EPISODE 1260

[INTRODUCTION]

[00:00:00] JM: Columnar databases store and retrieve columns of data rather than rows of data. Each block of data in a columnar database stores up to three times as many records as row-based storage. This means you can read data with a third of the power needed in row-based data among other advantages. The company, Altinity, is the leading enterprise provider for ClickHouse, an open source column store analytic database. Now a fully managed service developed and operated within Altinity Cloud. Altinity only bills for the compute, storage and support that is used. They provide enterprise support for analytic applications like tuning queries, Kafka support, and ClickHouse bugs, and their ClickHouse clusters run with out of the box security and privacy.

In this episode, we talk with Robert Hodges, CEO at Altinity. Before becoming CEO at Altinity, Robert worked as a senior staff engineer at VMware and was the CEO of Continuant before that. We discuss databases and data warehousing, ClickHouse, and how Altinity helps customers create enterprise analytic applications.

A few announcements before we get started. One, if you like Clubhouse, subscribe to the Club for Software Daily on Clubhouse. It's just Software Daily. And we'll be doing some interesting Clubhouse sessions within the next few weeks. And two, if you're looking for a job, we are hiring a variety of roles. We're looking for a social media manager. We're looking for a graphic designer. And we're looking for writers. If you are interested in contributing content to Software Engineering Daily, or even if you're a podcaster, and you're curious about how to get involved, we are looking for people with interesting backgrounds who can contribute to Software Engineering Daily. Again, mostly we're looking for social media help and design help. But if you're a writer or a podcaster, we'd also love to hear from you. You can send me an email with your resume, jeff@softwareengineeringdaily.com. That's jeff@softwareengineeringdaily.com.

[INTERVIEW]

[00:02:05] JM: Robert, welcome to the show.

[00:02:06] RH: It's great to be here.

[00:02:07] JM: I'd like to start by talking a little bit about the domain of data warehouses. So we've done a bunch of shows recently and in the past about data warehouses like Snowflake and Redshift, and Google BigQuery. Can you just tell me about the history of the data warehouse and what role it plays today?

[00:02:28] RH: Sure, I think that's a great question. So the data warehouse history goes pretty far back. It really began in the 80s when people began to design systems like Teradata and Sybase IQ. So these were systems that began to address what was turning into a new breed of application where people needed to scan large quantities of data to answer open-ended questions. And by open-ended, I mean that you had, for example, sales data, or customer data, or some other kind of interesting information about your company, and you wanted to ask strategy questions about it, which would mean that you would come in. You wouldn't know exactly what you were going to need to know in order to answer your question, and then you'd like to play around with it. These are problems that traditional relational databases of the time, which would be late 80s, early 90s, we're just not very well suited to solve. So this new breed of application developed and has evolved over a period of decades. So for example, the first products like Teradata introduce things like parallelization across nodes, extending to things like Vertica with very efficient column storage and compression. And then, of course, to the cloud data warehouses that we see today, which are things like Redshift, which was really the Pioneer here, and then Snowflake.

[00:03:47] JM: And tell me a little bit about the modern applications of the data warehouse.

[00:03:51] RH: Yeah, that's another great question. So what's happened is, initially, when databases were embedded in applications, they were used for transaction processing of one kind or another. That was the beginning of, really, the RDBMS revolution that we saw particularly in the 90s and into the early 2000s. What's happening now is people don't have a problem with data to run their sites. What they do have a problem with is presenting information that the software is collecting in a form that's easily digestible and understandable by users.

So as you look across applications that are being developed today, particularly SaaS applications, being able to present embedded analytics is becoming a fundamental feature of these apps. In fact, in some cases, for example, marketing software, it is the application. So what that means is that these applications are all trying to connect to these vast pools of data that they're collecting, be it market data, or web analytics, or network management records. And they want to be able to deliver to users a view of these pools of data that allow them to gain insights and run their business. So the data warehouse is a fundamental part of that problem. And what people are now trying to do is connect the applications to the backend data warehouses and build these applications.

[00:05:15] JM: How do the various data warehouses compare in architecture?

[00:05:21] RH: That's an interesting question. And one of the reasons is that there are so many ways that you can attack this problem that it's hard to say this is the one way to do it. But I think there are some general trends that are shared by almost all products that we see today. One of them is the adoption of column storage. So this was something that people realized early on that early databases would store data for a row in a table in a single location. This turns out not to be particularly effective for applications that are trying to read large amounts of data. And the reason is that even if you only want to know a small amount of information from each row, you have to read the whole thing.

So column storage is a feature that's almost universally adopted in data warehouses today. A second feature that is broadly adopted is compression. That's something that follows from the column store, storage format, and works much better than it does in row stores. A third feature, which is really fundamental, is SQL. And this is something where in some of the newer data warehouses that we've seen initially started out as NoSQL databases, but quickly realized that they needed to adopt this because It's known by users and it's so widely used by tools. So those are three sort of general features that you see in every architecture.

I think the newest thing that we're seeing at least in the last generation of data warehouses is of course running in the cloud and taking advantage of things like, first of all, object storage, which allows you to have vast amounts of cheap, replicated, storage, and then separation of compute and storage. So those are sort of new features in modern data warehouses that allow you to build new applications and handle larger amounts of data.

[00:07:08] JM: You work on a startup built around ClickHouse. Could you give me an abbreviated history of ClickHouse?

[00:07:15] RH: Sure. ClickHouse is really one of the newest entrants in the data warehouse market. And just in a nutshell, to describe it, it is the first SQL data warehouse that is open source, and can also play with the big kids. And by that I mean, the proprietary solutions. The history of it is pretty simple. It started in 2008 at Yandex, which is a Russian tech company. They were looking for a data warehouse that was very scalable, easy to run, didn't have encumbering licenses. They couldn't find it. So they wrote their own. They used it to underlie a web analytic product called Metrica. At the end of eight years, it was so successful, that it was used throughout the company, and they decided to open source it. There are a variety of reasons for that, but ranging from make it easier for people inside the Yandex to get at it to just showing off the world that they had some interesting technology. And so in 2016, they open sourced it under Apache, and that was laid the foundation to build this huge community around it that we see today.

[00:08:16] JM: What was ClickHouse used for when it was completely within Yandex?

[00:08:22] RH: The first use case was web analytics. So it was designed to power a product called Metrica, which is very similar to Google Analytics. It's very popular outside the United States, used worldwide in fact. And so that was the beginning use case. But what happened was that it was sufficiently powerful at doing that. And it was a data warehouse spoken language, or around SQL similar to MySQL, but it turned out to be so capable that they then applied it to other use cases like observability, analysis of network flow logs, log analysis, things like that. So by the end of those eight years, it was actually used for a wide variety of applications through Yandex. And as a result, that features to support all of them.

[00:09:09] JM: How does a data warehouse, from your point of view, fit into the overall emergent data architecture? Like in today's world, how does it fit in relative to the data lake? Are you a believer in the data lakehouse architecture? Give me a little bit of your point of view on the macro data infrastructure perspective.

[00:09:31] RH: Sure, that is a difficult question to answer in a general way, but I'll just give you my take on it. And it's based on where we see usage. So for ClickHouse, the place that the people are particularly using it is for built-in app analytics where you need to have very fast, very consistent response. And this could be something – I'll just give you a concrete example. You have somebody managing video downloads. There's actually a company called Mux that does this. And one of the features of their service is that they offer real-time analytics. So that as users are consuming these videos that they're serving up, you can actually see statistics on what's happening, like where are they getting excessive rebuffering? Are there particular areas where it looks like there's not sufficient network capacity to download quickly?

Data warehouses are essential for building this kind of application. And actually, as you look across applications, it's not just these corner cases like content delivery network management, but it's a whole wide variety of applications where people want to see more or less in real-time what is going on in the world and get quick answers. Data warehouses answer this, because they store the data, first of all, in this optimized storage format, column storage with high levels of compression. And they're also able to efficiently apply large amounts of compute and memory to the data so that they can answer questions quickly. I haven't actually personally built applications that use lakehouse. But I think, in general, as you read off data lakes, these are interesting for applications where you have very large amounts of data. But the problem is that it's very difficult to get consistent performance from them, because you're basically consuming data that comes out of things like S3. So the data is not optimized necessarily for consumption. So you need to go through steps to actually crunch it and then run processing on.

So I think there's definitely a spectrum here where the data warehouses are the pools of data that people need quick access to things like data lake or for longer term batch analyses where people don't care about having quick answers, but want to scan very large amounts of data and do, for example, run machine learning on it or something like that.

[00:11:58] JM: There is a pretty regularly cited paper from Michael Stonebreaker a while ago about the age of one size fits all is over, basically being about the rise of domain-specific databases. He talks about NoSQL and SQL databases, and I can't remember what other types he talks, but maybe graph databases. Are the tradeoffs in different data warehouses, are they different enough? Are they severe enough that an organization would ever want multiple data warehouse types?

[00:12:31] RH: Yes, I believe so. So first of all, I think Stonebreaker is totally right on this point that the database market in general is fragmenting. And one of the fundamental reasons for this is, in order to answer questions quickly, you have to have data organized in the correct way. And that goes back to why, for example, processing on data lakes tends to be kind of slow, because the data is not optimized for the questions that you're asking.

So first of all, in order to answer open-ended questions quickly, you do need columnar storage, high-compression, very efficient application of compute. Beyond that, there are some pretty significant differences in the data warehouses themselves. So for example, ClickHouse, the data warehouse that we work on, has the property that runs just about anywhere. So that means that you can actually run it on an Android phone. But you can also run it in containers. You can run it on Kubernetes. You can run it in racked equipment, you can run it on VMs.

So what that means is that for people running the data warehouse, they can then just make a choice of where's the appropriate place to run it based on economics, security, and other requirements. There are other data warehouse architectures, for example, Snowflake that just run in the cloud, and will most likely never run anywhere else. So what that means then is when organization, if you have different requirements, Snowflake may be appropriate for one, ClickHouse may be appropriate for another.

[00:14:03] JM: Have you seen applications that use ClickHouse on a mobile device?

[00:14:08] RH: It was done as a demo. That's actually a blog article on the ClickHouse site. It's really a proof of concept. What we do see – I think, actually, if I were a developer, I would use DuckDB for that, that it's sort of an analytic equivalent of SQLite, and it looks really great. We're all kind of fascinated by it. What we do see with ClickHouse is we see a lot of embedded applications. Just looking at our customer base, easily, 10% or 15% of the customers that we have are doing embedded applications. And for example, they'll ship appliances which have

ClickHouse embedded in them. That's very popular for applications, for example, that do network management.

[00:14:47] JM: Simple question. With all the popularity around Snowflake, how do you compete with a different solution?

[00:14:55] RH: We actually compete really well against Snowflake, and I think there's three areas where ClickHouse is really outstanding. The first is speed. And by speed, I mean, the ability to answer a generic query in a second or less and be able to maintain that level of performance regardless of the amount of data. So just simply by adding resources, we can actually get response times down to 20 milliseconds. This is something Snowflake just doesn't do in a consistent way.

The second thing is we're very portable. So if you need to run somewhere other than the cloud, we have the answer. For example, we can run in Kubernetes, which it means that we can run across a wide range of environments. The third thing is accessibility. And I think in a way that this is the biggest difference, because since the software is open source, as a developer, I can install ClickHouse software in 60 seconds. It's just by running a single Docker command, or running two commands on Linux. So that ability to put this into the hands of developers worldwide is a really fundamental distinction between a database like ClickHouse and an existing proprietary database like Snowflake.

[00:16:08] JM: Let's go through some of the semantics of ClickHouse. Take me through a write and a read to ClickHouse.

[00:16:17] RH: Sure. So writing is pretty simple. You do SQL inserts, but it's a little bit different from doing an insert into MySQL, or Postgres, or another row store. We tend to insert relatively large blocks of data. And we do it in one of three ways. We can just have files just blasted up through client software, say, 50,000 rows at a time. We can read from S3 compatible storage. And we can also read directly off Kafka queues. For large systems, I would say at least 50% of them are using Kafka to supply data.

Every time one of these blocks of data hits ClickHouse, what we do is immediately create what's called a part where we sort the data, we organize it into columns and write it. Another thing that's really important in ClickHouse is it has the ability to do materialized views, which are reduced or reorganized forms of data. And the materialized views are immediately updated every time one of these blocks hits. So that process means that as you ingest the data, for example, from Kafka, that as each block hits, it becomes, first of all, immediately queryable. Moreover, if you're competing aggregates from it, those are also immediately queryable. So that's the insert process.

The Select process is quite straightforward. You can set up ClickHouse in a variety of ways. But for large systems, the most common organization is to be sharded. In other words, have the system, the data broken up into disjoint pieces and replicated. So each of those pieces has multiple copies. And for reads, what you'll do is you'll come in through what we call a distributed table, which knows where all the shards are. You'll run a SQL query, and the distributed table will automatically break it up into pieces, distributed out to one replica for each shard, do the basic data collection on it, and then aggregate it at the point of entry and hand it back to the calling application.

[00:18:21] JM: How does ClickHouse think about reading stale data? Is that not much of an issue since this isn't really thought of as a transactional database? Like does it have particular sensitivity to like reading the data that may be a little bit stale? Or I guess I'm just curious about how it compares to like a transactional database in terms of how sensitive it is to the latency?

[00:18:43] RH: Well, I think that there's two issues here. One of the things that we are very sensitive to is latency. So the ability to insert something and have it be immediately curable, including in its aggregate forms, and that is something that we place a very high priority on. What we don't worry about so much is transactions. So the ClickHouse transaction model is not a full ACID compliant model, which, for example, has isolation and full durability across multiple tables, even a DDL. That's just not something that's very important in data warehouse applications, because we're primarily writing data in large chunks. And then we have very few updates on it. So a lot of the things that the transaction model gives you are not necessary.

As far as staleness of data, that's really in the eye of the beholder. So what will happen is that applications can make decisions, for example, how long they want to keep data around, and ClickHouse has the ability to, for example, trim data sets automatically so that old data that's no longer interesting is just removed automatically. Meanwhile, you can keep aggregates. In other words, down-sampled copies of the data. You can keep them in your materialized views for as long as you need.

So really, what we're all about is less about worrying about staleness of data, then very fast access to the information that you have, and then the ability to keep it around in an appropriate form for as long as you need it.

[00:20:16] JM: Can you say anything else about the eviction policies of ClickHouse?

[00:20:20] RH: Yes, absolutely. So we have what are called TTLs, or Time to Lives. This is a great feature that I first encountered when I was using MongoDB maybe 10 years ago. And the idea is that you put time to live on, for example, a row in a table. And what ClickHouse will do is you say, “Hey, I want this around for 90 days.” Once that 90 days is up, ClickHouse will, at its leisure, find that row and drop it from the table. So that's a really critical feature that was put into ClickHouse very early on.

Now, what we've done though is extended it so that it does much more than just dropped data. So for example, we have TTLs that allow you to put data into hot storage, for example, NVMe, SSD, and then use a TTL to move it to a different storage format, such as network attached storage, or even object storage, based on the amount of time that it's been sitting in the table.

Another way that TTL is can be used, which is quite interesting, is for aggregates. We can re-compute the aggregates, say, after 30 days or three months, so that we change the granularity and don't store as much data. So as a result, if you go and look at the aggregates in the table, the hot data as a granularity of maybe a minute, but you could actually extend that to hours or days or weeks as the data becomes older. These are all ways that we deal with the aging of data, which is a really, really important consideration when you're dealing with large data sets, where the most recent data is the most interesting, but you still want the ability to go back and scan the old data and answer questions there as well.

[00:21:58] JM: What's the replication policy for ClickHouse?

[00:22:01] RH: It's multi-master, eventually consistent. So when you have a set of replicas, there is a Zookeeper that's keeping track of things. Actually, that's being replaced. And ClickHouse will, like other products, such as Kafka itself, we will have our own consensus managed directly in the server. But basically, you can go to any copy of a replicated table, update the data there, and then it will, within usually a very small period of time, be automatically replicated out to any other copy.

There's also a really interesting feature that ClickHouse has that in these replicated systems, sometimes you'll get failures. And you'll end up having to resubmit a block of data, and it may land on another copy because your load balanced. So ClickHouse has what's called de-duplication, where we keep track of the hashes of the blocks that have been added. And if you add another block, or if you add a block, again, by accident, say because of a failure, we’ll just automatically discard it.

[00:23:02] JM: Tell me a bit about how you deploy ClickHouse. Like what's the server infrastructure look like? Are you deploying it on top of Kubernetes? What's the typical deployment look like?

[00:23:15] RH: I think there's really three types of deployments that we're seeing in the market today. So the first and the original one was just directly deployed onto hosts where you would manage the host individually. And those hosts could either be rack equipments. So running on bare metal, or they could be VMs in an environment like Amazon. So that was the initial sort of self-managed deployment.

The next thing that we saw, and we actually helped along because of some of the development we did, was to deploy on Kubernetes. So there's a quite capable ClickHouse Kubernetes operator, which allows you to spin up ClickHouse on any reasonably modern Kubernetes cluster. So what that means is you then give Kubernetes the responsibility of allocating things like compute and storage for you. And then you can just plunk down the cluster anywhere that Kubernetes runs.

The third is running in the cloud. And for us, that's something that we do. It's a major offering for us. It’s Altinity.cloud. And underneath, what we're doing is actually spinning up Kubernetes clusters for users and then using that as, if you will, our control plane to run the data warehouse clusters. We are not the only cloud service out there. By my count, there're at least six others. But our goal is to make this the preferred way to run ClickHouse us in the United States and European markets.

[00:24:40] JM: Tell me a little bit about the process of running Altinity and some of the go-to-market challenges or the engineering challenges.

[00:24:50] RH: Sure. I think the biggest challenge is just ensuring that ClickHouse itself is popular. Any open source business or business based on open source software is going to depend of the popularity of the underlying project. Now, fortunately for ClickHouse, that's becoming less of a problem as time goes on. The community, which started back in 2016, when the software was first open sourced, it was very much a niche quite popular in Eastern Europe, because people had heard about what Yandex was doing. So that's something that's always on our mind, is ensuring that the community is continuing to grow and that people, or developers in particular, are coming to ClickHouse and can easily get started with it.

Beyond that, the challenges that we have as a company is, first and foremost, just telling people who we are. So every software company has the issue of creating a brand and making sure that people who potentially want to consume your products learn about that brand. We still find today, even though we think of ourselves as pioneers on ClickHouse, we're still constantly running into customers that haven't heard of us. So one of the things that we do is just ensure that we're producing a steady stream of content that is interesting to developers, that we’re present in the various communities that consume ClickHouse and sort of to grow that community of people who are interested in doing business with us. The third thing is just the technical challenges. And in this case, our business is a little bit different from many other open source businesses in the sense that ClickHouse is truly a community-owned project. There are hundreds of people across the world who have committed changes to ClickHouse itself. Thousands more who participate by submitting issues or otherwise, helping provide information to the rest of the community.

So for us, the key technical challenge is not so much to evolve the database itself, although of course we're deeply involved in that, but really to build the cloud management on top of it, and specifically to build cloud management that will work not just in something like Amazon, but actually cloud management that works on any Kubernetes cluster that runs anywhere. So that's for us the key technical challenge to do that well.

[00:27:13] JM: Are most applications wanting to ETL their data into ClickHouse, or is it more of a streaming process?

[00:27:22] RH: So ETL versus streaming. It's interesting, because I'm going to assume by ETL you mean to load data in batches, versus having it flow off Kafka as quickly as it can be generated and consumed. It's hard to make a generalization. I can just tell you that about half of the people that we meet up with that are using ClickHouse are reading data out of Kafka. It's very common. So, right there, streaming is enormously popular. There's a lot of variation in what the rest of the people are doing.

By definition, if you're looking for something where you're trying to achieve very low latency from the time that an event happens in the real world to the time that you can permit a user to analyze it. Anytime you have that problem, you're going to be using streaming of some kind in order to get the data in quickly and make it available. So I would guess that maybe two thirds of the applications have this problem and end up using some form of streaming to load data. And then the rest of them are just loading from things like S3, or they're just taking chunks of data as they arrive in files from various applications.

[00:28:34] JM: What do you think of the idea of this unified data lakehouse architecture? The idea that accessing your data, you should not have to choose between the data lake and the data warehouse. You should have a unified system of access.

[00:28:51] RH: It kind of reminds me of something that was popular in the distributed systems community back in the early 90s where people said, “Hey, it doesn't really – We have distributed systems. They can talk over a network. It doesn't matter where your code runs. This can all be transparent.” And that was kind of a beautiful idea. But it turned out not to be true. The location of things is exceedingly important, because there are things like failures that become more complex as processing points get further and further away. This is sort of at the heart of distributed systems. There are things like latency, the ability to ensure high-performance. And you get exactly that kind of – When you look at lakehouse versus a data warehouse, where the data are stored in an optimized fashion, you get exactly those tradeoffs in these systems. So I think if you're willing to tolerate the fact that you may have complex failures or you may have difficult to solve latency problems. I think lakehouse is fine. But if you want low-latency, if you want the assurance that the data that shows up is actually what you expect, data warehouses really deliver on this. But there's obviously a cost. I mean, you can't put all your data in this format. So I think there's always going to be a balance between these two. And what you're actually seeing in the evolution of the products is there are enough applications that have each of these disjoint set of requirements that there's successful products in both areas.

[00:30:35] JM: As we've seen the evolution of the data warehouse in recent years, how have you seen the evolution of BI tools that are being used on top of that data warehouse?

[00:30:47] RH: Well, they've gotten a heck of a lot better, that's for sure. I think that – In fact, I'll just highlight a couple, Grafana, for example. And these are better – In particular, one of the really great things that has happened is there's now really good open source BI tools. This is a real sea change. So there are still tools like Tableau, for example, which is excellent. We have a lot of people who use it, and we ourselves do a lot of work to support it. But you also have Grafana. You have Superset. You have various other tools that are emerging. Those are two that are particularly popular. If you look at Grafana, what's great about that, it has the ability to display and manipulate time series data very efficiently, to be able to drill down, change the granularity of time. I love working with Grafana for that reason, and it's very popular for monitoring dashboards.

Another popular tool is Superset. I think one of the things that's really great about Superset is it has an enormous variety of ways to visualize data creatively. Plus, it has a great underlying architecture with a nicely designed semantic layer that allow you to minimize the load on data warehouses that are supplying this information. So these tools are something that didn't exist a few years ago. And they really put sort of very powerful software into the hands of developers who are building these new in-app analytics.

[00:32:05] JM: What do you think of the role of Kafka in data infrastructure these days? Like is it still just kind of this pub/sub buffer? Or are people using Kafka as a place to read analytic data from using ksql?

[00:32:24] RH: That is a hard question for me to answer, because we mostly see people who are using it with ClickHouse. And so as a result, they do most of their processing in ClickHouse itself. I actually don't know of – I'm sure we have customers that are doing this that are actually using ksql upfront. But for the most part, they're just tossing it into ClickHouse and doing the processing there. So for us, at least in our world, it's primarily a way of delivering events very quickly and also very scalably. Because, for me, the real power of Kafka is the fact that – There're, well, a couple of interesting things. One is that it has – Because of the way it partitions data, as your data streams increase in size, it's very easy to scale Kafka up. And in fact, the scaling and the partitioning of the queues that Kafka does exactly matches the kind of partitioning that we do inside ClickHouse as we add shards and replicas so that they can both scale sort of in parallel. That's really important. And then the other thing is just speed. That the applications that we see people want to run analytics get very quick answers, and they need the data from the outside world to arrive in a hurry. So Kafka is very good at that as well.

[00:33:39] JM: What are the outstanding data problems that you see among your users may or may not relate to the data warehouse directly?

[00:33:48] RH: I think that one of the fundamental problems that people are looking at right now is how to support built-in analytics in multi-tenant systems. This is actually a tough problem, because the thing that drives SaaS systems and makes them work, both for the people that produce them and the consumers, is that they can pull resources, and through that offer very efficient and cost-effective operation. This is actually kind of hard to do in data warehouses, and for two reasons. One is that when you have the data for multiple tenants in the same place, you have issues that get created when tenants go away, for example, or when tenants have different time to live on their data. So being able to manage that, ensure that you're meeting GDPR and other requirements, is a really difficult problem for which I don't think anybody has a full solution at this point. And some of it's just inherent in the nature of the data warehouse itself, that because they tend to be focused on immutable data, if you have to change data to manage tenants, then there's kind of unnatural conflict there that you end up rewriting data in a way that is going to really hit on your performance or cause huge write amplification, or other problems like that.

The second problem is being able to scale these systems. So data warehouses have this, or basically designed to just grab all the compute an I/O bandwidth that they can to answer questions quickly. And if that's the way that the data warehouse is designed, and most of them fundamentally do this, there's a conflict between that and concurrency, because concurrency requires you to enable a large number of people to operate simultaneously.

So what gets interesting in these applications is, if you can deliver the individual queries quickly, can you then build semantic layers on top, which enable you to have multiple users, for example, within a single tenant or even across tenants if it's appropriate? Be able to take advantage of the fact that you've already run the query once and either cache it or built kubes, do interesting things that allow you to increase the access to the data without putting more load on the data warehouse itself?

[00:36:03] JM: How do you see data infrastructure changing in the next 5 to 10 years?

[00:36:07] RH: It's a hard question to answer, but I'll just throw out a few things that I see which are really interesting. One is, I think, that we're going to see increasing dependence on visualization. This is something that has been kind of an aha experience for me for the last couple years. When I first took this job, which some two and a half years ago, I came in thinking that we were going to do a lot of work to integrate machine learning with the data warehouse. It actually hasn't turned out that way. The thing that – We have a small number of customers who are interested in ML combined with information from the data warehouse, but 100% of our customers are interested in visualizations. So I think what we're going to see is new and more flexible and more creative ways to visualize data. And moreover, they won't just be in BI tools. But they'll be actually components that embed in the frontend stack and enable people to efficiently build very diverse and interesting displays that are just part of the application itself. So that's something that I think where there's clearly a lot of work to be done, and particularly as regards the application development.

Another area where I think we're going to continue to see a lot of really interesting innovation is more open source projects. One of the interesting shifts in data has been the sort of enormous explosion of open source solutions to manage and query data. So we're going to continue to see more open source projects that allow people to query data in more efficient ways, perhaps deal with particular niche problems that are not well served right now. And in the end, we may see more open source data warehouses. That's something that we haven't seen. We haven't really seen a competitor to ClickHouse arise yet. I think, in part because these things take so long to stabilize, that if there is a competitor right now, it's probably still not mature enough to be really visible.

[00:38:09] JM: Any major priorities for the next couple years that you can share about ClickHouse and Altinity?

[00:38:17] RH: Yes, absolutely. So I think there're really two important ones. One is to ensure that ClickHouse reaches full feature parity with the incumbent proprietary data warehouses, and I would point to Redshift and Snowflake as good examples of that, also Vertica. And what that means, for example, is the ability to fully separate compute and storage, the ability to fully utilize object storage, which is something actually we're deeply involved with, and then a complete SQL implementation. These are all things that are proceeding very quickly and where we're benefiting from the fact that we have hundreds of contributors who can all work on these features.

The second thing for us is to build out the management of ClickHouse on Kubernetes. This is something that when we looked at this a few years ago, I was quite skeptical how well this would work, because to be honest, Kubernetes didn't deal with data very well for a long time. But the ability to use Kubernetes as a shim that will enable people to run data warehouses in any environment that is efficient for them is a true boon. And that's something that we're very focused on ensuring that that grows and is accessible for all users.

[00:39:30] JM: Well, Robert, thank you so much for coming on the show. It's been a real pleasure talking to you.

[00:39:33] RH: Thank you so much. It's been a pleasure here as well. And I look forward to talking to you again.

[END]