Michael Stonebraker Speaks Out

on Why He Wouldn't Have Done INGRES Knowing What He Now Knows, the Successes and Failures of Database Theory, the Value of Startups, What to Do About the Excess of Middleware, and More

by Marianne Winslett

Michael Stonebraker

Two members of the SIGMOD community have recently been elected to the National Academy of Engineering: Phil Bernstein and Hector Garcia-Molina. They join David DeWitt and Jim Gray as recipients of one of the greatest honors in the engineering world. On behalf of the SIGMOD community, I congratulate Phil and Hector! Interested readers can find interviews with all four of our NAE members in recent installments of this column. Congratulations also go to Avi Silberschatz, whose interview also recently appeared here, for receiving the IEEE's Taylor Booth Education Award this year.

Readers may have noticed that last issue's interview with Jim Gray lacked the extensive editing marks that have characterized previous columns. During the editing that accompanies the translation from the spoken word to a written document, we introduced the usual editing marks, and then removed them at the end. Their removal results in a much more readable document, at the price of hiding the divergence between the spoken and written word. The first Web video version of an interview is nearing completion, and soon readers will be able to download the videos and watch the original (slightly edited) interview. With the original material available, a divergence in content between the two media becomes less of a concern. In light of this, we will continue to remove editing marks.

Welcome to this installment of ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and we're in Madison, Wisconsin, site of the 2002 PODS and SIGMOD conferences. I have here with me Michael Stonebraker, who is soon to be an adjunct professor at MIT. Before that he spent almost thirty years as a professor

at Berkeley. His work there on INGRES was instrumental in showing that the relational model for data was not just a pipe dream and could be used in real database applications. Mike went on to found INGRES Corporation, and he was also a founder of Illustra Information Technologies, which commercialized his work on object-relational systems. Mike's PhD is from the University of Michigan. So, Mike, welcome.

Thanks.

Mike, as an assistant professor you set out to build a relational database management system at a time when none existed, a time when the relational model was viewed as a wild idea by most of the people in the database field. How long did you think it would take you when you first started to work on INGRES?

[When I started to build INGRES,] the traditionalists said, "You can't possibly build one of these new-fangled relational systems and, even if you can, nobody can understand these [new relational] query languages people are talking about."

I have a PhD in applied Markov theory—applied operations research—and I realized quickly that continuing with my PhD topic was no way to get tenure. So I had to do something new, and basically for no good reason, out of thin air, I decided to do databases. Back then, in 1971, Codd had just written his pioneering paper and it resulted in an immediate argument between the traditionalists and the relationalists. The traditionalists said, "You can't possibly build one of these new-fangled relational systems and, even if you can, nobody can understand these query languages people are talking about." And the relational guys all said, "Nothing as complicated as traditional databases can possibly be the right thing to do." So it was an obvious research task — several people thought of it — to try and build a relational system.

We built an initial prototype, putting in the first 90% of the effort required to create a real system, and it more or less worked. I think that the thing that distinguished INGRES from the typical academic project, and in retrospect one of the smartest things we ever did, was to then put in the next 90% of the effort to make INGRES really work. We spent five years with the standard contingent of one full time programmer and four or five smart graduate students and undergraduates. At the time I began, I had no idea what database systems were, so I had no idea how long it was going to take to build INGRES. It just seemed like a good idea at the time, and you've got to get tenure, and that was where I placed my tenure bet.

What do you view as the biggest successes and failures of the INGRES project?

I think the biggest success was that we made INGRES work, and that was pretty novel at the time. And, in retrospect, we made a bunch of very lucky accidental decisions. We picked UNIX, we picked C, when those were very wild ideas. And I guess the only reason we made those choices was that Ken Thompson is a Berkeley graduate, and came out and visited us, and I had confidence that he knew what he was doing. I think another factor in our success was that we, for some reason, stumbled onto the dictum, "If you get it wrong, just throw it away and rewrite it." So we rewrote big pieces of INGRES multiple times in the process of really getting it to work, and we didn't get frustrated. In sum, I think that the biggest success was that we got it to work.

I think the biggest failure of INGRES was acceptance by real world users. Because we had a reasonably workable system, people in the late 70s had moved from saying, "You can't get it to work and people can't understand the query languages," to, "It won't scale! Who is your biggest user? This stuff isn't ready for prime time!" I think we came within a little bit of convincing Arizona State University to put their student records database, all forty thousand students worth, on an INGRES database in 1978. And Arizona State could get past the fact that they had to get an unsupported operating system from AT&T, then in North Carolina; they could get past the fact that they had to get an unsupported database system from these goofy guys at Berkeley; but the project went down the drain when they realized that there was no COBOL available for UNIX. They were a COBOL shop, and so the feasible, possible penetration of INGRES into real-world applications was pretty much scuttled by the facts that UNIX was unsupported, the database system was unsupported, and that there was no COBOL. And so INGRES, as an academic prototype, couldn't overcome those obstacles. That was only possible by commercialization.

Knowing what I know now, I would never have started building INGRES, because it's too hard. And it turned out to be vastly more work then I could have, in my wildest imagination, contemplated.

If you could whisper a few words of advice to the young Mike Stonebraker who's just starting that project, what would you say?

Knowing what I know now, I would never have started building INGRES, because it's too hard. And it turned out to be vastly more work then I could have, in my wildest imagination, contemplated. If I'd been smart, I would have never started because as I said, this was—is—too hard. So I think my advice to my younger self would be to suspend your disbelief and just do it anyway. The way you climb Mt. Everest is one step at a time, and you just do it. So don't be dissuaded because you think things are hard.

The other piece of advice for my younger self relates to the fact that INGRES was largely written by Berkeley undergraduates. We would give them the best equipment we could afford and put it on their desktop at a time when that was a very expensive thing to do. And then we would recruit the smartest freshmen and sophomores we could find, give them wonderful equipment, and they would basically die writing code for us. So I'm a big fan of recruiting undergraduate talent, really good undergraduates, and creating an environment in which they can get a lot done.

Did you do that later on at Berkeley in any of your subsequent projects? And at MIT?

MIT is a very different— Right now at MIT, I'm kind of a missionary. There are no database people there, and so I'm trying to figure out how to get things to work at MIT.

As a veteran of two major start-ups, do you have any words of advice for people who are considering starting a company?

I think it's a great idea. I think that if you have what you think is a better mousetrap, trying to get a start-up off the ground is a great way to figure out whether you really do have a better mousetrap or not. And also, if you do get your startup off the ground, it's a great way to meet real users and get smart application people with hard problems talking to you about your stuff. And

that's a great source of new problems and interesting ideas. In academia or in the research community, we tend to get fairly isolated, and a startup is a great way to get very grounded in real world problems.

It sounds like you would also advocate spending sabbaticals in industry, maybe in a development group.

I think one of the big failures of the academic community is that if you're trying to get tenure, your objective is to pad your resume the best you can, and that means that you need to spend sabbaticals holed up writing papers, which doesn't produce well-rounded research people. I think I would advocate spending time off in industry, in start-ups especially, and in industrial research labs if they are grounded in real world problems. I think going to other universities is the least productive thing to do on a sabbatical because that means spending your time with more people who think just like you do.

Has anything useful come out of database theory so far? What topics should database theoreticians be working on now?

I guess that later on in this interview you want to talk about the Laguna Beach report on the future of database research, which came out in 1989 or 1988, some time around then. This was the time when doing recursive queries was a big deal in the theory community. So working on recursion was something everyone was doing, and at Laguna Beach there were ten or twelve practitioners, none of whom had ever seen a user who was interested in seeing recursive queries solved.

> One of the smartest things we ever did was to then put in the next 90% of the effort to make INGRES really work.

Not even transitive closure queries?

A couple of users had simple linear recursive queries. No real users had nonlinear recursion in their queries, which is what the database theory people were all working on. So we challenged the theoreticians: find somebody who wants your technology before you go off and solve a problem that nobody's interested in.

I think the big problem with the theory community is that they're very isolated. They go off and do their own thing, whether or not it's useful. I think recursion was one of their big failures. I still can't find anybody interested in nonlinear recursion. I think their big problem is that they are more interested in working on problems that are solvable, rather than problems that are important. And I think my advice to theoreticians is the same as what we just talked about: go spend some time in the real world and work on problems that people want solved.

Do you think that when theoreticians do spend time in the real world, they'll still be doing theory or will they discover there aren't theoretical problems out there? And will they come back as systems types, or will they find theory problems wherever they go?

I think one of the bigger successes of the theory community was concurrency control and serializability theory. I think they definitely made important contributions in that area. I think what happens to people like Avi Silberschatz who goes to Bell Labs and gets converted into a systems type---that certainly does happen, and when it does, that's a good thing, not a bad thing. In fact, John Hopcroft, I hear, at Cornell---who is not in the database area---is turning into a

systems type. So if that's the way to make the best contribution, so be it. I think the only problem I have with the database theory community is that they tend to go off on deep tangents, deep into outer space, and don't seem to manage to yank themselves back easily.

Were object-oriented databases a deep tangent into outer space from which the database systems *people took a while to yank themselves back?*

My personal perspective is that that topic was hotly debated a while ago, and having talked to the people starting up one of the object oriented database companies, my first reaction was: "Where's the market for this stuff?" It's elegant technology for which there is essentially no market. So it's a deep tangent in the sense that it was interesting stuff that nobody wanted; and the fact that nobody wanted it was, I thought, fairly obvious up front. The problem is that building a persistency mechanism for C++ is not something that there's a huge market for. It's a niche market. And I think the systems guys probably spent five years doing that. If you ask, "Well, should they not have done that?", the answer is that I don't know. I think that certainly interesting stuff came out of it, and ultimately the technology is probably not going to go anywhere.

OODBs are a deep tangent in the sense that it was interesting stuff that nobody wanted; and the fact that nobody wanted it was, I thought, fairly obvious up front.

There are still some object-oriented database companies in business. So, like you said, a niche market is out there.

I think that a much more interesting question is, "Why hasn't object-relational technology taken off?"

That is my next question! People are saying that object-relational technology hasn't had the impact that vendors expected. What does that mean? Is there a piece of research that we missed, a killer app that we haven't found yet, or what's the story there?

My feeling is that it's an obviously good idea, and I think it *has* taken off, it's been wildly successful in---let me back up and say that there's a great book called *Crossing the Chasm,* by Jeffrey Moore, that says that any new technology gets used first by the early adopters. If they like it, they tell their friends and more conservative people try it, and then it gets out into the broad market and then eventfully the late adopters pick it up. This process takes a while, and the most difficult transition is crossing the chasm from the early adopters to the mainstream.

I'd characterize object-relational technology as having been wildly successful among the early adopters. In big GIS systems, it's been very successful. In media asset management, it's been very successful. Most of the big content owners in the world, meaning CNN, BBC, Televisa, are wildly enthusiastic about the technology. So it's definitely been used by the early adopters and very successfully. It hasn't seemed to cross the chasm yet. And I think that there are two good reasons for that: number one, an absence of standards, which means that the extension capabilities in the various vendors' products are not the same, in an era where everybody wants their applications to run on everybody's platform. And I think that with J2EE, everybody except Microsoft seems to be standardizing on J2EE as the extension mechanism. So I think we're at least making progress there. I think the second impediment is Microsoft, which is not pushing the technology, and until recently Oracle was also not particularly pushing the technology. I think the fact that Informix and IBM were pushing object-relational technology hard, without the other two vendors pushing it, is just a big marketing problem. So I think those are its biggest problems.

I think that the current enthusiasm for XML is another statement of why extensible systems are a good idea. Informix was able to pretty well support XML, the various dialects of Xquery, and XPath by just adding user-defined operators. For example, "//" in XML (which is doing a hierarchical search) was implemented by Informix as a user-defined operator. And so the technology that provides extensibility is extremely useful in adapting to whatever the next thing that comes down the road is. And after XML there will be another new thing. That's guaranteed. So I think that object-relational technology essentially *has* to win, sooner or later, because XML is not the last new data type you are going to have to put in database systems.

So we just need to be patient---and perhaps do better marketing.

Well, I think either that or give Redmond, Washington, a brain transplant.

Has retirement meet your expectations? Do you have any words of advice for those of us who are approaching retirement? Should we all move to New England?

Oh, you're not that old.

I think the only problem I have with the DB theory community is they tend to go off on deep tangents, deep into outer space, and don't seem to manage to yank themselves back easily.

I'm already retired, actually. But I'm an early adopter.

First of all, "retirement" is kind of a misnomer for my current state. I guess Berkeley bribed all the people who could possibly qualify for the retirement plan to take a package about seven years ago. Their offer was irresistible if there was anything else you thought you wanted to do. And so I let them bribe me into becoming a retiree---a pensioner---so I'm a *pensioner*. I never had in mind *retiring*. So at the current time, I work half time for a database start-up in New York City and I'm on the faculty at MIT and am as busy as ever, much to my wife's dismay, who thought that leaving Berkeley meant that I would, I don't know, go sit in the garden or play golf or whatever retired people do. I can't imagine retiring. I think it would be boring.

So then the question comes, why we did move from California to New Hampshire in 1999. There are two different stories I can tell about that. One is that my wife has a huge extended family in New England and had been lobbying increasingly intensely in recent years to relocate closer to her family, and so it was time to live where she wanted to live for a while. So the move was made for non-professional reasons. Another contributing factor is the quality of life in California is decreasing -- it is getting very crowded there and I was getting tired of sitting on freeways in rush hour traffic all the time. Except professionally, New Hampshire has a wonderful quality of life. I'm a big booster of the state. It also has no taxes to speak of. As far as where people ought to live, I think New England is a beautiful place to live and if you get outside the urban areas, the quality of life is wonderful. California is a great place to live too.

Very few people have worked at more then one top-ranked department. Can you comment on the differences between Berkeley and MIT?

Oh, boy! First of all, let me give a couple of caveats. I'm a missionary at MIT. You know, they have had no one in databases since Mike Hammer left in 1981, and so I'm the lone database person there, whereas at Berkeley I was one of the in-crowd. So things are very different for me

at the two schools, so take whatever I say next with a grain of salt.

I think Berkeley is very, very collegial. Somehow, the Berkeley people constructed an environment where everybody worked together. People spent a lot of time in their offices. People talked to each other. There were a lot of joint projects and a lot of feeling among the faculty of trying to work for the common good. At MIT it feels like the faculty are more like individual gunslingers, and so there doesn't seem to be the same collegial atmosphere. One example of that is that at Berkeley, office space was handled by the department, whereas at MIT, the department handles academic appointments but if you want office space you have to get it from individual labs, such as the Laboratory for Computer Science, or the AI Lab, or the Media Lab. So that immediately fragments your loyalty.

I think there's very different emphasis on academic disciplines at the two schools as well. MIT is very strong in AI, very strong in speech. Berkeley, I think, is very strong in traditional systems. I think both schools have extremely good students and very, very smart faculty. So I guess the big difference is that the collegial natures of the places are different. So, for instance, at Berkeley everybody helps to make hiring decisions, at their annual retreat. Whereas, MIT appoints a small committee who makes decisions, which contributes to the department having a different feel.

If you magically had enough extra time to do one additional thing at work that you're not doing now, what would it be?

I think my pet peeve is one of the things I talked about this morning in my invited talk at SIGMOD 2002: there is just too much middleware. The average corporation has bought a portal system, has bought an enterprise application integration system, has bought an ETL (Extraction, Transformation, and Loading) system, has bought an application server, maybe has bought a federated data system. All of these are big pieces of system infrastructure that run in the middle tier; they have high overlap in functionality, and are complicated, and require system administrators. The average enterprise has more than one of all of these things, and so they have this spaghetti environment of middleware, big pieces of moving parts that are expensive to maintain and expensive to use.

> I think that if we're expected to be researchers, one thing we ought to do is be competent practitioners of our trade, and I'm not.

Everyone seems to recognize this problem, and the conventional commercial wisdom is to expand the role of an application server so it does what all of these packages do. Web Sphere, for example, from IBM, is becoming a very, very rich package which does a lot of middleware functionality.

I think a federated database system is a much better base on which to build a middleware platform than is an application server. And the reason is that application servers only manage code, and then the data is relegated to the bottom tier. If an application needs some data, it runs in the middle tier and requests data from the bottom tier. You end up moving data to the code. If you had a federated data system, so that the data manager was running at the middle tier and at the bottom tier---and object-relational engines are perfectly happy to store and activate functions---- then code and data could be co-mingled on a platform. And you could then do physical database design in such a way that you put the data near the code that needed it, and you wouldn't end up shipping the data to the code all the time. I think that's a much more long-term, robust

way to build sophisticated middleware. So I'd work on trying to prove that that was a good idea if I had some more cycles at work---but I don't.

Among all your past academic and industrial work, what do you view as your greatest successes, and what were your biggest failures?

I think that the greatest success was almost certainly INGRES. That certainly had a tremendous impact. It remains to be seen whether POSTGRES will have the same impact, but as we talked about earlier, that will require "crossing the chasm", and who knows whether that will happen. I think my greatest failure is certainly Cohera. It was a very sophisticated federated data system and there just isn't a big market for federated data systems at the current time. That was a start-up into which I put a lot of energy, and it basically didn't make it.

You would think, though, that federated databases would be very important in the marketplace today. You had the example in your talk today of Grainger with 60,000 suppliers. So wouldn't there be a big market for that?

You would think so. I mean, sooner or later, again it seems inevitable that federated database technology will have to be commercially important. But I think in the commercial market, timing is everything. I think it's just a little ahead of its time. For a startup to be successful, I think you have to have a good idea and you have to have good timing, and Cohera was ahead of its time.

If you could change one thing about yourself as a computer science researcher, what would it be?

I think there are two things that I would do. First of all, I would learn to code better. I never had time to be a real practitioner, and I think we talked earlier about how people with sabbaticals ought to go do something in industry. For me, if I had a sabbatical, I would go be a production programmer.

But you didn't do that on your sabbaticals? Too busy with the start-ups?

> I'd characterize object-relational technology as having been wildly successful among the early adopters.

It takes a lot of time and it just seems to always get pushed back by whatever is more important. So I've just never had the time to do that.

I think that if we're expected to be researchers, one thing we ought to do is be competent practitioners of our trade, and I'm not. I think the problem is that I have enough gray hair that I went through undergraduate and graduate school before we subjected our graduates to a mammoth amount of programming, and so I never got that in school and never had time to do it after that.

I think the other thing I would change is that I went through school at a time when a lot less was known about computer science and so, for example, I don't know very much about compilers. I find the students that we educate today know a lot about a lot of things that I don't know much about. And so I would go back and give myself the kind of education that we give our students these days, which wasn't available in my time.

Thank you very much for joining us today.