EPISODE 1537

[INTRODUCTION]

[00:00:00] GC: It was always possible to build distributed systems and replicated databases before, obviously. So, this is not new. What I think is new about the edge is this paradigm in which you can do all of that in an automatic transparent way, without having to understand my primary is here, this is what's happening. So, Turso will give you a URL, and this is the URL you're going to use to query, do anything. And then everything is done automatically for you. When you set up on your replica, your application doesn't have to be aware of that. I will route you automatically to the right replica. The same thing for the rights. So, you can just write from any replica, and that replica routes to the right place. You don't have to worry about any of that, you just get your URL and you write and read from it.

[EPISODE]

[00:00:50] AD: Glauber Costa, welcome to Software Engineering Daily.

[00:00:52] GC: Alex, thanks for having me.

[00:00:54] AD: Awesome. Yes, so you are the founder and CEO at ChiselStrike and I am very excited to have you on, because I feel like there's this SQL like Renaissance going on and I talked about it and know more about it. So, that's the lead. But tell me about ChiselStrike and about Turso and what you all do at ChiselStrike.

[00:01:11] GC: Awesome. Happy to. So, ChiselStrike is a company that I founded with my co-founder Pekka in the end of 2021. Pekka and I work together at the Linux kernel. So, just to briefly touch on my background, I started my career, my first 10 years, I was working for Linux kernel. I work with things like virtualization, that gave him Hypervisor, Star Systems, the x86 boot sequence, all sorts of low-level stuff like that. And Pekka, at the time, was the maintainer of the memory subsystem for Linux. This is how I started to get you hate him. Because I had to get my code through him. We still have a very good interesting love and hate relationship. I hate him 95% of the time, and I love him 5% of the time, but that's enough for us to get the company together.

After that, he followed me. I mean, he's been following this whole time. But then he followed me into a company, that Avi Kivity, the founder of the KVM hypervisor created, the company was doing like an operating system thing. It was a kernel for virtual machines. So, it was a kernel written in C++ and it didn't work. Technically, it did, but it didn't get any market traction. And then, it pivoted to a database like a petabyte scale, no SQL database called Sylow, which was a reimplementation of Apache Cassandra in C++.

So, we've been working together for 20 years, databases, kernels, operating systems, low-level stuff. At the end of 2021, we look around the market a little bit and we try to understand, like, we were very interested in starting something together, and we're trying to understand why – first of all, I don't want to write any database, just that there are too many databases. I later, as I started getting more acquainted with the framework in JavaScript ecosystem, I started to feel less about that because there are too many databases. Go look at JavaScript frameworks, it's a lot more.

But we have this feeling like there's just too many databases. I don't think the world needs yet another database. But are the databases like they are today, the thing that are going to power the future. So, let's look at what's going on. And then we started having chats with people. We started talking to friends and friends of friends, frontend developers, backend developers. And we understood that one of the things that – I have a thesis, like behind the creation of ChiselStrike, which is the following, giving part of my goal in here. It's the person making the decisions, like the persona of making decisions of which database to use is changing. And it's changing because of things like Cloudflare workers and cell functions and edge functions. You don't need necessarily like an application developer, can now go and have a very competent backhand without any help.

So, what do we do to bridge that gap? That was what we had in mind. One of the things about this space that's very interesting is how much we're seeing bus and reliance on the edge. So, we wanted to think about that space, and it was clear to us that, to do that, we would have to have a database that works very well from those environments that serves very well those people and matches very well what the edge is. So, that was the idea behind creation of Turso is our database, and ChiselStrike is the company.

[00:04:35] AD: Absolutely. So, you talk about the edge, you have this great post and I'm like, “What the heck is the edge?” Tell me about like these two different conceptions of the edge and sort of what you're aiming at with Turso.

[00:04:44] GC: Yes. It's interesting because lots of people mistake the edge for serverless and I think they're the same thing. They're not the same thing. So, serverless is, there's the usual joke like server – there are server somewhere, right? But servers allow a paradigm in which you essentially don't worry about servers. You'll just write your code. You're no longer writing services, you're writing functions. Those functions just run when something happens and you don't have to worry about provisioning, you don't have to worry about scaling those functions, they're just being executed.

I think because Cloudflare does edge through serverless, lots of people conflate the two. But if you look at Fly.io, for example, they do edge without doing surplus. And edge at the end of the day really just means a paradigm in which you can write an application that runs in multiple geographies much, much closer, then the cloud would allow you. So, for example, our database in the US alone allows you to have 11 regions. It's not only about like US one, East one, West one, et cetera. You have 11 regions to choose from and that's North America, actually. My Canadian friends will hate me because I've made this mistake. But like nine in the US, two in Canada. But it's just that, in North America alone, we have 11 regions to choose from, as opposed to the cloud.

So, I mean, now you really try to go the extra mile. We know that people go through great lengths to reduce the length latency of their application. But you're fighting physics, if you need to go from US, West one to the Midwest, and vice versa. So, you really want to put all those regions close to you. Again, you can do this with a serverless paradigm. Or you can do this with a server full paradigm as Fly is doing. Our database works well with both.

[00:06:33] AD: Awesome. Okay, and just I understand it, when I think of the edge like originally, it's CDN and sort of pops right with like static content on the edge closer to your customers. Now, with CloudFlare workers, with Fly.io, CloudFront functions, all that sort of stuff, you're getting compute closer to the edge, but a lot of compute starts to go back to like a central database, right? So, unless it's like, stuff that doesn't need a database, which is less interesting in most of the cases, you're not really pushing everything to the edge. But then, truth so helps sort of take the data to the edge as well?

[00:07:07] GC: That's right. That's why we wanted to write the database this way, right? Because essentially, again, and we do, I think you hit the right note with the evolution of this thing. The edge starts as the CDN. This is like – I've been calling this the Web edge, because in our industry, we would love to name different things with the same name just to spark confusion. But like the IoT, folks have been talking about the edge as well for a long time. But there's a difference. When you're talking about the IoT edge, you're going all the way to the devices, which is again, the edge of the network, that's why have the same name. But it's very different from the Cloudflare kind of Web edge. Because the Cloudflare Web edge, it's a server-side thing that is always online.

So, what you're trying to do is just be close to your users. The IoT edge is not always online. So, this is where you get things like CRDT, and again, you also have a very strong SQLite presence there, because we like you at the device. But this is a little bit different than the Web edge. And you start as the CDN, now just serving like static things. And with things like edge functions from Deno Deploy and Netlify Cloudflare workers, what you want to do is you want your compute to be close to the edge and why? To create, to allow your users to have those dynamic experiences, not just the static experience. Now, the missing piece of the puzzle is the data edge, which is what we try to bridge with Turso.

[00:08:32] AD: Awesome. So, I look at Turso and look at your sort of private beta announcement was a few months ago back in January. And it says it's a SQLite compatible database that you can query over HTTP, and replicate to many regions across the world, which is actually like, it totally blows my mind. Because that's not what I think of with SQLite, right? Querying over HTTP, multi-region, all that stuff. So, just so I understand like, it's SQLite, but I'm not sort of opening a local file, like I usually would with SQLite. I'm connecting over HTTP to your service. Is that correct?

[00:09:02] GC: Yes. Just to clarify, and I think we're going to have the opportunity to talk about that later, this is built on a fork of SQLite. So, we forked SQLite to bring this to the market. And when you're developing, so the idea is that when you are developing, you can do this development locally. Another thing, by the way, when we were discovering what are those people that are building applications at the front end? What is their workflow? You go to any framework tutorial, and even with things like Prisma, Remix, Astro, they all have a local SQLite example. Why? Because it's just the easiest thing to get started with, right?

We want to keep this experience, and the way your developer, should your platform allow. For example, you cannot do this with Cloudflare Wrangler, just because they don't have any local story. But should your tech stack allow, you can develop those things locally in a SQLite file. That's right. That's how you develop this.

But then what we've done, I mean Turso is not just libSQL or SQLite, Turso is a database built with SQLite at its core. So, we have something. But this is also part of the libSQL fork. It's a mode of SQLite, that allows you to have a server wrapper layer that essentially allows you to query over HTTP, right? So, that's what it's doing.

It's still SQLite. I mean, it's not like a Postgres server and there are reasons for that. And the reasons for that is like, we touched on that briefly, but the edge is very different than the cloud. Why can't you have the cloud in 11 regions? Because it wouldn't be economical, right? So, you put the edge there, which is essentially smaller servers with less power. If you really want to go to the edge, you need something that runs with fewer resources, and it's a lot simpler, right? So, SQLite is the right technology for this, which I think why we're seeing this renaissance. But when we essentially decided, let's put SQLite as the local database, but now we roll up a very thin server layer that gets those resources over HTTP.

Again, the reason for that is that the paradigm is functions. So, when you're executing a function, that function is ephemeral. It exists for 10 milliseconds, 50 milliseconds. You don't have time to download the full database. So, the next best thing is to just allow, get you as close as possible to your region, or do HTTP and why HTTP? Because most of those environments, don't even let you open a TCP connection, and then go from there. We do have a feature that is being part of our public beta now, that actually allows you to create a replica in your own infrastructure. So, if you don't have a serverless thing, if you don't have a serverless thing, if you're doing something like Fly.io, for example, **[inaudible 00:11:47]**, or many others, Fermion, whatever infrastructure you have, you can now build a replica inside your server for infrastructure.

So now, we can use files. I mean, now, it's a different story, right? But this is made, the HTTP thing is essentially bridging the limitations of Cloudflare workers, deploy versus cell functions and et cetera.

[00:12:09] AD: Awesome. Without HTTP, what's the story around how – when you're connecting? Is it more network based, like sort of security group stuff? Or is it like password type thing? Or what's that?

[00:12:20] GC: No, we were inspired by the success of MongoDB. So, we decided not to secure it at all. You can just go and delete it all. I'm just joking.

There is a user password stream that we discourage, but it exists and then you can create a JWT token. To get a token, you'll get this authentication token and use that.

[00:12:41] AD: Awesome. Okay. And then, how many regions in total do you support? You said 11 in North America? How many is it around the world?

[00:12:48] GC: Twenty-six.

[00:12:49] AD: Are those running in different cloud providers? Where are those – how are you running those regions?

[00:12:55] GC: Yes. So, at the moment, most of our infrastructure is on Fly.io. But we are already actually making good progress. We want to allow other pieces of infrastructure as well. Part of it, especially with the most important services to allow the on prem replicas to work. They work now, but with an easier setup, so we want to support some of those other providers out of the box. It is possible to our architecture would allow you to run even more regions in that, right?

[00:13:27] AD: Yes. Very cool. Okay, so you've had private beta open for a little, bit public beta is now available. People go sign up and do that. I guess, what sort of use cases have you really been seeing adoption with Turso?

[00:13:39] GC: So, the use cases that we see excitement the most about are really, unsurprisingly, the same use cases that were already moving to the edge with their compute. Ecommerce, I think is the big driver. Ecommerce is the use case in which you have a person at your store, and taking 100 milliseconds versus taking 10 milliseconds is a huge difference in the experience. So, you want to provide those dynamic experiences and you want to be there at the edge. The other one that we see a

lot is experimentation platforms, AV testing, things like that. I think there's a lot about feature flags as well.

Configuration. Vercel, for example, they do have a simple database that they support, as far as I understand that's ads ready and it's used for configuration. Again, if you have an application that needs to start, and then you need to check some configuration if the state is 100 milliseconds away, you've just introduced 100 milliseconds of startup time for no reason.

So, those are the main use cases that we see. But it is very early and I encourage, if folks are listening to this, if you have a use case that you think this will be a great fit for, that I haven't mentioned, get in touch.

[00:14:53] AD: Awesome. Yes. What about in terms of just like, is there like a maximum size I should be thinking about with Turso or something like that? Can it go up to hundreds of gigs or even terabytes? What should I be thinking about there?

[00:15:06] GC: Yes, that's a great question. For me, this is a funny question, because again, keep in mind that I spent almost 10 years at Scylla. Scylla is a petabyte scale database. So, when we were starting this, Pekka and I were always thinking, “Okay, so maybe let's allow the creation of a small database, a couple of terabytes and that should be enough, right?” But the use cases and the use cases that aren't in the web, and that's not needed. The usual use cases, outside of like machine learning and big data are much, much, much smaller than that.

It's doable to do a terabyte, but starts to push the envelope a little bit, because it puts a lot of stress at the replication layer. And the way we charge it, at least so far, is that we charge for the storage on the replicas. Although, we do have a feature that we're baking in to allow partial replication, so you can control which data goes where for legal reasons, sometimes, especially for the folks in Europe. But it's not a present thing. So, hundreds of gigs, I would say, it's a good maximum size. But a lot of those databases they can get quite small. So, they're very read intensive.

Usually, you don't write a lot. And it's one of the things and that's why SQLite, I think, and as a consequence, our fork is such a great fit for this, is that again, SQLite is not known to have very heavy write throughput. But you also don't need it. Those use cases or use cases are in which you don't write

that much. You write a little bit, but then you really want to allow read heavy workloads. Read performance can really scale up. That's pretty easy. So, I mean, we can support – we haven't tested that far, but potentially even millions of reads a second with compute costs scale with that as well.

But things in the 100,000s of reads per second, with maybe 10 to 200 gigabytes are the use cases that we support very well.

[00:17:13] AD: Awesome. One thing, I always say when evaluating new databases are like just sort of understand the tradeoffs of that database and what it's going for.

[00:17:20] GC: Absolutely.

[00:17:21] AD: It sounds like a few of the things of just like, is this a good fit? Or think about your database size, think how much right versus read you have on that stuff. Any other features that may be lost from a traditional relational database? Or like, “Hey, if you're using this in a traditional relational database, maybe it won't be the right fit for it.” Or anything else to look out for there?

[00:17:42] GC: One thing you didn't mention, by the way, was the consistency guarantees, right? And the structure that you expect from your data. Because what we've seen a lot with edge providers is that they will offer you a key value store. Key value is very simple to replicate, and sometimes its eventual consistent. So, this is another thing for me to understand. When I when I write this data, do I need transactionality? So, Turso will offer you transactionality. It will offer you access to data, that it’s passive replication, so going into like net nerds and IP here, the snapshot isolation. Your data can be a couple of whatever latency, hundreds of milliseconds old. But you're never going to break a transaction boundary. If you transact at something, that is guaranteed to be consistent. So, there's that.

It is a real SQL database allowed in like up to 26 regions. The tradeoff with that, is that again, by choosing – we could have built this with Postgres, and I understand that other people are trying to do, but then your costs would skyrocket for this level of replication. We're building this with SQLite, because we can offer like very little compute and very little, although again, you can get through to the higher compute things, but a little bit of compute and very little memory that SQLite uses. It's easy to put it everywhere. But then what you lose is again, the complexity of Postgres. Postgres has incredible

amount of extensions, incredible amount of things that allows you to do plugins and whatnot functions. So, you lose a little bit of that.

By the way, it's entirely possible to have both. You can have a central database with all of your data, doing part of your business intelligence, things that you don't need yet – I'm not claiming, by the way that you need the edge for everything. But there is a part of your data that if that data were to sit at the edge, that would have business value. So, that is the data that you move to the edge, right? The tradeoff is simply. But what I like about the most about our direction is that this is a simple – this is a tradeoff that for most users familiar with those technologies, it's already on your mind. When you think SQL versus Postgres, what we're bridging is that you have the tradeoff that SQLite is great for those use cases, as long as I don't need anything over the network. So, it can bridge that.

The other tradeoffs are still there. It's not going to be great for rights. Although there are ways to improve it. It's better for simpler things, not in terms of amount of transactions. But like if your usage of the database is simpler, that's about it.

[00:20:24] AD: Awesome. So, you mentioned one thing, just to clarify, if you have this configured with 10 different regions, one of those is going to be the primary, all rights are going to go through that primary that replicate out. What is sort of latency look like for replication to those different regions? Do you have a range of what that looks like?

[00:20:44] GC: By the way, just about that, I would clarify something just because I see lots of people getting confused with that. It was always possible to build distributed systems and replicated databases before, obviously. This is not new. What I think is new about the edge is this paradigm in which you can do all of that in an automatic transparent way, without having to understand my primary is here, this is what's happening. So, Turso will give you a URL, and this is the URL you're going to use to query, write, do anything, and then everything is done automatically for you.

When you set up on your replica, your application doesn't have to be aware of that. I will route you automatically to the right replica. Same thing for the rights. So, you can just write from any replica, and that replica routes to the right place. You don't have to worry about any of that, you just get your URL, and you write and read from it.

One of the things that we always encourage the people in the beta to do is like –we always know that benchmarking is a pain because if you don't count for startup times, you see the wrong numbers. But just create a replica far away from you, or create a primary far away from you. Start hammering the endpoint, then create a replica that is closer to you, and you will see that automatically your latency just goes down.

So, this is all automatic, including the primary. All of this to clarify, but now answering your question, the replication delay, assuming of course the system is not under an unreasonable amount of load is essentially the latency between the primary and the replicas. So, it could be 100, maybe half a second, a second, something like that. You may not have seen some transactions. But whatever you see, is transactionally guaranteed to be consistent.

[00:22:31] AD: Awesome. Do I start to see any longer delay or differences if I add more replicas? Does that make any impact on sort of performance across, I guess, especially replication performance across replicas, if I go from 2 to 20, or anything like that?

[00:22:46] GC: If the system is working well, no. But you seem to have a lot of knowledge about databases. You understand that databases sometimes don't work well. And this is true across the board. There is a reason why developers hate database so much. It's hard. But if the system is working correctly, and it's properly sized, and all that, then then you shouldn't have a bigger delay.

As I said, we're really architecting this for up to hundreds of replicas. Hopefully, who knows? Down the future, maybe even thousands. There's no need for this at the moment. But like, it's always fun as engineers to think about those scenarios. But you shouldn't have any visible major replication delays.

[00:23:33] AD: Do you see any use of dynamic replicas, where maybe people spin up a replica for a few hours or a day to help something –

[00:23:33] GC: Absolutely.

[00:23:41] AD: Really? Oh, man. That's just so wild that that's even possible.

[00:23:44] GC: Absolutely. Again, the reason that is that – and by the way, so there is – when you create a new replica, you need to move the whole data or some part of that data to that region, and there is a delay. So, you can't really do this at the millisecond level, might not even be able to do it at the second level. For us to replicate a small database, it takes around a minute or something like that. Of course, I mean, that's why we don't deal with terabyte size databases, because then we would have to move the whole terabyte to the other side of the planet. But absolutely. Let's say, you know that whatever reason, you're going to have something happening in Japan, you don't have any presence in Japan. It's going to last two days, come up with a replica, do your Japanese stuff, shut it down, move along, and our pricing model comports that, right?

Just for just for the record, like if you go to our website, you will see that right now, it's all free. There is a plan that you can pay for, but it's not enabled yet. We're just putting it out there so people can see what the price is going to be. That's one of the feedbacks that we received a lot during the private beta. Hey, but what if you charged me like millions of dollars? I'm afraid to build on this. We'll see what the price will be and some of what we would include. And we're hoping that in a couple of months, we'll be able to start onboarding people into the paid plan.

But the pricing model, obviously, open to change and open to feedback. But the pricing model that we're working on is essentially like based on the amount of storage and capacity and requests that you have done in each specific region, pro rata per day. So, you can essentially go and say, “Hey, let me spin up this replica for a couple of days. Leave it there, take it down, and really follow your traffic.”

[00:25:30] AD: Yes. Was that a hard conversation? I guess, like, you mentioned, doing a lot of talking with customers about what they're looking for in a database. Was that pricing conversation? What was that like? Do people like more fixed predictable stuff? Do they like the usage based? Does it just vary? What’s that look like?

[00:25:45] GC: Pricing is fun, because you ask 10 people and you get 15 opinions, right? But some of the things that we've heard a lot is that people like abstraction, right? People don't want to see pricing based on things that are not what they're doing. For example, compute, memory, and things like that. So, we heard a lot that I would like to see pricing, essentially, based on amount of storage, and amount of requests that I do. I try to even toy with a model in which we will essentially bundle one into the cost

of the other and just say, “Never pay me for – just pay me per request. Pay me per storage.” But it's hard, because then you always have those use cases that are very skewed from one or the other.

[00:26:34] AD: And you'll attract all of those people, because they’re like, “Wow. It’s a great deal for me.”

[00:26:36] GC: Exactly. The same problem that providers have, when they say we have an unlimited plan. Now we're getting all of the people who use a lot of data that is coming to you. So, if I were to say, “Hey, pay per request, don't pay for storage, then somebody would come with like a giant database on that.” And if I would say, “Pay per storage, don’t pay per request.” So, you're losing money, essentially. It's very hard. But at least we don't want to talk in terms of CPUs or machines or anything like that. We want to talk about request, bandwidth and those concepts that you understand.

I find it very – I don't like the – although, again, especially you in the audience that if you disagree, let me know. We're always open to have this conversation. But I don't like very much the model that is purely usage base. The model that I think works best is that when you have some plans that allow you to have some planning and some fair usage within that limit/ And then if you need more, then you add more. Because what happens with usage based is that you always have to set very high caps anyway, right? If it’s purely usage base, and you never understand, really, and this delta, this delta between the zero and your first plan, only matters for the really small developers anyway, and the prototype.

And then, what we've done, is that we just made our free tier very generous. Our free tier already allows for replication. I mean, you can essentially get your database replicating to three regions, which is cloud level of replication, in a free tier, and do like a billion requests per month, something like that. So, our free tier is very generous. For us, the way to bridge that gap between like a plan that is very cheap, that is used as bait, is just offer a free tier that allows you to run a production website, really, that doesn't have a lot of traffic. And then, after that, you have a paid plan with some limits. If you need more, then we're happy to add more.

[00:28:34] AD: Very cool. You mentioned a little bit earlier about like GPR, data sovereignty type stuff that you can do sort of partial replication, is that right? How do I sort of set that up, if I'm going to say replicate these users, but not these users or something like that?

[00:28:48] GC: So again, it depends on how engineer you want to be, because this is not a feature that exists. This is a feature that we are developing in. Although, I mean, this is obviously a great forum, because if you as a user have a need for this, I would love to talk to you, understand a little bit better, what are the things that matter to you. But what we're thinking is essentially, allow the user to specify with some declarative language, what kind of data must, can, and cannot be in which region? And think about those regions in like really, jurisdictional, geographical ways. Essentially, you could say, for example, this kind of data goes in Europe, and it cannot leave Europe in any circumstance. This kind of data can never go into Europe, or Russia, or – I don't actually think we have regions in Russia. But it's something that you could say.

The easiest way to do this is essentially to separate that per table and say, well, this table – so if you have a table of European users, the hardest way to do it is per row. So, you can essentially allow a row, a particular column, like a particular row. Column is too granular, but particular row in your database to live in a region. And then, obviously, there are the tradeoffs between those things. We want to keep this thing as simple as possible. We're going to lean on that edge simplicity. So, I actually liked the per table model. But we're not married to that yet, and we're still developing, we're still trying to talk to as many people as possible.

Again, people in our public beta, people who are listening to this podcast. If you have the need, we'd love to talk to you. But this is a feature that I think is very crucial to us, this ability to say – and it could be non-jurisdictional as well. It could be just need based. For example, you have a table of users in Japan, for example, then you don't you don't need to have this data replicated all over the place, right? So, you just replicate –

[00:30:46] AD: Yes. Interesting. Just thinking out loud to it, without that feature, would it be – I assume it would be feasible for me to have even three database instances, one that can be worldwide one that's like Japan only, and one, that's Europe only. And then maybe in my application, I'm sort of looking at that user and saying, “Oh, this is a Europe user, to the Europe database.”

[00:31:05] GC: You can. You can, and for example, our basic plan, our basic paid plan, we will already allow you to create more than one database. But the way we're looking to this is that we're going to give you a number of things, instances, and you can mix and match however you want. So, let's say the

plan allows you to create just tossing up a number because again, the plan is not available yet. So, I reserve the right to change it. But let's say, I will allow you to create eight things.

So, that could mean a database with seven other replicas, or that could mean eight individual databases, right? It's really, we want to give the user this flexibility. You could do this. But while you're losing with that is the ability to do joins across those tables, and keep the abstraction of SQL. We want to be essentially offering as much as SQL as possible. So, we want to do this on an abstracted way. But yes, it is entirely possible. I mean, it's not like if you have this use case today, Turso will not serve you at all. We want to essentially up the abstraction and allow you to have – this story is all about abstraction. I think the thing about the edge that's interesting is to, again, it's not geographical distribution. That exists for years, decades. It's the fact that this is now abstracted away to policy, and this is abstracted away. You don't have to think about where you're putting where. So, we want to go more and more towards that abstraction.

[00:32:35] AD: Yes. Very cool. Okay. So, I want to shift gears a little bit. I mentioned earlier, like, "Hey, if you're choosing your database, make sure you understand the tradeoffs they're making, whether if it's right for you." The other thing I tell people is like, "Make sure the team is focused on operational excellence. You can trust them." Because that's a core area you don't want someone sort of winging it, but see their pants. You mentioned your background, I think you understand a little bit, but all that work on the Linux kernel, ScyllaDB, like that is a serious team doing serious, serious use cases.

But tell you just like, what is it like building that foundation for a database service, and building that underlying infrastructure in the operational excellence? I mean, is everyone reinventing the wheel here, all these different databases companies, or what does that look like?

[00:33:18] GC: No. I don't think people are reinventing the wheel. While some people are, in some cases do call for a newer wheel. But remember, when I said in the beginning, we didn't really want it to just create a new database because that is reinventing the wheel. By the way, we see a lot of new databases. I'm not going to go and name anybody. But then, this is an era in which a lot of new databases are coming up. One of the things that lots of them try to do is break with SQL. Essentially, at the argument that SQL is not the language that we need anymore. I have a problem with that because you've probably heard that expression like never bet against SQL. And that we're a no SQL movement,

that MongoDB were crowned at. Again, Scylla was like that as well. Cassandra and et cetera, for some use cases in which you really do need to normalize those things and et cetera.

At Scylla, one of the things that was annoying the most for our customers is that this breaking with SQL comes at a cost. It comes at a cognitive cost that they don't want to pay. I've heard this from a person once that I met at a conference that was involved with the Manhattan database, not the Manhattan Project. That was the Twitter –

[00:34:37] AD: Is this the twitter one? Okay.

[00:34:38] GC: And he nailed it. Well, he told me is that nobody likes eventual consistency. Nobody wants eventual consistency. People tolerate eventual consistency, because it's the only way to achieve a certain thing.

Another thesis that I have that pairs with the one that I mentioned about who are the people making database decision, is that, outside of the super big data use cases, the recommendation engines and et cetera. Outside of the IoT sensors, the time series, if you exclude those folks, data is not really growing exponentially. Because what everybody says, is though data has grown exponentially, at Scylla, we use this graph all the time and data is growing exponentially. But when you exclude those use cases, it really isn't compute and storage are growing much faster than data.

For example, your product catalog is not growing exponentially, right? Lots of things are not growing exponentially. So, when you look at MongoDB, in 2010 or whatnot, when it was really popular, any website needed the horizontal scale, the web scale, might have heard the YouTube video, the guy like MongDB web SQL, everything needed this. Now, SQLite is enough for a variety of use cases, right? SQLite, just like this thing that is super simple. Why is that? Because now we have servers that you didn't have back then. The servers grew a lot faster. The storage grew a lot faster, and you can fit those things in an instance.

So, going more centrally to your question. I really think SQL has to stay. SQL has to be central to that. And then again, why would you write a new SQL database recovered? We've got three. So, you don't need more. What you need is essentially a new way to interact with those databases, and rethink, like, what are the things that people need today that they didn't need 10 years ago, but that's not going to be

at the database level. That's not going to be at the core, at the storage layer, the language layer. I don't think that's where the innovation will come from.

So, when you look at the new companies looking at databases, I don't think again, we're reinventing the wheel, because we're all trying to do something slightly different for different markets, maintaining that core of a SQL database, be MySQL, Postgres or SQLite, right?

[00:37:01] AD: Yes, I love that. Those are some great points, especially like, what type of data is growing. That core data, I agree, is like, usually pretty small. If it's not data that has a timestamp with it, it's probably not **[inaudible 00:37:14]** and things like that. That's a great point. I think I wore that question wrong. But I was thinking more just like in terms of building the foundation of just like backups and replication and sort of the operational and visibility to the customer and things like that, is that pretty hard to build that for a new database? I guess, how much of your team is focused on the pure operational aspects of sort of having a managed database company?

[00:37:43] GC: Yes. So, lots of things are made simpler by today's infrastructure. For example, as I mentioned, in the past, we are using Fly.io, as a platform. And although we may consider in the future, adding other platforms just to expose, just to essentially allow customers to have lower latency, we don't want to go down deep the Kubernetes hole. I happen to believe, this is again, a personal – maybe I shouldn't say that you're like a flame war, et cetera. But I happen to believe personally –

[00:38:13] AD: We're 35 minutes in. We're hiding it. We got the true believers here.

[00:38:16] GC: I happen to believe that Kubernetes was the mankind's second biggest mistake as a species, right? I totally don't like it. There you go, I said it. Maybe somebody's going to try to murder me today.

One of the things that make it a lot simpler is that we're not managing at this level, right? When you manage at this level, everything is a lot harder. So, in Fly.io, we still have VMs, and et cetera. But you have a lot of the infrastructure that they have essentially built for you already. So, there's that.

Again, also, it's not trivial. I don't want to trivialize that. But this is not the biggest problem we have, again, just because there's so much stuff around those solutions today, right? There's so much stuff

around those solutions. Backups for us kind of come for free, because what we've done just given getting a little bit technical, is that we've virtualized the SQLite Write Ahead Log. Again, SQLite doesn't allow you to do it, which is one of the reasons we've worked on the technical side. So, everything that that is seen as a new write is on the SQLite Write Ahead Log.

Other technologists try to hook up at the file system level and understand what's going into the Write Ahead Log. We don't think this is the right way to build it. This was essentially built this way by some other projects, because you can't get – SQLite will not accept contributions, which is something. But we just forked it. And then, we see every stream of changes the SQLite has, is seen by our technology. So, the same thing that we use to push to a replica, we can push to S3. And then you compact it every now and then, so you don't have an infinite Write Ahead Log. But you always have like the latest snapshot and a bunch of changes.

So, the same thing that we use for replication, we use for backups. And because we were making a database for replication. The backups were that hard. I guess, you can lose your data if Amazon goes down. It's possible, I guess, forever. If Amazon closes shop. Silicon Valley Bank did, maybe Amazon will one day, as well. But I'm not betting on it. So, this is fine. And we're just doing like a S3, D2, whatever.

[00:40:28] AD: Yes. When you say you're – so your snapshot, you're replicating the Write Ahead Log to S3. How frequently do you on every single come in? Okay. Cool. You mentioned a little bit about forking SQLite. You created libSQL there. You mentioned, part of that, sort of SQLite is open source, but they don't really accept external contributions. What are some differences? Some things you've added to libSQL? You mentioned the Write Ahead Log. What else was key for you?

[00:41:00] GC: SQLite is actually – I mean, they are open source. They're more than open source. They're public domain. Public domain is like really, like, no restrictions, right? But if you go to their website, what they say is that they are open source, not open contribution. Again, this is a statement that they made. So, the statement that we're making. The fun fact is that it's actually not impossible to contribute to SQL. I actually met a person who did it. It's just not welcome.

I want to make something clear, there's nothing wrong with that. SQLite absolutely is a very foundational core and well written and well-maintained piece of technology. I just don't think that this idea that for you to have something that is critical and mission critical and simple, you must not accept

contributions or make them very hard. Again, we come from Linux. Linux was much bigger, of course, but Linux was much was much bigger in code size. You had a lot of builds of Linux, that were targeting embedded systems, right? So, we built this – one, Linux didn't start as an embedded system, operating system. But when they need the roles, okay, so let's find a way to keep the big server and we did it, just by taking all those stakeholders into account. So, you had all the configuration flags and variables to disable parts of the subsystem, you would still have a large code base, but you will have a bill that would run on any platform. We just never believe that it's a necessity, although it is their right to make contributions this hard.

First and foremost, okay, we really want to – we've seen this renaissance in which everybody's looking at SQLite, and we wanted to have something that different people building with SQLite could convene, and could come. We know it's a hard task. We know it's going to take years until you develop this trust, and et cetera, and we're a small team. So, what we're doing is just welcoming people who can come. We already had some contributions from the community and we're building the features that we need, essentially. But the features that we built, first of all, was the virtualization of the Write Ahead Log. What is in libSQL is just the technology that allows the Write Ahead Log to virtualize, and then our product inserts whatever we want to do our business logic.

We also allowed you to create user defined functions in WebAssembly. Again, SQLite does support user defined functions, but you have to do it in C, and you don't have a language support to do this. So, with libSQL, you can just do create function, and then you name your function, you put a WebAssembly function there, that you can usually compile from your own language, like we do a lot of Rust. So, you can do that. You can allow those functions to operate like store procedures. And we're now toying with adding CRDTs, which is actually going to be a community contribution as well. We've been talking to an engineer from Facebook. I actually want to have a shout out to his project. VLCN.io, that essentially has an extension for SQLite for CRDTs, and we want to integrate that natively with libSQL as well.

First of all, I love the direction. Second, I think at some point, to allow fast rise from the edge, you need something like that as well in the future. So, those are those are a couple of examples. And then there's some minor differences. Some of them are very minor. For example, we wanted to add a row counter. Why? Because as I mentioned before, we want to charge you per rows. We don't want to be charging per CPUs and et cetera. SQLite tracks a bunch of things about the request. It doesn't track the number of rows.

This is so minor. Again, in Linux, it's something that a bystander would come and just contribute like it's a small patch. We're going to track the rows. Boom. And then you may have a discussion with this person. “Oh, are you sure this is not impacting performance? You run a bunch of stuff.” Again, Linux is not a project known to accept anybody's contributions willy-nilly. I mean, it is hard, but there is a process of path and a way, and it's doable. There's the sense that we want this community to form.

From the small stuff, like just really a row counter, so we can charge people per row. To more complicated things like CRDTs, WebAssembly, user defined functions, Write Ahead Log, virtualization, those are some examples of things that we have in libSQL.

The one I like the most, though, that I haven't mentioned is what we call bottomless storage. So essentially – this is not for Turso, although we use it internally to do the backups. But bottomless storage essentially allows you to have a local SQLite database, you don't even need the HTTP things, that is writing to S3. So, as you are dealing with your local file, this is all being backed up automatically to S3, and then you can do failover, you can get it back from S3. We also want to do a partial, essentially, partial match realization. So, we can keep some pages on s3 and some pages locally.

If you have a disaster scenario, it crashes, you start libSQL, starts right away. Again, this is not implemented yet. It's future. You don't have to download the 100-gigabyte database and then start operating, because you can pages as needed. So, there's a bunch of interesting things that we're doing in general for the product.

[00:46:13] AD: Yes, very cool. Well, I just have to like – yes, if you're building for this more serverless replicated environments, rather than the embedded or different things like that, that SQLite was originally used for, you have a need for. So, that's awesome.

Last thing I want to ask you, before we sort of wrap this up. You've seen a lot from working on the Linux kernel, working at Scylla, and the amazing performance stuff they're doing there with thread per core and everything. What's exciting you either in the world databases or systems or just performance, anything getting you excited?

[00:46:46] GC: Absolutely. First of all, every time abstractions rise, performance work, it's so much more interesting and harder. Because, I mean, you use to operate with some guarantees. They're just not there anymore, right? And it changes a lot. So, things for example, like code starts in serverless functions. I get very excited about that, because you have lots of those technologies. We're seeing, by the way, this is – I had no exposure to the TypeScript world before. But I'm seeing this thing with Prisma and Drizzle, right? So, you have like this very heavy server that is designed to operate in a server environment. It's a huge binary. They want to download that binary and keep it going.

If you have a long live server, it's fine. If you have a serverless function that has to do it over and over again, I mean, that's adding – it becomes a performance problem. So, you want those super, super light payloads to be in your serverless function. I get really excited about this kind of investigation.

Another thing that is more – again, this is something that impacts our product, but I reserve the right. But one thing that affects our product that I am very excited about is that when you start pushing this boundary of edge, and again, I do think SQLite is the right foundation to build upon, is that okay, now we're at the edge. What's next? And what's next is the browser, the device and et cetera, operating transparently. So, push your database and now have encryption problems to solve, and encryption problems are performance problems to some degree, because encrypting things is easy. Encrypting things in a way that doesn't destroy your performance is harder, right? So, how you're going to essentially push those partial databases, not only inside your Vercel functions, but maybe all the way up to your browser, and do it in a performant way that can allow like, writes not just suck so much. And it's still operate those databases. Super excited about it.

Also, what does it mean for GraphQL? If you can have a copy of your database with the data that you need in your browser? Do you still need GraphQL and things like that? Those are all things that I get very excited about.

[00:49:03] AD: Yes. It's so interesting with the updates and devices, and just library, all these new patterns and things like that. It's a fun time to be working on tech, and we didn't even get into –

[00:49:17] GC: If AI doesn't kill us, which in my opinion, Skynet and all. We'll have an interesting future ahead.

[00:49:25] AD: Yeah, awesome. Well, I really appreciate you coming on. For everyone's listening, go check out Turso. It's available in public beta now. So, you can check that out and reach out to Glauber and his team and let them know what use cases, what needs, or anything you have there, very receptive to feedback here.

[00:49:42] GC: Awesome.

[00:49:42] AD: Glauber, thank you for coming on.

[00:49:43] GC: Absolutel.y

[00:49:43] AD: Likewise.

[00:49:44] GC: Thank you.

[END]