データベースシステム　119－5
（１９９９．７．２１）

XML問い合わせ処理システム(xQues)の問い合わせ処理系

久保田　和己[1]　　金政　泰彦[1]　　石川　博[1]

XMLデータはWeb情報システムやEC/EDI応用で広く使われていくことが予想されるが、そのような応用では大量のXMLデータを通常対象とする。そのためには利用者がサーチ条件を指定できるようにして本当に必要なXMLデータのみを検索できるようにすることと、複数のXMLデータソースを統合利用できるようにすることが必要である。このためにXQLというXMLデータの問い合わせ言語を提案する。XQLは従来のデータベース標準（SQLやOQL）との整合性を考慮して設計されている。本稿ではXMLデータの問い合わせ処理系の要件と機能について説明し、XML問い合わせ処理システムxQuesのための問い合わせ処理系の実装について触れる。

A Query Processing Engine of an XML Query Processing System *xQues*

Kazumi Kubota[1], Yasuhiko Kanemasa[1] , and Hiroshi Ishikawa[1]

XML data are expected to be widely used in Web information systems and EC/EDI applications. Such applications usually use a large number of XML data. First, we must allow users to retrieve only necessary portions of XML data by specifying search conditions to flexibly describe such applications. Second, we must allow users to combine XML data from different sources. To this end, we provide a query language for XML data tentatively called XQL. We have designed XQL, keeping in mind its continuity with database standards such as SQL and OQL although we don't stick to its strict conformity. In this paper, we describe the requirements for a query processing engine for XML data and explain the functionality. We make comments on an implementation of a query processing engine for an XML query processing system called xQues.

はじめに

XMLデータはWEB情報システムやEC/EDI応用で広く使われることが期待されている。そのような応用は通常大量のXMLデータを利用する。そこで、第一にユーザがそのような応用の柔軟な記述のために検索条件を指定してXMLデータの必要な部分だけを検索できるようにする必要がある。第二に異なるデータソースのXMLデータを組み合わせて、新たなXMLデータを生成できるようにする必要がある。

本稿では、XML問い合わせシステム(xQues)の構成について述べ、検索インデックスの利用、実行最適化を含めた検索処理の実現について述べる。

1. XMLデータの表現と問合わせ

1.1 ＸＭＬデータのモデル

我々はＸＭＬデータを木構造のモデルで表現して考えている。このモデルでは、ノードは要素の値を保持するために用いられ、ノード間のリンクは要素を囲っているタグの名前に相当する。いちばん下のリーフノードには要素の値が書かれている。ある文献に関するデータをこのモデルで表現した例を図１に示す。木の一番上にはノード番号０番のノードがあるが、そこから、木を左下にたどってゆくとbibとラベルされたリンクを通って１番のノードに至る。さらにたどるとbookとラベルされたリンクを経て２番のノードに、さらにはtitleとラベルされたリンクを経て６番のノードに至る。６番のノードには"A"という値が記されている。これは

```
<bib>
  <book>
    <title>A</title>
  </book>
</bib>
```

というXMLの構造に対応している。

[1] 富士通研究所
FUJITSU LABORATORIES LTD.

1.2 ＸＭＬに対する問い合わせ

ＸＭＬデータに対する検索要求のパターンを考えてみると

- 指定した要素に特定の値をもっている要素を見つけたい。

言い換えると

- あるパターンに適合する部分を取り出したい。

ということになる。

これはたとえば先ほど示したような木構造モデルで、条件を満たすノード（のＩＤ）を得る。そのノードから木の上を移動して別のノードに至る。到達したノード（要素）の値を得る。あるいは、パターンに適合した要素から下のサブ要素をまとめて得るという操作に相当する。さらにもっと複雑な場合として、上の例でいうと著者が同じであるような本のタイトルを組にして得たいという場合もある。

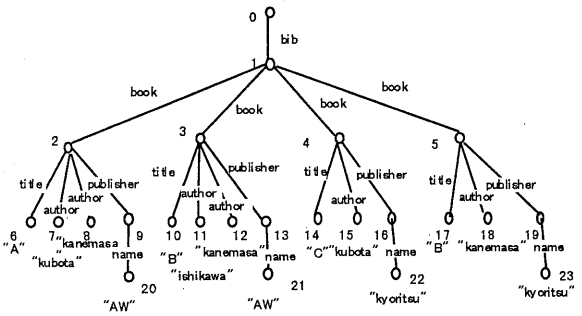

図１ XMLデータの木構造モデルの例

1.3 関連研究

同様の目的をもった研究のいくつかを示す。

Lore: A Database Management System (Stanford University) [Lore]は半構造データベース管理システムのための検索インデックス、問合せ言語、およびその実行エンジンと最適化についてのさまざまなアプローチを提供している。

XML Query Language XQL W3C QL Workshop proposal (Microsoft) [MS-XQL] はXSLを拡張してXMLのフィルタリングを行う形式の問合わせ言語である。

XML-QL: A Query Language for XML[XML-QL] は、SQL のような、SELECT WHERE の構造を持ち、さらに最近 データベース研究コミュニティによって提案された 半構造なデータのために問合せ言語の特徴を借用している。

2. 検索言語XQLとその実行

ここで我々が提案しているXML問合わせ言語 XQL[FJ-XQL]の例を示す。

```
SELECT
result:<$book.title, $book.author, $book.publisher.name>
FROM book:bib.book
WHERE $book.author.name = "kubota";
```

この問合わせの意味は

「bib.book.author が "kubota" であるような bib.book について、bib.book.title, bib.book.author, bib.book.publisher.name を検索結果として得たい」という意味である。

ここで、bib.book.publisher.name のように "." (ドット) で区切って表記したものをパス表記と呼ぶ。この意味は、先に述べた XMLデータを木構造で表現した場合の特定のエレメントを示すノードの、ルートノードからの絶対パスを指している。

2.1 問合わせ演算のパターン

XQLによる問合わせ演算は次に示す３つの演算の組み合わせとなる。

- 値によるノードの選択

```
SELECT
result:<$book.author, $book.publisher.name>
FROM book:bib.book
WHERE $book.title = "B";
```

この場合、$book.title が "B" であるような $book をすべての bib.book の中から選択して $book にバインドする。

- セルフジョイン

```
SELECT result:<$book1.title, $book1.author>
FROM book1:bib.book, book2:bib.book
WHERE $book1.title = $book2.title;
```

この場合、２つの範囲変数を用いて $book1.title と $book2.title が等しくなるような $book1と $book2 の組みを求める。

- プロジェクション

```
SELECT
result:<$book.author, $book.publisher.name>
FROM book:bib.book
WHERE $book.title = "B"
AND $book.author = "kanemasa";
```

プロジェクション演算は単独で用いられることはなく、値による選択やプロジェクション演算とともに SELECT の部分で出力結果形式の指定の際に用いられる。

この場合、WHEREの部分に記述された条件を満たすような bib.book が $book にバインドされているときに、$book バインドされた各要素に対して、$book.author, $book.publisher.nameに相当するサブ要素の値がこの演算の結果となる。ここで指定されたサブ要素以外のサブ要素をもつ bib.book があったとしてもそれは結果から取り除かれる。

2.2 問合わせ基本関数

xQues処理系は、XQLで記述されたユーザからの検索要求を以下に示す問合わせ基本関数の列に置き換えて処理する。その際に、どの順序で実行するのが効率的であるかを考慮した実行最適化やどのインデックスを用いて目的とする要素集合を得るかを判断して問合わせ実行計画を作成することになる。

問合わせ基本関数はつぎの６つの関数である。これらの関数は、先に述べたXMLデータを木構造で表現するモデルにおいて、木にそってノードをたどる動作を関数化したものである。その際にデータベースに格納されているノードにはすべてユニークな識別子（ＩＤ）がつけられている。また、要素のタグはラベルという形で格納されていて、ラベルにも識別子がつけられている。

(1) GetNodeIDbyPathAndVal
パス記述と値を指定して、それに該当するノードＩＤの集合を得る
(2) GetParentIDbyChild
子のノードＩＤを指定して、その親ノードＩＤの集合を得る（親ノードはこの場合１つである）
(3) GetChildIDbyParent
親のノードＩＤを指定して、その子ノードＩＤの集合を得る
(4) GetValuebyID
ノードＩＤを指定してその要素がもつ値を得る
(5) GetNodeIDbyPath
パス記述を指定して、それに該当するノードＩＤの集合を得る
(6) GetLabelIDbyLabeltext
ラベルの名前を指定してラベルＩＤを得る

2.3 実行計画の生成

実際の問合わせを例にしてセレクションとプロジェクション演算を含むようなXQL問合わせに対して作成される実行計画について考えてみる。
2.1で示した３つの演算パターンをそれぞれ

- VS　　値によるノードの選択
- SJ　　セルフジョイン
- PROJ　　プロジェクション

という省略形を用いて表記する。実行計画は演算パターンとそれに続く引数の列でからなる１つの処理単位を複数ならべたものである。引数の並びはVSとSJの場合、演算子、左辺値、右辺値であり、PROJの場合、範囲変数とパス名の並びである。
生成される実行手順を例にそって説明する。

(1) プロジェクション

```
SELECT
result:<$book.author, $book.publisher.name>
FROM book:bib.book
WHERE $book.title = "B";
```

この問合わせに対して作成される実行計画は

```
VS   OP_EQ $book.title  "B"
PROJ $book $book.author $book.publisher.name
```

である。
生成された実行計画はそのまま順に実行され、中間結果はスタックに詰まれる。
この実行計画を実行する際に呼び出される問合わせ基本関数の列は次のようになる。

1. GetNodeIDbyPathAndVal を用いて パス記述が bib.book.title で値が "B" であるノード集合を得る。
2. 1で得られたノード集合に対してそれぞれ GetParentIDbyChild を適用して bib.book となるノード集合を得る。これが $book にバインドされる。
3. 2 で得られたノード集合に対してそれぞれ GetChildIDbyParent を適用して $book.author となるノード集合を得る。
4. 2 で得られたノード集合に対してそれぞれ GetChildIDbyParent を適用して $book.publisherとなるノード集合を得る。
5. 4 で得られたノード集合に対してそれぞれ GetChildIDbyParent を適用して $book.publisher.nameとなるノード集合を得る。
6. 3 で得られたノード集合に対してそれぞれ GetValuebyID を適用して要素の値を得る。
7. 3 で得られたノード集合に対してそれぞれ GetValuebyID を適用して要素の値を得る。

(2) セルフジョイン

次に同様にセルフジョインの場合についても考えてみる。

```
SELECT result:<$book1.title, $book1.author>
FROM book1:bib.book, book2:bib.book
WHERE $book1.title = $book2.title;
```

この問合わせに対して作成される実行計画は

```
SJ   OP_EQ  $book1.title $book2.title
PROJ $book1 $book1.title $book1.author
```

である。この実行計画を実行する際に呼び出される問合わせ基本関数の列は次のようになる。

1. SJの右辺についてGetNodeIDbyPath を用いてパス記述が bib.book.title となるノード集合を得る。
2. 1 で得られたノード集合に対してそれぞれGetValuebyID を適用して要素の値を得る。
3. SJの左辺についてGetNodeIDbyPathAndVal を用いて パス記述が bib.book.title で値が 2で得られたそれぞれの値であるノード集合を得る。
4. 1.で得られたノード集合と3で得られたノード集合の組をつくる。
5. 3で得られたノード集合の組の右辺分に対してそれぞれ GetParentIDbyChild を適用してbib.book となるノード集合を得る。これが $book2 にバインドされる。
6. 1で得られたノード集合の組の左辺分に対してそれぞれ GetParentIDbyChild を適用してbib.book となるノード集合を得る。これが $book1 にバインドされる。
7. 6 で得られたノード集合に対してそれぞれGetChildIDbyParent を適用して $book1.titleとなるノード集合を得る。
8. 6 で得られたノード集合に対してそれぞれGetChildIDbyParent を適用して$book1.authorとなるノード集合を得る。
9. 7 で得られたノード集合に対してそれぞれGetValuebyID を適用して要素の値を得る。
10. 8 で得られたノード集合に対してそれぞれGetValuebyID を適用して要素の値を得る。

この演算の結果はノードの組の集合として得られる。ここでは、セルフジョインの右辺について先に値を求めて、その値に対応する左辺のノード集合を得た。これは反対に左辺について値をもとめて行ってもかまわない。

結果として、$book1 と $book2 が同じノードを指している場合の組も解に含まれる。したがって、この場合は bib.book がとりうるすべてのノード集合に対して、$book1と等しい $book2 の組みが解にふくまれることになる。

2.4 問合わせ演算の組み合わせ

実際にはここで述べた値による選択とセルフジョイン演算は単独で用いられるだけでなくANDなどを用いて組み合わせて条件指定されることもある。この場合、その組み合わせは次の６つに分類することができる。同種とはANDで結ばれている範囲変数に対する条件指定が同じ範囲変数に対して言及されているものか、別の範囲変数に対して言及されているものであるかという意味である。例を用いて説明する。

(1) 同種のＶＳ同士のAND

```
SELECT
result:<$book.author, $book.publisher.name>
FROM book:bib.book
WHERE $book.title = "B"
AND $book.author = "久保田";
```

この場合、１つめの条件指定も２つめの条件指定もともに同一の範囲変数 $book を限定する言及である。したがって、これらをAND で結んだものは、１つめの条件指定によって選択された $book を満たすノード集合と2つめの条件指定によって選択された $book を満たすノード集合の集合積となる。

(2) 異種のＶＳ同士のAND

```
SELECT
result:<$book1.author, $book1.publisher.name>
FROM book1:bib.book, book2:bib.book
WHERE $book1.title = "B"
AND $book2.title = "C";
```

この場合、１つめの条件指定と２つめの条件指定は異なる範囲変数を限定する言及である。

これらをANDで結んだものは、１つめの条件指定によって選択された $book1 を満たすノード集合と2つめの条件指定によって選択された $book2 を満たすノード集合の直積となる。この演算によって得られる解はノード集合の組の集合となる。

(3) 同種のＶＳとＳＪのAND

```
SELECT
result:<$book1.author, $book1.publisher.name>
FROM book1:bib.book, book2:bib.book
WHERE $book1.title = "B"
AND $book1.author = $book2.author;
```

この場合、１つめの条件を満たす$book1にバインドされたノード集合と２つめの条件を満たすノードの組の集合との間の集合積となる。したがって、求める演算結果は２つめの条件を満たすノードの組みの集合のうち、１つめの条件を満たすノード集合を$book1にとるようなノードの組みの集合となる。

(4) 異種のＶＳとＳＪのAND

```
SELECT
result:<$book1.author, $book1.publisher.name>
```

```
FROM book1:bib.book, book2:bib.book,
book3:bib.book
WHERE $book3.title = "B"
AND $book1.author = $book2.author;
```

この場合、１つめの条件指定と２つめの条件指定は異なる範囲変数を限定する言及である。したがって、この演算の結果は１つめの条件指定によって得られるノード集合と２つめの条件指定によって得られるノードの組みの集合の直積となる。この解はノードの３つ組の集合となる。

(5) 同種のＳＪ同士のAND

```
SELECT
result:<$book1.author, $book1.publisher.name>
FROM book1:bib.book, book2:bib.book
WHERE $book1.title = $book2.title
AND $book1.author = $book2.author;
```

この場合、１つめの条件指定の結果のノードの組の集合と２つめの条件指定の結果のノードの組の集合はともに条件を満たす$book1と$book2の組みの集合である。したがってそれらをANDで結んだ結果は２つのノードの組みの集合の集合積となる。

(6) 異種のＳＪ同士のAND

```
SELECT
result:<$book1.author, $book1.publisher.name>
FROM book1:bib.book, book2:bib.book,
book3:bib.book
WHERE $book1.title = $book2.title
AND $book2.author = $book3.author;
```

この場合、１つめの条件指定の結果のノードの組の集合は条件を満たす$book1と$book2の組みで、２つめの条件指定の結果のノードの組の集合は条件を満たす$book2と$book3の組みの集合である。したがって、これらをANDで結んだ結果はこれらの直積となる。この場合、解はノードの３つ組の集合となる。

実際にはこれらをさらにANDなどで結んだもっと複数の条件指定が考えられるが、これについては定められた順番で演算を処理していくことで上記のいずれかの場合と同様に処理することができる。ここではANDについて説明した。ORの場合については、同種のノード集合同士の演算の場合は集合和になる。異種の場合は直積を求めることになるが、実際には各範囲変数について、その範囲変数のパス記述がとりうるすべてのノード集合にたいして直積を求める演算処理が必要となり、データベース内のノード数が大きい場合、現実問題として処理をすることが現実的でないことが予想されるため、今回はこの部分について実装を行っていない。しかしながら理論上は同様の処理で解が得られることになる。

2.5 問合わせ実行の最適化

前節までで、XQL言語処理部における問合わせ実行計画生成と、実行エンジンでの各演算パターンにおける基本的な実行方式について説明した。

ここでは、いままでの話をもとにして、本システムで行っている最適化と行おうとしている最適化について述べる。現状で行っている最適化は非常に単純なものに限定されている。

現状で行っているものは次の通りである。

(1) 言語処理部での最適化

- WHEREの条件で指定された条件節を解くためのインデックスの選択

(2) 実行エンジンでの最適化

- ネステッドループをまわす場合の内外の入れ替え
- セルフジョインを処理する際の右辺と左辺の入れ替え
- 同種のＶＳとＳＪのAND 演算の際にVSの結果でSJの結果を絞る
- 中間結果が空集合となった場合の処理の打ち切り

これに対して今後導入を予定している最適化については以下の通りである。

ここでは問合わせ実行プランレベルでの処理の入れ替えなどの複数の処理にまたがる最適化を全体的な最適化、各処理単位での実行性能を向上させるための最適化を局所的な最適化と呼ぶ。

(3) 全体的な最適化

- 実行順序の入れ替え
  問合わせ実行計画を立てる際に、値による選択を先に処理しセルフジョインをあとで処理するような実行計画を生成する。また、同一の範囲変数に対する処理はまとまって処理するように順序を入れ替える。
- 構文解析による処理の省略
  入力されたXQL文を静的に評価して、結果が空集合になることが予知できるような部分については処理を省略する。また冗長な演算をまとめる。

(4) 局所的な最適化

- すでに得た検索結果の再利用
  VSの結果を用いてSJの処理を行う。
  たとえばVSとSJのANDの際にVSの結果を用いて

SJを行うことでループ数を削減する。

- 統計的な処理の選択
  問合わせ基本関数の平均的な実行時間をもとにして、処理を選択する。たとえばGetNodeIDbyPath の実行時間はGetNodeIDbyPathAndValに比較して約１５倍もかかってしまう。したがって、GetNodeIDbyPathを使わずにすむところではなるべくほかの方法を用いたり、GetNodeIDbyPathの結果を保存して再利用することで処理時間を短縮する。

3. システムアーキテクチャ

xQuesのシステムアーキテクチャを図２に示す。
各構成要素について簡単に説明する。

ＸＱＬ言語処理部

- XQLパーザ
  入力されたXQL文の構文チェックを行い構文木を生成する
- 問合わせ実行計画生成
  XQL構文木をもとに、問合わせの実行プランを生成する
- 問合わせ最適化
  問合わせの実行プランに対して、実行順序の最適化などの静的な最適化を行う

実行エンジン

- 問合わせ実行
  生成された実行プランを実行する。具体的には実行計画にもとづいて問合わせ基本関数群の呼び出しを行なう。
- 実行最適化
  実行時にすでに手元にある中間結果などを再利用し、統計的な方法を用いてネステッドループの内外をるなどの動的な最適化を行う。
- 検索結果生成
  検索結果にもとづいて出力のためのXMLデータを生成する。

インデックス格納部

- 問合わせ基本関数
  XMLインデックスに対して操作を行う６種類のコマンド群からなる。
- ＤＢ操作基本関数
  インデックスが格納されているデータベースに対してセッションのオープンクローズなどの制御を行う。
- XML検索インデックス格納用DB
  検索対象となるXMLデータの格納管理を行なう。XMLの木構造にもとづいたアクセスに必要なインデックスを提供する。

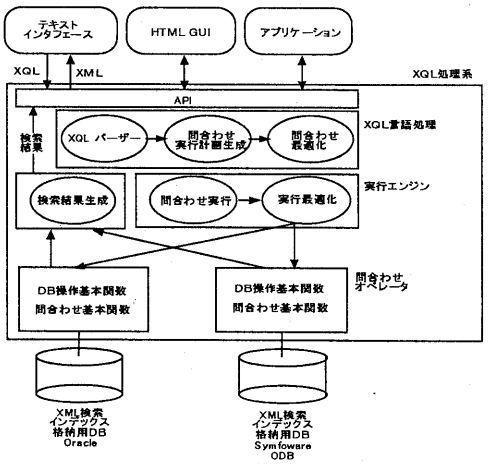

図２ xQuesのシステムアーキテクチャ

4. ＸＭＬ検索実験システム

本稿で述べたXMLデータに対する宣言的な検索機能を提供する問合わせ言語 XQL 処理系の実験システムを作成した。XMLデータに代表される半構造データを既存のデータベースに格納して、これに対して検索を行うための検索言語 XQLを実行する言語処理系を実装した。データを格納するDBMSとしてはORACLE8およびSymfoware ODBを用いている。実際には各DBMSごとに問合わせ基本関数などを準備することでDBMSを選ばずにインデックス格納部として用いることができる。

5. システムの状況と今後の課題

現在 本実験システムは一次試作の評価を行っている。現状では機能的に制限されている部分も多く、最適化もまだ多くの余地があるため、今後さらに改良を加えてゆく必要がある。

参考文献

[Lore] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, Jennifer : Lore: A Database Management System for Semistructured Data, http://www-db.stanford.edu/lore, 1998

[FJ XQL] http://www.w3.org/TandS/QL/QL98/pp/flab.doc, 1998

[Ishikawa99] Ishikawa, H., et al.: Document Warehousing Based on a Multimedia Database System, Proc. IEEE 15th Intl. Conference on Data Engineering, pp.168-173 (1999).

[MS XQL] http://www.w3.org/TandS/QL/QL98/pp/xql.html, 1998

[QL98] http://www.w3.org/TandS/QL/QL98/Overview.html, 1998

[XML-QL] http://www.w3.org/TR/1998/NOTE-xml-ql-19980819, 1998