Introduction to Native XML Databases

October 31, 2001

Kimbro Staken (/pub/au/120)

The need to process and store XML has spawned several new types of software tool, one of which is the "native XML database." This article explains the principles behind such databases.

What is a Native XML Database?

The term "native XML database" (NXD) is deceiving in many ways. In fact many so-called NXDs aren't really standalone databases at all, and don't really store the XML in true native form (i.e. text). To get a better idea of what a NXD really is, let's take a look at the NXD definition offered by the XML:DB Initiative (http://www.xmldb.org), of which the author is a participant.

A native XML database...

- Defines a (logical) model for an XML document -- as opposed to the data in that document -- and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order. Examples of such models are the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX 1.0.
- Has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage.
- Is not required to have any particular underlying physical storage model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files.

There's a lot that we can learn from this definition, but the three main points can be summarized succinctly.

1. The database is specialized for storing XML data and stores all components of the XML model intact.
2. Documents go in and documents come out.
3. A NXD may not actually be a standalone database at all.

As should be clear from this definition, NXDs don't really represent a new low-level database model, and aren't intended to replace existing databases. They're simply a tool intended to assist the developer by providing robust storage and manipulation of XML documents.

Native XML Database Features

While not all NXDs are exactly the same, there is enough similarity to provide a basic discussion of features. We won't discuss any particular products here; if you're interested in evaluating products you can find a comprehensive XML database products list (http://www.rpbourret.com/xml/XMLDatabaseProds.htm) maintained by Ronald Bourret. Most vendors have first generation products available now, with a few of the more mature vendors having second generation products available. The NXD model is still evolving and will continue to evolve for several years. There is significant variation between products so the following sections just provide a high level guide that applies for most current products.

XML Storage

NXDs store XML documents as a unit and will create a model that is closely aligned with XML or one of XML's related technologies like the Infoset (http://www.w3.org/TR/xml-infoset/) or DOM (http://www.w3.org/DOM/). This model includes arbitrary levels of nesting and complexity, as well as complete support for mixed content and semi-structured data. This model is automatically mapped by the NXD into the underlying storage mechanism. The mapping used will insure that the XML specific model of the data is maintained. Once the data is stored you must continue to use the NXD tools if you expect to see a useful representation of the data. For instance, if you're using a NXD that sits on top of a relational database, accessing the data tables directly using SQL would not be as useful as you might expect. The reason for this is simply that the data you will see is the model of the XML document (i.e. elements and attributes) rather then the business entities that the data represents. The business entity model exists within the XML document's domain, not within the domain of the underlying data storage system. To work with the data, you work with it as XML.

If a developer is comfortable working with XML tools such as DOM (http://www.w3.org/DOM/), SAX (http://sax.sourceforge.net/), XPath (http://www.w3.org/TR/xpath) and XSL-T (http://www.w3.org/TR/xslt) then they will probably be comfortable working with a NXD. The database will abstract away all the details of how the XML is stored and leave the developer free to build applications using XML technologies.

Collections

NXDs manage collections of documents, allowing you to query and manipulate those documents as a set. This is very similar to the relational concept of a table. NXDs diverge from the table concept in that not all native XML databases require a schema to be associated with a collection. This means that you can store any XML document in the collection, regardless of schema. In doing this, you are still able to construct queries across all documents in the collection. NXDs that support this functionality are termed *schema-independent*.

Having schema-independent document collections gives the database a lot of flexibility and makes application development easier. Unfortunately, it is also a feature that makes database administrators pull their hair out, due to the risk of low data integrity. The old adage of using the right tool for the job definitely applies here. If you need strong schema structure, then either make sure you use a NXD that supports schemas or find another way to store your XML data.

Some products support validation with DTDs, and a few can constrain entire document collections via a proprietary schema language. In the future it is likely that W3C XML Schema (http://www.w3.org/XML/Schema) will emerge as the schema language of choice for NXDs. However, current support is limited.

Queries

XPath is the current NXD query language of choice. In order to function as a database query language, XPath is extended slightly to allow queries across collections of documents. Unfortunately, XPath wasn't really designed as a database query language and comes up short in several ways when it's used as one.

Some of the more glaring XPath limitations include a lack of grouping, sorting, cross document joins, and support for data types. Because of these issues XPath needs to be expanded as part of a more comprehensive language. Many of the issues can be resolved by utilizing XSLT to fill in the holes, but a more database-oriented language is under development, in the form of XQuery (http://www.w3.org/TR/xquery/). Several vendors have already begun to release prototype XQuery implementations for use with their databases.

To improve the performance of queries, NXDs support the creation of indices on the data stored in collections. These indices can be used to improve the speed of query execution dramatically. The details of what can be indexed and how the indices are created will vary widely between products, but most support the feature in some form.

Updates

Updates are a real area of weakness for current NXDs. Most products require you to retrieve a document, change it using your favorite XML API, and then return it to the database. A few products have proprietary update languages that allow you to perform updates within the server, and a couple of Open Source NXDs support XML:DB XUpdate (http://www.xmldb.org/xupdate/) for the same purpose. This is likely to be a problem area until XQuery adds an update language. Until then, DOM manipulation will likely be the most common update method used with NXD products.

Application Areas

There is only one hard requirement for any application that wants to use an NXD: the application must use XML. Beyond that there are no strict rules for what type of applications should or should not be built with an NXD, though some loose guidelines can be offered. In general, NXDs excel at storing document-oriented data (e.g. XHTML or DocBook), data that has a very complex structure with deep nesting, and data that is semi-structured in nature. Basically, if the data is represented as XML and is "kind of fuzzy" (a nice technical term!) an NXD will probably be a good solution. An NXD can store any type of XML data, but probably isn't the best tool to use for something like an accounting system where the data is very well-defined and rigid.

Related Information

• The XML:DB Initiative (http://www.xmldb.org/)

• Ronald Bourret's XML Database Products List (http://www.rpbourret.com/xml/XMLDatabaseProds.htm)

• Ronald Bourret's XML and Databases Paper (http://www.rpbourret.com/xml/XMLAndDatabases.htm)

• W3C XPath (http://www.w3.org/TR/xpath)

• W3C XQuery (http://www.w3.org/TR/xquery/)

Some potential application areas include

- Corporate information portals
- Catalog data
- Manufacturing parts databases
- Medical information storage
- Document management systems
- B2B transaction logs
- Personalization databases

Beyond this, NXDs are simply a new tool and their ultimate utility will be determined by the creativity of the developers using them.

Wrapping Up

NXDs aren't a panacea and they're definitely not intended to replace existing database systems. They're simply another tool for the XML developers' tool chest, and when applied in the right circumstances they can yield significant benefits. If you have lots of XML data to store, then an NXD is worth a look, and might just prove to be the right tool for the job.
