XMLDB におけるデータ更新を考慮したモデル写像アプローチ

A Model-Mapping Approach Considering Data Update on XMLDB

佐村 敏治† 佐藤 隆士‡

Toshiharu SAMURA Takashi SATO

XML はデータ記述およびデータ交換の新たな標準として利用されている． XML をデータベースに格納するのに多くの研究者は関係データベースを用いることを提案している．関係データベースへの写像アプローチが様々ある中で，モデル写像アプローチ は，XML スキーマの必要がなく，データベーススキーマはどんな XML でも同じなので，頻繁に変更の生じる XML には扱いが容易である．しかしこのアプローチは XML ラベリング手法を除き更新操作に対してあまり議論されていない．本研究では更新を考慮したモデル写像アプローチを提案する．XML データの構造の 変化に対して更新を最小にするデータベーススキーマを定義し，他のモデル写像アプローチと同レベルの検索ができるように設計した．最後に実験結果についても説明する．

XML is emerging as a new major standard for description and exchange of data. More researches have proposed using relational database storing XML. In the various mapping approaches for storing XML, the model-mapping approaches do not require XML schema so that fixed database schema can be used. However, these approaches are not so discussed for update operation excluding XML labeling methods. In this paper, we present a model-mapping approach considering update operation. This method is to define the database schema to reduce the change to the update operations as much as possible, and is designed to retrieve the same level as other model-mapping approaches. The experimental results are also presented.

1. はじめに

XML(Extensible Markup Language)は，データ記述およびデータ交換の基盤技術として登場した．XMLを利用した応用技術の増加に伴い，XMLデータも増加するため，それらを効率的に格納・検索するシステムとしてデータベースが必要となる．XMLを格納したり運用したりするデータベースとして多くの研究者は関係データベースを提案している．

XMLを関係データベースに格納するモデルとして，構造写像モデルとモデル写像モデルに大きく分類される[1]．構造写像アプローチはXMLスキーマに基づき，データベーススキーマを設計するために構造情報を利用する[2]．ただしこれらのモデルは，XMLの構造が変化するとデータベーススキーマも変化しなければならないので，頻繁に構造が変わるXMLデータには適さない．一方，モデル写像アプローチはXMLの構造が変化してもデータベーススキーマは変化しないように設計されている．従ってXMLの変更を柔軟に扱うことができ，関係データベースの管理においても優位である[1]，[3]，[4]，[5]．しかし，どのモデルにしても更新の操作についてはあまり議論されていない．

本研究ではXMLを関係データベースで管理する場合のデータ更新を考慮したモデル写像アプローチを提案する．XMLの構造の変化に対してデータベーススキーマの更新を最小になるようにモデルを構築する．特徴として，更新に強いデータベーススキーマを実現するために双方向リストの概念を用いる．また従来のモデルにおいて更新時に大きなコストとなっていた絶対ロケーションパスを関係表として持たずに，マテリアライズドビューを生成することで，データ更新における複雑な変更を避ける．

本論文の構成は次のとおりである．まず2章では，本研究の関連研究としてXParent[4]を中心に述べる．3章において，更新を議論する場合の問題点と，我々の提案するモデルについて説明する． 4章で更新と検索についての評価実験の結果を示し，5章でまとめと今後の課題について述べる．

2. 関連研究

モデル写像アプローチでデータベーススキーマの顕著なモデルとして， XRel[1]，Edge[3]，XParent[4]，INode[5]等がある．

本論文では XParent[4]を例に説明する．

XParent は４つのテーブルから構成される．

Path(PathID, Len, PathExp)
DataPath(Pid, Cid)
Element(PathID, Ordinal, Did)
Data(PathID, Did, Ordinal,Value)

Path テーブルは，*PathExp* 属性で絶対ロケーションパスを示し，対応する *PathID* 属性で他のテーブルと結合する．Path テーブルは XPath 等を用いて高速検索に用いられる．*Len* 属性は階層の深さを表す．Element テーブルは，要素が出現した時，*Did* 属性の番号で割り当てられる．また同じ絶対ロケーションパス(*PathID* 属性)を持つ兄弟ノードの順序を *Ordinal* 属性で表す．DataPath テーブルは，Element テーブルの *Did* 属性の親子関係を表す．*Pid* 属性が親の *Did* 属性を表し，*Cid* 属性がその子要素の *Did* 属性を表す．このテーブルを用いて先祖子孫関係の問い合わせを行う．検索を高速に行うため，Ancestor テーブルと呼ばれる全ての先祖子孫関係を格納したテーブルを使うことも提案している．Data テーブルは，テキストノード及び属性値を *Value* 属性に格納する．

3. 更新を考慮したモデル写像アプローチ

3.1 更新操作における考察

写像アプローチを XML データベースとして構築するために更新操作は不可欠である．更新にはノードの挿入，削除，移動の 3 種類の操作が挙げられる．挿入に対して前章で述べた XParent[4]を例にとると次の変更が必要になる．

(1) 兄弟ノードの順序を示す *Ordinal* 属性の再構成
(2) 親子関係を示す *DataPath* 属性の再構成
(3) 絶対ロケーションパスに関する Path テーブルの変更

例えば図 1 のような挿入では，work 要素の子孫に対しても変更が必要となり，XParent では非常に複雑な操作となる．

† 正会員 明石工業高等専門学校電気情報工学科
samura@akashi.ac.jp
‡ 正会員 大阪教育大学情報処理センター
sato@cc.osaka-kyoiku.ac.jp

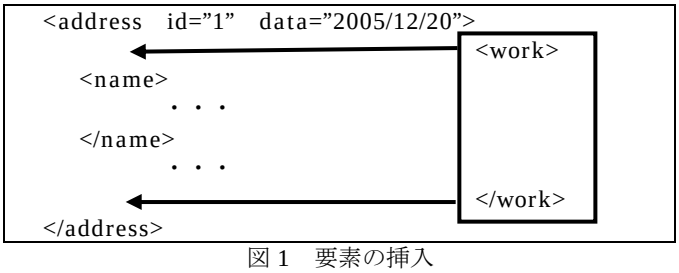

図 1 要素の挿入

Fig. 1 Insertion of Element

また削除は，ノードを検索しノードを削除する操作を合わせたものとなる．多くのコストがノード検索に占められることから，削除は検索の性能に依存すると考えられる[6]．移動は挿入と削除の組み合わせになる．

3.2 データベーススキーマとその特徴

本研究の目的は，更新に対して変更を最小にし，他のモデル写像アプローチと同等な検索ができるようなデータベーススキーマを設計することである．

我々は，3 つのテーブルと 1 つの関数，そして 1 つのマテリアライズドビュー(materialized view)で構成する新しいモデル写像アプローチを提案する（図 2）．またデータベーススキーマ属性の説明を表 1 に示す．

Table
Element(seq, name, sID, snID, eID, enID, aID, tID, pID)
Text(tID, Value)
Attribute(name, aID, Value)
Function
absPath (sid: integer}): text;
Materialized View
PathMV(sid, path)

図 2 本モデルで用いるテーブルスキーマ，関数，マテリアライズドビュー

Fig. 2 Table Schema, Function and Materialized View in Our Model

表 1 データベーススキーマ属性の説明

Table 1 Explanations of Database Schema Attributes

テーブル名	属性	説明
Element テーブル	*seq*(sequence)	要素の出現順序
	name	要素名
	sID(start element ID)	開始要素 ID
	snID	開始要素の次 ID
	eID(end ID)	終了要素 ID
	enID	終了要素の次 ID
	aID(attribute ID)	属性 ID
	tID(text ID)	テキストノード ID
	pID(parent ID)	親要素の sID
Text テーブル	*tID*(text ID)	テキストノード ID
	Value	テキストノード値
Attribute テーブル	*name*	属性名
	aID(attribute ID)	属性 ID
	Value	属性値

図 3 の XML 文書[7]を例に考える．図 4 は図 3 の XML 文書を木構造で表した図である．表 2 に Element テーブルを，表 3(左表)に Text テーブル，表 3(右表)に Attribute テーブルを格納した結果を示す．

```
<addresses>
  <address id="1" date="2005/12/20">
    <name>
      <first>John</first>
      <last>Smith</last>
    </name>
    <city>Houston</city>
    <state>Texas</state>
    <country>United States</country>
    <phone type="home">33-300-0300</phone>
    <phone type="work">333-500-9080</phone>
  </address>
</addresses>
```

図 3 XML 文書例

Fig. 3 Example of XML Document

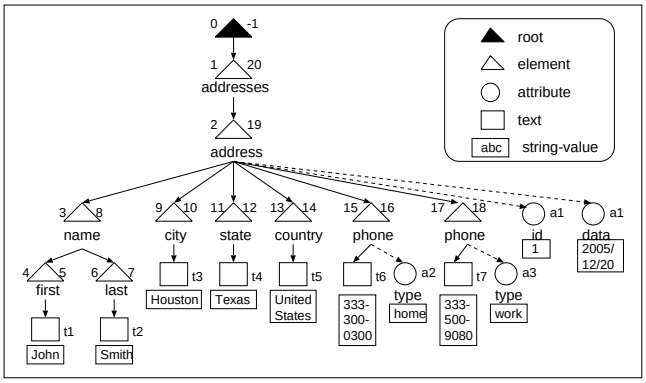

図 4 XML 文書(図 3)の木構造図

Fig. 4 Tree Structure Chart of XML Document (Fig. 3)

本モデルの特徴を述べる．

(1) 更新に強いデータベーススキーマを実現するため，双方向連結リストの概念を用いる

Element(要素)テーブルの属性として親要素 ID へのポインタ(*pID* 属性)と次要素へのポインタ(*xnID* 属性: x={s | e})を導入する．これにより要素の追加に対しては，前後要素の親要素および次要素 ID にのみ注目すればよい．親要素 ID へのポインタは親子関係の検索のために使用する．次要素 ID へのポインタは，高速に XML を構成するためと，更新を行うときに自身の ID を変更しなくてよいために使用する．格納時は，ID を出現順序通りに割り振っているが，次節で述べるように要素の挿入を行うと ID の順序は前後する．

表 2 Element テーブル

Table 2 Element Table

seq	*name*	*sID*	*snID*	*eID*	*enID*	*aID*	*tID*	*pID*
10	addresses	1	2	20	-1	0	0	0
20	address	2	3	19	20	1	0	1
30	name	3	4	8	9	0	0	2
40	first	4	5	5	6	0	1	3
60	last	6	7	7	8	0	2	3
90	city	9	10	10	11	0	3	2
110	state	11	12	12	13	0	4	2
130	country	13	14	14	15	0	5	2
150	phone	15	16	16	17	2	6	2
170	phone	17	18	18	19	3	7	2

表 3 Text テーブル(左表)と Attribute テーブル（右表）
Table 3 Text Table(left) and Attribute Table(right)

tID	*Value*
1	John
2	Smith
3	Houston
4	Texas
5	United States
6	333-300-0300
7	333-500-9080

name	*aID*	*Value*
id	1	1
date	1	2005/12/20
type	2	home
type	3	work

(2) 開始要素 ID と終了要素 ID を導入する

様々な要素の挿入に対応するため，また混合内容に対応するため，開始要素 ID(*sID* 属性)と終了要素 ID(*eID* 属性)を導入する．

(3) ノードの種類ごとにテーブルを用意する

ノードの種類ごとにテーブルを作成することで，XML の木構造を関係データモデル上に実現する．

(4) Element テーブルに *seq* 属性を導入する

3.1 節で述べたように *Ordinal* 属性は，更新により再番号づけを行うと大きなコストを生む．そこで本モデルでは，*Ordinal* 属性を使わずに，*seq* 属性を導入して順番をソートすることでノードの順序を表す．

(5) 絶対ロケーションパスを関係表として持たずに，マテリアライズドビューとして生成する

本モデルの大きな特徴は Path テーブルを持たないことである．3.1 節で説明したように Path テーブルは更新操作で大きなコストを生む．しかし XPath による検索では絶対ロケーションパスの利用は非常に便利である．

そこで我々は Element テーブルから絶対ロケーションパスを生成する関数を定義し，マテリアライズドビューにより動的に絶対ロケーションパスを生成する方法を導入する．Element テーブルから絶対ロケーションパスを生成するアルゴリズムについては論文[8]で述べている．表 4 に，表 2 から生成される PathMV のマテリアライズドビューを示す．

表 4 PathMV マテリアライズドビュー
Table 4 PathMV Materialized View

sID	*path*
1	/addresses
2	/addresses/address
3	/addresses/address/name
4	/addresses/address/name/first
6	/addresses/address/name/last
9	/addresses/address/city
11	/addresses/address/state
13	/addresses/address/country
15	/addresses/address/phone
17	/addresses/address/phone

PathMV は更新が生じたときに作りなおすようにする．この操作は XParent の Path テーブルと比較してあまり違いがないように思われる．しかし XParent による Path テーブルの更新作業は非常に面倒で，アルゴリズムも複雑である．本方法では，更新に対して PathMV を作り直すので，特に更新用のアルゴリズムを必要としない．

3.3 更新過程

本節では，更新過程の一例を示す．詳細な更新過程のアルゴリズムについては論文[8]を参照して欲しい．

更新過程：address 要素の id 属性が "1" のとき address 要素の子要素として work 要素(仕事用)を挿入する（図１）．

表 5 更新後の Element テーブル
Table 5 Element Table after Update Process

seq	*name*	*sID*	*snID*	*eID*	*enID*	*aID*	*tID*	*pID*
10	addresses	1	2	20	-1	0	0	0
20	address	2	21	19	20	1	0	1
21	work	21	3	22	19	0	0	2
30	name	3	4	8	9	0	0	21
40	first	4	5	5	6	0	1	3
60	last	6	7	7	8	0	2	3
90	city	9	10	10	11	0	3	21
110	state	11	12	12	13	0	4	21
130	country	13	14	14	15	0	5	21
150	phone	15	16	16	17	2	6	21
170	phone	17	18	18	22	3	7	21

更新後の Element テーブルを表 5 に示す．*seq*=21 の行が新しく挿入された行である．要素の挿入により，挿入の前の要素と親要素しか変更する必要がないことが分かる（変更箇所は表の網掛け部分）．

XParent 等の他のモデル写像アプローチで以上のような更新過程を扱える汎用的なアルゴリズムを考えることは非常に困難である．

4. 実験による性能評価

本モデルの有効性を調べるため性能実験を行う．実験環境は，Pentium4 1.7GHz の CPU，1024MB の RAM，70GB のハードディスクを用いる．OS は Vine Linux 3.2，使用した DBMS は PostgreSQL 7.4.8 である．時間計測には PostgreSQL の対話型インタフェース psql の”¥timing”を用いた．

4.1 更新による性能実験

更新についての頑健性を調べるために，Michigan Benchmark[9]を用いて性能評価を行った．ds0.1x(ファイルサイズ 45MB，要素ノード数 67696)のデータを用いる．

ベンチマークによる問い合わせとして，選択(QS)，結合(QJ)，集約(QA)，更新(QU)が用意されていて，本論文では更新問い合わせ(QU)について評価を行った．各問い合わせを SQL に変換して更新を行う．ただし，ベンチマーク問い合わせの QU5 と QU6 については，本モデルでは更新操作でないので省略する．

表 6 Michigan Benchmark を用いた更新時間
Table 6 Time of Update Process using Michigan Benchmark

Query	Query Description	Time(sec)
QU1	Point Insert	0.63
QU2	Point Delete	0.44
QU3	Bulk Insert	10.68
QU4	Bulk Delete	2.03
QU7	Restructuring	2.01

実験計測結果を表 6 に示す．時間的コストが発生するのは検索およびその結果を一時的な表やビューに格納する操作である．XParent において本ベンチマークで SQL 文を作成することは，テーブルに依存した形で生成することは可能であっても汎用的に生成することは困難である．

表 6 の結果には PathMV のマテリアライズドビューの作りなおしの時間は考慮されていない．もし最初から作りなおしを行うと，平均 70.56 秒程度の時間がかかり，更新時間のほとんどを占めてしまうことが分かる．対策としてトリガ時に更新要素とその子孫ノードのみを作りなおすようにすれば短時間による更新も可能だと考えられる．

4.2 検索による性能実験

検索についての性能実験として Bosak Shakespeare コレクションを用いる[10](全ファイルサイズ：7.5MB). 検索項目として論文[1]による XPath による問い合わせを用い，SQL に変換した結果で時間を計測した(図 5).

本モデルと XParent とを比較した実験結果を図 6 に示す. XParent には Ancestor テーブルを用いていない．単純な検索については大きな差はないことが分かる．しかし順序を決定する検索(Q5，Q6)については，XParent が *Ordinal* 属性を持ち，その値を直接参照しているため，短時間で検索できている.

```
Q1  /PLAY/ACT
Q2  /PLAY/ACT/SCENE/LINE/STAGEDIR
Q3  //SCENE/TITLE
Q4  //ACT//TITLE
Q5  /PLAY/ACT[2]
Q6  (/PLAY/ACT)[2]/TITLE
Q7  /PLAY/ACT/SCENE/SPEECH[SPEAKER='CURIO']
Q8  /PLAY/ACT/SCENE[//SPEAKER='Steward']/TITLE
```

図 5　Bosak Shakespeare コレクションを用いた検索

Fig.5　Query using Bosak Shakespeare

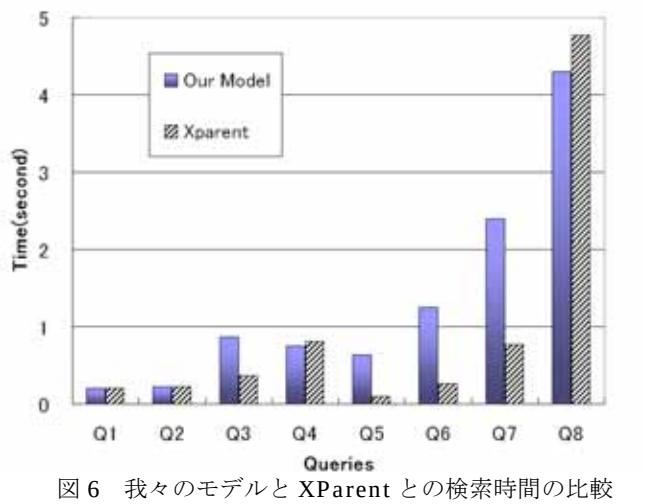

図 6　我々のモデルと XParent との検索時間の比較

Fig.6　Comparisons of Query time of Our Model and XParent

5. まとめ

本研究では，関係データベースを用いた XML データベースとして，新しいモデル写像アプローチを提案した．特にデータ構造が頻繁に変更される XML に対してデータベーススキーマの変更を少なくするように設計を行った．また検索についても性能劣化が生じないように，データベーススキーマを定義した．最後に実験により本モデルの有効性を示した.

ただし更新におけるマテリアライズドビュー作りなおしの議論や，いくつかの検索では他モデルより性能が悪い結果も得られ，今後の課題として残った.

今後，更新においては XUpdate[11]などを例にとり実装を行っていく．また，*seq* 属性については現在議論されているラベリング手法[12]などを使って *seq* 属性だけで先祖子孫関係を検索できるモデルを検討する.

[謝辞]

本研究に関して,第 17 回データ工学ワークショップ(DEWS2006)で多くの方から貴重な意見やコメントをいただいた．この場を借りて深く感謝する.

[文献]

[1] Yoshikawa, M., Amagasa, T., Shimura, T. and Uemura, S..: “XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases”, ACM Transactions on Internet Technology, Vol. 1, No. 1, pp. 110-141 (2001).

[2] Dhanmugasundaram, Tufte, J. K., Zhang, C., Gang, H., DeWitt, D. J. and Naughton, J. F.: “Relational databases for querying xml documents: Limitations and opportunities”, In Proceedings of the 25th International Conference on Very Large Data Bases, (1999).

[3] Florescu, D. and Kossmann, D.: “A performance evaluation of alternative mapping schemes for storing XML data in a relational database”, INRIA, p.31 (1999).

[4] Jiang, H., Lu, H., Wang, W. and Yu, J. X.: “Path Materialization revisited: an efficient storage model for XML data”, In Proceedings of the 13th Australasian Conference on Database Technologies, Australian Computer Society, Inc., pp.85-94 (2002).

[5] Lau, H. K. and Ng, V.: “INODE An Enumeration Scheme for Efficient Storage of XML Data”, Cooperative Internet Computing, Kluwer Academic Publisher, pp.165-184 (2003).

[6] Kit, L. H. and Ng, V.: “Enumerating XML Data for Dynamic Updating”, ADC 2005, pp. 75-84 (2005).

[7] Staken, K.: “XML:DB XUpdate Use Cases” (2006). http://www.xmldatabases.org/projects/XUpdate-UseCases

[8]佐村敏治, 佐藤隆士: “XMLDB におけるデータ更新を考慮したモデル写像アプローチ”, 第 17 回電子情報通信学会データ工学ワークショップ(DEWS2006) 論文集, 1C-oi1, (2006).

[9] The Michigan Benchmark, (2006). http://www.eecs.umich.edu/db/mbench/

[10] The Bosak Shakespeare collection, (2006). http://metalab.unc.edu/bosak/xml/eg/shaks200.zip

[11] XML : DB Initiative. XUpdate -XML Update Language, (2006). http://xmldb-org.sourceforge.net/xupdate/

[12] O'Neil, P. E., O'Neil, E. J., Pal, S., Cseri, I., Schaller, G. and Westbury, N.: “ORDPATHs: Insert- Friendly XML Node Labels” , SIGMOD Conference 2004, pp.903-908 (2004).

佐村 敏治　Toshiharu SAMURA

明石工業高等専門学校電気情報工学科助教授．1994 年神戸大学自然科学研究科博士課程修了，博士（理学）．1996 年福井工業大学講師．XML データベースの研究・開発に従事.

佐藤 隆士　Takashi SATO

大阪教育大学情報処理センター教授．1978 年岡山大学工学研究科修士課程修了．同年詫間電波工業高等専門学校助手．1985 年工学博士．全文検索，XML 検索などの研究に従事.