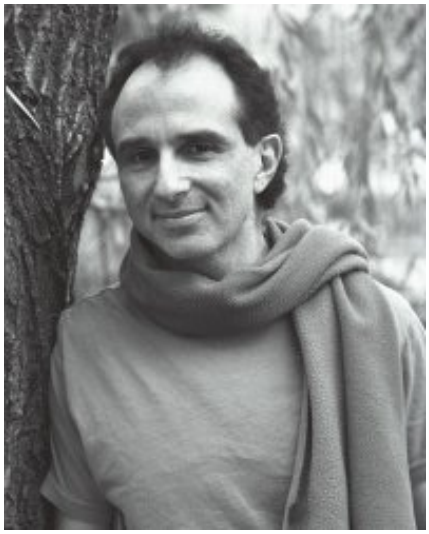
Dennis Shasha Speaks Out

on How Puzzles Helped His Career, What Drives Him to Write, How We Can Help Biologists, the Principles Underlying Database Tuning, Why He Wears Shorts All Year, and More

by Marianne Winslett



Dennis Shasha
http://cs.nyu.edu/shasha/

Welcome to ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today we are in Providence, site of the 2009 ACM SIGMOD/PODS Conference. I have here with me Dennis Shasha, who is a Professor of Computer Science at the Courant Institute at New York University. His research interests include biological computing, database tuning, cryptographic file systems, and pattern recognition. He is a prolific writer, both within and outside of computer science.

Let's start off with what our readers really want to know: do you really wear a scarf and shorts every day, through both summer and winter?

Yes, I'm afraid I do. In the winter I get a lot of funny comments, like people asking me, "So, what are you drinking, man?" And one time I gave some money to a guy on the street, and he said, "You know, I should give you money, you should buy some long pants!" So it has been worth it just for the comments!

Have you ever gotten frostbite?

No, actually I am warmer wearing shorts, surprisingly, because the only part of me that gets cold is my hands. Because I wear shorts, I put my hands in my pockets. Because shorts are so thin, I get warmed by my legs, so I am actually warmer in shorts than in long pants.

What about the reverse? In summer when you've got your scarf on, how does that work?

As I said, only a few parts of me that get cold. One part is the neck, and the other part is the hands. So it actually makes sense. Everything about me is a little bit strange, but it makes sense in that context.

Have you ever gone any place really cold during winter?

Waterloo, Canada in February. The waiter came down in the hotel and said, "Sir, you do realize we are in Canada, don't you?" But it was fine.

Biological computing: what's in it for database researchers?

I think it is a great field because computing, after all, started as an aid to science, and we are going back to that tradition when we help biologists. Biologists need a lot of help, in all kinds of areas. Above all, what they care about is experimenter time. They don't care about computer time, per se, as long as they don't need days of computer time. If a program runs for 20 minutes or 30 minutes, it's okay. They do care about reducing the number of experiments they do, and still getting more or less the same results. Also, they want to know how to get stronger results from the experiments they do. I help biologists design experiments using statistics and combinatorial design, so that they don't have to run experiments on every point of an entire search space, and they can examine just a subset of that search space.

Very often, scientists' search space for experiments involves many variables whose values they can modify. I work with plant biologists. They might add more nitrogen or less nitrogen, more carbon or less carbon, and so on. With all these possibilities, the search space can contain millions of points, if you consider all the possibilities. Using combinatorial design, you can reduce it to perhaps a few hundred, and still get a very well sampled experimental space. Then on the analysis side, what is really nice is that you need a little bit of everything. You need some machine learning, data mining, statistics, and there is a lot of simulation. So overall, there is a lot that computer scientists can do to help scientists, and if a database researcher is willing to be all of computer science to the biologists, then there is a lot to be done.

Should we be training the biologists differently? Do they need to know more computer science than they are learning, so they can do it themselves?

I think biologists do quite a bit themselves. For example, all biologists know how to use BLAST for comparing sequences. But on the other hand, there are also things that the computer scientists need to be doing. Those things often have to do with either a new kind of analysis that the biologists wouldn't be able to do by themselves, or just looking at the data differently, or sometimes even helping the biologist design a better experimental program. One biologist that I work with, a woman named Gloria, has been working on a certain kind of network for nitrogen uptake in plants. What she does is great, but what we would really like to do is to take this network and reverse engineer it, like electrical engineers would do.

What do you mean when you say "reverse engineer it"?

We'd like to find out which genes are connected to which other genes. It's as if we were probing an electrical circuit and finding out which circuit elements are connected to which other circuit elements. In doing that, an electrical engineer puts current into a circuit element, or maybe detaches a wire. With plants, we can't exactly detach a wire, but we can make genes overexpress, so that they produce more protein than they would normally, and then we can see the effects of overexpression. By doing that, we can march through the circuit, in principle, and find out exactly what is happening. I find that very exciting. Gloria and her lab have bought into this idea, and they are starting to do those kinds of experiments, so it is a wonderful collaboration.

But don't you get all sorts of side effects once that one gene starts producing too much protein?

Sure, but that is exactly what you want to find out. You find out the side effects, and you can find out which other genes that gene is affecting. So that helps you figure out, "Ah-ha, that gene must be connected to the other gene!"

It sounds like a close partnership that you are talking about.

Yes, it is really a great partnership. We have a meeting every week, and we have cookies, and we make a lot of mistakes in the other field… One time I said "Alabama" when I meant "alanine", because both words have the "ala" prefix and I just forgot. So we really have a good time together.

With Cathy Lazere, you wrote a book of biographies of great computer scientists, called Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists. Tell us a story from that book.

When we wrote *Out of Their Minds*, we wanted to write about the seminal thinkers of computer science. We don't mean Alan Turing, of course, so much as the people who have made computing practical in a technological sense. So we chose people who had already won the Turing Award, plus others who won the Turing Award soon after we interviewed them, although I am sure there was no causal connection. We started with algorithms people, like Michael Rabin, Dijkstra, Knuth, Lamport, and Cook and Levin for complexity theory. We interviewed languages people: McCarthy, Alan Kay, and Backus. We interviewed computer architects, and we interviewed people in artificial intelligence. I avoided the database field because I felt that if I had interviewed one person, maybe other people would have been upset. Probably that was a mistake, so perhaps including the database field would have been good too.

A typical story from that book is that of Backus. Backus in some ways personifies a very unusual kind of researcher, who nevertheless has the emotive influences that drive many researchers that I know. Mainly, he was easily angered. He had a very mixed, not very studious, childhood. At 25, he was graduating from the general studies program at Columbia, with a math degree, not knowing what he wanted to do. He got a job at IBM almost by accident: he was taking a tour of IBM, and somebody said, "You can interview here if you want. You can maybe get a job." He took a test and they hired him.

At IBM, Backus was programming in machine language, which he found horrible. He started what became the predecessor of FORTRAN, namely, a way of doing floating-point arithmetic. When he proposed FORTRAN, he said, "I am doing it because I find programming is too tedious, too difficult for people." Interestingly enough, he was opposed in this desire by John von Neumann, who thought that programming really wasn't that hard. But von Neumann was not your average person, and Backus understood this, and the IBM management understood it too. So when Backus started the FORTRAN project, he was able to get the support of the IBM management, who understood that if they could make software programming easier, then people would buy more computers. And that is what happened.

Later, Backus participated in the ALGOL discussions. He said it was very frustrating; people would just give an example for this, give an example for that, and nothing was getting anywhere. And so he developed Backus-Naur form, which is really very similar to the Chomsky notation for context-free grammars, just to make things clearer. So just because he was annoyed, he did a lot of his inventing.

You make it sound like he is an accidental computer scientist.

I think he is not accidental, because I think a lot of invention is annoyance. It is true that he might have gone into another field, because he wasn't a particularly outstanding math student, by any means. So he is an accidental computer scientist in that sense.

I have a colleague, David Mazières, who is at Stanford now, and he is also very motivated by anger. It is a fun anger, it is kind of enjoyable to watch. He gets angry, and then he does something about it. He builds a better system.

What angers you?

I am not one of those kinds of people. I like hard problems where I feel that a solution will really change things. Of course, it doesn't always work out that way, but sometimes it does.

Have you had a favorite hard problem in the past?

I have had a couple that I really enjoyed. One was the work that we did in tree matching. This work got started because a student of mine, Kaizhong Zhang, came to me and said, "You know, this journal paper claims to have a really fast algorithm for this problem, but the algorithm doesn't actually work." We looked at it together, and indeed, it didn't work. Sometimes a problem seems closed because somebody else claims to have solved it, but in fact the problem is not really closed because it has not actually been solved. Noticing this discrepancy can be a fun way to start a research project.

A sequence of accidental coincidences led me from one research area to the other: meeting people, understanding their problems, coming up with algorithms to solve them, and getting the algorithms used. That project on tree matching led to other projects on graph matching, and led eventually to my work with biologists, because one of the applications was RNA secondary structure.

You are a puzzle column writer for Dr. Dobbs' Journal *and for* Scientific American. *What's that like?*

Dr. Dobbs' Journal stopped publishing a couple of years ago, and *Scientific American* is going through economic problems, so it is not clear that the column is going to continue. But it has been fantastically fun. As the column writer, you write puzzles and people send in solutions. And when people send in solutions, you become friends with them. I have what I call my "puzzle brain trust": sometimes I invent a puzzle that I can't solve very well, so I send it out to various members of the puzzle brain trust, and sometimes they come up with much better solutions than mine. It has been a remarkable experience, because I haven't met my puzzle brain trust in person. In fact, I just discovered a couple of years ago that one of them happens to be deaf, and I had no idea. It has been just a wonderful experience in terms of the intellectual back and forth with many of the people I have communicated with. Nothing delights me more than finding somebody who has a better solution than I do.

The puzzle column also has helped me formulate problems for the biologists that I work with. Sometimes the discipline of making a puzzle out of a problem can help you simplify the problem to its essentials. When you do that, you really start to understand it, and then you start to ask the right questions, and then sometimes you find a better solution. For example, the combinatorial design that I use for biologists turned into a safe-cracking problem as a puzzle. Instead of a dial on a safe, you have many three way switches, and the goal is to open the safe, given that it will open only if two switches (and you don't know

which ones) are in a certain configuration. It's a combinatorial design problem, and when you explain it that way to biologists, they understand it – possibly mainly because they secretly want to break into safes!

When you publish these puzzles, do you always already have a solution?

I already have a solution, but sometimes it is not the best solution. Many of these problems are continuous or NP complete, and so it is difficult.

I did one puzzle about Zambonis, which are machines that clear off the ice after people have been ice skating on it for a while. My inspiration came from observing the Zambonis and noticing that although the drivers seemed to be having fun, they definitely were not using the most efficient route. About a week ago, I received an email from the Zamboni people, who said, "We would definitely like to find the solutions to this puzzle."

In the problem, you have an ice rink, abstracted to a bunch of points filling an oval shape. The Zamboni machine has to pass over every point, but cannot turn at more than a 45 degree angle. How do you make it go through all those points in as short a time as possible, so that people can start skating again? I suppose with enough computing power you could solve that problem, even though it is quite complicated because you have the 45 degree constraint. Anyway, some readers found a really nice solution. Another reader found a really beautiful solution for two Zambonis that was also kind of important, because the two Zambonis can work together without colliding. So we will see, maybe Zamboni will use it!

You've also written about recent Russian immigrants. What is the story there?

I like to do things that interest me, and I don't sleep much at night. I usually sleep 5 or 5 ½ hours, and then I take a few 10 minute naps during the day. When I wake up in the middle of the night, often I want to write, because I just like it. And sometimes daily experiences influence me.

In the case of the Russians, I was director of undergraduate studies in the early 1990s, when a lot of Russians were coming to New York. They were kind of bimodal. Some of them were just brilliant. Some of them thought they were brilliant, but weren't. There weren't many in between, for some reason.

One day a young man came into my office with his teacher, and he was a *very* young man, just 12 years old. His teacher said, "He's in my freshman Pascal class, and there is a problem." I asked what the problem was, thinking that the student was too young and couldn't understand the material. But the teacher said, "Viktor is too good." I said, "Oh, really, so tell me Viktor, what have you done? What is the biggest program that you've written?" Viktor said, "Well, I've done a simulator for high temperature superconductors." And I said, "Oh, that's good."

My colleague asked Viktor about the math he used, and Viktor was explaining Newton gradient methods and all those sort of things, and he certainly understood what he was doing. So then I said, "Okay, Viktor, which programming languages do you know?" "Pascal, FORTRAN, and Lisp, and C, and C++, and I think that is all," he said. "Do you know any graph theory?" "A little bit," he said. "Could you write us a connected components algorithm?" "In which language?" he asked. So I said, "Oh, I don't know, maybe a set oriented language." So he does that in his 12 year old handwriting, from the top of the board to the bottom, without a cross-out, and it is perfect.

My colleague said, "You have a loop there, and you have to compare two sets in that loop, that's an expensive operation. What will you do?" "Well, I will probably use bit vectors," Viktor said. Then my colleague said, "You could also just have a flag that changes its value if there is some change in the loop." And Viktor said, "Oh yes, but it wouldn't be so pretty."

I felt that I had to write a book about these people. I wrote with a Russian playwright, and we wrote about mathematicians and scientists but also business people and sex workers and writers. So it was quite an eclectic crew, and it was really a fun book to write.

When I write, I try to write something nobody else is writing. I really respect people who write wonderful textbooks, like let's say, a real database textbook, but I would never try to do that, because I feel that they are already doing a great job. Even if conceivably I could do a better job, it would only be marginally better. For example, I wrote my book about database tuning because there was no such book. There were books about database tuning for particular systems, but never the general principles.

Are there general principles? Can you generalize beyond what each separate system does?

I think there are a lot of general principles. One principle is that people often tend to tune things that they notice are bad, even if they are not important. A second principle is that very often there are systematic problems because people don't realize that one fundamental principle of all databases, in fact all of computing, is that starting something is expensive, but once it's going, it is quite cheap to have it continue. For example, reading a sector of a block is almost as expensive as reading the whole block. Emitting an SQL query every time you go through a Java code loop is much worse than having just one SQL query and then stepping through the resulting data. That theme occurs again and again, and I call it "starting is expensive, but running is cheap."

Another principle is that partitioning is not understood too well. People understand partitioning data across pieces of hardware, but they don't really understand that there is also temporal partitioning. Very often, you might want to do some work during the day, and other work during the nighttime. When you say that, people respond, "Well, of course," but it's not the first thing that comes to people's minds. For example, banks send out statements. When they used to send out paper statements, they would send 1/20 of the statements every day. And that makes sense. It doesn't matter that you don't get your statement at the end of the month – it could be on the 12th, but who cares as long as you get it every month? And similarly, for production databases, one can take advantage of temporal partitioning.

Those are three of the principles. Surprisingly, those principles are almost generative. You can apply them to almost every level of the database, from buffer management to application design.

Do you think that the general principles of database tuning would fit into a standard CS curriculum?

I think so, but of course I am very biased. I teach it, and students love it because it is very practical. I think we do a very good job in this community teaching a database course on SQL and normalization. Once we get to indexes and transaction processing, we have to recognize that most of our students are barely going to be conscious of that once they leave – indexing perhaps, but not how to build the index, just what indexes do. Although it is all very valuable, we should recognize that database tuning is what they will actually do if they use databases, either as database administrators, or even as sophisticated application programmers. Because how many will work at Oracle, Microsoft, or a few other places?

We tend to make our students take a course on operating systems, but most of them don't go on to design operating systems. Does that represent a failure of our educational system?

No, because operating systems and database internals give people a good sense of what system building is about, and that is important. Very often, students go on to write concurrent systems, so they have to understand system building, and operating systems and transaction processing are great ways to learn that. But, nevertheless, I think there is a place for database tuning, at least a little bit. I welcome anybody in the community to take advantage of my notes on database tuning, which are widely available. Philippe Bonnet has a wonderful infrastructure for a database tuning experiment, and that is also available. It uses mySQL, and we have some scenarios about a travel application, and some other scenarios.

Tell us about your life as a fiction author.

I also loved puzzles, because puzzles helped me overcome my problems in my first job, which was to design circuits for large mainframe processors at IBM. Although I technically had an engineering degree, really I'd only studied information theory. So when I got to IBM, I didn't really understand electricity, not really. I bought myself a Heathkit and I built stuff, and I really tried to understand what was going on. But what really helped was reducing what I had to do to puzzles. Then through those puzzles, I tried to understand more details afterwards.

One of my jobs was to design circuits to check other circuits. The idea was that if a circuit failed, another circuit could say that that circuit failed. Circuit elements were still expensive enough at that time that we didn't want to completely duplicate the circuit. We wanted something that was economical, yet still could check the other circuit. Take a decoder circuit for example, a 4-16 decoder. The output should have only one element that's set to 1, i.e., only one of the 16 lines should be a 1. How do you do that in just a few gates? That puzzle appears in the very first of my puzzle books. The naive solution is to take every pair of the 16 and see whether both outputs are 1, in which case there is a problem. But that is many, many, many tests, at 16 choose 2 circuits. One can design a much better and more efficient circuit by noticing that this is 4 bits, so if two of these lines are 1, then they must differ in one bit.

So puzzles always helped me. And then I wondered, how can I give puzzles to other people? I always really loved Sherlock Holmes, except that Sherlock Holmes periodically would just leave and then come back and say that he had discovered something, which would often be the key to the whole mystery. I felt that was unfair to the reader. Any reader who tried to solve the mystery himself/herself was robbed of that privilege if Sherlock Holmes pulled a wildcard out of his pocket, so I said, *no wildcards*.

Dr. Ecco is kind of like Sherlock Holmes. He has a friend named Prof. Scarlett, and Prof. Scarlett acts the Watson role. Prof. Scarlett explains the puzzle to us, but never does the reader get less information than Dr. Ecco. So if Dr. Ecco can solve it, the reader can too. And indeed, readers do solve it. Most of my puzzle books have puzzle contests where the solutions aren't given in the back, and the person who wins that contest gets free tickets for two to London and back from New York. It has worked out well.

When you read a murder mystery, do you usually figure out who did it before the end?

I try. Sometimes it doesn't work. But I like to, I like that idea.

Why is it that modern mysteries have to be murder mysteries? Why can't they just be mystery *mysteries?*

I agree with you. I think there is plenty that is mysterious and doesn't have to involve death. *The Name of the Rose*, for example – Dr. Ecco's name has two Cs, and Umberto Eco only has one C, but nevertheless, *The Name of the Rose* was a very inspiring book for me. That mystery fundamentally didn't need to involve death. And that book really involved the reader, took the reader to an entirely different world.

What will your next book be about?

Cathy Lazere and I wrote a book that's at the publisher's now. It's called *Natural Computing: DNA, Quantum Bits, and the Future of Smart Machines*. [It came out in May 2010 in English, May 2011 in French, and soon in Japanese.] This book is much more speculative than *Out of their Minds,* but in the same format: biographies and what people are doing now. We look at future directions in computing based on people who are trying to solve extremely hard problems. For example, how do you make spacecraft survive for a hundred years if they are going to make a long trip? Another typical hard problem is protein folding. Another is how to make robots that fend for themselves. There are many very difficult problems that I think require a new kind of computing, and the new kind of computing we seem to be seeing is based on a life analogy. Either it involves genetic algorithms, or more generally it involves feedback in very interesting ways, so that systems are built to be adaptable. Very often, not only do they involve nature in a metaphorical way, but also in a real way. For example, we interviewed Jonathan Mills from Indiana, who does analog computing, which people thought was completely dead, but which can be really good for certain things. This summer, in fact, we are doing an extremely intensive project with him on protein folding. After all, proteins fold in a millisecond, but the best computers for doing this, like David Shaw's Anton, take months to fold a small protein. And those are multi-million dollar machines! There is something wrong with that picture, because clearly, the protein is not computing like Anton does. So it is quite possible that there are other paradigms that could work better.

Beyond the puzzle aspects, are there other characteristics of the types of problems that you like?

I am really interested in very challenging problems in any part of computer science. I think there is going to be a lot more autonomy in computing systems. Even in sensor networks there has to be a lot more autonomy, because you can't possibly go out and repair all these sensors. It is even more so in some of the wild things that people are doing, like in DNA computing, where there are literally billions of little DNAs running around annealing with another, or even in cellular computing where on your thumb you have more cells, little pieces of bacteria, than all the people who live on the planet. It is just an enormous question of scale. There are going to be interesting problems where you don't have control of the computing elements: how can you nevertheless solve big problems? These problems will involve data in as much as each of these computing elements will have data. I think challenges like that will be entirely new and possibly give entirely fresh perspectives on all aspects of computer science, including databases.

Among all your past research, do you have a favorite piece of work?

I have enjoyed so many things that I have done. I really like that some of the work has surprising applications. We have gotten requests for the tree matching software from sociologists, linguists, all kinds of people. The time series work that we've done has been downloaded by the usual suspects on Wall Street, but also has been used by people who are interested in "query by humming", in music, and also in astrophysics! So sometimes, the work has found a new life in unexpected ways. I have really enjoyed large data puzzles, where there is a lot of data, but also some combinatorial structure to make use

of. I don't really like problems without a combinatorial structure, that are just large for large's sake; I guess nobody does. If there is a nice puzzle, something clever that we can do, then I really like it.

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

The advice I have is almost the same as the advice that Dijkstra gave when we interviewed him for *Out of Their Minds*. He said, "Choose problems that are at the limit of your ability, not too hard, but not too easy either." Before you have tenure, you have to publish, but nevertheless, it is important to realize that you want to publish things that people will appreciate, that people will say, "Yeah, this was clever, and I couldn't have done that," or, "If I could have done it, I would have been really proud of it." My other advice is to go out in industry, or government, or wherever, and find people who have real problems. When the founder of my institute, Richard Courant, came to America, he would take his students to large engineering organizations and tell them to talk to the engineers. Not to the physicists and not to the mathematicians there, but to the engineers, because they knew what the real problems were. I really believe in that advice because it has led me to database tuning, to the time series work, to work with the biologists and write a funny little book called *Statistics Is Easy* – every time I have done something it is because something real I have seen has motivated me to do it. Sometimes it is not research in the hardest sense, e.g., *Statistics Is Easy* is just an easy introduction to resampling statistics. But it is nevertheless useful because most computer scientists don't like statistics. I think utility is a great motivator, and I think it is sometimes underestimated in academia.

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

That's a good question. I am curious about many things, and sometimes that leads me to try to do too much. I am working with biologists, I am doing a DNA computing project; I think it is great for students, because they gain a lot from it, but sometimes it doesn't end in, let's say, a publication. I'm not really too worried about that, but I think that would be the one thing I would change: I would maybe say *no* more.

How do the students get jobs if the work does not lead to publications? Even if you have tenure, your students will be looking for jobs, and if they don't have the publications they can't get the jobs.

I quickly evaluate whether a student is likely to go to academia. Many students don't really want to. If the student does, then we concentrate more on publications. Of course, sometimes I am wrong. I had a wonderful student named Yunyue Zhu who won a best paper award in time series, beautiful publication record, and then he went off to Wall Street because the money was so good! So what can I do?

Is he still on Wall Street?

Yeah! You see the headlines about Wall Street, and you think the world is coming to an end. But there were a lot of people on Wall Street who did extremely well, and all my students there are completely employed. It is a very funny difference between the headlines and what's actually going on.

Thank you very much for talking with me today!

It's a pleasure, thank you so much, Marianne.