文書スキーマに適合するXML文書の関係データベースへの格納手法

寺田　憲正†　　吉川　正俊††

† 名古屋大学工学部 電気電子・情報工学科 情報工学コース
†† 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
E-mail: †terada@dl.itc.nagoya-u.ac.jp, ††yosikawa@itc.nagoya-u.ac.jp

あらまし　現在 XML は利用が広まりつつあるが，より XML を有効に利用するためにはテキスト以外のデータ型を持つ XML も利用できることが望ましい．そこで本論文ではスキーマによって XML 文書のノードに文字列以外の型が指定された XML 文書を関係データベースへ格納し，検索する方法を提案する．スキーマは RELAX NG を選ぶ．RELAX NG では詳細なスキーマ設計ができる．また関係データベースへの格納と検索に関しては XRel という関係データベースに XML 文書を格納，検索するシステムがあるのでそれを利用して RELAX NG スキーマに従う XML 文書に対してのサポート手法を提案する．

キーワード　XML データベース

Approach to Storing XML Documents which Conforms to Document Schema in a Relational Database

Norimasa TERADA† and Masatoshi YOSHIKAWA††

† Dept. of Information Engineering, School of Engineering
†† Information Technology Center
Nagoya University
Furo-cho Chikusa-ku, Nagoya, Aichi, 464-8601, Japan
E-mail: †terada@dl.itc.nagoya-u.ac.jp, ††yosikawa@itc.nagoya-u.ac.jp

Abstract　Today, XML is becoming widely used, but if we want to use XML more useful, XML had better be used not only text. So this paper suppose an approach which store XML documents in a relational database, and search from the database. But XML documents are conformed RELAX NG schema. The reason we choosed RELAX NG as schema of XML documents is that RELAX NG can design detailed schema. And we consider an approach to extend XRel system, which manages XML documents in a relational database.

Key words　XML Database

1. はじめに

XML(Extensible Markup Language) は文書やデータの意味や構造を記述するためのマークアップ言語の一つである．テキストで記述できるためデータの受け渡しが容易に行えるという利点から，現在 XML が利用される範囲は広がりつつある．しかし，より XML を利用しやすいようにするためには XML 文書をただの文字列データの集まりとして扱うだけではなく，文字列以外の型のデータとしても扱えるようになることが XML をより有効にするために必要である．

XML 文書のスキーマとしては DTD(Document Type Definition) が広く使われているが，DTD にはノードの型を指定することができないという欠点がある．そこで本稿ではノードの型を詳細に指定することのできる RELAX NG をスキーマとして選び，そのスキーマに従う XML 文書を関係データベースへ格納して検索するための手法を提案する．

現在，XML 文書を関係データベースで扱う XRel [?], [1] というシステムがあるため，それを改良して RELAX NG スキーマに従う XML 文書をサポートできるようにする．

したがって本論文ではまず XRel の説明をし，そのあとに XRel を RELAX NG スキーマに従う XML 文書をサポートするためにどのように拡張していくかを述べる．

2. XRel

ここでは XRel の説明として XML 文書の格納方法を簡単に説明する．

2.1 XRel における XML 文書の格納方法

XRel では XML 文書を木でモデル化し，そのモデル化された木のそれぞれのノード単位でデータベースに格納する．また，格納する時にはノードの種類に分けて表を作っている．要素ノードは Element, 属性ノードは Attribute, テキストノードは Text という名前の表に格納される．また経路を格納する Path という名前の表も用意されている．それぞれの表は以下のような構成になっている．

a） Element

文書 ID(docID), 経路 ID(pathID), 同じ親ノードをもつ兄弟ノード間における正順序情報 (index), 出現逆順序情報 (reindex), 出現位置 (pos) から構成される．キーは docID,pos である．

b） Attribute

docID,pathID,attribute,pos から構成され，それぞれ文書 ID, 経路 ID, 属性値, 出現位置を格納する．キーは docID,pathID,pos である．

c） Text

docID,pathID,value,pos から構成され，それぞれ文書 ID, 経路 ID,XML で規定された文字データの集まり, 出現位置を格納する．キーは docID,pos である．

d） Path

pathexp,pathID から構成され，それぞれ単純経路, 経路 ID を格納する．キーは pathexp である．

例えば，図 1 のような XML 文書を例に考えてみる．

```
<books>
  <book style="textbook">
     <title>Designing XML applications</title>
     <editor>
        <family>Bob</family>
        <given>Kraft</given>
     </editor>
     <author>
        <family>Nick</family>
        <given>Marcus</given>
        <family>Bob</family>
        <given>Pant</given>
     </author>
     <summary>
        This book is the guide to design
        <keyword>XML</keyword>applications.
     </summary>
  </book>
</books>
```

図 1　XRel の説明に用いる XML 文書の例

これを表に格納した結果の表が表 1 から表 4 にある．

表 1　Element

docID	pathID	index	reindex	pos
1	1	0	-1	0.1,18.3
1	2	0	-1	0.2,18.2
1	4	0	-1	0.3,3.1
1	5	0	-1	3.2,5.2
1	6	0	-1	3.3,4.1
1	7	0	-1	4.2,5.1
1	8	0	-1	5.3,9.2
1	9	0	-2	5.4,6.1
1	10	0	-2	6.2,7.1
1	9	1	-1	7.2,8.1
1	10	1	-1	8.2,9.1
1	11	0	-1	9.3,18.1
1	12	0	-1	16.1,17.1

表 2　Attibute

docID	pathID	attribute	pos
1	3	textbook	0.2,0.2

表 3　Text

docID	pathID	value	pos
1	4	Designing XML	1,3
1	6	Bob	4,4
1	7	Kraft	5,5
1	9	Nick	6,6
1	10	Marcus	7,7
1	9	Bob	8,8
1	10	Pant	9,9
1	11	This book is ...	10,16
1	12	XML	17,17
1	11	applications	18,18

表 4　Path

pathexp	pathID
/books	1
/books/book	2
/books/book/@style	3
/books/book/title	4
/books/book/editor	5
/books/book/editor/family	6
/books/book/editor/given	7
/books/book/author	8
/books/book/author/family	9
/books/book/author/given	10
/books/book/summary	11
/books/book/summary/keyword	12

3. XRel の拡張手法

本節では XRel が RELAX NG スキーマに従う XML 文書をサポートするための拡張の手法について述べる．RELAX NG スキーマの持つ大きな特徴はデータ型の定義であり，これにより文字列ではない型のノードを定義できる．RELAX NG ではデータ型以外にも様々に詳細なスキーマ設計が可能であるがここではデータ型のみに注目する．

3.1 RELAX NG の基本的な項目のサポート

まず RELAX NG スキーマの中でデータ型などの詳細な定義のない基本的なスキーマに従う XML 文書をどのように扱うかをここでは説明する．またここで図 2 の RELAX NG スキーマとそのスキーマに従う図 3 の XML 文書を例にとって説明する．

```
<element name="addressbook"
   xmlns="http://relaxng.org/ns/structure/0.9">
   <zeroOrMore>
      <element name="card">
         <element name="name"><text/></element>
         <oneOrMore>
            <element name="email"><text/></element>
         </oneOrMore>
         <oneOrMore>
            <element name="email">
               <element name="account"><text/></element>
               <element name="server"><text/></element>
            </element>
         </oneOrMore>
      </element>
   </zeroOrMore>
</element>
```

図 2　RELAX NG スキーマの例

```
<addressbook>
     <card>
          <name>Masatoshi Yoshikawa</name>
          <email>yoshikawa@itc.nagoya-u.ac.jp</email>
     </card>
     <card>
          <name>Norimasa Terada</name>
          <email>terada@dl.itc.nagoya-u.ac.jp</email>
          <email>
               <account>h013104b</account>
               <server>mbox.nagoya-u.ac.jp</server>
          </email>
     </card>
</addressbook>
```

図 3　XML 文書の例

図 2 のスキーマは次のような文法によって生成される．ここで文法の開始記号を **S** とする．

$$\mathbf{S} \rightarrow addressbook(\mathbf{A}*)$$

$$\mathbf{A} \rightarrow card(\mathbf{B}, \mathbf{C}+, \mathbf{D}+)$$

$$\mathbf{B} \rightarrow name(\varepsilon)$$

$$\mathbf{C} \rightarrow email(\varepsilon)$$

$$\mathbf{D} \rightarrow email(\mathbf{E}, \mathbf{F})$$

$$\mathbf{E} \rightarrow account(\varepsilon)$$

$$\mathbf{F} \rightarrow server(\varepsilon)$$

3.1.1　XML 文書の格納

一般に RELAX NG スキーマは必ず木でモデル化することができるとは限らないがここではそれが可能であるスキーマを前提とする．その前提の下で図 2 のスキーマを木でモデル化したものが図 4 にある．

図 4 の木モデルを深さ優先で探索していく．探索の中で要素ノードつまり，element という名前の要素ノードに注目する．element という名前の要素ノードを見つけたら，その name 属性を調べて，それを Path 表に格納する．実際，zeroOrMore タグに囲まれている内容は 0 個の場合も許すので，addressbook 要素だけの XML 文書も存在しうる．

その場合は Path 表への格納が無駄になるが，そのコストの大きさは重要ではないこととデータ型を考える時に有用であるため出現する可能性のある経路は全て Path 表に格納する．

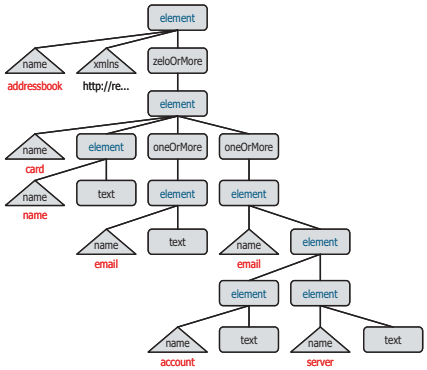

図 4　RELAX NG スキーマの木モデルの例

またこのスキーマの文法からわかるように email ノードには非終端記号 **C** から派生するものと **D** から派生するものがある．これに関しても区別して格納したい．つまり account,server を子ノードに持つ email と何も持たない email では異なる pathID を付与して格納するということである．ゆえに既存の XRel では pathexp が Path 表の主キーであったが，これを変更して pathID を主キーとする．さらに文法の違いを識別するために reverse 属性を追加する．この属性はルートノードまで上にたどった時に通るノードの pathID を順に格納した属性である．デリミタには”♯”を使う．

こうすることによって例えば//email[account=”terada”](値が terada である account を子に持つ email ノード) を検索したい時に **C** から派生する email ノードは全く調べる必要がない．この探索コストを削ることは大きな効果がある．**C** から派生する email ノードの数が大きいほどその効果は大きい．

例に挙げた RELAX NG スキーマでは図 5 のようになる．

pathexp	pathID	reverse
/addressbook	1	1
/addressbook/card	2	2♯1
/addressbook/card/name	3	3♯2♯1
/addressbook/card/email	4	4♯2♯1
/addressbook/card/email	5	5♯2♯1
/addressbook/card/email/account	6	6♯5♯2♯1
/addressbook/card/email/server	7	7♯5♯2♯1

図 5　図 4 の木モデルから得られる Path 表

例に挙げた文書を Element 表に格納すると図 6 のようになる．また経路表現的な問合せ処理を高速化するために経路表現に関する B+木索引を作る．その索引を図 7 に示す．Text 表や Attribute 表も同じように経路表現に関する B+木索引を作る．

3.1.2　XML 文書の検索

本節では XPath 形式の問合せから SQL 形式への変換方法について考える．Element,Text,Attribute 表に対してそれぞれ pathID に関する B+木索引を作る．作られた索引は関係最適化器が問合せに応じて適切に利用してくれる．

本節では同じ経路表現で派生する文法が異なるノードに対して異なる pathID を付与したが，これを検索でどのように利用

docID	pathID	index	reindex	pos
1	1	0	-1	0.1,8.4
1	2	0	-2	0.2,3.2
1	3	0	-1	0.3,2.1
1	4	0	-1	2.2,3.1
1	2	1	-1	3.3,8.3
1	3	0	-1	3.4,5.1
1	4	0	-2	5.2,6.1
1	5	1	-1	6.2,8.2
1	6	0	-1	6.3,7.1
1	7	0	-1	7.2,8.1

図 6　Element 表の例

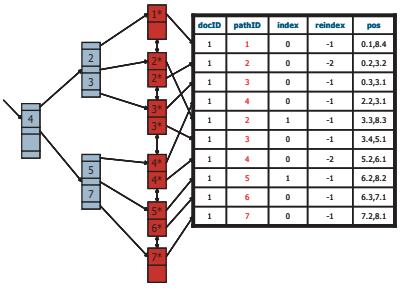

図 7　Element 表に対する B+木索引の例

するかに論点をおく．そこで次の問合せを例として説明する．

例　/addressbook/card/email[account=”h013104b”]

この問合せでは/addressbook/card/email と/addressbook/card/email/account で分けて考える．前者は Element 表に問い合わせて後者は Text 表に問い合わせることになる．

/addressbook/card/email/account については条件を value が”h013104b”として選択をほどこす．

Path 表には/addressbook/card/email という経路表現の行が二つあったが，一つは子要素ノードを持たず，もう一つは account と server という子要素ノードを持つ．例の問合せでは子要素ノードを持つ方だけを検索する必要がある．

reverse 属性を利用するために新しい関数 parent を導入する．

```
pathID parent(reverse x,int y)
```

引数として reverse 属性の x と int 型の y をとる．y はどれだけ上にノードをたどるかを示す．戻り値は pathID で x という reverse 属性を持つ経路から y だけ上にたどったときの値という意味である．例えば，parent(#6#5#2#1,1) は 5 という pathID を返す．内部的な処理としては reverse 属性の#を左から y 個取り除いたときに一番左に出現する数字を取り出す．

例の問合せにおいて reverse 属性を使うことで子要素ノードを持つ email ノード (Path 表では pathID が 5) を検索することができる．SQL 文を図 8 に示す．

3.2　データ型のサポート

RELAX NG の大きな特徴はこのデータ型にある．それにより XML 文書をテキストだけではなく，真にデータといった観点で扱うことができる．よって XRel においてもまずデータ型をサポートできるようにしたい．まずはデータベースの型に対応する型が存在する RELAX NG のデータ型をサポートすることを考える．

そのためには XRel の Text,Attribute 表にはどのような型のデータでも格納できるように表の列を作っておく必要がある．

```
SELECT    e.docID,e.pos
FROM      Element e,Text t,Path p,Path p2
WHERE     p.pathexp LIKE '/addressbook/card/email'
AND       p2.pathexp LIKE
                '/addressbook/card/email/account'
AND       t.value =‘ h013104b '
AND       e.pathID = p.pathID
AND       t.pathID = p2.pathID
AND       t.docID = e.docID
AND       e.pos.contain(t.pos)
AND       p1.pathID = parent(p2.reverse,1)
ORDER BY  e.docID,e.pos
```

図 8　例の問合せに対する SQL 文

その格納の方法としては以下のように三つ考えられる．

（1） 一つの Text(Attribute) 表に全ての型のデータを格納できる列を作る

（2） 元の Text(Attribute) 表は一つとっておいて，型ごとに表を Text 表と同様に作る

（3） Text 表を型の数だけ作り，文字列以外の型に対応する表にその型の value 属性を追加する

その三つの案で XRel を改良する場合に格納と検索の点でどのような利点と欠点があるかを考える．それにあたり図 9 に示すような RELAX NG スキーマを例として説明する．このスキーマは商品カタログを表すスキーマで discount には float 型と int 型のノードを別個に用意している．int 型の discount は値引きする料金を整数型で格納し，float 型の discount は値引きする割合を小数で格納する．

```
<element name="catalog" xmlns="http://relaxng.org/ns/structure/0.9"
   datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
   <zeroOrMore>
      <element name="item">
         <element name="id"><text/></element>
         <element name="name"><text/></element>
         <element name="price"><data type="int"/></element>
         <element name="discount"><data type="float"/></element>
         <element name="discount"><data type="int" /></element>
      </element>
   </zeroOrMore>
</element>
```

図 9　データ型を用いた RELAX NG スキーマの例

また図 9 のスキーマに従う XML 文書を図 10 に示す．そしてこの文書の中から価格が 1000 より小さい商品の全ての情報の抽出を例として後に挙げる三つの方法でどのように XML 文書を格納し，XPath 式を SQL 形式に変換して問い合わせるかを説明する．XPath 式は以下のようになる．

例 /catalog/item[price<1000]

問合せの内容としては/catalog/item/price を int 型のデータとしてみた時に 1000 より小さい値になる price ノードの親ノード item 以下を出力することになる．この場合 price ノードをテキストとしてだけ見ていると文字列での大小比較と数値での大小比較に違いがあり，正確に問合せに答えられない．例えば 800 と 1000 を比較したとき，数値では 800 の方が小さいが文字列としては 1000 の方が小さくなる．

```
<catalog>
     <item>
          <id>00000001</id><name>glass</name>
          <price>540</price><discount>0.1</discount>
          <discount>60</discount>
     </item>
     <item>
          <id>00000002</id><name>table</name>
          <price>1200</price><discount>0.2</discount>
          <discount>300</discount>
     </item>
</catalog>
```

図 10　図 9 に従う XML 文書の例

またどの案でもスキーマを解析して考えられる経路表現を Path 表に格納するというのは共通している．ここで Path 表を拡張する．type 属性を追加する．これはその経路表現がどのようなデータ型になるかを表す．実際には，型ごとに対応する番号を決めておいてその数字を格納するようにする．

この例では経路が同じでありながら型が違う discount ノードが存在する．こういったノードに対して Path 表においては違う pathID を付与する．表 5 に例のスキーマを Path 表に格納した結果を示す．

表 5　データ型を拡張した Path 表の例

pathexp	pathID	reverse	type
/catalog	1	1	0(*)
/catalog/item	2	2♯1	0(*)
/catalog/item/id	3	3♯2♯1	0(*)
/catalog/item/name	4	4♯2♯1	0(*)
/catalog/item/price	5	5♯2♯1	1(int)
/catalog/item/discount	6	6♯2♯1	2(float)
/catalog/item/discount	7	7♯2♯1	1(int)

3.2.1　All-Type 法

一つの Text 表に全ての型のデータを格納することができるように設計している．利点としては問合せが単純になることである．表が一つですむので索引も一つですむことが利点である．欠点は前提として型としては一つしか選べないとしていることと，実際いくつもの型を選べるようにすることは少ないことから表の中で値が NULL になる列が多くなってほとんどの表の領域が無駄になることである．

先に挙げた XML 文書ではどのようにノードが Text 表に格納されるか表 6 を示す．

例の問合せを SQL 文に変換する．述語の部分は Text 表に問い合わせて，経路部分に関しては Element 表に問い合わせる．この例では/catalog/item に関しては Element 表で，/catalog/item/price に関しては Text 表になる．別々に検索して後で先祖子孫の包含関係を contain 関数で確かめればよい．ここ

表 6　All-Type 法における Text 表

docID	pathID	value	value_int	value_float	⋯	pos
1	3	'00000001'	NULL	NULL	⋯	1,1
1	4	'glass'	NULL	NULL	⋯	2,2
1	5	'540'	540	NULL	⋯	3,3
1	6	'0.1'	NULL	0.1	⋯	4,4
1	7	'60'	60	NULL	⋯	5,5
1	3	'00000002'	NULL	NULL	⋯	6,6
1	4	'table'	NULL	NULL	⋯	7,7
1	5	'1200'	1200	NULL	⋯	8,8
1	6	'0.2'	NULL	0.2	⋯	9,9
1	7	'300'	300	NULL	⋯	10,10

で重要なのはどのようにデータ型に関する問合せであるかを XPath 形式の問合せから判断し，その判断した情報から関係データベースから適切な型のデータを抽出するかということである．

そのためには XPath 形式で算術演算子が出現した時に両方のオペランドを見てどのような型に関する問合せであるかを判断する．片方が定数であった場合には定数の型と一致する型のデータを調べるようにする．両方がノードであった場合には両方のデータ型が等しいならば演算をする．

図 11 にこの手法で例の問合せに答える SQL 文を示す．type 属性を問い合わせることによって int 型のデータを持つノードであることを確かめている．

```
SELECT    e.docID,e.pos
FROM      Element e,Path p,Text t,Path p2
WHERE     p.pathexp LIKE '/catalog/item'
AND       e.pathID = p.pathID
AND       p2.pathexp LIKE '/catalog/item/price'
AND       t.pathID = p2.pathID
AND       t.value_int < 1000
AND       e.docID = t.docID
AND       e.pos.contain(t.pos)
AND       p1.pathID = parent(p2.pathID,1)
ORDER BY e.docID,e.pos
```

図 11　All-Type 法での SQL 文

3.2.2　Single-Type 法

型ごとに Text 表と同じように作る．表の数が多くなる欠点があるが，領域の効率はこの案は優れている．なぜなら表に指定された型のノードの数しか格納しないからである．表 7 に例の XML 文書をこの方法で Text 表に格納した結果を示す．

この方法で例の XPath 式を SQL 文に変換する方法を考える．この場合も経路部分に対しては同様に Element 表を使い，その経路表現の条件は Path 表を使って確かめる．例においては/catalog/item は Element 表を使う．述語の部分は Text 表を使う．ただし int 型のデータに関する述語なので int 型のデータを格納している Text 表に問い合わせなければならない．

SQL 文は図 12 のようになる．

この方法では表の中の NULL 値はないので領域の無駄はな

表 7　Single-Type 法における Text 表

docID	pathID	value	pos
1	3	'00000001'	1,1
1	4	'glass'	2,2
1	5	'540'	3,3
1	6	'0.1'	4,4
1	7	'60'	5,5
1	3	'00000002'	6,6
1	4	'table'	7,7
1	5	'1200'	8,8
1	6	'0.2'	9,9
1	7	'300'	10,10

docID	pathID	value_int	pos
1	5	540	3,3
1	7	60	5,5
1	5	1200	8,8
1	7	300	10,10

docID	pathID	value_float	pos
1	6	0.1	4,4
1	6	0.2	9,9

```
SELECT   e.docID,e.pos
FROM     Element e,Path p,Text_int ti,Path p2
WHERE    p.pathexp LIKE '/catalog/item'
AND      e.pathID = p.pathID
AND      p2.pathexp LIKE '/catalog/item/price'
AND      ti.pathID = p2.pathID
AND      ti.value_int < 1000
AND      e.docID = ti.docID
AND      e.pos.contain(ti.pos)
AND      p1.pathID = parent(p2.pathID,1)
ORDER BY e.docID,e.pos
```

図 12　Single-Type 法での SQL 文

いのが利点であるが，ノードの数より tuple の数の方が多いのでまだ領域を減らすことができる．その方法を次に説明する．

3.2.3　Double-Type 法

この方法ではまず Text 表と型ごとの別の Text 表を用意する点に関しては Single-Type 法と共通である．異なる点は別の Text 表には文字列の value 属性とその表で管理する型の value 属性の二つの型の value 属性を持たせる点にある．他にも一つのノードに対して一つの行しか生成しない点が異なる．そのために文字列型のノードに関しては Text 表に格納して，他の型のノードはその型に対応する表に格納する．

表 8 に格納した結果の例を示す．

表 8　Double-Type 法における Text 表

docID	pathID	value	pos
1	3	'00000001'	1,1
1	4	'glass'	2,2
1	3	'00000002'	6,6
1	4	'table'	7,7

docID	pathID	value	value_int	pos
1	5	'540'	540	3,3
1	7	'60'	60	5,5
1	5	'1200'	1200	8,8
1	7	'300'	300	10,10

docID	pathID	value	value_float	pos
1	6	'0.1'	0.1	4,4
1	6	'0.2'	0.2	9,9

この方法で検索する場合には表と列の両方を意識して適切に問い合わせる必要がある．SQL 問合せを図 13 に示す．

```
SELECT   e.docID,e.pos
FROM     Element e,Path p,Text_int ti,Path p2
WHERE    p.pathexp LIKE '/catalog/item'
AND      e.pathID = p.pathID
AND      p2.pathexp LIKE '/catalog/item/price'
AND      ti.pathID = p2.pathID
AND      ti.value_int < 1000
AND      e.docID = ti.docID
AND      e.pos.contain(ti.pos)
AND      p1.pathID = parent(p2.pathID,1)
ORDER BY e.docID,e.pos
```

図 13　Double-Type 法での SQL 文

3.2.4　三つの手法の比較

これまでにデータ型をサポートするために考えた三つの格納手法を比較する．具体的な数値を使った比較は後の章で行う．

まず領域に関して三つの手法を比較したとき，All-Type 法は表の数は一つであるが，サイズが大きく表の中に NULL が多くなるため領域の無駄が多い．Single-Type 法と Double-Type 法は表の数はサポートする型の数だけ表を用意する必要があるが，それぞれの表に NULL はなく，領域の無駄はない．またそれぞれの表の列数に注目した時に Single-Type 法が最も小さい．表の行数に注目した時には Double-Type 法はノードの数と関係表の行数は等しくなって行数が最小になる．その反面，Single-Type 法は行数が多くなっているため容量が大きくなる．

表のサイズが問合せに及ぼす影響を考えてみる．XPath 形式の問合せを SQL 形式に変換した時に必ず複数の表の結合を伴う SQL 問合せになる．関係データベースにおいて結合はコストの大きい処理である．結合のコストは表の大きさや結合の条件に依存する．結合の条件を変えることはできないので結合する表のサイズを小さくすることを考える．

表のサイズは列のバイト数と行の数の積で決定される．したがって問合せを考えた時にはそれぞれの表のサイズが最小になる Single-Type 法がよいと考えられる．

Single-Type 法において問合せ処理のコストが最悪になる場合は同じ値 (例えば整数型) をそのノードの型 (整数型) と文字列型の両方で評価をしたい場合には同じノードでも二つの表に問い合わせなければならない．それに対して他の手法は一つの表で二通りの評価ができるため無駄なコストを省ける．しかしそういう問合せの頻度は実用上低いと考えられるため一般には Single-Type 法が問合せに関しては最もよいと考えられる．

4.　実装に関する考察

これまで述べてきたことに対する実装を実際に考えた時に必要になることがどのように RELAX NG スキーマを解析して Path 表を作るかということである．現在は Relaxer [3] というツールを使ってそれを行う方法を考案中である．

Relaxer とは浅海智晴氏が開発したスキーマコンパイラであ

る．RELAX NG スキーマを入力するとその定義をマップした Java クラスを生成する．この Java オブジェクトを Relaxer では Relaxer オブジェクトと呼ぶ．Relaxer オブジェクトは XML 文書との相互変換を行うことができる．

また Relaxer はコマンドオプションを指定することによって Java クラス以外の形式の出力を生成することもできる．入力した RELAX NG スキーマに対応する DTD や，JDBC アクセスクラス，Relaxer オブジェクトをモデル化したメタデータを出力することもできる．

現在は，Relaxer によって出力されるメタデータを用いて Path 表を作成することを考えている．

5. 三つの手法の比較実験

データ型をサポートするために考えた三つの格納手法を実験した結果を比較する．XMark [4] で XML 文書を生成し，その一部を用いる．XMark の DTD スキーマを Trang [5] を用いて RELAX NG に変換して，適宜データ型を指定する．実験では int,float,date,time 型の四つの型を扱う．

また Relaxer メタモデルを解析して XML 文書を格納する Java クラスを自動生成するプログラムはまだ作成できていないため，自動生成したプログラムではなく，自分で作った XML 文書を格納するプログラムを用いて XML 文書を格納した．

実験に使用した XML 文書は一つで容量は 266KB，要素ノード数は 5540 個，属性ノード数は 1188 個，テキストノード数は 3112 個，単純経路の種類は 35 種類である．この文書に対してそれぞれの手法において格納と検索の性能を比較する．

5.1 格納の比較実験

先に述べた XML 文書をそれぞれの手法で関係データベースへ格納する．このときの関係表の行数と格納するために生成する CSV ファイルの容量を表 9 に示す．この実験では Text 表以外に関してどの手法も共通の CSV ファイルを出力するため Text 表に格納するための CSV ファイルの領域を比較する．今回用いた文書では Text 表以外の Element,Attribute,Path 表に対応する CSV ファイルの合計容量は 162KB である．

表 9　三つの手法の CSV 容量比較

	CSV ファイルの容量 (KB)
All-Type 法	225
Single-Type 法	260
Double-Type 法	215

この表から行数が最も多い Single-Type 法が領域が最も大きく，表の中に NULL がなく行数もノード数と等しい Double-Type 法が最も領域が小さいことがわかる．さらに All-Type 法の方が Single-Type 法より領域が小さいことから，領域を左右する要因として列数よりも行数の方が大きいことがわかる．文字列以外の型を持つノードの数が少なければ Single-Type 法の領域は All-Type 法よりも小さくなる可能性があるが，実際に使用する場合にはそのようなことは起こりにくいのでやはり Single-Type 法が容量は最も大きくなる．

また参考文献 [6] による表サイズ見積もり方法を用いたときには表 10 のような結果が得られた．

表 10　三つの手法の表サイズ比較

	表サイズ (byte)
All-Type 法	98304
Single-Type 法	139624
Double-Type 法	114688

この見積もり方法では詳細なサイズまでは見積もることができない．また表の数が多くなると見積もりサイズが大きくなる方法であるため表の数が小さい All-Type 法にとって有利な見積もりになってしまう．この結果からわかることは Double-Type 法の方が Single-Type より表サイズも小さくなることである．この二つの方法は表の数も同じなのでこの方法で比較できる．

5.2 検索の比較実験

今度は XML 文書に対してそれぞれの手法に対して問合せ処理速度を比較する．それぞれの手法で生成される関係表に対して XPath 式から変換される SQL 問合せを実行する．float 型のデータの値を条件とする問合せと複数の型のデータを条件とする問合せのそれぞれで実行速度を比較する．それぞれの XPath 問合せ式を表 11 に示す．

表 11　実験に用いた問合せ

	XPath 式
Q1	/site/open_auctions/open_auction[initial<100.0]
Q2	/site/open_auctions/open_auction[initial<100.0 AND quantity<4]

実験環境は CPU:Intel Pentium M processor 1GHz，メモリ:760MB，OS Microsoft Windows XP Professional である．使用した DBMS は Oracle9i Release 1 である．問合せ実行時間は SQL*Plus の SET TIMING で計測した．

それぞれの手法において表 11 の問合せに対して 3 回実行した平均の実行時間を計測した結果を表 12 に示す．

表 12　三つの手法の検索速度の実験結果

	All-Type 法 (ms)	Single-Type 法 (ms)	Double-Type 法 (ms)
Q1	15	15	15
Q2	39	30	30

Text 表の結合は一つだけですむ Q1 ではどの手法においても速度は同じであった．それに対して Q2 では All-Type 法のみが速度が遅く，他の手法では同じ速さという結果が得られている．これは結合の数が問合せ処理速度を決める重要な要因であることと結合する表の大きさが問合せ処理速度を左右することを示している．All-Type 法は表のサイズが他の方法に比べて大きくなることから問合せ処理が遅くなるといえる．

これらの結果をまとめて考えると格納の点では Single-Type 法が最も容量が大きくなるが，それぞれの表のサイズは小さい．All-Type 法は表は一つであるがサイズが大きく，無駄な領域も多い．領域の点では Double-Type 法が最も優れている．

検索の点では結合が重要であり，結合の速度を左右するのは表のサイズである．そして表のサイズが最も大きいのは All-Type 法であり，問合せが複雑になるほどその影響は大きくなる．それに比べて Single-Type 法と Double-Type 法は表のサイズが小さく，問合せ処理を高速化できる．検索に関しては表のサイズが最も小さい Single-Type 法が期待できる．

6. 関 連 研 究

本節では XML 文書を関係データベースへ格納する他の論文を紹介する.

6.1 参考文献 [7]

この論文でのシステムは ShreX という名前が付けられている．XML スキーマと関係データベースにおける表の対応に関しては，XML スキーマに注釈を付けることで関係データベースにどのようなスキーマで対応付けるかをユーザが定義する.

注釈が付けられたスキーマの具体例を図 14 に示す.

```
<element name="SHOW" shrex:structurescheme="Dewey">
     <sequence>
          <element name="TITLE" type="string"
               shrex:outline="true"
               shrex:tablename="Showtitle"/>
          <element name="YEAR" type="integer"
               shrex:outline="false"
               shrex:columnname="Showyear"
               shrex:sqltype="NUMBER(4)"/>
     </sequence>
</element>
```

図 14　注釈を付けた XML スキーマの例

図 14 のように注釈部分には shrex の名前空間を用いて区別できるようにしている．詳細については紙面のスペース上省略する．ちなみに outline を true にしてある要素はその要素を一つの表に対応付け，false の場合には表の属性として対応付ける.

図 15 に図 14 のスキーマから構成される関係データベースのスキーマを示す.

```
TABLE SHOW(ID VARCHAR(128),    TABLE Showtitle(ID VARCHAR(128),
           Showyear NUMBER(4))                 ParentID VARCHAR(128),
                                               TITLE VARCHAR(512))
```

図 15　図 14 に対応する関係データベーススキーマ

この論文の問題点は XPath 形式の問合せの階層が深いほど関係スキーマに対する問合せも難しくなる．また XML 文書の復元はコストが大きい．そもそもこの論文は問合せに対するサポートについてあまり述べられていない．それに対して我々が提案する手法は関係データベースに XML 文書のノードの位置情報を問い合わせてそこから部分文書を返すことを目的としている.

6.2 参考文献 [8]

この論文でのシステムは LegoDB という名前が付けられている．LegoDB 特有の特徴は使うアプリケーションに対して適切なスキーマの対応付けを行うことである．元のスキーマの情報を保持し，かつ簡単に関係スキーマに変換できる p-schema と呼ぶスキーマに変換する.

図 16 から図 18 に，順に対応付けしたいスキーマの例，それを書き直した p-schema，関係スキーマを示す.

```
type Show =
     show[@type[String],
          title[String],
          year[Integer],
          reviews[String]*,
          ...]
```

図 16　対応付けしたい XML スキーマ

```
type Show =                        type Reviews =
     show[@type[String],                reviews[String]
          title[String],
          year[Integer],
          Reviews*,
          ...]
```

図 17　p-schema

```
TABLE Show                         TABLE Review
    (Show_id INT,                      (Review_id,
     type STRING,                       review String,
     title STRING,                      Parent_Show INT)
     year INT)
```

図 18　対応付けされた関係スキーマ

この例では p-schema で一つの属性として対応付けられない要素を外に出すことによって単純な変換が可能な形にしている．またアプリケーションに対して適切なスキーマを作るためのスキーマ書き換え規則を適用して XML スキーマを変換して，関係スキーマがアプリケーションに対して適切になるようにしている.

この論文もアプリケーションで XML 文書を扱うための適切なスキーマの設計を目的としているため XML 文書の復元は難しい.

文　　献

[1] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, Shunsuke Uemura：XRel: A Path-Based Approach to Storage and Retrieved of XML Documents Using Relational Database, ACM Transactions on Internet Technology, Vol.1, No.1, pp.110-141(2001).

[2] RELAX NG 入門
http://www.kohsuke.org/relaxng/tutorial.ja.html：
James Clark, 村田真, 川口耕介 (日本語訳)

[3] Relaxer
http://www.relaxer.org/

[4] XMark - An XML Benchmark Project.
http://www.xml-benchmark.org/

[5] Trang - Multi-format schema converter based on RELAX NG
http://thaiopensource.com/relaxng/trang.html

[6] 領域サイズの見積もり方法
http://otndnld.oracle.co.jp/deploy/maintenance/pdf/size_est.pdf

[7] S. Amer-Yahia, F. Du, and J. Freire. A Comprehensive Solution to the XML-to-Relational Mapping Problem In *Proceedings of the 6th Annual ACM International Workshop on Web Information and Data Management*, pages 31-38, November 12-13, 2004.

[8] P. Bohannon, J.Freire, P. Roy, and J. Siméon. From XML Schema to Relations: A Cost-Based Approach to XML Storage. In *Proceedings of the 18th International Conference on Data Engineering*, pages 64-75, February 26-March 1 2002.