EPISODE 1021

[INTRODUCTION]

[00:00:00] JM: A data warehouse serves the purpose of providing low latency queries for high-volumes of data. A data warehouse is often part of a data pipeline which moves data through different areas of infrastructure in order to build applications such as machine learning models, dashboards and reports. Modern data pipelines are often associated with the term ELT, which is an acronym for extract, load, transform.

In the ELT workflow, data is taken out of a source such as a data lake loaded into a data warehouse and then transformed within the data warehouse to create materialized views on that data. Data warehouse queries are usually written in SQL, and for the last 50 years, SQL has been the primary language for executing these kinds of queries.

DBT is a system for data modeling that allows a user to write queries involving a mix of SQL and a templating language called Jinja. Jinja allows the analyst to blend imperative code along with the declarative SQL.

Tristan Handy is the CEO of Fishtown Analytics, the company that created DBT, and Tristan joins the show to discuss how DBT works and the role that it plays in modern data infrastructure.

[SPONSOR MESSAGE]

[00:01:25] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW]

[00:03:14] JM: Tristan Handy, welcome to Software Engineering Daily.

[00:03:16] TH: Thanks so much.

[00:03:18] JM: What is the role of the data warehouse in modern software engineering?

[00:03:21] TH: Oh gosh! I think that the data warehouse used to be kind of the more boring place for computation workloads to happen in data engineering. All the exciting stuff was happening in MapReduce, in Spark. As the data warehouse has become more performant, is beginning to I think take over more of those workloads, and I think that you are seeing a lot of that transition from the prior computing architectures. Not to say that Spark is prior, but I think that a lot of people are finding that SQL plus the data warehouse can actually accomplish significantly more than they realized and they can use it with greater efficiency. Like it's easier to operate and larger numbers of their internal stakeholders are able to be first-class citizens in that infrastructure.

[00:04:18] JM: It seems like we went from a time where people were figuring out how to work with Storm, or Spark streaming, or Flink, or Apache Beam, or name your distributed stream processing framework as the source of analytics. To a time where we stand today where I don't think people would've anticipated this, but basically the role those stream processing frameworks play is to prep the data for the data warehouse and all the real nitty-gritty large-scale data analytics takes place in the data warehouse or in Apache Spark which some people might considered data warehouse. Do you think that's true?

[00:04:58] TH: I want to give a caveat that while being certainly adjacent to the stream processing layer, I am not myself a software engineer, and so I have literally never written a streaming job in my life. The thing that I'll say is that I think that there's a lot of evolution when it comes to the necessary SLA's for different use cases for data. I think that for a long time, it was batch based in 24 hours and like that was just kind of what you expected.

Then when people wanted to do more near real-time use cases, we kind of overcorrected a little bit. We were like, "Well, it must be in under five seconds," and it turns out that there's a tremendous amount of engineering effort even with great frameworks to deliver that type of latency. If you relax the constraint a little bit to say, "Well, I want my data to be within five minutes." You can start using data warehouses to accomplish those use cases. It's really about finding that sweet spot of like what level of latency do I really need and what's the tradeoff I'm willing to make for that.

[00:06:12] JM: We can talk more about different tools later on, but focusing back on the data warehouse, there are steps in a data workflow before the data gets into the data warehouse, and then there's the work that actually takes place within the data warehouse. Can you give an overview of those two separate parts of a data workflow? Kind of maybe preparatory or ingestion step of data engineering versus the actual data warehousing process of data engineering.

[00:06:45] TH: Yeah, and I think that this is something that's changing very much in real-time. There is this kind of scaling problem associated with data ingestion. The number of different data sources that your company might want to get data from is growing faster than the data engineering resources of a company will ever grow. As such, there have to be ways of reducing that scaling problem to something that's a little more linear, which is where a lot of the ingestion tooling is coming into play. Companies like Stich and Fivetran are off-the-shelf solutions that may not cover 100% of the surface area, but they help you cover the big majority of the use cases and just literally get the data in in whatever shape it presents itself in the source.

That is a big change from how ingestion used happen. It used to happen into an intermediate environment, which then the data preparation would be run and then the data would be loaded into the warehouse after the fact. Right now, the focus for ingestion is shifting to load the data in in whatever shape it presents itself in the source.

Then the transformation workloads happen inside the warehouse, and that's why I'm happy to talk as much or as little about DBT, which is the project that I spent my time working on. But that's why there is an opportunity there because we're seeing that shift from ETL to ELT.

[00:08:18] JM: This shift from ETL to ELT, you put it eloquently. My understanding is it's driven by – At least in part by the desire to give maximal control or maximum autonomy to the end user of that data warehouse. Rather than saying we're going to do ETL. We're going to extract the data from the source or ingest the data from the source. We're going to transform it into the normalized table schema that prepares it for the data warehouse and then we're going to give it to the dashboard or the BI user to directly query. Now we're saying, "Look, we're going to extract it. We're going to throw it in the data warehouse and you can build whatever materialized views on top of that somewhat perhaps messy data as you want or in the kind of ETLT kind of idea." It's like, "Okay, we're going to do a little bit of transformation. We're going to clean it up and then we're going to prepare it for the data warehouse," but you're still going to be able to defi ne schemas and materialized views within the data warehouse.

[00:09:22] TH: I agree that the main goal is to get this into more people's hands. The real unlock is the capability and the reduction in cost associated with doing compute inside the data warehouse. You literally couldn't throw billions of events into a warehouse in 2010 and just like say, "Hey, users. Have fun." But you can do that now. You can make an argument that like there are lower cost options there. It probably is cheaper to do what Pinterest does and run a massive Presto cluster as supposed to trying to process that scale of data on one of the commercial data warehouses today.

But I think that for most companies, the scale of data that they are operating at is not Pinterest level scale. So the assumption of just like throw it all into the warehouse and let everybody have at it actually works out very nicely for a large majority of companies even their event data, which as long as you're talking about – I would say, I'd use the word reasonable scale data, and that is like anything where you're talking about terabytes and not petabytes, then you don't necessarily have to think too hard about those kinds of choices, I think, today.

[00:10:45] JM: The data analyst who is using the data warehouse, what are the biggest frustrations of that data analyst?

[00:10:53] TH: I started my career out as a data analyst, and if somebody asks me what I do today, that's I think how I still self-identify 20 years later. I have a chip on my shoulder from 20 years of essentially being underestimated. I think that tools that have been built for data analysts don't enable data analyst to have the same kind of impact in their organizations as software engineers have in their organizations. They cost more so data analysts don't get to choose their own tooling. They historically are not open source. So analysts can like submit a PR to like implement a new feature. They don't enable scale and that like there's really like a tremendous amount of copy-paste associated with historical data analyst workfl ow.

So if you want to think about pain points of a data analyst, like the pain point is that like the career path sucks, and it is because you can build the core set of skills you need to be a proficient data analyst in the old model. You can build them over the course of six months to three years. Then because the workflow sucks, there's not a great way to like grow in that role over time, whereas a software engineer can become a better software engineer for 20 to 40 years and continue expecting to get commensurate pay raises very much so on an IC track. The data analyst needs to figure out what the next step in their career is. Are you going to go into management? Are you going to become a data scientist or a data engineer? There's this assumption that like you have to do something else, because like the workflow won't let you scale yourself. What's the main pain point? The main pain point is like I want to be able to continue getting better at what I do and have more and more impact as I grow.

[00:12:52] JM: I love that answer. It's very abstract. Do you have any more like specifi c things? I love the existential nature of that answer, but what about like writing my queries? What sucks about that?

[00:13:06] TH: Yeah. If you want to get really tactical, it depends on how like long ago you want to compare to like what people are doing now. I think that you could talk about reproducibility, because 10 years ago the job of data analysts was to like re-create the same report as last month and put new data into it. We kind of like solved that problem with the modern the modern BI stack circa 2014, 2015, Looker, Mode, etc. do a nice job of saying like, “You hit the run button and the report gets produced, and so the data analyst doesn't need to like update the reports from last month.” Great. We kind of solved that problem.

The thing that like the industry is in the process of solving now is more the like provenance problem. Where does the data come from? Where is it going? How do I prove that it is correct? Then that's when you start having to really think about this problem as a software engineering problem, because software engineers have had to prove that their systems work for as long as there have been systems. You need to start thinking about code quality as supposed to just like can I get this thing to work once. It's how do I ensure that it will continue to work indefinitely in modularity?

Really, that is what we spend all of our time thinking about. We’re trying to help data analysts apply software engineering best practices to their work. Our community loves the stuff that we’re doing, but it's funny that we as product people don't really have novel ideas. We are literally in the business of like stealing from software engineering practices and thinking how do I adapt this to the world of the data analyst?

When we think about like, “Okay, what's our product roadmap for the next year? Next two years?” The questions like, “Well, what practices from software engineering are not currently being employed by data analysts and how do we port them over?” There are certainly a list of those that are starting but not in a way like maturely being handled by data analysts.

[00:15:16] JM: Let's dig in.

[00:15:17] TH: Testing. Automated testing is a thing that software engineers – I don't know. You probably know the answer to this better than I do, but I think it's been a big focus for at least a decade and maybe two. Is that right?

[00:15:30] JM: Testing?

[00:15:31] TH: Automated testing? Yeah. When was –

[00:15:33] JM: Never been a focus for me.

[00:15:36] TH: When was test-driven development a thing?

[00:15:39] JM: I mean, I have my existential anxieties about test-driven development. How long it's been a thing? I mean, there was a class I took in college that was literally called software engineering and I remember day one we get the extreme programming book Ken Beck wrote. I think it's all about testing. Write your tests before you write your code. Then like every single job I got out of school involved writing reams and reams of unit tests. I would assume it came at least 5 to 10 years before I entered industry. So somewhere around 2000.

[00:16:12] TH: Yeah, I think it was in the early 2000's or somewhere about then. But my point is that software engineers have been doing this for a little while, and you really – Once the system gets complex enough, like human testers are not a good scalable way to like assert its ongoing quality.

[00:16:28] JM: For sure.

[00:16:30] TH: So we are in the process of spreading that gospel in the data analyst world. DBT provides capabilities of doing testing. I wouldn't say that they're like where we want them to be eventually, but is a good first step? It's really just like such a simple proposition, just like run SQL queries against your data to like make sure that it conforms to your expectations. Then you realize that like, in many cases, it was wrong. Your expectations were wrong and you just previously had not thought to like instruct the system on how to validate them. That's one thing.

CICD is another big thing that we're like in process of now. We've done some work on that front and there's, again, a lot more work to be done. I think that sometimes the way that you have to adapt problems or workflows from software engineering to the data analyst world is that you have to be a little bit more opinionated. I think that data analysts don't necessarily want to sit there and like build deployment pipelines from scratch. They want a common set of best practices, ways that like CICD should work, and maybe that will change over time. But there are increasingly mature tools in the software engineering space and folks like GitHub and GitLab are like really pushing this pretty hard. We are nowhere near that, but literally just the ability to like take code and automatically deploy it such that your analytics code is always being built off of master. That's a surprisingly new idea and we're just now like starting to see people doing that stuff.

[00:18:14] JM: I can reflect on times doing unit testing and not enjoying it. I recognized its importance and I recognized the impact a lack of consistent testing has on the data engineering environment where it feels kind of loosey-goosey where it's like, "Yeah, we got a machine learning model. It works 90% of the time, and we're okay with that." We want these things working more reliably.

[00:18:40] TH: I think it's just – I mean, outside of the fact that it produces negative business outcomes, it's also a big drag on the time of a data team, because you spend a tremendous amount of time fielding inbound tickets from your users who are saying like, "This freaking thing is broken. What is going on?" and the debug process on all of those types of problems takes a long time. It's better just to not have to deal with that in the first place. Be alerted when there are problems. Don't need to go through the communication process with the user. Just fix it and make sure that you're within your SLA's with you user population.

[SPONSOR MESSAGE]

[00:19:34] JM: I remember the days when I went to an office. Every day, so much of my time was spent in commute. Once I was at the office, I had to spend time going to meeting rooms and walking to lunch and there were so many ways in which office work takes away your ability to be productive. That's why remote work is awesome. Remote work is more productive. It allows you to work anywhere. It allows you to be with your cats. I'm looking at my cats right now.

But there's a reason why people still work fulltime in offi ces. Remote work can be isolating. That's why remote workers join an organization like X-Team.

X-Team is a community for developers. When you join X-Team, you join a community that will support you while allowing you to remain independent, and X-Team will help you find work that you love for some of the top companies in the world. X-Team is trusted by companies like Twitter, Coinbase and Riot Games.

Go to x-team.com/sedaily to find out about X-Team and apply to join the company. If you use that link, X-Team that you came from listening to Software Engineering Daily, and that would mean that you listen to a podcast about software engineering in your spare time, which is a great sign, or maybe you're in office listening to Software Engineering Daily. If that's the case, maybe you should check out x-team.com/sedaily and apply to work remotely for X-Team.

At X-Team, you can work from anywhere and experience a futuristic culture. Actually, I don't even know if I should be saying you work for X-Team. It might be more like you work with X-Team, because you become part of the community rather than working for X-Team, and you work for different companies. You work for Twitter, or Coinbase, or some other top company that has an interesting engineering stack, except that you work remotely.

X-Team is a great option for someone who wants to work anywhere with top companies maintaining your independence, not tying yourself to an extremely long work engagement, which is the norm with these in-person companies, and you can check it out by going to x-team.com/sedaily.

Thanks to X-Team for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:22:09] JM: The data analyst is using SQL for their queries, primarily. Are there any shortcomings with SQL as a language?

[00:22:18] TH: Oh, I love that question. I think that it depends on how you define a shortcoming and what you expect SQL to do for you. I think that in terms of data prep and descriptive statistics, I have not found a problem in the past four years that SQL couldn't solve for me. SQL does not attempt to – Well, Google is trying to do some stuff in this area with BigQuery. But by and large, ANSI SQL is not attempting to do predictive statistics or machine learning or anything like that. If you like understand SQL to be data transformation and descriptive statistics, the current version of ANSI SQL does a really fantastic job. Window functions have grown in their sophistication like very significantly in the last 10 years and it really is the increase in functionality from the window function spec as well as the implementation of that spec by the main data warehouses that allows you to make the kind of investment in SQL as a platform that organizations are making today.

When I was a data analyst prior to 2010, it wasn't just the data warehouses couldn't process the volumes of data that they do today. It was that like SQL had things that it couldn't do. One of my favorite examples of like things that you wouldn't expect SQL to be able to do that can is like TV advertising attribution, which involves calculating a standard deviation and a trailing window over minutes of activity on a website. You then have to do some fairly sophisticated looking at like, “Okay, this was two standard deviations above the norm. Therefore we are defining that is a “spike” likely caused by a TV ad,” and then you can define how long that spike is. This is all like stuff that not so long ago that was not possible. Standard deviation window function including a rolling window is like a fairly new thing to be supported in the major data warehouses. Without that that, you literally just couldn't do that analysis, and now you can. Yeah. You shouldn't be thinking about training a machine learning model in SQL. That's not what it's for, but you can essentially express any type of workload you want as long as you like understand what it's supposed to be doing for you.

The other part I think what you're getting to with that questions like SQL's declarative, and so you end up – There can be a lot of copy-paste. You can’t decompose things in the way that you might want, and those same things are true of HTML. HTML is a declarative language. As a result, you tend to, as a software engineer, not write HTML. You have abstractions on top of HTML that allow you to output HTML at the end of whatever pipeline you're working in.

I often think that SQL is not the right programming abstraction for data, but it's the right abstraction to actually be asking a question to the data warehouse. The question then is like – And in this is what we spend all of our time thinking about with DBT. DBT provides templating on top of SQL. So we are essentially building the Rails layer to use like a very simplistic example on top of HTML so that analysts can express their logic at the appropriate level of abstraction.

[00:25:57] JM: Right, for people coming from frontend world. I remember writing my first Rails application and there is that place where you write – I can't remember what the domain specific language in Ruby on Rails is.

[00:26:10] TH: ERB.

[00:26:10] JM: ERB. Right. ERB. You write – There's a templating language and you can write imperative logic, for loops and if statements and stuff in HTML. This was early on in my web development education. It was totally liberating. I started throwing all kinds of stuff in there. More recently I think of an example, like JSX, which is I don't know if you've seen that. Okay. So like React, which is a frontend JavaScript framework mixes HTML and JavaScript in this weird file format. It was weird. Now it's not so weird. But the point here is that there is an advantage to a mix of declarative and imperative language, and this extends to our data operations. Can you give a few examples of why imperative and declarative code mixed together is useful for a data analyst writing SQL queries?

[00:27:10] TH: Yeah. I think the declarative –

[00:27:14] JM: By the way. Sorry to interrupt you. But for people who don't know the imperative versus declarative things, like declarative would be like SQL or HTML where you're saying like, "This is what's going on Go versus –" I'm not saying Go lang. I'm just saying this is what's going on. It's not easy to explain imperative versus declarative, but imperative is like for loops and if statements and control logic as supposed to just a statement about what is going on. But yeah, probably the most easily on example, HTML and SQL are declarative. Imperative would be like JavaScript, Java.

[00:27:44] TH: With a declarative language, you're describing the result you want. Within imperative language, you're describing what you want the computer to do. The magic thing that any database does is that it has an optimizer, and the optimizer takes your declarative statement about what result you want and it maps that on to the most efficient set of operations to achieve that desired output.

The optimizer is one of the main reasons why it is both more accessible for a data analyst to program in SQL versus a like very experienced data engineered to express some logic in scalding. You don't actually have to think about how to go about processing the data. You are only saying like, "I would like my eventual data to look like this." That ends up like making it take less time to write the query as well as it opens the query writing process up to a larger set of users.

Oh! Here's a use case of like why you might want to go back to your original question. Why might you want to combine them together? Imagine that you've got – It's very common to load event data into a data warehouse and have one table per event type, and yet those tables often have much the same data loaded into them and sometimes you want to union data together from many of those tables. Let's say that you've 30 different events. You want to build one event table that has the data from – Or some subset of the columns from all of those tables union together.

If you were going to write that union statement by hand, in SQL that would both be very time-consuming and fragile. You have to write hundreds or potentially thousands of lines of code to like get all of that data normalized together. Using DBT, you can express that logic as like a call to a union function and passing in a list of the tables that you want to union together. DBT then goes and introspects the schema of all of those tables and constructs an appropriate union statement and you can accomplish the whole thing in 35 lines of code.

[00:30:02] JM: Contrast the experience of a data analyst working with raw SQL versus the experience of a data analyst working with DBT.

[00:30:13TH: So when you're working with raw SQL, you end up hitting copy-paste a lot because it's natural to have logic that ends up wanting to be used in multiple places. So you had this little library on your hard drive typically of SQL statements that you've written in the past and they're called things like monthly recurring revenue to_final_final and you like remember that it's there. When you need it in the future, like you go there and copy-paste some chunk of that into the like the new thing that you're doing, and that is a thing that many software engineers have done in the very beginnings of their careers and then had that burn them and realized that like there's a real wave of doing this thing.

But data analysts writing SQL without DBT kind of are never able to progress beyond that, because SQL itself doesn't give you tools to create functions or create any kind of reusability whatsoever. You really do need a framework sitting on top of SQL if you are going to think about it as like a first-class language that you are involved in writing.

[00:31:28] JM: Why can't we just version control our SQL files?

[00:31:32] TH: That's a part of it. Like instead of having that folder on your computer be just local to you, it can be a folder that's shared across your team, but that doesn't help you with the reusability part. The core problem is not really that like people aren't checking their SQL into git. It is that there is no abstraction present in SQL such that if you want to define something once and use it many places, you can do that.

You asked what's the experience like. The other part of the experience is not just like the creation part where you're like going and finding old scripts and copy-pasting stuff together. It's also the maintenance part, which is eventually the bigger problem, because when you have that copy-paste workflow, if somebody says, "Well, our accountants say that we have to define monthly recurring revenue slightly differently because I think it's called ASC606, which is a new accounting standard and like we have to do our revenue recognition differently than we did last year." It's a freaking nightmare. You have to go to every single place the you've ever written script that defines revenue and you have to redefine it. Then your pull requests to that git repo that contain all of the stuff, literally, like many thousands of lines long and no one wants to review that pull poor request because it's incredibly boring. SQL only pull requests are so low leverage that like you literally can't get anyone to pay attention to them.

[00:33:02] JM: One simple but clarifying example for me when I was looking at DBT was the setting of environment variables and just saying I have this line of code in my DBT precompiled. Eventually will become SQL information, and it says if the environment is testing, do one thing. If the environment is production, do another thing. Just the simple fact that you can have an environment variable template in DBT. Is that a good example of the productivity that you get out of DBT?

[00:33:43] TH: Yeah. Certainly. That's one of – It's like such a stupid thing, right?

[00:33:48] JM: By the way, is there a way to do that in SQL? Can you just like pull from an environment variable?

[00:33:51] TH: No.

[00:33:52] JM: Okay.

[00:33:53] TH: No. I mean, there's no such thing as a variable.

[00:33:56] JM: Right.

[00:33:57] TH: I mean, there's like proprietary scripting type SQL like TSQL or PLSQL or stuff like that, but like that's not a first-class thing that's supported consistently across databases. Again, back to my like we have seen a lot of traction with DBT and yet like we didn't invent any of these ideas. Terraform was just becoming really popular when we wrote the first line of code for DBT and it was just like, "Oh! Obviously, we need Terraform for data," and the ideas that Terraform popularized, which are like – Again, I'm not a Terraform user, so I understand the ideas behind the product but can't talk about it in detail. But just the idempotence, where like every time you run the thing, you can be confident that your achieving the same output.

The fact that like environments are a first-class thing, you should be able to like build the same thing across multiple different environments. It was very easy for us to build in environment awareness into DBT because it seems so obvious that like that's how the thing should work. But in most tooling for data analysts, like the concept of environments doesn't exist. If you want multiple environments, you like spin up multiple versions of whatever the thing is that you're using, which is like very silly. Environments are really just name spaces in DBT. You just like say, “Build it into this environment versus this environment.” Because the data warehouse is this like infinitely scalable compute resource, namespaces is good enough.

[00:35:31] JM: DBT is used to build and test data models. What is a data model?

[00:35:38] TH: Yeah, that's a fair question. That’s such an overloaded term that it's everything and nothing. The way that we think about a data model is it is a consistent interface to the data. If you're thinking about things in a Kimball-style Star schema or Snowfl ake schema, a data model is your fact orders table or your dim customer’s table or any of the intermediate stages that your data had to pass through in order to produce those eventual data models.

The data model is like the core unit of work in DBT. You can also talk about a machine learning model. That is a very different thing and there are a lot of people that are anointed us that we like picked an overloaded word. But the funny thing is you use the words that you use and then like your community grows around them and like we don't get to choose now that like we can't make it called something else. It's like what everybody calls it.

[00:36:38] JM: Is a data model a table? Is a set of tables?

[00:36:41] TH: Yeah. Okay. How does it map into the warehouse? A data model as a single table in a warehouse, or rather is zero or one tables in a data warehouse. The DBT can materialize data models in four different ways, or you can specify your own custom materializations. But it can build a table which gets rebuilt from scratch every time you run DBT. It can build a table that builds itself incrementally. It can build a view or it can have what's called an ephemeral model, which instead of actually being manifested in the data warehouse at all, the code from that model is pulled into downstream models as CTE's, common table expressions. That is a way to defi ne a logical chunk of code that you can test that doesn't need to have any footprint at all inside the data warehouse.

[00:37:32] JM: DBT builds a DAG for you, a directed acyclic graph. Is the DAG connecting different data models to one another?

[00:37:43] TH: Yeah. DBT has – All data processing is built around a DAG. You have your source data all the way in the left-hand side and slowly as you move from left to right gets more processed. DBT knows about two different types of nodes in that DAG. The first type of node is a data source. So DBT understands – You configure them in a YAML file where your original source data is from, and a lot of times that's from data engineers that serve the data up. It maybe it's from off-the-shelf tools like Stitch and Fivetran, but you register all them with DBT and then you build data models on top of that. DBT is able to – Using calls to its ref function, DVT is able to construct the DAG out of all of those nodes and then it's able to paint a visual representation of that DAG so that you can traverse it visually so that you can like drill in to any of those nodes and inspect what's going on there.

[00:38:47] JM: I might have raw data at the beginning of my DAG and then I have one data model that that date is passing through, basically, like a set of SQL statements or one SQL statement that the raw data is passing through and then I might have another node in the DAG that is another SQL statement and the raw data is essentially passing through these different nodes.

[00:39:13] TH: Yeah. That's really good way to think about it. There's lots of prior art in how do you go about transforming data in a warehouse to make it useful. People have been doing this since at least the 90s. That's when Kimball and Inman were like writing books about this stuff. When we started doing this back in 2015, 2016, we actually had not read all of those books, and so we're trying to figure it all out for ourselves. It was kind of the Wild West. You just like kept building stuff until you got to the answer you wanted and then you call that done.

So maybe a simple set of transformations was two or three deep. Something more complicated is maybe 10 or 12 deep. But overtime, we've adapted these like standard layers that the data passes through so that like these layers become kind of the internal and external APIs to your dataset. The two standard ones that we think about are the staging layer and the mart's layer. The staging layer is like providing a version of a data that is still at the same level of granularity as the source data, but it has had a bunch of cleanup done to it. Maybe there are keys that have been resolved. There's duplication that's been handled. It's like give me a clean version of the source data. Then at the mart’s layer, you get to do all of the like business logic transformation that makes the eventual datasets map more to the business processes, then the source data.

[00:40:49] JM: I spoke to somebody recently who said that you can think of these DAGs, these data modeling DAGs as – You can kind of think of them in two ways. You can either have where the nodes are data sources and the edges represent transformations on the data, or you can totally invert that and say the nodes are the transformations and the edges are the actual intermediate data sources. Do you agree with that kind of comparison between two different ways that you could represent a DAG of data?

[00:41:25] TH: I understand what you're saying there. It doesn't map on to my brain as being like a super important distinction. That may just be because I am like too stuck in my own world.

[00:41:35] JM: Is either of those closer to –

[00:41:37] TH: We think about the nodes in the graph as both the transformation and the dataset.

[00:41:43] JM: Got it.

[00:41:44] TH: Because the transformation, the code behind the transformation fully describes the dataset. This goes back to the idempotence in the like Terraform nature of DBT. If you know what the code is, then you can – We actually recommend that people blow away their entire like transformed data warehouse periodically and rebuild it from scratch. I mean, that might not be true when you're dealing with like very large dataset sizes, but like sometimes for like less mature DBT projects, it is actually good to blow them away and rebuild them from scratch because the code fully describes what the dataset will look like at the end of it.

[00:42:24] JM: Got it. The transformations in DBT are reusable across projects. This is one of the advantages of using DBT, is you can leverage the models that other people have built. You could take them off of GitHub or you can reuse transformations that other people throughout your organization have built. Describe this reusability of data transformations.

[00:42:53] TH: This is like I think one of the dumbest things ever, is that since before I was born since like as long back in computing history as like I'm aware of. There has been the ability to create libraries and import those libraries into other libraries. It's like a fundamental things to like how you write software code. By and large, that's not how data analysts have worked. Data scientists, certainly, like if you're using Python or R, like that's a standard thing for you. But in the SQL world, there's not been any concept of like a package or an import statement, and that's insane.

Really, one of the first – I mean, back in 2016 when we were like writing the very first versions of DBT, the package manager was one of the very first things that we did. It's very hard to like reverse engineer a package manager into an existing language. I think that there many good examples of like how that hasn't always gone well.

So that was something that was really important to us from the outset. Our mental model has always been data analyst should work like software engineers. So we needed that capability from the ground- up. Yeah, you can internally share code across your organization, and that's mostly relevant like when you're in a particularly large DBT deploy. But also, there are more and more like very standardized datasets that people are using. If you are using Stitch plus the Stripe API, or Fivetran plus the Strip API, your data is always going to look at the same. So there's no particular reason why somebody can't publish a set of data models that consume that data and apply standard accounting best practices to it to get to a place where like you're doing monthly recurring revenue reporting. So that was the original use case for DBT packages. We've built a ton of them and we're starting to see the community build more.

My hope over the medium-term is that data analysts won't have to start from scratch, like you get hired at an organization, you spin up an off-the-shelf data stack, like you don't start within a DBT project. You start with like the best in class Stripe transformations in Salesforce and whatever else. You just like start there, and then you start customizing it for your business, because that's what software engineers do. They don't start from scratch. They start with a common set of tools.

[00:45:24] JM: What does the DBT package manager do?

[00:45:26] TH: You publish the Strike package, let's say. The package manager inside of DBT allows you to reference that package inside of your project and resolve it to a particular release and then it grabs the code for you at that release and it allows you to reference it inside of your project.

[00:45:43] JM: Got it. After somebody publishes the DBT Stripe package, I could use that for – I think one example – You can correct me if I'm wrong, but like Stripe, I think it gives you – For your transactions, it gives you just a bunch of JSON like for each transaction and you may want to take a subset of those JSON fields and materialize them as a SQL table.

[00:46:12] TH: That's completely reasonable. That's kind of like a data prep thing you might want to do in a package. A lot of times, a product like Stitch or Fivetran will have done that for you already. The things in this vein that people find themselves writing within packages are things like – With Stripe, you can charge people monthly or you can charge people annually for like other durations. As a SaaS business, you always want to be able to report on your monthly recurring revenue. There are standard ways that you take multi-month contracts and split them out over the course of the period of months that represent that contract.

So the modeling required to do that is kind of like the most complicated thing in the world, but at the same time it's like reasonably complicated and it's easy to just grow up. So the package, the Stripe package that we published a long time ago, actually allows you to do that set of transformation such that you produce this output table that is all of – It is the output table that you need to answer all of the questions in your business about monthly recurring revenue.

[00:47:23] JM: Nice.

[SPONSOR MESSAGE]

[00:47:33] JM: As a company grows, the software infrastructure becomes a large complex distributed system. Without standardized applications or security policies, it can become difficult to oversee all the vulnerabilities that might exist across all of your physical machines, virtual machines, containers and cloud services. ExtraHop is a cloud-native security company that detects threats across your hybrid infrastructure. ExtraHop has vulnerability detection running up and down your networking stock from L2 to L7 and it helps you spot, investigate and respond to anomalous behavior using more than 100 machine learning models.

At extrahop.com/cloud, you can learn about how ExtraHop delivers cloud-native network detection and response. ExtraHop will help you find misconfigurations and blind spots in your infrastructure and stay in compliance. Understand your identity and access management payloads to look for credential harvesting and brute force attacks and automate the security settings of your cloud provider integrations. Visit extrahop.com/cloud to find out how ExtraHop can help you secure your enterprise.

Thank you to ExtraHop for being a sponsor of Software Engineering Daily, if you want to check out ExtraHop and support the show, go to extrahop.com/cloud.

[INTERVIEW CONTINUED]

[00:49:07] JM: Can you give me a hypothetical use case example for how DBT is helpful for an organization? Who at the organization will be using it?

[00:49:18] TH: Yeah. A lot of our very early adapters were in the direct to consumer e-commerce space and they tend to have fairly large data teams take a brand that is at scale that maybe has 500 employees at the company. They may have a dozen folks or 15 folks on the data team and they're starting to run into the kinds of scalability problems that we were kind of talking about at the beginning of this conversation.

You take DBT and you drop it into that environment and all of a sudden the data analysts in that team, the dozen or so of them, they’re no longer like this like group of individual vigilantes that are like solving finance or marketing are whatever problems. They’re this like cohesive group of people working together to curate the knowledge inside of that company.

When you don't have DBT as a data analyst at that organization, you – Or essentially a service provider to the business units. The head of marketing asks a question about customer acquisition costs. So you sit down at your computer, you write a bunch of SQL, you give the answer back. The other people on your team don't have any way of benefiting from your work. Your work isn't like incorporated into some larger hole.

So that team because they can all be working together on this DAG of data transformations that slowly overtime gets more and more sophisticated and better tested. All of a sudden, they are involved in each other's work. They're collaborating together. They're reviewing each other's pull requests. Honestly, the thing that DBT does and that we are involved in is an organization more than anything is like driving behavior change across teams, like taking data analysts from low-leveraged loan wolves to high-leveraged like teams that work together.

[00:51:27] JM: The modern data warehouses, Red Shift, Snowflake, BigQuery, perhaps Apache Spark. Give me your breakdown for why different companies use different data warehouses.

[00:51:42] TH: Yeah. Okay. Let me see how many people I can piss off.

[00:51:44] JM: Yes.

[00:51:46] TH: I have a real soft spot in my heart for Red Shift, because Red Shift was the first data warehouse in the cloud that you could buy for 160 bucks a month. Actually, I own a piece of art called red Shift. I bought it on everything but the house. It was like in an estate sale and it was like made in the 1970s and it's hanging in our office. It's called Red Shift. I love it.

[00:52:09] JM: What is it?

[00:52:10] TH: It's a painting.

[00:52:11] JM: Of what?

[00:52:12] TH: It's abstract. It's like squares.

[00:52:15] JM: Okay. Somebody told me that the reason it's called AWS Red Shift is because it's people shifting away from Oracle, and Oracle is red.

[00:52:27] TH: Seriously?

[00:52:27] JM: That's what somebody told me, or maybe I read on Wikipedia.

[00:52:30] TH: I didn't know that.

[00:52:31] JM: I don't think it's true, but I heard that. I thought it was hilarious. I think it has more to do with some kind of celestial phenomenon.

[00:52:39] TH: Sure.

[00:52:39] JM: Right? Is it some kind of like –

[00:52:40] TH: I mean, Red Shift is a celestial phenomenon that I am woefully underqualified to say anything about it at all.

[00:52:47] JM: It's a Red Square.

[00:52:47] TH: Yeah. It's like a series of squares. Yeah.

[00:52:48] JM: You see abstract red square in the sky.

[00:52:51] TH: Yeah. I'll send you a picture.

[00:52:53] JM: Okay.

[00:52:54] TH: But Amazon had like a phenomenal amount of success with it in the initial years because it was really the only game in town in the cloud. I think that they might have under invested in the product for a little while. As a result, both BigQuery and Snowflake, they made some architectural improvements, the primary one being the separation of storage and compute that really fundamentally changed how users can like access this resource in Red Shift historically, and they've done some work to change this, although I don't think that they have caught up really.

Red Shift users always feel storage constraint. You just like you want storage to feel abundant and cheap, and on Red Shift it’s just like that’s not. The next generation was really Snowflake and BigQuery. Both are phenomenal products. I don't know that anybody has public data on like BigQuery adaption or anything like that. I think that that's like just inside the big like Google Cloud numbers reported. Yeah. Because we get to see DBT usage numbers, we see BigQuery growing a lot.

Snowflake is the most commonly used data warehouse with DBT. You can – I don’t remember all of the details around that Snowflake raised recently, but the company was valued at $12 billion. It's very easy to get data points that show just how big this market is and how quickly it's growing.

Spark is a little bit less in our world today, because I think that Spark SQL is not the very most – It's not the primary – Or at least historically I don't think it's been the primary focus of the Spark maintainers. I think that's changing. I don't have like hard data points there, but I've heard that the SQL interface is an increasing priority for those folks and we are very excited to see progress in that area.

[00:55:02] JM: Do you have any idea how BigQuery compares to Snowflake?

[00:55:08] TH: I don’t have like heads up benchmark information for you. I've read stuff in the past that has essentially – Like at the end of the day, we know certain things as an industry about how to process data efficiently. My general belief is that like probably the people at Snowflake and the people at Google are roughly at the same level of sophistication in terms of like how efficiently can you use compute resources to like process a query. I don't think that there's like this fundamental difference in how good the warehouses are. I think that the real difference is how they present themselves to the user. Snowflake took the opinion that you want to be able to have direct control over the hardware resources that are processing your queries.

You can say like throw more hardware at this thing. BigQuery takes the assumption that you don't want to think about that and it's going to use its query planner to analyze your query and then look at the amount of data that it's going to have to process, and then transparently, to you, it's going to make decisions about how much hardware to throw at that.

I don't have a reason to believe that either one of those is better than the other. I do think that Snowflake, it presents itself as a little bit more natural of a transition from prior generation technologies, because people are used to thinking about like boxes, and BigQuery makes this assumption that it's just going to like handle it for you and I think that people sometimes feel like they want more control than that. But maybe that's a very outdated way of thinking about the world, and maybe Google does know best. I honestly don't have a strong opinion there.

[00:56:57] JM: Fishtown Analytics, your company, started as a consultancy. Now, I believe you have a cloud product, which is the main focus in terms of a business. What's the cloud product do?

[00:57:13] TH: Yeah. We still do consulting on top of DBT in the same way that like most companies that run open source projects do. We help companies implement and train and run conferences and things like that. But the cloud product certainly is a big focus for us.

DBT is split into two primary codebases today; DBT Core and DBT Cloud. Core is Apache 2. Use it. Do whatever you want. That provides the core templating functionality. It provides a command line interface. It provides a server to respond to requests and it provides all of the functionality to connect to databases and run that SQL. It does not ship with anything stateful. DBT itself is stateless. It doesn't ship with a user interface. It doesn't ship with a scheduler. There's no way to log into it.

DBT cloud provides this management and user interface layer on top of Core that gives analysts an experience that more closely matches the way that they want to interface with software.

We collect telemetry on both products, and as far as we're able to see about a third of the DBT Core user population uses cloud.

[00:58:33] JM: That's a good sign.

[00:58:34] TH: Yeah. I mean, it's also like dirt cheap and there is a like free forever plan as well. We make it super easy. We want anybody who finds the management layer useful to be able to use it.

[00:58:46] JM: When AWS inevitably builds AWS DBT, the six letter acronym, what are you going to do?

[00:58:56] TH: Yeah. They don't have to build it, because they can just use ours. Amazon is in particular known for implementing large open source projects internally and I don't think there we're really like –The community is like quite at the size to like get there notice quite yet. But I think that the way that things are growing for us, that's not a crazy thing to think could happen.

The thing that we care about for Core is the widest possible adaption. We care about this, like effecting this change in the world that we want to see. If that means that AWS gets involved in like AWS or any of the other like hyper-scale cloud providers, gets involved in DBT in some way, or even like gets involved in pushing people towards this way of thinking about the world, that's a win for us.

The way that we monetize is not with DBT Core. It's with the management layer on top of that. By and large, AWS is not in the business of building user interfaces and they're also not in the business of building software for data analysts. They primarily sell to software engineers. We're not – If we were selling a backend piece of technology like Elastic or something in that category, I think we would think about this problem a little bit differently, but like ultimately we are charging money for a user experience.

[01:00:29] JM: Right. Yeah, because there's just not that much data to store. It's like a –

[01:00:36] TH: No. We store literally just zero data.

[01:00:39] JM: Oh! You don't even store any data.

[01:00:40] TH: Well, I mean we like have a Postgres database that stores like users and logs of run results. But like the data warehouse stores the data.

[01:00:50] JM: Right. Got it. What about the user's queries? Those don't get stored or cached or anything?

[01:00:59] TH: Not today. Today, all of the user's code is stored in their git repo. In the same way that like CircleCI would connect to your git repo and execute the code as a part of a job. DVT cloud does the same thing. Then it stores logs of the runs.

[01:01:18] JM: Got it.

[01:01:19] TH: And can be configured to store those logs in your S3 bucket. The way that the product is designed already has a like low footprint in your data center, but we're hyper-focused on giving people the control that they need to be able to manage that stuff. We ultimately don't want to be the custodian of your data if you don't want us to be.

[01:01:45] JM: That's smart, and I think that's a smart way to compete. I think that's how we're going to see more and more of these open source companies compete. Just compete on the UI layer.

[01:01:53] TH: Yup.

[01:01:53] JM: Compete on the developer experience layer. Amazon's got some skeleton crew working on their AWS version of your product. They're there all doing infrastructure stuff. None of them are designers or whatever. You can out execute them on design part.

[01:02:10] TH: Yeah. Since I mentioned that before, can I give a plug to –

[01:02:13] JM: Yeah.

[01:02:13] TH: Have you talked to anybody on the show from Replicated?

[01:02:17] JM: Yeah. Twice.

[01:02:18] TH: We love their product. We'd not be able to –

[01:02:20] JM: All right. Shout out. They're a sponsor. Wow!

[01:02:21] TH: Oh, wow!

[01:02:23] JM: That's some great. Unpredicted non-sponsored content. I'm handing Tristan a $1000 bill right now.

[01:02:32] TH: We use Replicated to deploy on-prem to customers' clouds.

[01:02:36] JM: Beautiful shout out.

[01:02:37] TH: Legitimately would not be able to do that without that product. That would be out of our reach as a small organization.

[01:02:43] JM: Nice. I bed you'd get a discount on your next enterprise contract. All right, we're wrapping up. How do data pipelines look different in five years?

[01:02:52] TH: Oh geez! I think that they're better documented, they're better tested, they're more end-to-end. I think that like the pipeline will be understood all the way up to the interface with the user as supposed to just like being the stuff that happens in the compute layer. It will extend to the BI layer as well, because one of the biggest support burdens of the data analyst team is users saying like, "Why is my dashboard broken?" and the answer is like, "Well, some transformation upstream broke and like the data is not current." You really need that DAG to like extend all the way to the user. They will be more real-timey. There's work happening in that space from the Druid folks and Imply. There's work happening – There's a new company that just publicly launched like two weeks ago, Materialize.

[01:03:45] JM: Sure. Another recent guest.

[01:03:48] TH: Oh! Is that right?

[01:03:49] JM: Yeah.

[01:03:50] TH: The thing that I'm interested in with both of those products is that they think of SQL as a first-class API for a streaming platform and like full ANSI SQL support, including joins. Historically, joins have been the big problem for any streaming platform that says it has a SQL interface. Any data analyst will tell you that like if you can't give me joins, I’m like it’s not that useful. I think that you’re going to see the data stack and data pipelines, because they get more real-timey, they get used in workloads that are not just analytics-focused. They are also operational.

I think that we are using analytics to kind of bootstrap the like data infrastructure for the entire company, because it has lower latency or like less restrictive latency requirements. But as the latency gets better, you’ll see operational use cases happen, which are going to put SQL in the center of like running the operations for an entire business.

[01:04:58] JM: Great way to close off. Tristan, thanks for coming on the show. Great talking.

[01:05:01] TH: Thank you. It was a lot of fun.

[END OF INTERVIEW]

[01:05:13] JM: Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra’s architecture and its benefits, and we’re happy to have Datastax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

Datastax provides Datastax enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. Datastax enterprise enables teams to develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies including Cisco, Capital One, and eBay run Datastax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce.

To learn more about Apache Cassandra and Datastax's enterprise, go to datastax.com/sedaily. That's Datastax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to Datastax for being a sponsor of Software Engineering Daily. It's a great honor to have Datastax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[END]