Crossroads
The ACM Student Magazine

Managing XML Data Storage

by ***Jerry Emerick***

Introduction: XML Storage Demands

XML is becoming the data format of choice for a wide variety of information systems solutions. Common applications using XML include document transmission in B2B systems, message format construction for integration of Internet applications with legacy systems, binding of XML data to visual and non-visual controls, data storage and retrieval, and various data manipulation activities within applications. The benefits most often associated with the utilization of XML include platform independence, low cost of entry, and the ability to seamlessly share data. XML can also accommodate a variety of data including text, images, and sound. However, the excitement over XML does not come without challenges stemming from emerging and conflicting standards, new skill requirements, and the management issues caused by the growing amounts of XML data to be be managed [**7**].

The majority of both traditional business applications and Internet based applications depend on databases. Several different database models exist, however, the majority are relational [**7**]. To maintain data in a database, data must be retrieved and stored in a consistent, reliable, and efficient manner. Using existing database infrastructure to manage XML data demands seems logical. However, new, native XML database products are built to handle XML data demands natively without the baggage of converting to other database structures such as the relational structure. In addition, a variety of XML storage strategies, conversion processes, and levels of support for XML with the leading database products [**1**] exist.

XML Storage Requirements

XML documents and data storage requirements can often be thought of in two general categories: data-centric and document-centric data. Data-centric demands include typical invoice documents, purchase orders, and more loosely structured documents and is appropriate for items such as newspaper content, articles, and advertisements. If the XML document has a well-defined structure and contains updateable data used in diverse ways, the document is typically data-centric. Document-centric data tends to be more unpredictable in size and content than data-centric data which is highly structured with limited size data types and less flexible rules for optional fields and content [**1**].

XML storage systems must efficiently accommodate both of these types of data requirements since XML is being widely utilized in systems that manage both types of data. Most products focus on servicing one of these data formats better than the other. Traditional relational databases are typically better at dealing with data-centric requirements while content management and document management systems are typically better at storing document centric data. In systems that encounter XML, it is not uncommon to have both categories of data within the same application. Data-centric documents can originate in relational databases or as an XML document that may have been transmitted as part of a B2B system. Database systems must be able to expose the relational data as XML and store the received XML as relational data in order to seamlessly transfer, retrieve, and store the data required by the application.

Although it is possible to store document-centric data in a relational or object-oriented database, this will usually result in duplicating the work of a content management system. Similarly, since a content management system is usually built on top of an object-oriented or hierarchical database, trying to use it as a database will probably prove frustrating [**1**].

The difference between data-centric and document-centric data is not always easy to identify. For example, a purchase order might contain unstructured data such as comments or notes. In addition, document-centric data, such as magazine articles, might contain structured data such as the author's name and the date. Typically the requirements driving the data lean more heavily in one direction or the other. Therefore, taking the time to understand whether data is more document or data-centric will be advantageous when choosing the best storage strategy.

Strategies For Storing XML Data

In order to move data between an XML document and a database and vice versa, the data in the database must be mapped to the structure of the XML document. In the simple form, the entire XML document is mapped to a single column in the database. More complex strategies include mapping each element to a corresponding column in the database or mapping the structure of the XML document to the database [**1**].

The existing strategies can be broken down into three basic methods:

1. **Store the entire document in text form, such as a binary large object (BLOB) in a relational database.** This is a good strategy if the XML document contains static content that will only be modified when the entire document is replaced. This strategy is common for document-centric data as this data is typically retrieved and stored as a whole. All three leading relational database vendors (Microsoft SQL Server, Oracle Oracle8i, IBM DB2) support this method. Storing the entire document in text form is easy to implement since no mapping or translation is necessary, but it may also limit the searching, indexing, and granularity of the retrieval of the XML documents. On the other hand, the XML document remains as it was before it was stored, which minimizes the work to reassemble the document after retrieval.
2. **Store a modified form of the entire document in the file system.** This method is popular when the number of documents is small and the XML documents are infrequently updated and transferred between file systems. Again, the leading database vendors support this method, but it has obvious limitations that include scalability, flexibility in storage and retrieval, and security since the data sits outside of the database. Storing a modified form of the entire document in the file system is very limited since the file system does not make a very good database. This method works for a small number of XML documents and would typically be included in the design only as the XML document is moving through the system. Eventually, the contents of the XML document would end up in a database.
3. **Map the structure of the data in the document to the database.** For example, map an XML document containing a sales order to tables like ***Orders***, ***Items***, ***Parts***, and ***Customers*** or objects like *Order*, *Item*, *Part*, and *Customer*. If the structure of the XML document is not compatible with the structure of the database, the document must be transformed to match the structure of the database before storing it. Mapping the structure of the data in the document to the database is a very popular option and is the focus of many of the new XML enabling features in the leading database products and the subject of many

articles and books in the last couple of years [1]. The following sections will discuss this method in more detail.

Some common approaches, described in the next section, are emerging to take an existing relational database and map it or move it to an XML document and schema.

The Mapping And Translation Process

In order to model a relational database structure as XML, the XML structure must be able to accommodate a few basic concepts including primary keys, foreign keys, tables, and columns. This process can go to extremes in two different directions. In one direction an attempt can be made to map the relational database to an industry-standard schema, which typically involves a significant amount of custom programming to translate the relational schema to the industry- standard XML schema. At the other extreme, the existing relational database could be dumped into a default XML structure with elements representing columns and attributes representing column values. Most efforts will land somewhere in the middle [2].

The mapping and translations process can be broken down into a few basic steps. The first step is to create an XML schema with an element for each table and corresponding attributes for each non-key column that will be mapped. Columns that do not allow nulls can be marked required while those that do allow nulls can be marked as optional in the XML schema. Columns can also be nested as elements, but issues can arise when the same column name is used in more than one table. Therefore the more straightforward approach is to map columns as XML attributes where name collisions in the XML schema are not an issue [12].

The second step is to create the primary keys in the XML schema. One approach would be to add an attribute for the primary key column with ID appended to the column name. This attribute would need to be defined in the XML schema as type ID. Collision issues can arise again when creating primary keys within the XML schema. Unlike relational databases where a primary key only needs to be unique within a table, an ID attribute within an XML document must be unique across the entire document. One way to work around this issue is to append the element name (table name) to the primary key value (attribute value). This ensures the value is unique across the XML document [12].

The third and final step is to model the foreign key relationships. This can be

accomplished by nesting elements under the parent element. Alternately, an XML schema ID can be used to point to a corresponding XML structure containing an IDREF. Trade-offs exist between these two approaches related to performance and navigation ability through the XML document [**12**].

XML Document (Schema)	Relational Schema
Element	Table
Attribute or Nested Element	Column
ID Attribute	Primary Key
IDREF or Nested Element	Foreign Key
#REQUIRED, #IMPLIED	NULL, NOT NULL

Table 1: Summary of XML Schema and Relational Schema Relationship [**12**]

Many variations on XML schemas could be used to represent the same relational database. By understanding the capabilities and techniques available when constructing the XML schema, both the resulting document size and processing time of the document can be minimized. This understanding requires detailed knowledge of the capabilities of XML schemas as well as a thorough understanding of how the DOM and SAX API will actually process the XML document. These APIs are the de-facto standards for processing XML documents.

XML-Enabled Database Leaders

Many of the major database vendors are incorporating XML support into their products or providing tools for using XML within their databases. Some of the leading database vendors such as Oracle's Oracle8i, Microsoft's SQL Server 2000, and IBMs DB2 have much in common in terms of XML support. They all allow for an XML document to be stored as a single column in the database with limited indexing and searching of this document. They also allow an XML document to be broken apart into columns and tables in the database. The tools and techniques offered to achieve retrieval and storage of the XML data varies as does the ease with which developers can translate the XML data to a relational structure [**4**].

Oracle8i

Oracle8i can store and retrieve entire XML documents as columns, can access XML stored in external files or on the Web and can map XML document elements to tables and columns in the database[**11**].

Oracle's interMedia product provides full-text indexing of documents and the ability to perform SQL queries over documents. In addition, Oracle's interMedia product can search text, HTML, and XMLdocuments in a database, an external file, or on the Web. It can also use indexes that are part of an Oracle database to search documents that are internal or external to the database [**8**].

Oracle also offers an XML SQL utility for Java. This tool is a set of Java classes that allows XML data to be inserted into tables or object views. The Java classes can also generate XML documents from SQL query results. An XSQL Java Servlet is also available from Oracle that allows SQL queries to be executed to return results as XML and then transform the XML to HTML using style sheets. This set of tools can be effectively utilized in a relatively fast manner, when in the hands of an experienced developer with both intermediate Java and Oracle skills. Below is an example that utilizes the XML SQL Utility to create an XML document. In Figure 1 the XML SQL Utility is shown executing a SQL query against the Oracle database and then generating an XML document from the results of the SQL query.

```
SELECT CUSTNO, CUSTNAME FROM CUSTOMER WHERE CUSTNO = 1234;

<"xml version=1.0">

<ROWSET>

        <ROW id="1">
                        <CUSTNO>1234</CUSTNO>

                        <CUSTNAME> AJAX INC </CUSTNAME>
        </ROW>

</ROWSET>
```

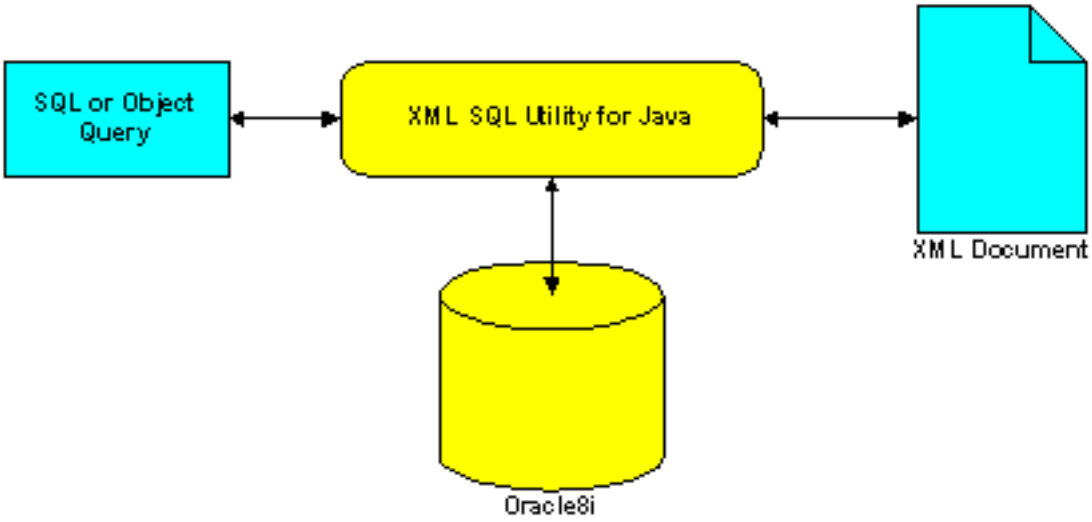

Figure 1

By default, ROWSET is the element name of the XML document element. ROW is the element name of each row in the query result. Column names are used for the element names. An application could change the default tag names by applying an XSL stylesheet to perform a translation. The utility can also generate either a string representation of the XML document or an in-memory XML DOM tree of elements. Additionally, the XML utility can be used to generate a DTD based on the schema of the underlying table being queried [**11**].

Oracle offers a rich implementation of XML support that provides access programmatically to data structured as XML. Features are provided for document-centric and data-centric structures.

IBMs DB2

IBMs DB2 database product offers XML support via their DB2 XML Extender product. XML Extender provides new functions that allow storage and manipulation of XML documents. Entire XML documents can be stored in DB2 databases as character data, as single column, or as external files, and can still be managed by DB2. Functions are provided to retrieve either the entire XML document or individual elements or attributes of the XML document. The XML Extender product also serves as a repository for the management of DTDs. Much like the other leading database products, DB2 stores an XML document as a single column or maps the XML document to multiple tables and columns.

A unique feature of DB2 is the ability to manage and index XML documents disparately

located within the file system, a single column, or spread across multiple tables and columns. The XML Extender product is designed to provide fast search capabilities against XML pages (files). This is useful for applications that share XML files or applications that search against XML files such as search engines. Indexing is performed using Data Access Definition (DAD) to define the XML elements and attributes that should be indexed. User-defined functions (UDFs) can be used to insert, select, or update an entire document or elements and attributes within a document. DAD is also used to define the mapping of DTD to the relational tables and columns [**3**].

Stored procedures are implemented as part of the XML Extender that allows XML documents to be retrieved or generated based on SQL queries. A DTD is mapped against the relational tables using DAD to provide the structure of the generated document. Dynamic queries are also supported that build the XML document based on the table names and column names in the query instead of the mapping defined in the DAD [**3**].

Microsoft's SQL Server 2000

SQL Server 2000 has introduced many new features focusing on XML support as well. SQL Server is providing support for XML Data Schemas, the ability to execute XPath queries, and the ability to retrieve and write XML data. SELECT statement result sets can be returned as XML documents by using a new `FOR XML` keyword. Multiple retrieval modes or formats are supported that vary the interpretation of the SELECT statement and the resulting XML document structure with increasing levels of control over the resulting XML document. The XML tree hierarchy can be automatically determined by the order of the columns in the SELECT statement and their corresponding table names. Alternatively, the developer can control the detailed content and structure of the XML document directly in the SELECT query. Columns can be mapped as elements or attributes or a schema can be specified to provide the mapping [**10**].

SQL Server has also added new features to support writing XML documents from the file system to the database using a new OpenXML function. To use the OpenXML function, an internal representation of the XML document must be created in memory using a new system stored procedure. The OpenXML function references the XML document in memory, specifies an XPath query to filter or locate the XML data within the document, and specifies whether relational columns in the database will be mapped to XML document elements or attributes. A schema can also be specified to determine the mapping between the XML document and the database tables and columns. This

basically allows the developer to load an XML document in memory and then use both SQL and XPath to return query data from the XML document in memory [**10**].

All three leading vendors offer similar capabilities for handling XML. Oracle offers a rich programming interface to XML data using the Java classes, servlets, and utilities. Microsoft's SQL Server offers the most flexibility for retrieving and structuring data as XML through the various modes described above. All three support the most common storage strategies although Oracle's object-relational capabilities may provide more flexibility in this area and prevent some of the additional XML translation that may be necessary in the other two products discussed. IBMs DB2 product appears to be a step behind the competition with their XML support; although it is possible to support the three common storage strategies, this tool's support for the strategies does not appear to be as rich.

There are some who criticize using the relational model for XML storage. The most common criticism is the overhead and extra programming associated with mapping and translating XML structured data into relational data. The hierarchical structure of XML documents is also somewhat of a mismatch to relational databases. The hierarchical structure of an XML document is typically represented by joins in a relational database. Numerous levels of nesting in XML documents can cause significant performance and design problems in relational databases when the nested XML document results in additional tables and join operations that can quickly degrade performance [**4**].

Proponents of using mature, relational databases to handle an organization's XML storage demands are quick to point to the XML storage and retrieval features that they have quickly implemented to make the task much easier on the DBA and the developer. The option to switch to a new native XML database as discussed in the next section is a decision most organizations will not make quickly unless it is an application-specific decision or a niche solution. Too much has been invested in training, procedures, and existing applications for organizations to abandon their current database architecture. Proponents also point to transactional capabilities, security, backup, recovery, and management tools in the existing relational database products that will probably take years for the new native XML database vendors to fully implement [**4**].

There are however a number of reasons to use existing database types and existing database products to store XML even if it is not in its native form. First, ordinary

relational and object-oriented databases are well-known, while native XML databases are new. Second, as a result of familiarity with relational and object-oriented databases, users understand their behavior, especially with regard to performance [4].

Native XML Databases

Due to both the real and perceived limitations of using relational database technology to store XML, new classes of databases are emerging as *native* XML databases. These databases differentiate themselves and earn the term *native* because they store the data structured as XML without the need to translate the data to a relational or object database structure.

Native XML databases are designed to work with XQL (eXtensible Query Language), which serves a similar purpose as SQL does in a relational database. XQL is designed to work with hierarchically structured XML documents and can provide querying features such as filters and joins. XML schemas are implemented in native XML databases to record rules for storing and indexing data and to provide data retrieval and storage information to the native XML database engines. In addition, all objects in a native XML database are typically directly accessible via a URL [6]. Working with a native XML database involves two basic steps: (1) Describing the data via Document Type Definitions (DTD) or XML schemas and (2) Defining a Native Database XML schema or Data Map to use for storage and retrieval [6].

The native XML product viewed as the leader in the market is Tamino from Software AG. By reviewing the capabilities of Tamino, the benefits of working with XML natively become clearer and it provides a view into where the leading database vendors are probably heading with native XML support. Tamino provides both native XML storage and SQL relational storage engine within the same product. This feature lets users query heterogeneous data via XQL and receive result sets in XML format [7].

When storing and retrieving XML documents using the XML engine, no translation to a relational format is necessary. Tamino can store nearly any type of document including XML formatted information, HTML pages, letters, spreadsheets, audio, video, still images, and data from SQL or object databases. Other XML native databases include dbXML, eXcelon, and x-Hive/DB.

Conclusions And Summary

Use of XML will continue to grow as organizations continue to find new uses for recording structure data. The need for XML databases to handle the data requirements of these applications will certainly follow.

It probably will not be long until the leading relational database vendors offer native XML storage and retrieval capabilities along with their current SQL and relational capabilities. This will probably go beyond translation to natively storing XML in the database. The XML processing capabilities that leading database vendors have added in the last two years are a natural progression toward this architecture. In the meantime, XML-intensive applications that require very fast and flexible XML storage and retrieval capabilities may be enticed to implement one of the new native XML database products. Larger IT organizations with heavy investments in leading database products will probably get by with the current XML capabilities of their database products and not bet the enterprise on more immature native XML storage products [**4**].

Even though native XML support may address scalability issues and reduce the amount of work involved by eliminating translation and mapping activities, there are still numerous issues with managing XML data. XML data is much more verbose. An XML document, in contrast to a tab or comma delimited record-based file includes tags for every field in addition to data values. This fact has implications for storage capacity requirements as well as for transmission and processing performance.

The impact on organizations due to the growth of XML should not be taken lightly. The obvious benefits of using databases to store XML can be out-weighed by improper and short-sighted use of the technology that results in systems that are difficult to maintain, inflexible, perform poorly, and become difficult to integrate. The support for XML in database management systems is growing. IT professionals need to understand the capabilities to reap the many benefits the technology has to offer.

Glossary

BLOB - Binary Large Object. A large file, typically an image or sound file, that must be handled in a special way because of its size.

B2B - Business to Business. The exchange of products, services, or information between businesses rather than between businesses and consumers.

DBA - A Database Administrator directs or performs all activities related to maintaining

a successful database environment.

DOM - The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.

DTD - Document Type Definition: A predecessor to XML Schemas. Defines the structure of an XML document.

SQL - SQL (Structured Query Language) is a standard programming language for getting information from and updating a database.

XML - The Extensible Markup Language (XML) is the universal format for structured documents and data on the Web.

XML Schema - Provide a means for defining the structure, content and semantics of XML documents.

XPath - XPath is a language for addressing parts of an XML document.

XSLT - A language for transforming XML documents.

References

1

Bourret, Ronald, *XML and Databases*, September, 1999, <**http://www.rpbourret.com/xml/XMLAndDatabases.htm**>(January 10, 2001).

2

Buck, Lee, *Modeling Relational Data in XML*, < **http://www.extensibility.com/tibco/resources/modeling.htm**> February 19, 2001).

3

Cheng, Josephine and Xu, Jane, *IBM DB2 XML Extender: An End to End Solution for Storing and Retrieving XML Documents*, Paper presented at ICDE 2000 Conference, San Diego, CA., <

4

Dejesus, Edmund, *XML Enters the DBMS Arena*, ComputerWorld, < **http://www.computerworld.com/cwi/story/0,1199,NAV63_STO53026,00.htm**> (January 25, 2001).

5

Didier, Marting, *Professional XML*, Wrox Press Ltd. <> (2000).

6

Hess, D.A., *Software AG, Inc. Tamino*, Datapro Information Services, a subsidiary of Gartner Group., <> (August 2000).

7

Lewis, William, *Pillar of the Community*, August 18, 2000,Intelligent Enterprise, <**http://www.intelligententerprise.com/000818/feat1.shtml**> (February 1, 2001).

8

North, Ken, *Oracle: Powered by XML and Java* <**http://www.devx.com/upload/free/features/xml/1999/01win99/knwin99/knwin99.asp**> (January 20, 2001).

9

Shanmugasundaram, Jayavel, *Relational Databases for Querying XML Documents: Limitations and Opportunities*, Department of Computer Sciences, University of Wisconsin-Madison.<> (2000).

10

Wahlin, Dan, *Customize XML Data with SQL Server*, XML Magazine, <> (February/March 2001).

11

Wait, Brad, *Using XML in Oracle Database Applications*,Oracle Corporation, <**http://technet.oracle.com/tech/xml/info/htdocs/otnwp/about_oracle_xml_products.htm**> (March 6, 2001).

12

Williams, Kevin, *ML Structures for Existing Databases*,WROX Press Ltd., <> (2001).

Biography

Jerry Emerick (**gjemerick@yahoo.com**) is a recent graduate of Grand Valley State University and an IT Project Manager for Gordon Food Service, a private company considered a technology leader in their industry. His research interests include database technology, XML, and Object Oriented Analysis and Design. He is also currently participating in eBusiness projects focusing on customer facing, Extranet applications.