# IT620 Course Project:
# An attempt to benchmark the XML capabilities in Oracle 9iR2

Chetan Vaity (02329901)

8 April, 2003

## 1 Introduction

This project is aimed towards exploring the XMark benchmark [1] and its application to Oracle XMLDB [6]; and in the process, studying the XML storage and retrieval capabilities of Oracle.

## 2 XML benchmarks

Commercial XML storage products are starting to appear in the market with the growing popularity of XML. Currently many major database vendors leverage their existing products well beyond the rudimentary XML support like conversion of purely relational data to XML documents which most products already provide with whatever one may need to meet the new requirements. However, these new requirements are still somewhat sketchy and though the differences between XML and relational or object-relational data are easy to grasp, the implications on the underlying data store are not fully understood yet.

XML, by definition is a textual markup language which means that unlike in the case of (O)RDBMS, data elements are ordered by nature; *string* is the core data type, from which richer data types, e.g. integers, floats and even user-defined abstract data types are derived. Externally provided schema information, which may or may not be present, helps to avoid excessive and expensive coercions between data types. Additionally, to cope with the tree structure of XML documents and the resulting intricate hierarchical relationships between data, regular path expressions are an essential ingredient of query languages and need to be evaluated efficiently. References may be used to model relationships that exceed the limitations of tree structures and require further mapping logics like logical OIDs for efficient management.

Due to their complexity, interaction, and interdependencies with various system components, most of the designs, with their obvious advantages and disadvantages, are hard to assess without putting them to the only conclusive test: a comprehensive quantitative assessment, or in short the right benchmark.

Some of the benchmarks for XML stores are noted below.

### 2.1 XMach-1

XMach1 [2] tests multiuser features provided by the systems. The benchmark is modeled for a web application using XML data. It evaluates standard and nonstandard linguistic features such as insertion, deletion, querying URL and aggregate operations. It measures the throughput of a generic XML-based web application consisting of a XML database and middleware components. The database contains both structured data and text documents.

### 2.2 XMark

Xmark [1] developed under the XML benchmark project at CWI, is a very recent benchmark proposed for XML data stores. The benchmark consists of an application scenario which models an Internet auction site and 20 XQuery challenges designed to cover the essentials of XML query processing.

### 2.3 XOO7

There are straightforward correspondences between the objectoriented schemas and instances and XML DTDs and data. XOO7 [3] was designed keeping in mind these similarities in data model of XML and objectoriented approach. XOO7 is an adaptation of the OO7 Benchmark.

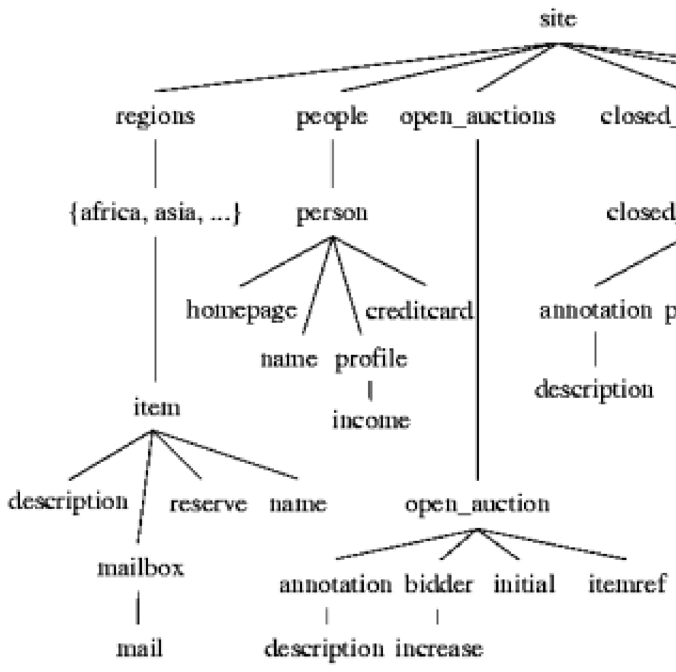
## 3 XMark

As noted earlier, this benchmark has been developed at CWI (the National Research Institute for Mathematics and Computer Science, Netherlands) and is gaining acceptance in the community.

XML processing systems usually consist of various logical layers and can be physically distributed over a network. To make the results interpretable, the systems engineering issues are abstracted and the benchmark concentrates only on the core ingredients: the query processor and its interaction with the data store. The benchmark does not consider network overhead, communication costs (e.g., RMI, HTTP, CORBA, Sockets, RPC, Java Beans, etc.) or transformations (e.g., XSL) of the output. All applications are run on the same machine. XQuery has been chosen as the query language. Updates other than bulkload are not considered as there is little agreement on semantics and a standard is yet to be defined.

### 3.1 XML document strucrure

The structure of the document is modeled after a database as deployed by an Internet auction site. The main entities are: person, open auction, closed auction, item, and category. The relationships between them are expressed through references with the exception of annotations and descriptions which take after natural language text and are document-centric element structures embedded into the sub-trees to which they semantically belong. The hierarchical schema is depicted in figure.



The semantics of the entities just mentioned is as follows:

1. **Items** are the objects that are on for sale or that already have been sold. Each item carries a unique identifier and bears properties like payment (credit card, money order, ...), a reference to the seller, a description etc., all encoded as elements. Each item is assigned a world region represented by the item's parent.
2. **Open auctions** are auctions in progress. Their properties are the privacy status, the bid history (i.e. increases over time) along with references to the bidders and the seller, the current bid, the default increase, the type of auction, the time interval within which bids are accepted, the status of the transaction and a reference to the item being sold.
3. **Closed auctions** are auctions that are finished. Their properties are the seller (a reference to a person), the buyer (a reference to a person), a reference to the respective item, the price, the amount of items sold, the date when the transaction was closed, the type of transaction, and the annotations that were made before, during and after the bidding process.
4. **Persons** are characterized by name, email address, phone number, mail address, homepage URL, credit card number, profile of their interests, a set of open auctions they watch.
5. **Categories** feature a name and a description; they are used to implement classification of items. A **category graph** links categories into a network.

The generated document does not contain any Entities or Notations. Neither is any distinction made between Parsed Character Data and Character Data; both are considered as just string types from the viewpoint of the storage engine. Namespaces are not introduced. The DTD for the XML document is provided to allow for more efficient mappings.

## 3.2 Some queries

The XMark queries are categorised into logical sections which test a particular functionality of the system. Some representative queries are presented below:

### 3.2.1 Exact match

- Return the name of the item with ID "item20748" registered in North America

-
  ```
  FOR $b IN document(‘‘auction.xml’’)
  /site/regions/namerica/item[@id="item20748"] RETURN $b/name/text()
  ```
- This simple query is mainly used to establish a simple performance primitive unit to help establish a metric to interpret subsequent queries. It tests the database ability to handle simple string lookups with a fully specified path.

### 3.2.2 Ordered access

- Return the initial increases of all open auctions

-
  ```
  FOR $b IN document(‘‘auction.xml’’)
  /site/open_auctions/open_auction
  RETURN <increase> $b/bidder[1]/increase/text() </increase>
  ```
- Queries in this section should help users to gain insight how the DBMS copes with the intrinsic order of XML documents and how efficient they can expect the DBMS to handle queries with order constraints. The above query evaluates the cost of array look-ups.

### 3.2.3 Casting

- How many sold items cost more than 40?

-
  ```
  COUNT (FOR $i IN document(‘‘auction.xml’’)
  /site/closed_auctions/closed_auction
  WHERE $i/price/text() >= 40 RETURN $i/price)
  ```
- Strings are the generic data type in XML documents. Queries that interpret strings will often need to cast strings to another data type that carries more semantics. This query challenges the DBMS in terms of the casting primitives it provides.

### 3.2.4 Chasing references

- List the names of persons and the number of items they bought (joins person, closed auction)

-
  ```
  FOR $p IN document(‘‘auction.xml’’) /site/people/person
  LET $a := FOR $t IN document(‘‘auction.xml’’) /site/closed_auctions/closed_auction
  WHERE $t/buyer/@person = $p/@id
  RETURN $t
  RETURN <item person=$p/name/text()> COUNT ($a) </item>
  ```
- References are an integral part of XML as they allow richer relationships than just hierarchical element structures. Queries in this section define horizontal traversals with increasing complexity. A good query optimizer should take advantage of the cardinalities of the sets to be joined.

### 3.2.5 Full text

- Return the names of all items whose description contains the word gold

-
  ```
  FOR $i IN document(‘‘auction.xml’’) /site//item
  WHERE CONTAINS ($i/description,‘‘gold’’)
  RETURN $i/name/text()
  ```
- To challenge the strength of the system in handling textual nature of XML documents, a full-text search in the form of keyword search is conducted.

### 3.2.6 Missing elements

- Which persons do not have a homepage?

-
  ```
  FOR $p IN document(‘‘auction.xml’’) /site/people/person
  WHERE EMPTY($p/homepage/text())
  RETURN <person name=$p/name/text()/>
  ```
- This is to test how well the query processor knows to deal with the semi-structured aspect of XML data, especially elements that are declared optional in the DTD.

# 4 XML schema

The XML Schema Recommendation was created by the World Wide Web Consortium (W3C) to describe the content and structure of XML documents in XML. It includes the full capabilities of Document Type Definitions (DTDs) so that existing DTDs can be converted to XML schema. XML schemas have additional capabilities compared to DTDs.

In XML Schema, there is a basic difference between complex and simple types:

- Complex types, allow elements in their content and may carry attributes
- Simple types, cannot have element content and cannot carry attributes.

Many datatypes (47 in number) for simple types are defined in XML schema, and others can be derived from the built-ins.

The occurence directives, minoccurs and maxoccurs, can be used to specify the cardinality of a child element. This mechanism, without losing the flexibility of DTDs, allows for greater control over occurence specifications. The W3C XML Schema vocabulary also includes constructs that can be used to define ordering, default values, mandatory content, nesting, repeated sets, namespaces etc. For more details, see the XML schema specification [5].

# 5 Oracle XDB

Oracle XML DB [7] is the term used to describe technology in the Oracle 9i Release 2 database that delivers high-performance storage and retrieval of XML. It delivers new methods for navigating and querying XML content stored inside the database.

Some of the important features of Oracle XMLDB are discussed below:

## 5.1 XMLType

XMLType is a native server data-type that allows the database to understand that a column or table contains XML. XMLType also provide methods that allow common operations such as schema validation and XSL transformations to be performed on XML content.

### 5.1.1 Unstructured storage

By default, an XMLType table or column can contain any well formed XML document. The content of the document is stored as XML text using the CLOB data type. It allows for higher rates of ingestion and retrieval, as it avoids the overhead associated with parsing and recomposition during storage and retrieval operations. When stored, any update operations on the document will result in the entire CLOB being re-written. One can use B*Tree indexes based on the functional evaluation of XPath expressions or Oracle Text inverted list indexes.

### 5.1.2 Structured storage

An XMLType table or column can be constrained to an XML Schema. Constraining the XMLType to an XML Schema provides the option of storing the content of the document using structured-storage techniques. Structured-storage decomposes or shreds the content of the XML document and stores it as a set of SQL objects rather than simply storing the document as text in CLOB. The object-model used to store the document is automatically derived from the contents of the XML Schema. This results in a slight overhead during ingestion and retrieval operations in that the document has to be shredded during ingestion and re-constituted prior to retrieval. The structured approach can update individual elements, attributes, or nodes in an XML document without rewriting the entire document. By tuning the way in which collections are managed, indexes can be created on any element or attribute in the document, including elements or attributes that appear with collections. Since this model is based on XML schema, it is not necessary for Oracle XML DB to store XML tag names when storing the contents of XML documents. This can significantly reduce the storage space required compared to unstructured storage.

## 5.2 Schema based structured storage

Oracle XML DB's XML schema functionality is available through the PL/SQL supplied package, DBMS_XMLSCHEMA, a server-side component that handles the registration of XML schema definitions for use by Oracle XML DB applications. The two main DBMS_XMLSCHEMA functions are `registerSchema()` and `deleteSchema()`.

As part of registering an XML schema, Oracle XML DB also performs several other steps to facilitate storing, accessing, and manipulating XML instances that conform to the XML schema. These steps include:

- **Creating types:** When an XML schema is registered, Oracle creates the appropriate SQL object types that enable the structured storage of XML documents that conform to this XML schema. You can use Oracle XML DB-defined attributes in XML schema documents to control how these object types are generated. The constructs defined by the XML Schema are mapped directly into SQL Types generated using the SQL 1999 Type Framework that is part of the Oracle database.
- **Creating default tables:** As part of XML schema registration, Oracle XML DB generates default XMLType tables for all root elements. You can also specify any column and table level constraints for use during table creation.

Using SQL 1999 objects to persist XML allows Oracle XML DB to guarantee DOM fidelity. Providing DOM fidelity requires the system to preserve all of the information contained in an XML document. This includes maintaining the order in which elements appear within a collection and within a document as well as storing and retrieving out-of-band data like comments, processing instructions and mixed text. By guaranteeing DOM fidelity, Oracle XML DB is able to ensure that there is no loss of information when the database is used to store and manage XML documents. To guarantee that DOM fidelity is maintained and that the returned XML documents are identical to the original XML document for DOM traversals, the system adds a system binary attribute, SYS_XDBPD$, to each created object type. This positional descriptor attribute stores all pieces of information that cannot be stored in any of the other attributes, thereby ensuring the DOM fidelity of all XML documents stored. Examples of such pieces of information include: ordering information, comments, processing instructions, namespace prefixes, and so on. This is mapped to a Positional Descriptor (PD) column. This attribute is for Oracle's internal use only.

Oracle XML DB provides the application developer or database administrator with control over how much decomposition, or shredding , takes place when an XML document is stored in the database. The schema processor recognizes a set of annotations that make it possible to customize the mapping between the XML Schema data types and the SQL data types, control how collections are stored in the database, and specify how much of a document should be shredded. Since these attributes are in a different namespace from the XML schema namespace, such annotated XML schemas are still legal XML schema documents. If one does not specify any annotations to the XML Schema to customize the mapping, the system will make certain default choices that may or may not be optimal.

All the XML primitive types are mapped to the appropriate SQL datatypes. eg: string → varchar2, float → number, byte → number(3), boolean → raw(1) etc.

Using attributes `SQLName`, `SQLType` etc in the XML Schema, the names and types of the SQL objects which are generated during registration can be controlled.

While storing complextype: the default storage of the VARRAY is in Ordered Collections in Tables (OCTs) instead of LOBs. One can choose LOB storage by setting the `storeAsLob` attribute to true. One can specify the SQLType for a complex element as a Character Large Object (CLOB) or Binary Large Object (BLOB). Here the entire XML fragment is stored in a LOB attribute. This is useful when parts of the XML document are seldom queried but are mostly retrieved and stored as single pieces. By storing XML fragments as LOBs, one can save on parsing/decomposition/recomposition overheads.

# 6 Work done

## 6.1 Installation

The Oracle database software version 9.2.0.1 for Linux on ix86 was downloaded and installed on one of the servers in the School. The version 9.2.0.2 has significant bug fixes in the XML DB component, but as this version could not be obtained, further work was carried out with the existing version.

## 6.2 DTD to XML schema and registering the schema

The DTD of the XML document was obtained along with the XMark benchmark. An effort was made to manually transform this into an XML schema document. Later, a tool, XMLSpy [8], was used which automatically did the transformation. Some necessary namespace declarations were added to this document and then the schema was registered using the `registerschema()` function.

## 6.3 Data generation and loading

The utility `xmlgen` associated with XMark was compiled for the present platform. XML files of sizes 100 MB and 1 MB were created. For loading, the utility SQL Loader `(sqlldr)` was used.

Only the 1 MB document could be successfully loaded into the database.

## 6.4 XPath to SQL

The XPath queries mentioned in XMark were translated to equivalent SQL queries involving the XMLDB functions. Note that the queries involving `xmlsequence` include the namespace declarations in the xpath functions. (This makes the translated queries appear too long)

1. Query 1

   -
     ```
     FOR $b IN document(‘‘auction.xml’’)
     /site/regions/namerica/item[@id="item20748"] RETURN $b/name/text()
     ```

   -
     ```
     SELECT extractValue(auction, ’/site/regions/namerica/item[@id=”item178”]/name/text()’ )
     FROM auction_tab;
     ```

2. Query 2

   -
     ```
     FOR $b IN document(‘‘auction.xml’’)
     /site/open_auctions/open_auction
     RETURN <increase> $b/bidder[1]/increase/text() </increase>
     ```

   -
     ```
     SELECT extract(value(t), ’/open_auction/bidder[1]/increase’,
     .     ’xmlns=”http://www.oracle.com/AU.xsd”
     .     xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”’)
     FROM auction_tab,
     .      TABLE ( xmlsequence( extract(auction, ’/site/open_auctions/open_auction’,
     .                  ’xmlns=”http://www.oracle.com/AU.xsd”
     .                  xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”’) )) t;
     ```

3. Query 3

   -
     ```
     FOR $b IN document("auction.xml")
     /site/open_auctions/open_auction
     WHERE $b/bidder[0]/increase/text() * 2 <= $b/bidder[last()]/increase/text()
     RETURN <increase first=$b/bidder[0]/increase/text() last=$b/bidder[last()]/increase/text()>
     ```

   -
     ```
     SELECT extractValue(value(t), ’/open_auction/bidder[1]/increase/text()’,
     .     ’xmlns=”http://www.oracle.com/AU.xsd”
     .     xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”’),
     extractValue(value(t), ’/open_auction/bidder[last()]/increase/text()’,
     .     ’xmlns=”http://www.oracle.com/AU.xsd”
     .     xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”’)
     FROM auction_tab,
     .     TABLE( xmlsequence(extract(auction, ’/site/open_auctions/open_auction’,
     .                  ’xmlns=”http://www.oracle.com/AU.xsd”
     .                  xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”’)) ) t
     WHERE extractValue(value(t), ’/open_auction/bidder[1]/increase/text()’,
     .     ’xmlns=”http://www.oracle.com/AU.xsd”
     .     xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”’)
     .                  * 2 = extractValue(value(t), ’/open_auction/bidder[last()]/increase/text()’,
     .     ’xmlns=”http://www.oracle.com/AU.xsd”
     .     xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”’)
     ```

4. Query 5

   -
     ```
     COUNT (FOR $i IN document(‘‘auction.xml’’)
     /site/closed_auctions/closed_auction
     WHERE $i/price/text() >= 40 RETURN $i/price)
     ```

- SELECT count(*)
  FROM auction_tab,
  . TABLE ( xmlsequence ( extract( auction, '/site/closed_auctions/closed_auction',
  . 'xmlns="http://www.oracle.com/AU.xsd"
  . 'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"') )) t
  WHERE extractValue (value(t), '/closed_auction/price',
  . 'xmlns="http://www.oracle.com/AU.xsd"
  . xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"') ¿= 40

5. Query 8

   -
     ```
     FOR $p IN document(``auction.xml'') /site/people/person
     LET $a := FOR $t IN document(``auction.xml'') /site/closed_auctions/closed_auction
     WHERE $t/buyer/@person = $p/@id
     RETURN $t
     RETURN <item person=$p/name/text()> COUNT ($a) </item>
     ```
   - SELECT extract(value(p), '/person/name/text()',
     . 'xmlns="http://www.oracle.com/AU.xsd"
     . xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"')
     FROM auction_tab a,
     . TABLE (xmlsequence( extract(a.auction, '/site/people/person',
     . 'xmlns="http://www.oracle.com/AU.xsd"
     . xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"') )) p,
     . TABLE (xmlsequence( extract(a.auction, '/site/closed_auctions/closed_auction',
     . 'xmlns="http://www.oracle.com/AU.xsd"
     . xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"') )) t
     WHERE extractValue(value(t), '/closed_auction/buyer/@person',
     . 'xmlns="http://www.oracle.com/AU.xsd"
     . xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"' )
     = extractValue(value(p), '/person/@id',
     . 'xmlns="http://www.oracle.com/AU.xsd"
     . xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"')

Note that all the queries given in the XMark paper assume a 100 MB dataset. In the current project, the values were scaled down for a 1 MB dataset.

## 6.5 Results

The query execution times in seconds for the above mentioned queries for unstructured and structured storage schemes are shown in table below.

| **Queries** | Unstructured (LOB based) | Structured (schema based) |
|---|---|---|
| Q1 | 0.207 | 8.812 |
| Q2 | 1.297 | 1.031 |
| Q3 | 2.307 | 2.429 |
| Q5 | 0.688 | 0.608 |
| Q8 | 245 | 780 |

It is seen that the response times for these queries is equivalant or more in the case of structured storage compared to unstructured storage. The reasons may be:

- The document is too small (1MB) for the stuctured storage paradigm to exhibit its advantage

- The overhead of querying from different tables and joining between them is perhaps exceeding any potential gain

In the query plan obtained from the system, only the top level XMLType table is referred and the internal tables are completely hidden. Thus, further investigations for the above results was difficult.

## 7 Problems faced

During the course of the project, some practical problems were encountered.

- 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.
  A particular problem was that even after the document was successfully loaded, simple XPath queries were returning empty sets. The attribute `elementFormDefault` needed to be defined to overcome this issue.
  Finally the XML schema had these declarations:
  ```
  targetNamespace="http://www.oracle.com/AU.xsd"
  xmlns:au="http://www.oracle.com/AU.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  ```
  and the XML document had the following declarations:
  ```
  xmlns="http://www.oracle.com/AU.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oracle.com/AU.xsd
  http://www.oracle.com/AU.xsd"
  ```
  The `schemaLocation` attribute does require rge URL mentioned twice, one of them being the *hint location.*

- Loading the XML data using SQL loader was especially problematic as all of it was being entered into one tuple (XMLType) of a relational table. SQL loader's buffering limits were reached and had to be extended using the `READSIZE` command line option. Also, the `concatenate` option was used to load the data in smaller chunks. Although this strategy eventually worked for the 1 MB file, the 100 MB file could not be loaded.

- The queries involving `xmlsequence` and `table` constructs were returning null results. This problem was eventually solved by specifying namespace declarations in the `extract()` and `extractValue()` functions. Further, all xmlsequence examples in the Oracle documentation showed queries with a join between the original table and the table generated with xmlsequence. Efforts were made to remove the join, as according to the `xmlsequence` syntax it should be possible to deal with the table returned by it. But, the problem is that the temporary table created is not a schema based table and hence XML functions do not work on it.

## References

[1] A.R. Schmidt and F. Waas and M. L. Kersten and D. Florescu and I. Manolescu and M. J. Carey and R. Busse, The XML Benchmark project, INS-R0103, CWI Centrum voor Wiskunde en Informatica, April 2001

[2] XMach-1 , Proceedings of German database conference BTW2001, Oldenburg, 7-9 March, Springer, Berlin 2001

[3] The XOO7 XML Management System Benchmark, Stiéphane Bressan, Mong Li Lee, Ying Guang Li

[4] XQuery: http://www.w3.org/XML/Query

[5] XML Schema: http://www.w3.org/XML/Schema

[6] Oracle XML DB, An Oracle Technical White paper

[7] Oracle9i XML Database Developer's Guide - Oracle XML DB Release 2 (9.2), http://download-east.oracle.com/docs/cd/A97630_01/appdev.920/a96620/toc.htm

[8] XMLSpy: http://www.xmlspy.com