The Age of the XML Database

By David McGoveran

No matter your job (Web developer, database administrator or designer, or an application or B2B integrator), it's time to learn about eXtensible Markup Language (XML) databases.

An XML database is a database that has XML documents stored in it, regardless of how. You need to know what these databases can and cannot do for you and when to use them. The number of database products capable of supporting XML is increasing rapidly and the product marketing and publicity machines are in full gear. Sadly, the claims being made are self-serving. Many vendors with an XML capability will tell you how XDBMS (XML DBMS) products are going to erode the Relational Database Management Systems (RDBMSes) market. XML consultants and some analysts preach the same sermon, with some claiming that XDBMSes will replace the RDBMS market altogether. Other vendors see XDBMSes as an unnecessary fad, doomed to vanish.

To sort out the truth from the flurry of unreasoned argument, you'll need to understand some of the jargon surrounding the technology. That alone can be a difficult task since each XML database vendor, consultant, or developer probably has something a little different in mind. This article is intended to help with the task.

Let's start by understanding the word *database* as clearly differentiated from a database management system (DBMS). By database, we mean an organized collection of data. By DBMS, we mean the software used to manage (including access) that data and its organization. We won't use database and DBMS interchangeably. It's both amusing and annoying that DBMS vendors often think they sell organized collections of data! Databases and DBMSes can be classified according to the data organizations, including hierarchical, network, flat-file, relational, object-oriented, and others. To this list, we may now add XML organization, a variant of hierarchical. Only RDBMSes support more than one physical organization inherently. This is a perpetual

source of confusion in the XML database debate and seems particularly beyond the grasp of the pundits of "new" types of DBMSes. We'll revisit the use of multiple physical organizations later, but for now let's focus on understanding XML database implementations.

Native XML Support

By native XML support in a DBMS, vendors and XML gurus may mean one or more of several things. Certainly, most will mean that XML documents are stored intact physically as XML. They almost always mean that each XML document is stored in its entirety in a single storage location as well. The practicalities of this latter requirement are questionable. File systems don't permit storage of objects of arbitrary size in a single file, so storing all XML documents without permitting some form of decomposition will have scalability limits. For most purposes, you can safely put this particular issue aside.

For the purist, native XML support also means that all manipulation — including searching, retrieval, inserting, updating, and deleting — is done using an XML language operating on standard XML objects and structures. The emerging standard query language is XPath, but you may encounter several proprietary and proposed standard XML query languages. Examples include XQL, X-Query, and XML-QL. Manipulation using an XML language is of obvious value when all you're concerned with is XML documents. This might be the case in, for example, a new Web-based, business-to-business (B2B) application. By contrast, an XML language can be unnatural if XML documents are the exception. An example there would be using XML to integrate two existing enterprise applications.

Finally, native XML support is sometimes taken to mean that the schema is derived automatically from the XML document, or at least that XML schema information is preserved and associated with the XML document. Schema information can be preserved through embedded tags, Document Type Definitions (DTDs), or XML schema. Remember that XML is both data and metadata. Internally, XML uses tags and has a self-defining hierarchical or nested structure. In addition, there may be an external or separate schema definition (using DTDs or the XML schema language). Naively considered, this XML property differentiates it from other types of data. While there's always some mechanism by which data types and schemas are specified for the DBMS, XML requires that the DBMS permit schema definition when the document is stored.

A conceptually simple test for the faithfulness of native XML support is called "round-tripping." This functionally descriptive name means that storing an XML document won't corrupt it. That is, if you store an XML document in the database and then later retrieve that document, you'll recover exactly what you stored. This property is important. Indeed, it's surprising that anyone would consider a product that lacks this property — and so corrupted data that was stored in it — to be a DBMS at all. Data integrity is a fundamental, defining property of information systems and anything less is unacceptable.

Imagine a vendor saying that you could store a "5" in their RDBMS, but that you might get back a "10!" Yet the equivalent change of values is exactly what happens with some XML database implementations. Even worse, because XML documents are "self-defining," it's possible for the schema to be corrupted, meaning that the document will be misinterpreted.

Most DBMSes that provide native XML support were built from the bottom up to support XML as the storage organization. XML is inherently hierarchical. It requires a hierarchical storage management facility such as that found in old-style hierarchical DBMSes or more recent object-oriented DBMSes. Early versions of XDBMSes lacked important features and functions of mature DBMSes in the areas of:

- Integrity
- Transaction management
- Storage efficiency
- Indexing
- High levels of concurrency
- Multi-document query
- Performance
- Scalability.

Even today's versions may be deficient in these important features. Indeed, they may be so limited in terms of transaction rates, levels of concurrency, and document sizes that they would not impress users of other types of DBMSes that commonly support strategic applications.

Early versions of XDBMSes lacked important features and functions of mature DBMSes.

Non-Native XML Support

Many RDBMS vendors, when they initially added XML support, took the XML document decomposition approach to storage. (XDBMSes often support it, too.) Parsing on insert and recomposing (or rendering) during retrieval consumes both time and processing resources. Some products also permit the document to be indexed automatically or on request. Decomposition techniques can vary considerably. For example, a mapping to the existing schema might determine which portions of the XML document are stored as records or rows of a particular type. Alternatively, each XML document might be treated as implying a new schema so that new tables or other data structures are created during storage.

The rendering process should recreate the XML document exactly as it was when stored (barring any interim modifications). Because of deficiencies in the XML document parsing process, or because schema information was not supplied, faithful document recomposi-

tion may be impossible.

If a document can be corrupted, why would anyone want to decompose it? There are several reasons. First, decomposing a document makes it easier to support multi-document query, multi-document integrity or consistency, fine-grained security, and so on. Second, decomposing into a standard schema aids uniformity and consistency when data comes in multiple formats or from multiple sources. For example, traditional business transactions are likely to be recorded in an RDBMS. If e-business transactions are then captured as XML documents, decomposing them in the existing RDBMS makes it possible to report on all business transactions and, therefore, to guarantee that they're processed exactly once. Similarly, addresses or credit information can be more easily verified and corrected.

RDBMS XML Support

You don't have to decompose XML documents to store them in a relational database — unless usage demands it. Support for complex, possibly highly structured data types (called domains) such as XML is inherent in the relational model and is becoming more prevalent in today's RDBMSes. RDBMS can support both native XML (documents as atomic values of complex data types) or non-native (decomposed) XML, depending on the product. If round-tripping is supported, users need not even know if the XML document has been decomposed. The unique relational separation between the physical schema and the logical schema guarantees that data can be stored in various ways while maintaining a consistent, logical view of columns, rows, and tables. Some users might see only the intact document, even though it has been logically decomposed. It's not free, but it's powerful.

Decomposing can have other positive effects, too. RDBMSes use a query language based on logic, enabling the use of a query optimizer. The optimizer enhances each request via logical and semantic simplification, and selects from among available access methods and operation sequences to achieve optimized performance. It does so in a way that's sensitive to the amount of data stored, its characteristics, and its organization. Procedural query languages such as XPath have no hope of such optimization since they dictate access through a fixed structure and sequence of operations.

Certain XML applications are best supported by an XDBMS.

Most RDBMSes can also be used to generate XML documents. This means that they can be used as the primary source of new XML documents, with the creation possibly initiated by some business event, a user query, or a database trigger. This functionality is particularly useful for dynamic Web content creation, enterprise portals, and message creation where the data source isn't XML and may be the enterprise database-of-record.

Why Use an XDBMS?

An XDBMS is a specialized type of DBMS. By definition, it's designed for XML storage, access, and manipulation that is as efficient as current product design allows. The self-defining and hierarchical character of XML gives XDBMSes an advantage over other DBMSes when the workload is predominantly XML. XDBMSes will have some advantages over both hierarchical and network DBMSes as well as RDBMSes. For example, if you know the physical schema of an XML database, you'll know how to query and update it. But there are problems with XDBMSes, too. Redundancy can result from:

- Metadata and data entanglement
- Enabling self-defining documents
- Automatic schemas creation.

Such redundancy leads to both storage inefficiencies and consistency nightmares when the schema changes. You can see this if you explore the impact on all your XPath queries or applications when:

- A tag is changed to avoid naming conflicts
- A new hierarchy level is inserted.

Nonetheless, certain XML applications are best supported by an XDBMS.

Presumably, you're using XML for its primary value — data portability. Before you decide on the type of XML database support you need, determine if XML persistence is needed. Many XML applications don't need it because the XML document is produced and consumed immediately. Examples include:

- Dynamic Web content creation and delivery
- Portals into legacy data
- Integrating applications using XML as the data transport
- e-Commerce or B2B application in which XML is used as a data transport.

By contrast, integrating XML and non-XML data, repetitive delivery of

static Web content, and staging XML messages all require some form of XML persistence.

What you do with XML will strongly influence your choice of DBMS as well as how you use that DBMS. Here are some quick guidelines:

- Determine whether there's really a need for XML persistence. Many uses of XML are primarily for data transport (i.e., they're messages). In that case, the XML document is consumed by one or more applications almost as soon as it's delivered and never reused. If this is the case, there's no need for an XDBMS.
- XML persistence can provide XML document reuse, staging for asynchronous message transport, and a history or audit trail. If an XML document is created and then reused without further modification, use a DBMS with native XML support. Such documents are sometimes called document-centric. Document-centric XML is usually designed for human consumption and can be extremely complex. While an RDBMS that supports round-tripping might be used, a dedicated XDBMS may be more effective. If you already have an RDBMS that can be used for the purpose, the choice will come down to considerations of relative performance and administrative overhead vs. the investment in a new product.
- The primary motivation for decomposing an XML document is to gain rapid access to its elements without the overhead of the entire document. Such documents are sometimes called data-centric because the data within the document is the atomic unit. If your XML documents are relatively small and will be modified repeatedly over time, use a DBMS that supports native XML storage and automatic indexing of XML tags. If the elements of an XML document will be modified repeatedly over time, use a DBMS that can decompose the document.
- Keeping XML documents in a database permits the entire collection to be queried, updated, or correlated to other, non-XML data. If the XML document is part of a collection that needs extensive querying, frequent partial updates, or consolidation of relational and non-relational data, use an RDBMS that can decompose the document, index it rapidly and automatically, and that supports mapping to your existing database schema.
- As often noted, XML is becoming the lingua franca of business transactions. Even though the usage might be document-centric, the need for high-volume, transactional, and secure database operations can be a crucial factor. In such cases, you need a DBMS technology with proven transaction, security, and integrity. If you need support for transactions, large numbers of users, fine-grained security, and cross-document integrity enforcement, use an RDBMS.
- XML's self-defining property is both an opportunity and a challenge. On the one hand, it may be possible to understand and use the data in a new type of document without ever having encountered one of that type before. On the other hand, there's no substitute for good data modeling that teases out the semantic relationships among data elements. This XML property can easily become a crutch, with the relationships among documents and their elements completely ignored or, worse, assumed visible through tags. Although RDBMS schemas are relatively easy to extend, most RDBMSes were designed for infrequent schema extensions or modifications and many instances of each entity type. Automatic schema extensions and arbitrary XML document storage is the province of XDBMSes. For now, if you need to store unpredictable types of XML documents, use an XDBMS, but don't ignore data modeling.

This may not be the "age of the XDBMS," but it's almost certainly the "age of the XML database."

Summary

There are many applications that need precisely what XDBMSes provide and, as competition heats up, RDBMS vendors will be forced to improve XML support. XDBMSes are unlikely to significantly dent the RDBMS hold on enterprise databases, but their value and importance cannot be denied. This may not be the "age of the XDBMS," but it's almost certainly the "age of the XML database," meaning that XML database support is coming of age. eAI

About the Author

David McGoveran is president of Alternative Technologies, Inc. He has more than 20 years' experience with mission-critical applications and has authored numerous technical articles on application integration. e-Mail: mcgoveran@alternativetech.com; Website: www.alternativetech.com.