DEWS2007 M6-5

関係データベースに基づくXMLデータのためのOLAP

キットチャントラ† 天笠俊之†,†† 北川博之†,††

† 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
†† 筑波大学計算科学研究センター
〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †kchantola@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

あらまし Extensible Markup Language (XML) は，ネットワークにおけるデータ表現・交換フォーマットになった．XML はあらゆる分野で標準的に使われるようになってきているため，今後はこれまでの単純な問合せ処理に加えて，新たな情報を発見するための複雑な分析処理が必要になることが予想される．そこで本研究では，XML の特徴を考慮に入れながら, XML データの多次元の分析を実行することができるように，XML-OLAP の分析手法を提案する．利用者は XPath を用いて XML データキューブを指定し，OLAP 拡張を施した XQuery によって分析処理を行う．システムの実装には関係データベースを利用し，与えられたデータキューブ定義や分析要求を SQL に変換することで処理を行う．最後に実験による評価を行い，提案手法の妥当性を示す．

キーワード XML, OLAP, XPath, XQuery

An Approach to XML-OLAP Based on Relational Databases

Chantola KIT†, Toshiyuki AMAGASA†,††, and Hiroyuki KITAGAWA†,††

† Department of Computer Science, Graduate School of Systems and Information Engineering
†† Center for Computational Sciences
University of Tsukuba
1–1–1 Tennodai, Tsukuba, Ibaraki 305–8573, Japan
E-mail: †kchantola@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

Abstract Extensible Markup Language (XML) has become an important format for data exchange and representation on the web. In addition to conventional query processing, more complex analysis on XML data is considered to become important in order to discover valuable information. In this research, we attempt to investigate an XML-OLAP, by which we can perform multidimensional analysis on XML data taking XML's features into account. Users are allowed to specify XML data-cube by XPath, and perform analytical processing by XQuery with OLAP extensions. The system is implemented on top of relational databases, and the given requests for data-cube specification and analysis are translated into SQL so that they can be processed using the underlying system. We show the feasibility of the proposed scheme by experimental evaluations.

Key words XML, OLAP, XPath, XQuery

1. Introduction

Since its emergence in 1998, the Extensible Markup Language (XML) [1] has become a de facto standard for data exchange and representation on the web. Now, XML has been used in a wide spectrum of application domains, such as web documents, business documents, and log data. For this reason, in addition to such simple query retrieval, more complex ways to make analysis of XML data are considered to be more and more important in order to extract useful information from massive XML data.

In our previous research [2], we proposed a model for OLAP analysis on XML data using relational databases. Specifically, for allowing users to specify facts and dimensions about XML data, we employ slightly extended XPath expressions. The system extracts corresponding XML fragments from the underlying XML database based on the fact and dimension specifications, and constructs multidimensional XML data-cube. The users then make analysis on the data-cube by issuing multidimensional queries. One notable feature of the work is that we take account of structure-based concept hierarchy, as well as value-based concept hierarchy, which is an im-

portant characteristic of XML data.

In this paper, we will discuss an approach to XML-OLAP system using relational database systems based on our previous work. Our contributions in this paper are as follows:

- We discuss roll-up operation for XML data-cube. It is an extension in SQL2003 for supporting OLAP operations for relational data-cube. We employ the syntax, and adapt it for XML data-cube. We then discuss its implementation using the functionality of relational database systems.
- We evaluate the performance of the proposed scheme by a series of experiments. The experimental results show that the proposed scheme can deal with 100MB XML data with reasonable processing time.

The rest of this paper is organized as follows: in Section 2, we introduce preliminaries which we describe about OLAP and XML. Then, in Section 3, we discuss related works. In Section 4, we show an overview of our proposed system and the definitions of fact and dimension of XML data, XML hierarchies, and data cube on XML data. We also discussed OLAP extensions to XQuery in this section. In Section 5 we describe our implementation issue which we will discuss relational XML storage, data cube construction, and query processing with both structure- and value-based grouping, and ROLLUP operation. In Section 6, we give experimental evaluation. Finally, in Section 7, we conclude this paper.

2. Preliminaries

In this section, we briefly overview OLAP, XML and its query languages, XPath and XQuery.

2.1 Online Analytical Processing (OLAP)

Online Analytical Processing (OLAP) is a category of software technology that enables analysts, managers, and executives to obtain insight into data through fast, consistent, interactive access to a wide variety of possible views of information. The information has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by users.

When considering OLAP, star schema, cube, and aggregation operations are the most important concepts. To represent the multidimensional data model, **star schema**, that consists of single fact table and some dimension tables, is used. Each dimension table contains columns corresponding to attributes of the dimension.

An OLAP system models the input data as a logical multidimensional **cube** with multiple dimensions which provides the context for analyzing measures of interest. To analyze the data with the cube structure, various aggregation operations, namely, drilling, pivoting (or rotating), and slicing-and-dicing, are used to change the number of dimensions and the resolutions of dimensions of interest.

The cube, in Figure 1, shows a sales cube of three dimensions, area, category, and price. Starting from the cube at the upper left, we can create a new cube with coarser granularity on the area axis by applying roll-up operation. We can go back to the finer granularity by drill-down operation. By slicing on the price dimension, we can get only the common price cube. Dice operation enables us to change order of the dimensions.

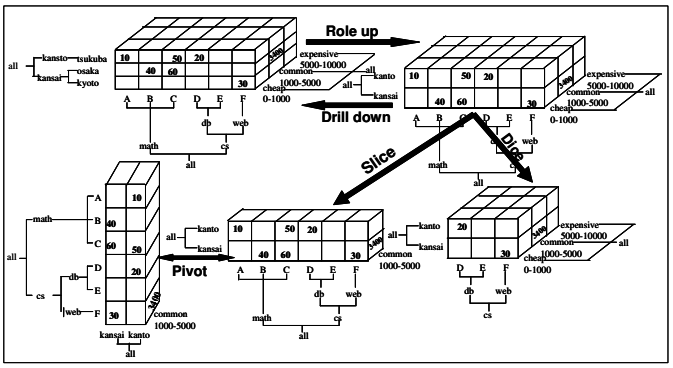

図 1　OLAP example
Fig. 1　OLAP example.

2.2 XML, XPath, and XQuery

XML has become the language of choice for data representation across a wide range of application. It has been designed to represent both structured and semi-structured data. An XML data is basically modeled as a labeled tree: elements and attributes are mapped into nodes of graph; directed nesting relationships are mapped into edges in the tree. The path of a node n in an XML tree refers to the path from root of the tree to n. The length of a path p is the number of nodes in p. If the length of path of a node a is smaller than that of node b, then we say, a is higher than b. Each element o in XML document can be considered as a sub-tree which is a tree with o as the root and contains all descendants of o.

XML data can be queried by XML query languages such as *XPath* and *XQuery*. **XPath** [3] is a language for addressing portion of an XML data. We can specify an XML sub-tree in term of a navigational path over XML tree by conditions on the element's label, value, and relationship among nodes along the path.

XQuery [4] is a query language designed to query collection of XML data. XQuery uses XPath as a sub-language to address specific parts of an XML document. It employs SQL-like FLWOR (FOR, LET, WHERE, ORDER BY, RETURN) expression for performing joins.

3. Related Works

Bordawakar et al. [5] investigated various issues related to XML data analysis, and proposed a logical model for XML analysis based on the abstract tree-structured XML representation. In particular, they proposed a categorization of XML data analysis system: 1) XML is used simply for external representation for OLAP results, 2) Relational data is extracted from XML data, and then processed with existing OLAP systems, 3) XML is used for both data representation and analysis. In order to support complex analytical operations, they also proposed new syntactical extensions to XQuery, such as "GROUP BY", " ROLLUP", "TOPOLOGICAL ROLLUP", "CUBE", and "TOPOLOGICAL CUBE". In our research, we employ the syntax of "GROUP BY ROLLUP" and "GROUP BY TOPOLOGICAL ROLLUP" to allow users to specify OLAP operation in XQuery.

Jensen et al. [6] proposed a scheme for specifying OLAP cubes

on XML data. They integrated XML and relational data at the conceptual level based on UML, which is easy to understand by system designers and users. In their scheme, a UML model is built from XML data and relational data, and the corresponding UML snowflake diagram is then created from the UML model. In particular, they considered how to handle dimensions with hierarchies and ensuring correct aggregation.

Pedersen et al.[7] proposed a federation of OLAP and XML, which allows external XML data to be presented along with dimensional data in OLAP query results. It enables the uses of external XML data for selection and grouping. It is the same to the third approach mentioned by Rajesh Bordawakar et al.[5]. They allow XML data to be used as "virtual" dimensions, and present a data model and multi-schema query language based on SQL and XPath.

4. An Overview of the Proposed XML-OLAP System

4.1 System Overview

The left side of Figure 2 shows an overview of our proposed scheme. According to the content of XML data, a user at first gives a fact path and some dimension paths in XPath expression to denote his/her interest. Referring to the given fact and dimension paths, the system produces an XML cube. After getting the cube, the user can make analysis of the XML data-cube using XQuery with OLAP extensions.

The following discusses how XML cube can be constructed in our system.

4.2 Formal Definitions

To construct an XML data-cube, we first need to specify fact and dimensions. Let us look at the definitions of fact and dimensions.

4.2.1 Facts about an XML Data

A fact-table in a traditional OLAP system stores data items which are to be analyzed. We attempt to define the facts in an XML data after the traditional OLAP way. In order to identify the facts, we use XPath as the query language.

[Definition 1] (Fact path) A fact path (p_f) is an absolute XPath expression that identifies data items of interest.

For example, when a user wants to get information of book sales from sales XML data as in the upper left side of Figure 3, the related data items can be obtained by the fact path p_f = `doc("sales.xml")//b`.

4.2.2 Dimensions

Having fixed the fact data, we might additionally need some dimensions whose values are used to group the facts together for the subsequent aggregation operations. In traditional OLAP systems, dimensions are given as independent tables associated with the fact table. In this work we try to define a dimension as an XPath query, but we need to care about the relationship between the fact data and dimensions. In order to ensure this, a dimension path is in either of the two cases: relative path from the fact path and absolute path with referential constraints.

[Definition 2] (Dimension path) A dimension path is an XPath expression (p_d) in either of the two forms:

(1) p_d is a relative path expression originated from the fact path p_f, or

(2) p_d is an absolute path expression contains at least one condition with the fact path p_f.

Figure 3 shows an example of fact and dimension paths. The circles on the top left document represent the facts corresponding to p_f. When we want to use the book title as a dimension for the subsequent analysis, a dimension path can be given as p_{d1} =`t`, which is a relative path from p_f. If we are interested in grouping the books according to price ranges which is represented in another XML data (the upper right document of Figure 3), we need to specify absolute path expression with referential constraints like p_{d2} =`doc("bookinfo.xml")//b[t =` p_f`/t]/p`. As can be seen from the example, for a given book, we can obtain corresponding price in another XML data by using title as the clue.

4.2.3 Concept Hierarchy

The concept hierarchy is a notable feature of traditional OLAP systems by which we can carry out flexible grouping operations over the data items stored in the fact table. As with the traditional OLAP systems, we assume that value-based concept hierarchies are given beforehand. We do not go into the detail of how to represent such a hierarchy, because it is beyond the scope of this paper. When dealing with XML data in the same context, we need a special consideration on the semistructured nature. Specifically, we have to take into account structure-based concept hierarchy which is naturally represented as the hierarchical structure of XML data.

Taking Figure 3 for example, all books (b) are categorized by the XML hierarchies according to the area or book category. The structure-based concept hierarchy allows us to aggregate facts using such XML data structure. We will discuss the detail later.

4.2.4 Data Cube on XML Data

We are now ready to define data cube on XML data using the concepts of the fact and dimension paths. Before going into the definition, we introduce some notations as helpers. For a given XPath expression p, $[[p]]$ denotes an evaluation of p, and the result would be XML nodes, string-values, or a boolean. Let $[[p]]$ denotes an evaluation of p where p represents an XPath expression.

[Definition 3] (XML data-cube) An XML-cube is defined as (p_f, D) where p_f is a fact path and $D = \{p_{d1}, p_{d2}, \ldots, p_{dn}\}$ is a set of dimension paths. A fact f in the cube is an $n+1$-tuple $(f, d_1, \ldots, d_n)$ where $f \in [[p_f]]$ and each d_i is obtained by evaluating p_{di}: $[[p_{di}]]_f$ if p_{di} is in a relative form or $[[p'_{di}]]$ where p'_{di} can be obtained by replacing each occurrence of p_f/p_r in p_{di} with $[[p_r]]_f$. n is the rank of the XML-cube.

Let us consider an XML data-cube as an example (Figure 3). It is defined as $(p_f, \{p_d\})$, where p_f =`doc("sales.xml")//b` and p_d =`doc("bookinfo.xml")//b[t =`p_f`/t]/p`. A tuple can be extracted as follows. Firstly, fact data can be extracted by evaluating fact path like $[[p_f]] = \{b_1, b_2, \ldots, b_6\}$. For each fact data b_i, we can identify corresponding dimension data in another XML data as specified by p_d. When evaluating p_d, we need to rewrite the path according to the fact data. For example, for the fact b_1, p_f`/t`, which is a part of p_d, is rewritten as $[[p_f$`/t`$]]_{b1} = \{$`"A"`$\}$, that turns

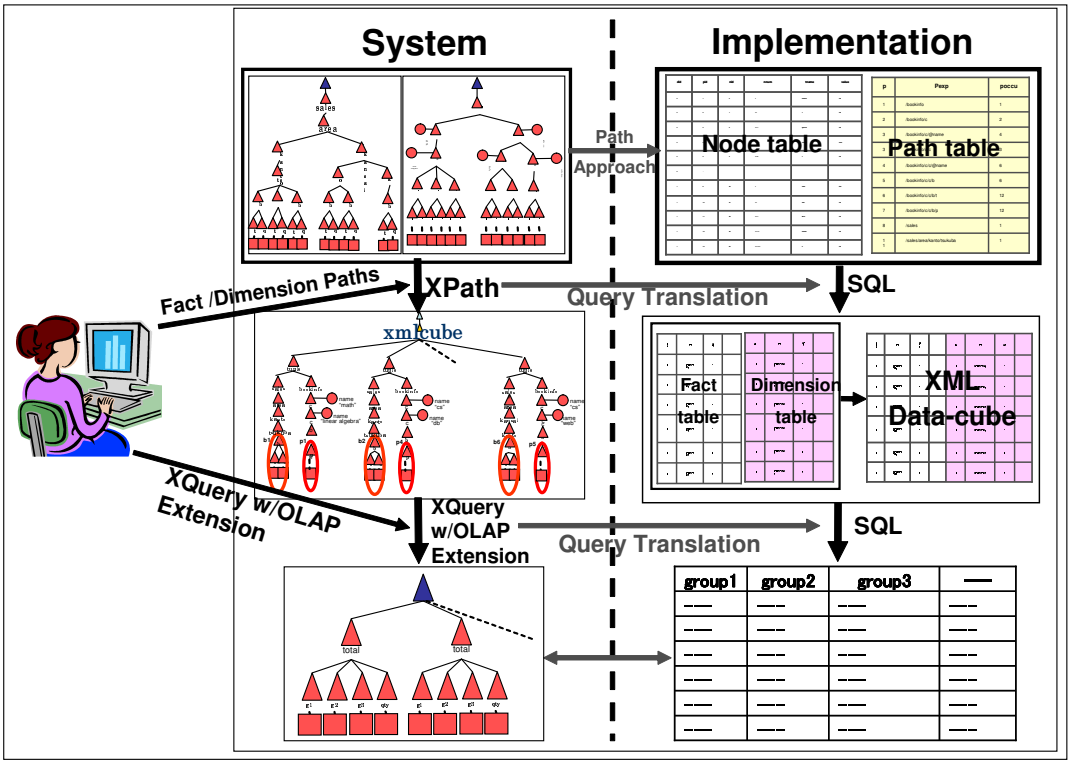

図 2　System overview.
Fig. 2　System overview.

out to be `doc("bookinfo.xml")//b[t = "A"]/p`. In this way, we can extract all tuples from the data cube, that are set of 2-tuple: $\{(b1, p1), (b2, p4), (b3, p3), (b4, p3), (b5, p2), (b6, p5)\}$.

In contrast to the existing OLAP, and XML-cube may contain much information more than the dimensionality (what we call "rank"). That is, each XML fragment may contain more information than a numerical value, such as elements, texts, attributes, and hierarchical information. In order to form a cube-like structure, we need to specify some of them as dimensions of the cube structure.

For instance, each tuple of the rank 1 XML data-cube in Figure 3 (lower side) contains two XML fragments of books coming from "sales.xml" and prices from "bookinfo.xml". According to the fragments, this XML data-cube potentially has five attribute values: title, quantity, area, price, and category. Assume that we are interested in getting the information related to the book sales area and price, we can create a cube by specifying the area and price as the dimension. Figure 4 shows the cube.

4.3 OLAP Extensions to XQuery

Once the data cube is constructed, we can make analysis using the dimensions, and related information such as XML hierarchies as clues to aggregate measures of the facts. In our system, we attempt to use XQuery as the user query language. However, the current version of XQuery does not support aggregation function. So we employ the syntax of OLAP extension for XQuery [5], GROUP BY ROLLUP and GROUP BY TOPOLOGICAL ROLLUP. The same as the roll-up operation in ordinary OLAP systems, ROLLUP enables a SELECT statement to calculate multiple levels of subtotals across a specified group of dimensions. It also calculates a grand total. ROLLUP is an extension to the GROUP BY clause so its syntax is extremely easy to use. The latter, TOPOLOGICAL ROLLUP, is similar to ROLLUP but for computing structure-based grouping over XML data.

For example, related to the example of sales XML data-cube, a user may be interested in viewing the total price of the sold books in each area from the lowest to the highest level by giving an XQuery as bellow:

```
FOR $s in xmlcube/tuple/pf
    $b in xmlcube/tuple/pd
WHERE $s//b/t = $b/t
GROUP BY TOPOLOGICAL ROLLUP($s//b)
LET $p := SUM($s//b/q * $b/p)
LET $a := $s/../b
RETURN
<book>
  <area> $a </area>
  <totalprice> $p </totalprice>
<\book>
```

To perform this query with XML data-cube, the system uses existing query translation to convert such XQuery to SQL which the detail will be discussed later.

5. Implementation Using Relational Database Systems

This section discusses an implementation of the proposed model and grouping operations (Figure 2, right). We try to make the best use of relational databases as the underlying data storage. The reasons are: 1) there are many commercial and open source products, 2) enormous amount of information resources are stored in rela-

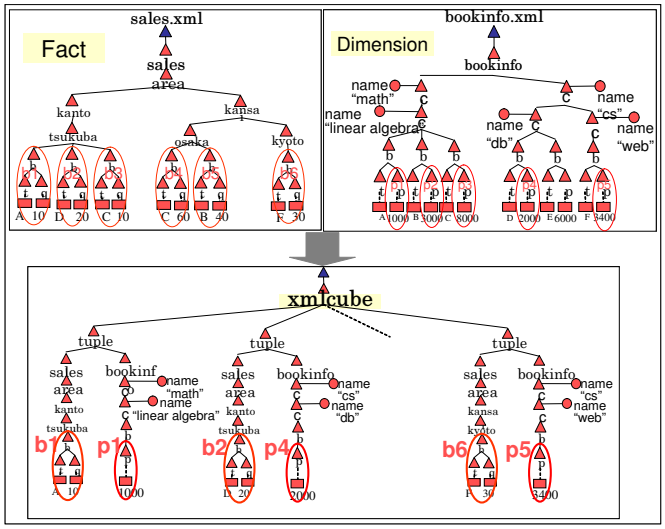

図 3　Facts, dimensions, and Sales XML data-cube.

Fig. 3　Facts, dimensions, and Sales XML data-cube.

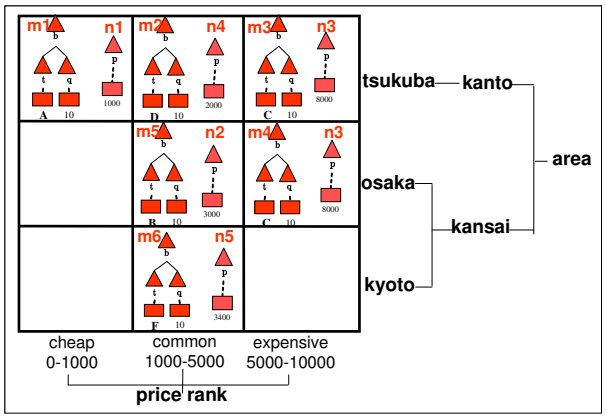

図 4　2 dimensinal XML-cube.

Fig. 4　2 dimensional XML-cube .

tional systems, and 3) we can leverage established relational XML storage techniques. In addition, we can utilize grouping, and OLAP extension functionalities, which are supported in most relational database systems, to implement value- and structure-based grouping of XML data.

5.1　Relational XML Storage

We employ the path-approach [8] for mapping XML data to relational tables, because we can manage any well-formed XML documents with fixed relational schema and realize practical subset of XPath solely by the use of SQL functionalities. Due to the limitation of pages, we just show the brief overview. In the path-approach, an XML node is basically mapped to a relational tuple of two tables, path table containing all absolute path expression of all XML nodes, and node table containing all XML node information. Table 1 (left) shows the path table extracted from “sales.xml” and “bookinfo.xml”. The attributes pid, $pexp$, and $poccur$ in the path table denote path id to join path table with node table, the absolute path of XML node, and occurrences of the path expression, respectively. In the node table (Table 1, right), there are did, pid, nid, $nnum$, $tname$, and $value$. Attribute did denotes the id of the

表 1　Path table and node table.

Table 1　Path table and node table.

pid	pexp	poccur
1	/bookinfo	1
2	/bookinfo/c	2
3	/bookinfo/c/@name	4
4	/bookinfo/c/c	3
5	/bookinfo/c/c/@name	6
6	/bookinfo/c/c/b	6
7	/bookinfo/c/c/b/t	12
8	/bookinfo/c/c/b/p	12
9	/sales	1
10	/sales/area	1
11	/sales/area/kanto	1
12	/sales/area/kanto/tsukuba	1
13	/sales/area/kanto/tsukuba/b	3
14	/sales/area/kanto/tsukuba/b/t	6
15	/sales/area/kanto/tsukuba/b/q	6
16	/sales/area/kansai	1
17	/sales/area/kansai/osaka	1
18	/sales/area/kansai/osaka/b	2
19	/sales/area/kansai/osaka/b/t	4
20	/sales/area/kansai/osaka/b/q	4
21	/sales/area/kansai/kyoto	1
22	/sales/area/kansai/kyto/b	1
…	…	…

did	pid	nid	nnum	tname	value
0	1	0	1	bookinfo	null
0	2	1	1.1	c	null
0	3	2	1.1.1	@name	null
0	3	3	1.1.1.1	CDATA	math
0	4	4	1.1.2	c	null
0	5	5	1.1.2.2	@name	null
0	5	6	1.1.2.2.1	CDATA	linear algebra
0	6	7	1.1.2.2	b	null
0	7	8	1.1.2.2.1	t	null
0	7	9	1.1.2.2.1.1	#TEXT	A
0	8	10	1.1.2.2.2	p	null
0	8	11	1.1.2.2.2.1	#TEXT	1000
…	…	…	…	…	…
1	9	46	1	sales	null
1	10	47	1.1	area	null
1	11	48	1.1.1	kanto	null
1	12	49	1.1.1.1.	tsukuba	null
1	13	50	1.1.1.1.1	b	null
1	14	51	1.1.1.1.1.1	t	null
1	14	52	1.1.1.1.1.1.1	#TEXT	A
1	15	53	1.1.1.1.1.2	q	null
1	15	54	1.1.1.1.1.2.1	#TEXT	10
…	…	…	…	…	…

XML document, pid is the path id referring to the path expression in the path table, nid, and $nnum$ are pre-order and dewey id used to identify the node, $tname$ denote the tag name, which is either of element name, “#TEXT”, “@attribute”, or “CDATA” depending on the type of the node. The last column is the value of the text or attribute node.

5.2　Extracting Fact and Dimensions

The first step is to extract fact and dimensions. As discussed in Section 4.2.1 and 4.2.2, a fact and its dimensions are XML subtrees specified by XPath queries. Hence, we can represent the fact (or a dimension) as a part of node table. This can be achieved by evaluating the fact (dimension) path, and storing the result as a new table. Those tables can be defined as either views or materialized views.

For example, from the relational tables (path and node tables), Appendix 1. shows the SQL query used to produce fact of the given example, and Appendix 2. contains the SQL query to create book price referring to the example dimension path.

5.3　Data Cube Construction

In the next we create an XML data-cube. For this purpose, we need to establish the relationships between the fact and the dimension as described in Section 4.2.4. We join the base relations by giving the referential constraints as the join key.

Appendix 3. consists of the SQL query which creates the XML data-cube table as shown in Table 2, by joining the fact and dimension tables using jkey as the relationship. Therefore the XML data-cube table contains all attributes from the fact and dimension and each record consists of data from the fact and dimension which have the same book title.

5.4　Query Processing

As discussed in Section 4.3, we use XQuery with OLAP extensions as the user query language. In order to process a query, we need to translate the query into SQL, because we make use of relational database systems as the query processing engine. In fact, there have been several works on XQuery to SQL query translation [9], and we can borrow those ideas. So, in this paper, we focus

表 2　XML data-cube example.
Table 2　XML data-cube example.

did	pid	pexp	nid	nnum	tname	value	jkey	did	pid	pexp	nid	nnum	tname	value	jkey
1	13	/sales/area/kanto/tsukuba/b	50	1.1.1.1.1	b	null	A	0	8	/bookinfo/c/c/b/p	11	1.12.2.2.1	#TEXT	1000	A
1	13	/sales/area/kanto/tsukuba/b	55	1.1.1.1.2	b	null	D	0	8	/bookinfo/c/c/b/p	32	1.2.2.2.2.1	#TEXT	2000	D
1	13	/sales/area/kanto/tsukuba/b	60	1.1.1.1.3	b	null	C	0	8	/bookinfo/c/c/b/p	21	1.1.2.4.2.1	#TEXT	8000	C
1	18	/sales/area/kansai/osaka/b	67	1.1.2.1.1	b	null	C	0	8	/bookinfo/c/c/b/p	21	1.1.2.4.2.1	#TEXT	8000	C
1	18	/sales/area/kansai/osaka/b	72	1.1.2.1.2	b	null	B	0	8	/bookinfo/c/c/b/p	16	1.1.2.3.2.1	#TEXT	3000	B
1	22	/sales/area/kansai/kyoto/b	78	1.1.2.2.1	b	null	F	0	8	/bookinfo/c/c/b/p	45	1.2.3.2.2.1	#TEXT	3400	F

on how to implement OLAP operations using SQL. Specifically, we discuss how to realize structure-based grouping and roll-up operations.

5.4.1 Structure-based Grouping

For value-based concept hierarchy, the task is easy, because we can assume that the concept hierarchy is given as an extra relational table, so we do not go into the detail. Here, we discuss how to realize grouping operation based on structure-based concept hierarchy. Our basic strategy is to utilize path expressions.

The main idea is that we can just use path expressions as the clue to perform grouping. Specifically, for a given data item, we need to compute the prefix of each data, and then perform grouping on the prefixes. The level of grouping can be controlled by the length of the path prefixes.

Now, we discuss how to perform grouping operations using path expressions. Let us introduce some notations. Given an XML node n, let $pexp(n)$ denote n's absolute path expression, and let $prefix(exp, i)$ denote path expression exp's i-th prefix, e.g., $prefix($"`/a/b/c`"$, 1) =$ "`/a`", and $prefix($"`/a/b/c`"$, 2) =$ "`/a/b`." Then, the grouping can be performed in the following way:

（1） Let the depth, which is the distance from the root, of the dimension be d, e.g., the depth of a path "/sales/area/kanto/tsukuba/b" is $d = 5$.

（2） Find the common prefix of all path expressions and let the depth be i, e.g., the three path expressions, "/sales/area/kanto/tsukuba/b", "/sales/area/kansai/osaka/b", and "/sales/area/kansai/kyoto/b" have the same prefix path "/sales/area". As a consequence, we get $i = 2$.

（3） The level-j ($i \leqq j \leqq d$) grouping can be computed by calculating $prefix(pexp(n), j)$ for each dimension value n, e.g., Referring to the previous three path expressions, let j be 3. The $prefix(pexp(n), 3)$ computes two level-3 groupings of the paths which have the same 3-depth prefixes, /sales/area/kanto and /sales/area/kansai.

In fact, the proposed grouping operation can be implemented in many ways, but an important remark is that it can be realized solely by the functionality of SQL. One possible way is to leverage the string match functionality provided by the database system. More precisely, we can make use of regular expressions to extract substrings, and use them with the `GROUP BY` clause. Assume that we would like to use the first two tags to group the facts, e.g., use `/sales/area` out of `/sales/area/kanto/tsukuba/b`, we can achieve this by:

```
SELECT ...
FROM ...
WHERE ...GROUP BY regexp_replace(dim.pexp,
            '^(/[^/]+/[^/]+)/.+', '\\1')
```

Another possibility is to introduce dedicated indexes based on Dewey encodings or prime numbers. They might be good for speeding up the grouping operations compared to the above approach. The comparison might be an interesting topic to research.

5.5 ROLLUP Operations

One possible way to implement roll-up operations (GROUP BY ROLLUP and TOPOLOGICAL ROLLUP) in the extended XQuery is to directly translate them into the counterpart in SQL2003, in which OLAP operations are supported. However, in many database systems, SQL2003 is not supported. For this reason, we try to realize the roll-up operations using the functionality of SQL-92, which is supported in most systems. Here we show how roll-up operations are applied to XML data-cube. As mentioned in Section 4.3, ROLLUP and TOPOLOGICAL ROLLUP create subtotals that roll up from the most detailed level to a grand total, we use UNION ALL, which enable us to compute set union over different grouping levels, to implement the operations. Since the space is limited and GROUP BY TOPOLOGICAL ROLLUP is similar to GROUP BY ROLLUP, let us look at structure-based GROUP BY TOPOLOGICAL ROLLUP example to implement the example in Section 4.3. The example rolls up the grouping of the area hierarchy in sales XML data, `/sales/area//b`, which has 3 levels. The query can be expressed as:

```
SELECT
FROM ...
WHERE ...GROUP BY regexp_replace(dim.pexp,
     '^(/[^/]+/[^/]+[^/]+[^/]+)/.+', '\\1')
UNION ALL
SELECT
FROM ...
WHERE ...GROUP BY regexp_replace(dim.pexp,
     '^(/[^/]+/[^/]+[^/]+)/.+', '\\1')
UNION ALL
SELECT
FROM ...
WHERE ...GROUP BY regexp_replace(dim.pexp,
     '^(/[^/]+/[^/]+)/.+', '\\1')
```

6. Experimental Evaluation

6.1 Experimental Setup

All experiments were performed in Sun Microsystems Sun Fire X4200 server whose CPU is a 2-way Dual Core AMD Opteron(tm) processor (2.4GHz). This machine has 16GB memory and runs Sun

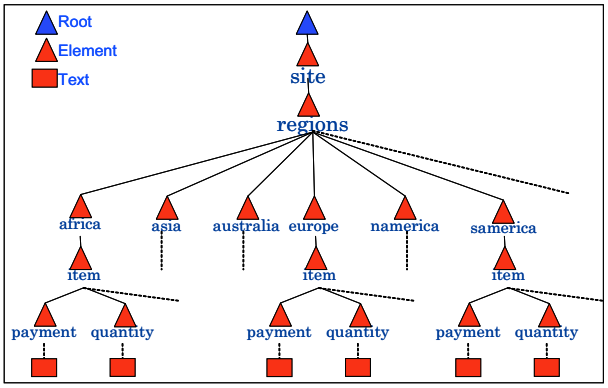

図 5　XMark data.
Fig. 5　XMark data.

OS 5.10. We used Java version 1.5.0_09 to parse XML data to relational tables, and PostgreSQL 8.1.4 to perform query processing.

For the experimental data, we used XMark data which is a comprehensive distributed system benchmarking and optimization suite. Figure 5 depicts the structure of the XMark data. We chose "regions" element for our experiment. Each "region" node contains child nodes representing world continents, like Africa, Asia, Australia, Europe, North America, and South America and some other child nodes. Each Continent node contains several "item" elements, and others. Each "item" node contains "quantity", "payment", and some other child nodes. We tested the following sizes of XML data: 10MB, 100MB, 200MB, 300MB, 400MB, and 500MB.

6.2　Benchmark Queries

For the benchmark query, we give a fact path, p_f = `doc("xmark.xml")//item`, and two dimension paths, p_{d1} = `quantity` and p_{d2} = `payment`.

We ran three queries to show the performance of roll-up functions of value-based, structure-based, and the combination.

a) Value-based Grouping (GROUP BY ROLLUP)

The value-based grouping will be done to group XML data based on the text node values. For example, we attempt to calculate all the subtotals of item quantity according to the payment method which we can obtain from the payment text values.

b) Structure-based Grouping (GROUP BY TOPOLOGICAL ROLLUP)

This grouping is based on the hierarchical structure of XML data, e.g., we calculate all the subtotals of the item quantity according to the XML hierarchical structure (/site/regions/.../item) representing regional containment relationship of items.

c) The Combination of Structure- and Value-based Grouping

Additionally, we can make grouping based on both value and structure of XML data. The result of the combining previous value- and structure-based example can be seen in Table 3.

6.3　Experimental Results

Table 4 and Figure 6 show the elapse times for data-cube construction. At first, "item", "quantity", and "payment" are created. After extracting those tables, the data-cube "payqty" is constructed. Table 5 and Figure 7 represent query processing time of roll-up operations. They are value-based (payment), structure-based (region), and their combination (regionpay). The results show that the processing time for data-cube construction is quite time consuming even for 100MB data. However, the important remark here is that once the data-cube is constructed, analytical query processing can be processed in reasonable time. In real systems, in many cases, data-cube construction is performed once in the midnight, and analytical processing are applied repeatedly in business hours. From the observation, we think that the performance of the proposed scheme is acceptable.

表 3　Structure- and value-based GROUP BY ROLLUP.
Table 3　Structure- and value-based GROUP BY ROLLUP.

region	continent	payment	quantity
/site/regions	/site/regions/africa	Cash	9
/site/regions	/site/regions/africa	Creditcard	4
/site/regions	/site/regions/africa	Money order, Cash	6
/site/regions	/site/regions/africa	Money order, Personal Check, Creditcard, Cash	3
…	…	…	…
/site/regions	/site/regions/africa		56
/site/regions	/site/regions/asia	Cash	11
/site/regions	/site/regions/asia	Money order, Cash	11
…	…	…	…
/site/regions	/site/regions/asia		210
…	…	…	…
/site/regions			2244

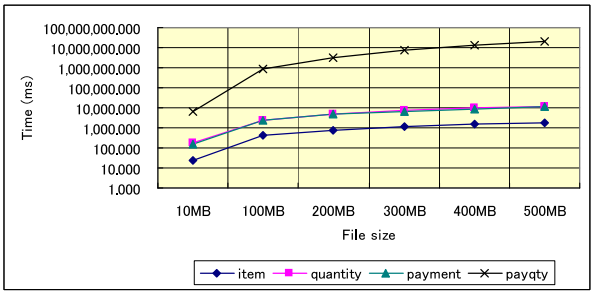

図 6　Table construction time.
Fig. 6　Table construction time.

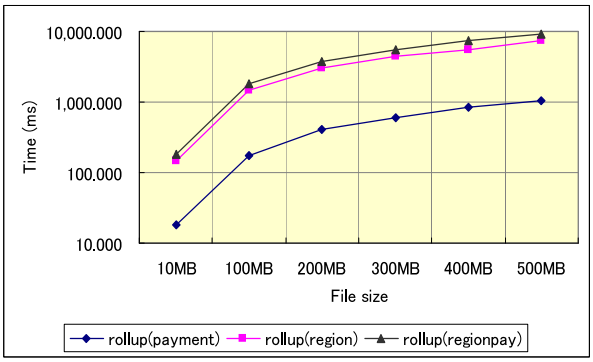

図 7　Query processing time.
Fig. 7　Query processing time.

7.　Conclusions

In this paper, we proposed a system for XML-OLAP which is constructed on top of relational databases. Our system supports both value- and structure-based hierarchy which enable users to make analysis of XML data taking account of features of XML data. We first introduced the concepts of fact path, dimension path, value- and

表 4　Elapse time for table construction (ms).

Table 4　Elapse time for table construction (ms).

table	10MB	100MB	200MB	300MB	400MB	500MB
item	23.914	405.430	796.759	1,218.629	1,578.560	1,813.991
quantity	170.968	2,319.777	4,741.477	7,077.531	9,478.404	11,491.323
payment	163.619	2,258.920	4,644.312	6,937.110	9,276.335	11,248.676
payqty	6,375.812	854,111.265	3,369,312.635	7,675,975.262	13,089,110.031	20,442,085.354

表 5　Elapse time for query processing (ms).

Table 5　Elapse time for query processing (ms).

table	10MB	100MB	200MB	300MB	400MB	500MB
rollup(payment)	18.383	176.208	411.081	611.993	839.870	1,048.303
rollup(region)	148.337	1460.107	2,984.948	4,457.116	5,441.833	7,446.223
rollup(regionpay)	180.320	1,809.195	3,721.951	5,464.89	7,340.956	9,347.355

structure-based concept hierarchy, and XML data-cube. We then discussed OLAP extension to XQuery. For the implementation issues, we use the path approach for mapping XML data to relations, and we utilize UNION ALL to perform GROUP BY ROLLUP operation for both structure- and value-based groupings. Our experiments with large collections of XML data show that the GROUP BY ROLLUP queries perform less than 10 sec. for 500MB XML data. The results show the effectiveness of our proposed technique.

For the future research, we try to improve the performance of data-cube construction. We also plan to investigate how to incorporate textual features such as word vectors of XML data into the analytical processing.

Acknowledgments

This research is partly supported by the Grant-in-Aid for Scientific Research (17700110) from Japan Society for the Promotion of Science (JSPS), Japan, and the Grant-in-Aid for Scientific Research on Priority Areas (18049005) from the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan.

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] Chantola Kit, Toshiyuki Amagasa, and Hiroyuki Kitagawa. Towards Analytical Processing of XML Data. In *DBWS 2006*, 2006.

[3] World Wide Web consortium: XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/1999/REC-xpath-19991116. W3C Recommendation 16 November 1999.

[4] XQuery: A query language for XML, http://www.w3.org/TR/xquery. W3C working draft 2001.

[5] Rajesh Bordawakar and Christian A. Lang. Analytical Processing of XML Documents: Opportunities and Challenges. *SIGMOD Record*, 34(2):27–32, 2005.

[6] Mikael R. Jensen, Thomas H. Moller, and Torben Bach Pedersen. Specifying OLAP Cubes on XML Data. *SSDBM*, pages 101–112, 2001.

[7] Dennis Pedersen, Karsten Riis, and Torben Bach Perdersen. XML-Extended OLAP Querying. *SSDBM*, pages 195–206, 2002.

[8] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, and Shunsuke Uemura. XRel: A path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology (TOIT)*, 1(1):110–141, 2001.

[9] David DeHaan, David Toman, Mariano P.Consens, and M. Tarmer Ozsu. A Comprehensive XQuery to SQL Translation Using Dynamic Interval Encoding. *Proc. ACM SIGMOD 2003*, pages 623–634, 2003.

付　　録

1. Fact: Books from Sales Data

```
CREATE TABLE fact AS SELECT p1.pexp, n1.*,
   n2.value AS jkey
FROM ptable p1, ptable p2, ntable n1,
   ntable n2
WHERE n1.did = n2.did
AND n1.pid = p1.pid
AND p1.pexp LIKE '/sales/%/b/%'
AND n1.tname LIKE 'b'
AND n2.pid = p2.pid
AND p2.pexp LIKE '/sales/%/b/t'
AND n2.tname LIKE '#TEXT'
AND n2.nnum LIKE n1.nnum || '%';
```

2. Dimension: Book Price from Book Category

```
CREATE TABLE dim AS SELECT p2.pexp, n2.*,
   n3.value as jkey
FROM ptable p1, ptable p2, ptable p3,
   ntable n1, ntable n2, ntable n3
WHERE n1.did = n2.did
AND n1.did = n3.did
AND n1.pid = p1.pid
AND p1.pexp LIKE '/bookinfo/%/b'
AND n1.tname LIKE 'b'
AND n2.pid = p2.pid
AND p2.pexp LIKE '/bookinfo/%/b/p'
AND n2.tname LIKE '#TEXT'
AND n3.pid = p3.pid
AND p3.pexp LIKE '/bookinfo/%/b/t'
AND n3.tname LIKE '#TEXT'
AND n2.nnum LIKE n1.nnum || '%'
AND n3.nnum LIKE n1.nnum || '%';
```

3. Sales XML Data-cube

```
SELECT *
FROM fact f, dim d
WHERE f.jkey = d.jkey;
```