Richard Hipp Speaks Out on SQLite

Marianne Winslett and Vanessa Braganholo

Richard Hipp

http://www.hwaci.com/drh/index.html

Welcome to ACM SIGMOD Record Series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today we are at the 2017 SIGMOD and PODS Conference in Chicago. I have here with me, Richard Hipp, who won the 2017 SIGMOD Systems Award and the 2005 Google O'Reilly Open Source Award for SQLite. Richard has his own consulting firm, Hwaci, and his Ph.D. is from Duke University.

So, Richard, welcome!

Thank you for having me here.

It's a pleasure. Can you take off your glasses for a moment and show off what you have there?

Oh, you mean, this?

This black eye! It must have been a real battle to win that SIGMOD Systems Award!

Oh, no, this is nothing. You should see the other guy. No, no! There was no fight or anything. This is actually a farming accident.

A farming accident?

I went to visit my parents last week. I was helping dad put up some new fencing on the farm, and I was cutting through the fence, and when I cut the last wire the whole thing sprung up, and a piece of wire hit me right there above the eye. By the grace of God, it missed my eyeball and didn't put an eye out. But I'm fine; everything is fine now. But this just goes to show that programmers should stick to programming computers and should not try to grow their own food. Leave farming to professionals!

All right, that's probably easy advice to swallow for our audience.

So, SQLite is the most widely deployed database engine in the world! Take that Oracle! SQLite is in just about every modern phone, PC browser, car, you name it, it might even be the single most widely deployed software component of any type! My first question: How did you happen to write such an insanely popular piece of software?

It was entirely by accident. I was working as a consultant doing some really interesting problems, and I wrote a product for my client and it had to pull the data off of Informix. And that worked great. Except for sometimes, the Informix server would go down for reasons that were out of my control, and then I'd pop-up a dialog box that says, "Can't access the database." And, of course, it was not my fault but I got the support call because it was my software painting the dialog box.

So, I thought, well, why can't I have a database that just reads directly off the disc? And I looked around and there were none available. I thought, "oh, I'll just write my own, how hard can that be?" Well, it turns out to be harder than you might think at first, but I didn't know that at the time. But we got it out there and I just put it out as open source. And before long, I started getting these phone calls from the big tech companies of the day, like Motorola and AOL, and, "Hey, can you support this?", and "Sure!" And it's like, wow, you can make money by supporting open source software? Who knew?

So, I built a small team and we've been doing that for about 10 or 12 years now. So, it was an accidental database, I didn't intend to take over the world.

The accidental database… You know you're in big company with that? Because Mike Stonebraker, when he first got started he had the exact same thought, he said, "How hard can this be?" They say that knowledge is power, but I think often ignorance is power.

Right. If I'd known how hard it would be I probably never would've have written it.

So, why is it so popular?

You know, I don't really know. I mean, I can guess. Well, one, it's really easy to use. I mean it comes as a single file source code, and so people just plop the single file of source down into their application and recompile, and they have a complete SQL stack.

The database is one file on disc. So, it's really easy to email to colleagues. It's fast. It's really easy to use. It's small, got a small footprint. That's why it's real popular on cell phones and things, because especially in the early days they were really interested in saving every byte of memory they could.

How small is it?

Compiled for size, it's less than a half megabyte.

Ooh, and how many lines of code, vaguely?

I think it's about 120,000 source lines of code; 200,000 if it includes the comments, all in one big file. We don't edit that one big file, okay, we actually have a bunch of little files that we developed and then there's a build process that stacks them all together.

But that makes it very easy to deploy because it's just one file that you drop in the middle of your application. And so, if you go look at popular applications like Chromium or Firefox or these open source things, you will find sqlite3.c in the tree which just gets linked in with the rest of their product and then they've got a complete stack. They don't have any external dependencies, and it just works.

What were the main technical challenges you faced in creating SQLite?

Well, a technical challenge is apart from the fact that I didn't know what I was doing? Well, so, SQLite doesn't have a server, you know, it's just a library. And so, you make a call into the library and then it returns, and there's not a thread or a process hanging around behind to take care of all the housekeeping details normally associated with a relational database, like concurrency control, rolling up LSM trees, or feeding the write-ahead log back into the database. These are all normally taken care of by some background thread. We don't have a background thread. So, we have to handle all of that in the foreground without causing unnecessary latency on the library calls. And we just don't have something persistent there to remember the state of things. And it causes you to have to think about the design of the engine very differently from when you're working on a client-server install.

Wouldn't it be great if SQL or something like that were extended in a way that you could have relations, you could have graphs, you could have arrays, you could have JSON, and it all worked together seamlessly?

Most developers are using SQLite as just a key-value store. Should we be trying to convince them to take more advantage of SQL?

I think so. I think that's the whole point. Well, I see so many developers that think that SQL is just the wire protocol for talking to the database engine. They don't really grasp that the whole concept of the declarative language, it's gonna do lots of really cool things for them. Once people get that, it makes a huge difference to them, they become so much more productive.

But, you think about it, the whole programming world has become so complicated. I mean, when I started in this business all these decades ago, my first computer had 4k of RAM, total. And that included the video RAM. And so, I could know everything that was happening in my computer, but that's no longer possible. And now we get people and we're trying to teach them how to program in four years? And there's just so much to learn and so much has to be left out, and I think that the whole declarative idea is being completely omitted. And people come out and, you know, it's easy to kind of think about things in a key-value store, that's a very familiar concept. But nobody's ever really taught the value of a declarative language and how much work that can really save. How many thousands of lines of code in your application you can save by specifying the join in SQL, rather than pulling all the information in and doing the join in your application, which is what we see a lot of people doing in actual applications. Yeah.

So, yeah, I think that about 90 percent of the use of SQLite is a key-value type thing. But that other 10 percent, people are really using it to the max, they're really putting these complex queries in there. And I see some of those queries sometimes and I think, wow, is my database computing this, that's amazing! But that's the way it's intended to be used.

If you had the ability to redesign SQLite from scratch, what would you change?

Mostly just piddly details. I mean, because SQLite is used in so many millions of applications, we cannot fix mistakes that I've made in the interface. So, for example, the database file format uses Big Endian numbers. And now all the processors are Little Endian. And so, we could save a few cycles here and there if the database stored everything as Little Endian.

And there are a few quirks in the language… If you say "integer primary key" that means one thing, but if you say "integer" and then put "primary key" into column name it works slightly differently and it's just wrong. But we can't change that now because there's a lot of databases out there that do make use of that distinction and that would break compatibility. And we're all about preserving compatibility to the hundreds of billions of instances that are out there already.

So, yeah, we could do a few tweaks like that to improve the performance, but I'm really kind of happy with the overall design and how it worked out. I wish I could claim that I had this brilliant insight and foresight and was able to predict that that's the way it was gonna work out (it was dumb luck). But it worked out well in the end.

You say that SQLite has aviation grade testing. What does that mean?

So, we follow a design process that's inspired by DO-178B, which is a Spec followed by the FAA, actually I think it's been superseded now with DO-178C, but I'm told it's not much different. And it's a Spec that the FAA requires for safety-critical systems, safety-critical software systems in the aircraft. It's a very detailed design Spec about the processes you do in developing the software. And the key point is that you have to test

it to 100 percent modified condition/decision coverage (100% MC/DC). Which, basically, means you have to test the machine level such that every branch instruction has been taken and falls through at least once. It's an incredible amount of testing.

This started because years ago I had this idea that, oh, I'll come up with this test suite and then I'll sell it to people and make money. That didn't really play out. But what we found is when we did this and spent a year developing all these tests and, of course, tons of time since then maintaining them, is that the number of bugs just dropped dramatically. And it will amaze you how many bugs pop up when your software is deployed on two billion cell phones. And, yeah, I used to think that I could write bug-free software and it got put on cell phones and then, no…

But once we got that and got this aviation grade testing in place, the number of bugs just dropped to a trickle. Now we still do have bugs but the aviation grade testing allows us to move fast, which is important because in this business you either move fast or you're disrupted. So, we're able to make major changes to the structure of the code that we deliver and be confident that we're not breaking things because we had these intense tests. Probably half the time we spend is actually writing new tests, we're constantly writing new tests. And over the 17-year history, we have amassed a huge suite of tests which we run constantly.

Other database engines don't do this; don't have this level of testing. But they're still high quality, I mean, I noticed in particular, PostgreSQL is a very high-quality database engine, they don't have many bugs. I went to the PostgreSQL and ask them "how do you prevent the bugs"? We talked about this for a while. What I came away with was they've got a very elaborate peer review process, and if they've got code that has worked for 10 years they just don't mess with it, leave it alone, it works. Whereas we change our code fearlessly, and we have a much smaller team and we don't have the peer review process. So, that's the basic difference. People hear aviation grade testing, that means it must be bug-free. Not really. It's low bug, but there are bugs. The key benefit is that it allows us to move fast and aggressively and make big changes to the code without fear of breaking things.

Now, why do you want to make big changes to the code if you've already written something that clearly is working for almost everybody?

Well, because, you know, you can always make improvements to the query planner. The query planner is an AI, and so there's no perfect solution. And so, no matter what we do there's gonna be somebody come along and find some query that it comes up with a bad plan for. And then a lot of times we have to make pretty radical changes to the query planner in order to come up with a good query plan for some bizarre bit of SQL. Of course, the first question we always ask is, why did you want to do this? Who would ever issue such a query? And it's often machine-generated queries.

It was an accidental database, I didn't intend to take over the world.

So, it occurs to me that one factor in your success is the fact that only 10 percent of the billions of installations are doing the SQL queries. Because if all those key-value store people were actually making full use of the abilities of the SQL language, heaven knows what kinds of strange queries they'd be asking…

Well, I'd suppose but, no, 10 percent of a billion is still a lot!

It is.

And the people who are doing the elaborate queries are companies that use it really heavily. Customers! I wish I could name them for you.

That's fine. We don't wanna know, we don't wanna know. So, if commercial database engines don't undergo nearly that amount of testing, why do you work so hard at testing an open source product that you give away for free?

Well, that's a good question. You know, the intense testing allows us to keep a really small team. The product is free, and so, we make all our money from support. And we've been following of the open source world and people are saying "you can't make money selling support." And they're basically right, you can't make a lot of money. So, a typical start-up would be 30 to 50 guys in an office building in San Francisco. And you cannot make enough money selling support to support that operation. But we only have three developers, including me. And we're a distributed company, we don't have office space, everybody works from home. We keep our overhead really low. And we only have three people. And so, with only three people this high level of testing allows us to produce a quality product without having a lot of eyeballs on it.

Wait, it almost suggests that a giant company could also get by with three people on some major product. If they did mega testing.

Possibly so. You know, the DO-178B Spec is very detailed. And what I've seen is a lot of companies trying to implement it and they get very legalistic about it. And, I mean, it's a really great thing if you do it well, but if you don't implement it carefully it can just become bureaucratic overhead. And then it actually multiplies the number of people rather than reducing them.

Okay. Got it.

It's a tricky thing; it's a knife edge there.

People use SQLite in some places where you really don't want a hacker to be able to get at the data. So, beyond aviation grade testing, are there other considerations you give to the security?

Yes, we do. Well, so the MC/DC testing is great for verifying that sensible queries that normal programmers would write actually work and give the correct answer. But to prevent attacks we use fuzz testing. And there are a number of great products out there that do this for us.

I think a lot of our audience might not know what fuzz testing is.

Fuzz testing is when you take a library or a product and you start pounding it with just seemingly random inputs and trying to get it to break. You're not trying to see if it gets the right or wrong answer, you just wanna break it, you wanna get a segmentation fault error or something like that. And that's an opportunity to break into the system.

And so, there are some researchers at Google, and what they've done is they have these fuzz testers where they start with a random input. But they also instrument the source code, or the object code. And they monitor the path through the object code that that test case took and it finds new behaviors, it learns from that. When it finds a new path it says, "oh, I'm gonna keep changing that mutation" and it finds new paths to the codes. It's a very powerful thing. It's only been out for two to three years. And it will find an amazing number of the bugs, even in very well tested software.

So, an example of this is they took a jpeg library and they started it with an empty file and started fuzzing it. And the fuzzer actually discovered valid .jpeg files. That's the power of the system. So, we feed these sorts of things into SQLite. Because it's a library, it's very easy to do this and we can give it thousands and thousands of queries per second, fuzzing them, trying to find bugs.

And when we first started doing that we did find a dozen or two dozen ways of crashing it. But since then, and we continue to test, we've fixed those and we haven't had any more problems.

So, SQLite can be safely used as, for example, a file format transferring data from across the internet. And you can receive a SQLite file from an untrusted source and bring it up and be confident that it's not going to –

Be a trojan horse…

Exactly.

So, what's the name of the Google tool for fuzz testing?

The first one was American Fuzzy Lop. And then the following one to that was OSS-Fuzz.

(Laughs) I laughed because the first name American Fuzzy Lop definitely sounds like the name of a kind of funny rabbit.

That's where he got the name. I won't try to pronounce the developer's name, it's Michael[1] and he's Polish. I will totally mispronounce his name so I won't try to pronounce it. But he wrote it and he picked that name because of the pun.

So, vaguely, how long would it take, when you've made some changes to your source, to run through your entire set of tests? Just vaguely.

Well, on one platform, because we run it on multiple platforms.

No, just one.

If it's a fast work station we can, running on multiple cores, we can do a complete set of tests in about 12 hours. But then we also run on multiple platforms, including some slow ones, like phones and that sort of thing. Some antique hardware, like ancient Mac Books that are still using Power PC processors, so we can test that it runs on Big Endian as well as the Low Endian. And so, it normally takes us about three or four days to do a complete test. But we can do full coverage testing, the 100 percent MC/DC testing in about three minutes. And so, after any change, we always do that. But then we have these long soak tests that do a lot of additional testing, and that's what takes a large amount of time.

[1] Editor's note: His name is Michal Zalewski.

Why did you choose the license for SQLite to be public domain rather than BSD or MIT?

Well, of course, technically, public domain is not a license, but the absence of one. Actually, SQLite Version 1 in 2000 used GDBM[2] as the storage engine. And that's a key-value storage thing. And it's GPL, and so SQLite Version 1 was GPL, it had to be because it was linking against the GPL library.

But GDBM is only key-value, I can't do range queries with it. Then I said, "I'm gonna write my own B-tree layer". And so, I wrote my own and at that point it became all my code and I thought, well, what license shall I do? And I thought, well, you know, BSD, MIT. And I thought, well, what's the point, why not just say it is public domain? And so, I put it out there as public domain. People can do whatever they want, I have disavowed all copywrite to it.

I would love to see people spend more time researching query languages as opposed to storage engines

And that was cool and that worked great. What I later learned is that the ability to put something in the public domain is kind of unique to countries that follow British common law. Other countries don't allow people to disavow their rights to the code. And so, it's a problem in that sense.

But it's to tradition and we've stuck with it. It's worked well and people understand it, and it makes people confident that they can use it their products and not have to worry about legal repercussions. Some companies still do worry about that because, I mean, anybody can grab some piece of code from somebody and put it out in the internet and say, oh, this public domain, you know.

So, another way that we do make money is we actually sell licenses for it. Actually, it's not a license, it's a warranty of title. It's an official document that says we do have the right to place this in the public domain and we will accept legal responsibility if anybody comes forward and accuses you of stealing it. And companies send us money for this piece of paper. And that's what we use to fund the development.

Okay. Anything to keep those lawyers satisfied. And if you were starting again today, would you still go the public domain route?

It's hard to say. I mean, it's fun to be unique in that, but practically speaking the two-clause BSD license or MIT license, accomplishes the same thing, and doesn't run into the problems that are in non-British common law jurisdictions. So, that would probably be a better choice.

After they graduate, most computer science grad students go to the university, or a big company or a start-up. So, what led you to be a consultant instead?

Oh, that's a long story. Because I've temperamentally ill-suited for working in the –

Corporate life?

Yeah. I don't play well with others.

Oh! The small team is just a matter of preference!

It is. I really like working for myself. And one reason we haven't grown to a big team is that there's no way in the world that I could manage a big team. And the people I have working for me are great and we get along well, and I think trying to manage 30 to 50 people would really be a disaster for me. I really enjoy working for myself.

Back in 1992, when I took my Ph.D., there were 500 applicants for any tenure track position. And I decided "I'm not gonna get in that rat race." And so, I started the consulting thing and it's worked out really well. I get to set my own hours, you know because when you work for yourself you can work any 80 hours of the week you want, work from home, work in projects that I wanna work on. And it's really worked out well. It's been a dream job. All of us on the team have really enjoyed working on SQLite.

Yeah, it sounds like if you graduated in a year when it was a seller's market, buyer's market, I'm not sure which direction, you might've taken a completely different path!

Well, I may have. But I guess it's providential that it worked out as well as it did.

Yeah! For everybody.

[2] https://www.gnu.org.ua/software/gdbm/gdbm.html

Yeah. I'm very happy.

But for someone who's graduating today, what considerations do you think should make them choose a consulting path?

Well, you get to choose your own destiny. You don't have to deal with the big company or the big university bureaucracy and the politics that comes with that. Maybe I'm sort of an urban prepper you might think of, in the sense that I wanna live off grid. Not really! Not really! It's just that I wanna do my own thing. No, I actually enjoy very much having the conveniences of modern city life. I don't wanna go off grid. But the concept is similar in that I want to decide for myself what I'm working on. I want the freedom to move around from lots of different customers.

You know, one really great thing about working in this is I've had the opportunity to visit so many different businesses and see the really different cultures there are in all these different places all over the world. And I can't really explain it unless you've experienced it, but you go to one company and see how they operate and how many people interact, and you go to a different company and have a totally different culture there. And then you step back and look and that culture kind of comes out in their products, and you can see it after you've visited their engineering facilities.

Interesting.

It's very interesting. And I don't know of any way of describing that other than just say go there and see for yourself and then you'll understand.

So, even inside the banks in Charlotte where you live, all those big banks, can you see the difference in culture in the products that come out of these giant banks?

A little bit. I don't have a whole lot of insight into the banks there in Charlotte. I'm thinking of like visiting, well, companies that you know well like Facebook and Google and Apple.

Oh, computer companies?

Yes. Motorola, AOL, these sorts of things. I get opportunities to go and visit and talk, and give talks there, and meet with the people and see their environment. And it's really exciting to be able to see all of these differences, coming in, not have to be a part of that company, but get a glimpse into their culture and see how they're all very different. You'd think that a bunch of companies all in the Bay Area would be very similar, but they're not, they're really very different. And when you travel to other countries and do this, it gets really, really different.

Yes, definitely. So, you don't see an evolution toward a single culture at computer companies? There's no convergence, it sounds like. If anything, it's divergence.

No, everybody pretty much copied Google's idea of giving the people free food, okay. That was a big win, apparently, and everybody copied that. So, there are some ideas that get around. And so, maybe you're right, maybe there was a lot more diversity a decade ago than there is today. But still, even today you can see a big difference in the different companies.

> ***SQL is the language that everybody loves to hate.***

Cool. Among all your past accomplishments, do you have a favorite piece of work other than SQLite?

I'm kind of proud of the version control system that we wrote specifically for SQLite called Fossil[3]. So, SQLite started out using CVS because back then that's what everybody used. But CVS is great if you had to use what came before CVS, you know. Some people bad mouth CVS, I know that those people never had to use what came before. But it has its limitations, and so we wanted to go with something better, and the other options weren't really making it for me. So, I wrote my own and called it Fossil.

And it's interesting in that Fossil stores all of its data in an SQLite database, and SQLite is controlled by Fossil, so we have this recursion here – if something breaks the whole project sort of collapses. It's a house of cards. But it works really well and it's been really flattering that thousands of other people have picked it up and started using it. So, it's a distributed version control system like Git. Nowhere near as many users, orders of magnitude fewer. But those who do use it are really enthusiastic about it.

And it keeps me in touch with the application development side of things. Because if you're developing just the database engine for so long, you get heads down and you don't see what's happening. But I can go out and work on Fossil and it's helping so many people, and then I also get to work on an actual application that uses SQLite and understand the pain of

[3]https://fossil-scm.org/home/doc/trunk/www/index.wiki

the users who have to deal with the interfaces that I design. And that's very good.

Eating your own dog food.

Yes. We're very much into dog food in SQLite.

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

Oh, I've got a long list. But my #1 thing right now I think would be a new version control system which the working name is Fit, it's a combination of Fossil with Git. Uses the Fossil user interface but it uses the low-level file format of Git. So that then you can work with Fossil's interface but push and pull to legacy Git users. And I think that would be huge.

Why? What's better about the Fossil interface?

Well, it's not the perfect interface, but all the users say "oh, the Fossil interface is great, it's so wonderful, I hate having to use Git." But then they're compelled to use Git because everybody else in the world is and they wanna collaborate. So, I think that by merging these two producing Fit, Fossil plus Git, that would be a really big win for a lot of people.

Sounds like it.

It sounds like a lot of work, too.

Maybe… Do you have any words of advice for fledgling or mid-career database people?

I would love to see people spend more time researching query languages as opposed to storage engines. I mean, storage engines are a very important thing, we need to work on that, but it's like query languages are ignored.

Do you mean like changing SQL or do you mean like…

Enhancing SQL. SQL is the language that everybody loves to hate. And of course, I guess, everybody in the database world has at one point or another tried to come up with a better SQL. I know I've tried multiple times myself with indifferent results.

But the relational model is great and it'll represent anything, but there're some problems that just work out better with like a graph model or an array, or something like that, or JSON. Wouldn't it be great if SQL or something like that were extended in a way that you could have relations, you could have graphs, you could have arrays, you could have JSON, and it all worked together seamlessly? And that would be really, really amazing.

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

Well, I think we'd all like to be smarter, right? The ability to communicate better. I don't know. I've had such a blessed career, truly. I mean, I fell into this, I didn't plan to be a database guy, that was never in my career plans it just happened, but it's been an enormous amount of fun. And if I were to design it I'd mess it up. So, I'm just gonna go with what we've got.

Sounds good. Thank you very much for talking with us today.

Thank you for your time.