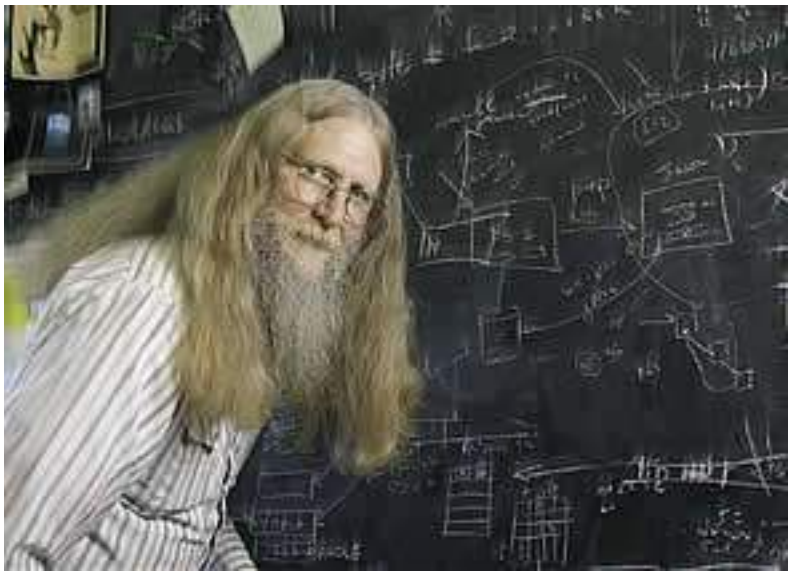
Bruce Lindsay Speaks Out

on System R, Benchmarking, Life as an IBM Fellow, the Power of DBAs in the Old Days, Why Performance Still Matters, Heisenbugs, Why He Still Writes Code, Singing Pigs, and More

by Marianne Winslett



Bruce Lindsay photo by Tom Upton

Welcome to ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today we're in San Diego at the 2003 SIGMOD and PODS conferences. I have here with me Bruce Lindsay, a member of the research staff at IBM Almaden Research Center. Bruce is well-known for his work on relational databases, which has been very influential both inside and outside of IBM, starting with his work on the System R project. Bruce is an IBM Fellow and his PhD is from Berkeley. So Bruce, welcome!

Thank you.

Bruce, I've heard you quoted as saying that relational databases are the foundation of western civilization. Can you expand on this point?

I believe that their adoption in business, government, and education has enabled the progress we've made in productivity, and just made things generally better. I believe that if the relational databases were to stop right now, you would be stuck in San Diego until they got going again. Without these tools to manage the complex affairs of government, business, education, and science our progress would be really much slower.

When I taught introductory database courses years ago, I would always tell my students your famous remark that three things are important in the database world: performance, performance,

and perform ance. Some people would say that this is no longer true today. Would you suggest a different mantra for today?

Well, I think that the remarkable progress in processing and storage makes it possible to meet many requirements with rather modest systems. But faster performance at any level raises everybody's boat on the same rising tide. If the database can be made to run faster on a small system, then you don't have to buy as much hardware to get the job done. In fact, performance is a critical measure of what you're doing. It's like house cleaning: you just have to do it, because if you don't, you can make silly things happen.

Like what?

Like algorithms that take an order of magnitude too long. The mantra "performance, performance, performance" is still important to maintain, because that's where we compete with our competitors and that's where our customers are always happier to go faster, sooner, better.

I think there is another aspect of databases that we often tend to ignore: the utilities, the stuff around the edges. You don't think about it much, but without the utilities you can't really make anything happen. Generic utilities help you load data efficiently, do efficient backups, and manage the organization of your databases efficiently.

Do you see any research issues in the peripheral utilities, or is that more a matter of just good engineering to keep their performance up?

There are tremendously challenging issues in the area of utilities, especially in our 7 by 24 world today, where there's no downtime to do backups anymore. Making all of these operations work online and without disrupting the ongoing production work is indeed an engineering and a design challenge.

Is there anything that the research community should be working on too, or have we done our part there?

I think the research community has done a very good job of ignoring these issues.

I've seen papers on bulk loading.

I've seen some too, but they were not really talking about ***online*** loading, as I recall. The idea of adding, say, 10 GB of records to a table while continuing to process queries on that table is something that I've never seen much in academic literature.

So, PhD students who are looking for topics, you heard it here!

Is there anything you would like to tell us about TPC-C?

Don't do this at home.

Anything else, or is that a succinct summary of the whole thing?

Relational databases are the foundation of western civilization

Well, the benchmarking business is dirty work. The idea is to get the numbers by hook and by crook. And the good news is that about 40-70% of the stuff we do in performance tuning actually ends up helping end users. But much of what we're doing in the TPC-C realm these days is in a performance range that goes beyond what any user is doing. The customers wish they had that many transactions to run, but their businesses just aren't that big. So I think it's a mixed bag.

Of course, any benchmarking effort is used in two different ways. One is marketing and sales, but I think the rather more important and interesting aspect of standard benchmarks is to compare yourself against the competition. If you're the only one out there running a particular workload, you really have little way of knowing whether you're doing a good job or a bad job on it. So from my point of view as a researcher, a benchmark offers a measurement point to help me understand where I'm doing well and where I'm not doing so well.

Is there any particularly sneaky but still totally legal aspect of TPC-C tuning that you would like to mention?

Well, we do things that are very, what should I say? Intense. We get down to the level of worrying about the physical column order in the table so the reference columns are near each other, minimizing cache misses during fetching. This is feasible in the TPC-C benchmark because there are only five tables and only ten to fifteen columns in each table. In a more realistic application, where there are many more queries to be considered, the tables are typically much, much wider, in the 80 to 100 column range; and there are dozens if not thousands of tables. Then this kind of analysis is no longer practical.

Tell us about Heisenbugs.

Oh, Heisenbugs. One drunken night in Berkeley, Jim Gray and I were working away on an operating system project, the CAL-TSS system [an NSF-sponsored research project in the 1960s that developed a capability-based operating system for the CDC 6400]. We'd naturally had the kind of bug that the more you look at it, the less you can find it. Every time you turn on a measurement, the bug goes away and you can't see it. I'd been studying a little bit of physics at that time, and I was struck about what they were saying in atomic physics about the relationship between the measurements and the thing being measured. And so it struck me that we were somewhat in the same game: when you looked, the bug went away, because the measurement or the observation affected the phenomena you were trying to see. Hence the term “heisenbugs”.

I have heard that colloquium speakers at IBM Almaden used to be in terror of you, that you patrolled the seminar rooms like a lion, but that you've given up this practice in the last few years. Is that true? And if so, what prompted the change?

Well, I think that probably the reason that I'm not doing as much is that I’ve gotten lazy, or else I don't understand as well what they're talking about. But I do admit that I have little sympathy for shallow arguments.

Who will keep the speakers honest then if you're not down there?

I'm hoping somebody else will step up to that role.

At the time that System R was built, it was impossible to realize what an impact it would have on the world, for better and for worse. With perfect hindsight, what do you wish had been done differently in System R?

Well, there are a few little things that I think we made mistakes on. I think one of the biggest mistakes in the SQL language was the "select *" notion. The "select *" constructs were developed to provide ease of use, but they really have been an implementation nightmare. "Select *" means select all the columns, but in what order?! The relational model says nothing about what the order of the columns might or should be, so you're really getting down to a physical storage issue here. More problematic, when you create a view as "select * from table where predicate", what happens when I alter the table and add a column? What does the view mean now? Does it have more columns than before, or not? We have to figure out what to do about those views when their underlying tables are changed. In fact, we have to make the meaning of the views be unchanged by the update, as otherwise existing applications would break. So I think that this is one example of not quite thinking it through all the way.

Three things are important in the database world: performance, performance, and performance

In hindsight, what do you view as the key elements of System R's ultimate success?

Oh, I have a revisionist's history story on that. I think there are three elements of the System R prototype and its deployment in test environments that had an effect on the eventual adoption of relational data storage systems. The first one is the obvious one that everybody suggests: the invention of a nonprocedural query specification was a tremendous simplification that made it much easier to specify applications. No longer did you have to say which index to use and which join method to use to get the job done. This was a tremendous boon to application development.

I like to tell the story of what it was like to be a DBA in those days, because the application backlog was huge. It was a good deal to be a DBA, because you controlled what got done next. It was necessary for anybody that wanted anything to get done to bring you gifts, you know, bottles of things. And therefore it was a good deal to be a DBA, because of this application backlog.

I think the ability to more simply specify how or what it was you wanted retrieved from the database, instead of how to do it, was a tremendous advantage and has paid off enormously. But the real reasons that the relational system won have nothing to do with the relational model; they have to do with the fact that the early prototypes, both System R and Ingres, supported ad hoc queries and online data definition. You could try out a query right away, rather than typing it into your application, compiling the application, running it, and having a failure, usually a syntax error. Even if there were no syntax errors, you still had the advantage that you could look at the query answers and see if you had the right tuples. Ad hoc online query execution was something that was unheard of in the database community at that time, so this was a tremendous boon to application developers and to people who were browsing the data.

Online data definition was also a great boon to the development of applications in database systems. Previously, if you wanted to add a new table or add an index or add a column to a table (or *dataset,* as they were called in those days), you had to shut down the database system, invoke some compiler-like tools to define the new data entities, start up the database system, and try it

out. This could take hours. With online data definition, it's a matter of just typing the right mumbo jumbo and the system says, “Sure, you got it, try it!”

[Note: The reader may enjoy comparing Bruce’s answers to the previous questions with Pat Selinger’s answers in her interview in the December 2003 SIGMOD Record.]

When the non-relational people saw what was happening, couldn't they somehow retrofit some tools to work with their model, so they could do these things online too?

It goes deeper than tools. It goes right to the heart of the system in very fundamental ways, such as maintaining the coherence between the metadata that the system uses at run time to figure out where things are and which way's up and which way's downand the actual use of the system. So a difficult and rather successful effort in the System R project was to figure out ways to synchronize the modification of metadata without impacting or interrupting the ongoing work involving other data objects.

Bruce, some of my previous interviewees have said that the core areas of database technology are played out as a research area. You've spent your whole career working on core areas, so I thought you might like a chance to comment on this assessment.

I think we've come quite a long way in many areas---certainly in the areas of transaction management, recovery, concurrency control. I think we have those under control now, and we have a pretty good feeling for what the fundamentals are in those areas. For areas further out from the core, such as query processing---we continue to evolve the query processing algorithms. We find new indexing methods and consider them. There's still polishing to do in the area of index utilization. There are many more things we could do with indexes than we're doing now.

But I think the action is moving to the periphery of the database. The area of manageability and usability in the database has become enormously important. The complexity of the systems is probably not much greater than the complexity of the systems they replaced, but the cost of people that can deal with that complexity has doubled in the meantime. So our ability to reduce the human expertise needed to keep these systems running, fed, and happy is very important. The ability of DBAs to manage more easily, to have fewer knobs and fewer buttons but still get the job done, is very important.

I must not go overboard here. After all, I've heard people say to me, “I would like to have a database with no configuration parameters. That way I wouldn't have to hire a DBA and pay somebody to tell me how to tune them.” And that's certainly possible. Let’s compare a cockpit of a 747 with the cockpit of a Cessna. You say, “Gee, the 747 stinks. Too many knobs and dials in there! Nobody can operate this darn thing.” But, darn it, 747s can do things that Cessnas can't. And that's probably part of the reason we have so many knobs and dials. Because we apply relational technology to such a wide range of applications, from decision support to scientific analysis and transactions, we need to have mechanisms to tailor the installation to the needs of the particular applications.

Do you think a self-tuning system could do that tuning automatically?

I think we're making progress, but we're getting into that dangerous realm of AI and machine learning where I think the successes in our industry have been few and far between.

You have said that important issues and tensions remain between object oriented and Java bean applications and their use or misuse of relational persistent stores. Can you expand on this comment?

The difficulty with the object oriented applications is that these programming systems deal only with whole objects. Persistent whole objects may have a hundred or two hundred attributes. The use of that object in an application may involve only a few of its attributes. The current interfaces that we use, partly due to weakness in the object oriented application programming environments, don't seem to be able to focus the access to the persistent store towards the attributes that are of interest. We've actually had some customers with major application programs with the following problem: whenever one attribute of the object, Java bean, whatever it is, gets updated, they immediately update all of the attributes---to the value they had before, of course. Well, at least that works---we have the right data in the persistent store! But there's a cost to be paid in performance, performance, and performance.

What do you think they need to do to fix that problem?

I think it will require quite a lot of engineering to deal with partially materialized objects in the object oriented world, but they certainly could do a lot better job of realizing which parts of the object have been updated and mapping them back to their corresponding storage locations.

One of the biggest mistakes in the SQL language was the "Select *" notion

Let me see if I've got this right. You're an IBM Fellow; therefore the road is paved with gold for you at IBM. You can work on anything you want, you have your own pot of money to spend as you like and people bow down before you as you approach. Right?

I don't think it's quite that grand! There is this widespread misconception that we actually have a budget. We don't. Last year, for the first time, they gave us each $5,000 to spend as we wish. So big deal!

The ability to choose your own area of research, though, is quite a privilege. I've enjoyed being able to decide what I thought was important to work on and spending my time in those areas.

The idea of bowing down is totally silly. At IBM and elsewhere, I believe you have to interact with people on a basis of what they are and what's happening. I got to where I was by paying no attention to what anybody's position was but paying a lot of attention to what they knew and how good they were at explaining it to me.

Can you correct the flaw in the following chain of logic? The 56 IBM Fellows are the people who have made the largest technical contributions to IBM. Therefore they are likely to have unique insights into what IBM should do to be successful in the future. Therefore they should take on management roles to expand their spheres of influence. Therefore you should be a manager.

Oh, I think that making me a manager is like teaching a pig to sing: it doesn't work and it annoys the pig. I believe that the most foolish thing that IBM could do would be to encourage their senior technical staff to focus on management. My experience is that technical progress is made in our area only by teamwork, because the things that we attempt are too big for one person to do alone.

And my experience is that I interact best with the technical team when I'm part of the technical team and I'm doing the same work that they're doing, and whatever leadership or ideas I can provide are just icing on the cake. Technical people should do technical work. They should be at the bench fussing with the chemicals, or shoving the code around, because that's where we'll learn what's really going on and can make the biggest contributions.

What about the other Fellows in the database area who are in some part of management?

Well, I think that there are only two: Pat Selinger has been moving up in management because she's tremendously good at working with people. She can get people to do whatever she wants them to. I don't know how she does it. She did it for years with me, and it's a complete mystery. And Hamid Pirahesh is stepping into the shoes of managing the database research at San Jose Almaden Lab because somebody had to do it. But I believe that the idea of pushing technical people into management is really a very bad idea.

As I was preparing for the interviews this year, Jim Gray suggested that I ask Pat Selinger about how she managed difficult people like Jim and you. More generally, in your view, what is the role of a manager in managing people like you and Jim?

Stay out of the way, and keep bad things from happening to us, and calm down the people we anger.

Is it true that you thrive on the adrenaline that comes from writing code?

Oh, yeah. That's the real contribution, to actually build something. There's nothing more fun than finding the last bug---ha, ha---and developing an algorithm and getting it to work. That's really where the rubber meets the road. All the rest is just talk.

But doesn't code-writing narrow your sphere of influence?

Not at all, because the people that I need to have influence with to get my job done are the people that write code. Because what do I deliver at the end of the day? A working product. And I can't provide the leadership needed to motivate the people who have to work with me to build this product, or to get them to accept any of my ideas, unless I can demonstrate to them that I can do what they are doing.

What led you to join industry and to stay at IBM, as opposed to spending some time in an academic position, joining a start-up, or working for several companies?

My decision when I got my degree was whether to pursue the academic path or an industrial path. My concern with the academic path is that you are not really a researcher very much of the time. You are a teacher, and I do like teaching. Teaching takes a lot of time. But you are also an administrator, and you are also a funding agency. What I noticed at the university is that the faculty spend an inordinate amount of time not doing research and not doing teaching, Instead they are doing management tasks, funding tasks, and so on. I felt if I was going to be in research, I would like to be somewhere where the funding was assured and the research projects were chosen on their own merits.

Now why not change companies? Well, basically I think that that’s because I'm a coward. It took me years and years to get any credibility, because of the way I look. So I had to overcome this, and having gained credibility, I said, why should I start over? I really believe that if you are

going to work with people in a team and accomplish things as a group, you must have mutual respect. You just can't walk into a room and say, “My name is Bruce Lindsay, I am an IBM Fellow, you must respect me.” You have to earn the respect by working with the people for long enough. Word of mouth sometimes helps: “Oh, he's all right. He won't actually eat you,” that kind of interaction.

Did Jim Gray cut off his hair before or after he left IBM?

He actually cut his hair before he came to IBM. At Berkeley we were working together, and Jim Gray's hair was pretty long. Then he got a job at IBM and we started to notice that his hair was getting shorter. Not all of a sudden did it get short; it got shorter bit by bit, and by the time he went off to work at IBM Yorktown, his hair was the normal length for that era. We actually had a cartoon (drawn by Davd Redell) attached to a plan in the office that depicted Jim Gray with shorter and shorter hair.

If you were just finishing your PhD in computer science today, what would you do?

Oh, I'd try to get a job.

An industrial one?

I think an industrial job, yes. You asked earlier why I didn't join a start-up. I believe some of that work is very exciting, but also I believe people need to have a life. Eighty hours a week were not appealing to me.

Making me a manager is like teaching a pig to sing: it doesn’t work and it annoys the pig

Now that multi-disciplinary research topics have become extremely common, are single area people becoming obsolete?

I think you do have to have some ability and understanding of a diverse number of fields. I am a little bit disappointed that some of the database people, new ones that I run into, really don't know very much about current directions in microprocessor architecture. You may say that microprocessor architecture is not relevant. I think it is. But more and more we really have to go outside the computer science area. We need to have expertise in physics or chemistry or life sciences to understand the problems that are important to work on and to bring our computer science skills to bear on solving them. After all, a solution of a problem in computer science, such as a better compiler, is not a solution to anybody's problem. The problem is only solved when a program that was written using that compiler solves a molecule or sends you your Visa bill.

Besides trying to move the company, does an IBM Fellow have increased responsibilities in the area of mentoring?

Oh, absolutely. It’s actually everybody's job to be a mentor, but Fellows should be good at it because they have demonstrated that they have the confidence to move forward and to do bold things. And that's basically what you're trying to teach somebody when you're mentoring them: stand up for yourself, speak out, and move forward.

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 I would work on indexing a little harder.

What aspect of indexing?

Oh, there are many aspects of indexing that I think need improvement. I think we can do a better job in the searching. I think there are exciting things to do in multi-dimensional indexing. I think there are a lot more advanced ways that we can use database indexes for indexing on columns that contain sequences or XML data, and for partial indexing. With partial indexing, we would index only some of the rows, based on some predicate. For example, we might not index the NULL values, or we might index only those salaries greater than $100K.

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

Be taller?

But what advantage would that give you?

I could look Mike Stonebraker in the eye.

Anything else?

If I could be better at understanding and proving theorems, I would be very happy.

What kind of topics would you prove your theorems about?

Well, I've been working on an algorithm for peer-to-peer replication, and I can't find a proof that's correct.

Thank you very much for talking with me today.