Beyond the Relational Database Model

David M. Kroenke, University of Washington

While revising the 10th edition of my book, *Database Processing: Fundamentals, Design, and Implementation* (Prentice Hall, 2005), I proposed in one chapter that XML data storage will eventually eliminate relational databases. The horrified reviewers strongly recommended softer language. "XML for data transport—OK," they said. "But eliminating relational databases? No way."

But why not? Why do we need relational databases when XML document stores will do the job better? Why take structures the user provides and break them into pieces? Although we can readily transform XML documents into normalized relational tables, why not simply store XML documents?

The more I thought about it, the more it became obvious to me that the relational database model is *doomed*.

XML data stores could eliminate the need for relational databases.

PROBLEMS WITH NORMALIZATION

In May 1985, while serving as Microrim's vice president of development, I spoke with a customer who had just installed the new R:Base 5000 and was having trouble with the documentation. "I want to use it to keep track of my invoices," she explained. "I looked under *I* in the index, and invoice wasn't there. I thought maybe you called them sales orders, so I looked under *S*, but I found no such entry."

During the next hour or so, I walked her through the process of creating five normalized tables—CUSTOMER, SALESPERSON, INVOICE, LINE-ITEM, and ITEM—and hooking them together with foreign keys. Then I explained how to rejoin them to get her sales orders back. Long into the call, she asked, "Why am I doing this?"

Why indeed! Ever been to a normalization garage? The one where you pull your car into the lot, and a crew runs out and removes your left front tire and places it in the pile of left front tires; removes your right front tire and places it in the pile of right front tires; and removes the steering wheel and places it in the pile of steering wheels. Then, when you retrieve your car, the crew reassembles it from the separate piles.

Why would they do this? For the convenience of managing the garage—to have fixed-length piles; to index the piles; to ensure that you don't have more than one steering wheel.

After 35 years of computer science and numerous iterations of Moore's 18-month cycles, isn't there a better way?

RELATIONAL MODEL RATIONALE

The relational model is a set-theoretic model for describing data constructs common in the business environment. Having any sort of theory in the 1970s was important for legitimacy. I recall a then-common joke: How do you know the real sciences? They don't have science in their name. Consider chemistry, physics, and biology versus administrative science, social science, and yes, computer science.

Relational databases also minimize data duplication, which ensures data integrity and reduces storage requirements. The latter factor was particularly important in the 1970s, when disk capacity was quite limited—the IBM 2311 stored only 7.5 Mbytes.

Further, the relational model provides a way to represent variable-length constructs with fixed-length components. In the days of magnetic tape, storing variable-length records like sales orders was easy—every time you changed a record, you had to rewrite the entire file anyway. However, it's impossible to do online airline reservations that way. New direct-access data records had to be fixed length.

In addition, normalization theory was the basis of hundreds of papers and successful tenure applications. This ensured the academic community would carry the model forward.

Finally, by following open standards, including the Structured Query Language, vendors created a buzz with dozens of relational DBMS products such as System R, DB2, Oracle, SQL Server, Ingres, dBase, R:Base, Pearl, Paradox, and Access.

XML DATA STORES

Suppose that instead of normalizing or breaking up a collection of XML documents, we could maintain it as an *XML data store*. An XDS would have

many advantages over a relational database, including seamless integration with user views as well as all the benefits of XML standards such as XML Schema validation and the Extensible Stylesheet Language for document materialization.

Would there be any disadvantages? No doubt about it, an XDS would duplicate data. But so what? Storage is free.

What about data integrity? For that, we need only find the duplicated data and make it consistent when necessary. Conceptually, this is merely an extension to cascading updates on foreign keys. And in fact, data duplication has many benefits. In a relational database, if you don't duplicate ITEM.ItemPrice in LINE-ITEM, you'll never be able to recover the original invoice; ITEM.ItemPrice will change, but LINE-ITEM.ItemPrice should not.

In addition to data duplication, there are other problems to overcome such as uniqueness. For example, how many documents will carry QuantityOnHand? Which document is the official one? Which document reflects what's available in inventory right now?

Another hurdle is the XDS concept itself. Would there be herds of XML documents wandering around the enterprise wherever they want? Would it be necessary to put a boundary around some XML documents—for example, create a customer relationship management XDS? If so, how do we do that?

Yet another objection is that XML documents are hierarchical. But is this really a disadvantage? My belief, based on nearly 40 years of database design, is that the human instruction set predisposes us to think in hierarchies. We have a Class and a list of Students in that class—a hierarchy; or, we have Students and a list of the Classes they take—a second hierarchy. Of course, there's an N:M relationship beneath CLASS and STUDENT, but I suspect that's relevant only to designing a normalization garage.

On rare occasions, we do perceive network structures; they appear as tables with rows and columns as entities. For example, a table can have a MOVIE column and THEATER rows, with the cells containing show times or prices. But how often do you see that structure? Look in a newspaper and you'll find the hierarchy THEATERs with MOVIEs or MOVIEs with THEATERs. (Yes, there are crosstabs, but they're a view, not a structure.)

An XDS would offer seamless integration with user views as well as all the benefits of XML standards.

In any case, it's possible to represent networks in XML documents; it's clumsy and brings DL/I databases to mind, but it can be done.

The last objection to using XML data stores is that it would hinder performance. Ironically, critics leveled the same objection against the relational database model four decades ago.

FROM HERE TO THERE

Several components are necessary to make XDS a reality—some exist, while others require research.

Tools and techniques are needed to efficiently manage data duplication, including when and how to enforce consistency. For example, what documents, if any, should be altered after an employee's name changes? Modifying original documents would arguably falsify them; on the other hand, querying documents without the change would fail to locate some employee data.

With so much data replication, XDS offers more design choices, but not all the principles are clear. For example, should a student transcript be represented as a sequence of separate documents or as a single document with data appended at the end of each term? The former approach preserves name-change history, but with all the associated problems.

We must know what version of a document we have and how it relates to other versions. Such versioning would be a boon to data mining and reporting applications that regularly—and unknowingly—mix apples and oranges today. We also need a standard, accepted, and efficient document query language.

In addition, we must think through the concept of a bounded XDS. The idea of free-floating groups of documents strikes me as chaotic, but perhaps that perception comes from years of thinking "centralized database." After all, the Web demonstrates the benefits of independent, autonomous processes standardized in minimal ways. Maybe we should aim in that direction before forcing XDS boundaries.

Also needed is a mechanism to resolve XML Schema ambiguities. When two documents meet in cyberspace, they must be able to determine their degree of similarity. Are they instances of the same semantic, or close enough? Do they have sufficient semantic similarity to breed and form a hybrid, or are they so different that no sensible combination will result?

Object-oriented database management systems taught us that no organization will adopt a technology, such as XML data stores, that requires converting billions of bytes of relational data to another format before realizing tangible benefits. One such migration tool is ADO.NET, which easily converts the contents of a relational database into XML documents. I just wish I had a place to put them. ■

David Kroenke *is a lecturer in information systems in the Management Science Department at the University of Washington Business School. Contact him at dk5@u.washington.edu.*

Editor: Richard G. Mathieu, Dept. of Decision Sciences and MIS, St. Louis Univ., St. Louis, MO; mathieur@slu.edu