CS764 Project

-George Kola

Native XML Databases versus XML-Enabled Relational - A Comparative Study

George Kola

Introduction

With the widespread adoption of XML, building databases to store XML content has become imperative. There have been two approaches to storing XML content – One is to have a fully native XML database and the other approach is to add features supporting XML to existing relational database. The former approach has been taken by object database vendors and the later approach has been taken by relational database vendors. In this study we intend to highlight some important features of the two approaches which makes one better than the other in certain cases. This is neither an in-depth nor an exhaustive study, but is intended to be used as starting point for further research in this area.

The study is made of two parts. In the first part we point out the features of two approaches and present certain cases where we expect one approach to be better than the other. In the second section, we discuss the XML support in Tamino, a native XML database from Software AG and Oracle 9i Release 2 an XML-Enabled RDBMS. We also present the result of running sample queries on the two products.

Native XML Databases

Native XML databases have a persistent DOM (Document Object Model) tree. This enables them to handle queries based on the structure of the XML document efficiently. They also allow dynamic schema modification and this offers better flexibility. Native XML databases are better suited for document-centric XML content where there is no fixed structure. Example of a document-centric content is a collection of short-stories.

XML Enabled Relational Databases

XML Enabled Relational Databases are traditional relational databases which have added some support for XML content. The most common feature added is to have an object-relational mapping between XML content and the tables in the RDBMS. The XML Content is shredded and stored in the RDBMS tables. The problem with this approach is that DOM fidelity is lost during the shredding. Also for certain content the object-relational mapping is very difficult to come up with. These databases handle such content by storing them as Character Large Objects (CLOB). The problem with CLOB is that if only a small portion of the XML is modified, the whole CLOB would have to be

read, modified and updated. So the performance of this CLOB storage model decreases rapidly as the size of the document stored increases.
XML Enabled Relational Databases are good for certain type of data-centric content which has a fixed structure and fits well in relational tables and does not have any hierarchical information. Example of such a data-centric content would be a 'Purchase Order'.
Another advantage of XML-Enabled RDBMS is that they carry the maturity of the RDBMS product with them. Organizations have been storing their critical data on RDBMS and have grown to trust them with their data. This is not true for Native XML products.

Cases where Native XML Database is better

Dynamically changing the XML Schema
If the XML Schema structure is changed dynamically, an example being adding a new node to the XML Schema, Native XML Databases handle it quite elegantly. XML Enabled RDBMS with Object Relational Mapping do not allow such dynamic changes.

Queries based on the structure of the document
Since XML-Enabled RDBMS do not typically have a persistent DOM-tree, they have difficulty handling complex queries based on the structure of the document.

Mixed-Content XML Document
If the XML document has mixed content, it is difficult to come up with a good Object-Relational mapping for such a content. Most mapping would involve expensive joins (expecting mixed content to map to multiple tables) and in this case Native XML database would be better.

Example of mixed type content

An XML element, "letter", that contains both other elements and text:

```
<letter>
Dear Mr.<name>George Kola</name>.
Your order <orderid>1024</orderid>
will be shipped on <shipdate>2002-12-24</shipdate>.
</letter>
```

Here text and other element are mingled together and that makes the content mixed.

Cases where XML-Enabled RDBMS is better

SQL queries on a data-centric XML document
If the XML document is data-centric and has fixed structure and the type of queries that you want executed map on to simple SQL queries on the underlying table, then XML Enabled RDBMS would be better.

XML Support in Tamino

Tamino is a native XML Database from Software AG. Before loading a document into Tamino, the corresponding 'Tamino Schema Definition' has to be registered with the database. Tamino Schema is similar to XML Schema with the restriction that Complex types are not allowed to be global. Tamino Schema also extends XML Schema with attributes for specifying the underlying storage for XML Schema types and for specifying the elements to create index on.

Collection Name **:** ContactInfo
Schema Name **:** AddressBook
Doctype Name **:** Entry

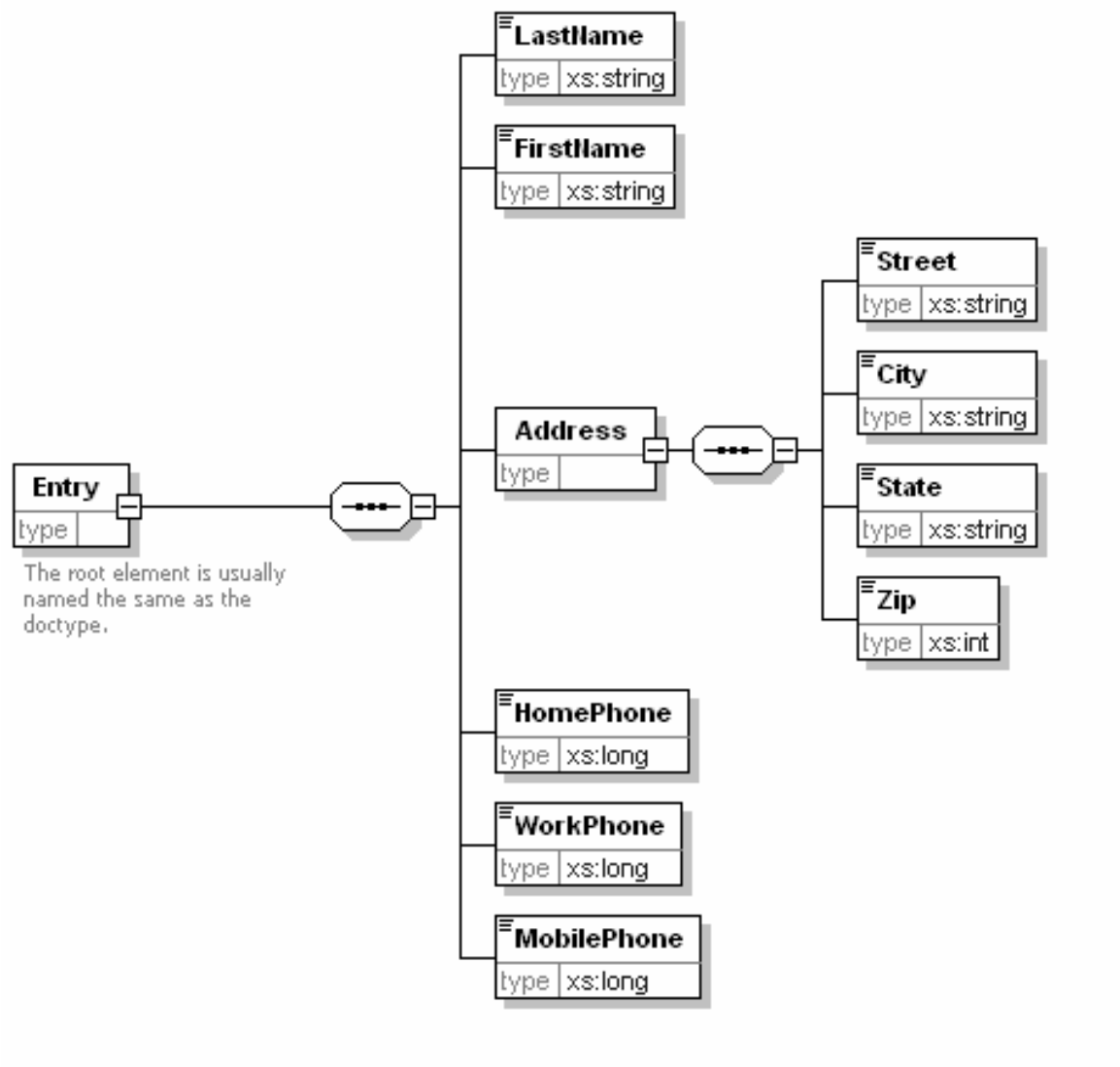


Fig 1: Tamino Schema Definition

Each schema registered with Tamino must have a 'Collection Name', a 'Schema Name' and a 'Doctype Name'. Doctype name is usually the root element in the schema. Schema Name is the name you would use to refer the schema and Collection Name is the name of the Collection you would want the schema to belong to. Because a complex-type cannot be made global, it cannot be used in another schema. For example in the above schema, the address type cannot be used in another schema.
Tamino extensions to XML Schema help you to specify attributes like whether you want to create an index on an element. Suppose in the above schema, if we want to specify that we want to create an index on Zip, it would look like

```
<xs:element name="Zip">
        <xs:annotation>
            <xs:appinfo>
                <tsd:elementInfo>
                    <tsd:physical>
                        <tsd:native>
                            <tsd:index>
                                <tsd:standard/>
                            </tsd:index>
                        </tsd:native>
                    </tsd:physical>
                </tsd:elementInfo>
            </xs:appinfo>
        </xs:annotation>
        <xs:simpleType>
            <xs:restriction base="xs:int">
                <xs:totalDigits value="5"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
```

The types of indices allowed on an element are standard, text and standard+text.

For querying the stored XML document, XQuery is used. The query is normally sent as a http request.

Example of a simple Tamino Query:

Tamino database is running on localhost
Database name in Tamino is dbproj
Tamino service is bound to /tamino in the apache webserver running on localhost

Collection Name is “ContactInfo”
DocType Name is “Entry”

We query the document based on the schema in Fig 1.

Query: Find all entries in the phone book whose HomePhone number has the area-code of 608.

The http query looks like below

http://localhost/tamino/dbproj/ContactInfo/Entry?_XQL=Entry[HomePhone>6079999999 and HomePhone<6090000000]

Here the http query used HTTP/GET, it can also use HTTP/POST.
The response got is an XML document which looks like below

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

```
<ino:response
xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
xmlns:xql="http://metalab.unc.edu/xql/">
  <xql:query>Entry[HomePhone>6079999999 and
HomePhone<6090000000]</xql:query>
<ino:message ino:returnvalue="0">
  <ino:messageline>XQL Request processing</ino:messageline>
  </ino:message>
<xql:result>
<Entry ino:id="54952"
xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefi
nition">
  <LastName>George</LastName>
  <FirstName>Kola</FirstName>
  <Address>
        <Street>1309 Spring Street</Street>
        <City>Madison</City>
        <State>Wisconsin</State>
        <Zip>53715</Zip>
  </Address>
  <HomePhone>6082569056</HomePhone>
  <WorkPhone>6082625386</WorkPhone>
  <MobilePhone>6507857825</MobilePhone>
</Entry>
</xql:result>
<ino:message ino:returnvalue="0">
  <ino:messageline>XQL Request processed</ino:messageline>
</ino:message>
</ino:response>
```

XML Support in Oracle 9i Release 2

Oracle 9i Release 2 claims to have a native XML database called XMLDB. XML data can be stored either in an XMLType Table or as a XMLType column in a table.

e.g:
CREATE TABLE PhoneBook of XMLType ;
CREATE TABLE PhoneBook2 (info XMLType);
You can optionally specify a schema name associated with a XMLType Table. For this the schema has to be registered with Oracle 9i. The schema registration takes a schema and an url to register the schema to. For example the schema in fig 2 is registered to url http://romano/PhoneBook.xsd. The schema used by oracle is standard XML Schema annotated with datatype information. If a schema is not specified, the XML data is stored as CLOB.

e.g.
CREATE TABLE PhoneBook of XMLTYPE

XMLSCHEMA "http://romano/PhoneBook.xsd"
ELEMENT "Entry";

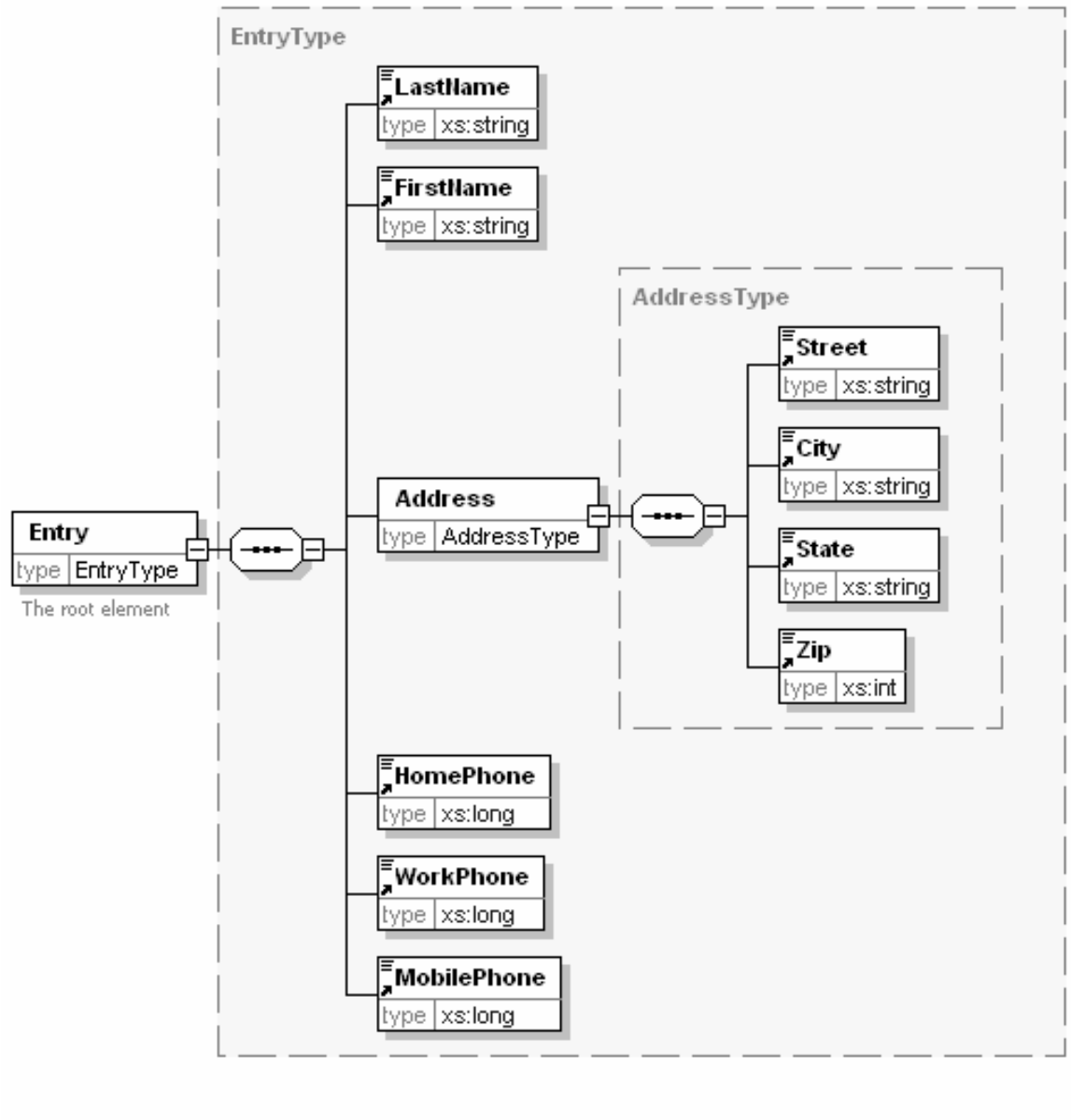


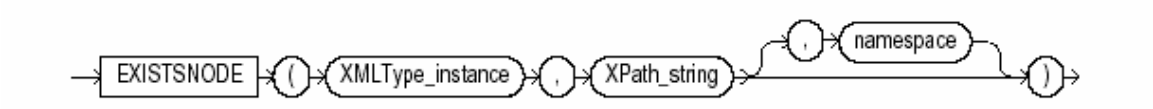
Fig 2. Oracle Schema Definition

The oracle extensions to XML Schema allow you to map the simple XMLType to Oracle datatypes. Complex-types are mapped to objects and there is an option to specify whether or not you want to store the complex-type element(e.g. Address in fig 2) as an inlined object. If a complex-type element is not inlined, accessing that element would require a join. If the schema is not annotated, the XML document would be stored as a CLOB. Also oracle allows you to store some elements in columns and the rest in a CLOB.

XML Queries on Oracle 9i

Standard SQL has been modified to support XPath queries. Three functions existsNode(), extract(), extractValue() have been added which support XPath expressions. Functions getClobVal(), getStringVal() and getNumberVal() helps to extract CLOB, String, Number from an XML Fragment.

existsNode() yntax:



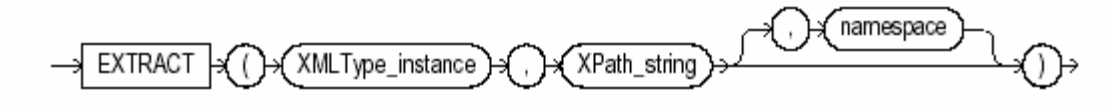
It evaluates to 1 or 0 depending on whether or not the given XMLType_instance satisfies the XPath expression respectively.

e.g.
Consider PhoneBook being a table of XMLType specified by schema in fig 2.

```
select count (*)
From PhoneBook  p
where existsNode(value(p),'Entry[HomePhone > 6079999999 and HomePhone <
6090000000]') =1;
```

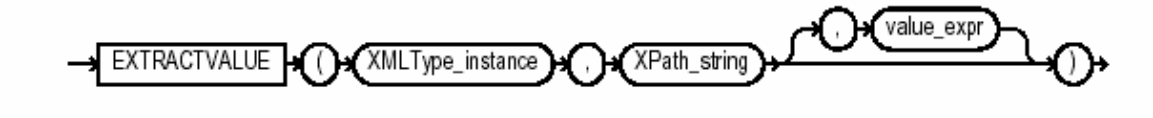
This returns the number of entries whose HomePhone has the area-code of 608.

extract() Syntax



It applies the XPath expression to XMLType_instance and returns an XMLType instance containing the XML fragment produced by the XPath string.

extractValue() Syntax



Syntax is similar to extract(). It is a shortcut function which directly extracts the value of the top XMLNode returned by XPath expression.

Experience with Oracle XMLDB

The annotation of the schema to specify the underlying datatype in oracle has to be done using an XML editor with that capability. Currently only XMLSpy from Altova has that capability and is available only on windows. This made us use Windows 2000 platform. The annotation of the schema is essentially an Object-Relational mapping. The annotation becomes complicated if you have complex types. The interfaces provided by oracle do not give good feedback if there are any mistakes in the annotation. It just tells you that a error occurred and does not tell what the error is. Also successful registration of the schema does not mean that everything is fine. You need to load data into the table to verify it. Schema and element mismatch might occur which could be due to a problem with the mapping.

Under the covers of Oracle XML Support

If an annotated XMLSchema is associated with a table, then all XPath expression gets converted to equivaled SQL expressions on the underlying data in the table. Also Oracle claims that they store a persistent DOM tree to handle/speed up structural queries. Only that with the annotated schema, whitespaces, tabs and other formatting information might get lost, but DOM fidelity is maintained. If whole document fidelity is to be maintained, then CLOB storage has to be used.

Ways of accessing XML Documents

In addition to having their own API, both Tamino and Oracle provide folder access (WEBDAV) to the XML documents. In addition there is also http and ftp access to the document.

Experiment Results

The experiments were done on a AMD Athlon 2000+ XP machine with 512MB RAM running windows 2000. The databases used were Tamino 3.1.1 Patch Level 4 from Software AG and Oracle 9i Release 2 from Oracle Corporation.

Some Limitations

The evaluation version of Tamino, that we used allows us only to store upto 20 megabytes of data. Storing 25000 phonebook records resulted in about 18 megabytes of data. But with that we were unable to perform queries because Tamino kept complaining that ‘journal space has been exhausted’. The initial journal space was set to 50 megabytes. At least in the trial version, there did not seem to be a way to increase the journal space on the fly. Because of this the maximum records stored on Tamino was limited to 10000 and all queries were done on that number of records.

Configurations used

Tamino
Default Tamino XML database

Oracle XMLDB with Object-Relational Mapping
We created a table of XMLType and associated an annotated XML Schema with it. The annotated Schema specified the Object Relational Mapping. We were not able to associate an XML Schema with a XMLType column of a table. So trying out that option was ruled out.

Oracle XMLDB with CLOB Storage
If a schema is not specified for an XMLType table, the storage defaults to CLOB. There was also an option which allows use to use CLOB storage even after specifying a schema. This option however did not work. Because the sql*loader would allow us to load only into XMLType column of a table, we created a table with a single XMLType column.

```
CREATE TABLE PhoneBook_Clob (info XMLTYPE);
```

Oracle Relational
Here we used just plain oracle RDBMS. The elements in the phone book schema were mapped to columns in Oracle table. We performed normal relational operations on them. Both input and output were tuples and not XML. Oracle has libraries to generate XML data out of tuples – the library just inserts the appropriate tags. Due to lack of time, we did not get to evaluate that.

1. The first experiment is to load data into the database

We tried to use the best way to load data into the database. For Tamino, there was a dataloader utility (inoxmld) which allows you to load multiple xml documents into the database. Oracle has sql*loader which allows us to bulk-load database tables. Unfortunately sqlloader doesnot support loading data into XMLType table. It supports loading data into XMLType column of a table. This method was used for Oracle with CLOB storage.
We were not able to use a table with XMLType column for Oracle with Object-Relational mapping because we were not able to associate a schema with an XMLType column. We had to use multiple insert statements to load data into XMLType table. Also by default, oracle does not check if the loaded XML conforms to the specified schema. We turned on the option to check the XML loaded conforms to the schema. (Adding a constraint on the table).

For oracle with CLOB storage, we do not have to specify a schema.

The Oracle tables have been created as follows

Oracle XMLDB with OR Mapping
Create Table PhoneBook of XMLType
XMLSCHEMA "http://localhost/PhoneBook_Oracle.xsd"
ELEMENT Entry;

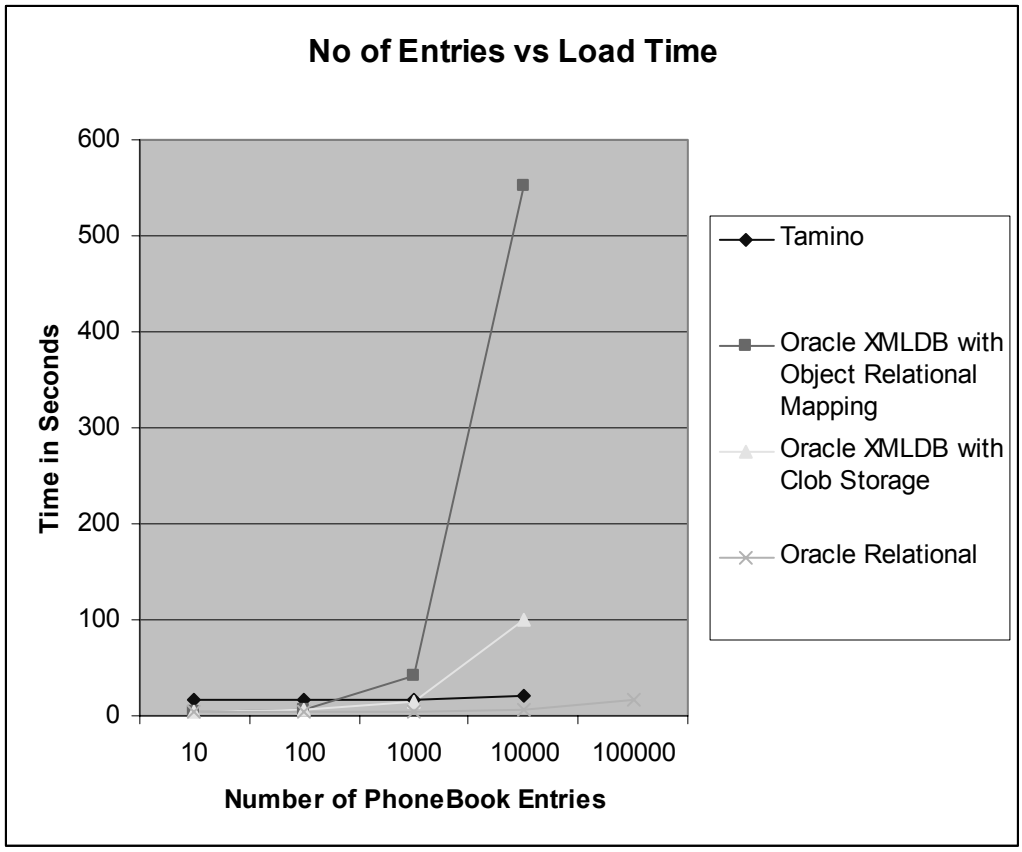
Oracle XMLDB With CLOB Storage
create table PhoneBook_Clob(info XMLType);

Oracle relational
create table phonebook_sql
(LastName Varchar2(16),
FirstName Varchar2(16),
Street Varchar2(64),
City Varchar2(16),
State Varchar2(16),
Zip Number(5),
HomePhone Number(10),
WorkPhone Number(10),
MobilePhone Number(10));

The document generator used is described in the appendix.

Load Time in seconds for the different configurations

Number of PhoneBook Entries	10	100	1000	10000	100000
Tamino	16.45	16.58	16.99	21.8	
Oracle XMLDB with Object Relational Mapping	4.9	7.15	42.4	552.47	
Oracle XMLDB with Clob Storage	4.7	5.6	13.59	99.07	
Oracle Relational	4.5	4.6	4.67	5.3	17.47



This shows oracle XMLDB with OR Mapping performing badly. It might be possible to write a PL/SQL stored procedure to speedup insertion of document into the database. But writing such a procedure did not seem straight-forward to us. May be the next release of Oracle would improve the sql*loader to support loading data into XMLType tables.

Query 1:
Find the number of entries whose HomePhone has the area code 608.

Tamino Query:

"http://localhost/tamino/dbproj/ContactInfo/Entry?_XQL=count(Entry[HomePhone>607 9999999 and HomePhone<6090000000])"

Oracle XMLDB with OR Mapping Query

```
SELECT COUNT (*)
FROM PhoneBook X
WHERE existsNode(value(X),  'Entry[HomePhone>6079999999 and
HomePhone<6090000000]') =1;
```

Oracle XMLDB With CLOB Storage
Modifying the query for Oracle with OR Mapping we get

```
SELECT COUNT(*)
FROM PhoneBook_Clob p
WHERE p.info.existsNode('Entry[HomePhone>6079999999 and
HomePhone<6090000000]')=1
```

We found that this query did not work correctly. The existsNode() always returned 0. We thought that the problem was the lack of schema definition(not knowing that HomePhone is a number type). So we modified the above query to

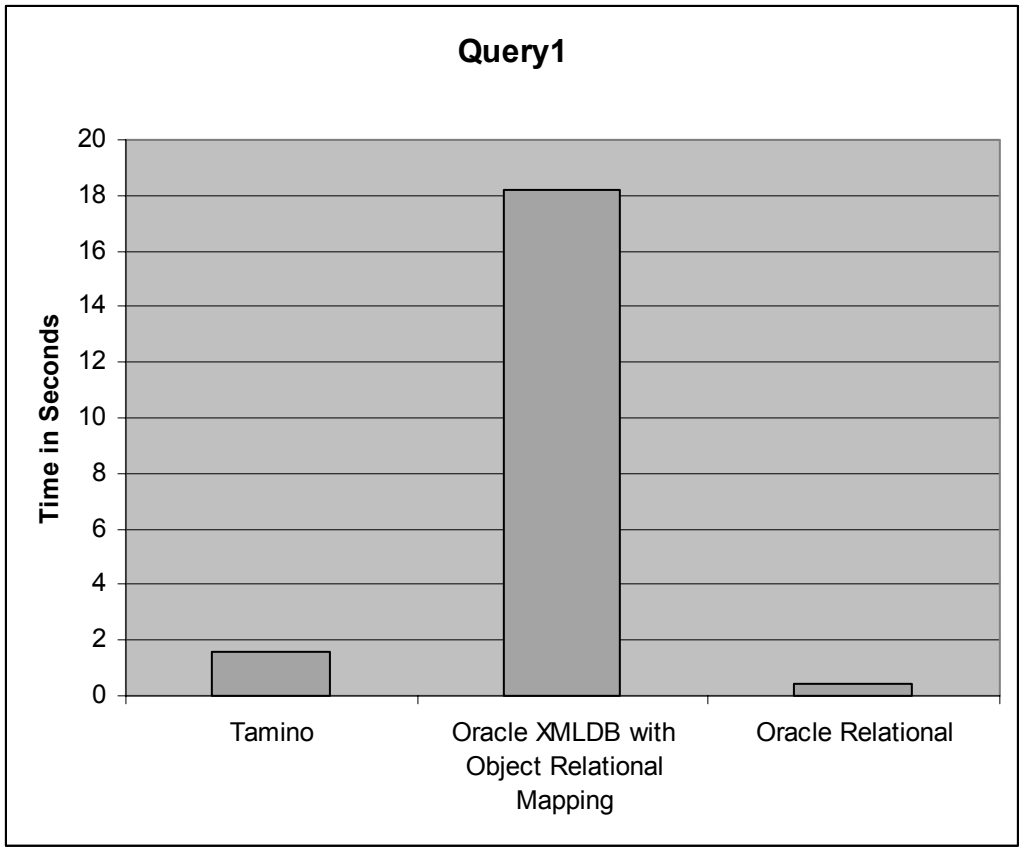
```
SELECT COUNT(*)
FROM PhoneBook_Clob p
WHERE  extract(p.info,'//HomePhone/text()').getNumberVal() > 6079999999 and
          extract(p.info,'//HomePhone/text()').getNumberVal() < 6090000000;
```

This also did not work. A little more probing showed that none of the XPath queries worked on the CLOB type XML document. The document said, Oracle text queries can be run on the XMLType column with CLOB storage. But modifying the XPath query to oracle text query is non-trivial. The document did not mention that XPath queries cannot be run on XMLType column.

Oracle Relational Query

```
SELECT COUNT (*)
FROM phonebook_sql
WHERE HomePhone > 6079999999 and HomePhone <  6090000000 ;
```

	Query Time in Seconds
Tamino	1.57
Oracle XMLDB with Object Relational Mapping	18.2
Oracle Relational	0.46

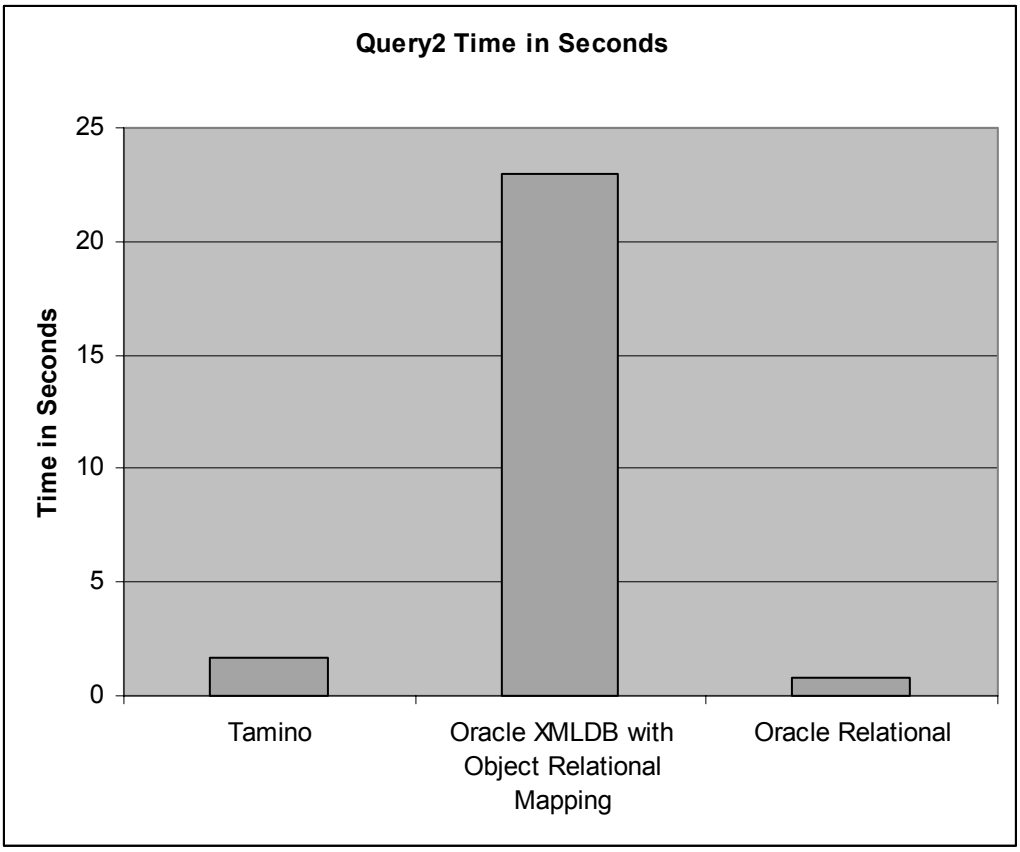


Here again, Oracle XMLDB performs badly. We tried to analyse the reason for the bad behavior. We found that Oracle internally converts the XPath expression into SQL query and runs it. The conversion seemed to be taking quite some time. Also we were not able to find out the details of the table used to store the XML content. The document claimed that the columns were hidden and truly they were!.We could not get information to improve the performance. Oracle has an option to create functional indices on XPath expressions and we found that it performs became slightly worse after building the index for the above query. Note in the query, we were just interested in the count and RDBMS performs the best.

Query 2:
Extracting the phonebook entries whose address lies in the zipcode begins with 537

Here we expect an XML output. Oracle-relation just produced tuples and it is not quite comparable.

	Query2 Time in Seconds
Tamino	1.7
Oracle XMLDB with Object Relational Mapping	23
Oracle Relational	0.8



Query 3 -- Query on a mixed content XML Document

Fig 3 shows the schema for mixed content. We wanted to mark up the nouns, verbs, adjectives and prepositions in a set of short-stories. We wanted to find the list of noun that also appear as adjectives. We created the schema for Tamino. When we tried to load mixed content we got error messages and the document was rejected. We had turned on mixed=true in the attributes and still could not get it working.

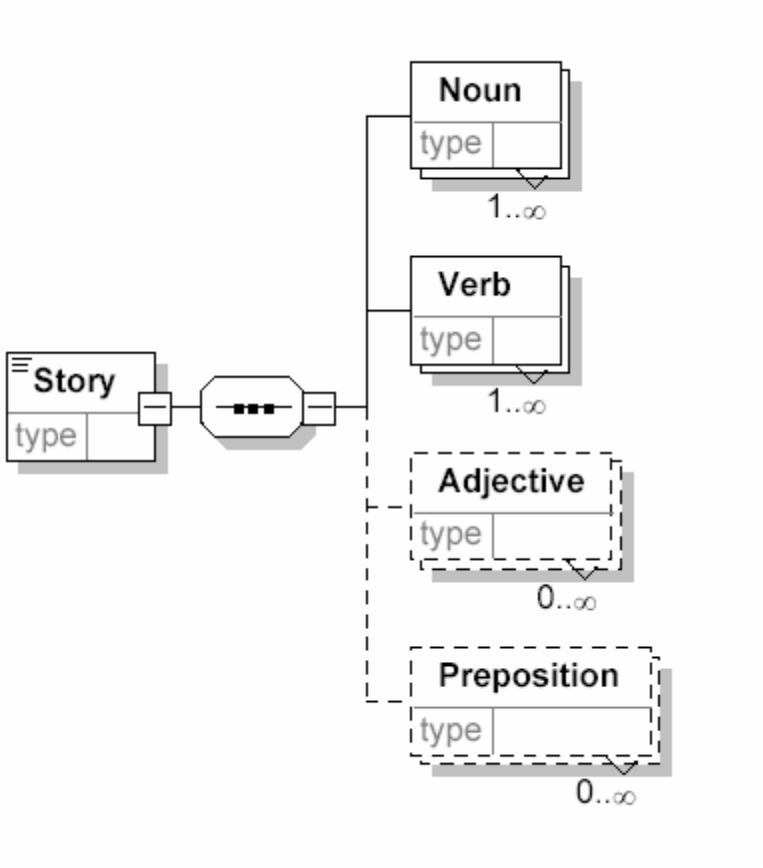


Fig 3. Mixed Content Schema

Conclusion

We have compared native XML databases and XML Enabled RDBMS. We find that native XML databases have been performing better then XML Enabled RDBMS for XML queries. The main reason could be that the XML Enabled RDBMS Oracle's XML support has not matured. The product used is the first release with these XML features.

References

[1] XML Database Products, http://www.rpbourret.com/xml/XMLDatabaseProds.htm
[2] Tamino Native XML Database, http://www.softwareag.com/tamino/
[2] Oracle 9i Release 2 Database, http://otn.oracle.com/products/oracle9i/content.html
[3] Oracle XMLDB, http://otn.oracle.com/tech/xml/xmldb/
[4] Oracle9i XML Database Developer's Guide - Oracle XML DB Release 2 (9.2), htttp://otn.oracle.com/docs/products/oracle9i/doc_library/release2/appdev.920/a96620/toc.htm

Appendix

Document Generators

1. PhoneBook Entry Generator

It takes as input four filenames and the number of entries to generate. It uses the list of States in the US and randomly generates LastName, FirstName, Street, City and Zip and randomly chooses a state. It also randomly generates HomePhone, WorkPhone and MobilePhone. LastName, FirstName and City are limited to 16 characters and the number of actual character in an entry is also chosen randomly. Street can have upto 64 characters, size again chosen randomly and it can have spaces in the string generated. The reason we use four filenames is that we want the input to be the same for all the configurations. The first file is used for Oracle XMLDB with OR Mapping, second for Tamino, third for Oracle with CLOB storage and fourth for Oracle-relational. The file generated also contain commands for the loader of that configuration.

2. MixedContent Generator

It takes a filename and the number of short stories to generate. Internally it assumes that a sentence can be made up of a noun, verb, optional adjective, optional preposition and an optional noun. Note that the sentence is not tagged. The number of sentences to generate is randomly chosen with an upperlimit of 500.