DOI:10.1145/2869958

It turns out riding across America is more than a handy metaphor for building system software.

BY MICHAEL STONEBRAKER

The Land Sharks Are on the Squawk Box

KENNEBAGO, ME, SUMMER 1993. The "Land Sharks" are on the squawk box, Illustra (the company commercializing Postgres) is down to fumes, and I am on a conference call with the investors to try to get more money. The only problem is I am in Maine at my brother's fishing cabin for a family event while the investors are on a speakerphone (the squawk box) in California. There are eight of us in cramped quarters, and I am camped out in the bathroom trying to negotiate a deal. The conversation is depressingly familiar. They say more-money-lower-price; I say less-money-higher-price. We ultimately reach a verbal handshake, and Illustra will live to fight another day.

Negotiating with the sharks is always depressing. They are superb at driving a hard bargain; after all, that is what they do all day. I feel like a babe in the woods by comparison.

This article interleaves two stories (see Figure 1). The first is a cross-country bike ride my wife Beth and I took during the summer of 1988; the second is the design, construction, and commercialization of Postgres, which occurred over a 12-year period, from the mid-1980s to the mid-1990s. After telling both stories, I will draw a series of observations and conclusions.

Off to a Good Start

Anacortes, WA, June 3, 1988. Our car is packed to the gills, and the four of us (Beth; our 18-month-old daughter Leslie; Mary Anne, our driver and babysitter; and me) are squished in. It has been a stressful day. On the roof is the cause of it all—our brand-new tandem bicycle. We spent the afternoon in Seattle bike shops getting it repaired. On the way up from the Bay Area, Mary Anne drove into a parking structure lower than the height of the car plus the bike. Thankfully, the damage is repaired, and we are all set to go, if a bit frazzled. Tomorrow morning, Beth and I will start riding east up the North Cascades Scenic Highway; our destination, some 3,500 miles away, is Boston, MA. We have therefore christened our bike "Boston Bound."

It does not faze us that we have been on a tandem bike exactly once, nor that we have never been on a bike trip longer than five days. The fact we have never climbed mountains like the ones directly in front of us is equally undaunting. Beth and I are in high spirits; we are starting a great adventure.

Berkeley, CA, 1984. We have been working on Ingres for a decade. First, we built an academic prototype, then

» key insights

- Explained is the motivation behind Postgres design decisions, as are "speedbumps" encountered.
- Riding a bicycle across America and building a computer software system are both long and difficult affairs, constantly testing personal fortitude along the way.
- Serendipity played a major role in both endeavors.

made it fully functional, and then started a commercial company. However, Ingres Corporation, which started with our open source code base four years ago in 1980, has made dramatic progress, and its code is now vastly superior to the academic version. It does not make any sense to continue to do prototyping on our software. It is a painful decision to push the code off a cliff, but at that point a new DBMS is born. So what will Postgres be?

One thing is clear: Postgres will push the envelope on data types. By now I have read a dozen papers of the form: "The relational model is great, so I tried it on [pick a vertical application]. I found it did not work, and to fix the situation, I propose we add [some new idea] to the relational model."

Some chosen verticals were geographic information systems (GISs), computer-aided design (CAD), and library information systems. It was pretty clear to me that the clean, simple relational model would turn into a complete mess if we added random functionality in this fashion. One could think of this as "death by 100 warts."

The basic problem was the existing relational systems—specifically Ingres and System R—were designed with business data processing users in mind. After all, that was the major DBMS market at the time, and both collections of developers were trying to do better than the existing competition, namely IMS and Codasyl, on this popular use case. It never occurred to us to look at other markets, so RDBMSs were not good at them. However, a research group at the University of California at Berkeley, headed by Professor Pravin Varaiya, built a GIS on top of Ingres, and we saw firsthand how painful it was. Simulating points, lines, polygons, and line groups on top of the floats, integers, and strings in Ingres was not pretty.

It was clear to me that one had to support data types appropriate to an application and that required user-defined data types. This idea had been investigated earlier by the programming language community in systems like EL1, so all I had to do was apply it to the relational model. For example, consider the following SQL update to a salary, stored as an integer

```
Update Employee set (salary
= salary + 1000) where name =
'George'
```

To process it, one must convert the character string `1000` to an integer using the library function string-to-integer and then call the integer + routine from the C library. To support this command with a new type, say, `foobar`, one must merely add two functions, `foobar-plus` and `string-to-foobar`, and then call them at the appropriate times. It was straightforward to add a new DBMS command, `ADDTYPE`, with the name of the new data type and conversion routines back and forth to ASCII. For each desired operator on this new type, one could add the name of the operator and the code to call to apply it.

The devil is, of course, always in the details. One has to be able to index the new data type using B-trees or hashing. Indexes require the notion of less-than and equality. Moreover, one needs commutativity and associativity rules to decide how the new type can be used with other types. Lastly, one must also deal with predicates of the form:

```
not salary < 100
```

This is legal SQL, and every DBMS will flip it to

```
salary ≥ 100
```

So one must define a negator for every operator, so this optimization is possible.

We had prototyped this functionality in Ingres,[8] and it appeared to work, so the notion of abstract data types (ADTs) would clearly be a cornerstone of Postgres.

Winthrop, WA, Day 3. My legs are throbbing as I lay on the bed in our motel room. In fact, I am sore from the hips down but elated. We have been riding since 5 A.M. this morning; telephone pole by telephone pole we struggled uphill for 50 miles. Along the way, we rose 5,000 feet into the Cascades,

Anacortes, WA: Day 1 - June 4, 1988

ALL PHOTOS BY MARY ANNE CASE

Figure 1. The two timelines: cross-country bike ride and Illustra/Postgres development.

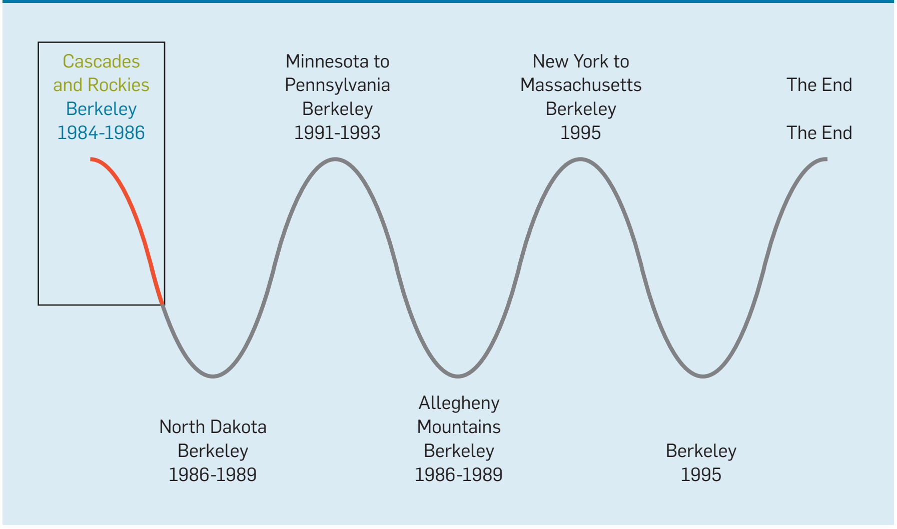
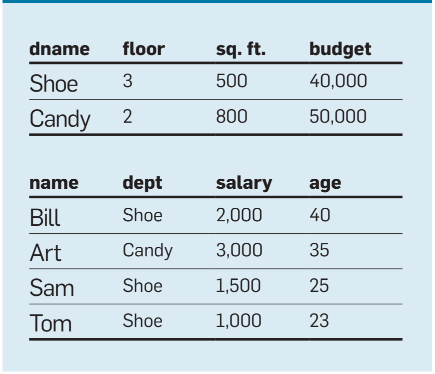


Figure 2. Correlated data illustrating why data users need referential integrity.



dname	floor	sq. ft.	budget
Shoe	3	500	40,000
Candy	2	800	50,000

name	dept	salary	age
Bill	Shoe	2,000	40
Art	Candy	3,000	35
Sam	Shoe	1,500	25
Tom	Shoe	1,000	23

putting on every piece of clothing we brought with us. Even so, we were not prepared for the snowstorm near the top of the pass. Cold, wet, and tired, we finally arrived at the top of the aptly named Rainy Pass. After a brief downhill, we climbed another 1,000 feet to the top of Washington Pass. Then it was glorious descent into Winthrop. I am now exhausted but in great spirits; there are many more passes to climb, but we are over the first two. We have proved we can do the mountains.

Berkeley, CA, 1985–1986. Chris Date wrote a pioneering paper[1] on referential integrity in 1981 in which he defined the concept and specified rules for enforcing it. Basically, if one has a table

```
Employee (name, salary, dept,
age) with primary key "name"
```

and a second table

```
Dept (dname, floor) with a
primary key "dname"
```

then the attribute dept in `Employee` is a foreign key; that is, it references a primary key in another table; an example of these two tables is shown in Figure 2. In this case, what happens if one deletes a department from the `dept` table?

For example, deleting the candy department will leave a dangling reference in the Employee table for everybody who works in the now-deleted department. Date identified six cases concerning what to do with insertions and deletions, all of which can be specified by a fairly primitive if-then rule system. Having looked at programs in Prolog and R1, I was very leery of this approach. Looking at any rule program with more than 10 statements, it is very difficult to figure out what it does. Moreover, such rules are procedural, and one can get all kinds of weird behavior depending on the order in which rules are invoked. For example, consider the following two (somewhat facetious) rules:

```
If Employee.name = 'George'
Then set Employee.dept = 'shoe'
If Employee.salary > 1000 and
Employee.dept = 'candy'
Then set Employee.salary = 1000
```

Consider an update that moves `George` from the shoe department to the candy department and updates his salary to `2000`. Depending on the order the two rules are processed, one will get different final answers. Notably, if the rules are executed in the order here, then George will ultimately have a salary of `2000`; if the rule order is reversed, then his ending salary will be `1000`. Having order-dependent rule semantics is pretty awful.

A fundamental tenet of the relational model is the order of evaluation of a query, including the order in which records are accessed, is up to the system. Hence, one should always give the same final result, regardless of the query plan chosen for execution. As one can imagine, it is trivial to construct collections of rules that give different answers for different query plans—obviously undesirable system behavior.

I spent many hours over a couple of years looking for something else. Ultimately, my preferred approach was to add a keyword `always` to the query language. Hence, any utterance in the query language should have the semantics that it appears to be continually running. For example, if Mike must have the same salary as Sam, then the following `always` command will do the trick

```
Always update Employee, E
set salary = E.salary
where Employee.name = 'Mike'
and E.name = 'Sam'
```

Whenever Mike receives a salary adjustment, this command will kick in and reset his salary to that of Sam. Whenever Sam gets a raise, it will be propagated to Mike. Postgres would have this `always` command and avoid (some of) the ugliness of an if-then rules system. This was great news; Postgres would try something different that has the possibility of working.

Marias Pass, MT, Day 15. I cannot believe it. We round a corner and see the sign for the top of the pass. We are at the Continental Divide! The endless climbs in the Cascades and the Rockies are behind us, and we can see the Great Plains stretching out in front of us. It is now downhill to Chicago! To celebrate this milestone, we pour a small vial of Pacific Ocean water we have been carrying since Anacortes to the east side of the pass where it will ultimately flow into the Gulf of Mexico.

Berkeley, CA, 1986. My experience with Ingres convinced me a database log for recovery purposes is tedious and difficult to code. In fact, the gold standard specification is in C. Mohan

Marias Pass, MT: Day 15

et al.[3] Moreover, a DBMS is really two DBMSs, one managing the database as we know it and a second one managing the log, as in Figure 3. The log is the actual system of record, since the contents of the DBMS can be lost. The idea we explored in Postgres was to support time travel. Instead of updating a data record in place and then writing both the new contents and the old contents into the log, could we leave the old record alone and write a second record with the new contents in the actual database? That way the log would be incorporated into the normal database and no separate log processing would be required, as in Figure 4. A side benefit of this architecture is the ability to support time travel, since old records are readily queryable in the database. Lastly, standard accounting systems use no overwrite in their approach to record keeping, so Postgres would be compatible with this tactic.

At a high level, Postgres would make contributions in three areas: an ADT system, a clean rules system based on the `always` command, and a time-travel storage system. Much of this functionality is described in Stonebraker and Rowe.[6,7] For more information on the scope of Postgres, one can consult the video recording of the colloquium celebrating my 70th birthday.[2] We were off and running with an interesting technical plan.

First Speedbumps

Drake, ND, Day 26. We are really depressed. North Dakota is bleak. The last few days have been the following monotony:

See the grain elevator ahead that signifies the next town
Ride for an hour toward the elevator
Pass through the town in a few minutes
See the next grain elevator …

However, it is not the absence of trees (we joke the state tree of North Dakota is the telephone pole) and the bleak landscape that is killing us. Normally, one can simply sit up straight in the saddle and be blown across the state by the prevailing winds, which are typically howling from west to east. They are howling all right, but the weather this summer is atypical. We are experiencing gale-force winds blowing east to west, straight in our faces. While we are expecting to be blown along at 17–18 miles per hour, we are struggling hard to make 7. We made only 51 miles today and are exhausted. Our destination was Harvey, still 25 miles away, and we are not going to make it. More ominously, the tree line (and Minnesota border) is still 250 miles away, and we are not sure how we will get there. It is all we can do to refuse a ride from a driver in a pickup truck offering to transport us down the road to the next town.

The food is also becoming problematic. Breakfast is dependable. We find a town, then look for the café (often the only one) with the most pickup trucks. We eat from the standard menu found in all such restaurants. However, dinner is getting really boring. There is a standard menu of fried fare; we yearn for pasta and salad, but it is never on the menu.

We have established a routine. It is in the 80s or 90s Fahrenheit every day, so Beth and I get on the road by 5 A.M. Mary Anne and Leslie get up much later; they hang around the motel, then pass us on the road going on to the town where we will spend the night. When we arrive at the new motel, one of us relieves Mary Anne while the other tries to find someplace with food we are willing to eat. Although we have camping equipment with us, the thought of an air mattress after a hard day on the road is not appealing. In fact, we never camp. Leslie has happily accommodated to this routine, and one of her favorite words, at 18-months old, is "ice machine." Our goal is 80 miles a day in the flats and 60 miles a day in the mountains. We ride six days per week.

Berkeley, CA, 1986. I had a conversation with an Ingres customer shortly

Figure 3. Traditional DBMS crash recovery.

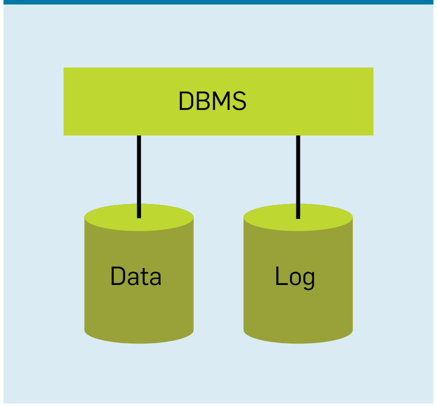
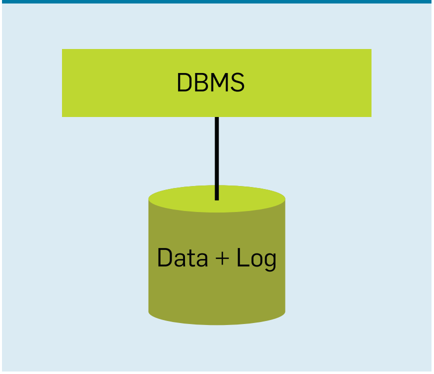


Figure 4. Postgres picture: No overwrite.



Drake, ND: Day 26

after he implemented `date` and `time` as a new data type (according to the American National Standards Institute specification). He said, "You implemented this new data type incorrectly." In effect, he wanted a different notion of time than what was supported by the standard Gregorian calendar. More precisely, he calculated interest on Wall Street-type financial bonds, which give the owner the same amount of interest, regardless of how long a month is. That is, he wanted a notion of `bond time` in which March 15 minus February 15 is always 30 days, and each year is divided into 30-day months. Operationally, he merely wanted to overload temporal subtraction with his own notion. This was impossible in Ingres, of course, but easy to do in Postgres. It was a validation that our ADTs are a good idea.

Berkeley, CA, 1986. My partner, the "Wine Connoisseur," and I have had a running discussion for nearly a year about the Postgres data model. Consider the `Employee-Dept` database noted earlier. An obvious query is to join the two tables, to, say, find the names and floor number of employees, as noted in this SQL command:

```
Select E.name, D.floor
From Employee E, Dept D
Where E.dept = D.dname
```

In a programming language, this task would be coded procedurally as something like (see code section 1).

A programmer codes an algorithm to find the desired result. In contrast, one tenet of the relational model is programmers should state what they want without having to code a search algorithm. That job falls to the query optimizer, which must decide (at scale) whether to iterate over `Employee` first or over `Dept` or to hash both tables on the join key or sort both tables for a merge or …

My Ingres experience convinced me optimizers are really difficult, and the brain surgeon in any database company is almost certainly the optimizer specialist. Now we were considering extending the relational model to support more complex types. In its most general form, we could consider a column whose fields were pointers to arrays of structures of … I could not wrap my brain around designing a query optimizer for something this complex. On the other hand, what should we discard? In the end, The Wine Connoisseur and I are depressed as we choose a design point with rudimentary complex objects. There is still a lot of code to support the notion we select.

Berkeley, CA, 1987. The design of time travel in Postgres is in Stonebraker.[5] Although this is an elegant construct in theory, making it perform well in practice is tricky. The basic problem is the two databases in the traditional architecture of Figure 3 are optimized very differently. The data is "read-optimized" so queries are fast, while the log is "write-optimized" so one can commit transactions rapidly. Postgres must try to accomplish both objectives in a single store; for example, if 10 records are updated in a transaction, then Postgres must force to disk all the pages on which these records occurred at commit time. Otherwise, the DBMS can develop "amnesia," a complete no-no. A traditional log will group all the log records on a small collection of pages, while the data records remain read-optimized. Since we are combining both constructs into one storage structure, we have to address a tricky record placement problem to try to achieve both objectives, and our initial implementation is not very good. We spend a lot of time trying to fix this subsystem.

Berkeley, CA, 1987. The Wine Connoisseur and I had written Ingres in C and did not want to use it again. That sounded too much like déjà vu. However, C++ was not mature enough, and other language processors did not run on Unix. By this time, any thought of changing operating systems away from Unix was not an option; all the Berkeley students were being trained on Unix, and it was quickly becoming the universal academic operating system. So we elected to drink the artificial intelligence Kool-Aid and started writing Postgres in Lisp.

Once we had a rudimentary version of Postgres running, we saw what a disastrous performance mistake this was—at least one-order-of-magnitude performance penalty on absolutely everything. We immediately tossed portions of the code base off the cliff

Code section 1.

```
For E in Employee {
        For D in Dept {
                If (E.dept = D.dname) then add-to-result;
        }
}
```

and converted everything else to C. We were back to déjà vu (coding in C), having lost a bunch of time, but at least we had learned an important lesson: Do not jump into unknown water without dipping your toe in first. This was the first of several major code rewrites.

Berkeley, CA, 1988. Unfortunately, I could not figure out a way to make our `always` command general enough to at least cover Chris Date's six referential integrity cases. After months of trying, I gave up, and we decided to return to a more conventional rule system. More code over the cliff, and more new functionality to write.

In summary, for several years we struggled to make good on the original Postgres ideas. I remember this time as a long "slog through the swamp."

The endless climbs in the Cascades and the Rockies are behind us, and we can see the Great Plains stretching out in front of us.

Another High

Carrington, ND, the next afternoon. It is really hot, and I am dead tired. I am on "Leslie duty," and after walking though town, we are encamped in the ubiquitous (and air-conditioned) local Dairy Queen. I am watching Leslie slurp down a soft serve, feeling like "God is on our side," as serendipity has intervened in a big way today. No, the wind is still blowing at gale force from east to west. Serendipity came in the form of my brother. He has come from Maine to ride with us for a week. Mary Anne picked him and his bicycle up at the Minot airport yesterday afternoon. He is fresh and a very, very strong rider. He offers to break the wind for us, like you see in bicycle races. With some on-the-job training (and a couple of excursions into the wheat fields when we hit his rear wheel), Beth and I figure out how to ride six inches behind his rear wheel. With us trying to stay synchronized with a faster-slower-faster dialog, we rode 79 miles today. It is now clear we are "over the hump" and will get out of North Dakota, a few inches behind my brother's wheel, if necessary.

Battle Lake, MN, July 4, 1988, Day 30. We are resting today and attending the annual 4th of July parade in this small town. It is quite an experience—the local band, clowns giving out candy, which Leslie happily takes, and Shriners in their little cars. It is a slice of Americana I will never forget. Rural America has taken very good care of us, whether by giving our bike a wide berth when passing, willingly cashing our travelers checks, or alerting us to road hazards and detours.

Berkeley, CA, 1992. In my experience, the only way to really make a difference in the DBMS arena is to get your ideas into the commercial marketplace. In theory, one could approach the DBMS companies and try to convince them to adopt something new. In fact, there was an obvious "friendly" one—Ingres Corporation—although it had its own priorities at the time.

I have rarely seen technology transfer happen in this fashion. There is a wonderful book by Harvard Business School professor Clayton Christiansen called *The Innovators Dilemma*. His thesis is technology disruptions are very challenging for the incumbents. Specifically, it is very difficult for established vendors with old technology to morph to a new approach without losing their customer base. Hence, disruptive ideas do not usually find a receptive audience among the established vendors, and launching a startup to prove one's ideas is the preferred option.

By mid-1992 I had ended my association with Ingres and a sufficient amount of time had passed that I was free of my non-compete agreement with the company. I was ready to start a commercial Postgres company and contacted my friend the "Tall Shark." He readily agreed to be involved. What followed was a somewhat torturous negotiation of terms with the "Head Land Shark," with me getting on-the-job training in the terms and conditions of a financing contract. Finally, I understood what I was being asked to sign. It was a difficult time, and I changed my mind more than once. In the end, we had a deal, and Postgres had $1 million in venture capital to get going.

Right away two stars from the academic Ingres team—"Quiet" and "EMP1"—moved over to help. They were joined shortly thereafter by "Triple Rock," and we had a core implementation team. I also reached out to "Mom" and her husband, the "Short One," who also jumped on board, and we were off and running, with the Tall Shark acting as interim CEO. Our initial jobs were to whip the research code line into commercial shape, convert the query language from QUEL to SQL, write documentation, fix bugs, and clean up the "cruft" all over the system.

Emeryville, CA, 1993. After a couple of naming gaffes, we chose Illustra, and our goal was to find customers willing to use (and hopefully pay for) a system from a startup. We had to find a compelling vertical market, and the one we chose to focus on was geographic data. Triple Rock wrote a collection of abstract data types for points, lines, and polygons with the appropriate functions (such as distance from a point to a line).

After an infusion of capital from new investors, including the "Entrepreneur-Turned-Shark," we again ran out of money, prompting the phone call from Kennebago noted earlier. Soon thereafter, we were fortunate to be able to hire the "Voice-of-Experience" as the real CEO, and he recruited "Smooth" to be VP of sales, complementing "Uptone," who was previously hired to run marketing. We had a real company with a well-functioning engineering team and world-class executives. The future was looking up.

Luddington, MI, Day 38. We walk Boston Bound off the Lake Michigan ferry and start riding southeast. The endless Upper Midwest is behind us; it is now less than 1,000 miles to Boston! Somehow it is reassuring that we have no more more water to cross. We are feeling good. It is beginning to look like we might make it.

The High Does Not Last

Ellicottville, NY, Day 49. Today was a very bad day. Our first problem occurred while I was walking down the stairs of the hotel in Corry, PA, in my bicycle cleats. I slipped on the marble floor and wrenched my knee. Today, we had only three good legs pushing Boston Bound along. However, the bigger problem is we hit the Alleghany Mountains. Wisconsin, Michigan, and Ohio are flat. That easy riding is over, and our bicycle maps are sending us up and then down the same 500 feet over and over again. Also, road planners around here do not seem to believe in switchbacks; we shift into the lowest of our 21 gears to get up some of these hills, and it is exhausting work. We are not, as you can imagine, in a good mood. While Beth is putting Leslie to bed, I ask the innkeeper in Ellicottville a simple question, "How do we get to Albany, NY, without climbing all these hills?"

Emeryville, CA, 1993. Out of nowhere comes our first marketing challenge. It was clear our "sweet spot" was any application that could be accelerated through ADTs. We would have an unfair advantage over any other DBMS whenever this was true. However, we faced a Catch-22 situation. After a few "lighthouse" customers, the more cautious ones clearly said they wanted GIS functionality from the major GIS vendors (such as ArcInfo and MapInfo). We needed to recruit application companies in specific vertical markets and convince them to restructure the inner core of their software into ADTs—not a trivial task. The application vendors naturally said, "Help me understand why we should engage with you in this joint project." Put more bluntly, "How many customers do you have and how much money can I expect to make from this additional distribution channel for my product?" That is, we viewed this rearchitecting as a game-changing technology shift any reasonable application vendor should embrace. However, application vendors viewed it as merely a new distribution channel. This brought up the Catch-22: Without ADTs we could not get customers, and without customers we could not get ADTs. We were pondering this depressing situation, trying to figure out what to do, when the next crisis occurred.

Oakland, CA, 1994. We were again out of money, and the Land Sharks announced we were not making good progress toward our company goals. Put more starkly, they would put up additional capital, but only at a price lower than the previous financing round. We were facing the dreaded "down round." After the initial (often painful) negotiation, when ownership is a zero-sum game between the company team and the Land Sharks, the investors and the team are usually on the same side of the table. The goal is to build a successful company, raising money when necessary at increasing stock prices. The only disagreement concerns the "spend." The investors naturally want you to spend more to make faster progress, since that would ensure them an increasing percentage ownership of the company. In contrast, the team wants to "kiss every nickel" to minimize the amount of capital raised and maximize their ownership. Resolving these differences is usually pretty straightforward. When a new round of capital is needed, a new investor is typically brought in to set the price of the round. It is in the team's interest to make this as high as possible. The current investors will be asked to support the round, by adding their pro-rata share at whatever price is agreed on.

However, what happens if the current investors refuse to support a new round at a higher price? Naturally, a new investor will follow the lead of the current ones, and a new lower price is established. At this point, there is a clause in most financing agreements that the company must ex post facto reprice the previous financing round (or rounds) down to the new price. As you can imagine, a down round is incredibly dilutive financially to the team, who would naturally say, "If you want us to continue, you need to top up our options." As such, the discussion becomes a three-way negotiation among the existing investors, the new investors, and the team. It is another painful zero-sum game.

When the dust settled, the Illustra employees were largely made whole through new options, the percentage ownership among the Land Sharks had changed only slightly, and the whole process left a bitter taste. Moreover, management had been distracted for a couple of months. The Land Sharks seemed to be playing some sort of weird power game with each other I did not understand. Regardless, Illustra will live to fight another day.

The Future Looks Up (Again)

Troy, NY, Day 56. The innkeeper in Ellicottville tells us what was obvious to anybody in the 19th century moving goods between the eastern seaboard and the middle of the country. He said, "Ride north to the Erie Canal and hang a right." After a pleasant (and flat) ride down the Mohawk Valley, we arrive at Troy and see our first road sign for Boston, now just 186 miles away. The end is three days off! I am reminded of a painted sign at the bottom of Wildcat Canyon Road in Orinda, CA, at the start of the hill that leads back to Berkeley from the East Bay. It says simply "The Last Hill." We are now at our last hill. We need only climb the Berkshires to Pittsfield, MA. It is then easy riding to Boston.

Oakland, CA, 1995. Shortly after our

down round and the Catch-22 on ADTs, serendipity occurred once more. The Internet was taking off, and most enterprises were trying to figure out what to do with it. Uptone executes a brilliant repositioning of Illustra. We became the "database for cyberspace," capable of storing Internet data like text and images. He additionally received unbelievable airtime by volunteering Illustra to be the database for "24 Hours in Cyberspace," a worldwide effort by photojournalists to create one Web page per hour, garnering a lot of positive publicity. Suddenly, Illustra was "the new thing," and we were basking in reflected glory. Sales picked up and the future looked bright. The Voice-of-Experience stepped on the gas and we hired new people. Maybe this was the beginning of the widely envied "hockey stick of growth." We were asked to do a pilot application for a very large Web vendor, a potentially company-making transaction. However, we were also in a bake-off with the traditional RDBMSs.

The Good Times Do Not Last Long

Oakland, CA, 1995. Reality soon rears its ugly head. Instead of doing a benchmark on a task we were good at (such as geographic search or integrating text with structured data and images), the Web vendor decided to compare us on a traditional bread-and-butter transaction-processing use case, in which the goal is to perform as many transactions per second as you can on a standard banking application. It justified its choice by saying, "Within every Internet application, there is a business data-processing sub-piece that accompanies the multimedia requirements, so we are going to test that first."

There was immediately a pit in my stomach because Postgres was never engineered to excel at online transaction processing (OLTP). We were focused on ADTs, rules, and time travel, not on trying to compete with current RDBMSs on the turf for which they had been optimized. Although we were happy to do transactions, it was far outside our wheelhouse. Our performance was going to be an order-of-magnitude worse than what was offered by the traditional vendors we were competing against. The problem is a collection of architectural decisions I made nearly a decade earlier that are not easy to undo; for example, Illustra ran as an operating system process for each user. This architecture was well understood to be simple to implement but suffers badly on a highly concurrent workload with many users doing simple things. Moreover, we did not compile query plans aggressively, so our overhead to do simple things was high. When presented with complex queries or use cases where our ADTs were advantageous, these shortcomings are not an issue. But when running simple business data processing, we were going to lose, and lose badly.

Wollaston Beach, MA: Day 59

We were stuck with the stark reality that we must dramatically improve transaction-processing performance, which will be neither simple nor quick. I spent hours with the Short One trying to find a way to make it happen without a huge amount of recoding, energy, cost, and delay. We drew a blank. Illustra would have to undergo a costly rearchitecting.

The Stories End

Sutton, MA, Day 59. Massachusetts roads are poorly marked, and we have never seen more discourteous drivers. Riding here is not pleasant, and we cannot imagine trying to navigate Boston Bound into downtown Boston, let alone find someplace where we can access the ocean. We settle instead for finishing at Wollaston Beach in Quincy, MA, approximately 10 miles south of Boston. After the perfunctory dragging of our bike across the beach and dipping the front wheel in the surf, we are done. We drink a glass of champagne at a beachside café and ponder what happens next.

Oakland, CA, February 1996. Serendipity occurs yet again. One of the vendors we competed against on the Web vendor's benchmark has been seriously threatened by the benchmark. It saw Illustra would win a variety of Internet-style benchmarks hands-down, and Web vendors would have substantial requirements in this area. As a result, it elected to buy Illustra. In many ways, this was the answer to all our issues. The company had a high-performance OLTP platform into which we could insert the Illustra features. It was also a big company with sufficient "throw-weight" to get application vendors to add ADTs to its system. We consummated what we thought was a mutually beneficial transaction and set to work putting Illustra features into its engine.

I will end the Illustra story here, even though there is much more to tell, most of it fairly dark—a shareholder lawsuit, multiple new CEOs, and ultimately a sale of the company. The obvious takeaway is to be very careful about the choice of company you agree to marry.

Why a Bicycle Story?

You might wonder why I would tell this bicycling story. There are three reasons. First, I want to give you an algorithm for successfully riding across America.

```
Until (Ocean) {
        Get up in the morning;
        Ride east;
        Persevere, and over-
        come any obstacles that
        arise;
        }
```

It is clear that following this algorithm will succeed. Sprinkle in some serendipity if it occurs. Now abstract it a bit by substituting `goal` for `Ocean` and `Appropriate Action` for `Ride east`

```
Until (Goal) {
        Get up in the morning;
        Appropriate action;
        Persevere, and over-
        come any obstacles that
        arise;
        }
```

Since I will be using this algorithm again, I will make it a macro

```
Make-it-happen (Goal);
```

With this preamble, I can give a thumbnail sketch of my résumé, circa 1988.

```
Make-it-happen (Ph.D.);
Make-it-happen (Tenure);
Make-it-happen (Ocean);
```

In my experience, getting a Ph.D. (approximately five years) is an example of this algorithm at work. There are ups (passing prelims), downs (failing quals the first time), and a lot of slog through the swamp (writing a thesis acceptable to my committee). Getting tenure (another five years) is an even less pleasant example of this algorithm at work.

This introduces the second reason for presenting the algorithm. The obvious question is, "Why would anybody want to do this bicycle trip?" It is long and very difficult, with periods of depression, elation, and boredom, along with the omnipresence of poor food. All I can say is, "It sounded like a good idea, and I would go again in a heartbeat." Like a Ph.D. and tenure, it is an example of `make-it-happen` in action. The obvious conclusion to draw is I am programmed to search out `make-it-happen` opportunities and get great satisfaction from doing so.

I want to transition here to the third reason for telling the bicycle story. Riding across America is a handy metaphor for building system software. Let me start by writing down the algorithm for building a new DBMS (see code section 2).

The next question is, "How do I come up with a new idea?" The answer is, "I don't know." However, that will not stop me from making a few comments. From personal experience, I never come up with anything by going off to a mountaintop to think. Instead, my ideas come from two sources: talking to real users with real problems and then trying to solve them. This ensures I come up with ideas that somebody cares about and the rubber meets the road and not the sky. The second source is to bounce possibly good (or bad) ideas off colleagues that will challenge them. In summary, the best chance for generating a good idea is to spend time in the real world and find an environment (like MIT/CSAIL and Berkeley/EECS) where you will be intellectually challenged.

If your ideas hold water and you have a working prototype, then you can proceed to phase two, which has a by-now-familiar look (see code section 3).

As with other system software, building a new DBMS is difficult, takes a decade or so, and involves periods of elation and depression. Unlike bicycling across America, which takes just muscles and perseverance, building a new DBMS involves other challenges. In the prototype phase, one must figure out new interfaces, both internal and to applications, as well as to the operating system, networking, and persistent storage. In my experience, getting them right the first time is unusual. Unfortunately, one must often build it first to see how one should have built it. You will have to throw code away and start again, perhaps multiple times. Furthermore, everything influences everything else. Ruthlessly avoiding complexity while navigating a huge design space is a supreme engineering challenge. Making the software fast and scalable just makes things more difficult. It is a lot like riding across America.

Commercialization adds its own set of challenges. The software must really work, generating the right answer, never crashing, and dealing successfully with all the corner cases, including running out of any computer resource (such as main memory and disk). Moreover, customers depend on a DBMS to never lose their data, so transaction management must be bulletproof. This

Code section 2.

```
Until (it works) {
        Come up with a new idea;
        Prototype it with the help of superb computer scientists;
        Persevere, fixing whatever problems come up; always remembering that it is never too late to throw
        everything away;
        }
```

Code section 3.

```
Until (first few customers) {
        With shoe leather, find real world users who will say, "If you build this for real, I will buy it";
        }
Recruit seasoned start-up executives;
Recruit crack engineers;
Until (success or run-out-of-money) {
        Persevere, fixing whatever problems come up;
        }
```

is more difficult than it looks, since DBMSs are multi-user software. Repeatable bugs, or "Bohrbugs," are easy to knock out of a system, leaving the killers, nonrepeatable errors, or "Heisenbugs." Trying to find nonrepeatable bugs is an exercise in frustration. To make matters worse, Heisenbugs are usually in the transaction system, causing customers to lose data. This reality has generated a severe pit in my stomach on several occasions. Producing (and testing) system software takes a long time and costs a lot of money. The system programmers who are able to do this have my admiration. In summary, building and commercializing a new DBMS can be characterized by

```
Have a good idea (or two or three);
Make-it-happen—for a decade or so;
```

This brings up the obvious question: "Why would anybody want to do something this difficult?" The answer is the same as with a Ph.D., getting tenure, or riding across America. I am inclined to accept such challenges. I spent a decade struggling to make Postgres real and would do it again in a heartbeat. In fact, I have done it multiple times since Postgres.

The Present Day

I will finish this narrative by skipping to 2016 to talk about how things ultimately turned out. For those of you who were expecting this article to be a commentary on current good (and not-so-good) ideas, you can watch my IEEE International Conference on Data Engineering 2015 talk on this topic at http://kdb.snu.ac.kr/data/stonebraker_talk.mp4 or the video that accompanies this article in the ACM Digital Library.

Moultonborough, NH, present day. Boston Bound arrived in California the same way it left, on the roof of our car. It now sits in our basement in New Hampshire gathering dust. It has not been ridden since that day at Wollaston Beach.

I am still inclined to accept physical challenges. More recently, I decided to climb all 48 mountains in New Hampshire that are over 4,000 feet. In a softer dimension, I am struggling to master the five-string banjo.

Leslie is now Director of Marketing for an angel-investor-backed startup in New York City, whose software incidentally runs on Postgres. She refused to major in computer science.

Illustra was successfully integrated into the Informix code base. This system is still available from IBM, which acquired Informix in 2001. The original Illustra code line still exists somewhere in the IBM archives. The academic Postgres code line got a huge boost in 1995 when "Happy" and "Serious" replaced the QUEL query language with a SQL interface. It was subsequently adopted by a dedicated pick-up team that shepherd its development to this day. This is a shining example of open source development in operation. For a short history of this evolution, see Momjian.[4] This open source code line has also been integrated into several current DBMSs, including Greenplum and Netezza. Most commercial DBMSs have extended their engines with Postgres-style ADTs.

I now want to conclude with three final thoughts. First, I want to mention the other DBMSs I have built—Ingres, C-Store/Vertica, H-Store/VoltDB, and SciDB—all have development stories similar to that of Postgres. I could have picked any one of them to discuss in this article. All had a collection of superstar research programmers, on whose shoulders I have ridden. Over the years, they have turned my ideas into working prototypes. Other programming superstars have converted the prototypes into bulletproof working code for production deployment. Skilled startup executives have guided the small fragile companies with a careful hand. I am especially indebted to my current business partner, "Cueball," for careful stewardship in choppy waters. Moreover, I want to acknowledge the Land Sharks, without whose capital none of this would be possible, especially the "Believer," who has backed multiple of my East Coast companies.

I am especially indebted to my partner, Larry Rowe, and the following 39 Berkeley students and staff who wrote Postgres: Jeff Anton, Paul Aoki, James Bell, Jennifer Caetta, Philip Chang, Jolly Chen, Ron Choi, Matt Dillon, Zelaine Fong, Adam Glass, Jeffrey Goh, Steven Grady, Serge Granik, Marti Hearst, Joey Hellerstein, Michael Hirohama, Chin-heng Hong, Wei Hong, Anant Jhingren, Greg Kemnitz, Marcel Kornacker, Case Larsen, Boris Livshitz, Jeff Meredith, Ginger Ogle, Mike Olson, Nels Olsen, Lay-Peng Ong, Carol Paxson, Avi Pfeffer, Spyros Potamianos, Sunita Surawagi, David Muir Sharnoff, Mark Sullivan, Cimarron Taylor, Marc Teitelbaum, Yongdong Wang, Kristen Wright, and Andrew Yu.

Second, I want to acknowledge my wife, Beth. Not only did she have to spend two months looking at my back as we crossed America, she also gets to deal with my goal orientation, desire to start companies, and, often, ruthless focus on "the next step." I am difficult to live with, and she is long-suffering. I am not sure she realizes she is largely responsible for keeping me from falling off my own personal cliffs.

Third, I want to acknowledge my friend, colleague, and occasional sounding board, Jim Gray, recipient of the ACM A.M. Turing Award in 1998. He was lost at sea nine years ago on January 28, 2007. I think I speak for the entire DBMS community when I say: Jim: We miss you every day.

References

1. Date, C. Referential integrity. In *Proceedings of the Seventh International Conference on Very Large Data Bases Conference* (Cannes, France, Sept. 9–11). Morgan Kaufmann Publishers, 1981, 2–12.
2. Madden, S. *Mike Stonebraker's 70th Birthday Event.* MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, Apr. 12, 2014; http://webcast.mit.edu/spr2014/csail/12apr14/
3. Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., and Schwarz, P. Aries: A transaction recovery method supporting fine granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems 17*, 1 (Mar. 1992), 94–162.
4. Momjian, B. *The History of PostgreSQL Open Source Development*; https://momjian.us/main/writings/pgsql/history.pdf
5. Stonebraker, M. The design of the Postgres storage system. In *Proceedings of the 13th International Conference on Very Large Data Bases Conference* (Brighton, England, Sept. 1–4). Morgan Kaufmann Publishers, 1987, 289–300.
6. Stonebraker, M. and Rowe, L. The design of Postgres. In *Proceedings of the 1986 SIGMOD Conference* (Washington, D.C., May 28–30). ACM Press, New York, 1986, 340–355.
7. Stonebraker, M and Rowe, L. The Postgres data model. In *Proceedings of the 13th International Conference on Very Large Data Bases Conference* (Brighton, England, Sept. 1–4). Morgan Kaufmann Publishers, 1987, 83–96.
8. Stonebraker, M., Rubenstein, B., and Guttman, A. Application of abstract data types and abstract indices to CAD databases. In *Proceedings of the ACM-IEEE Workshop on Engineering Design Applications* (San Jose, CA, May). ACM Press, New York, 1983, 107–113.

Michael Stonebraker (stonebraker@csail.mit.edu) is an adjunct professor in the MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA.