データベースシステム 119－4
（1999. 7. 21）

XML問い合わせ処理システム(xQues)の問い合わせ言語

石川 博[1]　久保田 和己[1]　金政 泰彦[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言語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XQLの実装について触れる。

A Query Language of an XML Query Processing System (xQues)

Hiroshi Ishikawa[1], Kazumi Kubota[1], and Yasuhiko Kanemasa[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 language for XML data and explain the functionality. We make comments on an implementation of XQL for an XML query processing system called xQues.

1. Introduction

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 will provide a query language for XML data tentatively called *XQL* (Xml data Query Language) [FJ XQL], which has coincindetally the same name as a query language proposed by Microsoft and et al. [MS XQL].

A query language called XML-QL [XML-QL] has already been proposed to W3C, which has largely inspired the design of XQL. The first workshop on XML query language was successfully held [QL98]. We have designed XQL, keeping in mind its continuity with database standards such as SQL [ANSI X3] and OQL [ODMG] although we don't stick to its strict conformity. We will describe the requirements for a query language for XML data in Section 2 and explain the functionality of XQL by using the example data in Section 3.

[1] 富士通研究所
FUJITSU LABORATORIES LTD.

2. Requirements

We consider the following requirements as mandatory for a query language for XML data:
- XML query languages must take XML data as input and give XML as output.
- XML query languages must understand features of XML data structures such as elements and tags. In particular, the users must be able to specify hierarchical structure of tag paths in a query.
- XML query languages must provide operations on XML data, that is, XML versions of relational operators such as select, join, sort, grouping.
- XML query languages must be able to combine heterogeneous XML data from different sources specified by different URIs.
- XML query languages must view XML data as ordered sets of elements and must preserve the orders. It must provide set operators over XML data such as union, intersection.
- XML query languages must allow specification of regular expressions on tag paths although full capability may be unnecessary because of its computational complexity.
- XML query languages must allow the users to define view on XML data analogous to relational views. That is , the users must be able to define XML data views as functions by using XML query languages and to specify such functions in a query.
- An XML query must be embedded in XML data.
- XML query languages must keep syntactic and semantic continuity with other standards such as SQL and OQL.
- XML query languages must be processed efficiently. The language processor must provide query optimization. The processor must use schema information if available although it doesn't assume the existence of schemas.

3. Design

3.1 Database schema

We use database schemas or DTD by slightly changing example DTD used in XML-QL[XML-QL] as follows:

```
<!ELEMENT book (author+, title, publisher)>
<!ATTLIST book year CDATA>
<!ELEMENT article (author, title, year?)>
<!ATTLIST article type CDATA>
<!ELEMENT publisher (name, address)>
<!ELEMENT author (firstname?, lastname, office+)>
<!ELEMENT office (CDATA | longoffice)>
<!ELEMENT longoffice (building, room)>
<!ELEMENT watch (name, brand)>
```

Here DTD for book elements indicates that a book has at least one author and one mandatory title and publisher. DTD for article elements indicates that an article has one mandatory author and title and one optional year. An article has a type as an attribute. Note that an author has at least one office, which has a variant structure of either literal data or a pair of building and room.

3.2 Functionality

We describe the functionality of XQL by using schemas introduced in the previous section. XQL has a select-from-where construct as a basis, similar to SQL and OQL. We have borrowed examples from XML-QL [XML-QL].

(1) Data match for select

The following query retrieves authors of books published by Addison-Wesley:

select $book.author
from bib URI "www.a.b.c/bib.xml",
book $bib.book
where $book.publisher.name
= "Addison-Wesley"

The basic unit of XQL is a *path expression*, that is, an element variable (explained just below) followed by a series of tag names such as "$bib.book". The user declares *element variables* in a from-clause such as bib and book. In particular, the user can bind XML data as input specified by URI to element variables such as bib. References of element variables are done by prefixing "$" to them such as $bib.book. In a select-clause, the user specifies XML data as a result such as $book.author. Results have a tag for author as default in this case. The user checks data match for select in a where-clause, such as $book.publisher.name ="Addison-Wesley".

We briefly describe the semantics of XQL based on set theory. A set of XML elements is either ordered or unordered. XML elements have hierarchical structures; XML elements contains other XML elements, (i.e., sub-elements). We consider sub-elements and attributes as semantically the same although attributes have no hierarchical structures. In fact, elements are restricted by conditions specified on their sub-elements and attributes. XQL queries produce a set of elements satisfying conditions. Assuming that an element Ea satisfies conditions Ca and C'a over an element variable a, we define the semantics of XQL queries as follows:

Query: *select a from a where Ca*
Semantics: { Ea | Ca}
Query: *select a from a where Ca and C'a*
Semantics: { Ea | Ca } intersection { Ea | C'a }

Of course, if "or" is specified instead of "and" in the second query, then "intersection" is replaced by "union" in the semantics.

(2) Data constructor

The following query produces new results consisting of authors and titles of books published by Addison-Wesley:

```
select result <$book.author, $book.title>
from bib URI"www.a.b.c/bib.xml",
  book $bib.book
where $book.publisher.name
= "Addison-Wesley"
```

Here "<>" in a select-clause creates new XML elements of a specified construct such as author and title tags. New elements have a name result. This allows extraction and combination of sub-elements at any level.

(3) Grouping

The following query groups book titles for each book publisher:

```
select result <$book.publisher.name,
  $book.title >
from bib URI "www.a.b.c/bib.xml",
  book $bib.book
where $book.author.lastname = "Ishikawa"
groupby $book.publisher.name
```

A groupby-clause indicates that result elements are grouped by book publisher name. Further, titles of result elements are automatically nested. We don't use nested query for grouping unlike XML-QL [XML-QL].

(4) Sorting

For example, an orderby-clause explicitly sorts book publisher names and titles in an alphabetical order by publisher name as follows:

```
select result <$book.publisher.name,
  $book.title >
from bib URI "www.a.b.c/bib.xml",
  book $bib.book
where $book.author.lastname = "Ishikawa"
orderby $book.publisher.name
```

In this case, book titles of result elements are not nested unlike groupby. Note that XQL also preserves orders of XML data specified in a query implicitly.

(5) Join

The following query joins books published after 1995 and articles by authors as a join key within the same XML data:

```
select result <$article, $book>
from bib URI "www.a.b.c/bib.xml",
   article $bib.article, book $bib.book
where $book.author.firstname
        = $article.author.firstname and
      $book.author.lastname
        = $article.author.lastname and
      $book.@year > "1995"
```

In a where-clause, the user specifies join of books and articles by authors within the same XML data. Attributes such as year are referenced similar to tags by prefixing "@" to them, such as $book.@year.

(6) Tag variable [XML-QL]

The following query retrieves heterogeneous elements such as book titles published in 1995 with either authors or editors whose name is Smith:

```
select result <$any.title, $AorE>
from bib URI "www.a.b.c/bib.xml",
  any $bib.%,
  AorE $any.(author | editor)
where $any.@year= "1995" and
  $AorE.lastname ="Smith"
```

We don't use tag variables introduced by XML-QL [XML-QL]. Instead, we allow regular expressions as path expressions so that the user can simulate tag variables by using element variables declared as regular path expressions (See also (7) in this section). For example, "$any.(author | editor) " matches path expressions such as "book.author", "book.editor", "article.author", and "article.editor". Multiple occurrences of "%" in a from-clause are supposed to be bound to the same path at the same time. The above query can be expressed alternatively by using a set operator union as follows:

```
(select result <$book.title, $book.author>
from bib URI "www.a.b.c/bib.xml",
  book $bib.book
where $book.@year= "1995" and
  $book.author = "Smith")
union
(select result <$book.title, $book.editor>
from bib URI "www.a.b.c/bib.xml",
  book $bib.book
where $book.@year= "1995" and
  $book.editor ="Smith")
union      ...
```

(7) Regular path expression

The following query retrieves last name of authors whose office is either a whole house or a room in a building:

```
select result <$author.lastname>
from bib URI "www.a.b.c/bib.xml",
  author $bib.author,
  office $author.(office | office.room)
where $office = "245"
```

Here "office" is bound to "$author.office" or "$auther.office.room". The above query is alternatively expressed like this:

```
select result <$author.lastname>
from bib URI "www.a.b.c/bib.xml" ,
  author $bib.author
where $author.(office | office.room) = "245"
```

(8) Join of data from multiple sources

The following query produces book author name and income by joining social security numbers of book authors and taxpayers at different data sources indicated by different URIs such as *b* and *t*.

```
select result <$author.lastname, $t.income>
from b URI "www.a.b.c/bib.xml",
  t URI "www.irs.gov/taxpayers.xml",
  author $b.book.author
where $author.ssn=$t.ssn
```

The user can specify join of heterogeneous XML data from different sources indicated by different URIs such as *d* and *t*.

(9) Embedding query

The following XML data result to sets of article titles and book titles published after 1995:

```
<result>
 <articles>
  <XQL>
(select $article.title
 from bib URI "www.a.b.c/bib.xml", article
$bib.article
 where $article.@year > "1995")
  </>
 </>
 <books>
  <XQL>
(select $book.title
 from bib URI "www.a.b.c/bib.xml", book
$bib.book
 where $book.@year > "1995")
  </>
 </>
</>
```

The user can embed an XQL query in XML data although XML parsers must be extended to recognize XQL.

(10) Function definition

The following finds declared income of employees by joining two sets of elements passed as parameters Taxpayers and Employees:

```
FUNCTION findDeclaredIncomes (Taxpayers,
Employees) as
(select result <$Employees.name,
    $Taxpayers.income>
from Taxpayers, Employees
where $Employees.ssn=$Taxpayers.ssn)
```

The user defines a function by specifying an XQL query in its body. The role of functions is similar to that of relational views. See (11) in this section for invocation of functions.

(11) Function invocation

For example, in a select-clause, the user inserts to an attribute *id* values of functions such as "PersonID ($author.firstname, $author.lastname)" and introduces "publicationtitle" as a new tag:

```
select result <@id =
  PersonID($author.firstname,
           $author.lastname),
  $author.firstname, $author.lastname,
  publicationtitle   $title>
from bib URI "www.a.b.c/bib.xml", any $bib.%,
author $any.author, title $any.title
```

4. Conclusion

We have proposed XQL as a query language for XML data of continuity with database standards. We expect to activate emergence of an easily understandable query language. We compare our XQL with other proposals. XML-QL [XML-QL] has much functionality in common with our XQL. It makes more emphasis on join of multiple data sources distributed over the Web. XML-QL is not necessarily nonprocedural unlike our XQL; Its multiple conditions are assumed to be sequentially evaluated. Another XQL of Microsoft and et al. [MS XQL] focuses more on filtering a single XML document by flexible pattern match conditions; It provides no join of multiple documents unlike our XQL.

We conclude this paper by making comments on the implementation. We have just started to explore approaches to mapping DTD to databases (RDB or ODB such as [Ishikawa96]) and to implement an XQL processor [Ishikawa99]. We are now developing a subset of XQL basic functions as a query language of an XML query processing system called *xQues* (*X*ML *que*ry processing *s*ystem). If any DTD or schema information is available, we try to map elements to tables and tags to fields, respectively; Otherwise, we divide XML data into nodes and edges and store them into separate tables for nodes and edges with neighboring data physically clustered for reducing I/O cost. When the users issue a query against global servers for XML data, if the servers can understand XQL, the system obtains query results as XML data; Otherwise, it obtains whole XML data specified by URIs and processes the query against them locally.

References

[ANSI X3] http://gatekeeper.dec.com/pub/standards/sql, 1998

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

[Ishikawa96] Ishikawa, H., et al.: An Object-Oriented Database System Jasmine: Implementation, Application, and Extension., IEEE Trans. Knowledge and Data Engineering, vol. 8, no. 2, pp.285-304 (1996)

[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

[ODMG] Cattell, R.G.G. and Barry, D.K., Eds.: *Object Database Standard: ODMG 2.0*, Morgan Kaufmann Publishers, Inc., 1997

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

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