

No Loss in Translation: Using XML Databases to Simplify and Streamline Processes



For over a decade, the simplicity, portability, and flexibility of XML have made it the accepted standard for formatting and sharing data via web services and service-oriented architecture (SOA). However, XML data that is easily transferred across machines and applications is not as easily stored and processed within those same machines and applications. As a result, the XML data is typically transformed into non-XML formats better suited for use within databases and applications. This transformation step adds a layer of complexity to the process.

What is XML? XML is a flexible text format used to create structured documents. Both the format and the data can be shared across the Web and elsewhere. An example of XML is seen in Figure 2. **What is an XML Database?** An XML database is a data storage format that allows data to be maintained in an XML format. The data can then be queried and processed as XML without transforming it into a relational schema. **What is XQuery and SQL?** XQuery is a language to extract and manipulate data from XML documents or databases. Its syntax resembles SQL, which is the language to extract and manipulate data from relational tables. XQuery provides a bridge between the XML and database worlds.

But what if the transformations were not necessary? What if XML data could be stored and delivered in a way that functions similar to relational databases? And what if this XML capability were robust enough to handle real-world demands such as the multitude of tax filings processed by the New York State Department of Taxation & Finance (NYS DTF)? The result would likely be quicker delivery of product, fewer errors, a streamlined development and testing cycle, and reduced maintenance demands to keep disparate formats synchronized. And that's exactly what NYS DTF experienced by implementing an XML database as part of improvements to its overall tax processing.

But what are XML databases? Simply put, XML databases store data in an XML format. *Why are they important now?* Two recent developments in particular are significant. One is the gradual maturity of XQuery and the extensions for it that bring SQL-like querying capabilities to XML. One of the main objections to XML databases up to this point has been the lack of robust query and index functions to rival what's available in relational databases. But that gulf is narrowing. The second, more significant development is the maturing of XML databases such as Mark Logic Server and eXist and the incorporation of XMLdatabase features into the major relational database products such as Oracle, SQLServer, and IBM DB2.

A closer look at the differences between maintaining the data's original XML format as opposed to transforming XML data into relational data will demonstrate the benefits inherent in XML databases.

EFFECTIVE, BUT NOT SIMPLE

What characterizes most Web applications is a multi-tier approach that separates the process into different components. Typically, the approach has three layers:

- Web page (presentation layer) on which a user enters data,
- code (processing layer) often written in Java, PHP, C# or similar languages, and
- relational database (data layer) that stores the data and makes it available for queries and further processing.

This structure works well because the separation of layers allows upgrades or changes to one layer to occur independently of the other layers. For example, the processing code can be modified or completely migrated from one language to another without impacting the Web page form that collects the data or the relational database that stores it. Development and design can proceed in a modular fashion with each layer dedicated to what it does best.

However, complications arise in transporting the data between these multiple layers. Data on a Web form is structured much differently than data stored in a database. Furthermore, a relational database (which is most often used in business applications) possesses a unique structure based on the relations among the various pieces of data in the dataset. The data itself is maintained within rows and columns of tables that are joined (related) by key fields. This architecture is very efficient for storage and retrieval within the database, but the data itself must undergo significant and complicated processing to ensure efficiency across the multiple layers.

Even a simple example, such as the class registration form shown below, demonstrates the manipulations required to move data through a process. In this example, the data from the ten fields in a form are extracted and stored across four distinct, but related tables in the database (a process known as “shredding”). When it’s necessary to retrieve the original form, the data must be extracted from the tables in the database and reassembled for the form.

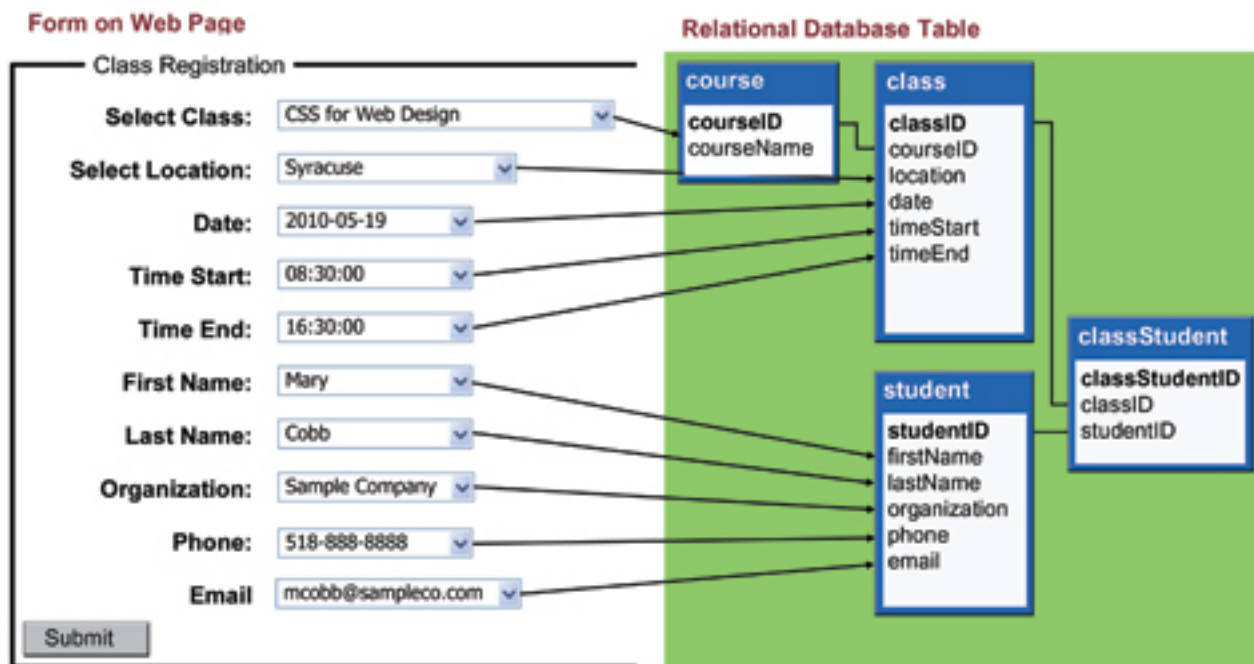


Figure 1. Shredding XML Data into Relational Tables.

Figure 1. Shredding XML Data into Relational Tables.

For complex environments, such as the Department of Taxation & Finance, these transformations can be monumental. There are over 3,600 forms at NYS DTF. With a relational database, these forms translate to thousands of tables. Although the forms are essential to processing the data, what really matters to the agency is the tax filing, which is the combination of forms submitted by individual taxpayers. In a “data shredding” environment, the filing context is dispersed among numerous tables and must be maintained and reconstructed as needed. The data itself can be stored, queried, and processed but it is far from a simple activity.

SIMPLE AND EFFECTIVE

While the multi-tier architecture can get complicated in matching the business model (e.g., tax filing) with the system design (e.g., relational table), its components are based on mature, proven, and widespread technologies that make it an efficient, reliable approach. But what if the data shredding and reassembling steps were not necessary? What if the data contained its own structure that could be maintained and carried through the entire process, without abandoning storage and query capabilities? What if the business model closely mirrored the system design? Well, that's exactly what XML databases make possible. The example of the class registration demonstrates how this is achievable.

The form that a student submits from a Web page to register for a class can have a built-in XML format. In an XML database, that XML form can be processed and stored as XML exactly as it appears on the Web page—without “shredding” the data into multiple tables. The three layers of the multi-tier architecture described above for the relational database are flattened into one uniform process. Indexes can be set and queries performed (using XQuery) within the XML database just as they are done for relational databases (using SQL). This approach couples the advantages of XML and its simple, open, portable data format with the advantages of databases and their storage, querying, and processing capabilities. The XML format carries throughout the entire process—from structuring the Web page form to being stored as is in the XML database. The example in Figure 2 shows the nearly one-to-one correspondence between the data in the form and the data in database. Of course, a database alone is not the solution. At the Department of Taxation & Finance, for example, many other factors such as improved workflow engines and integrated applications also came into play. But the presence of a ubiquitous data structure supported by the XML database facilitated an effective and simpler approach.

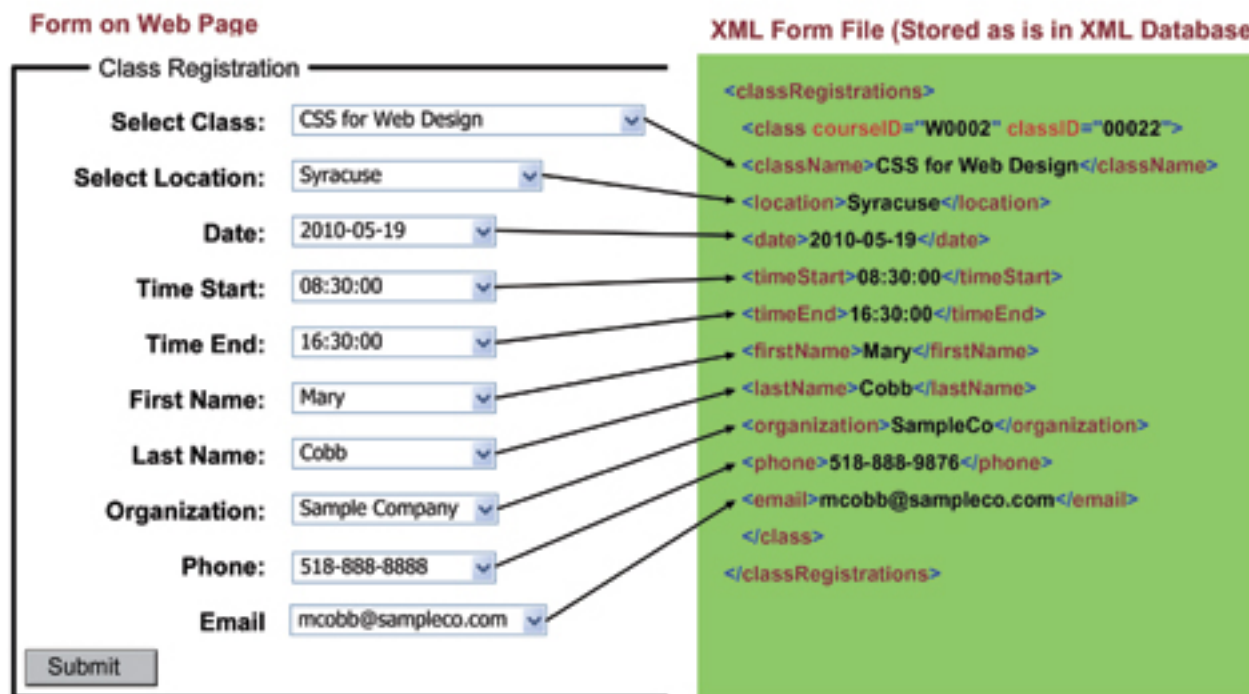


Figure 2. Aligning XML Data with XML Database.

Figure 2. Aligning XML Data with XML Database.

BUT IS IT PRACTICAL?

The example of the Department of Taxation & Finance illustrates that XML database usage on a meaningful scale within a government setting already exists. In 2009, the NYS DTF processed 11 million personal income tax returns along with thousands of corporate, sales, and withholding tax filings. As noted by James Lieb, director of Architecture and Web Solutions at NYS DTF, they have also delivered over 20 new Web applications in the last year and implemented two entirely new systems with backend XML databases. Lieb said, “Each has gone up quicker and with fewer problems than if we had tried to do them relationally.”

Likewise, the Center for Technology in Government has migrated all of its Web site material (including nearly 500 reports, news releases, and project descriptions) to an XML database to open new query and processing

capabilities. Duplication of content has been eliminated, leading to better consistency of information; text searching has been enhanced, leading to more robust information retrieval; and new capabilities are possible for creating user-defined subsets of information.

At the federal level, the Office of the Historian at the Department of State is using XML databases to prepare and publish its *Foreign Relations of the United States* (FRUS) series, the official historical documentary record of U.S. foreign policy. It also makes XML datasets available at DATA.gov.

Other examples throughout the public and private sectors exist as well. And, although the use of XML databases is only at the beginning now, the trend should continue to grow as awareness spreads and capabilities increase. The benefits of minimizing the complexity of translating data, as found by NYS DTF, will begin to cascade into a myriad of new applications that can allow government to focus on better serving its citizens.

Jim Costello, Web Application Developer, Center for Technology in Government