PASCAL
Beyond Reason

Consider a mathematical principle, say [the identity]:
(a+b) x (a-b) = (a2) - (b2).
Now, let us assume you are using a mathematical calculator made by Texas Instruments for numerical computations [using a method] involving the above principle. If the calculator is slow in calculating the above results in comparison with some other method, which would you blame, the calculator (implementation/vendor) or the [method] (logically sound and correct)?
(Anith Sen, Online Exchange)

Several visitors to DATABASE DEBUNKINGS (www.dbdebunk.com) alerted me to *Beyond Relational Databases* by Margo Seltzer (Databases, Vol. 3, No. 3, April 2005). Such things are a dime a dozen, and my instinctive reaction was to ignore it, when I noticed the byline:
MARGO I. SELTZER, Ph.D., is Herchel Smith professor of computer science and associate dean in the Division of Engineering and Applied Sciences at Harvard University. Her research interests include file systems, databases, and transaction processing systems. Seltzer is also a founder and CTO of Sleepycat Software, the makers of Berkeley DB. She is a Sloan Foundation Fellow in computer science, a Bunting Fellow, and was the recipient of the 1996 Radcliffe Junior Faculty Fellowship and the University of California Microelectronics Scholarship. She won the Phi Beta Kappa teaching award in 1996 and the Abrahmson Teaching Award in 1999. She received an A.B. degree in applied mathematics from Harvard/Radcliffe College in 1983 and a Ph.D. in computer science from the University of California, Berkeley, in 1992.

Now, there used to be a time where such illustrious credentials would guarantee pronouncements of substance. But after a long period of intellectual deterioration in the academia, driven by commercial contamination – beware of academics turned vendors – this has no longer been the case for quite a while, as often documented in my writings. See, for example, *Normalization for Performance: Et Tu Academia?* (http://www.dbdebunk.com/page/page/622733.htm) and *The Chasing of Dayflies* (http://www.dbdebunk.com/page/page/622733.htm). But I keep coming across nonsense from today's academics, and each time I tell myself it cannot possibly get any worse, they surprise me.

The article begins by stating "There is more to data access than SQL", and we could not agree more. It then refers to the proliferation of computing devices such as "PDAs, highly mobile tablet and laptop devices, palmtop computers, and mobile telephony handsets [that] now offer powerful platforms for the delivery of new applications and services [and the] many computing and network elements required to support" such as "text and multimedia messaging, location-based search and information services (for example, on-demand reviews of nearby restaurants), and ad hoc multiplayer games", and many future ones "impossible to predict today". From this Seltzer derives the following conclusion:
While these services differ from one another in major ways, they also share some important attributes. One – the focus of this article – is the need for data storage and retrieval functions built into the application.

This, in a section called "History of Relational Databases"!!! Anybody who knew history – let alone a PhD in computer science with specialization in databases – could not possibly draw this conclusion, but rather the exact opposite: if we learned anything from history, is that the last thing we want is data management *by applications*.
(Note: If Seltzer means embedded DBMSs/databases, that's something altogether different, and someone of her caliber should be able to formulate it correctly.)

Messaging applications need to move messages around the network reliably and without loss. Location-based services need to map physical location to logical location (for example, GPS or cell-tower coordinates to postal code) and then look up location-based information. Gaming applications must record and share the current state of the game on distributed devices and manage content retrieval and delivery to each of the devices in real-time. In all these cases, fast, reliable data storage and retrieval are critical.
Note the common confusion between levels of representation – *physical* storage and *logical* retrieval. As logic and mathematics applied to data management, the relational model has nothing to do with storage, hardware and network infrastructure, which, to reiterate, means that by supporting *physical data independence*, the model can be used on any platform where there is a *need for inferencing*.
Here's Codd in 1979:
The relational model for formatted databases was conceived ten years ago, primarily as a tool to free users from the frustrations of having to deal with the clutter of storage representation details. This implementation independence coupled with the power of the algebraic operators of n-ary relations and the open questions concerning dependencies (functional, multivalued, and join) within and between relations have stimulated research in database management.
(E.F. Codd, *Extending the Database Relational Model to Capture More Meaning*, ACM publication)

What is more, it can be implemented efficiently because it gives the implementor *complete freedom to do whatever is necessary at the physical level to maximize performance*, as long as it is not exposed to users in applications. Platforms can change, but does this mean that logic and mathematics are no longer applicable? Inferencing is inferencing, whether on a server, or on a PDA or mobile phone.

As soon as the discussion turns to data storage and retrieval, relational databases come to mind. Relational databases have been tremendously successful over the past three decades, and SQL has become the lingua franca for data access. While data management has become almost synonymous with RDBMS, however, there are an increasing number of applications for which lighter-weight alternatives are more appropriate.

While the average practitioner confuses SQL with relational, we expect somebody like Seltzer to make the proper distinction between the two. Seltzer admits it's the application-based data management that in the past made cross-platform portability prohibitive, and that relational technology, even in its bastardized SQL form, addressed this problem.

Relational databases were fundamentally a reaction to the escalating costs required for deploying and maintaining complex systems. The key observation was that programmers, who were very expensive, had to rewrite large amounts of application software manually whenever the content or physical organization of a database changed. Because the application generally knew in detail how its data was stored, including its on-disk layout, reorganizing databases or adding new information to existing databases forced wholesale changes to the code accessing those databases.

But then she fails to draw the proper conclusion:

In the 20 years that followed, two related trends emerged. First, the RDBMS vendors increased functionality to provide market differentiators and to address each new market niche as it arose. Second, few applications need all the features available in today's RDBMSs, so as the feature set size increased, each application used a decreasing fraction of that feature set.

This drive toward increasing DBMS functionality has been accompanied by increasing complexity, and most deployments now require a specialist, trained in database administration, to keep the systems and applications running. Since these systems are developed and sold as monolithic entities, even though applications may require only a small subset of the system's functionality, each installation pays the price of the total overall complexity. Surely, there must be a better way.

What Seltzer does *not* say – and, frankly, we're not sure she realizes it – is that (a) the DBMS was invented to address the problems of the very "better way" she is proposing (b) SQL replaced database technologies that were *much, much* worse (c) instead of building truly relational DBMSs (TRDBMS), vendors first botched things with SQL, then further messed products up with unnecessary complexities and violations of the model – even reintroducing pointers – due either to ignorance, or in order to patch up problems created by the botching. In fact, C.J. Date is on public record predicting years ago that by the time vendors are done with them, SQL products would be more complex than the ones they replaced (see, for example, *A Database Disconnect*, http://www.dbdebunk.com/page/page/622123.htm).

To the extent that an implementation is complex, or performs poorly, it has only itself, not the relational model to blame. And that is certainly true of SQL and its implementations, the complexities and inefficiencies of which are not due to its being relational, but rather to its *not being so*.

If so, what does the following have to do with the relational model?

We are not the first to notice these tides of change. In 1998, the leading database researchers concluded that database management systems were becoming too complex and that automated configuration and management were becoming essential. Two years later, Surajit Chaudhuri and Gerhard Weikum proposed radically rethinking database management system architecture. They suggested that database management systems be made more modular and that we broaden our thoughts about data management to include rather simple, component-based building blocks.

By the way, the "architecture" term thrown indiscriminately around could do with some analysis too.

Seltzer then turns to another academic turned vendor:

Most recently, Michael Stonebraker joined the chorus, arguing that "one size no longer fits all" and citing particular application examples where the conventional RDBMS architecture is inappropriate. As argued by Stonebraker, the relational vendors have been providing the illusion that an RDBMS is the answer to any data management need. For example, as data warehousing and decision support have emerged as important application domains, the vendors have adapted products to address the specialized needs that arise in these new domains. They do this by hiding fairly different data management implementations behind the familiar SQL front end. This model breaks down, however, as one begins to examine emerging data needs in more depth.

I will debunk Stonebraker's pronouncements in a future column. Here I will simply reiterate that *there are no relational, but only SQL vendors*; that *logical inferencing* is universal in data management, and that it should not ever be (a) confused with any "architecture" (whatever that means), or product (b) substituted for anything that is not at least as sound, general, and simple as that. Academics worthy of their PhDs should know that.

Fabian Pascal

Fabian Pascal is onafhankelijk IT-analist, consultant en auteur gespecialiseerd in data management. Zie ook zijn website www.dbdebunk.com