Middle-East Journal of Scientific Research 12 (2): 182-194, 2012
ISSN 1990-9233
© IDOSI Publications, 2012
DOI: 10.5829/idosi.mejsr.2012.12.2.71012

Towards Achieving an Optimum Performanceof XML Data into Both Types of XML Databases: XML-Enabled Databases and Native XML Databases

Amin Y. Noaman and Amal Almansour

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

Abstract: EXtensible Markup Language (XML) promises to be the standard language for data representation in e-business, particularly when that data is exchanged over or browsed on the Internet since it is nested and having a self-describing structure that provides a simple yet flexible means for business applications to model and exchange data. There are two alternative database types used for of storing and retrieving XML documents: native XML databases and XML-enabled relational databases. The main objective of this paper is to recognize the optimal situations on whether to use a native XML database or an XML-enabled relational database to achieve optimum performance in view point of the time for XML document reconstruction from the database and the time required to import from external XML sources into the databases. After running the required experiments, 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.

Key words: XML · IT · HTML · DBMS · DTD · SQL · Native · Enable · E-Business

INTRODUCTION

The remarkable growth of the World Wide Web in recent years has been enabled by the possibility to cheaply and easily distribute information and applications to users all over the world. As the information on the Web gets more complex and the number of users keeps growing, Web developers and Information Technology (IT) managers become aware of the limitations in current technologies and standards. The limitations came from the fact that the Hyper Text Markup Language (HTML) is wonderful for delivering information over the Web, but it does nothing to facilitate the integration of Web information with existing applications especially business applications.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. A DBMS is a collection of programs that provide a logical view of a collection of information that is independent of its physical storage. A DBMS provides update, access, search, security, concurrency, integrity, high availability and centralized administration to other programs andusers entirely through its logical view.

XML has clearly emerged as the dominant (Meta) language of choice for data interchange, messaging, metadata modeling, linking and annotation. It is likely to be equally as dominant in the areas of information modeling and management and that is having a profound impact on the evolution of database and content management technology. As organizations move to portals and XML-based forms to enhance their business processes, the volume of XML and the need for specialized XML storage beyond a centralized DBMS increases dramatically. Further, the increased reliance on web services to power the enterprise infrastructure and the diversification of specialized content management systems is also heightening the sense of urgency to manage and manipulate XML content throughout the enterprise stack.

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

The ideal customer for an XML database is one who is skilled at assembling components into a whole that is greater than the sum of its parts. There are essentially three categories of customer who are most likely to benefit from an XML-centric database:

- IT Savvy Enterprises who are competent in architecting and building large hybrid commercial solutions
- Application and Solution Providers who add value higher in the solution stack but require powerful XML capabilities.
- System integrators whose value proposition is based on providing integrated business solutions and who want to avoid building complex components that do not directly map to site-specific business needs.

A word of caution; often the ideal XML database consumer is tempted to build their own XML data store or extend another non-XML commercial product to provide XML support.While XML databases are clearly distinct from today's RDBMS systems, they are no less sophisticated. No sane enterprise or technology provider would choose to build an RDBMS rather than license one (other than RDBMS vendors); do not build your own XML database (or support for XML into a non-XML database) unless it is actually your business to do so.

The reasons for storing XML in a database system are the same as for relational data: consistent storage, transactional consistency, recoverability, high availability, security, efficient query and update operations and scalability. These are all features which make a database a much more appropriate repository for XML data than, for example, a file system. Thus, XML databases have gained increasing popularity and importance in recent years. XML databases are the next big wave that has impacts on the business world since the second generation of databases, relational database management systems. It is applicable in a wide range of application areas. High-level overview of how to use XML with databases and how XML is commonly stored in different types of database management systems is provided in [1].The requirements for an effective XML-centric data store dictate that all of the essential capabilities of a full blown DBMS system be made available. The trimming of this XML appliance needs to be accomplished through its singular focus on XML and not by eliminating functionality such as scalable query processing, trusted security, scalability, etc.

Overall View of XML with its Classes, Advantages and Capabilities: Trying to define XML is difficult; similar to trying to decide on one standard definition for e-business. Below are three different definitions of XML:

- XML is used to improve compatibility between the desperate systems of business partners by defining the meaning of data in business documents" [27].
- XML is a language allowing the exchange of structured data" [2].
- Short for eXtensible Markup Language, a specification developed by the W3C, XML is a pared-down version of SGML, designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation and interpretation of data between applications and between organizations" [3].

XML documents contain character data and markup tags. The character data is often referred to simply as content, while the markup tags provides structures for that content. The simplest way to describe the structure of XML is to compare it to HTML which most people are already familiar with. As in HTML, the structure in XML is built up using markup tags. There is however a very important difference between tags in HTML and tags in XML; unlike HTML tags, XML tags have no predefined meaning which means that users are free to define and use tags that best suit the data. XML documents can be broken down into two pieces the header and the content. The header specifies the version of XML specification that the document complies with. The content part of the XML document consists of elements hanging together in a logical tree structure, starting with the root element that contains all the other elements.

In view point of classes of XML documents, we see that most XML documents fall into two main types: document-centric and data-centric [1]. The difference between these types comes from the difference in what they represent. Both have profoundly different structure and characteristics. The type of XML document can have a large impact on the choice of XML DB. The first type is called document centric whichis primarily used for displaying information for humans, having irregular structure, mixed content (intermingled markup and character data) and significant ordering amongst elements, such as in user's manuals, static Web pages and marketing brochures. These documents tend to contain larger grain data and have a less consistent structure from document to document than data-centric XML documents. The second type is named data centric in which XML is used as storage or interchange format for data that is structured, appears in a regular order and is most likely to be machine processed instead of read by a human. Examples of data centric XML are inventory records, stock quotes, scientific data and sales orders. In most cases these documents are produced by programs and consumed by programs. Humans in general do not come in contact with them; they interact with a representation of the data in the XML document, rendered by a program.XML provides many advantages as a data format over others [4], including:

- It is an open source;
- Platform-independent;
- It is fully Unicode-compliant. This means that XML can deal with any character sets like English, Chinese or Arabic without conflict, thus applications that can read XML properly can deal with any combination of any of these character sets. This advantage of XML makes information sharing possible not only between different computer systems but also across different national and cultural boundaries;
- Human readable format makes it easier for developers to locate and fix errors than with previous data storage formats;
- Extensibility in a manner that allows developers to add extra information to a format without breaking applications that where based on older versions of the format; and
- A large number of off-the-shelf tools for processing XML documents already exist.

Since XML is a way to describe structured data there should be a means to specify the structure of an XML document. Document Type Definitions (DTD) and XML Schema languages are different mechanisms that are used to define the structure for XML documents [4]. With the help of a DTD and XML Schema, a user can give a consistent structure to XML documents: a user can define the elements and attributes that can appear in a document, define the number of child elements and the order in which they should appear, define whether an element can include text or is empty, define data types for elements and attributes and so on. An XML document that conforms to the structure and restrictions defined within DTD or XML Schema language is considered to be valid (Bray *et al*. 2004). The sections below will define these two technologies.Document Type Definition (DTD) is normally a separate document used to define the constraints on the XML document. It describes the structure of XML document in terms of elements, attributes and its data types, valid ordering, nesting etc. It is written in a language that does not conform to XML. The DTD is important when exchanging XML documents with other applications. The applications can validate the XML documents with the belonging DTD. But DTD has some major shortcomings like shallow support to data types, no support of namespaces and DTD not being in XML format itself (Lee and Chu 2000a). XML Schema is another emerging standard that takes care of these problems.

XML Query Languages and its Functionalities: More and more data on the Web is being presented in XML format and it is likely that large amounts of tomorrow's data and Web resources will also be available in it. Thus, there is a demand for languages to extract subsets of the data stored within an XML document. A number of languages have been created for querying XML documents including XPath [5], XQuery [6], LOREL [7], XQL [8] and XML-QL [9]. For more details, good surveys of various XML query languages are done in [10] and [11]. The main reason for using an XML query language instead of a SQL-based query language is that data in XML fundamentally differs from data in traditional models [12]:

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. An example of such a query might be "Find all the red things", represented in the XPath language (Clark and DeRose 1999) by the expression //*[@color = "Red"]. This query would be much more difficult to represent in a relational query language.

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, such as "What kinds of things in the 2002 inventory have color attributes", represented in XPath by the expression /inventory [@year = "2002"]/*[@color]. In a relational language, such a query would require a join that might span several data tables and system catalog tables.

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, such as "Find all objects that are 30 years old and have trekking as a hobby, if they have a hobby".

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, as in "Find the fifth red object" or "Find objects that occur after this one and before that one". It also means that we need facilities to impose an order on sequences of objects, possibly at several levels of a hierarchy. The importance of order in XML contrasts sharply with the absence of intrinsic order in the relational data model. XML query language functionalities were addressed in a comparative analysis of XML query Languages [11] and listed as "must have/should have" in the requirements [12] published by the W3C XML Query Language working group. 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.

Since existing database query languages like SQL do not meet the new query functional requirements posed by semistructured data like XML, the people working on this technology and the database people who want to incorporate this technology within their domain developed a number of XML query languages. These query languages and tools can be classified into two categories, namely:

- Languages and tools designed with a document focus such as XQL [13] and XPath [14].
- Languages designed with a database focus e.g. LOREL [7] and XML-QL [9].

XML Database Definition, Types and Requirements: 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 [15]. 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 (Salminen and Tompa 2001) 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 [16] 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).

Reasons for Storing XML in Databases: 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 [1]. 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 [17]. 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.
- Sooner or later, 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 [18].

XML Database Types: 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 [19, 20]. 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. The following sections describe the two XML database types.

XML-Enabled Databases: XML-enabled databases are defined as existing relational/object-relational or object-oriented databases that have been extended to provide support for XML documents. Usually, the extension comes as a middleware/top layer of the existing databases. Since an XML-enabled database is usually relational/object-relational, this thesis will focus only on this type.

XML-Enabled Database Storage Approach: Major XML-enabled database vendors—such as, Oracle, Microsoft and IBM—can store XML data using one of the following approaches [21-23]:

Storing the Xml Document as a Whole in CLOB/BLOB: XML document is stored as a whole in a single column of the relational table that has CLOB/BLOB or VARCHAR data type as a string of bytes or characters. In this approach, an exact copy of the data is stored. This is useful for special- purpose applications such as legal documents. This approach also called unstructured storage.

Shredding, or Decomposing, the XML into relational/object-relational tables: Shredding XML document involves looking at the XML data, defining a corresponding relational schema (for example, looking at parent/child relationships in the XML data and representing each child as one or more tables in a referential integrity constraint with its parent) and defining a mapping from the XML data to the relational schema. That mapping might be manually defined, programmatically defined (most frequently using XML schema as input) or defined by some combination of automatic programming followed by manual editing for fine-tuning. When shredding based on mapping between the XML data and the relational schema using XML schema as input, this approach may call a structured storage. Structured storage allows preserving fidelity of the data at the relational level-hierarchical structure is preserved, while order among elements is ignored.

Combining the above Two Approaches: Also a combination of the above approaches can be used. In fact, Oracle [24] allows developers to take a hybrid approach, storing frequently queried parts as structured storage, while keeping the remainder parts as CLOB/BLOBs. Either way, storing XML in a relational.

Native XML Databases: In [1], a native XML databases are defined as a databases designed from the ground-up especially to store, manage and query XML documents. It stores XML documents in their native form. Like other databases, they support features like transactions, security, multi-user access, programmatic APIs, query languages and so on. The only difference from other databases is that their internal model is based on XML and not something else, such as the relational model. A proposed definition for native XML databases came from the XML:DB Initiative, native XML database [25]:

- Defines a model for an XML document – as opposed to the data in that document – and stores and retrieves documents according to that model;
- Has an XML document as its fundamental unit of storage, just as a relational database has a row in a table as its fundamental unit of storage;
- Is not required to have any particular underlying physical storage model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files.

Native XML DB STORAGE Approaches: A native XML database stores XML documents in their native form. How exactly XML documents are stored within the native XML database depends on the architecture of that database. The architecture of a native XML database is categorized as one of two types: text-based and model-based [1]:

In a Text-based Native XML Database: XML documents are stored as text. Text can be stored as a file in a file system, as a CLOB/BLOB in a relational database, or in a proprietary text format. In such databases, text indexes are maintained to allow the query engine to jump to any point in any XML document very easily. After finding the right point via the indexes, the database can retrieve the entire document in one read, because it is stored in the original hierarchical format. This gives a tremendous speed advantage.

In a Model-Based Approach: The database does not store XML document as text. Prior to storing any data on permanent media, XML document is modeled i.e. transformed into an internal object model representing the source XML document. This model is then stored. Selected XML modeling has to be rich enough to model all the elements of XML documents. DOM is one of the possible choices for modeling XML. How the model is stored depends on the product. Some databases store the model in a relational or object-oriented database whereas others use a proprietary storage format suitable for their model. In the case of model-based native XML databases built on other databases are likely to have performance similar to those databases when retrieving documents for the obvious reason that they rely on those systems to retrieve data. However, in the case of a proprietary storage format the performance will most likely have the same (good) performance as text-based native XML databases when retrieving data in the order in which it is stored.

In general, with text-based storage, retrieval of parts of XML documents is much faster than with model-based storage XML databases. This is due to the longer time needed to reconstruct the XML document if having its model. On the other hand, some operations like modifying the nesting order of the XML elements can prove to be faster with model-based storage approaches, since it is faster to apply such transformations on the model of XML document compared to textual representation of one. It is thus not possible to decide beforehand on the preferred approach for a generic application.In all XML storage approaches indexes are used to speed up the retrieval of certain parts of XML documents.

The Evaluation Criteria: Query Execution Time was the evaluation criteria used to evaluate the performance of query processing in both XML DB. The experiments based on measuring the time each query takes for execution. Each query was executed five times, after excluding the fastest and the slowest runs, the query execution time will be the average of the middle three runs [26].

Experimental Environment and Achieved Results: 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.The proposed experiments architecture for the benchmark testing is shown below in Fig.1. The experiment methodology passes through five stages given as follows:

- 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.

Our obtained results are presented with respect to the database size. On the database size view, the charts show how changing the databases size effects on 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 1:

For Query#1: Exact match query shown in Fig. 2:

- In general, Tamino outperforms both Oracle storage approaches in all different XML document sizes by a factor of 16.5 times.
- The performance of both Oracle storage approaches was quite similar.

For Query#2: Ordered access query shown in Fig. 3:

- Surprising results came from Tamino: For XML document size=100, 500KB and 1MB, Tamino outperforms both Oracle storage approaches. It outperforms Oracle with unstructured/CLOB storage by a factor of 11.2 times and 10 times for Oracle with structured/object-relational mapping storage. By average it outperforms Oracle with a

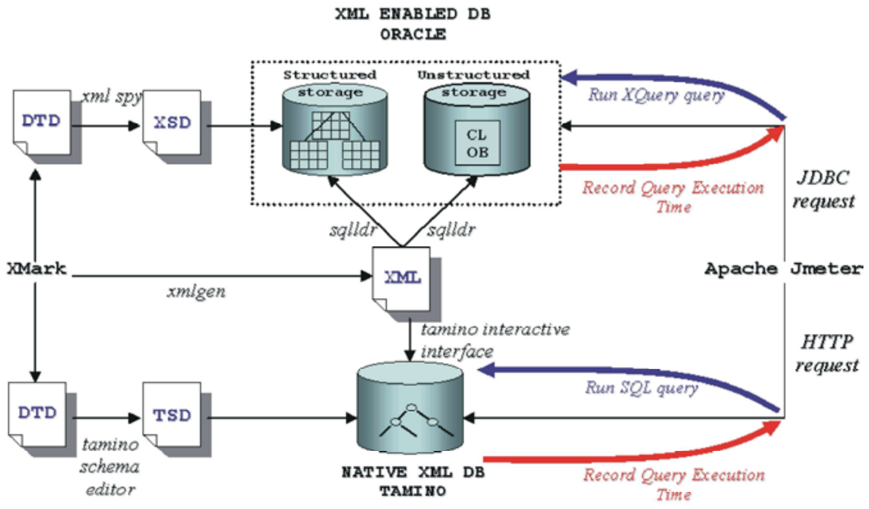
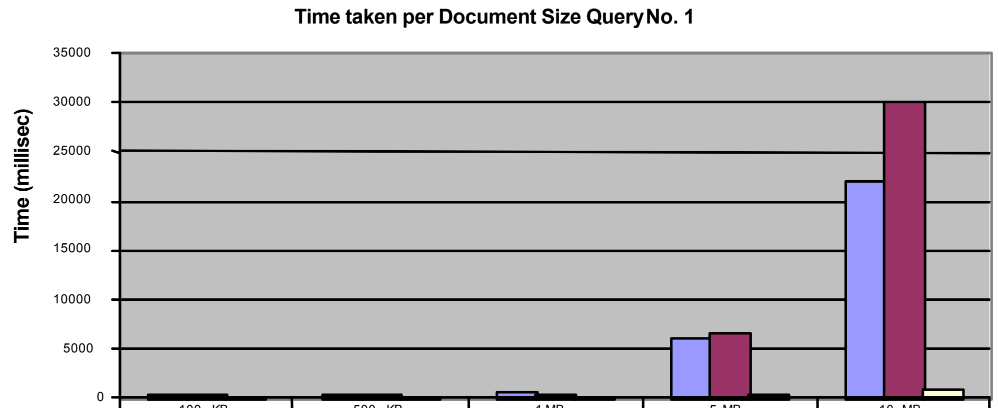


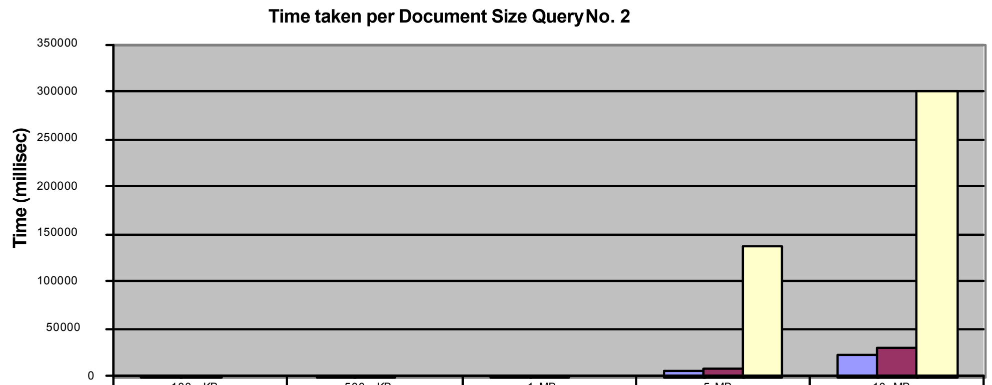
Fig. 1: The Proposed Experiments Architecture



	100 KB	500 KB	1 MB	5 MB	10 MB
🟦 Oracle CLOB storage	265.6666667	343.6666667	547	6000	22026.33333
🟥 Oracle O-R mapping storage	245	292	406.3333333	6630	29896
🟨 Tamino	15.33333333	41.66666667	62.66666667	255.3333333	984.6666667

Loaded Document Size

Fig. 2: Query execution times of query#1 on a different XML document sizes



	100 KB	500 KB	1 MB	5 MB	10 MB
🟦 Oracle CLOB storage	265.6666667	349	588.6666667	6213.666667	22239.33333
🟥 Oracle O-R mapping storage	255.3333333	307.6666667	458.3333333	6786.333333	30573
🟨 Tamino	15.66666667	47	62.33333333	137609.3333	299916.6667

Loaded Document Size

Fig. 3: Query execution times of query#2 on a different XML document sizes

factor of 10.6 times. But with XML document size=5 and 10MB, Tamino performs poorly comparing to both Oracle storage approaches. Both Oracle storage approaches were 16.4 times faster than Tamino.

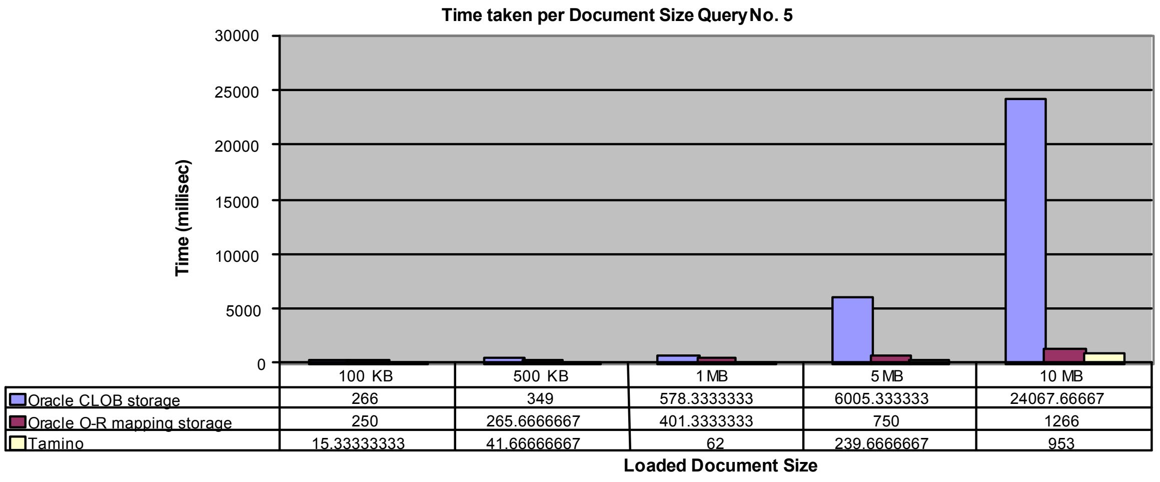
- Both Oracle storage approaches have almost similar performance.

For Query#5: Casting query shown in Fig. 4:

- Tamino yields the best performance in all different XML document sizes. It outperforms Oracle with unstructured/CLOB storage by a factor of 17 times and 6.7 times for Oracle with structured/object-relational mapping storage. By average it outperforms Oracle with a factor of 11.8 times. Note that the performance of Oracle with structured/object-relational mapping storages and Tamino were similar in XML document size=10MB.
- The performance of both Oracle storage approaches was similar in XML document size=100, 500KB and 1MB, but in XML document size= 5 and 10 MB, Oracle with unstructured/CLOB storage was the worst.

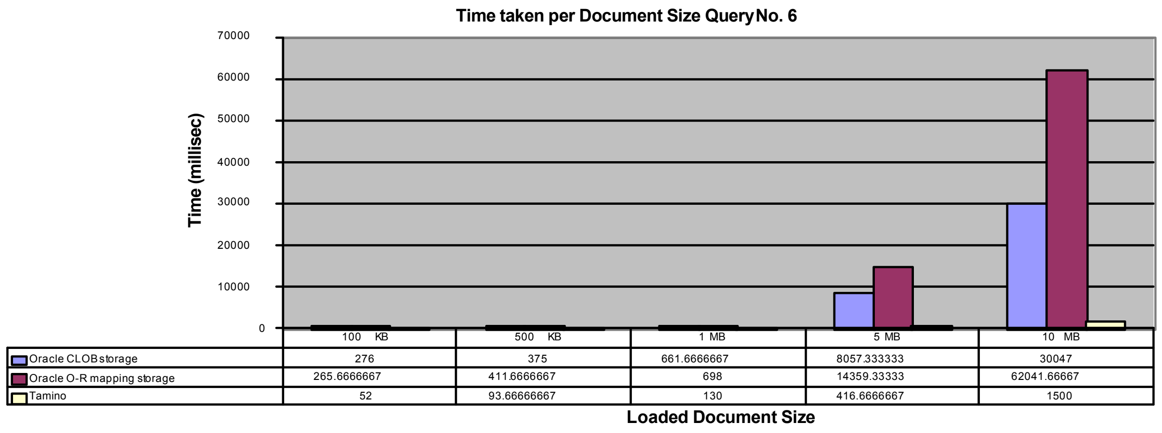
For Query#6: Regular path expressions query shown in Fig. 5:

- Tamino outperforms both Oracle storage approaches in all different XML document sizes. It outperforms Oracle with unstructured/CLOB storage by a factor of 10.7 times and 18.1 times for Oracle with structured/object-relational mapping



	100 KB	500 KB	1 MB	5 MB	10 MB
Oracle CLOB storage	266	349	578.3333333	6005.333333	24067.66667
Oracle O-R mapping storage	250	265.6666667	401.3333333	750	1266
Tamino	15.33333333	41.66666667	62	239.6666667	953

Fig. 4: Query execution times of query#5 on a different XML document sizes



	100 KB	500 KB	1 MB	5 MB	10 MB
Oracle CLOB storage	276	375	661.6666667	8057.333333	30047
Oracle O-R mapping storage	265.6666667	411.6666667	698	14359.33333	62041.66667
Tamino	52	93.66666667	130	416.6666667	1500

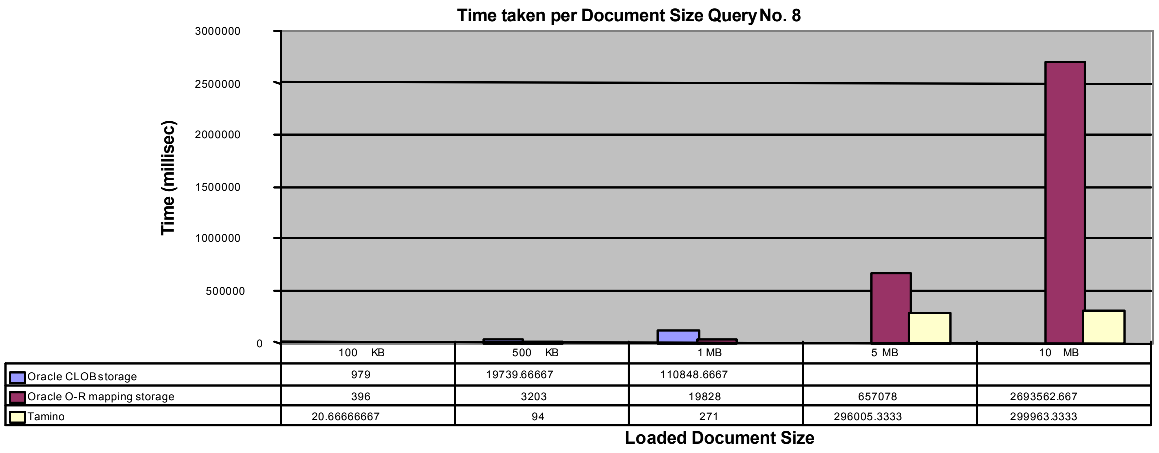
Fig. 5: Query execution times of query#6 on a different XML document sizes

storage. By average it outperforms Oracle with a factor of 14.4 times. Note that both oracle storage approaches show an average performance in XML document size=100, 500KB and 1MB. In fact, Tamino outperform both Oracle storage approaches by a factor of 4.8 times only in case of small XML documents. So, the query execution time gets higher as the XML document size get larger (XML document size=5 and 10MB).

- The performance of both Oracle storage approaches was similar in XML document size=100, 500KB and 1MB, but when XML document size get larger, XML document size=5 and 10MB, the performance of Oracle with structured/object-relational mapping storage become slightly worse. Inother words, in XML document size=5 and 10MB, Oracle with unstructured/CLOB storage outperforms Oracle with structured/object-relational mapping storage.

For Query#8: Chasing references query shown in Fig. 6:

- Tamino outperforms both Oracle storage approaches in all different XML document sizes. It outperforms Oracle with unstructured/CLOB storage by a factor of 222.1 times (excluding XML document size=5 and 10MB) and 27.5 times for Oracle with structured/object-relational mapping storage. By average it outperforms Oracle with a factor of 124.8 times.
- Note that Oracle with structured/object-relational mapping storage outperforms Oracle with unstructured/CLOB storage, in fact no results can be made from Oracle with unstructured/CLOB storage for this type of queries if the XML document size is large (XML document size=5 and 10MB).



	100 KB	500 KB	1 MB	5 MB	10 MB
Oracle CLOB storage	979	19739.66667	110848.6667		
Oracle O-R mapping storage	396	3203	19828	657078	2693562.667
Tamino	20.66666667	94	271	296005.3333	299963.3333

Fig. 6: Query execution times of query#8 on a different XML document sizes

Table 1: A summary of the main observations on the database size view

Query#	Tamnio vs. Oracle CLOB	Tamnio vs. Oracle O-R mapping	Oracle CLOB vs. Oracle O-R mapping
Query#1	In general, Tamino outperforms both Oracle storages in all different XML document sizes by a factor of 16.5 times.		Both Oracle storages have a quite similar performance.
Query#2	For XML document size<=1MB, Tamino outperforms Oracle CLOB by a factor of 11.2 times.	For XML document size<=1MB, Tamino outperforms Oracle O-R mapping by a factor of 10 times.	Both Oracle storages have almost similar performance.
	Surprising results came from Tamino: For XML document size>=5 MB, Both Oracle storages were 16.4 times faster than Tamino.		
Query#5	Tamino outperforms Oracle CLOB by a factor of 17 times.	Tamino outperforms Oracle O-R mapping by a factor of 6.7 times.	Both Oracle storages were similar in XML document size <=1MB.
Query#6	Tamino outperforms Oracle CLOB by a factor of 10.7 times.	Tamino outperforms Oracle O-R mapping by a factor of 18.1 times.	Both Oracle s were similar in XML document size <=1MB.
Query#8	Tamino outperforms Oracle CLOB by a factor of 222.1 times (excluding XML document sizes=5 and 10MB).	Tamino outperforms Oracle O-R mapping by a factor of 27.5.	Oracle O-R mapping outperforms Oracle CLOB.

Concluded Remarks and Future Works: Through our presented work in this paper, 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.Regarding performance, the experimental study done in the last part of the thesis determined that the native XML database Tamino has been performing better than XML-enabled relational database Oracle for XMark benchmark queries. The main reason could be that the XML-enabled relational database Oracle's XML support is not mature yet. With all the experiments results combined together, the following are the main findings based on these experiments:

- Native XML database Tamino has better performance than the XML-enabled database Oracle in data-centric single document domains, especially in large database sizes (beyond 5MB). This finding emphasizes the complexity associated with storing and querying XML data within relational databases and indicates that for some high performance applications, a native XML repository such as Tamino is a clear choice for managing and processing semistructured data.

- Changing the size of XML document has impact on the performance of the XML DB. Query execution time increased dramatically when the size of the XML document became considerably larger (beyond 5MB).
- Almost all of the query execution time is independent of types of queries in case of small XML document sizes. In other words, it doesn't change too much with types of queries (excluding heavy queries that involve joins).
- Native XML database Tamino yields the best performance in executing heavy queries (Chasing references queries) in large XML document sizes where Oracle with unstructured/CLOB storage was the worst, this is due to the fact that this type of queries are complex. It involves retrieving more cross-references between tables.
- For queries that process array lookup, the performance of native XML database Tamino was the worst - comparing to both Oracle storage approaches - in large XML document sizes.

As a future work to be done we expect our experiments to continue and include all the 20 XMark queries and large XML documents (beyond 10MB). Also, as XMark is relatively a simple benchmark and doesn't test data insert, update and delete operations. 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. Finally, more experiments with different kinds of document-centric and data-centric XML documents are required.

REFERENCES

1. Bourret, R., 2005b. XML and Databases, Working Paper, viewed 13 May 2005, from http://www.rpbourret.com/xml/XMLAndDatabases.htm, Last updated September, 2005.
2. Bouneffa, M., H. Basson and L. Deruelle, 2001. E-business: A new challenge for database management systems, Information & Communications Technology Law, 10(3): 299-308.
3. Balas, J.L., 2002. What is this XML thing and why do I need to know about it?, Computers in Libraries, 22(8): 39-41.
4. Obasanjo, D., 2001. An exploration of XML in database management systems, Working Paper, viewed 20 June 2005, from http://www.25hoursaday.com/StoringAndQueryingXML.html.
5. Clark, J. and S. DeRose, eds. 1999. XML Path Language (XPath) version 1.0, W3C Recommendation, 16 November 1999, viewed 15 October 2005, from http://www.w3.org/TR/xpath.
6. 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/.
7. Abiteboul, S., D. Quass, J. McHugh, J. Widom and J. Wiener, 1997. The LOREL Query Language for Semistructured Data, International Journal on Digital Libraries, 1(1): 68-88.
8. Rob, P. and C. Coronel, 2000. Database Systems: Design, Implementation & Management, 4th ed., Cambridge: Thompson Learning.
9. Deutsch, A., M. Fernandez, D. Florescu, A. Levy and D. Suciu, 1998. XML-QL: A Query Language for XML, Submission to W3C, 19 August 1998, viewed 10 January 2006, from http://www.w3.org/TR/NOTE-xml-ql/.
10. Fernandez, M., J. Siméon and P. Wadler, eds. 1999. XML Query Languages: Experiences and Exemplars, AT&T Bell Labs Technical Report, Draft manuscript, communication to the XML Query W3C Working Group September 1999, viewed 28 March 2006, from http://homepages.inf.ed.ac.uk/wadler/papers/xml-exemplars/xml-exemplars.pdf.
11. Bonifati, A. and S. Ceri, 2000. Comparative Analysis of Five XML Query Languages, ACM SIGMOD Record, 29(1): 68-77.
12. Chamberlin, D., J. Robie and D. Florescu, 2000. Quilt: an XML query language for heterogeneous data sources, in: Proceedings of the 3rd International Workshop on the Web and Databases (WebDB 2000), Dallas, Texas, May 2000, in Lecture Notes in Computer Science, Berlin: Springer-Verlag, 2000, 53-62.
13. Robie, J., J. Lapp and D. Schach, 1998. XML Query Language (XQL), in: Proceedings of the W3C Query Language Workshop (QL-98), Boston, MA, December 3-4, 1998, viewed 10 January 2006, from http://www.w3.org/TandS/QL/QL98/pp/xql.html.
14. Clark, J. and S. DeRose, eds. 1999. XML Path Language (XPath) version 1.0, W3C Recommendation, 16 November 1999, viewed 15 October 2005, from http://www.w3.org/TR/xpath.
15. McGoveran, D., ed. 2001. The Age of the XML Database, eAI Journal, October 2001, viewed 11 November 2005, from http://www.eaijournal.com/PDF/XMLMcGoveran.pdf.

16. Fohn, R., 2002. XML Database Systems, PowerwareInformatik, April 2002, viewed 10 January 2006, from http://www.powerware.ch/doc/XML Databases.pdf.
17. Rob, P. and C. Coronel, 2000. Database Systems: Design, Implementation & Management, 4th ed., Cambridge: Thompson Learning.
18. Champion, M., 2001. Storing XML in Databases, eAI Journal, October 2001, viewed 05 May 2006, from http://www.eaijournal.com/PDF/StoringXMLChampion.pdf.
19. 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
20. Kappel, G., E. Kapsammer and W. Retschitzegger, 2001. XML and Relational Database Systems - A Comparison of Concepts, in: Proceedings of the International Conference on Internet Computing, Las Vegas, Nevada, pp: 199-205.
21. Barrett, A., ed. 2001. Databases Embrace XML, Server Workstation Expert, August 2001, viewed 12 December 2005, from http://swexpert.com/F/SE.F1.AUG.01.pdf.
22. ZapThink, 2002. XML Data Storage Technologies and Trends: Native XML Data Stores (NXDs) and XML Extensions to RDBMS, March 2002, viewed 25 October 2005, from http:// www.softwareag.com/tamino/references/ ZapThink-XML_DataStores_March2002.pdf.
23. IBM, 2005. The IBM approach to unified XML/relational databases, IBM Technical Report on Unified XML/relational storage, March 2005, viewed 10 January 2006, from ftp://ftp.software.ibm.com/software/ data/pubs/papers/GC34-2496.pdf.
24. Oracle, 2002. Oracle9i XML Database Developer's Guide - Oracle XML DB Release 2 (9.2), Oracle Corporation, viewed 15 May 2006, from http://www.stanford.edu/dept/itss/docs/oracle/9i/appdev.920/a96620/xdb02rep.htm.
25. Staken, K., 2001. Introduction to Native XML Databases, O'Rilley XML.com, 31 October 2001, viewed 28 October 2006, from http://www.xml.com/pub/a/2001/10/31/nativexmldb.html.
26. Runapongsa, K., J.M. Patel, H.V. Jagadish, Y. Chen and S. Al-Khalifa, 2006. The Michigan Benchmark: Towards XML Query Performance Diagnostics, Information Systems Journal, Elsevier, 31(2): 73-97, viewed 10 January 2006, from http://www.sciencedirect.com/.