

# Storing XML in Databases

By Michael Champion

As eXtensible Markup Language (XML) becomes the data standard for e-business, the amount of XML data being exchanged will grow exponentially. While enterprises may initially conceive of those XML documents and messages as transitory images of data in a legacy infrastructure, a requirement to store that XML data for fast and accurate retrieval is likely. This will happen because:

- Only the XML view of the data will correspond to the e-business transactions the end user actually carries out. It will be easier to analyze and audit operations based on the unified XML view than the fragmented transac-

tions in the diverse back-end systems.

- Generating an XML view of data in an underlying system can be expensive; caching it in XML format may maximize performance where the same information is accessed repeatedly.
- Once there's a requirement for persistent XML storage, the features of a Database Management System (DBMS) will also be required. The "value proposition" for a DBMS is the same for XML data as any other type of data. A DBMS provides services to an application that are too critical, too difficult, or too costly to be left to application developers. These include the ability to commit or rollback transactions, efficient

XML	RDBMS (Normalized)
<ul style="list-style-type: none"> <li>• Data in single hierarchical structure</li> <li>• Nodes have element and/or attribute values</li> <li>• Elements can be nested</li> <li>• Elements are ordered</li> <li>• Schema optional</li> <li>• Direct storage/retrieval of simple docs</li> <li>• Query with XML standards</li> </ul>	<ul style="list-style-type: none"> <li>• Data in multiple tables</li> <li>• Cells have a single value</li> <li>• Atomic cell values</li> <li>• Row/column order not defined</li> <li>• Schema required</li> <li>• Joins necessary to retrieve simple docs</li> <li>• Query with SQL retrofitted for XML</li> </ul>

Figure 1 — Mismatches Between XML Data and Relational Databases

queries, scalability, and backup and restoration utilities.

This article explains how native-XML databases efficiently and reliably store XML content better than relational or partial XML databases. It includes a “walk-through” of the storage and retrieval process as it supports an e-business transaction.

## XML Database Management Approaches

While there are good reasons to store XML persistently in a database, this presents a challenge. There are fundamental mismatches between the XML-structured data and the data model virtually all mainstream RDBMS products support. Entire books have been written on this subject, but Figure 1 summarizes the essential differences.

You can think of solutions to the challenge as points along a spectrum from the “pure relational” approach to “post relational” approach (though most real-world systems are a combination).

### Pure Relational

The relational model of databases offers one answer: Normalize the XML data into rows and columns, with each cell containing an “atomic” text value. Any hierarchical or network data model can be translated into normalized relations, so in principle any document can be decomposed for relational storage. The techniques for normalizing tree structures are discussed in advanced RDBMS textbooks. Similarly, Wrox Press’s *Professional XML Databases* provides a checklist of 18 rules (Chapter 3) that give detailed suggestions about how to mode a structure specified by an XML DTD in a relational database.

The problem is that there’s a significant gap between relational model principles and the actual practice of RDBMS vendors and users. Normalizing XML structures into RDBMS relations can be fiendishly complex for designers, time-consuming for programmers, and operationally inefficient for Database Administrators (DBAs) or users. The more “document-like” the data model — that is, when there are recursive elements, mixed content, and a less rigid structure — the more difficult it is to devise practical RDBMS models for XML data. Furthermore, there are well-known challenges in using Structured Query

Language (SQL) to effectively query normalized recursive data models such as a bill of material. Relational purists use the challenges of building bills of material applications to illustrate SQL’s lack of adherence to the theories underlying the relational model (e.g., Fabian Pascal’s *Practical Issues in Database Management: A Reference for the Thinking Practitioner*, Chapter 7). Ordinary users can be forgiven for being daunted by the challenges of effectively normalizing and querying hierarchical XML documents using today’s SQL databases!

### Post-Relational

Relational database vendors have responded to the challenges of serializing the data from object-oriented pro-

There are mismatches between the XML-structured data and the data model RDBMS products support.

grams and object-oriented databases by adding features useful for simplifying XML data management. Most fundamentally, RDBMS systems have added Large Object (LOB) data types that allow arbitrary types and amounts of data to be stored and retrieved in a single “cell” of a table. Similarly, RDBMS vendors (and the SQL standard) have added other “post-relational” features such as:

- Support for “cells” containing repeating groups of data
- Full-text search capabilities.

These features make it easier for non-specialists to build effective database applications that don’t fit the constraints of the pure relational model. It’s not a big stretch for the RDBMS turned “object-relational” vendors to add convenient XML extensions to their products that exploit new post-relational and text-retrieval features. Additional utili-

ties ease the burden of modeling XML hierarchies to work with the underlying relational and post-relational storage models. World Wide Web Consortium (W3C) recommendations for XML, such as the XPath query syntax and the Document Object Model (DOM) Application Program Interface (API), are primarily supported in the utilities. Once SQL or proprietary text-search extensions have located records, the XML utilities provide tools for representing the results as XML and manipulating the XML with DOM, Xpath, etc. Current XML-enabled database systems don’t support a complete, seamless round trip of arbitrary XML content into and out of a database. None of them support the complete XPath specification as a query language into the database itself.

We see several features added to object-relational databases to make it relatively easy to store and retrieve XML data. Different vendors take different approaches, but all add XML support on top of existing features rather than as a fundamentally new storage model inside the database engine. Even Oracle 9i’s “native XML SQL data type” is essentially just a Character Large Object (CLOB) that supports some proprietary extensions to SQL for XML processing.

### Native XML

To work with XML data in a mainstream DBMS, either the end user or the DBMS vendor must use relatively sophisticated techniques to overcome the mismatch between XML data and existing technologies. An alternative approach is to build a DBMS from the bottom up to easily store, retrieve, and query XML-structured data. Such “native XML” database systems expose the data and the processing model via XML standards. An XML document is the fundamental unit of storage. XML DTDs or schemas, rather than RDBMS schemas, define the properties of document collections. XPath or another XML-specific query language locates documents meeting some search criteria. Some products allow XML data to be processed in the actual database engine (as opposed to some external utility) with Simple API for XML (SAX), DOM, and XML Style-sheet Language Transformations (XSLTs), XLink, etc.

A native XML database doesn’t ask the user to worry about how to map

XML structures onto some non-XML underlying data model or processing language. No native XML database product supports every detail of every XML specification, but the XML standards define a large percentage of the interfaces to native XML DBMS products. There's little need for a 1,000-page book on native XML database programming, because those familiar with XML specifications and tools already know about 95 percent of what they need to know to use native XML database products!

This conceptual advantage of using a native XML DBMS to handle XML data can translate into concrete operational advantages, too. The code required to partition XML documents into multiple tables and CLOBs and to translate queries and merge results from multiple underlying database structures provides additional processing overhead and points of failure compared with the more straightforward native XML approach. While performance and reliability depend on a complex mixture of factors, native XML databases provide a more scalable, reliable platform in which to store XML than object-relational databases. Since native XML databases tend to focus on doing one thing well rather than being a "universal database" solution, the workload for DBAs and Web administrators is considerably reduced. In short, the "total cost of ownership," (considering the hardware, programming, administration, and training) is likely to be significantly lower for XML applications built on a native XML database.

### Choosing an RDBMS or XMLDB

Of course, native XML databases aren't a universal solution for all data management needs — or even XML data management needs. When is an RDBMS the more appropriate back-end or when is a native XML DBMS the better choice? RDBMS systems are probably best for maintaining the integrity of data, and XML DBMS systems are best for maintaining XML documents. The distinction between "documents" and "data" is fuzzy and the rise of XML has further blurred the boundary. Let's try to clarify it a bit.

"Data" describes propositions about the world. RDBMS systems (to the extent they're truly based on E. F. Codd's

relational model of data) provide a well-established methodology for maintaining the logical consistency of these propositions. For example, the relational model of data provides a good methodology to keep information on customers, orders, fulfillment status, and accounts receivable in a consistent state. In a properly normalized database, a change to a customer's address will automatically be reflected in the address associated with that customer's orders.

XML has mechanisms that let a developer minimize redundant storage of common data (e.g., the customer address needed for order fulfillment) and to include this common data via external entity references or XLink expressions. Nevertheless, there's no


## Native XML databases aren't a universal solution for all data management needs.

well-established formal model or methodology for this and XML databases generally leave the processing of entity references or XLink expressions to the calling application. Even in a world where many customer transactions are conducted via XML, it probably makes sense to maintain mission-critical data in RDBMS systems. That is because the relational model has been developed and refined to maintain the crucial logical consistency among various propositions about business reality in the database.

Soon, however, many of the actual interactions between producers and consumers will be conducted via XML messages (SOAP-based Web services, asynchronous ebXML messages, etc.). So those orders, cancellations, credit checks, requests for quotations, invoices, etc. are documents that are the electronic equivalent of paper business documents. Such documents may be generated from data in an RDBMS, but once

produced, they must maintain a different conception of "integrity." The document must reflect the snapshot of reality that produced it, even if "reality" changes.

Consider again customer orders. Orders may reflect binding contracts between the producer and consumer. In that case, it's usually desirable to maintain the history of orders exactly as received, possibly guaranteed by a digital signature. Normalizing the XML data reflecting a digitally signed order for storage would make this impossible. In this case, we don't want to invalidate the digital signature on the order actually received to process a customer's change of address. Of course, XML-enabled RDBMS products generally allow storage of XML in round-trip form as a CLOB. However, there are significant restrictions on the ability to perform queries on CLOB data. Native XML databases, on the other hand, are well-suited for an application such as maintaining a reliable, non-repudiable log of XML business transactions that supports queries.

Furthermore, "documents" (as opposed to "data") may be produced by and written for humans, so it will tend to contain XML structures that are difficult to normalize into RDBMS form. The structures may also be complicated to extract and query using post-relational tools. Books that describe how to work with XML in RDBMS systems generally suggest that designers avoid mixed content and recursive content models in XML schema that will be stored relationally. We can easily turn around the argument and suggest that real-world XML content that's difficult to store relationally can be stored much more easily and efficiently in a native XML database! 

### About the Author



*Michael Champion is a member of the W3C's Document Object Model Working Group and co-editor of the core XML portion of the DOM Level 1 recommendation. He is currently senior advisor, New Technologies R&D, at Software AG, Inc. Voice: 734-905-1838; e-Mail: Mike.Champion@SoftwareAG-USA.com; Website: www.softwareagusa.com.*