社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－DBS－134 (II) (40)
2004／7／14

リージョンディレクトリを用いた関係データベースによる大規模XMLデータ処理

天笠俊之† 植村俊亮†

本稿では，関係データベースを基盤としたXMLデータベースにおいて，大規模XMLデータの効率的な処理を実現するための手法を提案する．大規模XMLデータでは，その性質からサイズ以上にノード数が増加するため，関係データベース上での問合せ処理が効率的に行えないという問題がある．しかしながら，多くの場合，ある問合せを処理のにXMLデータ全体は必ずしも必要ではなく，問合せに応じたデータの一部分さえあれば十分であることが多い．本稿では，このデータアクセスの局所性をうまく利用するために，ディスク上に置かれたXMLデータから，問合せ処理に必要な部分のみを関係表へ動的にマッピングする手法を考案した．これはStrong DataGuideに基づいたデータ構造であるリージョンディレクトリによって実現される．すなわち単純経路式から対応するXML部分データを知ることができ，その部分を関係表への動的にマッピング（アンマッピング）することによって，問合せ処理に不要な部分を関係表の上におく必要がなくなる．関係表のタプル数が減少するため，全体を一様にマップする従来手法より効率的な検索処理が期待できる．

Processing Large-Scale XML Data by RDBMS using Region Directory

Toshiyuki Amagasa † and Shunsuke Uemura†

This paper proposes a scheme for processing large-scale XML data efficiently by RDBMS-based XML databases. From the nature of XML data, their data size gives serious impact on the performance of query performance on RDBMS, due to the fact that the number of nodes grows quickly as the data size increases. However, in many cases, the entire XML data is not always necessary for processing XML queries, that is, only small portion is enough for answering the queries. This paper, therefore, attempts to develop a scheme for dynamically map (unmap) partial XML data to (from) relational tables that are necessary for given queries. To this end, a data structure called "region directory" is introduced. In fact, region directory is based on a well-known data structure, Strong DataGuides, that is used to compute summary of graph-structured data. A region directory enables us to know the regions of XML data related to a path expression. We can thus dynamically map (unmap) partial XML data to (from) relational tables. As a consequence, relational tables only contains relatively small numbers of tuples enough for processing particular XML queries, and this leads more efficient query processing than existing schemes.

1. はじめに

XML[1]は，1998年にW3C勧告となってから現在までの間に急速に普及が進み，計算機間の文書（データ）交換のフォーマットとして標準的に用いられるようになった．これに従い，XMLが対象とする応用も多様化が進んでいる．従来想定されていた構造化文書に加えて，最近では，SOAPのようなリモート手続き呼び出しの記述，MPEG-7やSVGのようなマルチメディアデータ記述，RDF, RSSのようなメタデータ記述，Webサーバやルータのログ記述のために用いられている．

XMLの応用範囲が多様化するとともに，そのデータサイズも拡大している．従来は，数KBから数MBの比較的小規模なデータを対象に用いられることが多かったが，最近では数十MBから数GBにもなる規模の大きなデータをXMLで記述する事例が出始めている．例えば，The Open Directory Project (ODP)[2]では，フリーのWebディレクトリデータをRDF/XMLフォーマットで配布しているが，そのサイズはディレクトリ構造に関する部分だけでも503 MB．コンテンツを含めると1.8 GBにもなる．また，分子生物学における遺伝子産物の統制語彙であるGene Ontology™[3]では，遺伝子産物と用語の関連を記述したXMLファイルのサイズは61 MBとなっている．この

† 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Infomation Science, Nara Institute of Science and Technology

ような大規模な XML データを効果的に運用するには DOM や SAX 等の API では役者不足であり，専用のシステムをゼロから構築することを除けば，XML データベースを利用するのは妥当な選択の一つであると言える．

XML データベースを構築する方法は多くの提案があるが，大別すると以下の三種類に分類できる: 1) 専用システムを構築する方法，2) オブジェクトデータベースを用いる方法，3) 関係データベースを用いる方法．この中でも関係データベースに基づく手法は，問合わせ最適化やトランザクション管理等の技術の蓄積があることや，システムが広く普及し既存の情報資源が豊富であることなどから有力であるとされている．このため，関係データベースに基づいた XML データベースの実現手法に関する研究はこの数年数多くなされている他，ほとんど全ての商用関係データベースシステムが XML をサポートしている．しかしながら，既存の手法は小規模の XML データを想定して実装されていることが多く，大規模な XML データでは処理効率が極端に悪かったり，そもそもデータベースへの格納が不可能であることが多い．

既存の手法において大規模 XML データの処理効率が悪化する主要な原因の一つに，XML データの特性を挙げることができる．すなわち，XML は木構造（ハイパーリンクを考慮するとグラフ構造）を有するため，サイズの増加に伴うノード数の増加が顕著である．例えば，XML データのサイズが 2 倍に増加した場合，ノード数は 2 倍以上となることが多い．一般に，XML データの関係表への写像はノードを基本単位として行なわれるため，関係表に写像されるタプル数もこれに伴い増加することとなる．これに対し，構造結合等に代表される XML 問合せ処理手法 [文献] は，処理に要する計算量が大きいため，結果としてデータサイズが処理性能に深刻な影響を及ぼすことになる．

この問題にはアクセスの偏り（局所性）を利用することが一つの手がかりになる．一般に，ある問合せ（群）を処理するには，関連する部分データだけがあれば十分であり，必ずしも XML データ全体が必要なわけではない．この事実に着目した研究はすでにいくつかなされている．Marian 等は，XQuery を対象に問合せを分析し，XML データから必要な部分だけを射影する手法を提案している[4]．また，中島等は DOM を対象として，処理に必要な部分だけを部分的に主記憶に保持する SPlitDOM を提案している[5]．

そこで本稿では，関係データベースを基盤とした大規模 XML データベース構築の一手法について述べる．XML データベースの実現方法は，対象とする XML データの性質やスキーマ情報の有無，関係スキーマの設計方法などに応じて多くの選択肢が考えられるが，ここではスキーマを持たない整形式のデータを対象とする．提案手法では，XML 部分データのマッピングを可能にするためにリージョンディレクトリを利用する．これは Strong DataGuide に基づいたデータ構造であり，任意の単純経路式から XML データの該当する部分（リージョン）を取得することができる．さらに，問合せエンジンと XML マッパが連携することにより，システムに発行された問合せ処理に応じて必要な部分だけを XML ファイルから関係表にマッピングすることを可能にする．

本稿の構成は以下の通りである．第 2 章はシステムの全体像を説明する．第 3 章ではリージョンディレクトリを用いた XML 部分データの関係表へのマッピング手法を説明する．第 4 章は関連研究を紹介する．第 5 章はまとめである．

2. システムの概要

図 1 にシステムの概要を示す．XML データは初期状態ではファイルシステム上に格納されている．XML マッパは，前処理として対象となる XML データを走査し，構造概要とファイル中における各ノードの位置を抽出してメタデータを構築する．抽出されたデータはメタデータとしてデータベースに格納される．

利用者（アプリケーション）が発行した問合せは，問合せエンジンによって処理される．問合せエンジンはまず問合せを解析し，それが処理に必要な XML 部分データが関係表に存在するかどうかを調べる．無い場合は，マップ処理を XML マッパに要求する．XML マッパは要求に従い[6,7] 等の手法に基づき部分データを関係表にマップする．マップ処理が終了すると，問合せエンジンは[6,7] に従って問合せ処理を行う．

3. XML 部分データの動的マッピング

3.1 リージョンディレクトリ

XML 部分データの関係表への動的なマッピン

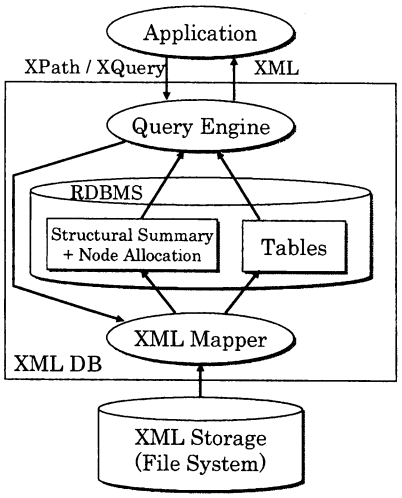

図 1　システムの概要

グを可能にするためには，システムが XML ノードのファイル中での位置を常に把握している必要がある．このために，本研究では，XML データの構造概要 (structural summary) を利用し，XML ノードと位置情報を関連付けたリージョンディレクトリなるデータ構造を提案する．これは，UNIX や Windows (DOS) のファイルシステムにおけるディレクトリ構造に類似しているが，XML は木構造を有しており，ノード間に厳密な包含関係が成り立っているところが異なる．

3.1.1 Strong DataGuide

XML を含む半構造データを対象とした構造概要は，これまでに多くの提案がなされているが[8)~10)]，ここでは最も単純な Strong DataGuide[8)] を用いることにする．なお，対象を木構造に限定した場合，1-index[9)]，D(1)-index[10)] と Strong DataGuide は等価であることが知られている．

情報源 s の DataGuide d は，1) s の全てのラベル経路が d において一つのデータ経路インスタンスを持ち，2) d の全てのラベル経路は s のラベル経路になっているものを言う．ここで，ノード o のラベル経路とは，o から辿ることのできるエッジ $(e_1, e_2, \ldots, e_n)$ のラベルの系列 $l_1, l_2, \ldots, l_n$ である．また，ノード o のデータ経路とは，o から辿ることのできるノードとラベルの系列 $l_1, o_1, l_2, o_2, \ldots, l_n, o_n$ である．d においては s 全ての経路が含まれているものの，s の複数のノードが d の一つのノードに集約されることから，d は s を要約していると言うことができる．

一般に，あるグラフから導出可能な DataGuide は複数存在するが，Strong DataGuide はその中の一つのクラスである．Strong DataGuide を定義するのに先立ち，ターゲット集合の定義を与える．情報源 s におけるラベル経路 l のターゲット集合を $T_s(l) = \{o \mid l_1, o_1, l_2, o_2, \ldots, l_n, o\}$，すなわちラベル経路 l を通って到達可能なノードの集合であるとする．また，情報源 s においてラベル経路 l と同じターゲット集合を持つ経路の集合を $L_s(l) = \{m \mid T_s(m) = T_s(l)\}$ で表す．同様に，DataGuide d でも同様に $L_d(l) = \{m \mid T_d(m) = T_d(l)\}$ である．このとき s の全てのラベル経路 l に関して $L_s(l) = L_d(l)$ が成り立つとき，d を s の Strong DataGuide であるという．すなわち，任意の経路式のターゲット集合という視点から，s と d が区別できないものを Strong DataGuide と呼ぶ．

対象とするデータが木構造の場合，Strong DataGuide の構築はグラフのサイズに対して線形の計算量で行うことができる．

3.1.2 リージョンディレクトリ

Strong DataGuide によって，ある経路式によって到達可能なノード集合をクラスタリングすることができる．本研究では，これを XML ノードのディレクトリとして利用する．具体的には，Strong DataGuide の各ノードから，そのノードのターゲット集合に含まれるノードのリージョンを索引付けする．

リージョン[6)] とは，そのノード (v) のデータ中での開始位置と終了位置の組 $(start(v), end(v))$ である．本論文では，これに根からの深さを加えた $(start(v), end(v), depth(v))$ をもちいる．これは，ノードのファイル中での位置を特定するだけではなく，2 ノード間の先祖，子孫関係を判定する目的にも使うことができる．すなわち，

- もしノード v が v' の先祖である場合，$(start(v) < start(v')) \wedge (end(v') < end(v))$ が成り立つ
- もしノード v が v' の親である場合，$(start(v) < start(v')) \wedge (end(v') < end(v)) \wedge (depth(v) = depth(v') - 1)$ が成り立つ
- もしノード v が v' の前にある場合，$end(v) < start(v')$ が成り立つ

これは良く知られた前置順，後置順を利用したラベリングや範囲ラベリングなどの手法[11)] と同様の性質である．

Strong DataGuide の各ノードからは，そのノー

ドのラベル経路式に対応するリージョン索引へのポインタを配置する．図2のXMLデータを使って説明する．図3は図2の木表現である．各ノードにはノードIDに加えてリージョンを併記している．図2のStrong DataGuideを計算し，リージョン索引を構築したものが図4である．例えば，元データにおいて経路式"/proc/paper/sect/title"にはノード&7と&8が対応する．このため，4では，"&7, &8"のノードからポインタを辿るとこれらのノードのリージョンを得ることができる．他のノードに関しても同様である．

```
<proc>
  <paper>
    <title>title</title>
    <abst>abstract</abst>
    <sect>
      <title>title1</title>
      content1
    </sect>
    <sect>
      <title>title2</title>
      content2
      <sect>
        <title>title2.1</title>
        content2.1
      </sect>
      <sect>
        <title>title2.2</title>
        content2.2
      </sect>
    </sect>
  </paper>
</proc>
```

図2 XMLデータの例

大規模なXMLデータを扱う場合，各ノードの索引部分，特に葉に近い部分には大量のリージョンが格納されるので，リンクリスト，ハッシュ，B+-tree等を利用して索引付けを行う．また，テキストノードはリージョンディレクトリには含まれず，必要に応じてリージョンの包含関係を利用して直接ファイルから読み取られることになる．

3.2 XML部分データの動的マッピング

3.2.1 単純経路質問の場合

利用者（アプリケーション）から発せられた問合せは，問合せエンジンによって処理される．ここでは問合せとしてXPath式を考える．問合せエンジンは，内部状態として既に関係表にマップされたXML部分データの一覧(M)を単純経路式として保持している．単純経路式とは，"/"で始まり，軸として"/"または"//"だけを含むXPath式のサブクラスである．初期状態では$M = \emptyset$である．

問合せエンジンは受け取った質問を構文解析し，単純経路式に分解する．例えば，

$$q = //title$$

なる問合せが発行されたとする．この例では質問自身が単純経路式となっているので，単純経路式への分解は行われない．次にqがMに含まれているかどうかを確認する．一般にXPath式の包含関係を判定するのは困難であるが，ここでは対象を単純経路式に限定($XP^{//}$)しているため，多項式時間で判定を行うことができる[12),13)]．この例では$M = \emptyset$であるので，問合せエンジンはただちにこの単純経路式に該当するリージョンをリージョンディレクトリ(RD)から取得する．

$$\begin{aligned} RD(//title) = \\ &\{(/proc/paper/title, \\ &\quad \{(21, 41, 2)\}), \\ &(/proc/paper/sect/title, \\ &\quad \{(85, 106, 3), (151, 172, 3)\}), \\ &(/proc/paper/sect/sect/title, \\ &\quad \{(209, 232, 4), (287, 310, 4)\})\} \end{aligned}$$

この結果は各ノードの根からの経路式とともに返却され，経路式の値ごとにグルーピングされている．これはXMLマッパがXMLデータ関係表にマップする際，現在処理中の部分データの根からの絶対経路式を知るために必要である．

XMLマッパは該当する部分文書をXMLファイルから読み込み，関係表へとマップする．このとき，このリージョンに含まれる全ての子孫ノードに関しても同様に関係表へのマッピングを行う．この例の場合は，各title要素に含まれる全てのテキストノードもその対象となる．この処理によって，問合せqを処理するのに必要なXMLデータは全て関係表へマップされるので，XMLデータ全体が関係表の上に存在しなくてもqに関しては処理を行うことができる☆．

XMLマッパは，内部状態としてマップ済みの領域を表1の形で保持している．各列は左から順にマップされた時刻を示すタイムスタンプ，単純経路式，マッピングの種類（後述），マップされたノード数，マップされた領域の長さの合計，

☆ XMLデータの関係表へのマッピング手法そのものについては，この論文の議論の範囲を超えるのでここでは述べない．

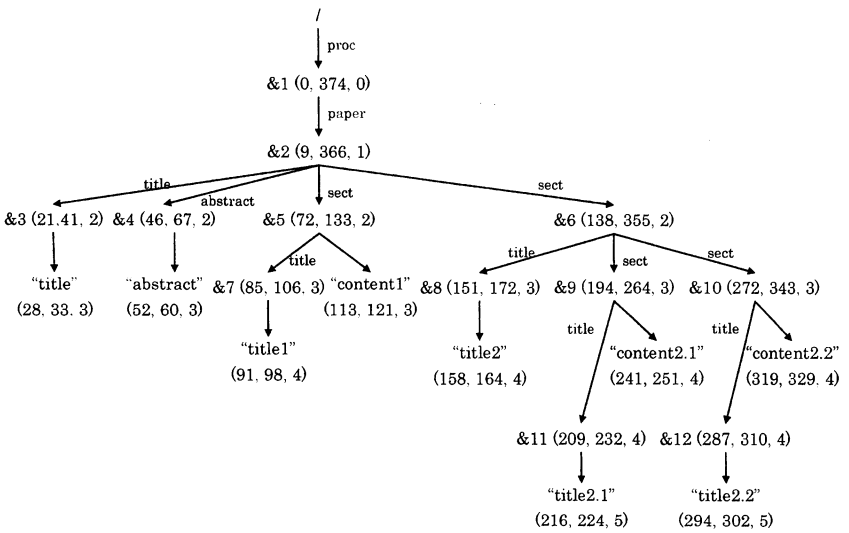

図 3　XML データの木表現

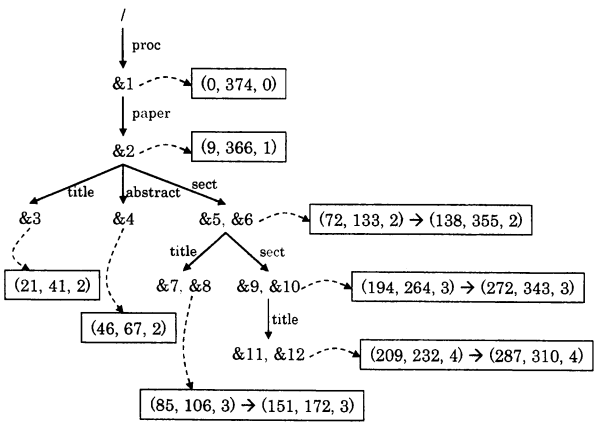

図 4　リージョンディレクトリ

表 1　マップ状態表 (I)

No.	path exp.	mode	# of nodes	total length	freq	last referenced time
1	//title	deep	5	113	1	2004/07/01 23:34:45

表 2　マップ状態表 (II)

No.	path exp.	mode	# of nodes	total length	freq	last referenced time
1	//title	deep	5	113	3	2004/07/01 23:40:00
2	//paper	shallow	1	855	1	2004/07/01 23:40:01

参照頻度，最後に参照された時刻である．

3.2.2　単純経路質問でない場合

問合せが述語を含むなど単純経路質問でない場合は，より複雑な処理が必要である．例えば，

q' = //*paper*[*title* = "*title*"]//sect/title

なる質問を考える．問合せエンジンはこれを構文解析して，図 5 に示す問合せ木[6] を得る．ここで，*title* ノードは最終的に解として出力されるノードなので，出力ノードと呼ばれる．この問合せ木を処理するのに必要なのは以下のノード

である．

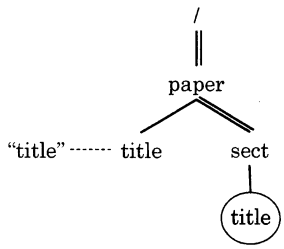

図 5 q' の問合せ木

- 出力ノード: //paper//sect/title
  この例では葉になっているが，一般に出力ノードは葉にあるとは限らない．
- 出力ノード以外の全ての葉ノード: //paper/title
- 分岐ノード: //paper
  出次数が 2 以上の中間ノード．すなわち分岐点になっているノードである．

これらは以下の二つの場合に分けて考える．

3.2.2.1 出力ノードおよび葉ノードのマッピング

出力ノードおよび葉ノードに関しては，述語の判定や結果の出力などにそのノードと子孫ノードの内容の両方を使う可能性が高いことから，第 3.2.1 節で述べたものと同様，リージョンに含まれる全てのノードを関係表へのマッピングを行う．リージョン全域を再帰的にマップするのでこれを「深いマッピング (deep mapping)」と呼ぶ．

この例では，//paper//sect/title と //paper/title が深いマッピングの対象となる．問合せエンジンはそれぞれの単純経路式のマッピングの要求を XML マッパに対して行う．XML マッパはマップ済みの領域とこれらを比較し，もしマップされていない領域であればマッピングの処理を行う．この場合，両方とも //title に含まれているので，実際のマッピング処理は行われず，参照頻度と最終参照時刻の更新だけが行われる．

3.2.2.2 分岐ノードのマッピング

分岐点になっている中間ノードに関しては，問合せ処理にその内容は使われず，出力ノードと葉ノードが共通の祖先を持っているかどうか，還元すれば実際のデータ上で両者に関連があるかどうかをチェックするためだけに必要となる．このため，全域を関係表にマップするのは必ずしも効率的であるとは言えない．この例の場合，paper 要素が分岐点になっているが，paper 要素をマップすることはすなわちデータのほぼ全域を関係表にマップすることになってしまい，データの局所性を利用することにつながらないからである．

このため，分岐ノードに関してはそのノードだけを関係表にマップする．これを「浅いマッピング (shallow mapping)」と呼ぶ．この処理が終わった時点で，XML マッパの内部状態は表 2 のようになる．

3.3 部分データのアンマップ

アクセスの局所性は時間に関しても考慮する必要がある．すなわち，関係表にマップされた部分データの内，アクセス頻度の少ないものは積極的に関係表から落としてやることによって，システム全体のパフォーマンスの向上が期待できる．

このために XML マッパの内部状態表の参照頻度，参照時刻等を利用する．具体的には，

- 参照頻度がある閾値よりも低い部分データに関しては関係表から別の一時表へ待避させる
- 一定時間アクセスのなかった部分データに関しては関係表から別の一時表へと待避させる

などの方針が考えられる．これはシステムに入力される問合せの傾向などから総合的に判断することになるが，細かいパラメータの設定や最適化の方法などは今後の課題である．

4. 関連研究

アクセスの偏り（局所性）を利用した XML データ処理の効率化に関しては，いくつかの先行研究がある．

Marian 等は，XQuery を対象として，問合せ処理に必要な部分のみを XML データから射影する手法を提案している[4]．このために，対象を XQuery のサブクラスである XQuery Core に限定し，任意の問合せから必要な部分データ（射影経路）を抽出するアルゴリズムを提案している．

中島等は DOM を対象として，処理に必要な部分だけを部分的に主記憶に保持する機構を追加した SPlitDOM を提案している[5]．この手法では，事前に必要なタグ名を設定ファイル中に記述しておき，その情報を元に必要な部分データだけを DOM パーサに返却する機構を設けている．これにより，JAXP (Java API for XML Processing) 準拠を維持したまま大容量 XML の効率的な処理を実現している．

これらは，データアクセスの局所性に注目している点においては本研究と同様であるが，XQueryプロセッサあるいはDOMプロセッサを対象としている点が本研究とは異なる．

また，ディスク上のXMLデータと関係表との動的なマッピング機構に関する研究として[14]がある．ただしこれはXMLデータをマッピングの単位として用いている他，DTDから得られるスキーマ情報を利用している点で本研究とは異なる．

5. まとめ

本稿では，関係データベースを基盤とした大規模XMLデータベース構築の一手法として，データアクセスの局所性に着目したXML部分データのマッピング手法を提案した．これを可能にするためにStrong DataGuideに基づいたリージョンディレクトリを導入した．これにより任意の単純経路式からXMLデータの該当する部分（リージョン）を取得することができる．さらに，問合せエンジンとXMLマッパが連携することにより，システムに発行された問合せ処理に応じて必要な部分だけをXMLファイルから関係表にマッピングすることが可能となった．

今後はシステムの実装と性能評価，更新のサポート，より複雑なXPath問合せやXQueryへの対応などに関して研究を進める予定である

謝辞 本研究の一部は，文部科学省科学研究費補助金（課題番号15017243），日本学術振興会科学研究費補助金（課題番号15700097）の支援によるものである．ここに記して謝意を表す．

参考文献

1) World Wide Web Consortium: Extensible Markup Language (XML) 1.0 (Third Edition), http://www.w3.org/TR/REC-xml. W3C Recommendation 04 February 2004.
2) : The Open Directory Project (ODP), http://dmoz.org/.
3) Gene Ontology Consortium: Gene Ontology (GO), http://www.geneontology.org/.
4) Marian, A. and Siméon, J.: Projecting XML Documents, *Proc. VLDB 2003*, pp. 213–224 (2003).
5) 中島哲, 小田切淳一, 井谷宣子, 吉田茂: XML高速処理技術SPlitDOMの機能拡張とWebアプリケーションへの適用評価, 第15回データ工学ワークショップ (DEWS2004) (2004).
6) 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 (TOIT)*, Vol. 1, No. 1, pp. 110–141 (2001).
7) Florescu, D. and Kossmann, D.: Storing and Querying XML Data using an RDMBS, *IEEE Data Engineering Bulletin*, Vol. 22, No. 3, pp. 27–34 (1999).
8) Goldman, R. and Widom, J.: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, *Proc. VLDB 1997*, pp. 436–445 (1997).
9) Milo, T. and Suciu, D.: Index Structures for Path Expressions, *Proc. ICDT 1999*, pp. 277–295 (1999).
10) Chen, Q., Lim, A. and Ong, K.W.: D(k)-index: an adaptive structural summary for graph-structured data, *Proc. SIGMOD 2003*, pp. 134–144 (2003).
11) Cohen, E., Kaplan, H. and Milo, T.: Labeling Dynamic XML Trees, *Proc. PODS 2002*, pp. 271–281 (2002).
12) Amer-Yahia, S., Cho, S., Lakshmanan, L. V.S. and Srivastava, D.: Minimization of Tree Pattern Queries, *Proc. SIGMOD 2001* (2001).
13) Miklau, G. and Suciu, D.: Containment and Equivalence for an XPath Fragment, *Proc. PODS 2002*, pp. 65–76 (2002).
14) Zhang, X., Mitchell, G., Lee, W.-C. and Rundensteiner, E. A.: Clock: Synchronizing Internal Relational Storage with External XML Documents, *Proc. RIDE-DM 2001*, pp. 111–118 (2001).