# EPISODE 1443

[INTRODUCTION]

**[00:00:00] JM**: SingleStore is a multi-use, multi-model database designed for transactional and analytic workloads, as well as search and other domain-specific applications. SingleStore is the evolution of the database company MemSQL, which sought to bring fast in-memory SQL database technology to market. Jordan Tigani is CTO of SingleStore and joins the show to talk through the architecture and engineering of the SingleStore platform.

If you're interested in sponsoring Software Engineering Daily, reach out to us at sponsor at software@engineeringdaily.com. We'd love to hear from you and bring your message to our audience. We reach over 250,000 developers per month.

[INTERVIEW]

**[00:00:40] JM**: Jordan, welcome back to the show.

**[00:00:42] JT**: Thanks very much. Great to be here.

**[00:00:44] JM**: The last show we did was on BigQuery a while ago, back when you were at Google. And now you're at SingleStore, obviously. How does working on a data warehouse system like BigQuery compared to a more comprehensive database system like SingleStore?

**[00:01:07] JT**: Great question. I think being at BigQuery and working on a data warehouse or a data platform that was really tilted towards analytics, it gave me exposure to serve one side of the problem. But the overall database market is a lot broader. And where SingleStore sits on the spectrum of sort of transactions and analytics is a bit kind of translated over from that. But I think a lot of the customers are the same. A lot of the problems are the same. And there's a lot of new ones as well. So that's been actually pretty exciting, is to sort of understand developers, understand applications a little bit better than I had before.

**[00:01:44] JM**: The transactional semantics of SingleStore are significantly different than that of BigQuery or a data warehouse. Can you talk about those differences?

**[00:01:57] JT**: Yeah, sure. So SingleStore is a database that can do both transactions and analytics. And, as such, there's a subset of customers that come from the transactional side. And so, I think there's a bunch of cases where somebody needs in their application, they need to do something that looks like analytics. Maybe that's a leaderboard. Maybe they want to show a dashboard. Maybe they want to react to data about what's happening in the rest of – In the system.

And I think as people are building more and more rich applications, these kinds of needs come up more often. And I think one of the ways that you tend to see that is somebody will start with a database like Postgres or MySQL, great databases, work super well. But then they will start to hit roadblocks. They will start to hit scaling problems, kind of things that you can't do. And so then they start adding databases. They may add Mongo, because they want to do flexible schema, or they need document database semantics. And then they want to do faceted search or full text search. And so they move some of the data out to ElasticSearch. And then when you kind of put these pieces together, it tends to be slow. And so they add Redis on top as a cache. And then they want to do some data science, and so they export it out to S3, and then run Spark over it. And each one of these pieces adds latency, adds complexity. Something can go wrong as potential cash problems, etc.

And so having a single database where you can do all these things, it makes your life a lot easier. Sort of the SingleStore approach is, "Hey, instead of all these databases, you can just use SingleStore." SingleStore has SQL, distributed, relational, full text search, semi-structured, etc. So most of these use cases, you actually don't need to know database. And it's fast enough that you don't need to put a cache on top of it. Sort of that removes a lot of the complexity from your system, as well as helps you scale.

There're also people coming from the other side, which is they're coming from the analytics side. And if you come from the analytics side, there's people that want to do – They want to do low-latency dashboards. They want to do real-time analytics, and maybe use streaming analytics. And the fact that actually a lot of the times even an analytical system, you have things that look like transactions. You have things where you need – You have data that's coming in quickly. You maybe have things that are exposed to end users. So you need really high-concurrency. Like, data warehouses tend to not scale to 10s of 1000s of users all hitting it at once. So you need something that also kind of looks more like an application database or looks like a transactional database.

Also, as people want to make more sort of not just reduce the latency of the dashboards they get or the recency of the dashboards they get, but they want to be able to make automated decisions. If you make automated decisions, you really are going to have real SLA on how far you can be behind. Because if you're making decisions based on data that's too far behind, then you're going to be making poor decisions. For those kinds of reasons. You also need something that can handle that transactional semantics and analytical as well.

**[00:05:05] JM**: How much can you talk about the architecture of SingleStore?

**[00:05:10] JT**: I can talk about a lot of it. I'm an engineer by training. Has been 20 years as an engineer. I worked on a lot of the pieces of BigQuery. I worked on Windows kernel. So I love talking about architecture, probably more than I should when I talk to customers, etc. And I want to just sort of geek out on how things work.

And to me, actually the key thing about SingleStore, I mean, there's a lot of great things in SingleStore. But kind of the key thing that allows us to do these transactions and analytics in the same place, which often people say can't be done. Sometimes we tell people, “Hey, we can do transactions and analytics.” And they say, “I don't believe you. Michael Stonebraker said this can't be done. I trust him over you.” I was like, “Well, we've got the benchmark results to prove it.” And they still say, “Well, we don't believe you.” But kind of, if you understand the architecture, you can kind of see that it is possible. And it's related to our storage system. We call it universal storage. Actually, in fact, we originally called it SingleStore. And that's kind of where we got the name for SingleStore. We used to be called MemSQL. But we had this single storage engine that can handle both row store and column store. And so could do both transactions and analytics in the same place. And that's where we got the name. And then we had to rename the feature so that people didn't get confused with the name of the company.

But anyway, the way it works is we started with this in-memory, very fast in-memory database. It was distributed. It was high-availability replicated, and it was a row store database. And it used skiplists, skiplists index. And skiplists, the way it was built, it was locked-free. Using lock-free kind of coding techniques. And that made it very, very high-concurrency. So you basically could have lots of threads all hammering on the same thing without having to do any blocking. And this lets us do a lot of updates very quickly. So there's really great in-memory database, I think about as fast as you can make a database. But then you need to add persistence.

So the typical way you add persistence in a database is you write to a transaction log. And especially with the advent of SSDs, you can do that very, very fast. So basically, all you have to do is append to a log. And so, then we had an in-memory database that could do – Because we don't actually have to hit the disk for any reads. So we can still do kind of sub-millisecond operations. And then you have to write to the disk and you just append to the end of a transaction log or a replay log when you do updates.

And then the next step, and this is sort of where what enables us to do transactions analytics in the same place, is you don't want to have to replay the entire log every time. No restarts. So when we write out snapshots of the row store data, instead of writing it out the same format, we transcoded into a column store.

Column stores are generally very good at analytics because of the way they can compress the data. So you have to read less data the way you only have to read certain columns. And there's a bunch of other tricks you can do in a column store so that basically you only have to read a very small amount of data.

Now, column stores are generally bad for transactions, because they're very hard to update. So if you want to update something in the middle of a column store, you basically have to rewrite the whole file. And that's one of the reasons why a lot of these cloud data warehouses tend to be not very good at doing updates on both.

So with SingleStore, when you want to do an update, we basically just hoist the segment, so the block of the file that you updated, we hoist that into the row store in-memory. So now the memory portion of it owns that data. And then, basically, we can do very fast updates on that. Lots of updates. Once that **[inaudible 00:08:54],** then we can write it back to the column store.

So to the user, this is all seamless, and it all happens under the under the covers. But the impact is that they can get fast analytics and fast transactions. We built our own column store format, because one of the other things you need to be able to do to be a transactional database, because you need to be able to do fast point lookups.

So if everything fits in-memory, that's great. But often stuff doesn't fit in-memory. So we have basically seekable column store. So we have a hash index into our column store that lets you basically just go right to exactly where the data that you need is. So that also helps solves one of the problem with – Generally, data warehouses have a hard time doing lookups, because you have to read an entire block of data in order to find just the thing that you're looking for. So those are the two top tiers of our storage system.

The third tier of our storage system is object store. So we will write to S3, GCS, Azure Blob Storage, or Ceph, or Cohesity, or MinIO if you're on-prem. And that allows us to get separation storage and compute. It allows us to sort of scale to petabytes. Virtually, unlimited scale. It also allows us to do – Basically restore the data as the table as of a particular time, because we can store the full transaction log.

And then also, it allows us to do separation of compute and compute, where you can have two different compute clusters talking to the same data. An example of where this would be useful would be if you were using SingleStore for your application, as your primary application state, that's one compute cluster. But then you might be doing analytics in a separate compute cluster. So they both kind of talk to the same data. But there's no overlap in the resources, the hardware that's being used. And then you can also have other teams spin up additional – We call them workspaces. That allow them access to the underlying data.

**[00:10:54] JM**: When you have the multitude of underlying storage systems, it seems like it could lead to inconsistencies between the different layers. How do you resolve those consistencies or lock against them?

**[00:11:11] JT**: That's a good question. I think that's where a lot of kind of the secret sauce goes in. Especially when you consider it's not just a single node instance, it's also a distributed system, then all of that coordination becomes very, very difficult. We have a kind of a global versioning system that we can use to basically resolve to understand relative timing across the system. And then the thing that actually helps when you're trying to patch together the updated portions, the in-memory portions with the column store portions, is it's not done on a row-level basis. It’s done on a segment-level basis. So it's actually pretty easy when you're doing a query to basically skip over a segment from the column store and then kind of union that with the data that's in the row store. Now, obviously, it gets trickier than that, and then when you also have data on object store.

But for the most part, SingleStore is built so that higher levels of the storage system, which are kind of the lower latency levels of the storage system, rarely have to wait on the lower levels of the storage system. So most of – Kind of the protocols are all built so that most kinds of time-intensive work can be done asynchronously behind the scenes.

**[00:12:31] JM**: So if I understand correctly, there are, I guess, three places where you're storing a piece of data, typically. There's a in-memory caching layer. There's a database layer. And then there's like a bucket storage layer. Is that correct?

**[00:12:53] JT**: Yes, that's a good way to think about it.

**[00:12:56] JM**: Okay. Why do you need the bucket storage layer?

**[00:13:00] JT**: So as for two things. One is for kind of just general persistence. Like if you want to – Or durability. Generally, you kind of have a higher durability at that layer, from in terms of disaster recovery, things that can go wrong. The other thing is it enables us to do separation of storage and compute so that we can actually blow away the compute cluster, which includes the local SSDs, and then recreated again. And basically, all of the data is still stored at the object storage layer.

So scalability, durability, and is also what kind of allows us to do, as I mentioned, the separation of compute and compute, where you have multiple different compute nodes all synchronizing against the object storage layer.

**[00:13:49] JM**: Okay. So can you talk a little bit more about what you use for the caching layer and the database layer? Are those all custom-written database infrastructure?

**[00:14:04] JT**: Yeah. I think one of the interesting things about a lot of the kind of databases these days, it seems like there's lots of database companies and lots of kind of a proliferation of databases. And a lot of them they sort of just take – They'll take Postgres and rewrite the storage system, or MySQL. And they'll rewrite a piece of it. And then sort of to help it scale, to help it deal with – To solve some particular problem, SingleStore and MqmSQL were really built from the ground up as a kind of a distributed in-memory database. So all this stuff, we've built ourselves. It's one of the reasons that it's taken us a long time, the company is 10-years-old, in order to build these things. So we built our own hash indexes, our own column storage format, our own row store, even our own RPC system in order to sort of really optimize the performance.

**[00:15:03] JM**: So if you want to build a search index against a SingleStore database, what's the best way to do that?

**[00:15:14] JT**: So SingleStore has a built-in CLucene. We took CLucene, which is an inverted index. And I believe it's the same thing that powers Elasticsearch. We ended up rewriting a bunch of it to work well our underlying infrastructure. So you basically can just – You can create a full text index on top of SingleStore, and it'll use that CLucene infrastructure. And so you can do the same sort of full text search engine type queries, queries over all of the fields, for example, that you would do something like Elasticsearch. But you can do that also in a relational database.

**[00:15:55] JM**: Okay. Got it. Can you talk about the query semantics of SingleStore? I guess there's a lot of different ways you can query the database. I guess maybe you could talk about how to optimize query semantics, or maybe talk about the query parser or the – Just maybe let's walk through the architecture of a query and the processing of a query.

**[00:16:22] JT**: So just one thing is SingleStore is MySQL compatible. Almost completely MySQL compatible. There's a lot of weird things. And MySQL is around for a long time. So there's a lot of like weird kind of cobwebs in the corners. And so we didn't necessarily implement all of those pieces. But, in general, it's datatype compatible with MySQL. It's even wire compatible with the MySQL protocol. SingleSTore has some cool things on the query processing side. I tend to geek out about the storage side, because I came from the storage world more recently kind of than the query processing world. But some really cool stuff in the query processing and optimization.

So there's one which is vectorization. So kind of starting from the bottom-up. But if you're reading from a column store, you're generally reading a column once in the same – The same field is listed multiple times. So it's actually pretty easy to use vector instructions from CPU vector instructions, because what you have is a vector of a particular set of data. And so we do some of vectorization using the Intel instruction set. And we'll even do operate directly over the encoded data format. So we don't have to decompress it. We can actually operate on the data in place. So that's one of the ways that we get good performance.

The other thing that we do is we do query compilation. So if you have a complex query, maybe your query is doing like computing, some value or truncating a date, etc. Instead of interpreting that, we can compile that to x86, raw x86, so that we can execute that as fast as possible. And then we cache those query plans. Soo the next time you run that query, we don't have to recompile it. It's already there. And we can execute that at basically raw machine speed.

Other kind of interesting things about SingleStore. So in a distributed database, the hard part is what happens if one node needs data that's in another location? Happens a lot of the time. Generally, the way you do that is either broadcast or shuffle. Broadcast is basically you take data and you send it to all the nodes. Shuffle is basically you put like with like and you shuffle things around. So SingleStore can do both of those. If you want really low latency, that's very hard to do. You basically have to have the data already where it needs to be.

And so we have two ways of doing that. One is something we call reference tables. Reference tables is sort of like a pre-broadcast where we basically make a copy of a certain table in every node. And that way, when you run the query, there's already a hash table built on top of it. So you're able to basically operate – Do joins with that table without having to do any data movement. The other one is co-partitioning. And it's a reasonably common thing. But I think most of the data warehouses don't do this, which is if you want to do a join against two tables, and they're both partitioned in the same way, then you don't have to do any shuffling or any data movement, because basically you can just scan through those on a single note. And in fact, SingleStore lets you define a sort order as well. And the sort order is the same, then it's a very simple merge operation.

**[00:19:52] JM**: And is there ongoing work to optimize the query system? Or does it feel like it's already sufficiently optimized?

**[00:20:05] JT**: You can never get sufficiently optimized. Like, there's always something you can do. It's also what makes actually benchmarks pretty interesting, is because benchmarks, something like TPC-H, TPC-DS. TPC-H I think is 25 queries. TPC-DS is 100 queries. And each one of those basically requires the optimizer to do something hard. So often, companies, database vendors will kind of – You run TPC, TPC-H, you're like, "Oh, everything is good, except query 13. Query 13 is terrible." And then so you figure out, "Okay, what is the optimization you need here?" And it turns out that you're not doing a push down in this case, or you're not moving something – Pushing it below the join, etc. So there's lots of different tricks. And to some extent, people end up doing benchmark engineering where, basically, you tune the optimizer to the benchmark, and that sort of when it doesn't work, or when it works poorly. I even have heard apocryphal stories of a database vendor that I won't name that basically hard coded the query plans for kind of the TPC benchmarks into their query planner, so that anytime they saw those, they would kind of jumped to those plans.

SingleStore, we don't do that. But we do kind of recognize that, "Hey, some of these queries don't work as well, because maybe our optimizer is –" We've got these 150 optimizations or kind of tricks that we do. We're missing a couple of the other ones. And so you kind of continue to crank through those. And some of those are from things that customers find, or some of those are things that you're just sort of like – It's sort of well-known in the database literature that in order to solve X problem, you have to do Y.

And, of course, in addition, I think a lot of the traditional database literature and database optimization was designed around kind of single box, single box databases, the scale up databases, like the old school, or the Oracle, the MySQL, where the decisions that you'd have to make with data could be moved around a lot more easily. And then trying to do some of those same types of optimizations in an explicitly distributed world is harder.

**[00:22:19] JM**: Tell me about what goes into standing up an instance of singles store. And maybe you can go into what elements of the database are partition from – I assume every database instances is partition. But there's partition from other databases. But I imagine there's some economies of scale to running lots of databases across a single company hosted databases. Yeah, I just like to get a sense of what goes into an instance.

**[00:22:50] JT**: Sure. So SingleStore, you can use it in two ways. One is as database as a service. That's what we run on. AWS, Azure and GCP. To stand up as a SingleStore instance, you just go to the SingleStore website and you click on the size that you want, and you spin it up. And very shortly, you will have a fully running SingleStore database of the size that you chose. Of course, under the covers, it's a lot more complicated than that. And under the covers, we use Kubernetes to manage our fleet of SingleStore instances. And we do some bin packing of those SingleStore instances. But each one of those instances really is discrete. At this time, we don't do any multi-tenancy. At some point in the future, we may do some multi-tenancy, perhaps on kind of a lower tier of service. But I think customers, especially, enterprise customers, they get grumpy if they're ever going to be operating – If they're going to be sharing hardware with anybody else. And so we make sure not to do that.

The Kubernetes operator that we use is we build a custom Kubernetes operator. We also ship that Kubernetes operator. So you can run single story yourself. You can run SingleStore via Kubernetes. We have customers that are running 1000s of SingleStore instances inside Kubernetes. And that basically gives you kind of a higher level of abstraction. So when a node fails, you can do X, Y and Z to scale up, to scale down. Provide some of the elasticity and ability to – So for a company that wants to run on -prem or wants to manage SingleStore themselves, they have the tooling and the ability to kind of manage a fleet of SingleStore instances.

Actually, one of the things we're adding very soon is the ability for those to then connect back to our control plane so you get the same SingleStore portal so you can have your on-prem instances and your AWS or GCP instances all sort of managed in the same method. And you can do the same sort of types of elastic scaling. And you get kind of the same new features. It is a distributed database. So it is kind of harder to run than just something like MySQL.

To simplify that, we have a Docker container. So there's a Docker container for the free version. So can download the Docker container to your laptop. You can just run SingleStore on your laptop. Running it on bare metal is a little bit trickier, just because you need to set up – There's a bunch of things you need to set up. But we've got tools and scripts to help you with that.

**[00:25:31] JM**: Cool, that makes a lot of sense. And has the economics of the deployment improved since you move to Kubernetes?

**[00:25:45] JT**: Yes. So we started on Kubernetes kind of from the earliest stages, from the early stages of our cloud offering. I think if you're going to run 1000 SingleStore clusters, having to manage those yourself is would be prohibitive without something like Kubernetes. We did have our own set of tools for deployment, and setup, and configuration, etc. And we still support those. But I think long term we see the world moving into kind of a Kubernetes world, just because it – As I mentioned, it gives you a higher level of abstraction. I don't think in these days anybody wants to manage a distributed database themselves. Like I think they want like something higher level, something where they can say deploy, deploy on X nodes, scale it up to Y nodes, scale it down. When node failures happen, there's basically a process and handlers for that rather than having to sort of do things more manually in sort of the kind of, I would say, the more pre-Kubernetes way of doing things.

**[00:26:48] JM**: And can you tell me more about, I guess, the internal infrastructure used to manage and monitor SingleStore databases? Like, when you look at the cloud product, maybe you could tell me some about what the hosted services that you use are to managing those SingleStore instances and the work around platforming them.

**[00:27:15] JT**: We use Grafana for our monitoring. And we provide Grafana to customers who want to do monitoring. We have a bunch of integration with enterprise, things like SSO, and

Okta, etc. I don't know what else we have beyond that. I think it's actually an area that we want to invest in. And I think being able to monitor your systems is important. And it's important to be able to do at scale. And a lot of the kind of initial things that we're doing is focusing, that we can monitor and understand the systems that we're running, and that we can automate those and make those work well.

And sometimes what takes a second, takes a backseat to that, is building systems that then customers can use for observability and understanding. Because I do think that from the on-prem version, you want to monitor the low-level things. You want to monitor CPU utilization, memory utilization. What happens when you run out of disk? Those kinds of things.

But for the managed service, you never want customers to understand those or even see those. So you want to basically translate things into kind of a higher level of abstraction so that customers understand stuck queries, they understand query volume. They understand the ingest pipelines. They shouldn't have to know or care about our disk utilization, because our SRE should be handling that.

**[00:28:43] JM**: Can you talk about the applications that people build on top of SingleStore, and particularly regarding the data layer, or in terms of like data intensive operations? I'd love to hear kind of a comparison to data warehouses and streaming systems, and I guess both from a performance level and an API level.

**[00:29:17] JT**: Sure, I'll give a couple of examples. One, I can't always say the names of the companies that are doing them. But one interesting one is Fathom Analytics. They're sort of building a privacy sensitive alternative to Google Analytics. They're a super interesting company. And they started out, kind of ran into some limitations with Aurora. And they wanted an alternative. Started using SingleStore. Found that it was great. And then they realized, "Hey, I don't need – I've got–" I believe it was that they were using Elasticsearch. And I was like, "Hey, I was using Elasticsearch for these certain use cases. I don't need to do that anymore. I can use SingleStore." They were using Redis because they needed a cache. They got rid of Redis.

The founder of the company, Jack Ellis, wrote a series of blog posts kind of going through all of his kind of decision-making and why he decided to do this. So you don't necessarily have to take my word for it, like this is a pattern that exists. And that's something that people need. Like, I worked at a startup company before I went to Google actually. I was building something that was very similar. It had a MySQL database, sharded MySQL. We want to do geospatial query. So we use Solar, which is sort of a precursor of Elastic. We used Memcached. It's kind of a precursor of Redis. And it was a nightmare to kind of keep all these things moving and working well together. I was the one that was working on that that part of it. And so something like SingleStore would have been great. So that's sort of on the data-intensive application side.

On the analytic side, there's a couple of different buckets. One is the fast dashboards. So we have just a number of customers, one very large computer company that doesn't like – Gets really grumpy when you use their name. But they have an executive dashboard. Does 800 queries per second. So first of all, like, 800 queries per second is something that it would be prohibitively expensive on most data warehouses. Also, these all have to succeed in under a second. And so, they have an SLA of – A very short SLA. And so they use SingleStore. And they started using SingleStore actually for this workload, because they couldn't find another database that would do this. And then they started finding a bunch of other things that like, “SingleStore, we can use this in this other team. SingleStore, we can use it for this, this and this.” I think, every quarter for the last six quarters, they've actually expanded their SingleStore usage.

It's definitely nice coming into the end of the quarter. Like we always see their upsell. Also, a large ride sharing company uses SingleStore. They also don't like us to use their name. But they use it for market segmentation. They have a two millisecond SLA for segmentation. And SingleStore can hit that two millisecond SLA with better than 99% of the time. A major bank uses SingleStore for fraud detection on the swipe. This was on the swipe. It's got to return in sub-50 milliseconds, because the entire transaction can only take 200 milliseconds. And they only have 50 milliseconds for fraud detection. But the interesting thing about that is that that's an analytic query. So like the analytic query has to not just do a lookup or not just do a quick update, but it has to use data about what's going on in the world to determine whether this particular transaction is fraud. That's just a couple different examples of how people are using it.

**[00:32:39] JM**: And why don't you use the terminology of data warehouse? Like, maybe you could talk a little bit about how, from the OLAP side of things, SingleStore compares to a data warehouse like Snowflake?

**[00:32:57] JT**: Sure. So it's great question you can use SingleStore for data warehousing use cases. And we have people that are doing that. But if somebody is deciding between Snowflake and SingleStore and they want pure data warehousing, they're going to choose Snowflake. So we try not to set up those comparisons. They just have a lot more features. They have a lot more analytics features, a lot more analytics ecosystem. There was one large hedge fund. The guy was like, “Look, I really want to use SingleStore. But all my data that I'm getting is in Snowflake. And so it would have been difficult.”

Now, on-prem, actually SingleStore is a – We find that people are migrating from on-prem data warehouses into SingleStore. Some Teradata, Vertica, Netezza, etc. And they do it as like a two-step modernization. So step one is you move to SingleStore, and you can do that on-prem. And step two, they move to cloud. And that way you don't have to do all of the changes at once. And often, they have they have on-prem hardware, etc., that they need to depreciate or that they want to they want to continue to use, or they wanted to make sure it's low-latency access to their applications, or it's behind their firewall. So there's lots of reasons that people will want to use SingleStore on-prem.

I guess the last one is just I think that even most of the cloud data warehouses, the Redshift, BigQuery, Snowflake, etc., don't even call themselves necessarily data warehouses anymore. They call themselves, “We're data platforms.”

I used to talk about data warehousing is not a use case and kind of what the people – What customers needed was something actually broader than just data warehousing. And if you think about it as just sort of solving the same problems that you solved 20 years ago, then you were unnecessarily limiting what you could do with your data.

**[00:35:01] JM**: Is there a work you could do over time to augment SingleStore with like full-on data warehousing functionality? Or do you think it's just prohibitive when if you're trying to do that much multi-modeling?

**[00:35:19] JT**: Great question. I think in the fullness of time, we'll get there. One of the difficulties of sort of trying to be this broad multi-model database is that you don't want to try to be everything to everyone. And you have to focus on certain areas. And so, we have decided not to push too hard against directly competing with the data warehousing vendors.

That said, a big use case – One of our three main kind of motions is augment, and sort of data warehouse augmentation. Because a lot of the time, somebody is using a data warehouse, or they're using a database, and there's a subset of workloads that are not fast enough. Maybe they can't land data fast enough. Maybe they need super high-concurrency. Maybe they need they need really low-latency.

And so we find a lot of customers using kind of either a Lambda architecture or land data in both places, where they landed in SingleStore, because they can do the updates very fast, and then they push the data back to another data warehouse. But we started out as sort of augmentation.

In fact, augmentation – So when I was at Google, we were really, really focused on big take outs, like taking out Teradata, taking out these kinds of other big systems. And those are massive undertaking. It's like you have to worry about – People have like 10,000 scripts that run against Teradata. They have all sorts of applications. And you want to make sure those things don't fall over, or they don't fail. And to try to do that all in one step is really, really difficult.

So I think the better approach, and the approach SingleStore is taking is, "Hey, we will augment the thing that you're doing with something that will help you either scale better, or get better performance, or latency, or better kind of simplicity." And then over time, "Hey, if you like SingleStore that was working, you may find that more and more of your workloads stay in SingleStore." And it's also a pattern that we see. People find it more cost-effective than some other systems. Sometimes people complain that other systems, it's easy to have other – Cloud data warehouses are – It's easy to have runaway costs, etc. And so SingleStore is often a good option if you're concerned about your TCO.

**[00:37:41] JM**: As you look at across the newer database market, I think some of the more innovative and successful databases I've seen are Rockset, PlanetScale. I'm trying to think of others. And then there's, of course, a variety of newer data warehousing technologies, like, I guess Presto has really been productized by Starburst. And if Pinot being productized by StarTree. Are there any sources of inspiration that you take when you look towards the future of SingleStore?

**[00:38:21] JT**: Absolutely. But I think we're trying to take our inspiration from customers rather than competitors. I think a lot of people are chasing sort of the decay of the data lake. I think there was a time when everybody needed to have Hadoop. And if you want to Hadoop, you have to have a data lake. And the data lakes quickly became a data swamp.

And I remember, when I was at Google, we had this customer advisory board with some of the execs, like the CTOs of Fortune 10 companies. Like some of these biggest companies. And we were trying to sell them actually on a data lake idea. And it's like, "What do you think about data lakes?" And like that was the one thing that all these people agreed with, was, "Oh, my God. I hate my data lake. And like, I wish I could get rid of it. Or I wish I could –"

And I think that it's sort of nice to have a dumping ground in a place where you don't have to worry about it, and it's cheap, and I can store all my data there. But if you're not careful about it, then it just becomes a mess. And there's a bunch of tools that are data catalogs and mechanisms for dealing with that.

And I think that we're seeing these query engines, things like Presto, being able to query the data where it sits rather than to have a manage storage system. And I know that the Databricks folks, for example, they talk about how great Delta Lake is because it gives you the advantages of a managed system with the kind of the cost and fungibility of a data lake. And kind of my view is that the database storage, or data warehouse storage, managed storage system, is strictly better than a data lake storage. I think that's because if you have control over – Like, a lot of the performance of a database is due to how the data is laid out. And if it's the customer's responsibility for laying out the data, then they're not going to do it right. I mean, it's something you have to sort of constantly groom and maintain is sort of how the shape of your data is for how the queries are being done. So from a performance perspective, if you have to give customers the physical access to the data, that also causes all sorts of problems.

First of all, from a governance and security perspective, it becomes very hard or impossible without bizarre hacks to do things like column-level security and row-level security. Like if I give you access, physical access, to a file, I can't tell you not to read parts of that file. But also, from a performance perspective, like if you want to be able to do fast updates, for example, and you're giving people access to parquet files, then it's going to be like you can't do the same sort of updates because you have to rewrite parquet files each time. And so maybe you can have some other separate system and they have to merge the system. But then you have a really complex client side. And then synchronization is hard. And like, there's just a bunch of stuff that I feel like is sort of a hack in the Data Lake storage case.

Anyway, I believe strongly in sort of the data warehouse or sort of managed data storage is going to be the way of the future. But there's also certainly some interesting things kind of in the other side. And a lot of smart people believe that I'm completely wrong. And I guess we'll see which one wins out in the end.

**[00:41:54] JM**: Well, that sounds like a good place to close off. Jordan, thank you so much for coming back on and sharing your broad wisdom.

**[00:42:00] JT**: Thanks so much. It's been a great conversation.

[END]