

Tame the Information Tangle

XML data management systems

By Paul Sholtz
New Architect Magazine
October 2002

Two major limitations hinder Web information queries. First, custom, page-specific parsers must preprocess data contained within HTML pages before meaningful information can be extracted. Second, data in traditional database management systems is generally only Web accessible through simple and inflexible forms-based interfaces.

Encoding information in XML and exposing it on the Web will help overcome these hurdles and enable fine-tuned, database-like queries on a global scale. Of course, if all the world's data is to be encoded in XML, we'll need more efficient ways to store and manage large volumes of XML data. To address that need, a new breed of document storage and management systems has appeared that's been specially optimized for publishing XML documents on the Web.

Representing Documents in XML

XML is a very powerful and flexible standard, so it's

important to clearly define how you want to use the XML data before you decide on a document management system. You may require the ability to store and persist XML data for a number of purposes, and different data storage systems may exhibit a range of performance characteristics depending on the structure of your XML, the frequency of document update and retrieval, and so forth. For example, XML is commonly used for machine-to-machine data interchange and processing, where no human being is ever involved. Keep in mind that the methods used to persist and query these types of XML documents may be very different from those used for XML documents designed for Web publishing.

The term "document-centric XML" is used to describe XML documents meant for human consumption. The document-centric approach is used to create XML encodings for books, email messages, advertisements, Web pages, XHTML documents, and many other types of semi-structured and unstructured document data. Note that these documents usually do not originate in the database itself, but instead are created by hand using text editors, word processors, or desktop publishing tools. These tools save the document in some common text-formatting standard, such as RTF, PDF, or SGML, and from there the information can be converted to XML whenever necessary.

As you know, storing the document in XML allows you to separate content from presentation. Updates, changes, and other content transformations can be applied and

Tame the Information Tangle

saved directly to the XML document. The presentation format of your choice (RTF, PDF, HTML, SGML, and so on) can be generated and rendered from the XML document later on. The Apache Cocoon Project defines an XML publishing framework that facilitates these types of transformations and maintains the separation of these concerns. Cocoon was designed to allow developers, business analysts, designers, and system administrators to work together (and independently) without stepping on each other's toes and messing up each other's work. (More information on Cocoon is available at xml.apache.org/cocoon. The sidebar "XML Conversion Tools" lists some freely available programs for converting various types of documents to and from XML format.)

XML documents can also be data-centric, as opposed to document-centric. The term "data-centric XML" describes XML documents that are intended primarily for machine consumption. This case arises in many business-to-business applications, where structured data interchange is common. It also occurs in applications where XML is used to coordinate interprocess communication between distributed objects, such as in SOAP or XML-RPC. In terms of their interaction with the underlying persistence system, the most important difference between these two models is that data-centric XML often originates in the database itself. The degree to which your XML data is document-centric or data-centric will likely impact your choice of data management software.

It's All Relational

The relational database management system (RDBMS) is the king of enterprise computing. Whether you like it or not, as an enterprise developer, you often have no choice but to integrate your software, business logic, and enterprise information with relational databases. Middleware developers know this well. For years, they have struggled to build and maintain complex object-relational mapping between their business logic and relational database tables. Today, a similar challenge awaits developers who intend to use XML as a part of their enterprise applications.

The most important step in this process is to map the schema of your XML document to the schema used in the relational database. The XML schema will be represented in as DTD, XML Schema, RELAX NG, or some other format. The database schema is defined by the structure of the tables that make up the database. Mapping between the document schema and database schema is usually performed on the element types, attributes, and text of the XML document. XML-relational mappings almost always omit some physical structure present in the XML (such as entities, CDATA sections, and encoding information) as well as some logical structure (like processing instructions, comments, and the order in which elements and PCDATA appear in the document). The reason for this has to do with some of the mathematical underpinnings of the relational data model. Although it may sound like important data is being

Tame the Information Tangle

lost, in reality this level of mapping is often sufficient for most applications.

A comprehensive discussion of schema mapping between XML and relational data types is beyond the scope of this article, although it's worth briefly describing two of the most common models used to perform this type of data transformation. The first and simpler way to do schema mapping is called table-based mapping. Here, the structure of the XML document closely mirrors that of the relational database table it is associated with.

Table-based mapping is relatively simple and can only be used for very straightforward schema mappings. A more powerful way to do schema mapping is by performing object-relational mapping. Software that does object-relational mapping for XML documents first takes the XML document and parses it into a tree of objects. The linking and relationships between the objects is defined by the hierarchical structure of the tags in the XML document. Once the object tree is complete, a mapping is made from the object tree to the database tables.

The real world is rarely this simple, for several reasons. First there is seldom a straightforward mapping between database tables and the business logic that is codified into the object structure/XML document structure. Our example is also simplified in that the object in question only consists of three primitive data types (all string types). When more complex data types are included, the

mapping also becomes more complex, and the database system must reference other tables through foreign keys.

Native XML Databases

Don't underestimate the difficulty of mapping XML documents onto relational tables and then back out into XML. Relational data mappings can be error prone, inflexible, and unless they are very carefully constructed they may slow down the application's throughput. If creating and maintaining all these relational data mappings seems like too much work for the scope of your XML application, one attractive alternative is to use a native XML database (NXD). The concept of a native XML database was first introduced by Software AG during the marketing campaign for its Tamino product line. Since then, the term has come into common usage among other companies developing similar products. NXDs are optimized for the storage and management of XML documents. Like other modern data management systems, they provide support for transactions, security, concurrent access, and query languages.

Formally, a native XML database can be defined as a data management system that exhibits the following characteristics:

- XML documents are the fundamental unit of logical storage in the system (similar to the way in which rows in a table are the fundamental unit of logical storage in a relational database system).
- The system defines a logical model for XML

Tame the Information Tangle

documents, and stores and retrieves documents according to that model. At the very least, the model must include support for elements, attributes, PCDATA, and document order. Some examples of logical models that meet these requirements include the XPath data model and the XML InfoSet.

- The system is independent of any underlying physical storage model. For example, it could be implemented using relational, hierarchical, object-oriented, or proprietary storage formats.

NXDs are often a good choice for storing document-centric XML information. For example, NXDs support XML query languages that let you perform highly specialized queries like "find all documents where the second paragraph contains an italicized word." Most NXDs provide other powerful and sophisticated text-searching features, such as thesaurus support, word stubbing (for matching all forms of a word: swim, swam, and swimming, for example), and proximity searches (find all instances where the word "lake" occurs within five words of "swim"). These are extremely useful features when you're working with traditional documents, although they are usually much less important if you are working with data-centric XML information.

There are other reasons you might want to consider using an NXD. Many such repositories are able to understand a DTD or an XML Schema, and can therefore provide data validation on the fly, as information is stored or updated. NXDs can also persist information such as

document order, processing instructions, comments, CDATA sections, and entity usage, while many systems that attempt to store XML data into relational databases cannot.

The following is a list of some additional features commonly supported in native XML databases. Many of these features support the management of document-centric XML information:

Document Collections. In the XML world, the notion of a document collection is similar to the concept of a table in a relational database. It is a grouping of discrete, independent units of logical storage (such as XML documents or rows, respectively) that share common characteristics. For example, your company may store all XML documents related to sales orders in one collection, all XML documents related to product manuals in another collection, and so forth. Some NXDs support the nesting of document collections.

Example 1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="USA">
    <title>Tribute to Britney Spears</title>
    <artist>Various Artists</artist>
    <price>10.98</price>
  </cd>
  <cd country="UK">
```

Tame the Information Tangle

```
<title>SpiceWorld</title>
<artist>The Spice Girls</artist>
<price>11.98</price>
</cd>
<cd country="UK">
  <title>Hot Rocks 1964-71</title>
  <artist>The Rolling Stones</artist>
  <price>32.49</price>
</cd>
</catalog>
```

Query Languages. Again, almost all NXDs support one or more query languages. Support for XPath and XQL, in particular, is quite common. XPath defines a simple language that lets developers address the various parts of an XML document, although XPath itself doesn't use XML in its own syntax. Instead, the expressions that XPath uses to identify nodes in an XML document look very similar to the expressions used when accessing a standard computer filesystem. XPath is commonly used in XSLT and is a W3C standard. To see how XPath addressing conventions work, consider the markup in Example 1.

- To select the ROOT element in this document using XPath, we would write: /catalog
- The following XPath expression selects all the CD elements inside the catalog element: /catalog/cd
- The following XPath expression selects all the price elements inside all the CD elements inside the catalog element: /catalog/cd/price

- XPath also defines a library of standard functions for working with strings, numbers and Boolean expressions. To select all CD elements from the XML document that have a price element greater than 10.98, we would write: /catalog/cd[price>10.98]

XQL is a query language used for retrieving information from XML documents. Despite its name, XQL bears almost no resemblance whatsoever to SQL, the data-query language widely used in the relational database world. For sample XQL statements again see Example 1.

- To select all cd elements in the document, we simply type: cd
- The child operator ("/") indicates hierarchy. To select all title elements that are children of cd elements, we type: cd/title
- To select all nodes where price of the CD is \$10.98, we would write: cd/price="10.98"

Note that unlike XPath, XQL is not a W3C standard. In the future, most native XML databases are expected to support XQuery from the W3C.

Transactions and Concurrency. Most NXDs include support for transactions and rollbacks. Keep in mind, however, that most systems implement locking at the level of the entire XML document, rather than at specific nodes of the document, so the ability to support large numbers of users concurrently may be somewhat limited.

Expect NXD vendors to support more fine-grained

Tame the Information Tangle

locking semantics (at the level of individual document nodes) in future product releases.

Of course, native XML databases are not a cure-all. NXDs aren't optimized for storing large amounts of data, especially when compared to modern relational database management systems. NXDs also have a difficult time managing documents that are interrelated, where one document contains references to another. Some NXDs provide simple translations (usually using XSLT) for stitching together information contained in multiple documents, but such transformations are often inflexible and can create bottlenecks in your system, especially if scalability is a concern.

Also, NXDs don't generally support aggregation. If you use an NXD to persist data, and you want to perform aggregate calculations (for example, calculating sums and averages), you'll probably have to add some business logic to the middle tier. Aggregation is a very important function that's required whenever you're working with structured data, although it's usually much less important for document-centric information.

Hybrid Database Solutions

If you're lucky enough to afford it, the simplest way to go is to use one of a new breed of RDBMS called an "XML-enabled database." An XML-enabled database is a conventional database (usually relational) that has built-in extensions for transferring data to and from XML

documents. Database vendors usually make enhancements to the SQL language and the data-type capabilities of their product to seamlessly support XML. XML-enabled databases tend to be optimized for handling data-centric XML, rather than document-centric XML. This is because data must be transferred to and from user-defined tables, rather than tables designed to represent XML documents. Nevertheless, XML-enabled databases deserve a quick look, as many of them can store document-centric XML information in a single column and use text-processing extensions for queries.

The Oracle 9i XDB supports both XML-enabled and native storage of XML documents and information. In some ways, this product blurs the distinctions between relational data and XML data. It allows users to view relational data as XML and vice versa using several new SQL operators.

The EXTRACTNODE command obtains a document fragment that matches an XPath expression and returns it to the caller as an XMLType data object.

XMLType is a new data type defined in Oracle 9i XDB designed to store an entire XML document. Like any other data type, XMLType columns can be used in both tables and views. When used as part of a view, the XMLType data type lets a developer create an "XML view" of relational data. In other words, it is possible to create a virtual XML document using any other data type present in the database.

Tame the Information Tangle

By issuing an EXTRACTNODE command and manipulating the resulting XMLType data correctly, developers can use XML data as if it were relational data. Some other new commands Oracle has added to SQL in support of XML include XMLSCHEMA (constrains an XMLType column to conform to a certain XML Schema), EXISTSNODE (determines whether a particular node specified by an XPath really exists), XMLTABLE (creates a table from a set of XPath nodes), and XMLTRANSFORM (applies an XSLT style sheet to XMLType data types).

Your Choice

XML holds a great deal of promise for Web applications and developers. Applications that publish information on the Web in the form of XML documents can integrate more easily with other applications using simple and reusable data formats. Moreover, when information is published to the Web as XML, it can be indexed and searched in a much more powerful and fine-grained manner than it can with HTML Web pages.

Developers looking for a way to store and manage XML documents should consider whether their applications are document-centric or data-centric. In general, integration with RDBMS-backed data management systems will make more sense for data-centric XML documents. Integration with NXDs will make more sense for document-centric XML information. Most XML

applications concerned with Web publishing will be document-centric in nature.

--

Contact Paul Sholtz
Phone: 917.438.7087
E-mail: paul.sholtz@doublegemini.com