XML and Databases

Last updated January, 2003

Table of Contents

1.0 Introduction

This paper gives a high-level overview of how to use XML with databases. It describes how the differences between data-centric and document-centric documents affect their usage with databases, how XML is commonly used with relational databases, and what native XML databases are and when to use them.

2.0 Is XML a Database?

Before we start talking about XML and databases, we need to answer a question that occurs to many people: "Is XML a database?"

An XML document is a database only in the strictest sense of the term. That is, it is a collection of data. In many ways, this makes it no different from any other file -- after all, all files contain data of some sort. As a "database" format, XML has some advantages. For example, it is self-describing (the markup describes the structure and type names of the data, although not the semantics), it is portable (Unicode), and it can describe data in tree or graph structures. It also has some disadvantages. For example, it is verbose and access to the data is slow due to parsing and text conversion.

A more useful question to ask is whether XML and its surrounding technologies constitute a "database" in the looser sense of the term -- that is, a database management system (DBMS). The answer to this question is, "Sort of." On the plus side, XML provides many of the things found in databases: storage (XML documents), schemas (DTDs, XML schema languages), query languages (XQuery, XPath, XQL, XML-QL, QUILT, etc.), programming interfaces (SAX, DOM, JDOM), and so on. On the minus side, it lacks many of the things found in real databases: efficient storage, indexes, security, transactions and data integrity, multi-user access, triggers, queries across multiple documents, and so on.

Thus, while it may be possible to use an XML document or documents as a database in environments with small amounts of data, few users, and modest performance requirements, this will fail in most production environments, which have many users, strict data integrity requirements, and the need for good performance.

A good example of the type of "database" for which an XML document is suitable is an .ini file -- that is, a file that contains application configuration information. It is much easier to invent a small XML language and write a SAX application for interpreting that language than it is to write a parser for comma-delimited files. In addition, XML allows you to have nested entries, something that is harder to do in comma-delimited files. However, this is hardly a database, since it is read and written linearly, and then only when the application is started and ended.

Examples of more sophisticated data sets for which an XML document might be suitable as a database are personal contact lists (names, phone numbers, addresses, etc.), browser bookmarks, and descriptions of the MP3s you've stolen with the help of Napster. However, given the low price and ease of use of databases like dBASE and Access, there seems little reason to use an XML document as a database even in these cases. The only real advantage of XML is that the data is portable, and this is less of an advantage than it seems due to the widespread availability of tools for serializing databases as XML.

3.0 Why Use a Database?

The first question you need to ask yourself when you start thinking about XML and databases is why you want to use a database in the first place. Do you have legacy data you want to expose? Are you looking for a place to store your Web pages? Is the database used by an e-commerce application in which XML is used as a data transport? The answers to these questions will strongly influence your choice of database and middleware (if any), as well as how you use that database.

For example, suppose you have an e-commerce application that uses XML as a data transport. It is a good bet that your data has a highly regular structure and is used by non-XML applications. Furthermore, things like entities and the encodings used by XML documents probably aren't important to you -- after all, you are interested in the data, not how it is stored in an XML document. In this case, you'll probably need a relational database and software to transfer the data between XML documents and the database. If your applications are object-oriented, you might even want a system that can store those objects in the database or serialize them as XML.

On the other hand, suppose you have a Web site built from a number of prose-oriented XML documents. Not only do you want to manage the site, you would like to provide a way for users to search its contents. Your documents are likely to have a less regular structure and things such as entity usage are probably important to you because they are a fundamental part of how your documents are structured. In this case, you might want a product like a native XML database or a content management system. This will allow you