EPISODE 861

[INTRODUCTION]

[00:00:00] JM: Starting a database company is not easy, and that's because a new database company needs to solve numerous problems in order to succeed. There are already lots of existing database companies. So a new company needs to find a way to strongly differentiate itself.

Databases are core infrastructure, so a new database company must earn trust with its customers. A database is a complicated distributed system, so a database company must have strong engineering to get its product to market.

Citus Data was founded in 2011. Around that time, companies were looking for solutions to both their online transactional workloads and their large scale offline analytics workloads. Citus Data builds its company around PostgreS, a popular database that has been around since 1996. Products from Citus Data include a PostgreS scalability extension, as well as a cloud hosted offering for spinning up PostgreS instances.

Citus Data was successfully acquired by Microsoft earlier this year, which was a huge success for the company.

In today's episode, Umur Cubukcu joins the show to talk about the story of Citus Data from its early days to its eventually acquisition. Umur describes the landscape of data systems in 2011 when the company started and explains how that evolved to the current ecosystem. Umur also talks about how to make an acquisition successful and give some perspective on the future of data platforms.

If you're looking for all 1,000 of our old episodes, check out the Software Daily App for iOS. It includes all our old episodes and including related links and greatest hits and topics. You could sort the episodes. You can make playlists. You can find all the episodes related to particular businesses, or particular open source projects, or infrastructure software, or cloud computing, or

various influencers in the software engineering world. If you want to become a paid subscriber, you can get ad-free episodes by going to softwareengineeringdaily.com/subscribe.

Altalogy is the company that has been helping us develop some of the software for Software Daily, although all of that is open source in the Software Engineering Daily GitHub repo. If you're looking for a company to help you with mobile or web development, I recommend checking out Altalogy. They've really done a great job and we're still working with them. We will have the Android version app of our app out soon, and keep your eyes open for that.

With that, let's get on to today's show.

[SPONSOR MESSAGE]

[00:02:51] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[INTERVIEW]

[00:05:11] JM: Umur Cubukcu, welcome to Software Engineering Daily.

[00:05:13] UC: Hey, Jeff. Great to be here.

[00:05:15] JM: You founded Citus Data in 2011 with your two cofounders. Describe the relational database market back then.

[00:05:24] UC: So, that's been a very different picture than what it is today. Well, one part of it that was similar is they're two open source databases on the relational side. That really could move the needle. One is PostgreS and the other is MySQL, and that's kind of remained largely the case on relational side still today. Now, there's of course MariaDB as kind of a derivable of MySQL.

So that side of it was the relational health, and of course you had the big, the traditional players, Oracle being the largest of them as well as Microsoft SQL Server, IBM Db2 and a bunch of kind of traditional behemoths if you will. That covered, I would say, 90 plus percent, almost 95 plus percent actually of the entire database market about a decade ago. It was all relational, much of it from a dollar's perspective. Obviously, it was kind of closed sourced and commercial. Then there was very little by way of open source NoSQL databases. That was actually the beginning of those movements around 2010, 11, 9. Actually, one of my cofounders was one of the earlier people at Amazon who started kind of playing with Hadoop in its kind of early beta days. Not even actually prior to beta, even alpha days.

We could see and actually kind of actively saw how a lot of the kind of NoSQL as a movement was forming, and that was kind of all the rage if you will. In fact, when we started Citus, people

told us, “Hey, why are you doing this on SQL? SQL doesn’t scale. You should do NoSQL. In fact, you should do Cassandra.” That was the zeitgeist when we founded Citus.

[00:07:07] JM: What was the original go-to-market strategy for Citus Data?

[00:07:10] UC: Originally, one way was to actually first build the database and put it on kind of a – It was a closed source version of PostgreS, which in different forms actually had been tackled before. You had forks of PostgreS prior to Citus in kind of 2005 to 2010 era with all of what’s called MPPs, the massively parallel processing databases, like Greenplum, like Aster Data, and Atiza. You have a lot of the – What’s Redshift now, which used to be ParAccel. Kind of all of those things had been in the market as forks of PostgreS from kind of a lot of earlier versions.

At Citus, actually we have a different take on this problem. You’re not going to start from scratch, because we see a lot of opportunity in terms of how quickly the data landscape is evolving. How the scale piece is still missing, and we need to fill that in. But if we were to build it from scratch, that’s just going to be overkill and that’s kind of the path, kind of the Hadoop movement and others kind of took.

So, that part of it to us was clear. We’re not going to start from scratch. We’re going to build this on one of an existing stack, whether MySQL or PostgreS. We had a long debate. Super happy that we landed on PostgreS for a lot of good reasons, because – When we started those discussions, by the way, MySQL wasn’t yet part of Oracle. So Sun acquired it and then kind of Oracle eventually. But that wasn’t kind of a given either.

So, that was one part of the equation. We said, “Okay, we’re going to do this on top of PostgreS from a business model. We’re going to adapt something that’s closer to what the kind of earlier folks of PostgreS did, but we’re going to build it in a way that’s much more kind of closer to how PostgreS does it,” instead of kind of forking in these broad strokes. We’re actually going to build our code in a way that really kind of extends PostgreS. But we still packaged it as a different binary, which you have to download. It was still close sourced when we first kind of began our journey, and that’s changed of course quite a lot since then till now.

[00:09:13] JM: I'd like to put the time in context both then and now. So, historically, the world of databases, it's been easier to model all of our data relationally in earlier times when we had things like user accounts and just simpler forms of data. But the explosion of applications and the cloud and the complexity of our infrastructure, it's led to things like complicated JSON objects. We have long and big log messages that we might be storing. How has the evolution of the structure and the volume of data changed our requirements for databases?

[00:10:02] UC: That's a very good question. Actually, it's not a new question either. When you think about databases going kind of all the way back to PostgreS' origins, back kind of in '95, even prior to that, PostgreS started life as kind of a non-relational database, kind of as an – Then it added the SQL APIs in '95 and became PostgreSQL. So that question of how much structure to impose and how to kind of tackle relational versus non-relational has been on for a longtime. I think part of it is we forgot that with kind of – That it happened in the 80s and the 90s and then kind of now we're revisiting it in a much kind of grander fashion today.

To that effect, a lot of these let's say less structured data types have been kind of on the periphery of the relational domain. First is XML, or kind of different data sources or data types have been kind of plugged in overtime into the different relational databases, but not kind of our second-class citizens. Maybe even third-class citizens. The dominant model has always kind of been relations and the relational paradigm around how that gets processed.

Now, I think a lot of the reason that JSON kind of came to rise is, one, the scalability of the relational database was very challenged. The paradigm was you had to build a bigger box and a bigger box and a bigger box as your data grew, and that kind of was okay if you were processing really valuable data, like transactions in your kind of credit card processing system. You could afford that. But as you started collecting data around periphery, then in kind of much larger volumes, that model didn't scale.

Then people talk about scalability. I think one of this part is just the strict performance and how much volume of data there is. But I think even more important part of it is the cost, both the dollar cost of being able to scale and the engineering cost and complexity of being able to scale. Relational didn't do well there. So, you had the kind of existing paradigm, which is I'm going to charge you $40,000, $50,000 per core, and that's kind of list prices that you're talking about

from kind of the large vendors back then and kind of in many ways still are. Then of course I'm going to give you huge discounts.

But when you think about it –

[00:12:29] JM: Sorry. Per core per year or per month?

[00:12:33] UC: It's outrageous, isn't it? It is per core perpetual is the typical model.

[00:12:38] JM: Oh, perpetual.

[00:12:39] UC: So let's say if I'm, vendors like Oracle, and I've got one core and you got different options you could add into your database. You got the standard version, enterprise version, a bunch of kind of data protection and other features. But broad strokes, say I'm charging you 40,000 per core and then I'm charging you about a baseline of 20% plus of that per year on maintenance.

Then, of course, there's models, say, where you can do it kind of on a, "Hey, can I get it for a three-year term?" It doesn't always have to be perpetual that you can have it for. But, typically, you would think about it, say, like as five years. If you're looking at 40,000, if you're looking at another 20% per year on top across 4, 5 years, you're talking about like tens of thousands per core. Again, these are list prices and they're public. You can find them, like they are kind of published transparently. Then you would get a large discount. It could be 60%, 70%, 80%. But still, like you're in thousands. Even in your laptop today, you've got a bunch of cores of running. If you wanted to do anything meaningful, you could quickly jump into millions.

That structure – Again, by the way, the database starts underneath, that is really powerful. It does a lot of powerful and useful things, not just for the development of the application, but the enterprise integrations, the governance. There's a ton of things that does.

However, as you can clearly see, it just doesn't scale. If I were to throw 128 cores at this, 256 cores, not only does the cost scale the equipment. The entire thing becomes very fragile. So my database, my kind of crown jewel is running on this huge machine, and I have very little kind of

control over it. Then if it's down, my entire business is down, and taking backups and then kind of maintaining it from a developer's perspective is nightmarish. So, that was kind of the operational side of it and the cost side of it.

I think what's really happened over the last 10 years, I'd like to think of like the cost or the value, dollar value per byte of data, if there is such a metric, really dropped. When I previously record, database could be a credit card transaction or kind a phone call billing, that's used for billing. The dollar value of those bits and bytes are relatively high. Then I can afford to use perhaps kind of one of those kind of old model databases in it. But if I am collecting engine telemetry data 95%, which is kind of the same always throughout and I'm looking for kind of small anomalies, like the data by itself. Every byte of data that I'm storing is much, much less valuable from a dollar perspective. So I need a different model to be able to process that.

That's really what draw all of the kind of the Hadoop movement, the NoSQL movement was, "Hey, you know what? I'm going to give you free software. It's going to be open source. It's going to be kind of free as in free beer. You can use as much of it you like, and I'm going to give this on kind of commodity hardware off-the-shelf as supposed to these specialized appliances you could go buy from Dell or from HP or whomever you like, and kind of essentially off-the-shelf free hardware, free software. I'm going to solve this problem for you."

I think that was the kind of fundamental premise, which got a lot of people excited for a good reason. But I think it was also a bit maybe too ambitious from – In a sense, kind of overpromised and underdelivered when you look at that entire lifecycle.

[00:16:19] JM: You mean Hadoop? Hadoop overpromised and underdelivered.

[00:16:21] UC: For example, exactly. Because the pain point is clear. You got the kind of incumbents. They are very expensive and they don't scale. That's a good problem to attack. But if the solution is I'm going to build what's been built over the last 20, 30 years with a lot of effort and kind of engineering kind of fundamentals and if I try to build that from scratch like in a matter of a few years. From scratch, I mean all the way from the file system. I'm not even using like the earlier paradigms, starting with HDFS and everything.

It's got be a useful set of applications, which kind of Google championed and kind of too good effect, but it's not a database. So if I treat that as a database building exercise from scratch over the course of like 10+ years, I think it really underdelivered, because it wasn't designed kind of like to do so.

[00:17:12] JM: Right. I think the way in which the Hadoop systems perhaps underdelivered could be described in the distinction between the early OLAP versus OLTP workloads, the early days of Hadoop. You would have these Hadoop jobs that would run on a nightly basis and you would wake up to an email that says, "Hey, your nightly Hadoop job ran and we have the customer purchasing amounts for yesterday's transactions summarized and sent to your inbox," and that's great, but it's not a paradigm that you can build real-time applications in.

[00:17:56] UC: That's very correct. I think when you think about, for example, it does a lot of things that are very useful, right? So, it does in fact at a very local storage, you could dump your data there. It does give you a lot of parallel compute, which you can use to kind of call things over, but in a kind of very offline fashion to your point, right? So I could run it kind of overnight. I could merge the data that I have, or things that are easily parallelizable. I can run across them.

I mean, maybe give a very kind of oversimplified example. But the way Google is using it is to kind of build inverted indexes kind of off the web and kind of other things as well. But if you think at the use case there, let's say I've got a file and I got a bunch of text written on it. If I want to count the number of occurrences of the letter A or something. So I can take that file and I could easily parse it out. I could say, "Hey, machine number one, take the first row of this file. Machine number two, take the second row." Then, "Hey, each machine, tell me how many A's you encountered. Then you'd give that to me. I'm going to add it up. Give it back to the user."

So, it's parallelized. That's my map and reduce. So I'm basically – It's a very effective way to kind of run parallel things across datasets. But what I've described isn't really a database per se. When I think about a database, it goes all the way from – There's an OLAP side. There's OLTP side. There is the performance and latency, but there's also the data – The governance, the security around it, the roles, access controls, everything that a DBMS does is kind of not included in that package just yet, which basically is the journey that went on.

So from kind of building, “Okay. This is the first use case. So then kind of people want to use this as a database. So let me build those one at a time.” But if you kind of start on that path, then you’re basically building a new database from scratch, and I think putting too much too broad of a context into something that’s specialized and useful, but treating it as kind of too broadly. I think that’s largely what happened with Hadoop.

[00:20:09] JM: You started Citus Data with Ozgun and Sumedh. How did you meet them?

[00:20:14] UC: We were all actually together at grad school at Stanford. So this was like earlier part of I guess 2000s, 2001, 2 through 2004. But before then, I knew Ozgun all the way from high school actually. So, we went to boarding school back in Istanbul in Turkey. Basically, it’s been almost 30 years actually since I met Ozgun. But we grew up together, but then actually at Stanford is when we really intersected, and Sumedh, he was also in the CS program at Stanford together with Ozgun. Afterwards, both Ozgun and Sumedh actually went to Amazon. Then kind of a lot of the foundations of, okay, what you see in the market, kind of those observations came to being, then we reunited. We founded Citus.

[00:21:05] JM: What was the division of labor between you three for Citus data when you got started?

[00:21:10] UC: So, practically, between the three of us, I took things – So Ozgun and Sumedh draw the technical side of things, and I took everything that’s practically like non-engineering, as would be the case in kind of an early startup. That included customer-facing functions. It included keeping kind of like the office running, whether talking with investors and really kind of looking at the go-to-market side of things.

But of course, back then, we didn’t think of it as a go-to-market. We thought of that as engineering and non-engineering. Of course, as kind of overtime, it evolved – We took more of between the engineering kind of functions, more of the overall, like the architect and kind of the CTO of the company, and Sumedh took more of kind of the VP engineering type of role. Then I took on things around both product and management as well as sales and marketing and everything else.

[SPONSOR MESSAGE]

[00:22:13] JM: DigitalOcean is a simple, developer friendly cloud platform. DigitalOcean is optimized to make managing and scaling applications easy with an intuitive API, multiple storage options, integrated firewalls, load balancers and more. With predictable pricing and flexible configurations and world-class customer support, you'll get access to all the infrastructure services you need to grow. DigitalOcean is simple. If you don't need the complexity of the complex cloud providers, try out DigitalOcean with their simple interface and their great customer support, plus they've got 2,000+ tutorials to help you stay up-to-date with the latest open source software and languages and frameworks. You can get started on DigitalOcean for free at do.co/sedaily.

One thing that makes DigitalOcean special is they're really interested in long-term developer productivity, and I remember one particular example of this when I found a tutorial in DigitalOcean about how to get started on a different cloud provider. I thought that really stood for a sense of confidence, and an attention to just getting developers off the ground faster, and they've continued to do that with DigitalOcean today. All their services are easy to use and have simple interfaces.

Try it out at do.co/sedaily. That's the D-O.C-O/sedaily. You will get started for free with some free credits. Thanks to DigitalOcean for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:24:15] JM: You mentioned that the initial go-to-market strategy for Citus was eventually modified to focus on PostgreS extensions, and I've spoken with Ozgun about PostgreS extensions at length. So we don't need to go into much detail. But from a business strategy standpoint, why were the PostgreS extensions a useful abstraction to focus your business around?

[00:24:46] UC: So, actually, extensions is kind of means to getting that. But, yes, the way we thought – We've evolved several times kind of at Citus from when we first built it. Like I said, it was first kind of a modern version of kind of a scalable PostgreS that used these smaller kind of

blocks of data underneath, which later we've referred to as shards. But they're much more kind of granular than that. Then we said, "Okay. This is how it actually can scale." It took some powerful ideas from that block-based architecture of GFS and applied it to databases and a bunch of things happening kind of under the cover. We said we could sell this as a closed source product, but in a way that's easy to consume. You could download it. You could use it for free. You always had the developer interest kind of at the core of it.

So we had a lot of debates, but then always conversed, "Okay. Let's not put the usage of the product behind registration model. Any developer should be able to go to the website, click and button and start using the product without talking to us." If they wanted to, we can talk and kind of understand what they're doing. So that has always been our approach.

Then now as things evolved, we realized, "Hey, from a workload perspective, that focusing only on analytics was a very limiting factor." Then B, it was also very crowded. To your point, Hadoop – Again, this is 2012, 12, 13, 14, the heyday of Hadoop moment. Then on the other side, you have kind of like the NoSQL with Cassandra and along with other things. But, really, as it relates to scalable workloads, more of the kind of like that Cassandra workloads happening. We said, "Okay. What are our kind of core strengths?"

One of the things we know is PostgreS is very versatile and it can actually handle both OLAP and OLTP workloads in a single machine. None of them – You can't say, "Oh, it's the perfect, the most optimized OLAP database or the most optimized OLTP database," but it does both and it does them pretty darn good.

So, we said, "Okay. That's actually one of our fundamental advantages. Let's build on that. Let's kind of enable those workloads." In fact, before we could do that, several of our customers did. They've asked us if you – Rewinding the Citus product back many years, we actually – In our maybe first version, you could only do batch writes into the database. So like bulk copy.

One of our customers asked us, "Hey, I'm going to do real-time updates, inserts against this data. Can I do it?" We said, "Sorry, you can't. You'd have to do batch." They said like, "Why can't I? Can PostgreS do this?" We said, "It can." "Isn't Citus PostgreS?" "It is." "So, why can't I do it?" "Actually, you can. Here's how you do it." We kind of showed what's happening in kind of under

the hood and they instrumented it themselves. Then this happened twice. The third time, it kind of embarrassed, like, “This should actually be in product. People need this.”

That was kind of one of the first – Again, before we went to open source or like kind of what’s happening in the background of, “Okay, what type of workloads do we tackle through using Citus?” As we built those things, we relied heavily on the extensibility of PostgreS. If you’re tackling a transactional workload, something that’s much more real-time to match, then the amount of overhead that you want to introduce to the planning, the query planning and execution cycles, are much, much smaller than if you’re running a 20-minute query and you have the luxury, both luxury and the downside if you’ll do both of planning kind of longer. Then you use different execution types. What we called kind of dynamic – First we called them executors. Then as workloads, we added more workloads, we called them dynamic executors, which basically would pick based on the query that you’re running what path of code to execute and what planning to use.

All of those rebuilds using kind of the extension, kind of like the modularity of the PostgreS code. We can re-leverage that. Came a point where we said, “Hey, we could really rebase the PostgreS code, the Citus code to newer versions of PostgreS relatively quickly. But what if we made this a pure extension of PostgreS where we could capture all of those – Function out the benefits, but also give our customers and users more transparency?” Because we aren’t building this thing from scratch. As far as you’re concerned, you don’t know that. You get a binary and it’s kind of all bundled with PostgreS and maybe it’s a fork and maybe it’s not. But, in fact, we’re using like the Vanilla community version of PostgreS. So we basically took kind of that effort and then turn Citus into an extension of PostgreS.

Those actually all came about the same time. That I believe was March of 2016, if I’m remembering the years. Whereby within – That would be, because on the commercial side of things, we also raised funds from BCs here in Silicon Valley. What we told them at the time is actually before we did our series A, we said we’re going to open source this. Actually, that was always – Like at that time, that was very clear to us. We’re going to not just open source this and say, “Hey, customers. Go use it to your liking and let us know what you like. We’re going to do several changes at the same time.” We said, “Actually, we’re going to put all these logic into a PostgreS extension. We’re going to open source that and we’re also going to provide this as a

service, pay as you go model in the cloud. Then we're going to – For our enterprise customers who have so far been using Citus, we're going to continue giving them that experience in that closed source fashion with additional thousand missiles, and that's how we're going to grow our business." So that was basically – Excuse me, the journey that we went through in terms of how we extensified Citus and how we open sourced it.

[00:30:38] JM: Yeah. We've talked to Marco. We've talked to Craig a couple of times about these different engineering problems that you've worked through, the engineering opportunities of Citus. I'd like to get more of the market characteristics from your perspective, since you're CEO of the company.

So if I'm a developer. Let's say I'm working on my hot new startup. I've got everything in Mongo. I think a lot of companies that get started, everything is in Mongo. They've got a transactional Mongo database. They're building a ridesharing company or something like that, and it's just transactional data in Mongo. Then their business starts to takeoff and they start to realize, "Okay, we need faster data access for analytical processing. We want to return large analytical workloads, and we don't really want to query the Mongo database to do this. So we need to start thinking about data infrastructure. We need to start thinking about ETL jobs, or maybe we need to rewrite our entire transactional layer to write to something else instead of Mongo."

I think this ties to the rise of these databases that are "NewSQL", where if I'm looking to solve this problem, this problem of migrating from my purely transactional Mongo workloads to something that is more capable of doing analytical processing. I might go to the Strata Conference and walk around the Expo Hall and I'm going to talk to all of the vendors and I'm going to ask the vendors, "Hey, how much are you going to charge me to do this? Are you going to help me with it? Why should I port my data to your thing? Is it open source? Is it close source? Is it NewSQL? Is it OldSQL? Is it NoSQL? Is it YesSQL?" It's just overwhelming. It's overwhelming even for me as a podcaster. My job is basically to talk to these companies and try to get a sense of how they compare to one another, and I simply cannot understand. There's so many of these things.

So if I'm in this situation, I'm building my hot, new flying car ridesharing company that's taking off, literally, and all my data is in Mongo. How do I assess this gigantic vendor landscape?

[00:33:03] UC: Yeah, and you are of course by no means alone, Jeff. Because I think sometimes this is bliss like if you don't know what's going on, that you could just kind of go over to what's available.

[00:33:13] JM: So I should just roll a dice and pick a random one.

[00:33:16] UC: Exactly. Pick what comes in. Just go over – No. But I got a more serious note. Yeah, that is a very common problem. Of course, for developers, for enterprises, larger companies have this problem at a different scale.

So what I tend to look at is a couple of things. First off, take the foundations of kind of like the NoSQL databases. Take Mongo. Take any which one. I think it's born out of two fundamental shortcomings of the relational database at the time. One is that we talked about scale. Hey, as you get bigger, kind of what do you with this? Yeah, there is structure. I have to enforce structure and I need more kind of flexibility, especially as I collect these newer data types and I don't really exactly know my data model and how it's going to evolve kind of ahead of time.

Now, I think if you look at it where things are today, PostgreS has the JSON B data type kind of putting that as a very kind of important resident in its kind of portfolio of data types and executions that kind of work with it. So that's one. Then two is scale, or course, which we work at Citus to address for a longtime.

So, now, I can get scale. I can get kind of semi-structured data and structured data together. So when should I use it? When should I not use it? Then I think that becomes a function of really part what your existing kind of experience and biases have been. What angle are you approaching the problem from? Have you been used to using PostgreS or MySQL or Oracle before? In which case kind of you have that relational view of the world, or you've been kind of using the NoSQL model. That's one thing where your existing stack is.

Then two, the type of application that you want to build or that you're building. What workloads is it going to tackle? Three, how you see them kind of evolve. The second one from a workload's perspective, I think what everybody wants to do, build an application that will one day scale. If I

didn't think that my application would scale, if I thought it was only going to be for just two users, I probably be building it anyways. You're always thinking, "Hey, the application is going to scale a lot regardless of whether it will eventually or not."

So you don't want to be locked in to an architecture that's going to not scale for you. In other words, let's say you're building a building and you're an app developer, but instead of – You're a construction, you're an architect, really, like a physical world architect. You have this vision of a 50-storey high-rise with a lot of kind of bustling activity, but then you're laying the foundations for it.

For vast majority of cases, you might be find building like a two, three-storey building is kind of contain the work that you need. But if as an architect I come with a blueprint, that will cap you at three floors. Of the foundation, that beyond three floors, won't go any up. Then I'm going to get a lot of pushback not just from – My boss is going to say, "Hey, you're not thinking about the future or from kind of my peers and elsewhere." So this notion of, "Hey, something needs to be scalable," I think is a real – Your concern upfront. Then I think – So you want to have scalability. The option to scale if you will baked into the system. We though overdoing it, right? Because if I want to build kind of a two-storey building and if I then undertake a skyscraper project with kind of all these cranes and other things, then I'm incurring a lot of overhead, a lot of cost. I think basically over-engineering kind of my system.

I think there are several good technologies that you can rely on. I know PostgreS is the best because that's what I've basically spent kind of all these years. But then you could pick MySQL. You could pick – I think if you're operating on the NoSQL side, you could use Mongo for certain applications. I don't think you need 8 different databases to provide one simple kind of application, like one graph database to do this and an in-memory database to do that and the different early on. I think that's a large company and a large service problem which you can think of in different ways. But you want basically an all-purpose database that can do things for you that you can iterate on fairly quickly. When the time comes, be able to not be in so much debt that you can kind of get out of and do different things with.

With all of those – Again, PostgreS, like I said, have a set of biases and a set of also knowledge about that product more so than any other for me personally. But PostgreS offers a very good

way to do that. I can start, it's reliable, it's got a huge ecosystem of people who know about it, developers who can work with it, tooling, etc., different vendors who provide it in the cloud, on-prem, wherever you like. So it's a safe bet. PostgreS isn't the only database that is a safe bet, but it's one of the few that is both kind of open source. There's a lot of developer credibility. So that's I think, for many cases, starting there is a good starting place.

Then as you think about your workload, then give kind of a couple of examples where you can stay entirely in that paradigm where it might make sense to decompose things kind of into different things. But let's say – This is something we see a lot within kind of Citus users. If you're building a SaaS application, you're building the next Salesforce or you're building whatever you like. Take your favorite pick of the day where the reason I say it's a SaaS business. A lot of them, not all, but a lot of SaaS businesses tend to be like B2B, business to business. You could have a SaaS product like a one password or something that is also consumer-based, where a consumer pays on a monthly basis. Many successful examples, but the vast majority is really kind of business to business.

In either of those cases, and especially B2B businesses, all of your customers have isolation. They have data that's only they should be able to see. Again, often in a B2C case as well. You only want one customer to see their passwords if you're one password. Then you've got then groups and teams that kind of aggregate on top of that.

So that paradigm of each user being able to see their own data as a SaaS startup is very common, and it's actually very well modeled by the relational database. Because if you think about the next Salesforce or the next B2B app, you have customers that are going to be residing in that database, your customers. You have your sales people, who are the users of that system. You've got your products. You've got your leads, your opportunities, like your pipeline. All of those things are actually things that reside in your data model, and they have relationships with them. You can't have an opportunity without an account. An account is basically the company that's associated with that opportunity and you can't have an opportunity without an owner, a sales rep, which is a user ID.

These are all – When you think about it from the app to your database, very relational things. You've got a user's table. You've got an opportunity's table. There's a foreign key between users

and the opportunity in the user's ID column. Your database enforces those constraints, because if it doesn't, then you will have to kind of at the application layer. Then you generate reports when you do them, say, "Give me all the opportunities for these accounts." Then you perform joins. You're performing left joins, etc. So it's very relational and it almost – If you try to model it otherwise as kind of these large JSON objects, then you're writing many, many duplications of data across many different kind of tables or collections and you're not enforcing relationships between them, because that is again what relational is about, etc. So in those cases, especially, just make sense to look at it as a relational database and then scale those multi-tenant workloads, whether you have one tenant or whether you have a hundred thousand tenants kind of in the future.

Actually, before I go into the different use case. Does that, Jeff, make sense from how you think about workloads as an example of, "Hey, I've got this database. What do I want it to do?" I think is the question you want to ask yourself, your app.

[00:41:43] JM: Yeah.

[00:41:44] UC: Depending on the workload, then you can actually dive deeper.

[00:41:48] JM: It does. Feel free to take me deeper, or I can ask you something else.

[00:41:51] UC: All right. All right. So, I can dive, diving deeper on that, should I say, like other workloads, right? You got the multitenant workload, where you can have actually teach tenant doing things, and then you're sharing data between tenants through what we call reference tables where you're kind of putting the data on every machine and making sure that when you perform joins, they're performing locally and you're enforcing constraints and all of that.

Now, there comes time where you actually want query across all of these tenants or query across these shards that you have created and then you want to say, "Okay. Give me all customers that have used this feature, or what's my total sales across all of the customers kind of in that database?" Your customers might not want to do that, because they only see their own data, but your product people might, your sales people might, or your compliance, your regulatory body might ask you to do that.

Kind of in each of those cases, you need a framework that kind of farms queries across multiple databases that are operating and get your results without having you to kind of worry about instrumenting that kind of from scratch. That's basically through Citus and PostgreS and using all of those technologies together give kind of a simple paradigm to do that. Hey, look, if your application could work fine with one gig of data on PostgreS. Now, down the road, you got a hundred terabytes. You should be able to run it with a hundred PostgreS machines, that your application shouldn't have to worry about all of these complexity that's kind of under the hood. That's kind of how we think about like holistic problem.

Now, if your workload isn't relational at all and if you've got basically one table and all you're doing on that table is putting and getting stuff and it's like this large log of events and you really don't have the notion of joins, where you're putting things in and taking them out. Then I think using a NoSQL, that it's a simple abstraction, that you can use that service.

Now I think when you think about many of those, like if your application relies on a lot of – Your application semantically, like not the way you built it, but semantic, logically, relies on different entities having relationships with each other. Then you might be well-advised just think about, "Okay. If I thought about this relationally, how would it look without creating 200 tables that interact with each other. Can I model this with 4 or 5 tables?"

I'll give you an example. In that event stream case, you could choose to say every device emits a signal as it kind of walks around my kind of security environment, and I got these massive large tables. Now, one way to model is every action, every signal that a device emits becomes a row and it includes all the information about that device in that JSON object where it's located and what it does, etc.

Another way to model it is to have a devices table that lists your 10 million devices and events table, which lists every action. Then you want to do something when you say, "Give me all the devices that the average – I don't know, age of the devices that have performed that action." Then you can join them on the spot on device ID. That way, you're normalizing some, but you're not going necessarily all the way to a full-blown relational model. You're still using JSON objects to capture your events. But you're also leveraging some of the goodness that the relational

database offers. You ensure that every device has a proper device ID. You can run aggregations across all of your devices table without touching other table and vice versa, etc.

I think then the problem on one extreme is having the relational, which you should probably think of nothing else but relational to model it. There is the other extreme, which is if your data model is simple enough that you could use like a small number of large tables to use NoSQL with. There is a healthy amount in the middle where it actually helps you to think about some things as relationally and then put the burden of that integrity on your database as supposed to your application, if that makes sense.

[00:46:09] JM: Yeah. I think you've been me a sense for why I always feel so overwhelmed at the Expo Hall. I think it's because I never have any particular application in mind. I'm always just strolling through the Expo Hall as kind of this like journalist person. But if I was able to say, "Hey, I've got my flying car rideshare company and I've got a NoSQL Mongo database, and here are the three applications, other applications I want to build. I want to build a billing system. I want to build a system for predicting different ride volumes over the course of the day, and I want to build a system that is going to measure weather patterns, because I need my flying cars to be only flying in acclimate weather." Then I could actually go to these different booths and say, "Here are my data models. Here's what's going on in my application. Here's where I'm going in the future. Tell me how you would solve my problem."

Then I would be able to judge more accurately how well is this company going to solve my problem. How well is that company going to solve my problem, or do I need some popery of different providers. How's that going to look? And then it would be easier for me to actually judge these things.

[00:47:32] UC: That's right, Jeff. That's a good way to kind of think about the problem, because you're not necessarily looking for, "Hey, give me just a toolbox." Question is what are you going to build with it, right? If you want to just buy a toolbox without – And that's also – You might just go and want to get a toolbox. In that case, you want something that's generic and multifunctional. In those cases, I think going with some of the safer choices, like PostgreS, or if you're coming from the MySQL side of the house, you could pick MySQL and can build things

on top. Then as you see the requirements of your application incorporate other things into it. I think it's a better idea than just going at it.

[00:48:13] JM: That's a pretty good sales pitch. Especially in light of all of these companies that are trying to build "data platforms", where you've got all these large legacy enterprises that are trying to figure out their big data plan and vendors are trying to figure out what's the best pitch to them. I think the PostgreS pitch is a pretty good one just because it's so old and reliable and also kind of it unifies a lot of these use cases in one place.

[00:48:42] UC: I think so, yeah. Also from like every integration down the line or when you build it is like as an extra thing to think about it and extra cost. It doesn't have to be that the database, it's not the license cost per se. It's the integration and the maintenance cost. The more systems you have, then your cost shifts to keeping the data in sync. If I'm proliferating and I'm using a new service to provide each part of just one unified app, then each – One of those things could be working fine. But then I have a data synchronization problem, and that's a big problem, by the way. You write into using one system into one place and you try to read, you have an ETL pipeline or you're moving data. Does it get synchronized? When something breaks, where has it broken? How do you trace back where that happened and going back to it.

So, I think in general, you want fewer pieces than like many pieces, but think about them deliberately such that, "Hey, not everything necessarily falls into one giant – Don't fall back on machine powering everything." But, actually, there are ways to scale that paradigm the way we've talked about. So, yeah.

[SPONSOR MESSAGE]

[00:50:04] JM: When I talk to web developers about building and deploying websites, I keep hearing excitement about Netlify. Netlify is a modern way to build and manage fast modern websites that run without the need for addressable web servers. Netlify is serverless. Netlify lets you deploy sites directly from git to a worldwide application delivery network for the fastest possible performance.

Netlify's built-in continuous deployment automatically builds and deploys your site, or your application, whenever you push to your git repository. You can even attach deploy previews to your poll requests and turn each branch into its own staging site. Use modern frontend tools and site generators, like React, and Gatsby, or Vue, and Nuxt.

For the backend, Netlify can automatically deploy AWS Lambda functions right alongside the rest of your code. Simply set up a folder and drop in your functions. Everything else is automatic, and there's so much more. There's automatic forms, identity management and tools to manage and transform large images and media.

Go to Netlify.com/sedaily to learn more about Netlify and support Software Engineering Daily. It's a great way to deploy your newest application, or an old application. So go to netlify.com/sedaily and see what the Netlify team is building. Also, you can check out our episode that we did with the Netlify CEO and founder; Matt Billman. That was a really enjoyable episode. I'm happy to have Netlify as a supporter of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:52:00] JM: Zooming out to these executive level decisions, independent database companies are feeling challenged in their competition with cloud providers. So, you've seen this with Elastic and Redis Labs and Confluent, more recently, CockroachDB. Where they're changing their licensing model in order to charge money to cloud providers that use their open source software without some other form of remittance. If you were on the board of one of these companies, would you support the decision to change the open source license to improve the business model relative to cloud providers?

[00:52:52] UC: I think, Jeff, that's a very kind of overall being able to make money from what you're working on is an existential question. So, what you can do to make that happen I think is always worth a discussion.

Now, how you do it, I have maybe some differences of opinion, but the spirit of it I'm not against. I think at the time, I'll give an example from Citus, which is of course like we went through that journey prior to Microsoft acquisition. But when we open sourced Citus, we had a long, long

debate on what license to do that with. I spoke with a lot of people. I spoke with people from the Apache Foundation, the founders of that foundation. I spoke with folks at MongoDB, at other database companies trying to get a sense of, “Okay, how do the AGPL workout for you? How decisions get made in kind of overall foundation?” All of that.

Then in the end, we converged on AGPL, which was controversial back at the time, because it’s not a popular license. Hey, there’s all sorts of concerns around, “Hey, does it limit uptake because it’s kind of a copy left,” but it’s at the same time recognized as an OSS license by the – It’s kind of the OSS foundations if you will. So it fits that bill, but it has restrictions on not how you use it. You can use it anyway which way you like. But, if you were to build on top of it, then it comes with conditions, which is basically you need to open source those. It tells the users, “Hey, you can use my technology. You can build your things on it without restrictions, really, which is a definition of OSS. But then if you’re making changes, all I’m asking you is to make sure that you open them.”

I think that’s a legitimate ask depending on what layer you’re building, because if you’re building, let’s say, an open source language framework. Then your choices are very limited. You have to practically think about it as a very liberal licenses where you’re kind of building things, whether it’s an MIT license or something along those lines. Then if you’re kind of at the top of the stack, not exactly top, but further up the stack, I should say, like on a database stand, you have I think more leverage to pull.

Now, specific to your question about what some of the other companies are doing, I think, worthwhile, the effort to say, “Hey, cloud company, Mr. and Mrs. X. If you’re using my technology, then I want some sort of either, hey, you could license it. In which, there’s an agreement that is kind of commercials in place. If you’re not, you should at least put what you’re building in open. I think that’s a reasonable ask.” Because these companies do need to – If they are VC-backed and need to return, kind of they have obligations to their shareholders. Then if they are not VC-backed or bootstrapped, which is increasingly few. I don’t know how many there are for many reasons, then my obligations to my customers.

But for me to give a better experience, I need to be able to fund this project. The only way I can fund it is through dollars from the customer. That’s the best source of funding. Not VC. Not

otherwise. But for me to be able to do that, I need to invest in RND, if you will. I need to hire and retain the engineering talent that's bringing in the software. For that, I need capital.

So, I think people understand that and I think it's a reasonable thing to do. I would do it the way which kind of we've done it through open source license as supposed to changing the license such it becomes close source or not a map OSS. In there are other ways to do it. But the spirit of it, I am supportive of.

[00:56:28] JM: I know we're running out of time here. I want to get some longer term perspective on the company, since you've been acquired by Microsoft. What was the biggest mistake you made in the duration of Citus Data being an independent company?

[00:56:46] UC: So we made many mistakes. No shortage of them since 2011, I think about 8 years. I think we did several right things as well, which took us like where we are. I would say I think we've thought of product and engineering and all of it as one for maybe a bit longer than we should have.

Meaning, everyone talks about product market fit, and that's a very real thing. The question is how do you measure it or how do you know you hit product market fit, etc. But kind of as we went, there was a time when we talked about Citus as just a scalable PostgreS, that, as an example. It's simple. I think you can think about it. It has some shortcomings, but it communicates a message. Now, the practicality of it is to do everything that PostgreS does in a scalable manner is a very large undertaking and we would do several of those things and do them really well. But we wouldn't do kind of like that kind of like the broader universe of those things.

So what that resulted is in actually a message market fit, but not a product message fit. So, almost I would – Knowing what I know, I would decompose the product market fit question into actually two separate questions. I want to say, "Hey, like a message market fit." What you say, does it resonate with the customer and says, "Okay, I get what you're saying." Then number two is product message. Given a message to the marketplace, does your product fit it?

I think that decomposition is probably valid in technology companies, where there's a message you make versus the message you communicate versus what you build. Like a large amount of engineering to incorporate.

I think had we done that sooner, then I think we could have converged into the use cases that our customers are using us for a lot sooner. Being able to be kind of more crisp with the messaging, and product already does those things. Basically, attract that type of customer as supposed to attract the more broad base of customers and can sift through it and then iterate faster.

So I think that's what allowed us to move quicker. I think from an engineering processes perspective, where we were located in Istanbul for a while and with YC, we came to the Bay Area. Of course, prior to that, we were already in the Bay Area with Stanford and other things. But we went back again to Istanbul, back here again. I think shuffle, if few streamlined that faster to be closer to our customer sooner, I think that would have, again, accelerated our cycles.

Then from a customer-facing perspective, once we found those use cases, then we could bubble down on those quicker by targeted messaging and targeted sales. That's one side of the product coin is engineering. The other side of the coin is sales and marketing. We could have invested in those in a more focused way. Overall, I think those would be my – There's more, but I think those are first few that come to my mind.

[00:59:58] JM: That's a really interesting answer, because I had some request from listeners for more episodes around those kinds of topics and just opportunities for people to think through stepping back and assessing the company on a more strategic, holistic basis. Because I think it's really hard to do sometimes if you're the CEO and there's nobody else who is thinking about the company in those terms and you're sitting there in your own head. You've got so much inertia in terms of the way that the organization is already structured. Like you said, just, "Oh, yeah. Product and engineering and everything is bucketed together." Overtime, maybe your sales are going well, so you never really rethink some of these axioms that the company has been built under. You just don't really have any oversight. I mean, that's probably a place where investors can help you or maybe like your cofounders, you can occasionally have like, "Let's

take a step back. Let's do our three-month plan meeting and should we refactor the company?" or something like that.

[01:00:57] UC: Yeah, and I think your cofounders are a huge asset in that, because you're thinking about – When you're going through ups and downs, you're generally doing it together. Then that the emotional ups and down are really balanced when you can talk with cofounders going through the same thing. I think that's one type of the support cycle.

The other can be your board if you can rely – And it might not be in a sense your kind of immediate board members, which you ask for business and company-wide thoughts. At the same time, that's also – I think that's an entire different maybe podcast. An entire different part how to deal with that, but at the same time your board isn't necessarily always the same level of engagement and kind of detail as your customers and/or other advisors might have. So you balance those perspectives as much as you can. But, yeah, I think taking that quick iteration as much as you can is I think the real asset.

[01:01:59] JM: Citus obviously went through the acquisition by Microsoft. An acquisition is this really – It's a singular moment in a company, and many people only get their companies acquired once. So it's this process where you're likely to have no experience before and there's a high likelihood you will never have an experience like it again. So it's very hard to play it correctly. So it's in some sense a totally zero-sum, totally finite game, but it's not zero-sum because you want to have a good relationship post-acquisition. So it makes it quite a tough strategic process to optimize.

I'm sure your investors were really helpful in this process. How did you formulate a strategy to go through the acquisition in an ideal manner?

[01:02:51] UC: So, here is – Jeff, probably, it's all about, you know, the ideal I think is a very, very high bar, but we did as good a job as we could with a few important parameters. We basically said there are three things that really mattered to us kind of in a deal, if you will. In this case, kind of like an acquisition.

We said, “Hey, the well-being of our product and our vision for it.” So we basically build something. We’ve been building it over like many years and we’ve done a lot at Citus, but there’s a lot more that we want to do, and we do have this kind of converge, how we can deliver the modern relational database. That’s basically what we want to do. So we need to see through that one side of it on the product vision.

The other is from our team’s perspective. They’ve been with us for a longtime, and have been through ups and downs and more of the ups and downs thankfully, which is where we are, but they put their heard and soul. We want them to be rewarded for it. The third are investors, where our shareholders, which have been nagging with us from kind of early on and have supported the company. So we want to be able to give a good return to our investors. Those are the three pillars of how we’re thinking about acquisition. If we have an offer or offers that basically meet those criteria, then we’re willing to entertain that and take that to the next level. If not, that’s fine. We’re going to continue building. We believe in this thing and we have enthusiastic customers. We like to have kind of a community of people who support us. So we’ll continue doing that.

So that was basically our calculus, and I think like beyond that it’s one of finding the right partners and kind of acquirers to form with, in our case, Microsoft has – We’ve been really fortunate to meet the team there across a longer time period and then both leadership level and all the people we worked kind of like day-to-day. We built kind of a lot of rapport and a lot of, “Hey –” At a personal level, we like he people that we’re working with, which in my opinion, matters a lot, because if the first pillar isn’t that object. Kind of delivering, going on on the product aside, then that maybe matters less.

By the way, there’s nothing wrong with that. There are different acquisitions whereby you could make an exist and then you could do other things, or it could be just a technology sale or it could be just a team, etc. Every acquisition is unique in its own way. For us, say, we wanted to take that forward and we really care about the team that we’re going with and working with and thinking about what’s ahead.

In the case of Microsoft, those things align about how they’re thinking about open source, how they’re thinking about PostgreS as a database. That can kind of grow overtime, and how

relational, kind of modern relational scale out structure could work. All of those things aligned, and we said, “Let’s make that happen.”

[01:05:55] JM: Well, that’s a good enough answer. Thank you, Umur. That’s really interesting. It’s been great talking to you. It’s been a privilege to talk to you and the Citus data folks throughout the lifetime of the independent company, and I’m really excited to continue the conversation as you are part of Microsoft. So congrats on the acquisition. Congrats on the long 8-year process, and thanks for coming on the show.

[01:06:17] UC: Hey, Jeff. Thank you very much for having me. One other thing I realize. I should give a shout out too, is one of the things we did right before the acquisition was actually donate 1% of our company to PostgreS, to the foundation, or to PostgreS kind of organizations. I personally been very happy to have done that. I think it goes a lot to the team. Our board has supported that decisions, and maybe for other companies that are operating in the open source space, it doesn’t have to be – Obviously, PostgreS could be any project.

I think we must say it wouldn’t been possible to build this company without all the work and the effort of the entire community. That’s one way of giving back. There’s other ways of giving back obviously, like many of them. I think they’re all orthogonal to each other, whether it’s contributing code or organizing events or reporting bugs or writing documentation. There is no end to how you can contribute. I think you can contribute with equity and it can valuable if Red Hat were to give 1%. The next step would be a fairly sizable – Again, that’s not to say they’re not contributing. They’re contributing in many, many ways. These are all orthogonal dimensions, which we can add and I would encourage people to think about that and if you can do more of it.

[01:07:32] JM: That’s a wonderful norm to encourage, because if you do – Right now, I mean, there are plenty of people that contribute to open source for no payment. But if you could encourage a norm of people giving money to open source when they have a windfall, that would encourage more people to start open source projects. Even just like saying, “Look, I think there’s a thing that should exist in the world. I have no idea how it would make money, but I see it would help a bunch of people,” and they just crack the collective action problem by just building this thing themselves with no expectation of recompense. But if you encourage this

norm of just giving people money or giving communities money, then it will encourage people to contribute back to open source. So these things really do affect incentives.

[01:08:17] UC: Yeah, exactly, and it can adapt overtime. So, awesome, Jeff. I really enjoyed talking with you as well. It's a pleasure to be here. Yeah, I look forward to staying in touch and talking again.

[01:08:29] JM: Sounds good. Okay. Maybe the next we'll talk about organizational refactoring. I'm sure that wasn't easy.

[01:08:34] UC: It sounds good.

[01:08:35] JM: Sure, it wasn't simple. Okay. Thanks, Umur.

[01:08:37] UC: Thank you.

[END OF INTERVIEW]

[01:08:42] JM: CD is a continuous delivery tool from ThoughtWorks. If you have heard about continuous delivery, but you don't know what it looks like in action, try the GoCD Test Drive at gocd.org/sedaily. GoCD's Test Drive will set up example pipelines for you to see how GoCD manages your continuous delivery workflows. Visualize your deployment pipelines and understand which tests are passing in which tests are failing. Continuous delivery helps you release your software faster and more reliably. Check out GoCD by going to gocd.org/sedaily and try out GoCD to get continuous delivery for your next project.

[END]