行う．XQuery問合せの例を図4に示す．

[1] 名前空間とは，いくつかのマークアップ語彙を組み合わせて新たなスキーマを構築することを可能にする機構である．あるURI（Uniform Resource Indicator）によって識別されるスキーマ部分の集合を名前空間と呼ぶ．これによって，要素名や属性名の衝突を避けることができる．

2. XMLデータベースとは？

XMLは，1998年にW3Cの勧告となって以来急速に普及が進み，データ記述，交換のための標準フォーマットとして広く認知されるようになった．その結果，我々の周辺には膨大なXMLデータが情報資源として存在している．従来のWWWデータはもちろん，ワープロや表計算などのオフィスデータ，果てはプログラムが生成する設定ファイルの類まで，そのような例は枚挙に暇がない．

このような状況において，大量のXMLデータを格納し，高速かつ柔軟な検索手段を提供することのできるXMLデータベースが重要である．XMLデータベースにおいて，格納の対象となるのはXMLデータそのものである．格納されたXMLデータに対して，XPathやXQueryなどの問合せ言語で質問を発すると，該当するXML（部分）データを返す．

本解説は，このようなXMLデータベースの仕組みを分かりやすく説明することを目的にしている．XMLデータベースと一口に言っても，その実現方法はいろいろである．次では，まず，XMLデータ専用のシステムに基づくネイティブXMLデータベースを説明する．次に，汎用のデータストレージである関係データベースを使い，XMLデータを格納，検索することのできる関係XMLデータベースを紹介する．続いて，これらの手法において，高速な検索を実現するための技法として，ノードラベリング法と構造結合演算を紹介する．

3. ネイティブXMLデータベース

ネイティブXMLデータベースとは，XMLデータに特化した専用のデータベースシステムのことを言う．一般に，データベースに格納されるデータはメインメモリよりも大きいため，普段はディスク上に格納され，必要に応じてメモリ上に呼び出される．しかしながら，ディスクはメモリに比べて極めて遅いため，ディスク上のデータ配置，ディスクキャッシュ，索引などを工夫することによって，高速なアクセスを実現している．

関係データベースやオブジェクトデータベースなどの既存のシステムは基本的に構造化されたデータを念頭に設計されている．これに対してXMLデータは半構造性を持っており，ゆるやかな構造は持つものの，それは可変的である．このため，既存の技術をそのまま応用するだけでは十分とは言えない．XMLデータの性質はアプリケーションによって決まるので，アプリケーションの要求に応じて，適切な処理手法を選択することが重要である（両極端な例としては，定型データをXMLデータとして記述することもできるし，逆にスキーマを持たないXMLデータも許されている）．

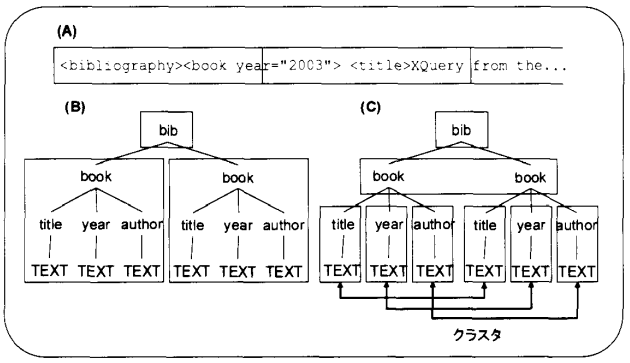

図5　データ配置

次では，ネイティブXMLデータベースに関連する技術のうち，ディスク上のデータ配置法と索引について説明する．

3.1 XMLデータの格納法

XMLデータにおいてスキーマは必須ではなく，構造も緩やかな制約しか持たないので，データの配置方法も自由度が高く，さまざまな方法が提案されている．これらは大きく次の三通りに分類できる(図5)：

1. XMLデータを先頭からディスクページのサイズに分割し，順に格納する方法（図中(A)）．最も単純だが，XMLデータの構造を利用した問合せはデータベースだけでは処理できない点が問題となる．

2. XMLデータを構文解析して得られた木構造において，ディスクページのサイズに格納可能な部分木ごとにデータを分割し格納する方法（図中(B)）．クラスタの単位より細かい要素を取得したい場合には，追加の処理が必要になる．

3. 要素の名前やセマンティクスから，関連した要素をまとめて近接したディスクページに格納する方法(図中(C))．2.とは逆に，ある程度の大きさの要素を抽出したいときに，該当する部分木の再構築が必要になる．

例えば，Natixでは基本的に2.の方式を取っている[5]．ただし，利用者の指示によっていくつかの格納方法から自由に選ぶことができるシステムや，データの性質に応じて自動的に格納法式を選ぶシステムもある．

3.2 XMLデータの格納法

上でも見たように，一つの物理格納法でもってシステムへのあらゆる処理要求に対応するのは困難である．そこで，検索を高速に行うために索引を構築する．既存のデータベースでは，一次元データのためのB+-treeや多次元データのためのR-treeなどが知られているが，XMLデータを対象とした索引には次のようなものがある．

全文索引 XMLデータの内容を指定して，それを含む要素（属性）を検索する．例）“XML”を含むすべてのTITLE要素を検索せよ．

構造索引 XMLデータの構造を指定し，該当する要素を検索する．例）すべてのTITLE要素を検索せよ．

構造索引は，後程説明するノードナンバリングとも密接に関連するので，ここでは全文索引について説明する．情報検索の分野では，単語を検索キーとしてそれを含む文書を高速に検索するためのデータ構造として，転置ファイル（inverted file）がよく使われる．転置ファイルは，検索キーとして全文書から抽出した自立語を持ち，その値として文書そのものへのポインタを持つ．このとき検索キーはあらかじめアルファベット順にソートしておく．こうすることによって，与えられた検索語を含む文書を高速に検索することができる．XMLデータを対象に転置ファイルを応用すると，例えば，文書へのポインタの代わりに，各出現位置へのポインタを持たせることができれば，単語そのものの検索は処理することができるように思われる．しかしながら，この場合，与えられた単語そのものの出現位置は得ることができるが，上の例のように，ある単語を含む要素名といった，XMLに特有の問合せには対応することができない．

そこで，XMLに対応した全文索引技術が研究されている．主な手法の一つは，単語と要素の出現位置に着目し，両者を整数値の包含関係で表現しようとするものである[15]．具体的には，各単語の出現位置をXMLデータの先頭から順に整数値で表現し，要素の出現位置はその要素が含むすべての単語の出現位置を含むような区間として表現する．この手法は，後で説明する範囲ノードラベリングと基本的に同じ考え方である．その利点として，位置情報の包含関係を調べることによって，単語を直接含む要素だけではなく，単語と祖先要素，もしくは要素どうしの包含関係をも表現することができる点がある．

4. 関係XMLデータベース

4.1 あらまし

XMLの特性に特化した方式でXMLデータを効率よく扱おうとするのがネイティブXMLデータベースであるのに対して，汎用のデータストレージを利用してXMLデータを管理しようとするのが関係XMLデータベースのアプローチである．XMLデータの格納に関係データベースを用いる主な利点は次の通りである：

1) 関係データベースは世界中に普及し企業向けのエンタープライズ製品からオープンソースの手軽なものまで数多くの実装が世に出回っている．

2) 稼働中のシステムに膨大な情報資源が格納されており，これらとの連携を図るのが容易である．

3) 問合せ最適化やトランザクション処理など過去30年にわたる技術の蓄積がある．

以上の理由から，関係XMLデータベースはこの数年来活発な研究，開発が行われている．

XMLデータの関係データベースへの格納は，双方のデータモデルに本質的な差異があるため単純には行えない．これはXMLデータが木構造であるのに対し，(正規化された) 関係データベースにおける関係表は平坦でなければならないことによる．したがってXMLデータを関係表に格納するためには，両者の間で適切なマッピングを行わなければならない．実際，このようなマッピングは無数に存在するため，関係XMLデータベースの実現方法も無数に存在すると言える．それぞれの手法には一長一短があるため，格納するデータやアプリケーションの特徴を考慮して，適切な方法を選択することが重要となる．

4.2 XMLデータから関係表へのマッピング

XMLデータを関係表に格納する最も単純な方法は，関係データベースが提供する不定長文字列，BLOB (Binary Large Object)，CLOB (Character Large Object) などの機能を用いて，XMLデータを長大な文字列として単一のカラムに格納する方法である．この場合，データの出し入れの単位はXMLデータそのものとなってしまう．したがって，XPathやXQueryを用いて，文書全体よりも粒度の細かい要素や属性の抽出を行う検索を処理するために，関係データベース自身の機能を用いることができず，専用の索引機構が必要になってしまう．

XMLデータをそのまま文字列として扱うのではな

く，関係表とのマッピングを考える手法は，大きく構造写像アプローチとモデル写像アプローチの二つに分類することができる．構造写像アプローチは，個別のXMLデータの構造（スキーマ）に基づいて関係スキーマを設計する方法である．これにはいくつかの手法が提案されているが，Shanmugasundaram等は，DTDを解析し，対応する関係スキーマを自動設計する手法を提案している[11]．提案手法では，まず元となるDTDから要素の出現順序の情報を取り除くことによって簡単化を施し，DTDグラフを抽出する．例えば，図3に対応するDTDグラフを図6に与える．この結果，DTDグラフには，ある要素の出現回数が1回のみか複数回か，必須の要素かどうかといった情報だけを保持することになる．次に，要素の出現回数や参照関係などを考慮して，DTDグラフから関係スキーマへの変換を行う．基本的に，すべての要素に対して対応する関係表を作成する手法をBasic Inlining，複数の要素から共通に参照される要素を独立した関係表にし，単一の親しか持たない要素は親要素に対応する関係表の属性に変換する手法をShared Inlining，両者を組み合わせた手法をHybrid Inliningという．例えば，図3に示したDTDからShared Inliningによって次のような関係スキーマが得られる．

```
bibliography (bibliographyID: integer,
              bibliography.bookid: string)
book (bookID: integer, book.year: string,
      book.title: string, book.authorsid: string,
      book.publisher: string, book.price: string,
      book.comment: string)
authors (authorsID: integer, authors.authorid: string)
author (authorID: integer, author: string)
```

問合せについては，与えられたXPath式を変換後の関係スキーマ上のSQL質問に変換する手法を提案している．同様の研究では，文書スキーマやXMLデータの統計情報から問合せ処理時のコストを予測し，

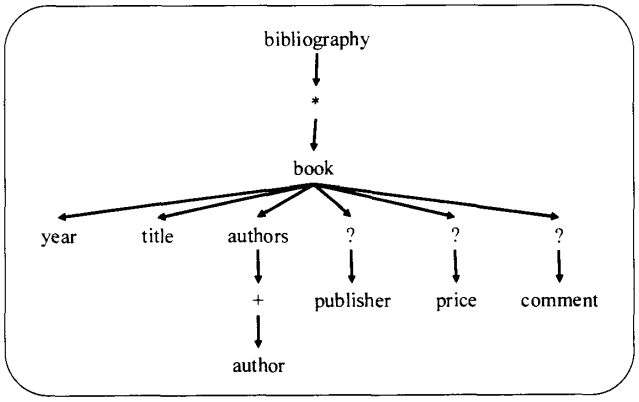

図6　DTDグラフ

最適な関係スキーマを導出する研究[2]などがある．

構造写像アプローチは，単一もしくは少数の固定されたスキーマに従うXMLデータを大量に取り扱う場合に効率が良いのが特徴である．しかしながら，文書スキーマを持たない整形式のXMLデータは格納することができない．また，文書スキーマの種類が多い場合，関係スキーマの設計が煩雑である．さらに，文書スキーマが頻繁に更新される場合，それに合わせて関係スキーマも変更しなければならないなどの欠点がある．また，XPathに特徴的な任意の子孫の探索//elemが標準のSQLの機能では素直に表現することができない点も指摘しておく．

モデル写像アプローチは，XMLデータモデルに基づいて関係スキーマを設計する手法である．この手法は，個別の文書スキーマに依存しないため，固定した関係スキーマを用いて，任意の整形式XMLデータを格納することができる．このため，多様な文書スキーマに基づくXMLデータを単一のデータベースで扱いたい場合に有効である．ここでは，モデル写像アプローチの一例として経路アプローチ[14]を紹介する．

経路アプローチでは，XMLデータを固定された四つの関係表（Element, Text, Attribute, Path）で表現する．基本的に，XMLデータモデル中の各ノード（要素ノード，テキストノード，属性ノード）はそれぞれ対応する関係表（Element, Text, Attribute）中のタプルに変換される．このとき各ノードはルートからの経路式と，データ中での出現順を保持するためのノード番号とともに格納される．ノード番号にはこれまでに多くの手法が提案されており，その特徴は様々である．経路アプローチではルートからの先祖子孫関係を判定できることが求められる．ここではそのようなノード番号の例として範囲ラベリングを前提に説明を進める．範囲ラベリングでは二つの整数の組（*start, end*）を用いる．ここで，1)兄弟ノードでは範囲が重複せず，兄の*end*より弟の*start*が大きくなる，2)任意の親子ノードにおいて，親ノードの範囲が必ず子ノードの範囲を含むよう，再帰的に番号付けを行う．このような番号付けの分かりやすい例には，ノードの前置順と後置順，各ノードのXMLファイル中の開始バイト位置と終了バイト位置などがある．これによって，ノードのXMLデータ中での出現順序を表現するだけでなく，番号の包含関係を調べることによって，任意の異なる2ノード間に先祖子孫関係があるかどうかを判定することができる．後述するが，この性質は

Path

pathID	pathexp
0	#/bibliography
1	#/bibliography#/book
2	#/bibliography#/book#/@year
3	#/bibliography#/book#/title
4	#/bibliography#/book#/authors
5	#/bibliography#/book#/authors#/author
6	#/bibliography#/book#/publisher
7	#/bibliography#/book#/price
8	#/bibliography#/book#/comment

Element

docID	pathID	st	ed	idx	reidx
0	0	82	392	1	1
0	1	99	376	1	1
0	3	122	159	1	1
0	4	165	252	1	1
0	6	258	294	1	1
0	7	300	316	1	1
0	8	322	366	1	1
0	5	181	207	1	2
0	5	215	237	2	1

Text

docID	pathID	st	ed	value
0	3	129	151	XQuery from the Experts
0	5	189	198	Chamberlin
0	5	223	228	Draper
0	6	269	282	Addison Wesley
0	7	307	308	30
0	8	331	356	An introduction to XQuery.

Attribute

docID	pathID	st	ed	value
0	2	100	100	2003

図7　経路アプローチによる格納例

問合せ処理を行う上で極めて重要である．その結果，各ノードを経路式とノード番号によって唯一に識別することが可能となる．例えば，図1のXMLデータは図7のように変換される．

問合せ処理は，Path式をSQLに変換することによって行う．例えば，//authors/author[2]は，次のSQLに変換される．ここで，任意の深さにあるauthors要素（//authors）を探索するために，SQLのあいまい検索機能（%）を用いている点に注目されたい．

```
SELECT   e1.docID, e1.st, e1.ed
FROM     Path p1, Element e1
WHERE    p1.pathexp LIKE '#%/authors#/author'
AND      e1.pathID = p1.pathID
AND      e1.idx = 2
ORDER BY e1.docID, e1.st
```

経路アプローチの特徴は，文書スキーマに依存しない固定された四つの関係表を用いて，任意のXMLデータを格納できる点と，関係データベースの機能を一切拡張することなしにXMLデータの格納と検索が可能になることである．最近では，Microsoft SQL Server 2005のXMLデータ型が，経路アプローチを発展させた方式で実現されている[10]．

5. ノードラベリング

ここまで見てきたように，XMLデータをデータベースに格納する際には，木構造から平坦な関係表への構造変換を行う必要がある．しかしながら，XPathやXQueryなどの問合せは元の木構造に対して行われるので，ノードの接続関係等，構造に関する情報を関係表の中でも表現することが重要である．このためノードラベリング手法は関係XMLデータベースにおいて重要な基盤技術の一つになっている．また，ネイティブXMLデータベースにおいても，索引を構築する際のノード位置情報を表現する手段として用いることができる．これらの理由から，ノードラベリングはこの数年来活発に研究されている．

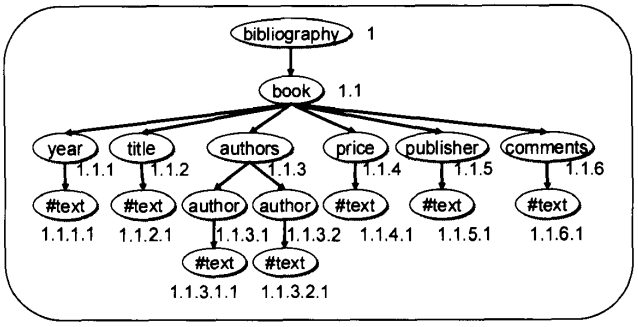

図8　Deweyオーダ

ノードラベリング手法には，2ノード間の関連の判定能力が高いことと，元データへの変更（要素や属性の追加，更新，削除）に対して頑健であることが求められる．前者は特に問合せ処理の効率化に大きな影響がある．ノードラベルからノード間の関連を知ることができれば，データそのものを参照するI/Oコストが削減でき，また不要な探索空間を減らすことができる．結果として検索性能の向上を図ることができる．これに対して，後者はデータベースの更新性能に関係する．XMLデータが更新されるとデータの構造が変化するため，それに合わせてノードラベルも更新する必要がある．ノードラベルによっては，最悪の場合でほぼすべてのノードラベルを振り直す必要があるため，データベースの更新性能に重大な影響を及ぼす．このため，更新が発生しても，その影響範囲を局所的に抑えられることが望ましい．

ノードラベリング手法の最も基本的な手法の一つは，節4.2で述べた範囲ラベリングであるが，この他にもいくつかの興味深い方式が提案されている．Deweyオーダ[12]は，図書館学の分類法であるデューイ十進分類法（DDC: Dewey Decimal Classification）を応用したノードラベリング手法である（図8）．ノードラベルは，次のように再帰的に与えられる：

1) ルート要素のノードラベルは1，

2) ある，兄弟ノードにおいてi番目のノードのノードラベルは，その親ノードのノードラベルをpと

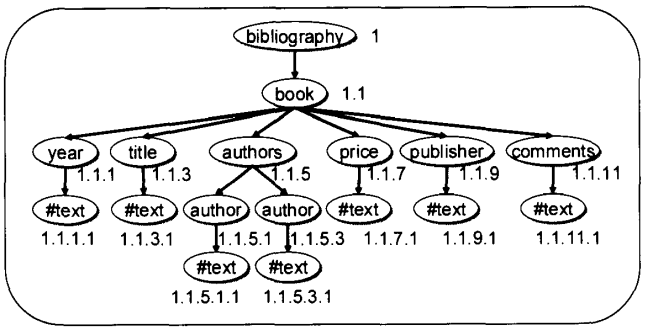

図9 ORDPATH

するとき“$p.i$”.

Deweyオーダは，その定義から明らかなように，先祖子孫関係だけではなく親子関係の判定も可能であり，また兄弟ノードにおける順序も特定することができる．更新に対しては，挿入（削除）個所の全ての右兄弟ノード，およびそれらの子孫ノードが影響を受ける．したがって，新規のノードが末子として追加される場合は効率よく処理ができるが，新規ノードが途中に挿入されたり，削除されるような場合には大きな手間がかかることがある．近年では，Deweyオーダのバリエーションとして，DTDなどのスキーマ情報を取り入れたSPIDERS-Dewey[6]などが提案されている．

Microsoft SQL Server 2005では，ORDPATH[9]と呼ばれるラベリング手法が採用されている．ORDPATHは，ノードの挿入や削除などに対して頑健となるようにDeweyオーダを改良したものとみなすことができる．例えば，図9は，ORDPATHでラベリングを行ったXML木の例である．この図から分かるように，ORDPATHでは，まず奇数のみを用いてDeweyオーダによるラベリングを行う．偶数はノード挿入時に用いる．例えば，図9の二つのauthorノードの間に新しいauthorノードを挿入する場合，そのノードのORDPATHは，1.1.5.2.1となる．木におけるノードのレベルは，ノードラベルの奇数の数によって分かる．

5.1 構造結合

XMLと関係データベースとの間のモデルの不整合は，問合せ処理でも問題になる．XPathやXQueryが木構造を前提とした処理を要求するので，同等の処理を関係表の上で効率よく実現する必要がある．特に，XPathに特徴的な子孫の探索（//）は，ナイーブな実装では（部分）木の全数探索が必要になるので大きな問題である．

実際，関係XMLデータベースにおいてXPath式E1//E2を評価する処理は次のように類型化できる：

- 要素E1のノード集合Aを獲得する
- 要素E2のノード集合Dを獲得する
- AとDの間に先祖子孫関係が成り立つペアのみを抽出する

問合せがE1/E2の場合は，先祖子孫関係の代わりに親子関係を調べることになる．これに対応する処理は，ノードを範囲ラベリング法でラベル付けした場合には，SQLで次のように記述することができる：

```
SELECT   D.docid, D.nodeid
FROM     Element A, Element D
WHERE    -- "E1" と "E2"をそれぞれ A と D に束縛
AND      A.docid = D.docid
AND      A.start < D.start
AND      A.end < D.end
ORDER BY D.docid, D.start
```

関係XMLデータベースにおけるXPath式の評価は，基本的にこの処理の組み合わせに帰着される．このように，二つのノード集合に対して，XMLデータ上で関連のあるすべてのペアを列挙する処理を構造結合[1]と呼ぶ．構造結合は関係XMLデータベースの問合せ性能に直接関連する技術であり，この数年来極めて活発に研究が行われている．

Al-Khalifa等は，構造結合を効率よく処理するためのアルゴリズムとして，Tree-Merge JoinとStack-Tree Joinを提案している[1]．Tree-Merge Joinは関係データベースにおける併合結合演算の自然な拡張であるのに対して，Stack-Tree Joinはスタックを用いてXMLデータ上の深さ優先探索を効率よく実行するアルゴリズムになっている．実験の結果，両者ともI/OとCPUコストに関して最適であるものの，最悪のケースではStack-Tree JoinがTree-Merge Joinよりも高性能であることが示されている．

索引構造を用いて構造結合を高速化しようとする試みも数多く行われている[4, 7]．これらの基本的な考え方は，各ノードの位置をあらかじめB+-treeやR-treeなどの索引に格納しておき，構造結合の際にはこれらを手がかりにノードの読み飛ばしを行うというものである．その結果，ノード同士の比較回数を減すことができ，性能向上を図ることができる．

上では，述語を含まない単純な親子（先祖子孫）関係だけを含むようなXPath式を対象にしているが，近年では述語を含むより複雑な問合せ（例えば，

book[title="XML" AND @year="2003"]) を対象としたアルゴリズムもよく研究されている．これらは，XPath式を構文解析した結果が複数の枝を含むためholistic twig queryと呼ばれる[3, 8]．

6. おわりに

XMLデータベースが，1990年頃活発に研究開発が行われたオブジェクト指向データベースと異なる点として，標準化が進んでいること，関係データベースシステムベンダ自身が開発を進めていることを挙げることができる．今後，XMLデータベースと関係データベースの共存の姿が進化を伴いながら形作られていくことが予想される．

参考文献

[1] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava. Structural joins: A primitive for efficient XML query pattern matching. In *Proc. ICDE 2002*, pp. 141-152, 2002.

[2] P. Bohannon, J. Freire, P. Roy, and J. Siméon. From XML schema to relations: A cost-based approach to XML storage. In *Proc. ICDE 2002*, pp. 64-75, 2002.

[3] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal XML pattern matching. In *Proc. SIGMOD 2002*, pp. 310-321, 2002.

[4] S.-Y. Chien, Z. Vagena, D. Zhang, V. J. Tsotras, and C. Zaniolo. Efficient structural joins on indexed XML documents. In *Proc. VLDB 2002*, pp. 263-274, 2002.

[5] T. Fiebig, S. Helmer, C.-C. Kanne, G. Moerkotte, J. Neumann, and R. Schiele. Anatomy of a native XML base management system. *The VLDB Journal*, 11(4): 292-314, 2002.

[6] K. Fujimoto, T. Shimizu, D. D. Kha, M. Yoshikawa, and T. Amagasa. A mapping scheme of XML documents into relational databases using schema-based path identifiers. In *Proc. Int'l Workshop on Challenges in Web Information Retrieval and Integration (in conjunction with ICDE 2005)*, 2005.

[7] H. Jiang, H. Lu, W. Wang, and B. C. Ooi. XR-tree: Indexing XML data for efficient structural joins. In *Proc. ICDE 2003*, pp. 253-263, 2003.

[8] H. Jiang, W. Wang, H. Lu, and J. X. Yu. Holistic twig joins on indexed XML documents. In *Proc. VLDB 2003*, pp. 273-284, 2003.

[9] P. E. O' Neil, E. J. O' Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. Ordpaths: Insert-friendly xml node labels. In *Proc. SIGMOD Conf. 2004*, pp. 903-908, 2004.

[10] S. Pal, I. Cseri, G. Schaller, O. Seeliger, L. Giakoumakis,and V. V. Zolotov. Indexing XML data stored in a relational database. In *Proc. VLDB 2004*, pp. 1134-1145, 2004.

[11] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *Proc. VLDB 1999*, pp. 302-314, 1999.

[12] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. In *Proc. SIGMOD 2002*, pp. 204-215, 2002.

[13] World Wide Web Consortium. Namespaces in XML. http://www.w3.org/TR/REC-xml-names/. W3C Recommendation 14 January 1999.

[14] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura.XRel: A path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology (TOIT)*, 1(1): 110-141, June 2001.

[15] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman. On supporting containment queries in relational database management systems. In *Proc. SIGMOD Conf. 2001*, pp. 425-436, 2001.

[16] 田島敬史：XML用木パターン言語XPath解説，オペレーションズ・リサーチ，第50巻，第6号，pp. 373-378, 2005.