THE PERFORMANCE OF UPDATING XML IN TRADITIONAL DATABASES

Pensri Amornsinlaphachai[*] and Kwanjai Deejring

Received: Oct 4, 2007; Revised: Dec 3, 2007; Accepted: Dec 26, 2007

Abstract

Most researches in the XML area have concentrated on storing, querying and publishing XML, while not many have paid attention to updating XML; thus the XML update area is not fully developed. This work provides the overview of a solution for the update of XML documents via ORDB (Object-Relational Database) to advance the techniques in this area through preserving constraints, maintaining performance in the presence of data redundancy, permitting joins of documents in updates and allowing the updates of documents whose structure is known partially or whose structure is recursive. The main contribution is to compare the performance of the solution and the existing ones. Thus experimental study to evaluate the performance of XML update processing has been conducted. The experimental results show that updating multiple XML documents storing non-redundant data yields a better performance than updating a single XML document storing redundant data; an ORDB can take advantage of this by caching data to a greater extent than a native XML database.

Keywords: XML updates, ORDB, traditional databases, XML constraints

Introduction

XML (eXtensible Markup Language) has become an effective standard for representing semi-structured data on the Web since it provides a natural data structuring mechanism for hierarchical and recursive data; moreover it is flexible in that it allows the authors to define their own tags and structure for documents and can handle data whose occurrence is optional. Many researchers in the XML area have focused on storing, publishing, and querying XML documents. XML consequently provides most of the features normally expected for a database model. However, there is an omission in that most existing work does not pay much attention to modifying XML or does not mention it at all.

Nowadays, there are two dominant approaches for managing XML repositories. The first approach is to use native XML databases to handle the data. The second approach maps XML onto a traditional database (e.g., relational database (RDB), object-relational database (ORDB) and object-oriented database (OODB)). Two possible reasons behind the immaturity of the XML update area are as follows. Firstly, XQuery has not provided update features because the W3C Consortium wanted to release

Computer Department, Faculty of Science and Technology, Nakhon Ratchasima Rajabhat University, Thailand 30000. Tel: 086-1003876; E-mail: kokkoy@hotmail.com
[*] *Corresponding author*

the standard of XQuery as soon as possible (Chamberlin, 2003). Secondly, existing work is focused on updating XML by employing a native XML database. Thus a host of work such as preserving constraints must be created from scratch which can take a long time. Research in the XML update area is not fully-fledged. Our work has identified five main problems as follows:

- The work published presently can update XML documents but only without checking constraints. Even commercial products cannot guarantee the integrity in the database when XML data is updated (Babcock, 2002).
- Normally, all XML data is kept in one document; thus data redundancy may occur. This can lead to data inconsistency and low performance when updates are performed (Arenas and Libkin, 2004).
- No XML update language supports joins of XML documents (Obasanjo and Navathe, 2002; Lu *et al.,* 2003).
- Regular path expressions are used to query/update XML whose structure is unknown or only partially known. Using regular path expressions, especially a descendent path expression (‘//’), can slow the process of querying/updating data (Wang and Liu, 2003) because the query engine must traverse all possible paths in XML.
- In XQuery, there is no specific facility to query data whose structure is recursive; however the effect can be achieved by creating a recursive user-defined function. Until now no technique is proposed to translate this recursive feature into Structured Query Language (SQL) (Krishnamurthy *et al.,* 2004; Prakash *et al.,* 2006).

Our work presents a solution for the update of XML documents via ORDB to solve these problems and evaluates the performance of XML update processing.

The rest of this paper is organized as follows. The next section presents the goal of our work, to devise a more effective solution for updating XML data and solve the open problems as mentioned previously. The following section shows the results of experimental study, including performance aspects. A conclusion is provided in the final section.

Our Solution for Updating XML

XML updating has been relatively well-researched in the area of native XML database, whereas in the area of applying traditional databases to manage XML, only one work (Tatarinov *et al.,* 2001) has presented an XML language for updating XML data. This work employed a RDB but only the syntax and semantics of the language are presented. In our solution, the more advanced technology of ORDB is exploited to update XML documents.

The purpose of using a traditional database, ORDB, in this research is different from that of other work. The previous work uses OODB (Zwol *et al.,* 1999), RDB (Tatarinov *et al.,* 2002; Lv and Yan, 2006) and ORDB (Pardede *et al.,* 2006) as the database management systems (DBMS) of XML documents to store and query XML data, but our approach uses ORDB to preserve constraints during updating and to indicate the target-elements in XML documents which should be updated. The updates are performed on XML documents; thus it is not necessary to maintain the order of elements in ORDB and users can query data from XML documents instead of ORDB. This reduces the cost of data conversion, since nowadays, the major expense of exchanging messages between Web Services comes from converting data such as between a database and XML format (Watson, 2005). The overview of our approach is illustrated in Figure 1.

In the solution, Document Type Definitions (DTDs) are used in our mapping since most XML documents still stick to DTDs (Mignet *et al.,* 2003). Not only the XML structure but also XML constraints are mapped to ORDB since a DTD defines the constraints on the logical structure of XML documents (Lu *et al.,* 2005).

Non-redundant data is kept in separate multiple XML documents so avoiding the storage of redundant data in one single XML document; then the separate documents are linked together. To update XML data, an XML update language, as an extension to XQuery, is designed and this language is translated into SQL to update XML data stored in ORDB. Then the changes in ORDB are propagated to the XML documents.

With this solution, the problems mentioned in the previous section can be solved as follows. Firstly, the preservation of XML constraints is handled by the ORDB engine. Secondly, non-redundant data is stored in linked XML documents; thus the problem of data inconsistency and low performance caused by data redundancy are solved. Thirdly, joins of XML documents are converted to joins of tables in SQL. Fourthly, fields or tables involved in regular path expressions can be tackled in a short time by the use of mapping data. Finally, a recursive function is translated into SQL commands equipped with a programming capability.

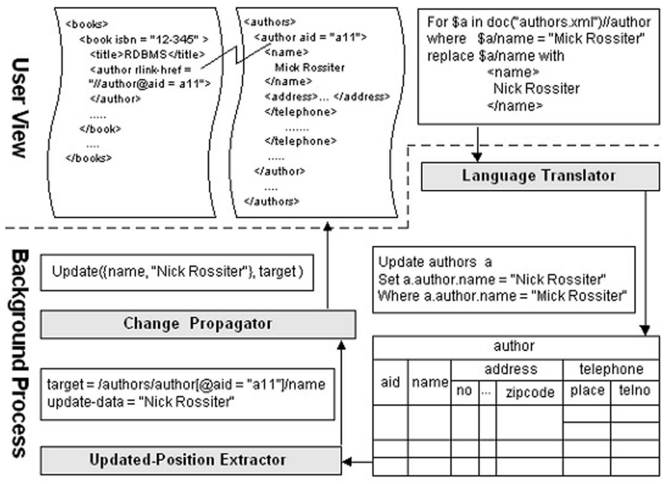
The detail of mapping XML to ORDB is presented in Amornsinlaphachai *et al.*, 2006a whereas the detail of translating the XML update language into SQL is proposed in Amornsinlaphachai *et al.*, 2006b. The techniques of propagating the change from XML to ORDB and the implementation for the solution can be found in Amornsinlaphachai *et al.*, 2005.

Results and Discussion

It is important to verify the method for updating XML, developed in the previous section. This was done through an experimental study in which a diverse range of 17 update queries were executed and the results carefully inspected to check that they were as expected. In addition, to gain an insight into the performance of the update techniques, runs were repeated with variable database size, cache state, degree of redundancy and methods for linking XML structures.

Experiment Platform and Methodology

In the experiments, three types of databases are used. The first is X-Hive, a trial version of a commercial native XML database (nxd), used to keep redundant data of a single XML document. The second is Oracle ORDB employed to keep redundant data of a single XML document (sxd). The third is Oracle ORDB utilised to keep non-redundant data of linked XML documents (lxd). The XML update language for X-Hive is XUpdate (XMLDB, 2002). All experiments are conducted on a 1.3 GHz Pentium M machine with 768 MB main memory and 20 GB disk running Windows XP. The experiments are designed to evaluate the performance for updating: (a) native XML database storing redundant data, (b) ORDB storing redundant XML data and (c) ORDB keeping non-redundant XML data. Varying sizes of the databases ranging from 5, 10, 20 to 40 MB are used.



```
<books>
  <book isbn = "12-345" >
    <title>RDBMS</title>
    <author rlink-href =
    "//author@aid = a11">
    </author>
    .....
  </book>
  ....
</books>
```

```
<authors>
  <author aid = "a11">
    <name>
      Mick Rossiter
    </name>
    <address>... </address>
    </telephone>
      .......
    </telephone>
    .....
  </author>
  ....
</authors>
```

```
For $a in doc("authors.xml")//author
where   $a/name = "Mick Rossiter"
replace $a/name with
          <name>
           Nick Rossiter
          </name>
```

```
Update authors  a
Set a.author.name = "Nick Rossiter"
Where a.author.name = "Mick Rossiter"
```

```
Update({name, "Nick Rossiter"}, target )
```

```
target = /authors/author[@aid = "a11"]/name
update-data = "Nick Rossiter"
```

author						
aid	name	address			telephone	
		no	...	zipcode	place	telno

Figure 1. Overview of the solution to updating XML documents

The number of redundant records for the 5 MB data size varies from 10, 20, 40 to 80 records, while the number of redundant records for the 10, 20, and 40 MB data sizes is two, four and eight times respectively the number of redundant records for the 5 MB data size. The size of linked XML documents is smaller than a single XML document since such documents do not contain redundant records. In updating the linked XML documents, each update command affects 10 records. Thus the number of records in a single document affected by a command varies according to the proportion of redundant records.

To study the effect of data caching on the performance of updating XML, the experiments are conducted in cold cache, warm cache and hot cache. In cold cache, the database is restarted for each individual update command. In warm cache, the database is restarted for each individual command as well; however before running the command, five unrelated commands will be run first. In hot cache, the same command is run twice in succession and the performance measured for the second run.

Another objective is to compare the performance of update features. We designed two sets of 44 update commands for two ORDBs. Each set of commands consists of 14 replace, 14 delete and 16 insert commands covering the 17 features shown in Figure 2. For the native XML database, 41 update commands, a subset of the 44 commands of ORDB, are used. The number of commands for the native XML database is less than for ORDB since XUpdate does not support recursion (C17) and navigation by reference traversal (C11). In the XML update commands, some commands contain two features since some features are simple or always appear along with other features. The feature C11 appears along with the feature C17, while the feature C2 (update without join) appears in the various commands updating a particular document without a join to other documents.

Each experiment is executed five times and the longest and the shortest elapsed times are ignored; thus only an average of three elapsed times is reported.

Discussion of the Experimental Results

The graphs and tables, to evaluate the performance of XML update processing according to the purposes of the experimental study, are produced as follows.

Performance with Different Data Redundancy and Data Caching

Figure 3 contains four graphs, one for each size of the database. Each graph plots average elapsed time for replace, delete and insert operations against the number of redundant records in the three possible cache states: cold, warm and hot. In cold cache all required data is on the disk, whereas in warm cache most of the required data is in memory and in hot cache all required data is in the memory.

C1 Exact Match	C10 Join documents in update
C2 Update without join of documents	C11 Navigation by reference traversal
C3 Change selectivity	C12 Handling missing elements
C4 Allow condition on text	C13 Element ordering
C5 Support aggregation	C14 Using regular path expression
C6 Support quantifiers	C15 Mix between data-centric and document-centric
C7 Joins based on values	C16 Hierarchical and sequence update
C8 Joins based on pointer	C17 Recursion and reference traversal
C9 Casting	

Figure 2. The 17 update features for the experimental study

The times for the operations individually are available in Amornsinlaphachai, 2007. Note that the update time in the graphs and the tables excludes serialization time since, usually, serialization can be performed only once after all updates finish.

The commands C14 (regular path expression) and C17 (recursion) are excluded from the average time of nxd since X-Hive does not support C17 while the elapsed time of C14 run on X-Hive is so long that it can affect the overall performance of the systems. The command C3 (change selectivity) is also not included in the average time of sxd because it takes a relatively long elapsed time. The performance of each command will be discussed in the next section.

From the graphs in Figure 3, in cold cache sxd (single XML document database) has the worst performance for every data size, whereas nxd (native XML database) has the best performance. However, when the data size is 40 MB and there is considerable data redundancy, lxd can outperform nxd. This is because lxd does not contain redundant data; thus, although the number of updated records will be constant for every degree of redundancy, the data size will be smaller when nxd has more redundant data. Therefore the performance of lxd is better when the degree of redundant data in nxd is greater.

For warm and hot caches, lxd has the best performance in all cases while nxd has the worst performance when the data size is smaller than 20 MB. When the data size is 20 and 40 MB, then nxd can outperform sxd. This is because the update time in sxd consists of SQL-time and DOM-time (time for updating DOM of XML) and when the data size is doubled, DOM-time is increased about twice whereas the update time of nxd is increased to a lesser extent. Also when the data size is increased, there are more records to be updated; thus it means that for ORDB with sxd, there are more data to be preserved for the rollback purpose, whereas we have not found that nxd has a mechanism to preserve data for the rollback purpose. The latter reason is also the reason why the performance for sxd is

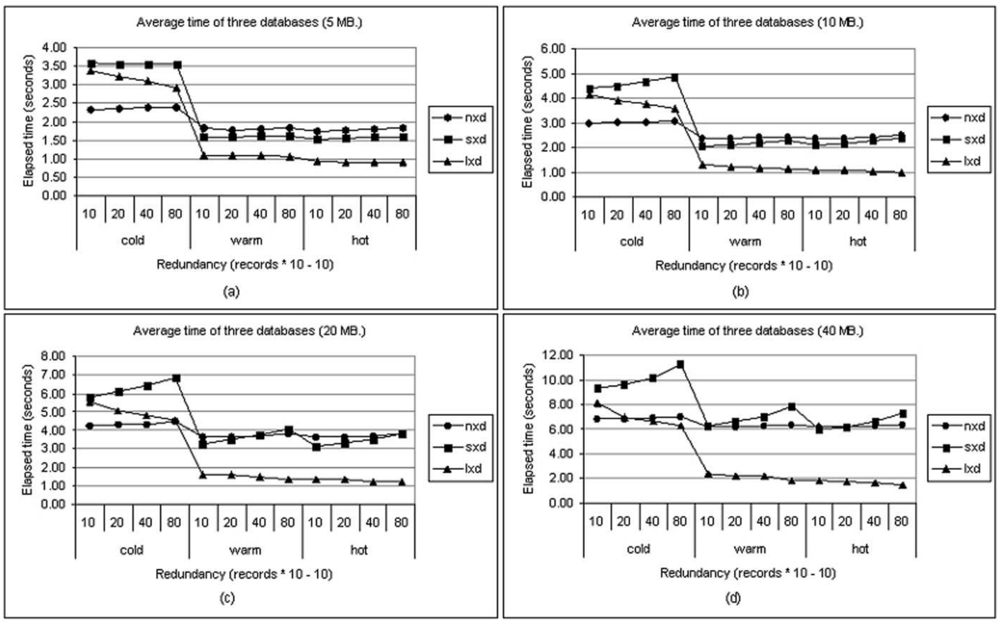


Figure 3. Average time of nxd, sxd, and lxd in cold, warm, and hot caches nxd: native XML database; sxd: single XML document database; lxd: linked XML document database

affected by redundant data more than that for nxd, as can be seen clearly when the data size is bigger than 5 MB.

The time in cold cache for lxd is about four or five times that in warm cache and the time in cold cache for sxd is double that in the warm cache. By contrast, the times in cold and warm caches for nxd are similar, showing that lxd and sxd gain more benefit from caching data than nxd. The difference in time between the cold and warm caches for lxd is more than that for sxd. This is because caching data has only a little effect on DOM-time but a much greater effect on SQL-time. For lxd the DOM-time is small when compared to the SQL-time; thus when the cache state is changed from cold to warm, most of the difference in time is derived from the change in SQL-time, causing significant difference in the times for cold and warm caches. On the other hand for sxd, the DOM-time is nearly equal to the SQL-time in cold cache and the DOM-time is changed only a little when the cache state is changed from cold to warm. Thus the difference in time between cold and warm caches in sxd is less than in lxd.

The Performance of the Seventeen Update Features

We present here only some important data, which can be used as representative for the performance against the update features. The representative data consists of insert and delete commands performed in hot cache with data size 40 MB, which is the biggest size of data tested. Only the results for 40 MB data size with 800 redundant records are presented since the difference in performance between the update features is similar for each size of redundant data. In practice both cold and hot caches can be used to capture actual performance. However in the real world, applications are run more in warm or hot cache than in cold cache; therefore we choose one of these, hot cache, to show the performance against each update feature. The replace commands are not presented here since nxd does not directly support replacing a complex element: several update commands are executed on simple elements instead of one

Table 1. Insert time of three databases in hot cache (40 MB, 800 redundant records)

nxd		sxd		lxd	
C17	-	C12	4.95	C12	1.38
C5	3.75	C15	5.14	C17	1.41
C15	3.94	C4	5.15	C15	1.44
C4	4.19	C9	5.17	C4	1.47
C10	4.32	C6	5.19	C14	1.48
C3	4.33	C14	5.21	C9	1.51
C1	4.57	C132	5.32	C8	1.76
C12	4.65	C131	5.34	C1	1.81
C8	4.72	C8	5.42	C3	1.82
C131	4.75	C7	5.58	C6	1.82
C132	4.75	C10	5.59	C132	1.91
C7	4.85	C5	5.72	C131	1.92
C9	4.94	C3	5.84	C10	2.03
C6	5.62	C1	5.93	C7	2.06
C16	9.37	C16	9.26	C5	2.26
C14	85.32	C17	9.74	C16	3.34
AVG	4.91	AVG	5.91	AVG	1.84
(a)		(b)		(c)	

update command on the complex element.

The representative data is summarized in Tables 1 and 2. Table 1 shows the performance difference between the update features of insert commands for the three types of database nxd, sxd and lxd. The commands are given in ascending order of elapsed time.

From Table 1(a), the command C14 testing *regular path expression* (//) produces an unacceptable performance. This indicates that nxd has a weak point in handling a *regular path expression* because all possible paths in XML documents must be navigated. On the other hand, from the Tables 1(b) and (c), sxd and lxd can handle this type of command well since the XML update language is translated into SQL; thus the real path expression is not genuinely involved in executing the update command. Additionally the path in the regular path expression can be determined by using mapping information; thus fields or tables involved in the expression are solved in a short time.

Command C16 (*hierarchical and sequence update*) comprises a sequence of update commands; thus its elapsed time is longer than for one single update command. If C14 and C16 are not taken into account, the command C6 testing a quantifier takes the longest elapsed time because not only is the tree searched to match the condition but also the grouping of data is calculated. The elapsed times of the rest of the commands are close to each other. In this group, the command C9 takes the longest elapsed time since the content of XML is text; thus it takes time to cast the text type to a numeric type to calculate the data.

For sxd from Table 1(b), the command C17 testing *recursion* takes the longest elapsed time because in translating the XML update language, we repeat the deletion and insertion of data from and into the temp table. The elapsed time of the command C16 is the second longest since C16

Table 2. Comparison of insert time and delete time

Cmd.	nxd		sxd		lxd	
	Insert	Delete	Insert	Delete	Insert	Delete
C1	4.57	4.53	5.93	5.91	1.81	1.84
C3	4.33	4.25	5.84	12.38	1.82	2.00
C4	4.19	4.07	5.15	5.19	1.47	1.44
C5	3.75	3.69	5.72	5.74	2.26	2.21
C6	5.62	5.53	5.19	5.20	1.82	1.81
C7	4.85	4.77	5.58	5.69	2.06	1.97
C8	4.72	4.68	5.42	5.59	1.76	1.74
C9	4.94	4.86	5.17	5.20	1.51	1.46
C10	4.32	4.26	5.59	5.59	2.03	2.03
C12	4.65	4.63	4.95	5.16	1.38	1.41
C131	4.75	-	5.34	-	1.92	-
C132	4.75	-	5.32	-	1.91	-
C14	85.32	85.44	5.21	5.21	1.48	1.42
C15	3.94	3.83	5.14	5.16	1.44	1.39
C16	9.37	9.28	9.26	9.47	3.34	3.36
C17	-	-	9.74	9.75	1.41	1.41
AVG	4.91	4.86	5.91	6.07	1.84	1.82
STDEV	1.36	1.84	1.43	1.59	0.48	0.53
%RSD	27.78	30.34	24.22	26.19	26.20	29.12
	(a)		(b)		(c)	

performs a sequence of update operations; thus more than one subcommand is executed.

The elapsed times of the rest of the commands are close to each other. C5 takes a longer time than C6 since, in our language translation, a quantifier is translated into the count() function along with the conditions of the quantifier; thus before grouping and counting the data in each group, the conditions can eliminate unwanted data to decrease the size of data. The command C8 testing *joins based on pointers* takes a shorter time than C7 since C8 uses a foreign key and a primary key to join data.

For lxd from Table 1(c), the sequence of elapsed time of lxd is different from sxd since the number and the structure of tables in sdx and lxd are not the same. The lxd has more layers of objects inside tables than sxd.

The command C16 takes the longest elapsed time because C16 is a sequence of update operations. The command C5 takes the second longest elapsed time because C5 involves three tables with the need to both group and count data. The command C7 takes the third longest elapsed time. Although C7 involves only two tables, it does not use the key for joining data. The command C10 takes the fourth longest elapsed time because it involves four tables. The elapsed times of the rest of the commands are not much different. Similar to sxd, C7 takes a longer time than C8 and C5 takes a longer time than C6.

Table 2 compares the performance between insert and delete commands for the three types of database. From this table, there is not much difference between the performance of insert and delete commands except for C3 (*change selectivity*) with sxd in Table 2(b). Here, the delete command takes much more time than the insert command because two conditions, '>=' and '<=', need to be satisfied in the scan of a full-table. The deletion of the data requires the DBMS to preserve the previous version of the data for rollback purposes, an operation performed for each target record found. For lxd, there are no redundant records so the number of operations is much fewer than in sxd.

From the values of the standard deviations in Table 2, the performance difference between most update features for nxd. sxd and lxd are not significant. However the performance for lxd is, on the whole, much better than for sxd and nxd; indeed with lxd all features are handled better than with nxd. A particularly weak point of nxd is handling *regular path expressions*. In deletions with sxd, the capability for handling *change selectivity* is inadequate.

Conclusions

Meanwhile a standard for updating XML documents has not been proposed. The existing XML databases and XML update languages have limitations in their capability for updating data. In our technique, the technology of ORDB is exploited to increase the capability of existing XML update approaches in the aspect of controlling constraints during updating XML data, making it easier to join XML documents in updating, allowing the updates of documents whose structure is known partially or whose structure is recursive, and improving the performance of the updates by using regular path expressions. With this approach, there is no need to maintain the order of elements in ORDB and the cost of converting ORDB data back to XML format is eliminated since the change in ORDB is propagated to XML already. Our approach makes it possible to query XML data from XML documents instead of just ORDB. For example, using Kweelt (Sahuguet, 2001) which is an implementation of XQuery for querying XML documents directly, the result from querying is returned in XML format without any conversion. Although DOM has to be serialized back to an XML document, serialization is performed only once after all updates finish.

We have conducted the experimental study. The experimental result shows that overall updating non-redundant data outperforms updating redundant data. Caching data affects the performance of updating. The native XML database has a weakness for handling regular path expressions.

There are many interesting avenues for further work in the XML update area since

XML update is still in its infancy. The possible extensions to our research include updating the XML structure, transaction processing, concurrency control, security for accessing XML data and query optimisation through using the results obtained as parameters for a cost model.

References

Amornsinlaphachai, P. (2007). Updating semi-structured data, [Ph.D. thesis]. School of Computing, Engineering and Information Science, Northumbria University, UK, p. 314.

Amornsinlaphachai, P., Rossiter, N., and Ali, M.A. (2005). Updating XML using object-relational database. Proceedings of British National Conference; July 5-7, 2005; Sunderland University, UK. Springer-Verlag, Berlin, p. 155-160.

Amornsinlaphachai, P., Rossiter, N., and Ali, M.A. (2006a). Storing linked XML documents in object-relational DBMS. Journal of Computing and Information Technology, 14(3):225-241.

Amornsinlaphachai, P., Rossiter, N., and Ali, M.A. (2006b). Translating XML update language into SQL. CIT, 14(2):81-100.

Arenas, M. and Libkin, L. (2004). A normal form for XML documents. ACM Transactions on Database Systems (TODS), 29(1):195-232.

Babcock, C. (2002). Internet insight: XML users consider nonstandard third-party software to ease update process (appears in Ziff Davis' eWeek 11 Feb. 2002). Available from: http://www.charlesbabcock.com/xquery.htm. Accessed date: August 25, 2005.

Chamberlin, D. (2003). XQuery from the Experts: A Guide to the W3C XML Query Language. 1st ed. Addison-Wesley Professional, Boston, 484p.

Krishnamurthy, R., Chakaravarthy, V.T., Kaushik, R., and Naughton, J.F. (2004). Recursive XML schema, recursive XML queries, and relational storage: XML-to-SQL query translation. Proceedings of the 20th International Conference on Data Engineering, ICDE 2004; March 30- April 2, 2004; Boston, MA, USA, p. 42-53.

Lu, L., Liu, M., and Wang, G. (2003). A declarative XML-RL update language. Proceedings of 22nd International Conference on Conceptual Modeling (ER 2003); October 13-16, 2003; Chicago, Ill, USA, Springer-Verlag, Berlin, p. 506-519.

Lu, S., Sun, Y., Atay, M., and Fotouhi, F. (2005). On the consistency of XML DTDs. Data & Knowledge Engineering, 52:231-247.

Lv, T. and Yan, P. (2006). Mapping DTDs to relational schema with semantic constraints. Information and Software Technology, 48:245-252.

Mignet, L., Barbosa, D., and Veltri, P. (2003). The XML web: a first study. The 12th International World Wide Web Conference (WWW2003); May 20-24, 2003; Budapest, Hungary, p. 500-510.

Obasanjo, D. and Navathe, S.B. (2002). A proposal for an XML data definition and manipulation language. Proceedings of VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb on Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web-Revised Papers; August 20-23, 2003; Hong Kong China, p. 1-21.

Pardede, E., Rahayu, J.W., and Taniar, D. (2006). Object-relational complex structures for XML storage. Information and Software Technology, 48:370-384.

Prakash, S., Bhowmick, S.S., and Madria, S. (2006). Efficient recursive XML query processing using relational database systems. Data & Knowledge Engineering, 58(3):207-242.

Sahuguet, A. (2001). Kweelt: more than just "yet another framework to query XML!" Proceedings of 2001 Association for Computing Machinery (ACM) Special Interest Group on Management of Data (SIGMOD) Conference; May 21-24, 2001; Santa Barbara, CA., p. 602.

Tatarinov, I., Ives, Z., Halevy, A.Y., and Daniel, S.W. (2001). Updating XML. Proceedings of 2001 Association for Computing Machinery (ACM) Special Interest group on Management of Data (SIGMOD) Conference; May 21-24, 2001; Santa Barbara, CA., USA, p. 413-424.

Tatarinov, I., Viglas, S.D., Beyer, K., Shanmugasundaram, J., Shekita, E., and Zhang, C. (2002). Storing and querying ordered XML using a relational database system. Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data; June 3-6, 2002; Madison, WI, USA, p. 204-215.

Wang, G. and Liu, M. (2003). Query processing and optimization for regular path expressions. Proceedings of Advanced Information Systems Engineering, 15th International Conference; June 16-20, 2003; Klagenfurt, Austria, p. 30-45.

Watson, P. (2005). Databases in grid applications: locality and distribution. Proceedings of the Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases, BNCOD 22; July 5-7, 2005; Sunderland, UK, Springer-Verlag, Berlin, p. 1-16.

XMLDB. (2002). XUpdate. Available from: http://www.xmldb.org/xupdate/xupdate-wd.html. Accessed date: April 19, 2004.

Zwol, R.V., Apers, P.M.G., and Wilschut, A.N. (1999). Modeling and querying semistructured data with MOA. Proceedings of Workshop on Query Processing for Semistructured Data and Non-standard Data Formats; October 31, 1999; Jerusalem, Israel, p. 1-5.