EPISODE 1216

[INTRODUCTION]

[00:00:00] JM: A data warehouse is a centralized repository that an enterprise may use to store selected data from production systems. Data is transformed into a structured form that makes it easily accessible for business intelligence or other operational users. SQL compliant databases are frequently used for data warehouses due to the popularity of SQL as a tool in business data analytics. Postgres is a free and open source relational database management system. Postgres-based databases are widespread and are used by a variety of organizations from Reddit, to the International Space Station, and Postgres databases are a common offering from cloud providers such as AWS, Alibaba and Heroku.

Josh Drake and Thomas Richter are experts on Postgres. They join the show today to talk about the staying power of Postgres and why Postgres is a good choice for data warehousing and how cloud technology is changing relational database management systems.

[INTERVIEW]

[00:00:53] JM: Guys, welcome to the show.

[00:00:54] JD: Thanks. Good to be here.

[00:00:57] JM: I'd like to start off with a simple question. What is the difference between a data warehouse and a database?

[00:01:04] JD: Well, I mean, a database is, generally speaking, a piece of software, right? A data warehouse runs on a database. It's the type of data that it's storing that's really the difference. Something like an OLTP is going to be transactional. That's where you have things like sales orders, your inventory shipping in and out, that type of thing. Whereas a data

warehouse will be, say, OLAP, which is more of a non-changing dataset based on all your transactional dataset that you would report, get report off of.

[00:01:42] JM: And how does the use cases between OLTP and OLAP, how do those change what kinds of software we should be using? Like why can't a Snowflake satisfy the same use cases as a MySQL database? So I'm not going to speak to Snowflake or MyuSQL. They're very different technologies. MySQL is relational, which would be more geared toward OLTP. But really you know just to be bold, the market's kind of decided where it's going in terms of its OLTP, and that's going to be Postgres-based. The biggest differences are, and to create a metaphor that is a little bit more adequate for people who may not be in the industry, it's very similar to, say, a sports car versus a 4x4 truck. They both do basically the same thing, but they're better at the particular specialties.

And when you take something like a Greenplum, which is an MPP data warehouse based off of Postgres that's open source, it is engineered or finely-tuned to handle a volume of data and understand – I'm not talking about a transactional volume. I'm talking about a size volume of data that is significantly larger than the average normal OLTP database.

[00:03:12] TR: Yeah, and what I would add maybe is the requirement profile is extremely different, right? On the one side you would have many, many small interactions. Let's say, for example, you have an OLTP database on the back of a web service and you want to basically keep every state that the user of your web service could enter. You would want to keep that in the database. So your frontend that's actually delivering this web service is a stateless and everything happens in your database. That means many, many tiny interactions that are pretty self-similar, pretty predictable, but they need to complete extremely quickly, and really you're talking about sub-second and sometimes kind of millisecond range speeds that you want to see.

And on the other hand you may have a analyst doing a reporting on the data warehouse, for example, wanting to find out all the products you ever sold in the last 15 months to your different geographies. And you maybe break that down and slice and dice it across multiple

dimensions. Now that analyst may expect that query to return in real-time, and real-time for him is I press the button and 10 seconds later my screen updates. Or that analyst may say, “Okay, I’m going to queue up a few of these complex operations and then they will be done every night and every morning they will come as a report, for example.” So those kind of requirements are inherently different. On the one side you have some teeny tiny updates that are quite predictable and where you would actually set up the database to do that well. And on the other hand, and the first is OLTP. And on the other hand, with the OLAP world, you have basically querying a large proportion, if not all of your data and trying to slice and dice it across different dimensions. So you're basically asking some very complex questions, but you're ready to wait for it. And there you're back to the 4x4 versus the sports car. They probably do very different things, but they're inherently the same technology.

[00:05:13] JD: Right. And on that same note, with OLAP, as we mentioned, it is that a lot of these will be queued up. If you are trying to get, say, a trending analysis over the last 20 years, you expect that that's going to take time. The calculations that are being performed per the query and the report that's being delivered and the roll-up windows and all these various different things, those aren't things you typically do with OLTP.

[00:05:44] JM: So in most of the shows I’ve done about data warehousing, data warehousing is thought of as something that you must have a unique technology for your data warehouse. Like you must have a Redshift or a Snowflake or some other tool that is dedicated to being a data warehouse. The subject of this show is the idea of using PostgreSQL for your data warehouse. And Postgres can also be used as an OLTP database. So could you just give a high-level overview for why you advocate using Postgres as a data warehouse?

[00:06:25] JD: So I think there's going to be a couple of sides to this. I’ll present my site as a consultant. The real functional capabilities of Postgres is that it's been around for so long, right? It's a mature database. It's heritage dates all the way back into the 80s with University of California ingres. it is open source. It's truly open source, and it doesn't have a central authority that controls it. It's not like a my sequel where Oracle controls MySQL development, period. This is a true community project where you have dozens and dozens of contributing companies

and hundreds and hundreds and hundreds of contributing individuals to deliver this amazing piece of software to the public for free.

Now what has happened over time is that as it's become more economically viable where the ecosystem starts to look at it as a target solution, which for the last decade or so has been very prevalent, you can see it especially with cloud providers. Postgres is one of the few relational databases that's actually growing in popularity. When you take all of these different providers of services and specialties and they're contributing back to the development of the database, you start getting a customer-centric or a user-centric delivery of the software. So there was a time, for example, that we didn't have roll-up queries. Well, as users and clients started demand, the need for roll-up queries, those were delivered in the Postgres codebase and we were able to do roll-up queries, which are a pivotal part of reporting and data warehousing.

And so I think that when you talk about Postgres, because of that, because of its openness and its extensibility, it has really opened the door to all levels of data development, right? It's not just data warehousing. You also have things like graph, which is very popular within the Postgres world.

[00:08:32] TR: Yeah, and to add to that, I would say there's solutions in the market that are absolutely aimed at being just one thing. And you've mentioned Snowflake, you mentioned Redshift, but there're also been solutions that have always been built with the intention to be very versatile. And maybe a classical example of that is large oracle deployments that are capable of doing pretty much everything, of course, at a certain price point. And another example of that is also Microsoft SQL server. Now those are of course extremely proprietary, extremely close source kind of software. So they're probably addressing a very different market and a very different need also in terms of economics than the Postgres market does.

However, they've shown that they're extremely versatile and that there's inherent value in that versatility. And I'll give two examples in a second, but the beauty is now that as Postgres is maturing and it is getting things like, for example, parallel processing within a single query, becoming part of its repertoire, or the roll-up queries as JD has just mentioned, it's getting

increasingly attractive. And the point there is that you can do, for example, things like using relational constraints to make sure my data is clean. One example is let's say you want to make sure that sales reports are never ingested multiple times because that would falsify the kind of conclusions that you're drawing. So you set some unique constraints and thereby you made sure that even if the intern tries really, really hard to ingest your sales data from Luxembourg 15 times, it will only go in once.

[00:10:15] JD: Sounds like you experience with that, Thomas.

[00:10:19] TR: I've been trying that when I was an intern. Yeah. Well, and the other angle you can, for example, do is you can, for example, keep partial aggregates and things like that and update them on the fly. So you can add transactional type workloads even into data warehousing. You can have things like triggers business logic both for actually modifying data if you basically discover certain patterns or, for example, send alerts if certain things don't pass certain probability checks. Yeah. And it's basically that versatility that requires a Postgres-like concept, as in my data is very transactional, strong consistencies. I can make granular modifications. I have a predictable outcome, and that is just tremendous value for keeping things debuggable. I think that's one very important point, predictable and clean.

So I would say those are really the three things where you can, if you take a slightly more – How can I say? Slightly more transaction-inspired approach to data warehousing, you can save yourselves a lot of headaches.

[00:11:34] JD: So I would agree with Thomas, and I want to back up for a moment because he brought up an interesting point. He brought up Oracle versus, say, Microsoft SQL. And technologically both of those are very good platforms. And I think that Oracle would rather go down in flames than actually ever open source their enterprise product. But interestingly when you bring up Microsoft SQL, Microsoft also supports Postgres. And I think that one of the advantages to Postgres outside of the technology of it, which is exceedingly good, is that a lot of the larger vendors are recognizing where the market is going and standardizing on Postgres in some way shape or form. I mean, for example Microsoft owns Citus. So they have federated

capabilities as well as there's your cloud. And then you have your Amazon and Aurora and their RDS. But then you also have you know like Swarm. And, Thomas, I believe you have, I believe it's called Nitrous, correct? Where you have your own cloud services that you're providing around the unique Swarm technology with Postgres.

[00:12:48] TR: Yes, that's correct. Postgres Nitrous or PG Nitrous as we call it available on AWS.

[00:12:54] JD: Yeah. So I think that there're two sides to this, right? It's not just why Postgres is great, because I don't think there's any arguing that. In fact even the most diehard Oracle people will say it's a good database. It's also answering the market demand or the user demand of having essentially the Linux of databases. And I think that Postgres is what's adhering to that, because back in the day Linux was good as a hobby, and then it was good as a print server or a mini web server, and now it runs the majority of super computers in the world. So it services all kinds of workloads. Well the same thing is happening with Postgres. You have your document store capability like a la MongoDB. You also have your data warehouse capability because it's open source. And as it continues to evolve, it's only going to scale farther and continue to reduce the demand and the need for specialized software outside of the Postgres ecosystem, right? Swarm is in the Postgres ecosystem, but something like a Snowflake. And I'm not taking away from their skills or their product, but I think that people will find that that continues to be a niche product because the 90% are served from open source Postgres.

[00:14:17] JM: So I guess I'd like to go a little bit deeper into what a data warehouse setup with Postgres looks like. I mean, if I'm using Postgres, do I have to have some kinds of ETL jobs that dump data into Postgres? Or am I literally just using the same transactional database? Like is my entire database in Postgres? Just give me some best practices for what my setup is going to be like if I'm using Postgres as as my data warehouse.

[00:14:56] JD: So at its most basic level, you'd just be using Postgres, right? Whether or not you're replicating the Postgres from reporting off of it or you're running a workload that's off

hours that won't interfere with your OLTP transactions. That at its core works just fine, and it always depends on the level of data you're talking about, right? Most people don't think about the fact that a data warehouse may only be a hundred gig worth of data. They always think about it on the green column level where you're dealing with, say, a petabyte of data. So it really varies.

And as the size of the data grows, you may do things like ETL from your OLTP to your data warehouse or replicate to a replica and then ETL from that replica to a Postgres data warehouse where you can run all your queries. That's a common way to do it so you don't interfere with your main transactional processing. And then once it gets to wherever you're doing your data warehousing, that's where things get interesting because you do get an opportunity to take advantage of extensions such as Swarm64's extension.

Thomas, you want to expand on that a bit?

[00:16:13] TR: Yep. So one of the things that we do is basically we teach Postgres a wide range of new tricks that it needs to handle the data and also that it actually stops interfering as much as it would otherwise do with your day-to-day workloads, your OLTP workloads. Whereas you can go as far as ETLing between, let's say, a replica and your data warehouse. There's also cases in which it makes even more sense to just have a replica of Postgres that's directly used as the data warehouse. So you don't take an extra ETL step providing that this is what you need and this is basically the data source that you need most. Then that's actually a very, very elegant system design.

And the way it works in our case is basically we're adding new index types that makes it easier to access data especially if you access the majority of your data as is more common in OLAP cases than OLTP cases. Also, we are changing the way how queries are planned to make them much more parallel and optimize the execution and also prevent mis-estimations, which can become very, very expensive in complex queries. And the third thing that we do is basically just make the execution more efficient. So we are basically implementing new join types that allow

you to combine billions of rows with tens of billions of rows and it still works and works very fast. We do things like optimizing the way data travels, etc.

So by taking all these steps, and just to be very clear, these are all extensions into open source Postgres. So we're not a fork. We're not doing what Greenplum did many years ago where they took an old version of Postgres and turned that into a new product. We are basically providing these pieces as extensions and as a result can really make your life a lot easier. So as in your example that you gave, Jeffrey, instead of having an ETL job in between, there are many scenarios where it just works out that you have maybe your replica and you do your data warehousing directly on the replica. And depending on workload, sometimes it even makes sense to just do everything in the single database. And yeah, you have basically one system, very little complexity. And that's possible because Swarm 64 moves things out of the way. We have column store, transaction safe column store indexes that can provide the data so it doesn't interfere with your other processing. We have a faster processing with fewer resources. Again, reducing the interference.

Yeah. So there're many, many options that you have there, but I would say the truth strength is aside from all the transaction guarantees and consistency guarantees we talked about 10 minutes ago, the true strength is also it allows you to do this hybrid workload OLTP and OLAP at the same time.

[00:19:03] JD: Right. And I mentioned the extensibility before. It doesn't have to just be a binary replica, right? And to identify that, a binary replica is an exact copy of your OLTP. We can also use the built-in logical replication so that we only have to replicate out specific datasets that you would actually be reported against. And that also allows you things like data siloing where certain people who are only allowed access to certain information can report against that information without ever touching the main transactional store. So it adds a lot of feature capabilities not only in the idea of data security, but just in general, extensibility and working within the tool set that you have available to you. And something like a Swarm64 just continues to increase that value because it's not a fork and it allows you to just essentially

install an extension and all of a sudden you have all these great new features available to you for data warehousing.

[00:20:10] JM: Can you tell me more about how the extensions ecosystem enables this multifaceted Postgres functionality?

[00:20:18] JD: Yeah. So Postgres for the longest time used to have a what was called contrib, in other words contributed modules. And a lot of business people would be hesitant to use them because they weren't “officially part of Postgres” even though they shipped with them. It was a separate package and it made people nervous.

So a few years back, and I don't remember exactly when, but it's been some time, community decided, “Okay, let's give these a little bit more credence.” And there is still contrib, but instead let's create a universal API that allows us to build extensions. And this is actually gone so far as to allow people to write their own storage engines and index types for Postgres itself. But these extensions range in all kinds of capability. Probably the most popular extension is PostGIS, PostGIS, which allows for geographical information system processing within PostgreSQL. In fact it's one of the things in Postgres that we do better than anybody else is GIS, and it's just an extension. So you have an entirely different community or a company, depending on what the extension is, building in a business model, technology, users clients off of a particular need building demand against a uniform API, just Postgres, but they're adding their own particular features. And you can just download and install those features.

Now that may be as simple as a new server-side programming language. It could be something as extensive as for example, Swarm64 with new index types and parallelism and compression and so on and so forth. It can be time series data. There're a couple of companies out there developing graph. There's even a couple of projects that allow – If you install the extension, it will use GPUs, your graphical processing units, video cards basically, to do specific scientific calculations because GPUs are better at those calculations than your normal CPU or Xeon chip or something like that. It's really a great way to have – So if you download just Postgres, you have a full featured database. But like Linux, if you download just Linux, you have a full

featured operating system, but you still need something to do with it. So if I need GIS, I have PostGIS. If I need data warehousing, I have Swarm64, those types of things.

[00:23:01] TR: And I can vouch for the fact that a community has done a great job there in terms of embracing this extensibility. And what's also really interesting is there're some very special purpose index types, but they can be absolutely genius for very specific needs. Things like, for example, inverted indexes and so on. So there're many, many interesting things that are developed because it is made so extensible.

And of course, and I think JS was saying that or implying this maybe slightly, there's also a certain variation in quality. There're some things that are more ideas and there are a lot of things that are quite enterprise level quality. So that's the other thing. But it's great, because it means the barrier to enter is low enough for people to engage. And that's why we have all these great features.

[00:23:53] JD: And that's absolutely true. There're definitely variations in quality. There are a lot of people that take a weekend to scratch an itch and put it out in the world and it's an extension for Postgres. And we need those, right? That's your proof of concepts. That's how you take something and run with it and build a better ecosystem. But then you have what I think Thomas is kind of alluding to. You have your enterprise class extensions where you have your enterprise class support, your consultants, your people that are really going to enable you as a client or a user to succeed with the software.

[00:24:28] JM: Why don't we hear about more people using Postgres as a data warehouse?

[00:24:34] JD: Interesting. Thomas, you want to start with this one?

[00:24:37] TR: Yeah. Let me give it a shot. So number one is whereas Postgres has always been excellent in enforcing kind of parallel access methods. And what I mean with that is you have many, many concurrent users modifying data or reading data and you want to give each of these users and each of their connections certain consistency guarantees. And I would

argue that one of the strongest elements in Postgres is what they call the multi-version concurrency system that allows you to have a strong set of rules that every user will basically or every user's query will basically adhere to so you don't have certain ghost data appearing in the middle of your query. And if you ask the same query under certain conditions and certain guarantees, basically you will get the same answer, or you will, in other cases, not get the same answer and you can know exactly why you did not get the same answer. So it's all very predictable clear set of rules.

Now, this is incredibly difficult to do in parallel with many people basically accessing the data at the same time, and Postgres has always been awesome at making that possible, that many, many individual connections can access data in parallel. What Postgres has until recently not been very good at is allowing a single user to use the entire machine to answer his complex query. So your classical kind of analyst workload where somebody is trying to – As we said earlier, right? Trying to do some trend analysis over 20 years. That would run very, very slow. And only recently has that improved. I think we're going back to version 9.5 I believe when parallelism was introduced. Possibly 9.6 I think –

[00:26:22] JD: It was 9.6, with, I think [inaudible 00:26:24] was the first one.

[00:26:26] TR: Yeah. And since parallelism got a lot better and we've seen all these changes. So we've been there kind of when 9.6 shipped and basically tried the first parallel implementations and it got significantly better in recent versions. So more and more things turn parallel. There're fewer and fewer parallelism inhibitors. And I think that's really a reason why Postgres in the beginning wasn't really such a viable option for the kind of problems that require this massive multi-processor approach, right?

[00:26:59] JD: I would actually agree with that, and I think that it's important to point out the development model here, in that PostgreSQL tends to develop incrementally. And so, for example, the parallelism in 9.6 is not anywhere near the same parallelism in 13. And Thomas is also correct in that. What's happened is the parallelism has allowed us to achieve in this type of workload a level of performance that we couldn't achieve before because we're able to grab –

Well, instead of having a single lane highway, we have eight lanes of I5 running down the state in terms of being able to retrieve and move data. And under transactional situations that's rarely a concern, because as mentioned, we're dealing with small amounts of data, processing relatively small amounts of data. But when you're dealing with hundreds of gigabytes, to terabytes, to petabytes, you need to be able to utilize the entire CPU band with memory bandwidth and storage bandwidth as well all together and the parallelisms allows you to do that.

[00:28:11] TR: Yeah, and then we allow you to dial it up to 11 with Swarm64 and then you don't have an eight-lane highway. You have a 64-lane highway, for example. Yeah, that's what we do. But there is a – I think Postgres has turned a much, much more viable starting point now even if you use it without the Swarm extension to actually run these kind of queries than it was maybe five years ago.

[00:28:35] JD: I would agree with that.

[00:28:36] TR: Yeah. If you're interested more on parallelism – For example, we have a series of blog posts on Postgres parallelism, which also talk about Postgres parallelism not only with Swarm64. So very invited to check out our blog on swarm64.com on that topic. It's a very interesting one.

[00:28:52] JD: Well, speaking of that, Thomas, you also added, if I recall correctly something that a lot of people don't think about in that with your index, your new custom index, don't you compress the data making it a lot more efficient to retrieve?

[00:29:10] TR: Yeah. That's true. So one of the key – Let's go a little bit away from the traditional transactional setup where you would store data in rows in order to make it very easy to modify, because usually when you modify a row, you tend to modify more than one value. And if it's just the timestamp that something has changed, for example. But usually modifications happen with multiple modifications in a row. And by contrast, of course, all these,

or I would say the vast majority of OLAP or analytics optimized products are all based on a columnar storage or some kind of hybrid form.

And what we found basically is that the ideal way to add this into Postgres but still stay extremely – Like we want to be 100% compatible and we want to stay extremely close to the Postgres core data itself. So we found the best way to do this is basically using a column store index. And what we do is we take the data and we put it in a columnar format and then we compress the hell out of it, because the important part is that the more you compress the data, the faster you can retrieve it as JD was saying. And that's really a concern if you run on this, as we said, 64-lane highway. It takes a lot of gasoline to fuel all these 64 cars going down the 64-lane highway. It is quite difficult to do this with regular Postgres because it takes quite a bit of time to retrieve all that data. So with Swamp64 and the index, it actually gets a lot easier to feed all these individual workers with data so they can run at maximum speed and complete the query for you as fast as they should and as you want it.

[00:30:58] JD: Well, and something I'd like to bring up on that, Command Prompt has been doing some testing as a partner against one of our client workloads. And we're not dealing – We don't need the 64-lane highway for this particular case. In reality we only need maybe half that to achieve the result that we want. And what we've found is Postgres was great. It actually performed the queries at an acceptable level, right? Nobody was pulling their hair out. But when we installed Swarm64 and ran the same queries against it, and all of a sudden people weren't not or weren't just not pulling their hair out. They weren't able to pick up their coffee and take a sip before it returned. We have seen exponential improvements in performance even on what I would consider a standard reporting database. Not just a very large data warehouse. The technology has really been able to achieve something that it's more of a, "Huh! Well, that's awesome." Instead of, "Okay, I've got a couple of minutes here. I'm going to go get a cup of coffee." And it I think it's going to be a game changer as people start to realize what is actually possible without having to add a bunch of very difficult logic to say their application or ETL processes to their infrastructure and things like that. This extension kind of – It puts you in a place where it solves a lot of your problems. It takes a lot of complexity out of delivering the results that you need from your data and it does so in a very performant way.

[00:32:40] JM: So are there any limitations of Postgres that remain that are still bottlenecks to making it a strong choice for a data warehouse format?

[00:32:54] TR: Well, I would say one thing that doesn't quite allow Postgres to go multi-petabyte scale yet is that the – So in the community, there's a lot of movement around native sharding, and this is something that we're also very actively watching and supporting as we can. So that the notion that you want to scale across more than one compute node, but not just in the replicated fashion as we've discussed in the beginning. That's completely possible, no problem, whatsoever. Many, many different solutions to that. But actually more in a shared, very little or share nothing kind of logic. As in you really have all your shards only containing a fraction of the data and then they work together to solve that problem.

And Swarm64 has decided to not take the route that other solutions have taken and diverge from what the community is doing, but rather support what the community is doing there. And I would say as you're moving into multi-petabyte where all the other mechanisms for scale out that we already support are not working, then I would say we still have to wait for Postgres 14 and probably Postgres version 15 until this is starting to be ready for prime time. So the ability to really distribute data across a whole cluster but without any kind of strong sharing, but really more like share nothing or share very little type logics. That would be my take.

[00:34:22] JD: So let me elaborate on that a little bit, and I just want to be specific, because we're talking about extremely large data sets, right? Like, you said, multi-petabytes. Most people can't even envision what that means. It's a huge data set. Your average data warehouse where you're talking about, say, 20 to 100, to maybe even 250 terabytes , Postgres can do that now. And I think he's right. In order to get to the true fortune 50 need of data warehousing, we are looking at a couple more years of development. But in terms of serving the 90%, the real interesting things for me would be – Well, the parallelism and having a more – I don't know that we'll ever have true federated Postgres. I think that adds a level of complexity that would be difficult. But I do think that we will continue to see improvements in the federation abilities that we have. And I know the community has talked a lot about transaction managers and things

like that. But I think that it's important to recognize that even now we can meet most people's needs.

In terms of taking on, say, a Greenplum, it's challenging for a community project to take on a company with – And I don't know what their exact sales are, but let's call it 800 million a year in sales dedicated to data warehousing. But then, again, the Greenplum solution, which is Postgres-based, only services a very small subset of the market. The rest of the market is able to use especially now with something like Swarm64 and/or parallelism capabilities can use Postgres as it sits today.

[00:36:24] JM: Do you have any perspective on the difference between the UX or the performance uses of Postgres as a data warehouse versus Snowflake or Redshift?

[00:36:39] TR: I would say that the difference is primarily there's a dimension of skill I would argue and a dimension of freedom. So if you take Postgres on its own, you have to have more skill in order to bring it to the level of what, for example, a Snowflake would give you out of the box simply because it's pre-configured to do that. You would of course have infinitely more freedom on all dimensions, be it commercial dimensions with the licensing. I mean, there's nothing much more friendly than the Postgres license. Yeah.

Also, on technical dimensions, given all the extensions and all the things you can do with it and also features, I would say that possibly Postgres is the most fully featured. Definitely the most fully featured open source database in the world. And of course with all these extensions by far does more than Snowflake. However, as I said, you have to invest the skill in order to get to that level of freedom and capability.

Now, what we are trying to do actively is to basically limit the complexity of that learning curve. So, for example, our PG Nitrous on Amazon, and we're looking at further clouds where we're going to make that available is now step-by-step developing to make it easier and easy and easier for you to deploy things and have them readily set up.

So, for example, in the release that is going to go live within the coming days, we will have cloud formation templates in Amazon that just allows you with a few clicks to have everything ready and readily set up. And we're going to expand that into replication schemes and certain forms of scale out and so on you can do. And that will make it much easier. So a pure play Postgres takes more skill, gives you more freedom and capabilities. On the other side, you've got Snowflake, which makes things very easy for you. And what we're trying to do is basically be in the middle and bridge that gap step-by-step. We're not there yet, but we're kind of every day working on narrowing that gap and making it as easy as, for example, using a Snowflake. However, on the capability side and on the commercial benefits, you retain the benefits of Postgres, and that's very important to us.

[00:38:56] JD: Yeah, I think that's kind of key. I mean, something like a Redshift, which let's remember, Redshift is actually based on Postgres, or a Snowflake. They're closed source and they have their data, put your data into it and they decide at whatever level based on your EULA or service level agreement what you're able to do with that data and how you're able to get it out. With something like a PG Nitrous with a Swarm64, the core of it is still just Postgres, and that allows you a certain level of freedom that a lot of people don't realize is important until it's too late, because ultimately you want to own your data, you want to own how you interact with your data, you want to own how you can get your data out, because data is the new dollar. And so you want to be able to protect that and grow that in a way that is safe or respects extensibility and allows you to move freely between various platforms, whether it'd be – It's not even as long as it's Postgres-based, right? One of the great things about Postgres is that because it believes so much in freedom that it can interact with almost any other platform. Some of the platforms I won't even mention because it's just terrifying that it can do it, but it's because of that extensibility and freedom to allow people to control their data that that exists.

[00:40:24] JM: Well, as we begin to wind down, do you guys have any perspective on the future of the data warehouse ecosystem or the future of Postgres?

[00:40:35] JD: Well, I mean that's how I make my living, right? Command Prompt is the oldest Postgres provider in North America. And I can tell you that the only thing that we have seen

through the years is consistent and persistent growth. And I attribute that to the community building such a fantastic database. I think that what we're going to see in the next couple of years is a stabilization point where Postgres actually isn't getting all that more nifty for lack of – That's a really silly term, but it won't. The changes will be very mature changes, optimization of the planner, different ways of dealing with statistics, possibly a new and innovative index type. But what you're going to see is a continued stabilization so that you're essentially writing to a data API and that data API is provided by Postgres and the storage layer is provided by Postgres and extensions and it's really going to free the world, the developing world – That sounds bad. The developer world to basically know that they have a stable API no matter what they're doing. They can just write to it. And you'll see this with other databases too. CockroachDB is a perfect example. They took the Postgres API so that they wouldn't have to invent all their own new drivers.

And so what happens now is that there are changes that have to be made, but the same API exists between those two platforms. And when it comes to data warehousing specifically, I think that Thomas is right. You're going to see stronger parallelism, which is going to allow you to work with even larger data sets that will actually break us into those, say, one to five percent data set manipulations that right now we're not good at. And in reality, what'll be great is that five years from now Postgres will be the same Postgres that it is in terms of usability as it is ten years from now. Just like Linux as a whole, there are changes. But as a whole the same Linux that I ran in 1995, the same commands, the same features, the basics of it haven't changed. It's all been enhancement. And I think that's what we're going to see with Postgres is in the next few years we're going to see small minor enhancement to make the experience that much better, but we're not going to see lightning rods necessarily, because Postgres already has all the lightning rods under control.

[00:43:15] TR: Yeah, I would agree with that. Although I would say that certain planner improvements, certain statistics improvement are easy to underestimate in terms of the impact they can have. So from my own experience, I can tell that it can make such a massive difference how a query is planned. So Postgres tries to find an optimal plan by comparing many, many different ones it tries. And then on a cost-based approach, basically chooses the

best one or the one that thinks it's best. Now in reality you find that this isn't always the case. And I think, really innovations, in that domain are having a very, very significant performance impact. So that's why I would say I would agree with the directions Postgres is going. However, I would not discount just how much performance impact that will have in the coming years.

And the second thing I wanted to mention also with this API thought, I would wholeheartedly agree, and I would say that the access method is probably going to strengthen. So there's this foreign data concept in SQL and Postgres has brought that in and kind of wholeheartedly embraced it. And you can, for example, read parquet files from Postgres and you can access into Kafka streams from Postgres and you can do all sorts of things, but you're within Postgres. And I think that's incredibly versatile. So I would definitely expect that to expand in future really around the thought of you have a front-end API that's the same for Postgres or for EnterpriseDB or for Cockroach, and then you can use that actually but also to access other data sources. And that saves you not only a lot of complexity. It also saves you a lot of double storage of data in different places. So I would say there's some exciting things to come in that direction.

[00:45:04] JD: I would agree. I by no means wanting to discount the work that would be done. I was more trying to add strength to the API argument that we're going to end up in this stabilized API. But you're absolutely right. I mean, I've seen changes to the plan or go from all of a sudden at one point a query takes 10 minutes and then in the next release of Postgres it takes considerably less than that just because planner analysis. And I also agree that it's actually one of those funny things, because at one point in my career I had someone say, "Hey, are you a C hacker and could you contribute to Postgres?" Well, I actually do have code in the backend of Postgres, but here's the difference. Sure, I can hack on PSQL and make it do new nifty things if I want to, but I wouldn't even touch the planner file. It's just a different level of expertise and thought process. And so my hat's off to the people that can actually manipulate and make the planner better, because it is a truly an exercise in just mental Olympics and it's fascinating to me.

[00:46:09] JM: Cool. Well, guys, thank you so much for coming on the show and giving me some insights as to Postgres as a data warehouse. It's a completely novel concept to me.

[00:46:19] JD: You're very welcome. Thanks for having me.

[00:46:22] TR: Yeah, thank you very much. It's great fun.

[END]