Efficient XML Data Management: An Analysis

Ullas Nambiar[1], Zoé Lacroix[1], Stéphane Bressan[2], Mong Li Lee[2], and Ying Guang Li[2]

[1] Arizona State University
{mallu,zoe.lacroix}@asu.edu
[2] National University of Singapore
{steph,leeml,liyg}@comp.nus.edu.sg

Abstract. With XML rapidly gaining popularity as the standard for data exchange on the World Wide Web, a variety of XML management systems (XMLMS) are becoming available. The choice of an XMLMS is made difficult by the significant difference in the expressive power of the queries and the performance shown by these XMLMS. Most XMLMS are legacy systems (mostly relational) extended to load, query, and publish data in XML format. A few are native XMLMS and capture all the characteristics of XML data representation. This paper looks at expressive power and efficiency of various XMLMS. The performance analysis relies on the testbed provided by XOO7, a benchmark derived from OO7 to capture both data and document characteristics of XML. We present efficiency results for two native XMLMS, an XML-enabled semi-structured data management system and an XML-enabled RDBMS, which emphasize the need for a delicate balance between the data-centric and document-centric aspects of XML query processing.

1 Introduction

The *eXtended Markup Language* (XML) is designed as the standard for information interchange on the Web. XML is a subset of SGML (Standardized General Markup Language) designed to provide structure to textual documents and augments HTML (Hyper Text Markup Language) by allowing data to carry its meaning and not just presentation details. XML's development was not furthered directly by the mainstream database community, yet database researchers actively participated in developing technology for XML, particularly query languages. This led to the development of query languages such as XML-QL [18], LOREL [3] and XQL [27]. These languages designed by the database community are biased toward the data-centric view of XML that requires data to be fully structured. But XML was developed primarily as a document markup language that would be more powerful than HTML yet less complex than SGML and does not require the content to adhere to structural rules. The document characteristics of XML representation not only relies on the data representation expressed through markups but also on the ordering of the document components within the file. This leads us to question whether the query languages designed by

K. Bauknecht, A M. Tjoa, G. Quirchmayr (Eds.): EC-Web 2002, LNCS 2455, pp. 87–98, 2002.

the database community and the data management systems that use these languages to manipulate XML data, capture the whole essence and power of XML. In essence, these languages and systems have a data-centric view of XML, and merely use XML to publish the data. On the other hand many systems use implementations of XPath [14], a language designed to identify parts of XML documents, to query XML data. Such systems then subscribe to the document-centric view of the XML. Thus the initial use of XML has been largely polarized with a large number of users (coming from the database community) developing systems for the data-centric characteristics of XML, while many others in areas like Bio-informatics research, Medical Systems and Geographical Information Systems (GIS) use XML for its ability to manipulate documents. Recently the World Wide Web Consortium (W3C) published XQuery [12] as a candidate for a standard query language for XML, combining both the data and document centric characteristics of XML.

In this paper, we study XML management systems (XMLMS) by addressing the issues of data representation and storage, and XML query functionalities and processing efficiency. We first analyse the design of XMLMS and identify XML characteristics and XML query capabilities and their corresponding consequences on the performance. In the second part of the paper, we run experiments to coroborate our analysis. We use XOO7 [26,9], a benchmark for XML databases, to compare the current XML data management systems. The systems we compare are : LORE [3], Kweelt [28], XENA [31] and an XPath [14] implementation[1].

Section 2 explores XML in terms of its known representations and describes four XMLMS that use these representations. . We also investigate how XML data affects processing efficiency of database systems. We briefly introduce XOO7, the benchmark we use in Section 3. In Section 4 we analyze the results from the performance study of the four XML data management systems we chose against the benchmark queries, and conclude in Section 5 by highlighting our contributions and the possible extensions of this work.

2 Representational Diversity and Processing Efficiency of XML

XML was designed to overcome the shortcomings of its two predecessors, SGML and HTML. SGML is complex, in particular for simple applications such as publishing on the web. The principle drawback of SGML is its strict adherence to marked-up structure of the documents. On the other hand, HTML designed at CERN as a watered down version of SGML, provides a common set of tags for display. So parsers can be incorporated into Web Browsers which made HTML the first language of the World Wide Web. But this *flexibility in usage of its syntax* that made HTML popular also makes it a bad candidate for exchanging

[1] The developers of the commercial product dismissed our request to use actual name of the product.

data over the Web. HTML allows multiple interpretations of the same data and makes it impossible to add semantics to the data published in HTML. In contrast, XML is a very versatile yet easy to use markup language, that has the extensibility of SGML but remains as simple as HTML.

2.1 Data-Centric versus Document-Centric View of XML

The initial attempts at developing tools for storage of XML content were biased by the legacy of the researchers who worked on providing the solutions. Solutions developed by the database community focused at using XML as *yet another data format* and the use of relational and sometimes object-relational data processing tools. While this use of XML is acceptable, it raises the question of harnessing the full power of XML. XML is inherently semi-structured. Although, like SGML, XML documents can use a DTD to derive their structure, DTDs are not a must for all XML documents. Thus XML documents can take any structure. On the other hand, relational and object-relational data models have a fixed pre-defined structure. We can represent totally structured data using XML in combination with DTDs or XML Schema specifications [20,30,5], which we term as *data-centric* characteristic or format of XML. The documents subscribing to the data-centric view of XML will be highly structured. Similar to traditional (relational) databases, the order of sibling elements is unimportant in such documents. On the other hand, *document-centric* XML content is highly unstructured, and both the implicit and explicit order of elements is important in such documents. The implicit order is carried by the order of the elements (as siblings in a tree-like representation) within the file, whereas an explict order would be expressed by an attribute or a tag in the document. Although it is easy to express the explicit order in relational databases, capturing the implicit order while converting a document-centric XML document into relational database proves to be a problem. Besides the implicit order, XML documents differ from a relational representation by allowing deep nesting and hyper-linked components. Implicit order, nesting and hyperlinks can always be represented in tables but with costly transformations in terms of time and space.

2.2 XML Processing Efficiency

Query languages designed to operate using the data-centric view cannot exploit the implicit order present in XML. But relational systems can efficiently process most data-centric queries. XML management systems that focus on a data-centric representation are less expressive but should be able to give good performance. On the other hand, systems using query languages that exploit document characteristics of XML have greater expressive power but are likely to be less efficient.

An adequate XML query language should definitely provide support for issuing all XML queries including: (1) Relational queries, (2) Document queries, and (3) Navigational queries. We classify XML queries that have expressive power similar to Datalog [15] and [19] for relational model as *Relational queries*. Queries

that use the implicit and explicit order of elements in an XML document, as well as textual functionalities are classified as *Document queries* while the queries that require traversal of XML document structure using references/links as supported by XLink/XPointer specification [17] and [16] are *Navigational queries*. A detailed classification of essential XML query characteristics is given by [26].

The ability to express and process document and navigational queries in addition to the traditional relational queries affect significantly the performance. A totally unordered XML data will require least processing time. The simple explanation is the similarity of such data with relational data and hence the ability to use optimized approaches from relational database community to process such data. In contrast, fully ordered data, requires the preservation of the structure of XML document for processing any query. Existing approaches at processing queries over fully ordered data require loading the entire document into main memory and creating a tree structure of the document. Similar analysis can be made for nested or hyperlinked documents versus flat data.

2.3 XML Management Systems

From the above discussion we divide current XML management systems into *XML-Enabled* and *Native XML databases*. XML-Enabled databases (usually relational) contain extensions (either model- or template-driven) for transferring data between XML documents and themselves and are generally designed to store and retrieve data-centric documents. For a detailed classification of XML management products refer to [8]. An example of a native XMLMS is Kweelt [28], a proposed implementation of Quilt [13]. Kweelt stores data in flat files and hence favours the document-centric nature of XML. LORE [3] is not exactly native but it is a semi-structured data management system later revised to handle XML documents [23]. The main difficulty of the conversion was indeed tackling the implicit order. Most of the existing XML data management systems are XML-enabled and built on top of relational or object-relational systems. They are used to publish data in XML and allow XML queries to be translated into SQL statements. We use XENA [31], designed at the National University of Singapore as an XML-enabled data management system. It stores the XML data into tables automatically according to the XML schema. XENA then retrieves data from tables by converting an XPath query into several SQL queries automatically, using the XML schema.

Our decision of choosing the above mentioned XML data management systems was primarily motivated by the easy availability of their source code and the detailed documentation about their implementations.

3 XML Database Benchmarks

Semistructured data models and query languages have been studied widely in [1] and [10]. In [22] several storage strategies and mapping schemes for XML data using a relational database are explored. Domain-specific database benchmarks

for OLTP (TPC-C), decision support (TPC-H, TPC-R, APB-1), information retrieval, spatial data management (Sequoia) etc. are available [25]. XOO7 [26] and [9], XMach-1 [6] and XMark [29] are the three benchmarks currently available that test XMLMS for their query processing abilities.

Table 1. Comparing Benchmarks over XML system characteristics

System Characteristics	XMach-1	XMark	XOO7
Selection Queries	√	√	√
Projection Queries	√	√	√
Reduction: Remove a selected element	√	√	√
Restructuring: Reorder sub elements	√	√	√
Construction: Output new structure	√	√	√
Remote Execution: Data and Evaluator on different machines			
Preserve Implicit Order			√
Exploit Schema	√	√	√
Schemaless Document Manipulations			√
XLink and XPointer Manipulations			
Streaming Data Processing: Schema generation on the fly			
Transaction Processing			
Text Search	√	√	√
View Processing			
Stored Procedures and Triggers			
User Defined Functions		√	√
Aggregate Manipulations			√
Update Element/Database			
Delete Element/Database	√		
Append Data	√		
Extract results based on Similarity Criterion			
Navigational Queries			√

XOO7 design attempts to harness the similarities in data models of XML and object-oriented approaches. Although XML attempts to provide a framework for handling semistructured data, it encompasses most of the modeling features of complex object models [2] and [4]. There are straightforward correspondences between the object-oriented schemas and instances and XML DTDs and data. XOO7 is an adaptation of the OO7 Benchmark [11] for object-oriented database systems. XOO7 provides 18 query challenges. The current implementation of XOO7 tests XML management systems which store their data locally. For a detailed description of XOO7 refer to [26].

XMach-1 tests multi-user features provided by the systems. The benchmark is modeled for a web application using XML data. It evaluates standard and non-standard linguistic features such as insertion, deletion, querying URL and aggregate operations. Although the proposed workload and queries are interesting, the benchmark has not been applied and no results have been published yet. XMark developed under the XML benchmark project at CWI, is a benchmark proposed for XML data stores. The benchmark consists of an application scenario which models an Internet auction site and 20 XQuery challenges designed to cover the essentials of XML query processing. These queries have been evaluated on an internal research prototype, Monet XML, to give a first baseline.

Table 1 compares the expressive power of queries from XOO7, XMark and XMach-1. As can be seen XOO7 is the most comprehensive benchmark in terms of XML functionalities covered. Both XMark and XMach-1 focus on a data-centric usage of XML. All three benchmarks provide queries to test relational model characteristics like selection, projection and reduction. Properties like transaction processing, view manipulation, aggregation and update, are not yet tested by any of the benchmarks. XMach-1 covers delete and insert operations, although the semantics of such operations are yet to be clearly defined under XML query model. In [26] detailed information about the data and schema used by these benchmarks is provided.

We choose to use XOO7 to analyze the performance of chosen XML data management systems. Our decision is motivated by the fact that XOO7 is a comprehensive benchmark as can be seen from Table 1 and also empirical evaluations show the ability of the XOO7 queries to distinguish all the desired functionalities supported by an XML database [9]. In the absence of queries exploiting the document-centric features, XMark and XMach-1 may not be able to clearly distinguish XML-enabled systems from Native XML management systems.

4 Empirical Study and Analysis of XML Databases

In this section we present results of experiments conducted to study the expressive power and processing efficiency of the four data management systems: LORE, Kweelt, XENA and a commercial implementation of XPath, which we call DOM-XPath. We compare the systems in terms of their response times for relational, document and navigational queries taken from the XOO7 benchmark [26]. The experiments are run on a 333 Mhz system running SunOS 5.7 with 256 MB RAM.

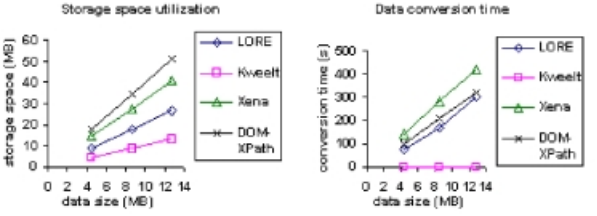

Fig. 1. Space Utilization and Data Conversion Time

4.1 Data Conversion: Time and Space Requirements

We recorded the time and space utilized by the XMLMS for converting the datasets provided as part of XOO7 to their proprietary format. The space utilization is measured in terms of secondary storage space used by each system

for the various databases in the benchmark. Figure 1 compares the space and time requirements of the various XML data management systems we test. We use datasets of three sizes: small, medium and large for our tests. Small dataset is of size 4.2MB, medium of size 8.4MB and large has size 12.8 MB. The datasets are designed using the schema provided in XOO7 [9].

Kweelt queries the ASCII file directly and does not need to convert the XML data into another format. So the storage space it needs is the same as the size of the XML data. XENA stores XML data in MySQL tables. Although the conversion from XML format to relational tables ends up removing the redundant tags around the XML data, XENA ends up generating a number of relational tables to represent the XML data. In fact XENA creates two groups of tables. One group is based on the XML schema with one table per entity; the other group is for the management of these tables. Hence XENA requires almost double the space used by actual XML data after conversion. LORE creates Dataguides [24] for the datasets to help in efficient query processing. Dataguides are a concise and accurate summary of all paths in the database that start from the root. Hence LORE requires almost three times the space of the actual XML data, as can be seen from Figure 1. The commercial implementation of XPath, DOM-XPath, also creates three binary files for an XML dataset. One of the files is a proprietary database that preserves the native XML structure by storing the entire document tree of the dataset thereby occupying much larger space than the XML dataset.

Not surprisingly, Kweelt, is most efficient compared to other systems in terms of space usage. Since Kweelt processes directly raw XML data, it requires no time for data conversion. As expected, XENA, an XML-enabled database system requires the most amount of time to convert from the XML model to a relational model. LORE requires time for generating Dataguides and we assume the XPath implementation is also generating indexes, task that requires time.

4.2 Response Time Analysis

We divide the XOO7 queries into three groups: Relational queries, Navigational queries and Document queries. Table 2 depicts the proposed classifcation of XOO7 queries. In the following, we will illustrate the performance of the various systems for the representative queries in each group.

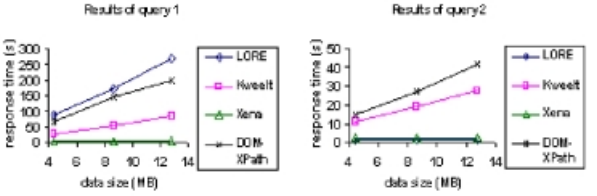

Fig. 2. Response time for Q1 and Q2 (relational queries)

Table 2. XOO7 benchmark queries

ID	Description
	Relational Queries
Q1	Randomly generate 5 numbers in the range of AtomicPart's MYID. Return AtomicPart according to the 5 numbers.
Q2	Randomly generate 5 titles for Documents then return the first paragraph of Document with the titles.
Q3	Select 5% of AtomicParts via buildDate (in a certain period).
Q5	Join AtomicParts and Documents on AtomicParts docId and Documents MyID.
Q7	Randomly generate two phrases among all phrases in Documents. Return the documents with both the phrases.
Q8	Repeat query 1 but replace duplicate elements using their IDREF.
Q13	For each BaseAssembly count the number of documents.
Q14	Sort CompositePart in descending order where buildDate is within a year from current date.
Q16	Return all BaseAssembly of type "type008" without any child nodes.
	Navigational Queries
Q4	Find the Compositepart if it is later than BaseAssembly it is using.
Q6	Select all BaseAssemblies from an XML database having same "type" attribute as the BaseAssemblies in another database but with later buildDate.
Q9	Select all AtomicParts with corresponding CompositeParts as their sub-elements.
Q10	Select all ComplexAssemblies with type "type008".
Q15	Find BaseAssembly of not type "type008".
Q17	Return all Connection elements with length greater than Avg(length) within the same composite part without child nodes.
Q18	For CompositePart of type "type08", give 'Result' containing ID of CompositePart and Document.
Q19	Select all of the CompositePart, Document and AtomicPart.
	Document Queries
Q11	Among the first 5 Connections of each CompositePart, select those with length greater than "len".
Q12	For each CompositePart, select the first 5 Connections with length greater than "len".
Q20	Select the last connection of each CompositePart.
Q21	Select the AtomicPart of the third connection in each CompositePart.
Q22	Select the AtomicPart whose MyID is smaller than its sibling's and it occurs before that sibling.
Q23	Select all Documents after the Document with MyID = 25.

Figure 2 compares the performance of the four XMLMS for two relational queries in the XOO7 benchmark. Query $Q1$ tests simple selection processing efficiency while query $Q2$ uses selection having string comparison. XENA gives the best performance in both queries because it leverages the power of its backend relational database. LORE gives interesting results: it is efficient for query $Q2$ as we expected whereas has poor response for $Q1$, and has most response time for $Q1$. The default data type in LORE is string, hence string comparison is very fast, but comparisons on other types require frequent type casting and drop its performance. Both Kweelt and DOM-XPath are implemented based on DOM, but Kweelt always gives a better performance than DOM-XPath (This is also the case for the navigational and document queries). There are two possible reasons. First, they may be using different parsers. Second, DOM-XPath, being a commercial product, is required to handle additional issues like admission control, which may introduce some additional workload. Kweelt, being a research prototype, concentrates on optimized query processing only.

Overall for the relational queries, the two native XMLMS, Kweelt and DOM-XPath, give relatively poorer performance than the two XML-enabled systems. LORE does not perform well when data type coercion is required. XENA leverages the query processing power of the relational database engine and yields the best performance. Kweelt and DOM-XPath always need to follow a particular path to check whether an element or an attribute satisfies certain conditions, thus more processing is needed.

In Figure 3 we compare the performance of the XMLMS for two representative navigational queries, $Q4$ and $Q15$. Query Q4 tests the parent-child relations in XML data, while query $Q15$ measures the ability to preserve the structure of original XML data i.e. preserving the paths and the sibling orders as those in the

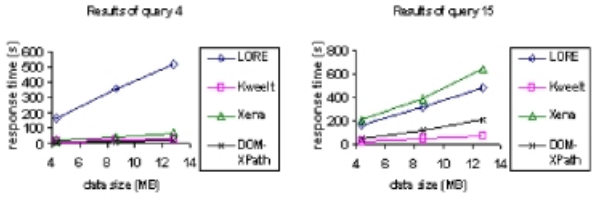

Fig. 3. Response time for Q4 and Q15 (navigational queries)

original XML document. LORE shows the worst performance. XENA is much faster than LORE in query $Q4$ but slower for $Q15$. To keep the parent-child relations, XENA saves the parent index for each element. The indices on the path fields are built automatically. The response time of XENA increases almost 10 times from query $Q4$ to query $Q15$. Unfortunately, we could not ascertain how LORE maintains the response time to be almost a constant for the both the queries. Kweelt and DOM-XPath store the XML data in the original form, therefore they perform relatively better.

In general, the two XML-enabled systems show poor performance for navigational queries. The primary reason is they change the structure of the XML data to proprietary formats, so they need additional time to reconstruct the original XML data. The two native XML management systems store the XML data in the original form, hence they simply return the elements satisfying the conditions following the original structure.

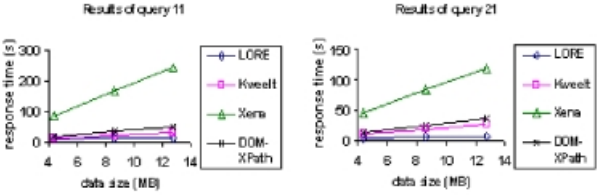

Fig. 4. Response time for Q11 and Q21 (document queries)

Figure 4 gives a performance comparison for two document queries in the XOO7 benchmark. Query $Q11$ tests whether elements in a certain range satisfy given conditions while query $Q21$ selects elements with a particular order. The implicit order is required to answer both the queries. XENA gives the worst performance for both queries. The results given by XENA strengthen our belief that it is difficult to preserve the implicit order of elements in a relational database. LORE performs the best in the two document queries because it makes use of DataGuide to record the orders of elements. Kweelt and DOM-XPath perform relatively better than XENA. They simply check the element orders based on

the knowledge from the DOM tree. We did not care about the attribute order as it is not required by the W3C working group.

From the experiments with the XOO7 benchmark, we can see that the basic XML-enabled management systems are inadequate to perform navigational queries and document queries, but they can be improved by introducing some techniques, like DataGuide, to record the original structure of the XML data. The native XML implementations prove to be more efficient for navigational queries and document queries. New techniques for representing and extracting relational data in native XMLMS such as storing meta information about tuple structure, creating relational style indexes or incorporating a small relational optimizer would help improve the performance for relational queries.

5 Conclusion and Future Work

To evaluate the underlying XML technologies (e.g. XPath, XPointer, XQuery, etc.) and efficiency of an XMLMS using them, a benchmark becomes inevitable. In this paper, we identify current XML representations and analyze their effect on the performance of the systems. Then we corroborate our study by running experiments with XOO7. Our results confirm that XML-enabled relational database systems which use the data-centric view of XML process more efficiently the queries that only manipulate data and do not use the implicit order or the hyperlinks in documents. On the other hand, native XMLMS designed to handle raw XML data and documents are efficient in processing document or navigational queries whereas they show poor performance for relational queries. Currently no system seems to offer a needed balance between the data-centric and document-centric approaches. The choice of an XMLMS then depends on the type of data that needs to be stored and the type of queries that will be expressed. A user with data centric needs will favor an XML-enabled DMS whereas one wishing to store and manipulate documents will chose a native XMLMS.

Acknowledgements. We thank the XENA project team for providing us with the source code and valuable comments in setting up XENA.

References

1. S. Abiteboul. Querying semi-structured data. In *Proc. of Intl. Conf. on Database Theory*, pages 1–18, Delphi, Greece, January 1997. LNCS 1186, Springer Verlag.
2. S. Abiteboul and S. Grumbach. COL: A Logic-Based Language for Complex Objects. *EDBT*, pages 271–293, 1988.
3. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The Lorel Query Language for Semistructured Data. *Journal on Digital Libraries*, 1997.
4. S. Abiteboul and M. Scholl. From Simple to Sophistic Languages for Complex Objects. *Data Engineering Bulletin*, 11(3):15–22, 1988.
5. P. Biron and A. Malhotra. *XML Schema Part 2: Datatypes.* W3C, 2001. Recommendation – available at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502.

6. T. Bohme and E. Rahm. XMach-1: A Benchmark for XML Data Management, 2000. Available at http://dbs.uni-leipzig.de/projekte/XML/XmlBenchmarking.html.
7. A. Bonifati and S. Ceri. Comparative analysis of five xml query languages. *SIGMOD Record*, 29(1):68–79, 2000.
8. R. Bourett. Xml database products, May 2001. available at http://www.rpbourret.com/xml/XMLDatabaseProds.htm/.
9. S. Bressan, G. Dobbie, Z. Lacroix, M. L. Lee, Y. G. Li, U. Nambiar, and B. Wadhwa. XOO7: Applying OO7 Benchmark to XML Query Processing Tools. *Proceedings of CIKM. Atlanta.*, November 2001.
10. P. Buneman. Semistructured Data. In *Proc. ACM Symp. on Principles of Database Systems*, Tucson, 1997.
11. M.J. Carey, D.J. DeWitt, and J.F. Naughton. The OO7 benchmark. *ACM SIGMOD Conference*, pages 12–21, 1993.
12. D. Chamberlin, D. Florescu, J. Robie, J.Siméon, and M. Stefaescu. *XQuery: A Query Language for XML.* W3C, 2000. Available at http://www.w3.org/TR/xmlquery.
13. D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. In *Proceedings of the Workshop WebDB (in conjunction with ACM SIGMOD)*, Dallas, TX, 2000.
14. J. Clark and S. DeRose. *XML Path Language (XPath).* W3C, 1999. Available at http://www.w3.org/TR/xpath.
15. C. J. Date. *An Introduction to Database Systems.* Addison-Wesley, 1995.
16. S. DeRose, R. Daniel, and E. Maler. *XML Pointer Language (XPointer).* W3C, 1999. Available at http://www.w3.org/TR/WD-xptr.
17. S. DeRose, E. Maler, D. Orchard, and B. Trafford. *XML Linking Language (XLink).* W3C, 2000. Available at http://www.w3.org/TR/xlink.
18. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: a query language for XML. Available at http://www.w3.org/TR/NOTE-xml-ql/, 1998.
19. R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems.* Addison-Wesley, 1998.
20. D. Fallside. *XML Schema Part 0: Primer.* W3C, 2001. Recommendation – available at http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/.
21. P. Fankhauser, M. Marchiori, and J. Robie. *XML Query Requirements.* W3C, 2000. Available at http://www.w3.org/TR/xmlquery-req.
22. D. Florescu and D. Kossman. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database, May 1999. Report 3680 INRIA, France.
23. R. Goldman, J. McHugh, and J. Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. In *ACM SIGMOD Workshop on the Web and Databases (WebDB'99)*, 1999.
24. R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. of Intl. Conf. on Very Large Data Bases*, Delphi, Greece, August 1997.
25. J. Gray. *The Benchmark Handbook: For Database and Transaction Processing Systems.* Morgan Kaufmann, 2nd edition, 1993.
26. U. Nambiar, Z. Lacroix, S. Bressan, M. L. Lee, and Y. G. Li. Benchmarking XML Management Systems: The XOO7 Way. *Proceedings of IIWAS, Linz, Austria.*, September 2001.

27. J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL). In *Proc. of the Query Languages workshop*, Cambridge, MA, December 1998. Available at http://www.w3.org/TandS/QL/QL98/pp/xql.html.
28. A. Sahuguet. KWEELT : More than just "yet another framework to query XML!". *Sigmod Demo*, 2001.
29. A. R. Schmidt, F. Waas, M. L. Kerste, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, April 2001.
30. H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. *XML Schema Part 1: Structures*. W3C, 2001. Recommendation – available at http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.
31. Y. Wang and K. Tan. A Scalable XML Access Control System. *10th World Wide Web Conference*, May 2001.