Storage and Retrieval of XML Documents using Object-Relational Databases

Takeyuki Shimura, Masatoshi Yoshikawa and Shunsuke Uemura

Graduate School of Information Science Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0101, Japan
{takeyu-s, yosikawa, uemura}@is.aist-nara.ac.jp

Abstract. This paper describes general storage and retrieval methods for XML documents using object-relational databases. The storage method decomposes tree structure of XML documents into nodes, and stores them in relational tables according to the node types. By using this method, being independent of DTDs or element types, any XML documents can be stored in databases. Also it is possible to utilize index structures(e.g. B^+ trees, R trees, etc.) which are provided in database management systems. As for retrieval, we show the transformation of XQL queries into SQL queries. It is possible to realize the storage method by doing minimal extension to object-relational databases and the retrieval method by adding a preprocessor of a query language. We also performed experiments using XML documents on the plays of Shakespeare, to show the effectiveness of our methods.

1 Introduction

XML (eXtensible Markup Language) [WWWC98a], designed as a subset of SGML[ISO86] and recommended by W3C (World Wide Web Consortium), is a document description metalanguage to represent data and documents on the World Wide Web. The potential of XML is unlimited, and many new applications using XML are currently planned (e.g. [Bos97]). Therefore, efficient storage and retrieval of XML documents in databases is an important research issue. With the increase of sophisticated XML documents, databases managing XML documents are required to support queries on structure, content, and attributes[1].

In this paper, we propose general storage and retrieval methods for XML documents using object-relational databases. As databases managing XML documents, we adopted commonly-used object-relational databases which have functionality of adding abstract data types. Figure 1 shows the logical architecture of our system. The differences between our methods and related work are as follows. Firstly, database schemas for storing XML documents are independent of DTDs or element types. Secondly, in retrieving XML documents, the system rewrites declarative queries for XML into SQL queries which are executed in object-relational databases. Also, it is possible to store XML documents by doing minimal extension to object-relational databases and the retrieval method by adding a preprocessor of a query language.

[1] The term ‘attribute’ is used differently in the context of databases and in XML. We call the former ‘database attribute’ and the latter ‘attribute’.

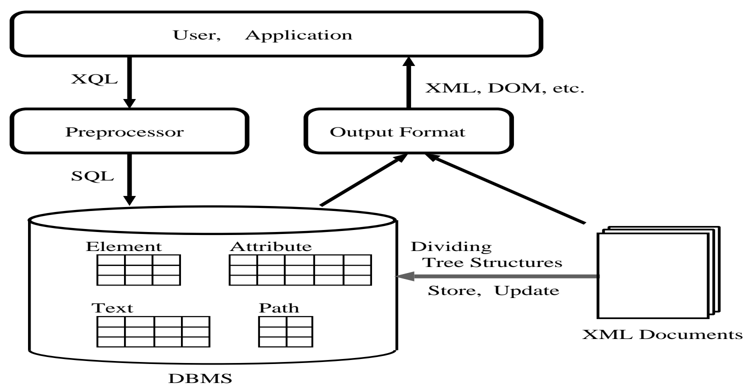

Fig. 1. The logical architecture of our system.

1.1 Related Work

Storage of structured documents There are two major approach to the storage and retrieval of XML documents in databases. XML documents are regarded as structured data in one approach, and as simple character string in another approach.

When structured documents are regarded as structured data, the tree structure representing an XML document is mapped to database schemas. In this approach, database schemas are designed in accordance with the DTD of structured documents. Once such database schema is designed, XML documents stored in databases are guaranteed to conform to the DTD. However, because commonly-used database models (such as relational model and object-oriented model) is not powerful enough to naturally represent the concept of "choice" in element type declarations in the DTD, database schema can not represent the DTD. In [CACS94], a mapping of DTDs to extended database schemas was proposed. However, this approach has a drawback in that (even a small) change of logical structure of XML documents influence on database schemas.

In our approach, since database schemas are independent of DTDs or element types, changes in logical structure do not influence on database schemas. Also, our storage method does not require extended facilities on database models nor database schemas. Also, conformance of XML documents to DTDs, if any, are not guaranteed by DBMSs but by XML processors. Validation check by an XML processor is executed when documents are inserted or modified.

When structured documents are regarded simply as character strings, an XML document is stored in a database attribute. Operations on tree structure are replaced by string operators, and abstract data types which have functions to execute string operators are added to databases. Under this approach, queries on structured documents are described in extended SQL[BCD+95]. Integration engine internally distributes queries to SQL on relational data and to commands on full-text system, and integrates the both query result before they are returned to users or applications. The system provides an interface through which users can view documents as if they were stored in database systems.

Though we also regard structured documents as simple character string, substance of XML documents is stored in databases. We regarded generality as an important design factor. In our approach, we realize storage and retrieval of XML documents using only database management systems.

Index Scheme for structured documents Index schemes for structured documents are described in [SDDTZ97]. The paper presents position-based indexing and path-based indexing to access document collections by content, structure, or attributes. In position-based indexing, queries are processed by manipulating ranges of offsets of words, elements or attributes. In path-based indexing, the paths in tree structures are used. Our storage method of XML documents adopts both of the two indexing schemes and enjoys the advantages of them.

The rest of the paper is organized as follows. Section 2 describes a storage method for XML documents. Section 3 describes a retrieval method for XML documents stored in databases. Section 4 reports the experimental results and demonstrates the effectiveness of our method. We conclude the paper in Section 5.

2 Storage Method of XML Documents

XML processors guarantee that XML documents stored in databases follow tagging rules prescribed in XML or conform to a DTD. Hence, XML documents stored in databases are valid or well-formed.

2.1 A Tree Structure Representing an XML Document

An XML document can be represented as a tree, and node types in the tree are of the following three kinds: Element, Attribute and Text. These node types are equivalent to the node types in XSL[WWWC98b] data model. Though there are other node types such as comment, processing instruction, etc, we do not treat them in this paper.

- Nodes of type **Element** have an element type name as a label. Element nodes have zero or more children. The type of each child node is of one of the three (Element, Attribute and Text).
- Nodes of type **Attribute** have an attribute name and an attribute value as a label. Attribute nodes have no child node. If there are plural attributes, the order of the attributes is not distinguished. This is because there is no order in XML attributes.
- Nodes of type **Text** have character data specified in the XML Recommendation as a label. Text nodes have no child node.

Figure 3 shows the tree structure representing the XML document in Figure 2.

2.2 Design Strategies for Storing XML Documents

We have the following policies for the storage of XML documents:

- Database schemas should not depend on DTDs or element types, and databases shall store any XML documents.
- Index structures which are provided in database management systems shall be used.

```
<books>
<book style="textbook">
<title>Designing XML applications</title>
<editor>
  <family>Bob</family> <given>Kraft</given>
</editor>
<author>
  <family>Nick</family> <given>Marcus</given>
  <family>Bob</family>  <given>Pant</given>
</author>
<summary>
This book is the guide to design<keyword>XML</keyword>applications.
</summary>
</book>
</books>
```

Fig. 2. An example of an XML instance.

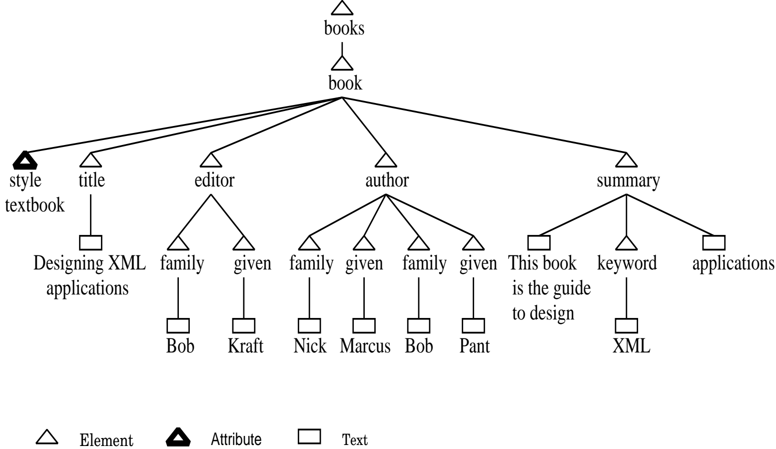

Fig. 3. An example of tree representation.

- Storage method shall be realized by doing minimal extension to object-relational databases.
- Functionalities of XML query languages shall be supported.

As for the storage of XML documents, the key issue is the mapping from the tree structure of an XML document to tuples in relational tables. We decompose the tree structure into relations so that we can easily access and reuse by the unit of logical structure and we can use index structures(e.g. B^+ trees, R trees, etc.) provided in database systems.

Regarding query languages for XML, much discussion have been made on the requirements for languages (e.g. [Wor98]). So far, only a few XML query

```
<SimpleAbsolutePathUnit> ::= <PathOp> <SimplePathUnit> |
                             <PathOp> <SimplePathUnit> '@' <AttName>
<PathOp>                 ::= '/'
<SimplePathUnit>         ::= <ElementType> |
                             <ElementType> <PathOp> <SimplePathUnit>
```

Fig. 4. The syntax of 'SimpleAbsolutePathUnit' stored in databases.

languages including XQL[RLS98] and XML-QL[DFF+98] have been proposed. XQL is a natural extension to the XSL pattern syntax, and it provides a concise, understandable notation for pointing to specific elements and for searching for nodes with particular characteristics. On the other hand, XML-QL, based on research results on semistructured data, has operations peculiar to data manipulation such as joins and supports transformations of XML data. XML-QL integrates information extraction in the WHERE clause and transformation or restructuring in the CONSTRUCT clause. XQL considers that transformation operation is separated from the query language. As for transformation, for example, [SLR98] uses XQL within XSL. In their approach, an XQL query is performed first, then the results of its XQL query are fed into XSL to perform transformations.

An XQL basic query is represented by a line command which connects path operators ('/' or '//') with element types. '/' is the child operator which selects from immediate child nodes. '//' is the descendant operator which selects from arbitrary descendant nodes. Furthermore, the symbol '@' precedes attribute names. By using these notations, all paths of tree representation can be expressed by element types, attributes, '/' and '@'. Strictly speaking, paths can be expressed by *SimpleAbsolutePathUnit* defined in Figure 4. In this paper, we call SimpleAbsolutePathUnit *simple path*.

Basically, we decompose XML documents into simple paths, and store them in databases. However, using only simple paths, retrieval allowing for the hierarchy or order within a document can not be handled. Therefore, we retain for each node, simple path and a pair of positions of the node within the document. Such pair is usually called a *region*(i.e. a pair of a start position and an end position). Because of this mechanism, functionalities of XML query language can be supported properly. Also, the inclusion relationship and the order relationship between nodes can be maintained.

As an example, Figure 5 shows simple paths, regions, and occurrence order information on node type Element, which are derived from the tree structure in Figure 3. In Figure 5, the element types in node type Element and the attribute names in node type Attribute are absorbed into the simple paths. Moreover, nodes of type Element are assigned to order information, which represent occurrence order within sibling nodes having the same parent node. Occurrence order information is composed of occurrence plus order information, and occurrence minus order information. Occurrence plus (or minus) order information is the index number of the node within the parent node. The indexes are zero-based, so 0 represents the first element node. The occurrence order information is used

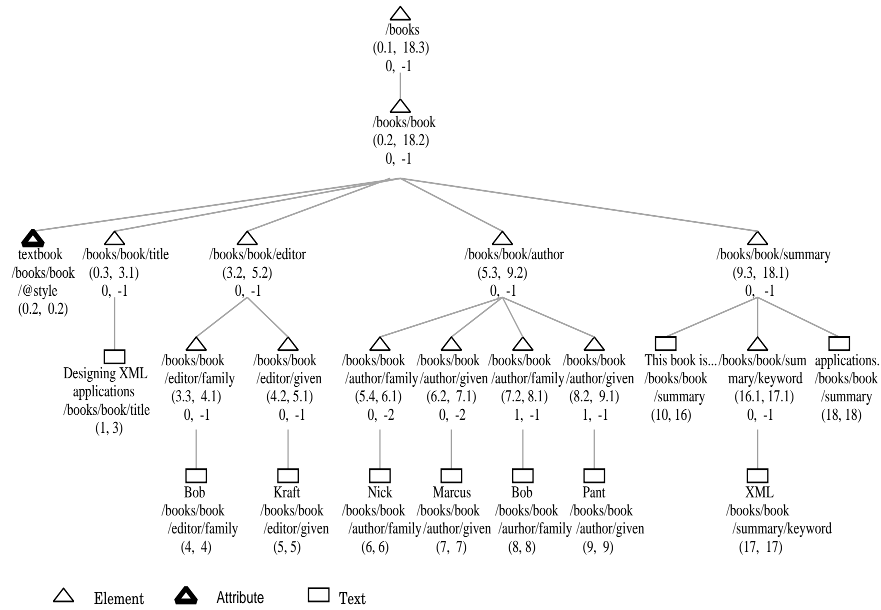

Fig. 5. The simple path and the region of each node derived from the tree structure in Figure 3.

to support the index functions in XQL.

Because there are many ways to assign the region of each node, we will not assume specific assignment. In this paper, we assign the region of each node as follows. Each word occurrence is assigned an integer number corresponding to its position within the document. Each tag is assigned a real number. Its integer part indicates the position number of the preceding word and decimal part indicates the position of the tag being concerned in the current sequence of tags. The reason because tags are not assigned a position is so that they do not interfere with proximity searches on words[SDDTZ97]. In general, regions of two nodes may have the inclusion relationship but they do not have the overlap relationship.

Decomposition of XML documents into simple paths, type classification and computing region of each node are executed when XML documents are parsed using XML processor. Next we will show how data in Figure 5 are stored in relational tables.

2.3 Addition of Abstract Data Type

Abstract data type to be defined is the only type which manages region(positions) of each node type within a document. An instance of REGION type keeps positions, which are a pair of numerical values (r, s) representing a start position and an end position, where $0 < r \leq s$. REGION type can use following two predicates.

– BOOLEAN contain(REGION *pos*)
  This predicate takes an instance *pos* $(r_a,\ s_a)$ of REGION type as its argu-

ment, returns TRUE if and only if (r, s) contains $(r_a,\ s_a)$.

– BOOLEAN precede(REGION *pos*)
This predicate takes an instance *pos* $(r_a,\ s_a)$ of REGION type as its argument, returns TRUE if and only if (r, s) precedes $(r_a,\ s_a)$.

These predicates are used to decide the inclusion relationship or the order relationship of regions within same document.

2.4 Relational Database Schemas for Storing XML Documents

Relations for storing XML documents are four kinds : Element, Attribute, Text, and Path. The relations Element, Attribute and Text store data about each node type described in Section 2.1. The relation Path stores data about simple paths. Each relation has the following database attributes.

– The relation Element stores data about Element nodes. Database attributes are *docID*, *pathID*, *index*, *reindex* and *pos* to store document identifiers, path identifiers, plus occurrence order, minus occurrence order and regions respectively.
– The relation Attribute stores data about Attribute nodes. Database attribute are *docID*, *pathID*, *attvalue* and *pos* to store document identifiers, path identifiers, attribute values and regions respectively.
– The relation Text stores data about Text nodes. Database attribute are *docID*, *pathID*, *value*, and *pos* to store document identifiers, path identifiers, collections of character data and regions respectively.
– The relation Path stores data about simple paths. Database attribute are *pathexp* and *pathID* to store simple paths and path identifiers respectively.

2.5 Storage of XML Documents to Relational Tables

Data about each node described in Section 2.1 is stored, being based on database schema described in Section 2.4. For example, Figure 6 shows that the tree structure in Figure 3 is stored in relational tables. In many XML documents stored in database, if plural XML documents follow the same DTD, there are many same simple paths. Therefore, by storing the correspondence between pathID and simple path in the relation Path, the number of tuples can be reduced. Each occurrence of database attribute *pathexp* in relation Path is subject to simple path specified in Figure 4.

Element

docID	*pathID*	*index*	*reindex*	*pos*
1	1	0	-1	0.1, 18.3
1	2	0	-1	0.2, 18.2
1	4	0	-1	0.3, 3.1
1	5	0	-1	3.2, 5.2
1	6	0	-1	3.3, 4.1
1	7	0	-1	4.2, 5.1
1	8	0	-1	5.3, 9.2
1	9	0	-2	5.4, 6.1
1	10	0	-2	6.2, 7.1
1	9	1	-1	7.2, 8.1
1	10	1	-1	8.2, 9.1
1	11	0	-1	9.3, 18.1
1	12	0	-1	16.1, 17.1

Attribute

docID	*pathID*	*attvalue*	*pos*
1	3	textbook	0.2, 0.2

Text

docID	*pathID*	*value*	*pos*
1	4	Designing XML applications	1, 3
1	6	Bob	4, 4
1	7	Kraft	5, 5
1	9	Nick	6, 6
1	10	Marcus	7, 7
1	9	Bob	8, 8
1	10	Pant	9, 9
1	11	This book is ...	10, 16
1	12	XML	17, 17
1	11	applications.	18, 18

Path

pathexp	*pathID*
/books	1
/books/book	2
/books/book/@style	3
/books/book/title	4
/books/book/editor	5
/books/book/editor/family	6
/books/book/editor/given	7
/books/book/author	8
/books/book/author/family	9
/books/book/author/given	10
/books/book/summary	11
/books/book/summary/keyword	12

Fig. 6. A storage exmple of XML documents.

If tree structures are stored in the relational tables in Figure 6, the source XML documents can be rebuilt because of preserving document identifier and region of each node type.

By dividing tree structures into nodes and storing them according to the node types, we enjoy the following advantages.

- Database schemas to store XML documents do not depend on DTDs or element types. Any XML documents can be managed, being based on the four relational tables.
- Index structures provided in database management systems can be used. B^+ trees on database attributes other than database attributes *pos* and R(or R^*) trees on database attributes *pos* can be constructed. By constructing index structures, queries for XML documents can be efficiently processed.
- It is possible to realize our storage method by doing minimal extension to object-relational databases. The abstract data type which is added to database systems is only REGION type described in Section 2.3. Predicates

of REGION type can decide the inclusion relationship or the order relationship. If this abstract data type is not added, by using simple comparison predicates such as $<$ or $\geq$, we can carry out equivalent operation to predicates of REGION type. Therefore, the described storage method can apply to not only object-relational databases but also relational databases.

3 Retrieval Method of XML Documents

In our architecture, XML documents are decomposed into paths of their tree representation, and stored in the four relations in Figure 6. Their relational tables, in which XML documents are stored, are hidden from users or applications. Users or application consider XML documents as trees, and they specify queries in XML query languages. In this paper, we employ XQL as such an XML query language. In this section, we describe a framework to rewrite XQL queries into SQL queries. However, the query rewriting in detail is omitted due to the limitation of space.

XML documents are decomposed into fragments and they are stored in relational tables. Therefore, identification of sub-documents (i.e. a set of document identifier and region) is expected to be efficient using such tables. However, rebuilding entire documents or large sub-documents from fragments in tables will be inefficient. Hence, when such (sub-)documents are required, we take an approach to scan XML document files.

3.1 Query Rewriting

Using a notation which connects path operators('/' or '//') with element types, etc., XQL can extract sub-documents enclosed with elements. '/' is the child operator which selects from immediate child nodes. '//' is the descendant operator which selects from arbitrary descendant nodes. The '//' can be thought of as a substitute for one or mode levels of hierarchy. Also, in the query, filer clause '[ ]' which is analogous to the SQL WHERE clause, indexing which is easy to find a specific node within a set of nodes, etc. can be specified.

Since data about XML documents such as simple paths are stored as string in databases, functions of pattern matching in SQL-92[DD93] can be used. For example, as basic queries, if XQL queries do not include filer nor indexing, the outline of generating SQL queries is as follows:

(1) Simple paths stored in databases start with path operator '/', and they connect element type with '/'. If XQL queries include path operator '//', every occurrence of '//' in simple paths is replaced with '%/' by using LIKE predicate in the WHERE clause. Then, using the replaced simple paths, pathIDs are selected out from the relation Path.
(2) Pairs of docID and pos in the relation Element are retrieved based on each pathID obtained in (1).

As an example, Query 1 shows that an XQL query which connects element type with path operator can be rewritten into SQL.

Next, Query 2 shows that an XQL query which has a filter is rewritten into SQL. If condition about text is specified in filter, as for rewriting, a query in SQL can be produced by adding relation Text in the FROM clause and condition of

text in the WHERE clause. In the WHERE clause, *pos* is REGION type and one of predicates described in Section 2.3 is used.

Furthermore, XQL queries having indexing can also be transformed into SQL queries by using database attribute *index* or *reindex*. We give such an example in Query 3.

```
Query 1:   /books//author
  SELECT    e1.docID, e1.pos
  FROM      Element e1, Path p1
  WHERE     e1.pathID = p1.pathID
  AND       p1.pathexp LIKE
              '/books\%/author'
  ORDER BY  e1.docID, e1.pos

Query 3:  //book/author/family[0]
  SELECT   e1.docID, e1.pos
  FROM     Element e1, Path p1
  WHERE    e1.pathID = p1.pathID
  AND      p1.pathexp LIKE
             '%/book/author/family
  AND      e1.index = 0
  ORDER BY e1.docID, e1.pos
```

```
Query 2:  //book[summary/keyword = 'XML']
  SELECT   e3.docID, e3.pos
  FROM     Element e1, Path p1, Text t2,
             Path p2, Element e3, Path p3
  WHERE    p1.pathexp LIKE '\%/book
  AND      p2.pathexp LIKE
             '\%/book/summary/keyword
  AND      p3.pathexp LIKE
             '\%/book/author/family
  AND      e1.pathID = p1.pathID
  AND      t2.value = 'XML'
  AND      t2.pathID = p2.pathID
  AND      e3.pathID = p3.pathID
  AND      e1.pos.contain(t2.pos)
  AND      e1.docID  = t2.docID
  AND      e1.pos.contain(e3.pos)
  AND      e1.docID  = e3.docID
  ORDER BY e3.docID, e3.pos
```

4 Implementation

We have performed experiments to store XML documents in a database and retrieve them based on the methods described in Sections 2 and 3. This section describes the implementation and shows the experimental results.

We have used PostgreSQL[POS] which is freely available as an object-relational database. PostgreSQL supports B^+ tree and R tree index structures, as well as user-defined types and functions. As an XML processor, we have used XML Parser for Java[IBM98] which is freely available. XML Parser for Java is a validating XML processor, and it supports SAX(The Simple API for XML)[Meg98]. The module to obtain regions of nodes and path expressions is implemented using SAX, which is an interface for event-based XML parsing. Also, the module to rewrite XQL into SQL is coded in C language.

4.1 The Result of Experiments

We ran some experiments using actual XML documents to see executing. The XML documents used for the experiments is the collection of the plays of Shakespeare, documents[2] tagged by Jon Bosak. These data is summarized in Table 1 and 2. The number of tuples in the Relations “Element”, “Attribute”, “Text” and “Path” is 179,618, 0, 147,525 and 57 respectively. Data size required in s-toring test data are larger than that of source data(Table 2). However, since the price of disk is sharply decreasing, and

[2] <URL:http://sunsite.unc.edu/pub/sun-info/xml/eg/shakespeare.1.10.xml.zip>

Table 1. Details of test data

Item	Number or Data Size
Total XML documents Size	7.65 Mbytes
Number of Documents	37
Average Document Size	206.71 Kbytes

Table 2. Data Size required in storing test data

Item	Data Size
Relation Element	4.10 Mbytes
Relation Attribute	0 bytes
Relation Text	7.32 Mbytes
Relation Path	1.5 Kbytes
Total	11.42 Mbytes

Table 3. Processing time for sample XQL queries and number of their query results

Sample XML query	Time 1 (sec)	Time 2 (sec)	Time 3 (sec)	Number of results
/PLAY	0.15	0.01	0.26	37
/PLAY/ACT	0.18	0.01	0.29	185
/PLAY/ACT[index() = 2]	0.16	0.01	0.35	37
/PLAY/ACT[-3]	0.18	0.01	0.31	37
/PLAY/ACT/TITLE	0.19	0.01	0.33	185
//SCENE/TITLE	0.34	6.52	0.44	750
/PLAY/ACT//TITLE	0.35	3.15	0.34	951
//ACT//TITLE	0.37	6.38	0.04	951
/PLAY/ACT/SCENE/SPEECH[SPEAKER='CURIO']	1.30	7.04	0.87	4
//ACT[//SPEECH/SPEAKER='CURIO']	1.04	8.96	0.53	4

since documents are much smaller in size than multimedia data such as video and audio, we believe that increase of data size in storing XML documents is not a big problem. The machine we used is Ultra Sparc II 360MHz with 640MB memory. Database-server and client are on this machine, and transmission of data uses socket of UNIX domain.

Table 3 shows the time required for processing some XQL queries using our system and two other systems. Time 1, 2 and 3 indicates the processing time of our system, the XQL module implemented by DataChannel[DM99], and Sgrep[JP98] which can realize similar retrieval functions to XQL, respectively. Time 1 is the total time of connecting database-server from a client, rewriting XQL into SQL, sending the rewritten query, fetching the query results and cutting connection. Measurement of processing time is the average of ten trials. In measuring Time 2, all 37 XML documents are parsed, and then 150MB data is retained on main-memory of the server. As for Time 3, XQL queries are executed after constructing index structure peculiar to Sgrep.

5 Conclusions

We have proposed general storage and retrieval methods for XML documents using object-relational databases. Described storage method can apply to not

only object-relational databases but also relational databases, and store XM-L documents having any document structure. As for retrieval, we have shown methods to rewrite XQL into SQL.

Further extensions to the storage and retrieval of XML documents under considerations include storage methods considering data types, corresponding to XQL extensions, integration XML data with other data stored in databases.

References

[BCD+95] G. E. Blake, M. P. Consens, I. J. Davis, P. Kilpeläinen, E. Kuikka, P. - Å. Larson, T. Snider, and F. W. Tompa. Text / relational database management systems: Overview and proposed sql extensions. Technical Report CS-95-25, UW Centre for the New OED and Text Research, Department of Computer Science, University of Waterloo, June 1995.

[Bos97] Jon Bosak. XML, Java, and the future of the Web, March 1997. http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.html.

[CACS94] Vassilis Christophides, Serge Abiteboul, Sophie Cluet, and Michel Scholl. From structured documents to novel query facilities. In *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 313–324, May 1994.

[DD93] C. J. Date and Hugh Darwen. *A Guide to The SQL Standard, 3rd ed.* Addison-Wesley, Reading, MA, 1993.

[DM99] DataChannel and Microsoft. DataChannel-Microsoft Java XML Parser (Beta 2) 1. 0. http://www.datachannel.com/xml_resources/developers/, February 1999.

[DFF+98] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. XML-QL : A Query Language for XML, Aug 1998. http://www.w3.org/TR/NOTE-xml-ql/.

[IBM98] IBM Corporation. XML Parser for Java. http://www.alphaworks.ibm.com/, Feb 1998.

[ISO86] ISO 8879: 1986. *Information Processing – Text and Office System – Standard Generalized Markup Language (SGML)*, Oct. 15 1986.

[JP98] Jani Jaakkola and Pekka Kilpelainen. sgrep (structured grep) version 1.92a. http://www.cs.helsinki.fi/ jjaakkol/sgrep.html, December 1998.

[Meg98] Megginson Technologies Ltd. SAX 1.0: The Simple API for XML. http://www.megginson.com/SAX/, May 1998.

[POS] Postgresql home page. http://www.postgresql.org/.

[RLS98] Jonathan Robie, Joe Lapp, and David Schach. XML Query Language (XQL), Sep 1998. http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[SDDTZ97] Ron Sacks-Davis, Tuong Dao, James A. Thom, and Justin Zobel. Indexing documents for queries on structure, content and attributes. In *International Symposium on Digital Media Information Base (DMIB'97)*, Nov. 1997.

[SLR98] David Schach, Joe Lapp, and Jonathan Robie. Querying and Transforming XML. In *Position papers for W3C Query Language Workshop.* 1998. http://www.w3.org/TandS/QL/QL98/pp/query-transform.html.

[Wor98] World Wide Web Consortium. QL'98 - The Query Languages Workshop. http://www.w3.org/TandS/QL/QL98/, December 1998.

[WWWC98a] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/1998/REC-xml-19980210, February 1998. W3C Recommendation 10-February-1998.

[WWWC98b] World Wide Web Consortium. Extensible Style Language(XSL) Working Draft, 12 1998. http://www.w3.org/TR/1998/WD-xsl-19981216.

This article was processed using the LaTeX macro package with LLNCS style