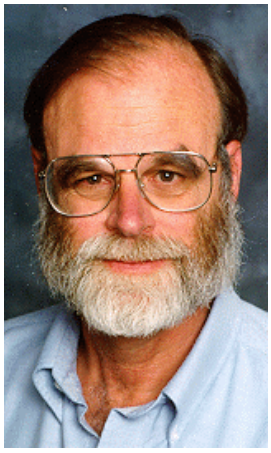
Jim Gray Speaks Out

on Chasing the Object-Relational Rainbow, Why Performance Is a Nonissue, Bad Ideas that Went Good, Reinventing the Field, Sailboats, Lunatic Fringe Papers, Whether to Try for a Home Run, and More

by Marianne Winslett[1]



Welcome to this installment of ACM SIGMOD Record's series of interviews with distinguished members of the database community. For this interview, we're in Madison, Wisconsin, the site of the 2002 PODS and SIGMOD conferences. I have here with me Jim Gray, who received a Turing award in 1998 for his contributions to computer science, especially in the area of transaction processing. Jim served on the President's Information Technology Advisory Committee, which advises the United States President on technical matters, and on an advisory committee for the Library of Congress. Jim got his PhD at Berkeley in 1969 and has worked at IBM, Tandem, DEC, and now Microsoft Research. So, Jim, welcome.

It's great to be here. Thank you.

We're glad to have you.

You said recently in an interview with FTP in February 2002[2] that "we have been chasing the object-relational rainbow and we are near the pot of gold". Other people have been telling me that object-relational products haven't been as successful in the marketplace as they expected. Is there a missing piece of research there, or does object-relational technology need a killer app, or did we misread the market, or what?

I think people are just impatient. It takes longer than you might think for technologies to get into products and get useful. Hash joins were around for about fifteen years before they became more or less central. Now most database systems have hash joins.

In '85, a whole bunch of start-ups started to do object-relational systems. To motivate what they were doing, they argued for the need to eliminate the impedance mismatch, and they argued for encapsulation. They made all the good arguments for why one would want to store *objects* in the database, rather than just numbers, character strings and blobs. All those arguments were salient. They were all correct. But frankly, there was quite an impedance mismatch between the products they made and people's real problems. The OODB folks did an okay job on---in fact, they did a great job---on objects, but they didn't do a very good job of putting stuff on disks, or transactions, or security, and the list goes on. Meanwhile, IBM, Oracle, Microsoft stole their thunder to some extent by saying, "Oh, *we* can store objects; we'll do it this way." It's kind of unnatural, but what's happening now is that first there's been a unification of languages and databases with things like SQL-J or ADO. They provide a pretty good object model for databases. What I mean by that is that we now have good answers to the question: "What kind of object is a database?" "What kind of object factory is a database?" Conversely, there's a pretty good model for how you embed objects inside of relational systems. Right now these things are not very standardized, but I think probably over the next decade the SQL standard, or some successor of the SQL standard, will have objects built into it.

The activity I'm closest to in the object-relational world is what's going on with the folks at Microsoft.

---

[1] This interview has fewer editing marks than might be expected; they were removed during the final editing process. Overall, the relationship between the videotaped and printed versions of this interview is about as close as in previous interviews published in this column.

[2] "A talk with database guru Jim Gray", by Lee The, in *FTP Online*, at http://www.fawcette.com/interviews/gray/.

What they're doing is integrating the common language run time system with a database system, so that you write a program in any language you like, and it runs as a stored procedure inside the database where it can operate on records and fields of records using a record-set object model. I believe the people at Oracle are doing similar things, and people are IBM are doing similar things, with Oracle and DB2, respectively.

The simple metric for measuring the success of object-relational technology is "what can you buy?", or "what are people promising in the next year or two?" The answer to these questions is that all these systems are offering object-relational features, and customers are starting to use them. So, I'd say we just had to be a little more patient.

I understand that your PhD thesis looked at parsing theory. What led you to switch from parsing to databases, and what led you away from the theoretical side of things?

Well, I'm sort of interested in everything, and I felt I could have written a practical thesis or a theoretical thesis, and I was, I guess, kind of in a hurry to get out. I did my PhD in a year and a half.

If you're in hurry to get out, you do a theoretical thesis. The reason for that is that if you do a theoretical thesis, then you do the theorems, you do the proofs, and you describe what you've done, and you're *done*. And in a systems thesis or practical thesis, you have to build a system and then you have to explain what you did. That kind of thesis is twice as much work as a theoretical thesis, because you have to build a working system and then you have to write the thesis about it.

While I was doing my thesis, it was a nascent time in computing. It was in the late sixties and there was lots of ferment in operating systems. In fact, I was at the time working on an object-oriented operating system, and my spare time was spent with the likes of Butler Lampson, Dave Redell, Paul McJones, Charles Symoni, Howard Sturgis, and Bruce Lindsay. I was also working on a simulation system to model urban systems, to understand how you could plan urban growth better. I was doing a lot of different things, but you have to pick some topic that you're going to do your dissertation about, and it's easier to write about things theoretical.

OODB vendors did a great job on objects, but they didn't do a very good job on putting stuff on disks or transactions or security or …

When I finished my thesis, I kept up my interest in theory while I continued my interest in system. I was first a postdoc at Berkeley for two years and then I moved to IBM Research in Yorktown.

You asked me, "How did you get from systems to database systems?", and the answer is interesting. My manager's manager, a guy named Leonard Lu, who's a very, very nice guy, but also a very good manager, came and sat in my office one day. I was working on operating systems at the time. He said, "You know, Jim, we've got a lot of operating systems, but we don't have any good networking systems, and we don't have any good database systems, so if you wanted to actually make a contribution to IBM, it would be really great if you would work on better database systems or better network systems." And I listened to that, and I thought about it. It was true that I was working on things that a lot of other people had made contributions to, and I didn't actually have a really good idea for enhancing operating systems. I'd been working in this area of object-oriented operating system, (capability-based operating systems is another term for it), and it seems that the brightest people had gone down that path and it hadn't worked very well. So I did the natural thing: I started working on database operating systems.

Does that mean that your code went into System R?

Oh, yeah. Yeah, I was actually working in the System R group. My part was to climb the value chain up from how the processes are structured, and how virtual memory works, and how start-up authorization and so on works. Then I focused on concurrency control and recovery issues, to make systems that are available all the time and that make concurrency go away, from the point of view of the user. In essence, Raymond Lori did all the IO and gave us a low-level block interface, and I got to do all the configuration,

startup, process structure, locking, logging, and inter-process communication. This was all done on VM/370 using a real simple language called PLS.

What was that story about the sign that your manager put on your door there?

One of the things that my research advisor Mike Harrison taught me to do is to write things down. So whenever I would go on a trip, I would write a trip report; and whenever I'd talk to people and we had an idea, I would write a memo about our discussion, to document it. One consequence of this is that I wrote lots of papers and went to lots of conferences. One consequence of that is that I got to be very famous for the work of a lot of other people which is not fair – but that's life. I've continued to tell people, you know, that was *Franco* Putzolu *'s* idea or, you know, or that Irv Traiger and I thought of that, and so on. But, since I wrote it down or gave the talk, I got credit for it in the world outside our group.

The net result is that I was much more the researcher and much less the developer, and yet we had this project called System R. Franco wrote 20,000 lines of code a year and it worked, I wrote 10,000 lines a year that sort of worked. The project had all sorts of deliverables, and the managers wanted to show prototypes, and so there was a certain amount of pressure to deliver. At one point my boss came in and put a sign on my door that said, "Code faster." He was somewhat dismayed by the amount of time I spent writing papers, traveling around, and goofing off. I actually coded *too* fast. I created a lot of bugs. [laughs] But, yeah, I wrote, I don't know, 50,000 to 70,000 lines of code in System R. Concurrency control, recovery, system start-up, security, and administration are the parts that I worked on, mostly in the lower half, the so-called RSS layer, of the system.

Are those types of core database kernel topics a played-out research area now, or is there more research that we should be doing there?

At one point my boss came in and put a sign on my door that said "Code faster."

Moore's Law keeps operating and that changes a lot of things over time. Most of the issues I struggled with have evaporated. I counted instructions. I worried about how big things were. I worried about how fast things were. I'm still in shock about the fact that transcendental functions run in under a microsecond. The processor just runs out the Taylor series, and the next thing you know, you've got the hyperbolic arccosine sitting in the register! The world that I was programming in back then has passed, and the goal now is for things to be easy, self-managing, self-organizing, self-healing, and mindless. Performance is not an issue. Simplicity is a big issue.

We used to think that a thousand pages in the buffer pool was a big deal. Now a couple hundred million pages in the buffer pool is a big deal, and the algorithms are very different. The way you do checkpointing, the way you find things, the way you do page replacement in the buffer pool is different now. The concurrency issues are very, very different now, mostly because there is so much more concurrency than there was when we worked on the problems. Concurrency is not what most of the database community should be working on today. In that era there were 50 to 75 people working on concurrency control, and I think it would fine if there are still 50 or 75 people working in that area.

There is still good work to be done in the core, but the low-hanging fruit lies in other areas.

Some researchers say that smart disks are a bad idea, that we tried them before in the guise of database machines, and that they didn't work. I've been told that you think that smart disks are a good idea now. If that's true, why are they a good idea now?

Interestingly, I was talking to Dave DeWitt[3] yesterday, and he asserts that smart disks were not a bad idea in the old days but are a bad idea now. This is the exact opposite of my view. I thought that they were a bad idea back then, because in the old days they were cast as special-purpose computers or special-purpose software that somehow had better performance; the goal was better performance. That argument was false, because the issue wasn't really better performance: even at that time the issue is how to have something

[3] See the interview with David DeWitt in the June 2002 issue of the SIGMOD *Record*.

that a lot of people used, so you could make money on it.

The problem people had in that era was that they would build special-purpose hardware, and by the time they had their hardware working, the general-purpose hardware had gone far enough ahead of them that they didn't really have much of an edge on general-purpose hardware. What's happened lately is that, first, disks have gotten very large, and they've gotten very inexpensive. Second, if you look at a disk unit it has of course a disk in it, and a head, and some mechanical things, and then it has typically a printed circuit card in it with a processor and memory and an interface (which is these days either IDE or SCSI) to a network. These networks are proprietary networks, or bizarre networks like fiber channel SCSI, ATA, and you could replace that network interface with something like Ethernet. So now inside each disk drive you've got a multi-megahertz, probably multi-*hundred*-megahertz processor, with several hundred megabytes of RAM (I'm thinking a few years from now) and a network interface.

Of course you want to have this disk unit to report errors, and you want to be able to configure it and talk to it, so you need to have a little HTTP server and a web interface to this disk unit, to be able to ask questions either through web services or through SOAP or something like that. So inside this disk is an operating system with a network stack and a web server stack. Now it turns out that this processor in the disk unit is more powerful than the processors Sybase and Oracle and DB2 were developed on in the late 80's. And so you can actually run any database system you can think of inside this disk drive. And, of course, it will boot any operating system you can think of. So you can put general-purpose software inside of these disk drives and, in fact, I think that makes very good sense because the thing we want to do is to optimize the disk arms. Disk arms in this world are the most precious resource. The electronics price goes to zero. The only thing that has real cost in this picture is the physical components. By putting more intelligence closer to the disk arms, I think we could do a better job.

Various people (Eric Reidell at CMU, Kim Keaton at Berkeley) did dissertations a few years ago that showed that about 200 megahertz is all you need to keep a disk busy running any of our database benchmarks. There's no question that the disks of a few years from now will in fact have more than enough processing power and storage to run all the database software we've got today. So that's why I think that gradually the software stack will migrate to disk drives.

Incidentally, this creates interesting computer architecture: computer architecture in which there are no computers. All the processors have moved to the disks, or they've moved to the NICs, or they're moved to the printers or to the displays, or they've moved to the keyboards or microphones. They're all moved next to the transducers.

Then the question is, well, just exactly how does this system organize itself? When new devices are added, how do they integrate into the rest of the computer system? This is something else that I think forces us to have much more intelligence in each device. That's why I think that today we're beginning to see this trend, and I think five or ten years fromnow it will have happened. Given that it takes us so long to cook our software, we need to be working on the software today so we will be ready in five or ten years.

It sounds like there might be research issues there that people should be looking at.

Oh yeah, self-organizing systems. Autonomic computing, I think, is the term IBM uses for it. Plug and play is the term that PC guys use.

How does RAID work in this system? Here's a disk, here's another disk, so how do they work together to give fault tolerance? Or is RAID the wrong concept? Do we just do replication at the database level?

The original B-tree paper was bounced; the data cube paper was bounced. The original transaction paper was bounced. Any paper that is non-linear is going to get bounced.

How, when new devices are added, does the system rebalance itself? How do you find anything? What does query optimization look like when you have ten thousand disk drives, each of which is a database system?

So there are scalability issues, there are manageability issues ---those are the two different kinds of issues. Fundamentally, we will have lots and lots and lots more nodes in our network if every time you get a disk or any other kind of device, you get a new network node.

In this future where disk space is almost free, what's the status of strict serializability? Should we be looking at versioning approaches to concurrency control? Should we be rethinking recovery?

Certainly we should be thinking about versioning. There is a really beautiful piece of work that was done by Dave Reed about 20 years ago, called Swallow. It was done at a time when CDs were write-once, read-many. What Swallow did was to never overwrite anything. It was an object oriented database, and it just threw objects into a buffer pool. From time to time, it would throw a commit record into the pool, and when the commit record arrived in the pool, and then the transaction was committed. In Swallow, objects had times during which they were valid. When an update came along, it ended the validity of the current value of the object to be updated, and created a new object with a new valid time start. So there's been some really interesting work in this area, and it's a little bit like smart disks, it's a little bit like object-oriented databases. There are ideas such as this that were bad ideas at one time, but that are good ideas now because there's been a technology shift.

I always tried to be in a situation where I could quit the job I was doing that very day if the need came. That was liberating.

But didn't Mike Stonebraker just tell us in his SIGMOD 2002 keynote address that this kind of multi-versioning approach wouldn't work because it would be too slow?

What he said is that the POSTGRES implementation of the vacuum cleaner was not successful in 1996. (POSTGRES never overwrote any data, and its "vacuum cleaner" was in charge of reclaiming storage occupied by data that had been updated.) And what we're talking about is 2006 or 2016. I actually tracked the price of disks pretty carefully, and the price of disks has gone down by a factor of a hundred since the vacuum cleaner idea failed. The price of disk arms has gone down by about a factor of ten in that time. These ratios really change things. So first, should we be keeping multiple versions of data? Absolutely! Should we organize them the way POSTGRES did? Probably not. How should we organize them? Good question. So that seems to me to be a good research topic.

You've been active in the database field for almost 30 years. Will the field still exist 30 years from now?

Well, no. We computer science people have the patent on the byte and the algorithm. We have the patent on information. That's what we computer science people do: we do information and we do algorithms. In the database field, we have been pretty narrow in what we define as our patent; we currently do SQL. We don't, for example, do the display of or visualization of data. But the "MOD" in "SIGMOD" stands for the *management of data*. Well, the management of data seems to me to include getting the data, storing it, organizing it, analyzing it, presenting it---and especially the presenting part. We've done a lot on the query part but we've drawn this epsilon-ball around the topic of querying, and not ventured too far beyond that ball. So if we continue as we have---and this again goes back to Mike Stonebreaker's comment---if we continue to be very narrow in our definition of what we do, we're dead as a field, because what we do is going to have less and less relevance to the rest of the world. If we are ecumenical, and we say, "We manage information," *that* is not a dying field. We've got a glut of information. Many, many people I talk to say, "I just wish I had more time." So the whole business about taking data and analyzing it and simplifying it and telling people exactly the information that they want, rather than all the information they could have---I don't think that problem is going to go away. I think it's going to become more and more and more extreme. So if you take the broad view of the focus of SIGMOD, then I think actually databases are a growing field. If you take a narrow view, I think that many of the things we're doing now won't have much impact on the people 30 years from now, especially things having to do with performance. Again, going back to the discussion about transcendental functions---I don't count instructions any more. I do count I/Os, but maybe the day is coming when I stop counting I/Os.

Traditionally, in the database community, the closer the research got to humans, the less successful it's been. The early work on natural language query answering, and work on user interfaces, traditionally have not turned out well. Have times changed? Are we now able to succeed where we failed in the past?

Let me challenge that. Natural language database work turned out to be mostly natural language work and not very much database work. But Query by Example gave people a visual metaphor, and Query by Example could have given rise to VisiCalc, the first spreadsheet program. It didn't, somebody who saw QBE (which predated VisiCalc) and then saw VisiCalc would have said that they are the same idea. The QBE ideas did give rise to many products and ideas. There is a unification between VisiCalc and databases that hasn't happened yet, and but we are getting there. This is an excellent research area.

The whole idea of having a nonprocedural programming language was also a great step forward. The goal with SQL was to make it like COBOL, easy to program. We can fault the SQL designers for not being quite ambitious enough, or quite visionary enough, for some people. But SQL was a real step up from where we were with programming languages like PL/1 or data access models like the CODASYL viewpoint put forth by DBTG. I think we are definitely ready for another step up. Compared to FORTRAN, ALGOL, COBOL, and PL/1, XQuery is closer to the SQL trajectory. XQuery is nonprocedural, so it's at that next level above FORTRAN and ALGOL, but it's not two or three orders of magnitudes better than SQL. I think given Moore's Law and the cost of people and the cost of computers, we're ready for a pretty radical departure from how people currently ask questions of databases, and I don't know what it's going to be.

That was my next question.

I don't have a plan. But it is the case that many, many people find Query by Example or VisiCalc or their successors very useful. Microsoft Access is very successful, and it's fundamentally taking a lot of the QBE ideas and just generalizing them. So I think some visual interface, where you're operating on objects that make sense to you, would make databases accessible to many, many more people.

As I was preparing for this interview, people kept telling me to ask you about boat sinkings, potential and otherwise, and near drownings. Is there any story there that you would like to share?

> Performance is not an issue any more. … Even in the 80s, the issue wasn't really better performance. The issue was how to have a product that a lot of people used.

That's a bit embarrassing.

You don't have to tell it to us.

But it's humorous. It's humorous.

I am a romantic. I believe one should have an integrated life; that one should integrate work and play. I had this fantasy about sailboats that said that you bought a sailboat and then you sailed it around. So, I bought a sailboat and lived on it for ten years. When I first bought the boat I went to the library and got a book about how to sail. I'd never sailed, really. One of the many things I did not anticipate is that you need a place to *park* a boat. When you buy a car, you don't worry about a parking space; but with a boat, you have to rent a slip.

So, the slip-search began. I would go out to a really nice yacht harbor, and I would say, "I would like a slip." The guy would say, "All right," and he'd hand me the application form. I'd fill it out and I'd say, "How long is the waiting list?" He'd say, "Oh, 15 to 25 years." I realized that this was not going to work. I got a slip in a fairly sleazy place next to a sugar refinery, not the kind of neighborhood I had in mind. I practiced sailing in the Alameda Estuary -- a good place to learn to sail.

At a certain point I said, "Forget this --- I'm going to move to San Francisco." So I threw off the mooring lines and moved to San Francisco. I went to the harbormaster and said, "Look, I'm just tying up here at the end tie. Here's my phone number. Rent to me day to day. If you want me to move the boat, just call this number and I'll be out in three hours." They *have* to rent to you if you're transient. So, the boat stayed there as a transient for 3-months.

So I'm walking back and forth, back and forth, on the dock every morning and late every night. I notice a boat slowly sinking at its dock. I bailed it out several times. I called the owner and offered him 500$ for the boat. He accepted over the phone and in an hour I had a sinking boat and a good slip in San Francisco.

I now had two boats, which is not especially convenient, so I needed to get rid of the SouSea. It really was sinking, and I didn't want it to sink at the dock. So the challenge was: how do you dispose of a boat? You should call Coast Guard and they will take it away, but I didn't know that at the time. So late at night I, and Bruce Nelson of RPC fame, and a few other people, motored out by Alcatraz. I got out onto the SouSea with my little hatchet and I chopped a little hole in the bottom. The night was pitch black. When I came back on deck, Bruce and my drunken friends were motoring away, and I was standing on a sinking boat in the middle of San Francisco Bay in the pitch black! I don't think my friends fully appreciated the fact that you can't *find* anything out there unless the thing you are looking for is lit. *I* did. So I sat on this sinking boat, and fortunately the boat refused to sink. My friends wandered around out there looking for me, because the joke had ended. I sat there hollering and sinking. That was an interesting scene. But they came back and we went back to the empty slip after watching the SouSea sink.

I guessed that they did come back, because you're here with us today. And you gave them a stern talking to.

Oh, no, no. I still laugh at it and they do too.

Among all your past research, what is your favorite piece of work?

I don't count instructions any more. I do count I/Os, but maybe the day is coming when I don't count I/Os any more.

I actually think the concurrency control transaction model is really beautiful work. A lot of other people did the same work. I happened to be first to publish it. I think it's been rediscovered or discovered dozens or hundreds of times and everybody who's discovered it or rediscovered it is proud of their discovery and rediscovery.

The data cube work is also nice.

Fundamentally, the things I really like are when something is elegant and relevant. You know, when it's a crisp idea and you can state it like that [snaps fingers], and it's not obvious, and it actually has fairly substantial impact. You don't get many of those in your life, at least, I haven't.

Has the President's Information Technology Advisory Committee (PITAC) had a lasting impact?

I go to Washington because I believe that a dollar invested in scientific research gets a payback of ten dollars to society. I've been going to committee meetings, and it's difficult when you sit in committee meetings to believe that anything you're doing has any positive effect.

One of the committees I sat on was this thing called the Presidential Information Technology Advisory Committee, better known as PITAC, and this committee fundamentally said, "Spend money on information technology research." And it said why money should be spent this way. The committee's existence was engineered by some people in the government, and so we had a receptive audience. And the receptive audience, in fact, did pony up, in round numbers, maybe a couple hundred million dollars extra every year. I think the US National Science Foundation Information Technology Research (ITR) grants are a direct consequence of the PITAC. The granting agencies had the PITAC report that they could wave around in Congress and in the executive branch, and as a consequence, the ITR program got funded. I know quite a few scientists who are funded by PITAC and are doing really great work, and so I think that PITAC will have incredibly lasting significance.

Incidentally, we are now in a very critical juncture in science funding, which is to say that High Performance Computing and Communication (HPCC) money was rolled into the NSF base funding, and the ITR was new money on top of that base. In two or three years, we hope that ITR money will roll into the NSF base. There is no new initiative on the horizon that offers a vision that NSF should pursue in the IT space. I'm on the NSF

CISE advisory board, and we are all puzzling about how we can come up with that vision. If we come up with the wrong one, it might not get funded, and that would be horrible. And if we come up with the wrong one, then the right one won't get funded, and *that* would be horrible too. So it's really important to find the right vision. Most people can't have much impact on this, but there are a few senior people in the database community who can have impact. So if a junior person has a really good idea, they should go talk to a senior person. We senior people should get our act together and put forth a good vision.

Do you have any words of advice for fledgling or mid-career database researchers or practitioners?

> If we continue to be narrow in our definition of what we do, we're dead as a field.

You and I were just at a talk by Mike Stonebraker, and Mike's advice was, "go for the home run."

Directly contradicting David Patterson's advice.

Right! Don't go for singles and doubles, don't even go for the outfield, go for the bleachers. And that's a very courageous thing to do. But you have to ask yourself why you're doing what you're doing.

I don't believe in an afterlife, so I think this is it, and I'm trying to spend my time as best I can, and I'm trying to spend my time so I'm proud of what I've done, and I try not to do any things that I'm not proud of. So I have, in fact, only worked on things that I thought could really be significant, and I haven't really worried very much about getting tenure or actually getting a job. I mean, when Irv put this sign on my door saying, "code faster," I said, "Irv, it's easy. Just fire me. You know, if you don't like what I'm doing, just tell me to get out of here." I more or less ignored what he told me to do, and I always tried to be in a situation where I could quit the job I was doing that very day if the need came. I think that was liberating. Of course, it made me a manager's nightmare.

But not everyone wants to live that way, so I think if you want to get tenure and you want to play it safe and so on, more power to you. The world needs a couple of lunatics and it needs a lot of just, you know, solid research. We don't want every paper at the conference to be a lunatic fringe paper. We want 5% to be lunatic fringe papers, but not 0%, please. I think what Mike is troubled about is the 0% part. So, if you're visionary, and you know that you're visionary, then you should definitely go for the visionary thing. If instead you're a scientist, and you follow the scientific method, and you repeat the previous experiments, and you extend the previous experiments, that's a well-traveled road. You can either try to be Pasteur and do really innovative experiments or you can be one of the not so well-known scientists who, in fact, came after Pasteur and built on his work.

You just mentioned conference papers. Some people say the system for accepting conference papers is broken. Do you agree with that?

We've been trying to have a lunatic fringe acceptance mechanism of some sort both in VLDB and in SIGMOD. There is a consensus in both program committees that we should do this, but we haven't figured out how to do it. All the attempts have simply failed. The one mechanism we have is just to invite people, but that's very elitist.

> We're ready for a pretty radical departure from how people currently access databases, and I don't know what it's going to be.

So no, the conference paper acceptance system is not broken. In fact, for the people who are doing science and doing the linear progression type of work, it works perfectly. What it doesn't work for is the non-linear advances. There are famous stories: the original B-tree paper was bounced; the data cube paper that I mentioned earlier was bounced. The original transaction paper that we sent in was bounced. Any paper that is non-linear is going to get bounced.

They all did get published though.

They all got published.

So what was the secret there, to move between the bounced state and the accepted state?

Send it in again. Send it in again.

Did you change it or just send it in again?

The data cube paper got sent to a less prestigious conference, the International Conference on Data Engineering, which at the time accepted almost everything. The five minute rule paper, which I thought was really nice, was essentially invited. I also write papers which I know will never get published. I just put them on my web site. I can have a completely different attitude than the typical person reading this interview or seeing this interview, because I'm not trying to get tenure. I'm not trying to get promoted. I'm just trying to do good work. And candidly, the people who are trying to get tenure and the people who are trying to be promoted are going to be judged by their colleagues at their university. I suspect that if they are doing great work, their colleagues will somehow figure out a way to make a case for them. I would actually count on that process. Certainly in industry that's the case. So, is the process broken? Yeah, the process is broken for certain kinds of things, but this is the system we've got and there are people working on trying to fix some of the problems, and it's more or less a work in progress.

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'm working at Microsoft. I have less contact with the development teams at Microsoft, and particularly the SQL group, than I would like. There just isn't enough time, what with research and doing the other things we're doing, such as the Sloan Digital Sky Survey and the committee work in Washington, D.C., and so on. So my contact with the development teams at Microsoft is much more sporadic than continual. These are very, very exciting times in the database community; there is this synthesis of file systems and objects and databases and XML all happening right now. I'd love to be in the middle of that, and I'm not, and I'm just missing out, frankly.

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

I guess I would try to be more careful. I go by intuition a lot and sometimes my intuition is wrong. I keep score, and sometimes I'm right and oftentimes I'm wrong.

What was your biggest wrong thing? We know about your biggest rights. You got your Turing award for them.

I thought that a scale-out to lots and lots of systems running in parallel side by side was going to have gone big time by now. Dave DeWitt and I wrote a paper on parallel data systems, and I thought by 2000 the phase shift would have happened. What's actually happened is that people have been able to build bigger and bigger and bigger computers, 64-way, 256-way, SMPs. So there hasn't actually been the scale-out pressure that I expected. I didn't think that we would be looking at these incredibly powerful mainframes. I thought scalability past, say, 16 processors was going to be tough, because frankly in 1990 nobody had really built successful machines much beyond 16 processors. But through a variety of software changes, and hardware changes, and better caching algorithms, and better bus structures, and so on, the hardware guys have delivered on scaling up inside a single node. So my premise that we had to do something immediately and urgently was wrong. I still think that scale-out will be the dominant architecture in the end, but I was clearly wrong about the time scale.

Thank you very much for being with us today.

Thank you, Marianne. It's great that you're doing these things.