World Applied Sciences Journal 19 (7): 972-985, 2012
ISSN 1818-4952

DOI: 10.5829/idosi.wasj.2012.19.07.71012

A Comparative Study Between Two Types of Database Management Systems: XML-Enabled Relational and Native XML

Amin Y. Noaman and Amal Abdullah Al Mansour

Faculty of Computing and Information Technology,
King Abdulaziz University, Jeddah, Kingdom of Saudi Arabia

Abstract: As there is an increase use of XML documents to store and exchange data over the web, there is a great need for a database management system able to store, retrieve and manipulate XML data with an efficient manner, here the XML database (XML DB) can cope this request. Most XML DB which is often called XML-enabled databases is a legacy database systems (mostly relational) extended to store, retrieve and manipulate XML data; few of them are native XML databases that directly and naturally support XML data by capturing all the characteristics of XML data representation. The two XML DB types have different characteristics and performance which make it difficult to choose the best XML DB to suit XML data. Thus, the major objective of this paper is to explore and compare between the performances of both: XML-enabled database and native XML database. Regarding this matter, our paper search for deciding whether an XML- enabled relational database, as a relatively mature technology, or native XML database is suitable for storing XML data in a way that allow an efficient query performance. Our runing experimental proved that the native XML database is performing better than XML-enabled relational database for XMark benchmark queries.

Key words: XML DB · Native XML · XML-Enabled Relational database · XMark benchmark · XML Server · SQL · HTTP · JDBC and W3C

INTRODUCTION

eXtensible Markup Language (XML) standard promises to be the standard for data representation in e-business, particularly when that data is exchanged over or browsed on the Internet. It's nested; self-describing structure provides a simple yet flexible means for business applications to model and exchange data. For example, a business can easily model complex structures such as purchase orders in XML format.

As companies are keeping increasingly large amounts of business critical data in XML format, it becomes increasingly important for them to be able to store, query and manipulate their XML data efficiently. This is where XML database comes in. Managing large amounts of data efficiently and securely is a problem that is traditionally solved by database management system.

The reasons for storing XML in a database system are the same as for relational data given by: consistent storage, transactional consistency, recoverability, high availability, security, efficient query and update operations and scalability. These are all features which make a database much more appropriate repository for XML data. Thus, XML databases have gained increasing popularity and importance in recent years. XML databases can be classified into two main types: XML-enabled database and native XML database. In the first type, an XML-enabled database, extensions are added to a preexisting database management system to support XML documents. An XML-enabled database can be built on top of an existing object-oriented or relational/object-relational database management system. This type can work and has been successfully implemented but ultimately does not harness the full power of XML. XML's semistructured data model does not map well to existing highly structured data storage systems. Using object-oriented approach to store XML documents seems well-suited to complex data like XML. It can manage hierarchies/trees structure. Besides the object-oriented approach, a relational/object-relational approach is also proposed to store and manage XML data. Relational/object-relational database seems to be

Corresponding Author: Amin Y. Noaman, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Kingdom of Saudi Arabia.

appropriate for storing XML documents. Such integration would provide several advantages: maturity, stability, portability, scalability and seamlessly querying data represented in XML documents and relations [1].

The second type of XML databases named native XML database, is simply one that was designed from the ground up to store XML documents. It contains data structures to maintain the hierarchical structure of XML data and uses its knowledge about them to optimize query processing. It might make use of a preexisting technology such as object-oriented data storage techniques, but its mission is to store, retrieve and update XML documents. Native XML databases accept and manage XML-based data in its native form. So, there is no need for any mapping between the database and the structure of an XML document. As a result of familiarity with object-oriented databases and relational/object-relational databases, users understand their behavior, especially with regard to performance. Since there is no conclusive proof that either XML-enabled relational database or native XML database is an efficient means of storing XML data and also there is little known performance results in regard to native XML databases, there is a need to compare the performance of both XML databases regarding to the query processing efficiency to come up with results that help to decide which type outperforms the other. To test the performance of the database engine, the time for XML document reconstruction from the database and the time required to import from external XML sources into the databases were measured in this paper. With all the experimental results combined together, we concluded that the native XML database needs more disk space to store both data and index than the XML-enabled database and that native XML database is better than XML-enabled database for handling the larger data size, the latter is good for handling smaller data sets.

Various Issues of XML Query Language: In this section, we shall discuss different issues of XML query language such as: the need of XML query languages, what makes it different from traditional database query language such as SQL and what are the additional features this query language should possess.

XML Query Language Against SQL-Based Query Language: The main reason for using an XML query language instead of a SQL-based query language is that data in XML fundamentally differs than data in traditional models [2] in view point of the following:

Data Structure: XML data is "nested"; it has hierarchical, tree-like structure. In contrast, relational data is "flat" and it is organized in the form of a two dimensional array of rows and columns. Moreover, the structure of XML data is not as rigid as that of relational data. Its structure is unpredictable and irregular. Querying such data and getting the desired result is quite complex compared to querying the data having a fixed structure. This kind of queries called, XML queries or semistructured queries.

Depth of Nested Data: XML data is nested and its depth of nesting can be irregular and unpredictable. Relational databases can represent nested data structures by using structured types or tables with foreign keys, but it is difficult to search these structures for objects at an unknown depth of nesting. In XML, on the other hand, it is very natural to search for objects whose position in a document hierarchy is unknown.

Metadata: In XML, the metadata (information that describes the structure of the data) is distributed throughout the data itself in the form of tags rather than being separated from the data. Relational data, on the other hand, is such that every row of a table has the same columns, with the same names and types. This allows metadata to be removed from the data itself and stored in a separate catalog. In XML, it is natural to ask queries that span both data and metadata.

Inapplicable Values: Because of its regular structure, relational data is "dense"- that is, every row has a value in every column. This gave rise to the need for a "null value" to represent unknown or inapplicable values in relational databases. XML data, on the other hand, may be "sparse". Since all the elements of a given type need not have the same structure, information that is unknown or inapplicable can simply not appear. This flexible nature of XML puts in front a query requirement that poses a query with optional predicates.

Ordering: In a relational database, the rows of a table are not considered to have an ordering other than the orderings that can be derived from their values. XML documents, on the other hand, have an intrinsic order that can be important to their meaning and cannot be derived from data values. This has several implications for the design of a query language. It means that queries must at least provide an option in which the original order of elements is preserved in the query result. It means that facilities are needed to search for objects on the basis of their order.

XML Query Functionalities: XML query language functionalities were addressed in a comparative analysis of XML query Languages and listed as "must have/should have" in the requirements published by the W3C XML Query Language working group [3]. Pointes below enumerate all these requirements:

Supported Operations: The XML query language MUST support operations on all data types represented by the XML Query Data Model.

Text and Element Boundaries: Queries MUST be able to express simple conditions on text.

Universal and Existential Quantifiers: Operations on collections MUST include support for universal and existential quantifiers (∃, ∀ and ~).

Hierarchy and Sequence: Queries MUST support operations on hierarchy and sequence of document structures.

Combination: The XML query language MUST be able to combine related information from different parts of a given document or from multiple documents.

Aggregation: The XML query language MUST be able to compute summary information from a group of related document elements.

Sorting: The XML query language MUST be able to sort query results.

Composition of Operations: The XML query language MUST support expressions in which operations can be composed, including the use of queries as operands.

NULL Values: The XML query language MUST include support for NULL values.

Structural Preservation: Queries MUST be able to preserve the relative hierarchy and sequence of input document structures in query results.

Structural Transformation: Queries MUST be able to transform XML structures and MUST be able to create new structures.

References: Queries MUST be able to traverse intra- and inter-document references.

Identity Preservation: Queries MUST be able to preserve the identity of items in the XML Query Data Model.

Operations on Literal Data: Queries SHOULD be able to operate on XML Query Data Model instances specified with the query ("literal" data).

Operations on Names: Queries MUST be able to perform simple operations on names, such as tests for equality in element names, attribute names and processing instruction targets and to perform simple operations on combinations of names and data.

Operations on Schemas: Queries SHOULD provide access to the XML schema or DTD for a document, if there is one.

Operations on Schema PSV Infoset: Queries MUST be able to operate on information items provided by the post-schema-validation information set defined by XML Schema.

Extensibility: The XML query language SHOULD support the use of externally defined functions on all data types of the XML Query Data Model.

Environment Information: The XML query language MUST provide access to information derived from the environment in which the query is executed, such as the current date, time, locale, time zone, or user.

Closure: Queries MUST be closed with respect to the XML Query Data Model.

XML Database Systems: The use of XML documents to store and exchange data is becoming increasingly promising. As more and more organizations and companies use XML within their information management and exchange strategies, data management issues like storage, querying, retrieval and manipulation of XML arise. This is the main driving force behind the evolution of XML databases. At this moment, there are mainly two types of XML database systems in order to store XML data: The first is the "XML-Enabled database" which is usually relational/object-relational database that has been extended to hold XML data. The second is the "Native XML database" which is specifically designed for storing and querying XML documents. This paper will focus mainly on managing XML documents using XML-enabled database and native XML database.

An XML database is a new kind of database that is designed for storing, accessing and manipulating XML documents, regardless of how it achieves this. Native XML database and XML-enabled database are both considered as XML databases but with different names. If the XML is not stored internally as XML, it is called an XML-enabled database. If an XML document is stored as XML internally, then it is called a native XML database. An XML database is defined in [4] as a collection of XML documents and their parts, maintained by a system having capabilities to manage and control the collection itself and the information represented by that collection. It is more than merely a repository of structured documents or of semistructured data. As is true for managing other forms of data, management of persistent XML data requires capabilities to deal with data independence, integration, access rights, versions, views, integrity, redundancy, consistency, recovery and enforcement of standards. The technical requirements mentioned in [5] to support this kind of databases are:

- It must provide the basic functionalities, such as a common query language (e.g. SQL, XPath), ACID functionality (Atomicy, Concurrency, Isolation, Durable), administrative tools, backup and recovery etc.
- It must provide Create, Read, Update and Delete (CRUD) functionality. Query languages like XPath and XQuery doesn't provide this functionality.
- It should support the management of schemas (DTD or XML Schema) to define the data structure and the validation of input according to those schemas.
- Data integrity mechanisms such as primary and foreign key constraints are absolutely required.
- Strong indexing mechanism must be provided.
- Support for common XML-based APIs (DOM. SAX, COM or Java based) in order to manipulate data.
- Programmatic support for connecting to legacy systems or proprietary interfaces, e.g. SAP.
- Must provide simplified integration with transformation and transport utilities.
- Little database maintenance (providing basic functionality for database maintenance, e.g. exporting, importing, backup, re-indexing etc).

Normally XML documents will be seen as just a file containing a collection of data, which are transferred from one system to another. So, on the first sight there is no reason to store XML in a database. However, there are two main reasons for adopting databases to store XML documents:

- XML itself is not a database, but the other XML-based technology around it and XML itself creates a database like environment [6]. On one side, this environment provides many features found in databases like storage (XML document), schemas (DTDs, XML Schemas), query languages (XQuery, XML-QL, XPath, XQL), programming interfaces (SAX, DOM and so on). On the other side, this environment lacks many of other features like indexing, security, transaction, multi-user access, triggers, backup and recovery management which belong to Database Management System (DBMS) according to [7]. With the growing use of XML documents in data exchange among organizations and in making large Web sites, demands persistent storage mechanisms for XML documents. XML documents have to be managed in an efficient way so that they are available for transfer at any time without the need of conversion. They also should be available for querying and analysis. This is one of the main reasons behind the evolution of XML databases.
- Many of the actual interactions of either business to business (B2B) or business to consumer (B2C) will be conducted via XML messages (SOAP-based Web services, synchronous ebXML message etc.). So those orders, cancellations, credit checks, requests for quotations, invoices, etc. are documents that are the electronic equivalent of paper business documents. Such documents may be generated from data in a RDBMS, but once produced they must maintain a different conception of "integrity". The document must reflect the snapshot of reality that produced it, even if "reality" changes. So for legal and documentation reasons it will be better to store the XML "snapshot of reality" also in a database. Beside it will be easier to analyze and audit operations based on the unified XML view than the fragmented transactions in the diverse back-end systems [8].

Current XML databases are divided into two main types: XML-enabled databases and native XML databases. The idea behind the first type is to use existing databases to store and manage XML documents.

Using existing databases and their products to store XML provides several advantages even in this type XML will not be stored in its native form [9]. First, relational/object-relational or object-oriented databases are well known and are in the database industry for quite a long time. Second, users are familiar with these databases and with their performance. Thirdly, the traditional databases are considered a safe choice by the corporate and they hesitate to switch to new technology suddenly. Relational databases were among the first that wanted to disclose their data as XML. Storing and managing XML documents in relational databases needs mapping the hierarchical, tree-type structures to relational structure. Therefore the first effort was directed towards enabling or extending the capabilities of these databases to incorporate XML. This effort gave birth to the so-called "XML-enabled databases". Storing and managing XML natively is adopted by the second type, the XML community initiated this effort and thus developed the so-called "native XML databases". A native XML database is often considered as a database being built from scratch for the specific purpose of storing and querying XML documents. In such databases, the mapping between XML and the database is not required since it stores XML as it is.

Approach of Investigating the Performance of XML Database System: Since there are two types of XML database system to store and manage XML documents, there is a need to compare the query processing efficiency of the two DBMS in order to judge about their future chances. To meet this need, one database management system from each group is selected and then its query processing efficiency is compared against that of the other one within a detailed benchmark tests. Two well-known commercial database systems: Oracle 10g release 1 from Oracle Corporation and Tamino XML Server 4.1.4.1 from Software AG systems are chosen to represent each group. A standard benchmark XMark is used to provide a better representative performance evaluation between the two XML database systems. The XML DB chosen for experimentation in this paper are Oracle 10g release1 [10] to study XML enabled databases and Tamino XML Server 4.1.4.1 [11] to study the native XML databases. The focus of this section is to describe the two database systems that were chosen to benchmark and outline the reasons why they were chosen. In general, the main criteria on selecting such commercial database systems is the popularity of these systems, since the customer can rely on a familiar vendor, its technology, licensing and support.

Oracle 10g Release 1: Oracle 10g was chosen because Oracle claims it was one of the first, if not the first, relational database to incorporate XML support. Also, Oracle 10g offers a number of solutions to manage XML data. Moreover, Oracle relational features are familiar, which helps in understanding the database in shorter time. All these reasons compelled us to choose Oracle 10g for this article. Oracle 10g is a complete database management system developed especially for handling the relational data. The growing use of XML led Oracle to introduce a new technology to accommodate XML data into its database management system. This new technology is known as Oracle XML DB. Oracle XML DB is a set of storage and retrieval technology designed especially to handle XML. Oracle XML DB fully absorbs the W3C XML data model into an Oracle 10g database and provides new access methods for navigating and querying XML [12]. The main idea behind this technology is to process XML and at the same time provide SQL access to the same data. Oracle XML DB is not a separate XML database management system. It is just a layer on the top of Oracle 10g database. The main features of Oracle XML DB are listed below that let us use it in our comparative study:

- One of the main features of Oracle XML DB is the XMLType data type. This is a predefined object type that can store an XML document. Like any object type, XMLType can be used as the data type of a column in a table or view. The latter usage is significant because it allows any data, relational or XML, to be viewed as a "virtual" XML document.
- Provide support for forthcoming ISO SQLX standard extensions to SQL for constructing XML in SQL including the operators XMLElement(), XMLAttributes(), XMLForest(), XMLAgg() and XMLConcat().
- Full W3C XML Schema 1.0, XPath 1.0, XSLT 1.0 and DOM Core support for native XMLType data type implemented deep inside the engine and exposed to SQL, PL/SQL and Java API's. Users can augment built-in processing of XML documents with stored procedures, functions and triggers as for any other types and tables.
- Automatic object-relational storage for XML documents based on XML schema, with optional fine-tuning of the mapping via schema Annotations (already supported in XML Spy 4.3 tool). Includes optional support for full DOM fidelity and mixed content storage.

- SQL extensions for XPath-based extract(), extractValue(), existsNode() and updateXML() operators for manipulating documents, including the ability to rewrite a subset of XPath expressions to use underlying object-relational indices and full-text indices for maximum performance.

With regards to ORACLE XML DB Queries; Standard SQL has been modified to support XPath queries. Functions such as existsNode(), extract(), extractValue() and XMLSequence() have been added which support XPath expressions to search the XML document. Other functions such as getClobVal(), getStringVal() and getNumberVal() helps to extract CLOB, String, Number from an XML Fragment [13]. Some of these functions will be used through the experiments.

Tamino XML Server 4.1.4.1: Tamino XML Server is the leader in the XML DB market, as it is the first vendor to market a native XML database. A recent study conducted by the IDC showed that Software AG's Tamino XML Server currently held 47.5 percent of the XML DBMS market in 2001 [14]. The remaining market is shared by rest of the vendors. Once the decision was made to use a native XML DBMS in this paper, the decision to use Tamino XML Server usually follows quite easily. Software AG is the most experienced vendor in the field, with a host of independent product endorsements and awards and the capability to deliver the services that turn the technology into business success around the world. Listed below, some ideas realized in Tamino XML DB and considered as useful features [11]:

- High-performance and native XML storage to keep electronic documents stored in Tamino intact for the long-term use of transient business documents.
- Storage of all types of data, Tamino XML Server accepts well-formed XML documents and any other type of non-XML data for storage, including office documents, audio, video, PDF files, etc. There is no need to create schemas before content can be stored.
- Open standards support (HTTP, XML, XQuery, XPath, XML Schema, Namespaces ...) to provide maximum flexibility in application design and to avoid vendor lock-in.
- Backup, replication and clustering support features to serve enterprise-level high-availability, reliability, transactionality, security and performance.
- First commercial W3C XQuery implementation allows users to query for content and to compose documents on the fly, with output from many documents.
- Services for accessing external databases and applications help to consolidate data from various sources (including external databases and applications) in one place.
- Many APIs and development tools, a number of services supporting J2EE and. NET, enabling developers to quickly create applications, solutions or products around Tamino.
- Multi-platform availability (Windows, Solaris, Unix, Linux) for usage in smaller applications, as well as in enterprise-scale applications.
- Tamino provide a Web based interface for administration gives flexibility in a great extend.
- Tamino supports UTF-8 and UTF-16 encoding systems. If an incoming XML document has another encoding like ISO-8859-1, Tamino converts it to unicode before storing. Analogously, a user can specify the encoding when retrieving data.
- There are some new features introduced in Tamino XML Server. They are Tamino X-Node and Tamino X-Tension. Tamino X-Node provides access to external relational database as well as to Software AG’s Adabas. Tamino X-Tension provides a mapping function that may be required for accessing external data sources.

To retrieve the data from the database, Tamino uses the query language X-Query the query language of Tamino and a successor of XQL query language. X-Query is developed by the Software AG itself and still under improvement. X-Query is not related to XQuery, the query language of the W3C. It is based on XPath specification of the W3C, but has extended XPath to make it possible to perform content-based retrieval. Tamino also allows user-defined function to be added to the query through a feature called X-Tension. One of the drawbacks of X-Query appears while querying the results of the joins. Querying the results of the joins can be done using variables. So the join could be executed by two successive queries. The first is used to store results like ID's or general values of elements and attributes and the second could query a semi-join construct, specified by the variable. But, the problem here, variables are not supported, by Tamino X-Query. When a query is sent to a Tamino database, first it transforms it into unicode,

Table 1: XMark benchmark specification

Document	Single-Doc
Queries	20 XQuery Comprehensive queries
User	Single-user
Database Type	Auction Site
Update	No
Bulkload	No
Metric	Time
Generator	Xmlgen
Text used from	Shakespeare's Plays
XML document Class	Generally it is a data centric document but with some elements makes up the document centric side of the document

parses it and checks for syntactical correctness. In the following optimization step, it can match the query against the associated schema definition. It then selects the appropriate index for the evaluation of the query; selects the documents with the use of the index and remaining parts of the query are evaluated on the result. The result of a query in Tamino is a well-formed XML document. The query is normally sent as an http request, i.e. it requires that a Web server is running on the same machine where the Tamino database server is installed.

Xmark Benchmark: XMark [14] stands for "XML Benchmark" is a well-known benchmark for XML data management and has been developed at the CWI, which is the Research Institute for Mathematics and Computer Science in the Netherlands. XMark is a single-user, single-document benchmark. It consists of a scalable document database modelling an Internet auction website and a concise and comprehensive set of XQuery queries which covers the major aspects of XML query processing. Although most XML database systems consists of various logical layers and may be physically distributed over a network, Xmark abstract from these system engineering issues. It only concentrates on the query processor and its interaction with the data store. Thus, it doesn't take into account the network overhead and communication costs. XMak bencmark specification is summarized on the Table 1. The main sources of information on the benchmark were the publications by its authors [14-17] and the project's homepage: http:// monetdb.cwi.nl/xml/.

It was decided to use XMark for benchmarking in this paper and this decision was based on several reasons:

- A complete benchmark framework was already available for free download. The XMark team provided software for generating the XML data and the benchmark's queries but no software to measure the time per operation was provided.
- XMark attempts to cover the major aspects of XML query processing ranging from small to large document and from textual queries to data analysis and ad hoc queries [15].
- XMark provides a data generation tool that allows for efficient generation of XML documents of different sizes ranging from small to very large. The data generation tool is available on the benchmark's homepage: http:// monetdb.cwi.nl/xml/ downloads.html. It is written in C, so it can be compiled and used on every system with a C compiler. To make it platform-independent it does not rely on supplied random number generators (RNG) but instead comes with its own RNG routine.
- XMark offers 20 queries; each query is intended to challenge a particular aspect of the query processor. The queries are available for download, too, but they are ready to use if the tested system is already capable of executing XQuery code.
- The XMark generated documents are modeled after a database as deployed by an Internet auction site, a typical e-commerce application.
- XMark is quite easy to understand for users who are acquainted with the matter.
- XMark used for benchmarking on several researches. Recently, it has been used to benchmark Monet XML which is a research developed at the CWI and the project's homepage: http://monetdb.cwi.nl/ Home/index.html.

Xmark benchmark takes on the challenge and features a tool kit for evaluating the retrieval performance of XML stores and query processors: a scalable document database and a comprehensive set of queries [14].

EXPERIMENTAL RESULTS AND DISCUSSION

An experimental study on the query processing efficiency of an XML-enabled database system and a native XML database system on a selected set of XMark

benchmark queries will be presented in this section. The experiments are conducted on two XML database systems: Oracle and Tamino using interfaces based on JDBC and HTTP with the help of XMark benchmark. The evaluation criteria used for the experiments is the query execution time. All the details about experiments: environment, architecture, methodology and challenges will be covered in this section. At the end of this section, the result of running a selected set of queries on both XML database systems will be discussed. Generally speaking the native XML database system performed better. To provide representative benchmark testing results with two different XML database systems, all benchmark tests are performed under the same conditions. All benchmark tests are performed with the same set of XML documents and the same queries. For the benchmark testing, five different documents with different sizes were generated. For each document size; a couple of document copies are generated so it can be loaded with the two different XML database systems. The same document was loaded into each XML database system. Each query was executed five times on each document. After getting the results, the maximum and the minimum result value are dropped and the mean average is calculated of the remaining three values. This mean average represents the result of the query execution time on that document. Furthermore all benchmark tests are executed on the same machine using the same version of the Operating System to avoid fakes of test results caused by different environmental settings.

The proposed experiments architecture for the benchmark testing is shown below in Fig. 1. The experiments methodology section will explain all the steps shown in the figure below.

The experiment methodology passes through the following steps:

- Generating the XML test documents.
- Preparing both XML DB.
- Loading XML documents into both XML DB.
- Preparing the test queries for both XML DB.
- Measuring the query execution time for both XML DB.

For the purpose of the experiments, a subset of queries from the selected benchmark will be chosen for testing. The same queries used for both Oracle and Tamino. The covered queries are: query#1, query#2, query#5, query#6 and query#8. The chosen queries are a part of XMark project. These queries were adopted to comply with XQuery specification [18]. It is obvious that different storage models will require different query processing techniques. For example, if a relational/object-relational database is used to manage XML documents, queries in XPath or XQuery need to be converted to SQL queries, whose processing will be handled by the underlying database. For this reason all the benchmark queries will be translated to equivalent SQL queries involving the XML DB functions. (eg: XQuery to SQL) so it can be used with Oracle. To run the XMark queries

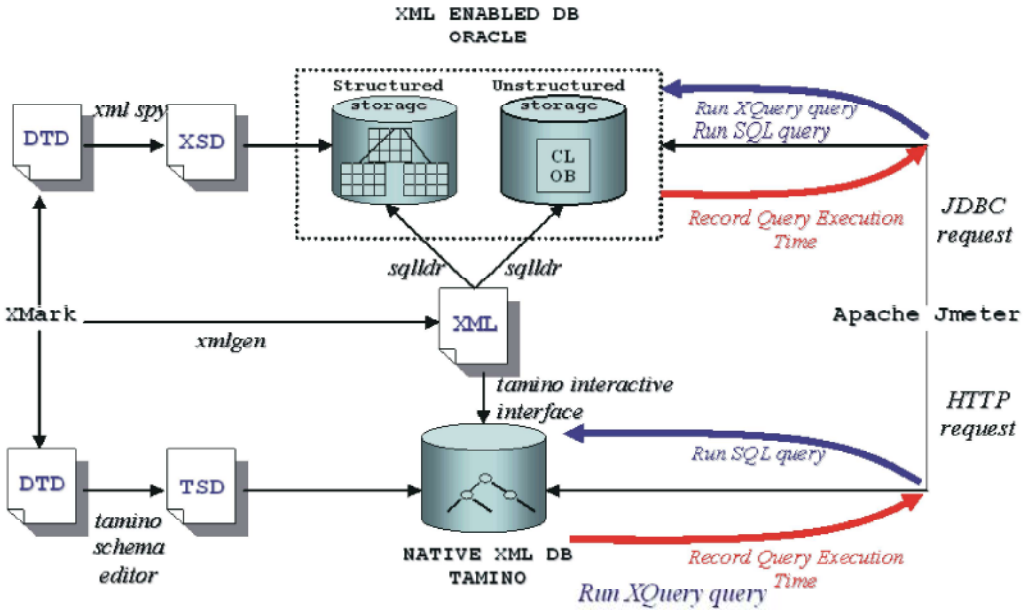


Fig. 1: The Proposed Experiments Architecture

in Oracle, SQL queries was used in SQL-Plus environment using a select clause while in Tamino, the new XQuery (based on the up coming W3C recommendation) was used for querying the stored XML document. The query is normally sent as an HTTP request via MS Internet Explorer 6.0 browser. Note that, retrieval speeds will depend upon two important characteristics: *the structure of the XML documents containing the data* and *the nature of the query used to retrieve the data*.

Measuring the Query Execution Time for Both XML DB: Query execution time is difficult to achieve in a representative way because each XML DB has a different mechanism to compute it. To assist in testing the query execution time of the different XML DB under various types of loads, open source application called JMeter [19] from the Apache Software Foundation was used. Stefano Mazzocchi from the Apache Software Foundation developed JMeter as a Java desktop application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions (FTP, Java, SOAP/XML-RPC, JDBC). JMeter was used in these experiments to analyze query performance for both XML DB.

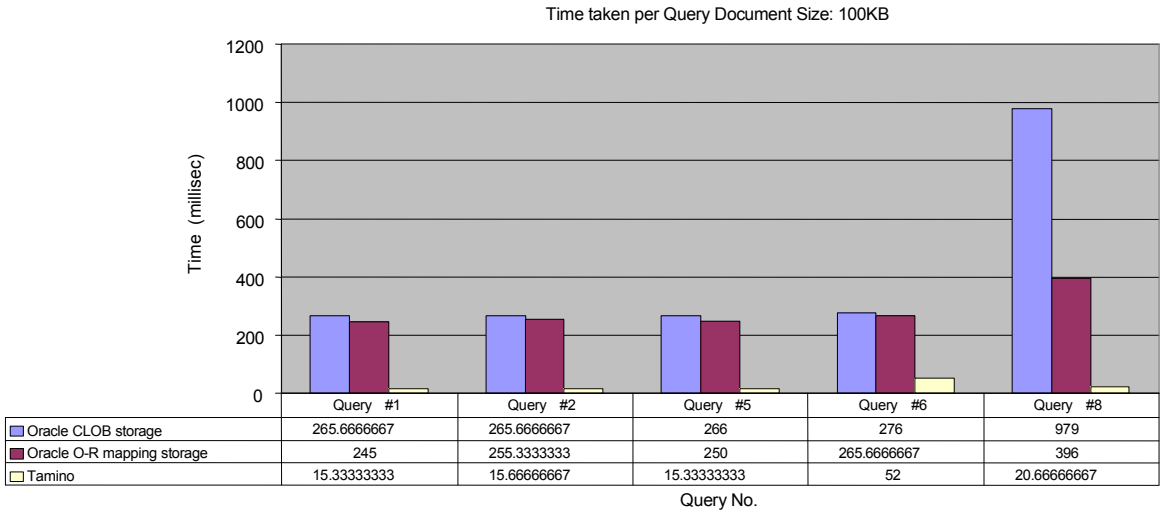
With Oracle XML DB- Sending JDBC Request: Running Oracle queries for both storage approaches are implemented using JDBC requests. Sending JDBC request needs first to specify:

- The right URL and the database: jdbc: racle: hin: localhost:1521:orcl
- The right driver class: oracle.jdbc.OracleDriver
- he right username/password: scott/****

- With Tamino XML DB- Sending HTTP request.
- unning Tamino queries are implemented using HTTP requests. Tamino service is bound to /tamino in the Microsoft IIS webserver running on localhost.

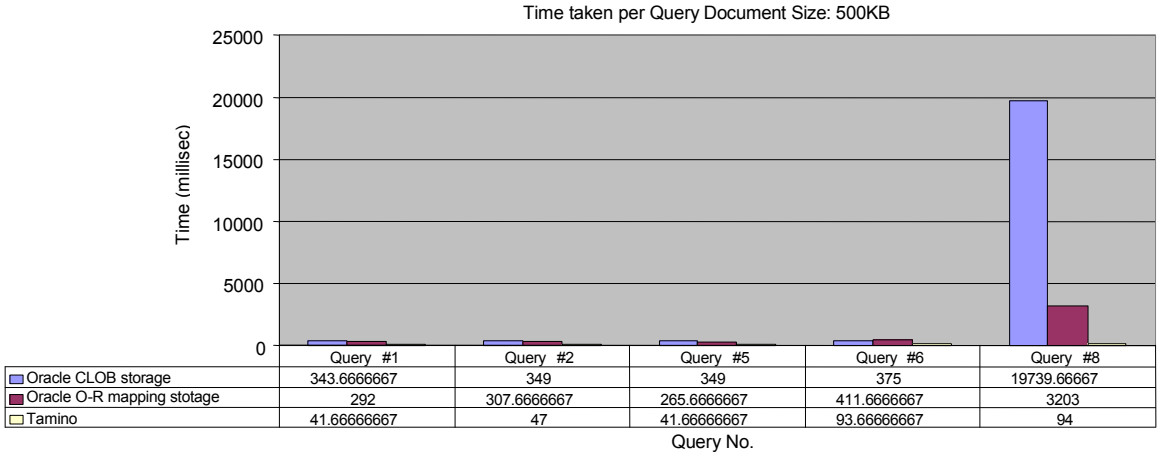
Benchmark Testing Challenges and Limitations: uring the experiments, some practical challenges and limitations were encountered like:

- The hardware used for the experimentation was not adequate to run extensive tests and monitor a number of performance parameters.
- The XMark queries were rewrote and converted from XQuery to SQL manually. And with complicated XQuery queries, it was really hard to make the conversion to SQL queries.
- X-Query, the query language provided by Tamino, is not powerful enough. X-Query is not related to XQuery of the W3C. In particular, XPath, a component of X-Query, is not fully integrated. For example building join expressions with X-Query is only possible in a restricted way. For this reason, all the queries were rewritten for the second time using the new XQuery based on the up comming W3C recommendation, since this limitation with X-Query wasn't clear in the beginning, it consumed more time to realize and to switch to the new XQuery.
- Large XML documents with 100MB and 1GB sizes couldn't be used in the experiments due of a loading failure while using SQL*loader.
- An evaluation version of Tamino was used through the experiments (Tamino considered an expensive product) and this version allows only to store up to 20MB of data.
- In Oracle with structured/object-relational mapping storage, No annotation was provided with the schema, so default mapping was performed and in this case the system will make certain default choices that may or may not be optimal.
- Considerable time was spent in discovering the namespace declarations required in the XML schema document for registering into the system and the namespace declarations in the XML document to be loaded.
- There are very few benchmarks available for XML DB benchmarking and most of them require support of XQuery which is not supported with relational/ object relational databases.
- Oracle XML DB uses SQL with XPath expression. The query techniques have moderate expressive power.
- Communication costs can have a big effect on query performance and may obscure the true performance of a database. For this reason most database benchmarks try to ignore these infrastructure issues as much as possible and produce results independent of differences in communication protocols, whether they are HTTP, JDBC, or ODBC. These issues have something to do with the implementation of the database.



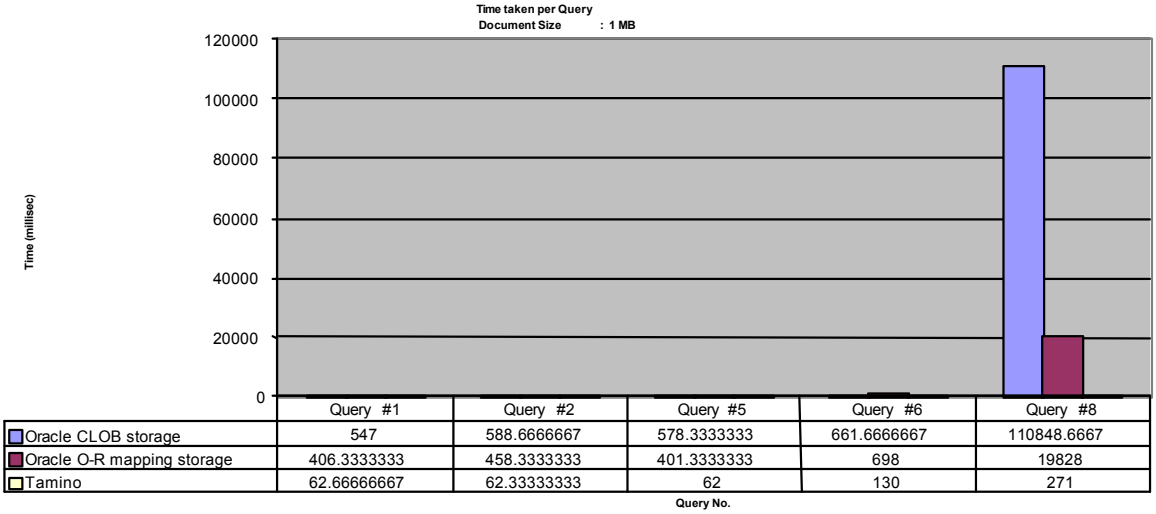
	Query #1	Query #2	Query #5	Query #6	Query #8
Oracle CLOB storage	265.6666667	265.6666667	266	276	979
Oracle O-R mapping storage	245	255.3333333	250	265.6666667	396
Tamino	15.33333333	15.66666667	15.33333333	52	20.66666667

Fig. 2: Query execution times of different queries on an XML document with size=100KB



	Query #1	Query #2	Query #5	Query #6	Query #8
Oracle CLOB storage	343.6666667	349	349	375	19739.66667
Oracle O-R mapping stotage	292	307.6666667	265.6666667	411.6666667	3203
Tamino	41.66666667	47	41.66666667	93.66666667	94

Fig. 3: Query execution times of different queries on an XML document with size=500KB



	Query #1	Query #2	Query #5	Query #6	Query #8
Oracle CLOB storage	547	588.6666667	578.3333333	661.6666667	110848.6667
Oracle O-R mapping storage	406.3333333	458.3333333	401.3333333	698	19828
Tamino	62.66666667	62.33333333	62	130	271

Fig. 4: Query execution times of different queries on an XML document with size=1MB

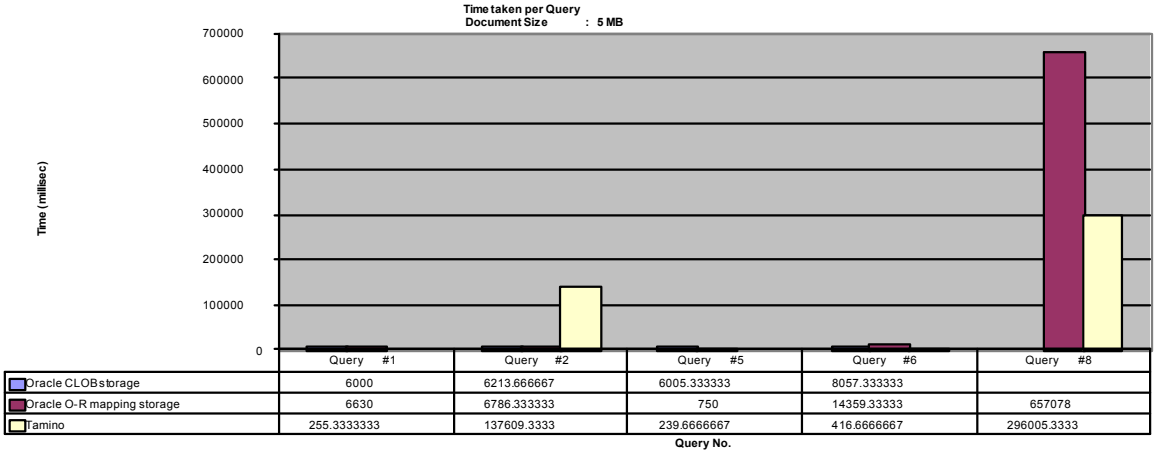


Fig. 5: Query execution times of different queries on an XML document with size=5MB

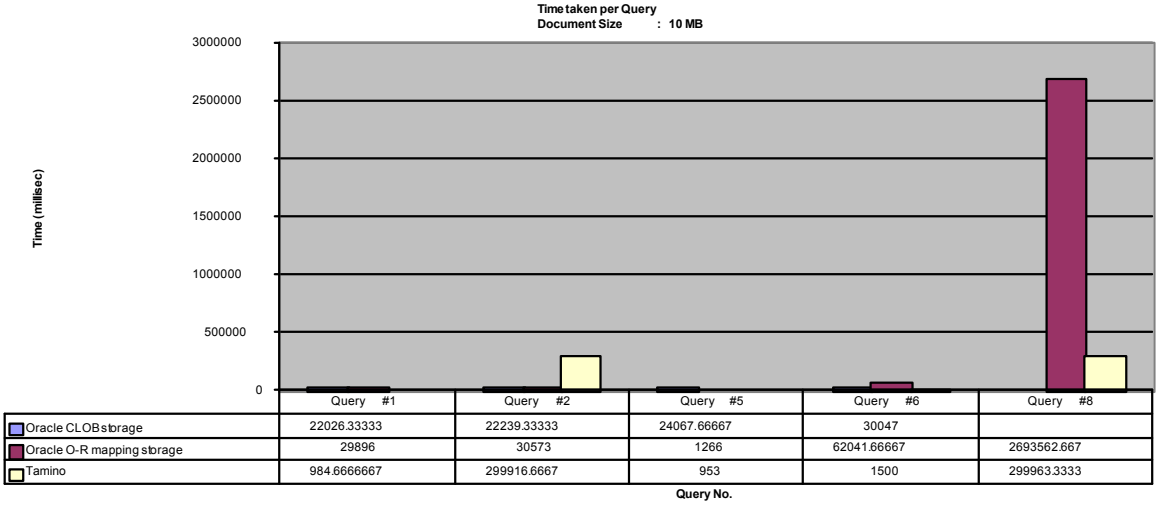


Fig. 6: Query execution times of different queries on an XML document with size=10MB

Table 2: A summary of the main observations on the query complexity view

Size	Tamnio vs. Oracle CLOB	Tamnio vs. Oracle O-R mapping	Oracle CLOB vs. Oracle O-R mapping
100KB	Tamino outperforms Oracle CLOB by a factor of 20.8 times.	Tamino outperforms Oracle O-R mapping by a factor of 14.5 times.	Both Oracle storages have almost a similar performance for all queries except query#8, Oracle CLOB performs poorly in query#8.
500KB	Tamino outperforms Oracle CLOB by a factor of 47.6 times.	Tamino outperforms Oracle O-R mapping by a factor of 11.6 times.	
1MB	Tamino outperforms Oracle CLOB by a factor of 88.3 times.	Tamino outperforms Oracle O-R mapping by factor of 19.7 times.	
5MB	Tamino outperforms Oracle CLOB by a factor of 22.6 times after excluding query#2 and query#8. Both Oracle storage approaches outperform Tamino in query#2	Tamino outperforms Oracle O-R mapping by a factor of 16.4 times after excluding query#2.	Both Oracle storages have almost a similar performance for only query#1 and query#2.
10MB	Tamino outperforms Oracle CLOB by a factor of 22.5 times after excluding query#2 and query#8. Both Oracle storage approaches outperform Tamino in query#2.	Tamino outperforms Oracle O-R mapping by a factor of 20.5 times excluding query#2.	Both Oracle storages have almost a similar performance for all queries except query#5 and query#8.

Benchmark Testing Results: This section presents the results of executing the chosen queries on the chosen databases. Following the results there is a summary of the main observations of the benchmark testing results. Charts and graphs from Fig. 2 to Fig. 6) are used throughout this section to aid the reader. The results are presented from point of view: the query complexity. On the query complexity view, the charts show how the query type affects the query execution time.

Based on the results obtained from the above charts, a list of the main observations is displayed below, followed by a summary of them on Table 2:

For XML Document with Size=100KB Shown in Fig. 2:

- In general, Tamino outperforms both Oracle storage approaches in all queries. It outperforms Oracle with unstructured/CLOB storage by a factor of 20.8 times and 14.5 times for Oracle with structured/object-relational mapping storage. By average it outperforms Oracle with a factor of 17.7 times.
- The result shows that the performance of both Oracle storage approaches was similar for all queries except query#8. Oracle with unstructured/CLOB storage performs poorly in query#8 (chasing references query).

For XML Document with Size=500KB Shown in Fig. 3:

- Tamino yields the best performance. It outperforms Oracle with unstructured/CLOB storage by a factor of 47.6 times and 11.6 times for Oracle with structured/object-relational mapping storage. By average it outperforms Oracle with a factor of 29.6 times. Note that, Both Oracle storage approaches show acceptable executing time for query#1, query #2, query #5 and query #6, but it is getting quite high after executing query#8. Tamino outperforms both Oracle storage approaches with a factor of 6.5 times (after excluding query#8).
- In general, the performance of both Oracle storage approaches was similar for all queries except query#8. Oracle with unstructured/CLOB storage performs poorly in query#8.

For XML Document with Size=1MB Shown in Fig. 4:

- Tamino outperforms both Oracle storage approaches in all queries. In general, it outperforms Oracle with unstructured/CLOB storage by a factor of 88.3 times and 19.7 times for Oracle with structured/object-relational mapping storage. By average it outperforms Oracle with a factor of 54 times. Note that, both Oracle storage approaches show average performance while executing query#1, query#2, query#5 and query#6. Tamino outperforms both Oracle storage approaches in this set of queries by factor of 7.2 times.
- Both Oracle storage approaches have an average and similar performance for all queries except query#8. In query#8, both storage approaches yield a bad performance although Oracle with structured/object-relational mapping storage shows better performance comparing to Oracle with unstructured/CLOB storage.

For XML Document with Size=5MB Shown in Fig. 5:

- Tamino outperforms both Oracle storage approaches in all queries except query#2. It outperforms Oracle with unstructured/CLOB storage by a factor of 22.6 times (after excluding query#2 and query#8) and 16.4 times for Oracle with structured/object-relational mapping storage (after excluding query#2). By average it outperforms Oracle with a factor of 19.5 times. Tamino performs poorly in query#2 comparing to both Oracle storage approaches. The query execution time for query#2 in Tamino is the first one to provide surprises in the experiments. Both Oracle storage approaches outperform Tamino by factor 21.2 times in query#2.
- The performance of both Oracle storage approaches was similar for query#1 and query#2. Oracle with unstructured/CLOB outperform Oracle with structured/object-relational mapping in query#6. But it has poor performance in query#5 and has no results in query#8. (Query#8 could not be run with the given configuration and system software).

For XML Document with Size=10MB Shown in Fig. 6:

- Tamino shows interesting results, it was the slowest to execute query#2, but it executes other queries relatively faster than both Oracle storage approaches. It outperforms Oracle with unstructured/CLOB storage by a factor of 22.5 times (excluding query#2 and query#8) and 20.5 times (excluding query#2) for Oracle with structured/object-relational mapping storage. By average it outperforms Oracle with a factor of 21.5 times.

- The performance of both Oracle storage approaches was almost similar although Oracle with unstructured/CLOB storage performs slight better than Oracle with structured/object-relational mapping storage in query#1, query#2 and query#6. While Oracle with structured/object-relational mapping storage outperform Oracle with unstructured/CLOB storage in query#5. (Query#8 could not be run with the given configuration and system software).

CONCLUSION AND RECOMMENDED FUTURE WORKS

In this paper, it was shown that there are different storage options to store XML data in databases, each having its own merits and demerits. XML data can be stored in XML-enabled relational databases and native XML databases. XML-enabled relational databases use two approaches for storing XML, shredding it into relational/object-relational tables and by keeping it as a whole in a single column of the relational table that has CLOB/BLOB or VARCHAR data type. On the other hand, native XML databases store and manage XML data in its native form. None of the three options represents the single best choice for handling XML documents. The provision of different storage options provides the developer and the designer with a possibility to choose the suitable storage option depending on the structure and the size of the XML documents that needs to be stored and the type of queries that should be managed by the XML DB.

We have discovered that there are good reasons to use either XML-enabled relational databases or native XML databases, depending upon the needs of your particular XML applications. XML-enabled relational databases are generally ahead of native XML databases in regards to data integrity, query capability, concurrency and transaction control, standardization and administration. While native XML databases have matured a great deal in the past few years, these areas are still in need of improvement. On the other hand, native XML databases offer better performance and flexibility especially when dealing with large XML documents. While neither of the XML-enabled relational databases two approaches works well at all; but, most business applications do not deal with XML data alone. They have existing relational data and also continue to produce relational data and native XML databases still need more time to become a comparable alternative to relational databases. Through this paper, we notice that XMark is relatively a simple benchmark and doesn't test data insert, update and delete operations. So, future research will look into using other benchmarks that allow comparing the performance of both XML DB types using data insert, update and delete operations. Also, more experiments with different kinds of document-centric and data-centric XML documents are required.

REFERENCES

1. Shanmugasundaram, J., K. Tufte, G. He, C. Zhang, D. DeWitt and J. Naughton, 1999. Relational Databases for Querying XML Documents: Limitations and Opportunities, in: Proceedings of the 25th International Conference on Very Large Data Bases Conference (VLDB'99), Edinburgh, Scotland, September 7-10, 1999, 302-314.
2. Chamberlin, D., 2003. Influences on the Design of XQuery, In: H. Katz, D. Chamberlin, D. Draper, M. Fernandez, M. Kay, J. Robie, *et al*., XQuery from the Experts: A Guide to the W3C XML Query Language, Boston: Addison-Wesley, viewed 13 May 2006, from http://www-128.ibm.com/ developerworks/ xml/ library/x-xqbook.html.
3. Bray, T., J. Paoli and C.M. Sperberg-McQueen, 1998. Extensible Markup Language (XML) 1.0, W3C Recommendation, 10 February 1998, viewed 10 August 2005, from http://www.w3.org/TR/1998/REC-xml-19980210.
4. Salminen, A. and F.W. Tompa, 2001. Requirements for XML document database systems, in: Proceedings of the 1st ACM Symposium on Document Engineering (DocEng 2001), New York, NY: ACM Press, pp: 85-94.
5. Fohn, R., 2002. XML Database Systems, Powerware Informatik, April 2002, viewed 10 January 2006, from http://www.powerware.ch/ doc/XMLDatabases.pdf.
6. Bourret, R., 2005b. XML and Databases, Working Paper, viewed 13 May 2005, from http:// www.rpbourret.com/xml/XMLAndDatabases.htm, Last updated September, 2005.
7. Rob, P. and C. Coronel, 2000. Database Systems: Design, Implementation and Management, 4th ed., Cambridge: Thompson Learning.
8. Champion, M., 2001. Storing XML in Databases, eAI Journal, October 2001, viewed 05 May 2006, from http://www.eaijournal.com/PDF/ Storing XML Champion.pdf.

9. DeJesus, E.X., 2000. XML Enters the DBMS Arena, Computerworld, October 2000, viewed 10 January 2006, from http://www.computerworld.com/ news/ 2000/story/0,11280,53026,00.html.
10. Oracle, 2004b. Oracle XML DB, Oracle Corporation, viewed 12 May 2006, from http:// www.oracle.com/ technology/tech/xml/xmldb/index.html.
11. Tamino, 2004b. Tamino XML Server fact sheets: Concept, Benefits and Features, Software AG, June 2004, viewed 12 May 2006, from http:// www1.softwareag.com/tr/Images/FS_Tamino_Concept-June04_tcm70-5581.pdf.
12. Murthy, R. and S. Banerjee, 2003. XML Schemas in Oracle XML DB, In: Proceedings of the 29th International Conference on Very Large Data Bases (VLDB), Berlin, Germany, September 9-12, 1009-1018.
13. Lee, G., 2005. Mastering XML DB Queries in Oracle Database 10g Release 2, Oracle Corporation, March 2005, viewed 28 October 2006, from http://www.oracle.com/technology/tech/xml/xmldb/Current/TWP_Mastering_XMLDB_Queries.pdf
14. Schmidt, A.R., F. Waas, M.L. Kersten, D. Florescu, I. Manolescu, M.J. Carey and R. Busse, 2001a. The XML Benchmark Project, Technical report INS-R0103, Amsterdam, The Netherlands: CWI, April 30.
15. Schmidt, A.R., F. Waas, M.L. Kersten, D. Florescu, M.J. Carey, I. Manolescu and R. Busse, 2001b. Why and How to Benchmark XML Databases, ACM SIGMOD Record, 30(3): 27-32.
16. Schmidt, A.R., F. Waas, M.L. Kersten, M.J. Carey, I. Manolescu and R. Busse, 2002a. XMark: A Benchmark for XML Data Management, In: Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), Hong Kong, China, pp: 974-985.
17. Schmidt, A.R., F. Waas, M.L. Kersten, M.J. Carey, I. Manolescu and R. Busse, 2002b. Assessing XML Data Management with XMark, In: Proceedings of the First VLDB Workshop on Efficiency and Effectiveness of XML Tools and Techniques (EEXTT2002), Hong Kong, China, Informal Proceedings, pp: 144-145.
18. Boag, S., D. Chamberlin, M.F. Fernndez, D. Florescu, J. Robie and J. Simon, 2005. XQuery 1.0: An XML Query Language, W3C Working Draft, 15 September 2005, viewed 12 December 2005, from http:// www.w3.org/TR/xquery/.
19. Apache J Meter, 2004 Apache JMeter 2.0.2, The Apache Software Foundation, viewed 12 May 2005, http://jakarta.apache.org/jmeter/index.html.