JOURNAL OF KOREA MULTIMEDIA SOCIETY, VOL. 9, NO. 6, JUNE 2006(pp. 700-708)

Storing and Retrieval of Multiversion XML Documents in Relational Databases

Min Jin†

ABSTRACT

In this paper, we propose a method of managing versions of XML documents by using relational databases. Data structures based on relational tables are developed for accommodating versions of XML documents. The structure information, the contents, and changes of the versions are stored in relational tables. Thus, SQL can be exploited in queries such as horizontal queries, vertical queries, and delta queries without parsing the documents. The structure information and contents of all versions are not represented explicitly in the tables, those of certain versions which are called snapshot versions are represented. Other versions are represented indirectly as sequences of operations that are stored in the corresponding tables. The experiment shows the space performance.

Keywords: XML, Version, RDBMS, Snapshot version, Delta

1. INTRODUCTION

XML is becoming a de facto standard for representing data in Internet data processing environments. The volume of XML documents used in Internet data processing environments is getting larger and these documents are evolving continuously. This fact gives rise to the issue of version control of XML documents. XML data can be stored in file systems, object-oriented database systems, special purpose XML storage systems, or relational database systems. Relational databases are widely used in conventional data processing and provide inherent well developed services such as transaction management and query optimization that might be exploited in managing XML data. Hence, lots of research have been done in using relational databases for storing XML documents[7,8,11-16,20]. It gives rise to research in managing versions of XML documents stored in relational databases.

There have been research on managing versions of XML documents[2,4,6,18,19]. There are a few issues to be addressed in managing versions of XML documents. The first issue is the representation of each version in the evolution environment. The history of changes to an element should be also traceable. The second issue is the efficiency of storage. When all versions of XML documents are stored separately, surmountable data can be duplicated. Thus it leads to the waste of storage. In general, there is a trade-off between storage space and access time in managing versions. The third issue is the processing of queries against versions of XML documents.

In this paper, we propose a method for managing versions of XML documents by using RDBMS. There is a fundamental issue to be addressed in managing XML documents via RDBMS due to the discrepancy between hierarchical structure of XML and flat structure of relational databases [12,13]. Special data structures based on relational tables are developed for accommodating versions of XML

※ Corresponding Author : Min Jin, Address : (631-701) 449 Wolyoung-dong, Masan, Kyungnam, S. Korea, TEL : +82-55-249-2653, FAX : +82-55-248-2554, E-mail : mjin@kyungnam.ac.kr

Receipt date: Feb. 22, 2006, Approval date : June 29, 2006

† Div. of Computer Engineering, Kyungnam University

※ This work was supported by Kyungnam University research fund.

documents. We propose a novel method for representing versions of XML documents, in which *k-distance* versions are explicitly represented, other versions are represented as sequences of operations stored in the corresponding tables. The *k-distance* could be determined by the rate of accumulated changes or fixed number. Other versions can be constructed by applying the sequence of operations on the appropriate snapshot version. This method is a compromise solution to version control between explicit and implicit method. All versions are explicitly represented in the explicit method, whereas most of versions are not represented explicitly in the implicit method like in RCS[17] and SCCS[9]. There are two ways for representing versions in the explicit method. All versions are represented in a single space like the method in [1]. In the other way, each version is represented separately. The former method uses less storage space than the second. However, cost of version construction in the first method is higher than that in the second method.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 shows the data structures for representing versions of XML documents in relational tables. Section 4 describes our scheme for managing versions of XML documents in relational databases and discusses the performance. Section 5 offers conclusions.

2. RELATED WORK

Traditional version control schemes such as RCS[17] and SCCS[9] have been developed for managing versions of texts. In RCS, the most current version is represented intact while the previous versions are stored as reverse editing scripts. In SCCS, a pair of timestamps is associated with each segment of the text and represents the lifespan of the segment. Although these line-based schemes provide basic intuitions for managing versions of XML documents, they are not appropriate for managing versions of XML documents with hierarchical structures. There are no logical structures represented in both RCS and SCCS, so that it is difficult and expensive to reconstruct the logical structure of a version of XML documents. Hence, schemes for storing and retrieving logical structures of versions of XML documents have been developed[3,4,5,19]. RBVM[5] unifies the logical and physical representations. The versions of XML documents are represented as other XML documents, the DTD of RBVM scheme can be easily derived from the original DTD of documents. Elements for representing reference records are included and an ID attribute is added to each element. The objects unchanged in the new version are represented as references to the objects in the old version. This scheme can handle simple queries rather than complex queries for which different storage representations are needed. A scheme for supporting complex queries and temporal queries have been developed, in which successive versions of an XML document are represented as an XML document called *V-Document*[19]. In this scheme, two attributes *vstart* and *vend* which represent the intervals of valid versions are assigned to each element. The version interval of an ancestor element always contains those of its descendant elements. A technique in which an element appearing in multiple versions is stored once has been developed[1]. The logical structure of each version is preserved by using keys and timestamps. A new version is merged with the existing archive and a certain version is extracted from the archive.

Copy-based UBCC scheme with page usefulness management[3] was introduced geared towards efficient version reconstruction while using small storage overhead. With the notion of *lifespan* which is represented by two timestamps (V_{start}, V_{end}), versions of the XML document are represented[19]. While the performance of RCS degrades poorly as the changes grow larger, the copy-based UBCC

sheme[2] achieves the desirable effect when changes grow larger. When the page usefulness of a certain version gets down below the minimum usefulness, the objects in these pages are copied into a new page. It leads to efficient page retrieval in the construction of versions. It also supports rearrangement and duplication of parts of contents unlike the edit-based UBCC[3]. Editing scripts are stored separately from document objects in this scheme whereas they are stored together in RCS scheme, in which the size of scripts gets large and version construction time increases as time goes.

3. RELATIONAL TABLES FOR REPRESENTING VERSIONS OF XML DOCUMENTS

Version control has been an important issue in conventional text and software management. It is also an important issue in managing XML documents. As we have seen in section 2, several methods for managing versions of XML documents have been developed[1,2,5,19]. In this paper, we propose a method for managing versions of XML documents using RDBMS in order to exploit the benefits of it in managing versions. Unlike the method[1], selected versions are represented separately and others are represented by using the delta technique.

Special tables are devised for accommodating the evolution of XML documents. The *snapshot table* contains the structure information of XML documents of certain versions including the current version. The structural information of every version of the documents is not explicitly represented, instead the structure information of *k-distance* versions which are called snapshot versions are stored explicitly. The original document is assumed to be a snapshot version. The current version of the document is active. The materialized versions which are not snapshot versions originally, however they are materialized during query processing, are stored in separate table called the *materialized table* to be used in further query processing. The modifications on each version of the document in the evolution process are kept in the *delta table*. The *XID table* contains the information of elements and attributes which are inserted and deleted in the versions. The *version derivation table* keeps the information of the hierarchical derivation history of versions.

3.1 XID Table

Every element and attribute is stored in the *XID table*. A unique identifier which is called *XID* is assigned to each of them. The names of them are represented in the *name* column. The V_{start} column represents the version in which the element or attribute is created and the V_{end} column represents the version in which it is eliminated. *Now* means the current version. The *root* column shows whether the element is a root or not. This column is applicable to elements only. The *attribute* column shows whether it is an attribute or not. The *docID* column represents the identification of the document. An element or attribute which is contained in each version is inserted in *XID table*. If it is deleted from a version, it is marked in this table as deleted by putting the version number in V_{end} column of the element or attribute. Figure 1 shows three versions of an XML document. Version 1 which is the original document is a snapshot version and version 3 is also a snapshot version.

3.2 Snapshot Table

The snapshot table contains the structure information of the versions which are chosen snapshot versions. The *version* column identifies the version of which the snapshot is constructed. A version can be specified as a snapshot version by either explicitly or implicitly. The structure order in the version is described in terms of bit structured schema, which is represented in *NSN(node sequence number)* column. *Level* column keeps the

```
Version 1
<book>
 <title> Systematic MS SQL server </title>
 <authors>
   <author>
    <name> Min Jin </name>
    <email> mjin@zeus.kuyngnam.ac.kr </email>
   </author>
 </authors>
 <chap ctitle=Database Concept>
   <sect> Database Definition </sect>
   <sect> Relational Database </sect>
 </chap>
 <chap ctitle=Install MS SQL Server>
   <sect> MS SQL Server Products </sect>
   <sect> MS SQL Server Installation </sect>
   <sect> Check Installation </sect>
 </chap>
 <chap ctitle=MS SQL Server Tools>
   <sect> Service Manager </sect>
   <sect> Enterprise Manager</sect>
 </chap>
 </book>
 Version 2
   ...
 <email> mjin@kuyngnam.ac.kr </email>
 <chap ctitle=Database Concept>
   <sect> Database Definition </sect>
   <sect> Relational Database </sect>
   <sect> Components of Database </sect>
 </chap>
 <chap ctitle=Install MS SQL Server>
   <sect> MS SQL Server Products </sect>
   <sect> MS SQL Server Installation </sect>
 </chap>
    ...
```

```
Version 3
<book>
 <title> Systematic MS SQL server </title>
 <authors>
   <author>
     <name> Min Jin </name>
    <email> mjin@kuyngnam.ac.kr </email>
   </author>
   <author>
    <name> B. J. Shin </name>
    <email> challenger@kyungnam.ac.kr
   </author>
 </authors>
 <chap ctitle=Database Concept>
   <sect> Database Definition </sect>
   <sect> Relational Database </sect>
   <sect> Components of Database
 </chap>
 <chap ctitle=Install MS SQL Server>
   <sect> MS SQL Server Products </sect>
   <sect> MS SQL Server Installation </sect>
 </chap>
 <chap ctitle=MS SQL Server Tools>
   <sect> Service Manager </sect>
   <sect> Enterprise Manager</sect>
   <sect> Query Analyzer</sect>
 </chap>
  </book>
```

Fig. 1. Versions of an XML document.

hierarchical depth of the element or attribute. Although this information is implied in NSN column, it is represented in a separate column. The values of simple elements and attributes are stored in *value* column. For the empty elements, *NULL* is filled in the column. No value is assigned to complex elements. The current active version is represented in the snapshot table, of which version column is set to current. Figure 3 shows an example of a snapshot table.

3.3 Materialized Table

Versions which are not snapshot versions originally, however, are constructed when processing queries are stored in a separate table to be used in further query processing. The materialized table has the following information similar to the snapshot table. Figure 4 shows an example of a materialized table.

The *time* indicates when the version is materialized and the *owner* represents the person who materialized the version.

3.4 Delta Table

The delta table contains modifications of versions of the document. For each modification operation,

docID	XID	Name	Vstart	Vend	Root	Attribute
1	1	book	1	now	1	0
1	2	title	1	now	0	0
1	3	authors	1	now	0	0
1	4	author	1	now	0	0
1	5	name	1	now	0	0
1	6	email	1	now	0	0
1	7	chap	1	now	0	0
1	8	ctitle	1	now	0	1
1	9	sect	1	now	0	0
1	10	sect	1	now	0	0
1	11	chap	1	now	0	0
1	12	ctitle	1	now	0	1
1	13	sect	1	now	0	0
1	14	sect	1	now	0	0
1	15	sect	1	2	0	0
1	16	chap	1	now	0	0
1	17	ctitle	1	now	0	1
1	18	sect	1	now	0	0
1	19	sect	1	now	0	0
1	20	sect	2	now	0	0
1	21	author	3	now	0	0
1	22	name	3	now	0	0
1	23	email	3	now	0	0
1	24	sect	3	now	0	0

Fig. 2. An XID table for versions of the XML document in Figure 1.

the corresponding information is stored in this table. Modifications can be classified into three categories; *insert, delete, and update*. One of which is shown up in the *task* column for each operation. V_{from} represents the version in which the operation occurrs, V_{to}represents the version which is created due to the operation. If V_{from}= V_{to}, the operation occurs within the version, it means the operation does not trigger to create a new version. The current value of the operand is placed in Val_{old} and new value of the operand is stored in Val_{new}. When the operationtriggers to change the order, current order and new order will be placed in Pos_{old} and Pos_{new} respectively. Figure 5 shows an example of delta table.

3.5 Version Derivation Table

There exists at most one parent version for each version. The version derivation hierarchy table has the following information. The snapshot column indicates whether the version is a snapshot version

docID	XID	Version	NSN	Level	Value
1	1	1	1	1	
1	2	1	1.1	2	Systematic MS SQL Server
1	3	1	1.2	2	
1	4	1	1.2.1	3	
1	5	1	1.2.1.1	4	Min Jin
1	6	1	1.2.1.2	4	mjin@zeus.kyungnam.ac.kr
1	7	1	1.3	2	
1	8	1	1.3.1	3	Database Concept
1	9	1	1.3.2	3	Database Definition
1	10	1	1.3.3	3	Relational Database
1	11	1	1.4	2	
1	12	1	1.4.1	3	Install MS SQL Server
1	13	1	1.4.2	3	MS SQL Server Products
1	14	1	1.4.3	3	MS SQL Server Installation
1	15	1	1.4.4	3	Check Installation
1	16	1	1.5	2	
1	17	1	1.5.1	3	MS SQL Server Tools
1	18	1	1.5.2	3	Service Manager
1	19	1	1.5.3	3	Enterprise Manager
1	1	3	1	1	
1	2	3	1.1	2	Step by Step MS SQL Server
1	3	3	1.2	2	
1	4	3	1.2.1	3	
1	5	3	1.2.1.1	4	Min Jin
1	6	3	1.2.1.2	4	mjin@kyungnm.ac.kr
1	21	3	1.2.2	3	
1	22	3	1.2.2.1	4	B. J. Shin
1	23	3	1.2.2.2	4	challenger@kyungnam.ac.kr
...	...	...	..	...	...

Fig. 3. A snapshot table for the versions in Figure 1.

docID	XID	Version	NSN	Level	Value	Time	Owner
1	1	2	1	1		2006-5-15 15:30	1000
1	2	2	1.1	2	Systematic MS SQL Server	2006-5-15 15:30	1000
1	3	2	1.2	2		2006-5-15 15:30	1000
1	4	2	1.2.1	3		2006-5-15 15:30	1000
1	5	2	1.2.1.1	4	Min Jin	2006-5-15 15:30	1000
1	6	2	1.2.1.2	4	mjin@zeus.kyungnam.ac.kr	2006-5-15 15:30	1000
1	7	2	1.3	2		2006-5-15 15:30	1000
1	8	2	1.3.1	3	Database Concept	2006-5-15 15:30	1000
1	9	2	1.3.2	3	Database Definition	2006-5-15 15:30	1000
1	10	2	1.3.3	3	Relational Database	2006-5-15 15:30	1000
1	11	2	1.3.4	3	Components of Database	2006-5-15 15:30	1000
...	...	...	...	...	...	...	...

Fig. 4. An example of a materialized table.

docID	XID	V_{from}	V_{to}	Task	Val_{old}	Val_{new}	Pos_{old}	Pos_{new}
1	6	1	2	update	mjin@zeus.kyungnam.ac.kr	mjin@kyungnam.ac.kr	1.2.1.2	1.2.1.2
1	20	1	2	insert		Components of Database		1.3.4
1	15	1	2	delete			1.4.4	
1	2	2	3	update	Systematic MS SQL Server	Step by Step MS SQL Server	1.1	1.1
1	21	2	3	insert				1.2.2
1	22	2	3	insert		B. J. Shin		1.2.2.1
1	23	2	3	insert		challenger@kyungnam.ac.kr		1.2.2.2
1	24	2	3	insert		Query Analyzer		1.5.4

Fig. 5. A delta table for the versions in Figure 1.

or not. Figure 6 shows an example of aversion derivation table.

docID	Version	Parent	Owner	Time	Snapshot
1	1		1000	2006-2-10 10:15	1
1	2	1	1010	2006-2-20 12:30	0
1	3	2	1000	2006-5-10 14:30	1

Fig. 6. A version derivation table for the versions in Figure 1.

4. MANAGING VERSIONS OF XML DOCUMENTS IN RELATIONAL DATABASES

4.1 Representing Versions

The original XML document is considered as the first version. When modification operations occur, the information should be stored in the corresponding tables. Modifications are classified into three categories. The first one is when modifications occur within a version. The second one is when modifications trigger to create a new version which is not a snapshot version. The third one is when modifications trigger to create a new version which is a snapshot version. For each category, the corresponding information is stored in the delta table depending on the operations as shown in Figure 5.

4.2 Version Construction

The structure information of snapshot versions is stored in the snapshot table explicitly. This information can be exploited in processing queries against these versions. However, queries can be thrown against versions of which structures are not stored explicitly. It is required to construct the structure of non-snapshot versions. The algorithm for version reconstruction is shown in Figure 7.

4.3 Performance

Queries in evolution environments of XML

```
Algorithm VersionConstruct( VNumber ) {
/* Assume the structure of VNumber is required */
/* When the version is a snapshot or exits in the materialized table, the structure is given, */
/* otherwise the structure should be constructed */
if VNumber is a snapshot version
   then retrieve the structure of the version from the snapshot table
   else if VNumber exists in the materialized table
      then retrieve the structure of the version from materialized table
      else begin
           find the latest version which appears in the snapshot or materialized table among its ancestors
           let it be LatestVersion
           find the path from VNumber to LatestVersion and put the versions on the path in VersionList
           /* VersionList contains a list of versions on the path including VNumber */
           CurVersion ← LatestVersion
           /* Apply the operations in the delta table to the corresponding version */
           /* along the version list */
           repeat
              NextVersion ←the earliest version in VersionList
              /* NextVersion is the earliest version in the VersionList */
              Delete NextVersion from the VersionList
              apply the operations on CurVersion to NextVersion in delta table
              CurVersion ←NextVersion
           until NextVersion = VNumber
        end
   /* Store the structure of the version with VNumber in the materialized table */
   store the structure of it in the materialized table
}
```

Fig. 7. An algorithm to construct the structure of a version.

documents can be classified into three categories such as horizontal, vertical, and delta queries. Horizontal queries are on the structures of certain versions, which are called structural queries. Vertical queries which are called change queries are on the changes of elements. The snapshot query belongs to horizontal queries, while evolutionary history and continuous query[17] belong to vertical queries. Unlike the previous work of managing versions of XML documents, SQL can be used in querying the evolution of XML documents in our approach. Our scheme is good at supporting delta queries due to the fact that the information is stored in the delta table.

The storage requirement depends on the rate of changes, c_r, and the rate of snapshot versions, which is the inverse of distance, d, as shown in Fig. 8. The XMark[10] dataset was used in the experiment.

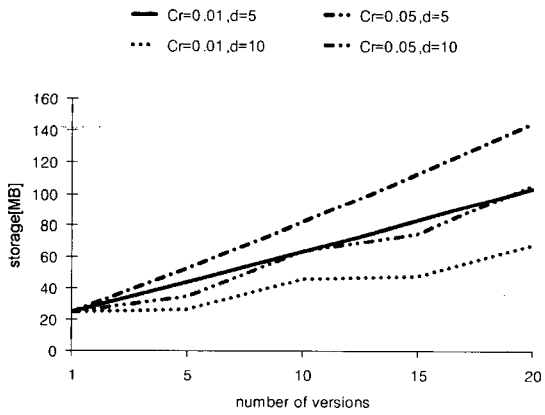

Fig. 8. Storage space performance.

5. CONCLUSION

We have proposed a scheme of managing versions of XML documents using relational databases. The hierarchical information of XML docu-

ments are represented in relational tables and the version information is also stored in the tables. The structural information of XML documents can be retrieved from the tables without parsing XML documents. We can exploit the facilities of RDBMS in managing versions of XML documents. Thus SQL can be used in querying changes of XML documents. All versions of XML documents are not stored explicitly. The *k-distance* versions which are called snapshot versions are explicitly represented, whereas other versions are represented as sequences of operations that are stored in the corresponding tables. Changes made to elements of XML documents during the evolution are also traceable. As the number of versions increases, the requirement of storage space increases in linear form depending on the change rate and the rate of snapshot versions. The benefits of the proposed method will be verified against more general datasets.

6. REFERENCES

[1] P. Buneman, S. Khanna, K. Tajima, W. C. Tan, "Archiving Scientific Data," *Proceedings of ACM SIGMOD*, pp. 1-12, 2002.

[2] S.Y. Chien, V. J. Tsotras, and C. Zaniolo, "Version Management of XML Documents," *Proceedings of International Workshop on the Web and Databases*, pp. 75-80, 2000.

[3] S.Y. Chien, V.J. Tsotras, C. Zaniolo, "Copy-Based versus Edit-Based Version Management Schemes for Structured Documents," *The 11th RIDE-DM Workshop*, pp. 95-102, 2001.

[4] S.Y. Chien, V.J. Tsotras, C. Zaniolo, D. Zhang, "Storing and Querying Multiversion XML Documents using Durable Node Numbers," *Proceedings of 2nd International Conference on Web Information Systems Engineering*, pp. 232-241, 2001.

[5] S.Y. Chien, V.J. Tsotras, C. Zaniolo, "Efficient Management of Multiversion Documents by Object Referencing," *Proceedings of the 27th VLDB*, pp. 291-300, 2001.

[6] S.Y. Chien, V.J. Tsotras, C. Zaniolo, "Efficient Schemes for Managing Multiversion XML Documents," *The VLDB Journal*, pp. 332-353, 2002.

[7] M. Fernandez, Y. Kadiyska, A. Morishima, D. Suciu, W.C. Tan. "A Framework for Publishing Relational Data in XML," *ACM Transactions on Database Systems*, Vol. 27, No. 4, pp. 438-493, 2002.

[8] D. Florescu, D. Kossmann, "Storing and Querying XML Data Using an RDBMS," *IEEE Data Engineering Bulletin*, Vol. 22, No. 3, pp. 27-34, 1999.

[9] M.J. Rochkind, "The Source Control System," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 4, pp. 364-370, 1975.

[10] A.R. Schmidt, F. Waas, M.L. Kersten, D. Florescu, I. Manolescu, M.J Carey, R. Busse. "The XML Benchmark Project," *The Technical report, INS-R0103, CWI*, 2001.

[11] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, J. Funderburk, "Querying XML Views of Relational Data," *Proceedings of the 27th VLDB Conference*, pp. 261-270, 2001.

[12] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, B. Reinwald, "Efficiently Publishing Relational Data as XML Documents," *Proceedings of the 26th VLDB Conference*, pp. 65-76, 2000.

[13] J. Shanmugasundaram, E. Shekita, J. Kiernan, R. Krishnamurthy, E. Viglas, J. Naughton, I. Tatarinov, "A General Technique for Querying XML Documents Using a Relational Database System," *SIGMOD Record*, Vol. 30, No. 3. pp. 20-26, 2001.

[14] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. Dewitt, J. Naughton, "Relational Databases for Querying XML Documents: Limitations and Opportunities," *Proceedings*

of the 25th VLDB Conference, pp. 302-314, 1999.

[15] B.J. Shin, M. Jin, "Storing and Querying XML Documents Using a Path Table in Relational Databases," *The 1st International Workshop on XML Schema and Data Management held in conjunction with ER2003*, pp. 285-296, 2003.

[16] I. Tatarinov, S.D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, C. Zhang, "Storing and Querying Ordered XML Using a Relational Database System," *SIGMOD Conference 2002*, pp. 204-215, 2002.

[17] W.F. Tichy, "RCS-A System for Version Control," *Software-Practice and Experience*, Vol. 15(7), pp. 637-654, 1985.

[18] F. Wang, C. Zaniolo, "Publishing and Querying the Histories of Archived Relational Databases in XML," *WISE 2003*, pp. 93-102, 2003.

[19] F. Wang, C. Zaniolo, "Temporal Queries and Version Management for XML Document Archives," *Journal of Applied Logic, Special Issue on Temporal Reasoning and its Applications*, pp. 1-20, 2004.

[20] C. Zhang, J.F. Naughton, D. DeWitt, Q. Luo, G. Lohman, "On Supporting Containment Queries in Relational Database Management Systems," *Proceedings of ACM SIGMOD Conference*, pp. 425-436, 2001.

Min Jin

He received the B.S degree in computer science and statistics from Seoul National University in 1982, and the M.S degree in computer science from KAIST in 1984, and the Ph.D. degree in computer science and engineering from the University of Connecticut in 1997. He has been working at Kyungnam University since 1985. He is currently a professor in the division of computer engineering. His research interests include data modeling, object-oriented database, version management, and XML storage and processing.