2004－DBS－134　(II)　(39)
2004／7／14

リレーショナルデータベースを用いたXQuery処理システムの実現

春日 史朗† 坂田 哲夫† 小谷 尚也† 芳西 崇†

†日本電信電話株式会社 NTTサイバースペース研究所
〒239-0847 神奈川県横須賀市光の丘1-1

E-mail: †{kasuga.shiro, sakata.tetsuo, kotani.naoya, honishi.takashi}@lab.ntt.co.jp

あらまし 近年，情報流通のフォーマットとしてXMLが注目を集めている．XMLは優れたデータ表現と拡張性を備えており，さまざまな情報処理分野への応用が期待されている．XMLが様々なシステムに利用されるに従い，XMLをRDBに格納し，検索する必要性が高くなっている．一方で，XMLスキーマ定義により，強く型付けされた情報をXMLで表現することが可能となり，XMLの適用範囲は企業の基幹系システム等へ広がりつつある．従来方式のRDBへのXML格納では，スキーマレスなXML文書の格納に重点が置かれおり，物理記憶領域を多く利用し，様々なXML文書の検索要求に応えようとするものが多かった．しかし，強く型付けされたXMLデータを利用する大規模なシステムにおいては，物理格納領域の低減による格納効率の向上や，結合処理の低減による検索効率の向上が重要である．本稿ではこれらの問題に対し，RDBの処理に適したXMLの格納・検索方式を提案し，さらに，その格納・検索方式に基づいたXQueryの処理方式を提案する．また，提案方式がRDBにおけるXMLデータの格納・検索を効率的に実現可能なことを示す．

キーワード XML，RDB，モデル変換，XQuery

An Implementation of XQuery Processing System using Relational Database

Shiro KASUGA† Tetsuo SAKATA† Naoya KOTANI† and Takashi HONISHI†

†NTT Cyber Space Laboratories, NTT Corporation
1-1 Hikari-no-oka, Yokosuka-shi, Kanagawa, 239-0847

E-mail: †{kasuga.shiro, sakata.tetsuo, kotani.naoya, honishi.takashi}@lab.ntt.co.jp

Abstract Recently, XML has emerged as the standard for the exchange of data. XML has a fine data expression and extention and it is expected to apply for a variety of information processing fields. As XML attracts attention, it has been necessary to store and search for XML to RDB. On the other hand, it became possible for XML to express the well-typed information using XML schema, and it can be widely applied to the fundamental system of the company. We had a problem about efficiency of storing and searching for XML data, because former methods of storing XML to RDB forcus on storing schemaless XML document, not XML data described by XML schema. This paper proposes the method of storing XML in the conventional RDB efficiently, to these problems, and we propose the processing system of XQuery based on this storing method further. Moreover, this paper shows that a proposal system can store and search XML data efficiently to RDB.

Keyword XML, RDB, Model Conversion, XQuery

1. はじめに

近年，情報流通のフォーマットとしてXML（eXtensible Markup Language）[1]が注目を集めている．XMLは優れたデータ表現と拡張性を備えており，さまざまな情報処理分野への応用が期待されている．

XMLが様々なシステムに利用されるに従い，XMLをデータベースに格納し，検索する必要性が高くなっている．XMLの格納・検索には，当初，オブジェクト指向データベース（OODB）を用いることが多かったが，最近では，データの信頼性や，トランザクション処理の性能などで技術的に確立しているRDBに格納する方式も多数提案されている[2][3]．一方で，XMLSchema[4]やRelaxNG[5]などの「XMLスキーマ定義言語」が提案され，強く型付けされた情報をXMLにより表現することが可能となり，従来の半構造データだけでなく，型や構造が厳

密に定義されたデータを扱うシステム（企業の基幹系システムなど）にも適用範囲が広がっている．以降では，スキーマレスで型付けの弱いXML を「XML 文書」，強く型付けされた XML を「XML データ」，両方を表す場合は単に「XML」と呼ぶこととする．RDB に XML を格納する多くの方式では，スキーマレスな XML 文書の格納に重点が置かれおり，RDB 上の物理記憶領域を多く利用し，様々な XML 文書の検索要求に応えようとするものが多かった．しかし，強く型付けされた XML データを利用するシステムにおいては，検索の自由度よりも，物理格納領域の低減による格納効率の向上や，結合処理の低減による検索効率の向上が重要である．

また，オブジェクトリレーショナルデータベース（ORDB）で行われている XML 型を用いた格納・検索方式では，XML 文書を文書単位で入出力することに主眼が置かれており，部分木単位の入出力，更新，削除操作，複数 XML 文書の結合が制限され，XML 利用アプリケーションの開発の妨げになっていた．

本稿ではこれらの問題に対し，RDB に適した XML データの格納・検索方式を提案し，さらに，その格納・検索方式に基づいた XQuery[6]の処理方式を提案する．また，提案方式が RDB における XML データの格納・検索を効率的に実現可能なことを示す．

2. 従来技術と問題点

XML スキーマ定義により強く型付けされた XML データを，RDB に格納し検索を行う場合，従来から RDB 上で管理されてきたデータを扱うことになるため，RDB 上への格納は RDB 本来の格納・検索機能を利用して実現することが望まれる．

従来より，XML を RDB に格納する方式として様々な方式が提案されているが，大きく分けると，以下の 3 つがある．

方式(1) XML を長大テキストとして単一カラムに格納し，テキスト検索インデックスを作成する

方式(2) XML をノードとリンクに分解して RDB 上の固定のテーブルに格納する[3]

方式(3) XML スキーマ定義に従い，XML の要素を RDB のテーブルとカラムにマッピングして格納する

方式(1)は SQL99[7]で規格化され，商用 RDB の XML 型などで実現されている．この方式では，

- スキーマが定義されない XML 文書を格納できる

という利点があるが，

- XML の検索で RDB の検索機能を使えないため，独自のクエリエンジンが必要となる
- 一つ一つの XML を独立して処理するため，複数の XML をまたがる XQuery 処理が困難である

という問題があった．

また，方式(2)では，

- スキーマが定義されない XML 文書を格納できる
- XML の木構造を辿る操作が容易に実現できるため，XQuery を用いて複数の XML を合成する処理が行える

という利点があるが，

- XML スキーマ定義による型定義が RDB の型管理機能に反映されないため，格納効率が悪化する

という問題があった．

方式(3)は，XML スキーマ定義で型付けされた XML 中の要素を，RDB 上のカラムに最適な格納形式で格納することができ，格納・検索効率の向上に有利である．本稿の格納・検索方式も，この方式(3)に分類される方式である．この方式のカギとなるのは，元の XML データの構造を表す情報（構造情報）を，RDB 上で表現する方式である．例として，XML データを，図 1 に示すテーブル群にマッピングしたとすると，テーブルを結合するためのキーとなる構造情報の格納方式には下記が存在する．

方式(3-1) 親レコードへの参照を使う方式

方式(3-2) 先行順 ID，後行順 ID を使う方式[2][8]

方式(3-3) Dewey の 10 進分類法を使う方式[9]

最も単純な方式(3-1)では，構造情報として，全てのレコードに個々のレコードを識別する識別子（ID）と，親レコードを識別する親 ID を設定する．この方式では，例として，テーブル C のカラムと，テーブル D のカラムを条件として，テーブル A のレコードを特定し，そのレコードに属する，テーブル E の指定のカラムを返却する検索は，全ての検索経路中のテーブル（A，B，C，D，E）を結合することで実現する．

方式(3-2)では，構造情報として，2 つ整数値（先行順 ID，後行順 ID）に用いる．この方式では,格納時には,木構造中のノードに対して，先行順巡回で付与した番号を先行順 ID カラムに格納し，後行順巡回で付与した番号を後行順 ID カラムに格納し，これら 2 つの整数値のみで構造情報を表現する．検索時には，階層構造中の祖先，子孫のテーブル間の結合を，途中のテーブルを飛ばして行うことが可能となるため，結合するテーブル数を低減することができる（テーブル C を飛ばして，A と C，A と D の結合が行えるため，結合するテーブル数は A，C，D，E の 4 つに低減される）．

方式(3-3)では，識別子として，Dewey の 10 進分類法を用いる．この方式では，各レコードの識別子として，親レコードの識別子に，自レコードの兄弟間における順番を表す数字を，ピリオド区切りで付加した可変長文字列（”1”，”1.1”，”1.1.1”など）を用いる．この方式では，識別子文字列のピリオドで区切られた数字に対し，頭から指定個数の数字を比較することで，検索時に結合するテーブル数をさらに低減させることができる（C，D，E の 3 つに低減される）．

ここで，方式(3-1)では，

- 格納の際に，構造情報を格納する物理記憶領域が少なくて済む．（ID と親 ID の 2 つの整数値）

という利点があるが，

- 検索の際に，検索経路中の全てのテーブルを結合する必要があり，検索処理コストが高い．

という問題があった．また，方式(3-2)では，同様に，

- 格納の際に，構造情報を格納する物理記憶領域が少なくて済む．（先行順 ID と後行順 ID の 2 つの整数値）

という利点があるが，

- 検索の際に，先祖子孫関係にあるテーブルを結合で途中のテーブルの結合をスキップすることができるが，共通の先祖テーブルを持つ子孫テーブル同士を結合するには，共通の先祖テーブルを挟んで結合する必要があり，子孫テーブル同士を直接結合することはできない．

という問題があった．また，方式(3-3)では，

- 格納の際に，構造情報を格納のため，個々のテーブルに対し可変長文字列による識別子が必要であり，多くの物理記憶領域が必要となる．

という問題があった．

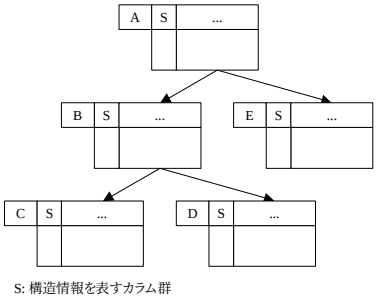

図 1 テーブルの階層構造

3. 提案方式

前述の，既存方式における問題に対し，本稿では，スキーマに強く型付けされた XML データを対象とし，RDB 上での格納効率と検索効率の向上を実現する，XML データ格納・検索方式を提案する．

提案する方式は，従来方式で述べた方式(3)に分類される方式であり，XML データと RDB 間で図 2に示す 3 つのモデル変換方式（スキーマ変換，データ変換，検索式変換）により実現されている．

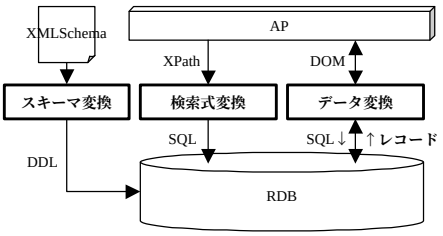

図 2 提案方式の構成

3.1. スキーマ変換方式

スキーマ変換方式は，XML スキーマ定義と，RDB スキーマ定義の対応付けを行う方式である．本方式では，XML スキーマ定義における複数の型定義を，RDB 上の個別のテーブルに対応付ける．XML スキーマ定義は，XMLSchema で定義する．ここで，XMLSchema は，要素型定

義を基本単位とし，複数の型定義により構成される．各型定義は，任意個の「属性」とひとつのテキストを持つことができ，各属性，テキストは単純型（数値，文字列など）のデータを格納する．文書要素（XML データの最上位要素）の指定を基点として，型定義と型参照を辿ることで，XMLSchema の定義から型定義の木構造が表現できる．図 3の XMLSchema の例より，文書要素が要素 A と指定される場合の型定義の木構造は図 4となる．

```
<schema>
  <element name='A' type='AType'/>
  <element name='B' type='BType'/>
  <element name='C' type='string'/>
  <element name='D' type='string'/>
  <element name='E' type='EType'/>
  <element name='F' type='integer'/>
  <complexType name='A'>
    <sequence>
      <element ref='B' maxOccurs='10'/>
      <element ref='E' maxOccurs='10'/>
    </sequence>
  </complexType>
  <complexType name='BType'>
    <sequence>
      <element ref='C' maxOccurs='10'/>
      <element ref='D' maxOccurs='10'/>
    </sequence>
  </complexType>
  <complexType name='EType'>
    <sequence>
      <element ref='F'/>
    </sequence>
  </complexType>
</schema>
```

図 3 XML スキーマ定義の例

ここで，親子関係にある型定義 X，Y に対して，親子間の「多重度」を定義する．多重度とは，親型定義 X のノード Xi に対し，包含される子型定義 Y のノード Y1，Y2，…，Yp が取りうる個数のことである．その最小の個数を m，最大の個数を n とすると，多重度は m..n と表される．（XMLSchema では，最少個数 m を minOccurs 属性，最大個数 n を maxOccurs 属性により指定する）

次に，XML データのスキーマ定義から，RDB 上のスキーマを生成する．この処理は，XMLSchema の各型定義をデータベーススキーマのテーブルとし，各型定義の属性を各テーブルのカラムに対応することで自動的に行うことが可能である．なお，多重度が 0..1 の子要素は，親要素に対応するテーブルに含めることも可能である．また，XMLSchema の型情報が定義されていれば，適切な型のデータベース要素型を選択することが可能である．例として図 4の XMLSchema から生成される RDB のスキーマ定義を図 5に示す．

なお，各テーブルには，XML データの構造情報を格納するカラム S（構造 ID 列）を追加しておく．

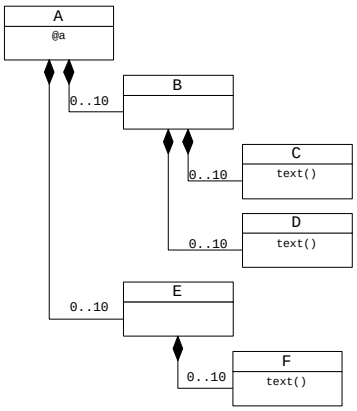

図 4スキーマ木構造の例

```
CREATE TABLE "tabA" ("S" integer)
CREATE TABLE "tabB" ("S" integer)
CREATE TABLE "tabC" ("S" integer, "text" text)
CREATE TABLE "tabD" ("S" integer, "text" text)
CREATE TABLE "tabE" ("S" integer)
CREATE TABLE "tabF" ("S" integer, "text" integer)
```

図 5 RDB のスキーマ定義の例

3.2. データ変換方式

データ変換方式は，XML データの木構造と RDB に格納する複数のレコードを相互変換する方式である．本方式では，XML データの木構造を複数テーブル上の複数のレコードに分解して格納する．この際，元の XML 木構造を表す構造 ID を各レコードに付加する．

3.2.1. 構造 ID 付与方式

構造 ID を定義する前に，準備としてテーブル間の倍率を定義する．

先に 3.1 スキーマ変換方式で述べた型定義の親子関係から，対応するテーブル間にも親子関係が定義され，親子間で定義される多重度に相当する数値（倍率）も同様にテーブル間で定義される．倍率は，ある親テーブル内のレコード１つに対して，子テーブル内のレコードがいくつ所属できるかを表している．ここで親子関係を有するテーブル T_X,T_Y 間で倍率を次のように定義する．T_X，T_Y 間の倍率 $M(T_x,T_Y)$は，対応す

る型定義間の多重度を m..n とするとき，$M(T_x,T_Y)=n$ である．ここで，任意のテーブル T_1 と，その子孫テーブル T_N の間で倍率が定義でき，T_1 から T_N を辿る経路上のテーブルを T_1，T_2，‥，T_{N-1}，T_N とすると，

$$M(M_1,M_N)=M(T_1,T_2)\times M(T_2,T_3)\times\cdots\times M(T_{N-2},T_{N-1})\times M(T_{N-1},T_N)$$

と定義される．なお，$T_2\cdots T_{N-1}$ は，T_1 と T_N を結ぶ最短経路上のテーブルとする．

以上を踏まえ，ある親テーブル T_P と子テーブル T_C について，T_C に挿入するレコード R_C の構造 ID「S_C」を，

$$S_C=S_P\times M(T_P,T_C)+L(R_C)$$

と設定する．なお，親レコード R_P の構造 ID「S_P」，子レコード番号「$L(R_C)$」とすると，子レコード番号は，同じ親レコードを持つ同じテーブルの子レコード（兄弟レコード）の間で一意の番号であり，あるレコード R_j の $L(R_j)$は，R_j のテーブル T_C とその親テーブル T_P の倍率 $M(T_P,T_C)$より，$0\leqq L(R_j)<M(T_P,T_C)$となる任意の整数値（ただし他の兄弟レコードの番号と重複しない）である．また，最上位のテーブル T_T のノードをレコードに変換する際の構造 ID「S_T」は，テーブル T_T に対応するテーブルに既に存在するレコードが未使用の任意の値を用いる．この構造 ID は，任意のテーブル T_a と，その子孫テーブル T_d とすると，

$$S_a=\mathrm{floor}(S_d/M(T_a,T_d))$$

の関係にあり，子孫テーブルの構造 ID（S_a）から，任意の先祖テーブルの構造 ID（S_d）が算出できることを示している．

アプリケーションから渡される入力 XML データを個別のレコードに変換する際，生成された各レコードに，構造 ID 列 S に対する XML データの構造情報を表す「構造 ID」を付与する．

また，レコードに分解して格納したレコードを XML に復元する場合は，テーブル群に対して構造 ID の範囲検索を行い，得られたレコードを木構造に再構成することで容易に実現できる．

3.3. 検索式変換方式

検索式変換方式は，XML 検索式（XPath により記述）を，RDB 検索式である SQL に変換する方式である．XPath 検索式は，結果を指し示すパスと，複数の述語より構成される．例として，図 4の構造に対する検索式として，

```
/A[B/C='x'][B/D='y']/E/F
```

を想定する．ここで本方式において，検索式中のパス（/A/E/F，/A/B/C，/A/B/D）の示す先は，必ず対応関係にあるテーブル中のカラムを指すため，各パスは単純な select 文に変換することができる．

select 文(1) …

```
    select * from "TabF"
```

select 文(2) …

```
    select * from "TabC"
where "text" = 'x'
```

select 文(3) …

```
    select * from "TabD"
where "text" = 'y'
```

ここで，select 文(1)の返すレコードの構造 ID と，select 文(2)の返すレコードの構造 ID は，

```
floor("TabF".S/M("TabA","TabF"))
    =
floor("TabC".S/M("TabA","TabC"))
```

の関係がある．これを述語（[B/C='x']，[B/D='y']）からパス（/A/B/F）への絞り込み条件として用いることで，図 6に示す SQL へ変換することができる．このクエリは，テーブル結合が少ない回数で済むため，高い検索効率が実現できる．（"TagF"，"TagC"，"TagD"の 3 つのテーブルの結合）

```
select "text"
from "TabF"
where floor("TabF".S/M("TabA","TabF")) in (
    select
floor("TabC".S/M("TabA","TabC"))
    from "TabC"
    where "text"='x'
)
and floor("TabF".S/M("TabA","TabF")) in (
    select
floor("TabD".S/M("TabA","TabD"))
    from "TabD"
    where "text"='y'
)
```

図 6 変換後の RDB 検索式

3.4. XQuery 処理システム

提案する XPath 処理方式を用いて，XQuery 処理システムを構築することができる．XQuery 処理システムの構成を図 7に示す．本システムは次の構成要素より成る．

XQuery プロセッサ・・・AP から入力された XQuery を，FLWOR 句，XPath，コンストラクタに分割し，FLWOR 句とコンストラクタを処理する機能である．

XPath プロセッサ・・・本稿の提案する XPath 処理方式を実装し，XMLSchema から RDB スキーマの生成，XPath の SQL 変換と実行，XML のレコード化，レコードの XML 化を行う機能である．

XML データ格納用テーブル群・・・XML データを格納するテーブル群である．

モデル変換情報リポジトリ・・・XML-RDB モデル変換情報を格納する RDB 上のリポジトリである．

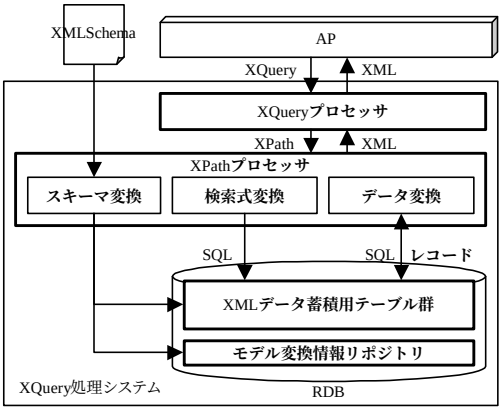

図 7 XQuery 処理システム構成

3.4.1. XQuery の処理方式

XML を 1 カラムに格納する方式では，一つの XML 単位で操作することは可能であるが，複数の XML を結合することが困難であった．本システムでは，XPath 式は単純な一つの SQL に変換して処理されるため，複数パスの結合を容易に SQL 化することができる．例として図 8の XQuery は，2 つの XPath の示す先を結合する検索例であるが，図 9に示す SQL として RDB の結合機能を用いて実行が可能である．

```
for $a in xxxx
for $b in yyyy
where $a/@a = $b/@b
return <result>
    <A>{$a/text()}</A>
    <B>{$b/text()}</B>
</result>
```

図 8 XQuery による結合の例

```
SELECT t1.text, t2.text
FROM TabA t1, TabB t2
WHERE t1.AtrA = t2.AtrB
```

図 9 SQL への変換例

4. 評価

4.1. 評価方法

本稿では，下記の 3 つの構造情報の表現方式について，RDB の処理コストを比較する．

(type1) 提案方式
(type2) 先行順 ID，後行順 ID に用いた方式
(type3) Dewey の 10 進分類法を用いた方式

評価は，図 4の構造に対する検索式として，

```
/A[B/C='x'][B/D='y']/E/F
```

に相当する検索式を，それぞれに対して実行し，検索処理時間を求める方法で行った．

なお，評価環境を図 10に，評価モデルを図 11に示す．

OS	Solaris8
CPU	UltraSPARC 1.0GHz×2
メモリ	4GB
HDD	20GB
RDB	PostgreSQL7.4.1

図 10 評価環境

評価データ	XMark[10]の xmlgen により生成した people 情報
文書数	200，2000，20000
サイズ/文書	平均 6KB

図 11 評価モデル

4.2. 評価結果

評価結果を，方式別の検索時間の計測結果を図 12に示す．また，文書数 2000 の場合の方式別データベースサイズを図 13に示す．

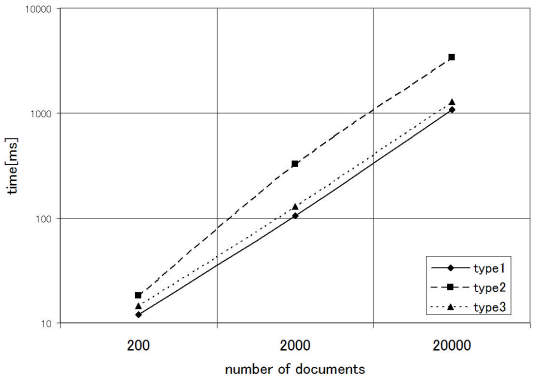

図 12 方式別の検索時間の比較

方式	DB ファイルサイズ(文書数 2000)
type1	17.23MB
type2	21.41MB
type3	23.82MB

図 13 方式別のデータベースサイズの比較

5. 考察

図 12より，提案方式である type1 は，テーブルの結合回数が少ないため，type2 と比較して，処理速度が高速であることが分かる．図 13より，type1 は，構造情報に必要な物理記憶領域が小さいため，type2 と比較して約 20%程度，type3 と比較して約 30%少ないディスク使用量となっている．これらにより，提案方式は，type2 と type3 に比較して，高い格納効率と検索効率を実現した方式であると言える．なお，検索処理に際しては，テーブルの結合に Hash join が使われていた．そのため，Hash 生成コストが，文書数に比例して増加しているが，データによって，Sort-Merge join に切り換えることで，処理時間を低減することは可能であると考えられる．

6. まとめと今後の課題

以上より，提案する方式では，検索する際に，従来必要であった，テーブル結合の回数を削減し，結合のための演算処理も整数演算で行えるため，検索が高速に処理可能となる．また，従来方式に比べて，XML データを RDB に格納する際に，構造情報を単一整数値のみで表現でき必要な格納領域を削減することができる．本稿では，これらにより，XML データを RDB へ格納し検索するシステムの検索パフォーマンスを向上できることを確認した．また，複数パスを同時に処理する SQL に変換可能であるため，XQuery による XML データの結合に対応することができることを示した．今後の課題としては，XML 処理に適した RDB における実行プランの最適化などがあげられる．

文　献

[1] W3C, “Extensible Markup Language (XML) 1.0”, http://www.w3.org/TR/REC-XML, 2000.

[2] T. Grust, M. van Keulen, J. Teubner, “Accelerating XPath Evaluation in Any RDBMS”, ACM Transactions on Database Systems, Vol. 29, No. 1, March 2004, Pages 91–131.

[3] 吉川, 志村, 植村, “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”, 情報処理学会論文誌: データベース, Vol.40, No.SIG6(TOD3), pp115-131, 1999.

[4] W3C, “XML Schema Part 0: Primer”, http://www.w3.org/TR/xmlschema-0/, 2001.

[5] OASIS, “RELAX NG Specification”, http://www.relaxng.org/spec-20011203.html, 2001.

[6] W3C, “XQuery 1.0: An XML Query Language”, http://www.w3.org/TR/xquery/, 2003.

[7] Jim Melton, “(ISO-ANSI Working Draft) XML-Related Specifications (SQL/XML)”, WG3:DRS-020 = H2-2002-365, WD 9075-14 (SQL/XML), August, 2002.

[8] Paul. F. Dietz, “Maintaining order in a linked list”, ACM Symposium on Theory of Computing, pages 122-127, 1982.

[9] Donald E.Knuth, “The Art of Computer Programming”, Addison-Wesley Publishing Company,1973.

[10] A. Schmidt, F. Waas, M. Kersten, M.J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In Proc. of the 28th Int'l Conference on Very Large Databases (VLDB), pages 974-985, Hong Kong, China, August 2002.