Oracle8*i* - The XML *Enabled* Data Management System

Sandeepan Banerjee
Vishu Krishnamurthy
Muralidhar Krishnaprasad
Ravi Murthy
(sabanerj, vkrishna, mkrishna, rmurthy@us.oracle.com)

Abstract

XML is here as the internet standard for information exchange among e-businesses and applications. With its dramatic adoption and its ability to model structured, unstructured and semi-structured data, XML has the potential of becoming the data model for internet data. In the recent years, Oracle has evolved its DBMS to support complex, structured, and un-structured data. Oracle has now extended that technology to enable the storage and querying of XML data by evolving its DBMS to an XML enabled DBMS - Oracle8i.

In this paper, we will present Oracle's XML-enabling database technology. In particular, we will discuss how XML data can be stored, managed, and queried in the Oracle8i database.

1.0 Introduction

XML is becoming the internet standard for information exchange. Businesses need to be able to communicate among other businesses and workflow components using XML. However, a large part of the business data sits in rows and columns of relational and object-relational tables, and will continue to be so since the database provides excellent queriability, scalability and availability. It is imperative, therefore, to be able to convert this data to and from XML in the simplest way possible. We expect that for the next several years, XML will be predominantly used as a standard of information exchange. Consequently, a large amount of XML will be converted to and from object-relational and pure relational database structures.

There is, however, a different scenario in the case of messaging subsystems and document management subsystems. The XML data corresponding to these subsystems tend to be unstructured or semi-structured.

In this paper we will discuss, how using the Oracle8*i* server along with some simple utilities, we can achieve both the goals of storing and retrieving structured and unstructured XML documents in a simple, scalable and efficient manner.

The rest of the paper is structured as follows. We will first discuss the infrastructure support in Oracle8*i* namely the Object-relational technology, extensibility framework and the Java support. Then we will discuss the various options available for mapping XML to database structures. We will then consider how to store structured and unstructured XML documents and query over them. Finally we will discuss the forthcoming standard for expressing XML schemas and its relevance to the mappings discussed above.

2.0 Oracle8*i*'s Infrastructure support for XML

We will discuss the various infrastructure components available in Oracle8*i* that can be leveraged to support XML [Oracle8i].

2.1 Object-Relational infrastructure

With the popular Oracle8 and Oracle8*i* releases of its server, Oracle has evolved its flagship database to an object-relational engine closely following the evolution of the SQL99 standard. Object-relational technology enables you to encapsulate data and its associated behavior into a single unit. Oracle's object-relational engine allows you to define such *object* types, *collections* of types as well as *references* to object types.

For instance, consider the case of a *Purchase Order.* A purchase order includes details about the purchase, the customer, and a list of line items that are shipped with the order. If normalized into relational tables, there will be two tables, one for the purchase order, one for all the line items and the customer information and address will be broken into multiple columns (such as street, city, zip etc.). Using user-defined type mechanism, this can be modeled as a purchase order object containing a shipping address, customer object and a list of line items. The type definitions are given below:-

```
CREATE TYPE addressType AS OBJECT
(
 street varchar2(100),
 city   varchar2(40),
 state  varchar2(2),
```

```
 zip    varchar2(20)
);

CREATE TYPE customerType AS OBJECT
(
 custNo number,
 custName varchar2(30),
 custAddr addressType,
);

CREATE TYPE lineItemType AS OBJECT
(
 lineItemNo number,
 lineItemName varchar2(30),
 lineItemPrice number,
 lineItemQuan number
);

CREATE TYPE lineItemList
   AS TABLE OF LineItemType;

CREATE TYPE PurchaseOrderType AS OBJECT
(
 purchaseNo number,
 purchaseDate date,
 customer customerType,
 lineItemList lineItemListType
);
```

We can now create a table definition that holds instances of the purchase order type.

```
CREATE TABLE purchaseOrderTab
      AS TABLE OF purchaseOrderType
 NESTED TABLE lineItemList
     STORE AS lineItemListNestedTab;
```

We can store instances of purchase order in this table,

```
INSERT INTO purchaseOrderTab VALUES(
 purchaseOrderType('1001',SYSDATE,
   customerType(100,'Hose',
      addressType('200 Redwood Shrs','Redwood
                    City','CA','94065'),
   lineItemListType(
            lineItemType(901,'Chair',234.55,10),
            lineItemType(991,'Desk',3456.63,  20)
    )
  );
```

We can execute queries on these using the SQL extensions for object-relational access.

```
SELECT VALUE(p) FROM purchaseOrderTab;
                -- gets the purchase order object

SELECT e.customer.custAddr
FROM purchaseOrderTab e;
                -- get the customer addresses
```

The object-relational infrastructure provides the support for storing structured object instances in the database. XML, inherently being a structured data format, can be easily mapped to object-relational instances. This mapping will be discussed further in later sections.

2.2 Extensibility Architecture

Normally, the database provides a set of services - for example, a basic storage service, a query processing service, services for indexing, query optimization and so on. Applications use these services to avail themselves of database functionality.

In Oracle8*i*, these services are made *extensible* so that data cartridges can provide their own implementations. When some aspect of a native service provided by the database is not adequate for a specific domain, a developer may provide a domain-specific implementation. For example, if you build a Spatial data cartridge for Geographical Information Systems, you may need the capability to create spatial indexes. To do this, you would implement routines that create a spatial index, insert an entry into the index, update the index, delete from it, and so on. The server would then automatically invoke your implementation every time indexing functionality was needed for spatial data. In effect, you would have extended the Indexing Service of the server to handle spatial data.

An example use of the extensibility infrastructure is *inter*Media text searching. The text kept in LOBs is indexed using the extensibility indexing interface. *inter*Media text provides operators such as CONTAINS which you can use to search within the text for substring matches.

The extensibility framework provides the infrastructure for specialized XML cartridges to be built, where the indexing and optimization of access to XML is accomplished by the cartridge.

2.3 Java support

Oracle8*i* provides native support for Java in the DBMS, by providing a native Java VM that is closely integrated with the database for high performance and scalability. In addition, the database system natively supports JDBC, SQLJ, an ORB and the EJB framework. In addition, Oracle8*i* also comes with a HTTP listener, which means that it can act as a web server as well.

The object-relational framework provides a more natural way to maintain a consistent structure between a set of Java classes at the application level

and the data model at the data storage level. In Oracle8*i*, the object-relational facilities have been tightly integrated with the Java environment in the following ways:

1. Server object-relational schema can be mapped to java classes. The JPublisher utility can generate this mapping automatically.
2. Java is one of the language choices for implementing object type methods and data cartridges.
3. Objects can be manipulated (stored and retrieved) using JDBC or SQLJ.

Support for Java within the database is vital, since a lot of the XML infrastructure, such as parsers etc. are available in Java and can be readily used inside the server. Also, the components for XML built in Java can be run inside the server or outside in the application tier.

3.0 XML in the database

There are several different aspects to using XML in the database. The most common case is to use XML as a interchange format where the existing business data is wrapped into XML structures. In this case the XML format is used only for the interchange process itself and is transient. The other scenario is to store and query XML documents in the database. Oracle8*i* supports both of these models.

3.1 Generation of XML

XML can be generated from object-relational tables and views. The benefits of using object-relational tables and views as opposed to pure relational structures are discussed below. Oracle has released a free utility available at the Oracle Technology Network [XMLSQL]. This utility converts the result of a SQL query into XML by mapping the query alias or column names into the element tag names and preserving the nesting of object types. The result representation can be in text or a DOM (Document Object Model) tree, the generation of the latter avoids the overhead of parsing the text to directly realize the DOM tree.

There is a clean relationship between structured XML instances and object-relational types. Columns map to top level elements. Scalar values map to a elements with text only content. Object types are mapped to elements with its attributes appearing as sub-elements. Collections map to lists of elements. Further, we can also map object references and referential constraints to IDREFs in the XML document.

For instance, the following Java code generates an XML instance corresponding to a SQL query.

```
public void testXML()
{
  DriverManager.registerDriver(
    new oracle.jdbc.driver.OracleDriver());

  //initialize a JDBC connection
  Connection conn =
      DriverManager.getConnection(
              "jdbc:oracle:oci8:scott/tiger@");

  //initialize the OracleXMLQuery;
  OracleXMLQuery qry =
       new OracleXMLQuery(conn,
              ”select * from purchaseOrderTab”);

  // set the document name
  qry.setRowsetTag(“PurchaseOrderList”);

  // set the row element name
  qry.setRowTag(“PurchaseOrder”);

   // get the XML result
  String xmlString = qry.getXMLString();

   // print result
  System.out.println(“ OUPUT IS:\n"+xmlString);
}
```

The query selects all the top level elements from the purchase order table and we use the generic mapping to get the following XML document:-

```
<?xml version=’1.0’?>
<PurchaseOrderList>
 <PurchaseOrder num=”1”>
  <purchaseNo>1001</purchaseNo>
  <purchaseDate>10-Jan-1999 20:33:23.3
   </purchaseDate>
  <customer>
   <custNo>100</custNo>
   <custName>Hose</custName>
   <custAddr>
    <street>200 Redwood Shrs</street>
    <city>Redwood City</city>
    <state>CA</state>
    <zip>94065</zip>
   </custAddr>
  </customer>
  <lineItemList>
   <lineItem>
    <lineItemNo>901</lineItemNo>
    <lineItemName>Chair</lineItemName>
    <lineItemPrice>234.55</lineItemPrice>
    <lineItemQuan>10</lineItemQuan>
   </lineItem>
   <lineItem>
    <lineItemNo>991</lineItemNo>
```

```
      <lineItemName>Desk</lineItemName>
      <lineItemPrice>3456.63</lineItemPrice>
      <lineItemQuan>20</lineItemQuan>
    </lineItem>
  </lineItemList>
 </PurchaseOrder>
 <PurchaseOrder>
   <!-- more purchase orders. -->
 </PurchaseOrder>
</PurchaseOrderList>
```

The XML document created is an exact structural replica of the object type. Using object views, you can create such object-relational mappings from existing relational tables.

3.2 XML Storage options

There are several storage options available for storing XML data.

LOB storage:

CLOB storage: Oracle8*i* provides for the storage of unstructured data as 'large objects' or LOBs. Unstructured XML documents can be stored in Character LOBs (CLOBs).

BFILE storage: Although more useful for multimedia data, BFILEs which are external file references can also be used. In this case the XML is stored and managed outside the RDBMS, but however, can be used in queries on the server. The metadata for the document may be stored in object-relational tables in the server for fast indexing and access.

Oracle8*i* allows the creation of *inter*Media text indexes on these LOB columns, in addition to URLs that point to external documents. This text cartridge leverages the extensibility mechanism and provides full text indexing of these documents. Oracle8*i* has extended this mechanism to work on XML data as well. The text cartridge can recognize XML tags, and section and sub-section text searching have been extended to support searching within an XML element content. The result is that queries can be posed on unstructured data and restricted to certain sections or elements within a document.

Object-Relational storage:

A natural way to store XML is as object-relational instances. The object-relational type system can fully capture and express the nesting and list semantics of XML. Complex XML documents can be stored as object-relational instances and indexed efficiently. With the extensibility infrastructure, new types of indices, such as path indices may be created for faster searching through XML documents.

3.3 Storage of structured XML documents

XML data coming into the database may be of two forms. The data may be in the form of structured documents, where the structure is known *a priori* and is the same for all instances. In this case, the document can be stored in relational or object-relational structures. In this case as well, the object-relational type system can provide a direct mapping to the XML document. This mapping is relatively straight forward and the Oracle XMLSQL utility offers an insert mechanism that can map an XML document directly into a given table or view.

The mapping is done by matching the element tag names with the column names in the table. Elements with text only content map to scalar columns and elements containing sub-elements map to object types. Lists of elements map to collections.

The example XML document given earlier can be inserted into the database using the XMLSQL utility as follows:-

```
String xmlDoc = " …the actual xml document… ";
Connection conn =
      DriverManager.getConnection(…);
OracleXMLSave sav = new
       OracleXMLSave(conn,"purchaseOrderTab");
sav.insertXML(xmlDoc);
```

The advantage of storing an XML document as an object-relational instance is that the structure of the document is preserved in the database as well. This allows the XML document to be viewed and traversed in SQL in a way similar to a XPath traversal on the document.

For instance a XPath traversal such as,

```
//purchase_order[pono=101]/shipaddr/street
```

can be easily represented as an attribute traversal in SQL -

```
SELECT  po.shipaddr.street
FROM    purchase_order_tab po
WHERE  po.pono = 100;
```

Mapping to Object-Relational storage enables existing database applications to work against XML data. Further, the rich functionality provided by Oracle8*i* on Object-Relational columns such as indexing, partitioning, parallel query, etc. can be leveraged.

However, using such a mapping the original document is not exactly reproducible - for instance, comments are lost. But this can be avoided by storing a copy of the original document in a CLOB (discussed below) and using the object-relational mapped data for query efficiency purposes. Another potential problem could arise due to the ordering amongst the elements. In order to preserve the element ordering, we can have a special column in the underlying table and order the results using that column.

3.4 Storage of unstructured XML documents

If the incoming XML documents do not conform to one particular structure, then it might be better to store such documents in CLOBs. For instance, in an XML messaging environment, each XML message in a queue might be of a different structure.

Oracle8*i* provides interMedia Text cartridge for indexing CLOB columns. This cartridge uses the extensibility mechanism to implement operators such as CONTAINS to search the text data. This has been extended to support searching of XML documents, including section and subsection searches.

```
SELECT *
FROM   purchaseXMLTab
WHERE
 CONTAINS(po_xml,"street WITHIN addr") >= 1;
```

A CLOB storage is ideal if the structure of the XML document is unknown, arbitrary or dynamic. However, much of the SQL functionality available on object-relational columns cannot be exploited. Also, concurrency of certain operations such as updates may be reduced. However, the exact copy of the document is retained.

3.5 Hybrid approach

In the previous section we discussed how structured XML documents can be mapped to object-relational instances and unstructured XML documents to LOBs. However, in many of the cases, the user would like to have a better control of the granularity of the mapping. For instance, when mapping a text document, such as a book, in XML, the user would not want every single element to be exploded and stored as object-relational. Storing the font information and paragraph information for such documents in an object-relational format, does not serve any useful purpose with respect to querying. Storing the whole text document in a CLOB reduces the effective SQL queriability on the entire document.

The alternative is to have user-defined granularity for such storage. In the book example, the user would like to query on top-level elements such as chapter, section, title etc. and the contents within a section can be stored in a CLOB.

The user can specify the granularity of mapping at table definition time. The server will automatically construct the XML from the various sources and decompose queries appropriately. The advantage of this approach is that it gives the flexibility of storing useful and queryable information in object-relational format while not decomposing the entire document. It also saves time in reconstructing the document, since the entire document is not broken down. It also enables text searching on part of the document which are stored in LOBs.

3.6 XML Querying and SQL Interoperability

XPath has been devised as a standard for XML document navigation in XSLT scripts and XPointers. This can be extended for navigation in XML documents stored in the database system. Thus if the user has stored *purchase_order* as an XML document, to retrieve the first line item numbers of a particular document, you can execute a query such as,

```
SELECT extractNode(e.po_xml_column,
                   '//line_item_list[1]/itemno')
FROM   purchase_order_tab e
WHERE e.pono = 100;
```

This example shows how SQL and XML querying can coexist - XPath behaves as a sublanguage within a SQL operator. We propose two operators, *extractNode* and *existsNode*. extractNode() operator extracts the given nodes from the XML document and returns an XML fragment. existsNode() on the other hand returns a boolean value indicating if the Xpath traversal on the document yielded any nodes. The boolean operator is useful in predicates. For example, the following query lists all documents which contain a line item with itemno = 100.

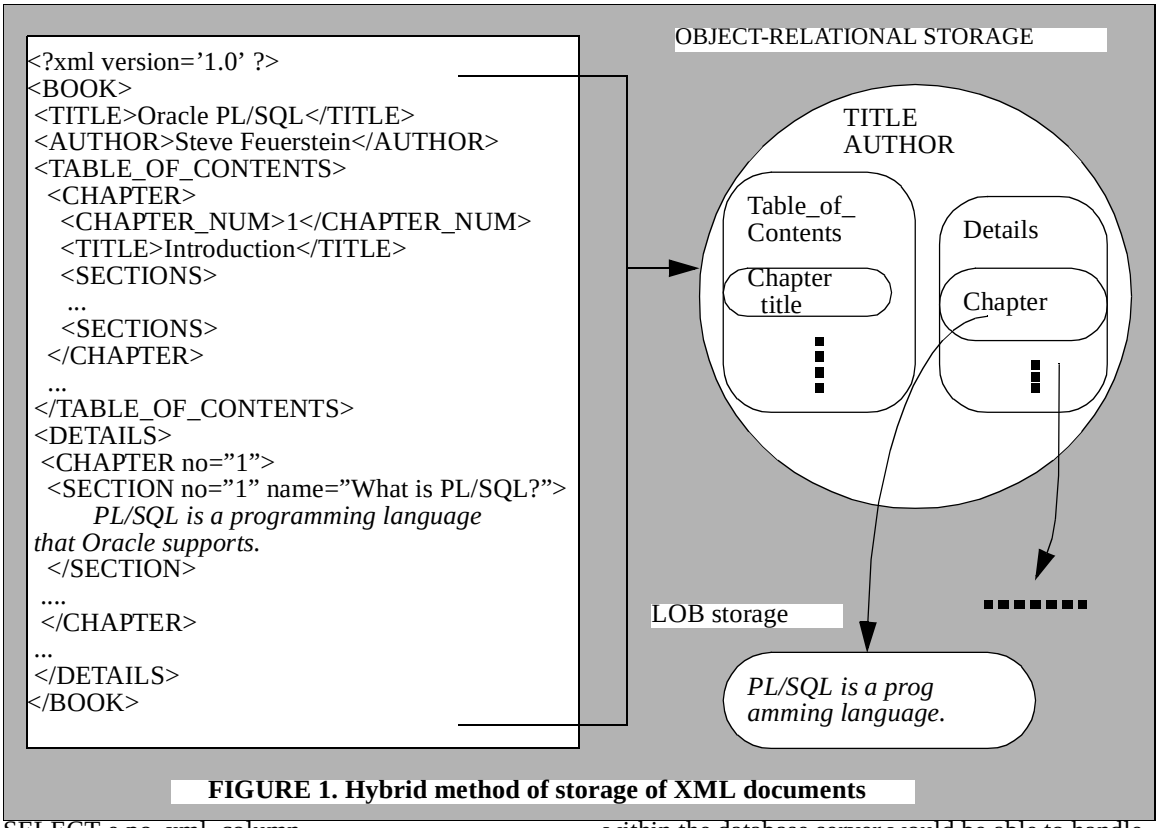

FIGURE 1. Hybrid method of storage of XML documents

```
SELECT e.po_xml_column
FROM purchase_order_tab e
WHERE
 existsNode(e.po_xml_column,
          '//line_item_list[itemno="100"]/') != 0;
```

3.7 XML Transformations

The XML that is generated from the database will be in a canonical format that maps columns to elements and object types to nested elements. However, applications might require different representations of the XML document in different circumstances.

3.7.1 Transformation of query results:

This involves querying on the original document and transforming the result into a form required by the end-user or application. For instance, if an application is talking to a cellular phone using WML, it might need to transform the XML generated into WML or other similar standard suitable for communicating with the cellular phone.

This can be accomplished by applying XSL-T transformations on the result XML document. The XSL-T transformations can be pushed into the generation phase itself as an optimization. A scalable, high performant XSL-T transformation engine within the database server would be able to handle large amounts of data.

3.7.2 Indexing and Querying on transformations:

There are cases when we need to be able to create indexes and query on transformed *views* of an XML document. For example, in a XML messaging scenario, there might be purchase order messages in different formats. The user, however, would like to query them in a canonical fashion, so that a particular query can work across all the purchase order messages. In this case, the query is posed against the transformed view of the documents.

The user can register standard XSL-T transformation scripts for transforming XML documents to the canonical format, with the database. When indexing or querying, the database can automatically create the transformed virtual documents and satisfy the query or build the index.

3.8 Indexing approaches

The naive implementation for the extractNode() and existsNode() operators is to parse the XML document and perform the path traversal and extract the fragment. But this would not be a perfor-

mant and scalable solution. The second approach would be to utilize the inverted text index which *inter*Media text creates when it parses a document. This can give the offset into the document where the element occurs. But this would not work for object-relational data. A third but different approach would be to index the XPath path expressions themselves.

A lot of pioneering work has been done in this respect in the research community, especially in Lore [Lore]. Lore creates a *data guide* for all semi-structured documents stored in the database, which is an index of all possible path traversals through all the documents. We can follow a similar approach and index on all possible XPath path expressions on the XML document, using the extensibility mechanism. This, however, does pose some challenges in generating the optimal execution plans - such as bottom up vs. top down evaluations.

4.0 XML schemas and mapping of documents

W3C has chartered a Schema working group to provide a new, XML based notation for structural schema and datatypes as an evolution of the current Document Type Definition (DTD) based mechanism. XML Schemas can be used for constraining document structure (elements, attributes, namespaces)[XML-Schema1] as well as content (datatypes, entities, notations)[XMLSchema2]; the datatypes themselves can either be primitive (such as bytes, dates, integers, sequences, intervals) or be user-defined (including ones that are derived from existing datatypes and which may constrain certain properties -- range, precision, length, mask -- of the basetype.) Application-specific constraints and descriptions are allowed. XML Schema provides inheritance for element, attribute, and datatype definitions. Mechanisms are provided for URI references to facilitate a standard, unambiguous semantic understanding of constructs. The schema language provides for embedded documentation or comments.

For example, you can define a simple data type as:

```
<datatype name=”positiveInteger”
          basetype=”integer”/>
  <minExclusive> 0 </minExclusive>
</datatype>
```

Though an exhaustive discussion of XML Schema is beyond the scope of this paper, it is clear even from the simple example above that it provides a number of important new constructs over DTDs -- such as a basetype, and a ‘minimum value’ constraint. When dynamic data is generated from a database, it is typically expressed in terms of a database type system. In case of Oracle, this is the object-relational type system described above, which provides for much richness in data types -- such as NULL-ness, variable precision (e.g. NUMBER(7,2)), check constraints, user-defined types, inheritance, references between types, collections of types and so on. XML Schema can capture a wide spectrum of schema constraints that go towards better matching generated documents to the underlying type-system of the data.

Consider the simple Purchase Order type expressed in XML Schema:

```
<type name="Address" >
  <element name="street" type="string" />
  <element name="city"   type="string" />
  <element name="state"  type="string" />
  <element name="zip"    type="string" />
</type>

<type name=”Customer”>
  <element name=”custNo”
              type=”positiveInteger”/>
  <element name=”custName” type=”string” />
  <element name=”custAddr” type=”Address” />
</type>

<type name=”Items”>
  <element name=”lineItem” minOccurs=”0”
                                maxOccurs=”*”>
   <type>
    <element name=”lineItemNo”
                type=”positiveInteger” />
    <element name=”lineItemName”
                 type=”string” />
    <element name=”lineItemPrice”
                type=”number” />
    <element name=”LineItemQuan”>
     <datatype basetype=”integer”>
      <minExclusive>0</minExclusive>
     </datatype>
    </element>
   </type>
  </element>
</type>

<type name="PurchaseOrderType">
   <element name="purchaseNo"
               type="positiveInteger" />
   <element name="purchaseDate"  type="date" />
   <element name="customer” type=”Customer” />
   <element name="lineItemList"  type="Items" />
</type>
```

These XML Schemas have been deliberately constructed to match closely the Object-Relational purchase order example described in Section 2.1 above. The point is to underscore the closeness of match between the proposed constructs of XML Schema with SQL:1999-based type systems. Given such a close match, it is relatively easy for us to map an XML Schema to a database Object-Relational schema, and map documents that are schema-valid according to the above schema to row objects in the database schema. In fact, the greater expressiveness of XML Schema over DTDs greatly facilitates the mapping.

The applicability of the rich schema constraints provided by XML Schema is not limited to data-driven applications. There are more and more document-driven applications that exhibit dynamic behavior. A simple example might be a memo, which is routed differently based on markup tags. A more sophisticated example is a technical service manual for an intercontinental aircraft. Based on complex constraints provided by XML Schema, one can ensure that the author of such a manual always enters a valid part-number, and one might even ensure that part-number validity depends on dynamic considerations such as inventory levels, fluctuating demand and supply metrics, or changing regulatory mandates.

5.0 Summary

XML is emerging as the standard for data interchange on the web. Oracle8*i* is XML-enabled to handle the current needs of the market. Oracle8*i* is capable of storing structured XML data as object-relational data, and unstructured XML document as interMedia Text data. Correspondingly, Oracle8*i* also provides the ability to automatically extract object-relational data as XML. In Oracle8*i*, efficient querying of XML data is facilitated using standard SQL. Oracle8*i* also provides the ability to access XML documents using the DOM (Document Object Model) API.

Oracle8*i* will continue to evolve to meet the needs of the web. Oracle8*i*'s highly scalable, robust, database platform will be evolved to become a leading XML server providing efficient and seamless XML support using standard APIs, languages and protocols.

6.0 References

1. [Oracle8i] Oracle8*i* documentation set, *http://www.oracle.com*
2. [XMLSQL] Oracle XMLSQL utility, *http://technet.oracle.com/tech/xml/oracle_xsu/*
3. [XSL-T] XSL transformations, *W3C Recommendation, 1999, http://www.w3.org/TR/xslt*
4. [XPath] XML Path language, *W3C Recommendation, 1999, http://www.w3c.org/TR/xpath*
5. [XML] Extensible Markup Language, *W3C Recommendation, 1998, http://www.w3c.org/TR/1998/REC-xml-19980210*
6. [XML-Schema1] XML Schema Part 1 - Structures, *W3C Working Draft, 1999, http://www.w3.org/TR/xmlschema-1/*
7. [XML-Schema2] XML Schema Part 2 - Datatypes *W3C Working Draft, 1999, http://www.w3.org/TR/xmlschema-2/*
8. [Lore]. Query Optimization for XML, J. McHugh and J. Widom, *Proceedings of the Twenty-Fifth International Conference on Very Large Data Bases, Edinburgh, Scotland, September 1999.*