Agentic coding in analytics engineering (w/ Mikkel Dengsøe)
1 message

Analytics Engineering Roundup <analyticsengineeringroundup+the-analytics-engineering-podcast@substack.com> Sun, Sep 7, 2025 at 8:01 AM
Reply-To: Analytics Engineering Roundup <reply+2uy1zy&9qden&&1ed96cf069b2531c5d71d26b1316645a5167099a179b8a686225e389c22049a3@mg1.substack.com>
To: rayserrano2735@gmail.com

Agentic coding in analytics engineering (w/ Mikkel Dengsøe)

The cofounder of SYNQ discusses his tests (and tips) with agentic coding tools

DAN POPPY
SEP 7

READ IN APP ↗

What does agentic coding look like in analytics engineering? Mikkel Dengsøe, co-founder at SYNQ, recently wrote a series of posts on his experiences as an analytics engineer with agentic coding tools. In this episode of The Analytics Engineering Podcast, he walks through a hands-on project using Cursor, the dbt Fusion engine, the dbt MCP server, Omni's AI assistant, and Snowflake.

Tristan and Mikkel cover where agents shine (staging, unit tests, lineage-aware checks), where they're risky (BI chat for non-experts), and how observability is shifting from dashboards to root-cause explanations delivered to the right person at the right time. Along the way: practical prompts, why "one model at a time" keeps you in control, and a testing philosophy that avoids alert fatigue while catching what matters.

To see real-world use cases of agentic coding and to learn directly from data and AI leaders, join us at Coalesce 2025 in Las Vegas, Oct. 13-16.

Please reach out at podcast@dbtlabs.com for questions, comments, and guest suggestions.

Listen & subscribe from:

The Analytics Engineering Podcast
dbt Labs, Inc.
Podcast

- Spotify
- Apple Podcasts
- Amazon Music
- TuneIn
- RSS feed

Key takeaways

Can you talk a little bit about your background?

Mikkel Dengsøe: Yeah, so I can start from the beginning. I've been in data for, I think it's coming up to 15 years now, and started my career in data at a Danish shipping company, which was very much zero to one. When I came in, there was no data warehouse, and the only way we could know how many containers were shipped was by an IT guy pulling that out of the system every six months. I then spent two years there building up their data warehouse on SQL Server, which was super fun. After that, I spent five years at Google, which was a very different gear.

That's a natural transition. Just global shipping company straight to Google.

Exactly. And that was very much a hundred-to-end where, in my case, I worked with the ads data and you get a perfectly curated data table that you can work with and everything kind of works. Then after that I joined a company called Monzo. For those who are not familiar, it's a scaling fintech out of the UK and that was very much the one to a hundred. When I joined we were 30 data people, but scaled to a hundred over two years. We had 10,000 dbt models and we built every internal tool under the sun for dbt. Super interesting. And then three and a half years ago I went on to found SYNQ alongside Peter and Steve, which is a data observability platform.

Tell us a little bit more about SYNQ.

We are a data platform that primarily works with companies using tools like dbt already, but have issues going from important data to business-critical data. That might be customer-facing dashboards, machine learning models, or something else. They want better monitoring—we often deploy anomaly monitors—and they also want workflows such as incident management for when things go wrong. We were founded in 2022, so now we're in early stages of working with scale-ups and startups, and now also onboarding enterprises and larger companies. It's been a fun journey.

In your series of blog posts, you went through the modern data stack and said, “What's the most current version of this tool and how effectively can I AI-ify that?” Whether that's using Cursor to build dbt models or using the agent experience inside of Omni—what made you decide to get into this and write about it?

The first part of it is just: it's super fun to tinker with these tools and try them out. It's magic. And we were also building an MCP server at SYNQ, so I had a lot of interest in seeing how it works with others and what we can learn. It was also driven by being able to have conversations with our customers. When they ask about it, being able to speak from the point of view of having actually tried this and seen what works and what doesn't.

The early days of using Redshift were such a visceral experience relative to what came before. If I hadn't interacted with it directly, I wouldn't have understood how big a state change cloud data was. This feels like another one of those moments: if you don't have hands-on experience, you're not going to really get it. Fair?

Spot on. And I think pretty much every data team should be doing this unless they have a very good reason not to. The risk and the stakes can be pretty low if you use it for internal workflows like data modeling and writing tests. You're still in control. I recommend everybody do it.

What tasks did you try to accomplish?

It's three different blog posts: the data modeling part, the testing part, and then exposing it in Omni's AI agent where people can ask questions about the data. There's a fourth post: once the data is live, how can you use the SYNQ MCP to do things like root-cause analysis and planning changes. I started with data modeling. I had raw data from different JSON sources, some XMLs, some profiles—extracted and put into Snowflake—and then did the data model.

So the data was already loaded into Snowflake?

Yeah, exactly. For the data modeling, I started from the sources and then worked through staging, marts, and finally metrics using the semantic layer. Each step looks a little different when you use AI tools because the behavior differs. In terms of tooling, I used Cursor with the dbt-MCP plugged in. If you're not familiar, dbt-MCP lets you, via prompt, interact with dbt tools—execute `dbt build`, get models, or get everything upstream of a given model—so you can chain work without explicitly doing it.

Cursor + dbt-MCP. What model did you use?

I just used the default in Cursor, which I believe is Claude. There's an important distinction: Cursor is really good at writing code, but it can't execute queries on your behalf. If you want to extract raw data and query Snowflake to get rows out, you have to do that in Claude Desktop. That became key. Early on, as I built models, the first thing I did was get a snapshot of sample data from Snowflake—10,000 rows of a source. I fed that into Cursor and said, “These are examples of what this data looks like.” Using that data, Cursor could model in a clever way. For example, a column called `quarter` like “2025 Q1”—Cursor understood to translate it into a datetime and do the transformations.

I've used the dbt MCP server a decent amount—less in Cursor, more in Claude Desktop. Your stack was Cursor + Claude models + Claude Desktop. And Cursor cannot directly execute queries in Snowflake, but Claude Desktop can. Is that because there’s tool use Claude has that Cursor doesn't?

I believe so. In Claude Desktop, if you write queries against dbt-MCP, Claude can visualize a graph, show outputs of a SQL statement, etc. Cursor, as far as I know, couldn't. My middle ground was to take sample data out of Snowflake, put it into a CSV, and feed that back into Cursor so it could look at raw data.

As part of its own context window?

Exactly. That was key for my workflow. Then when I wanted to write unit tests, I could use real data examples from the sample. Or when automatically documenting the data, I asked Cursor to specify examples in the docs based on the most common occurrences within a column. Letting Cursor peek at raw data was a core pillar.

It's a little hacky, right? Cursor should really be able to interact directly with Snowflake or Databricks to investigate the shape of the data. Agents should be empowered to do that.

I would say so. There might be a way I didn’t know about, but I patched the gaps by uploading into the context window.

So that's the state of the art today.

Seems so. To be clear, I think the limitation is IDE differences—Cursor vs. Claude Desktop—rather than dbt-MCP itself.

Once you had sample data in context, did you have to suggest conversions, or did it naturally do them?

It got the defaults pretty right, but I guided it on what I wanted from the source data. I wanted control over everything, so I asked it to do one model at a time rather than auto-generate a whole stack. That way I could review each step and stay in control.

Your prompt workflow was "Build me a model with this name that stages the data from this table," basically?

Yeah. When it proposed code I didn't like, upstream it was usually simple (regex to parse dates, etc.). Downstream, in marts and metrics, I started describing my ideal data product: user jobs-to-be-done and the final output. That's when Cursor got creative and invented metrics I hadn't anticipated—like "apartment price relative to time on market." I pruned ones I didn't want, but some were good surprises.

Which layer did it help most?

Testing. Modeling was good—especially staging—but testing accelerated significantly. SQL is a bit like English; for simple datasets you can express intent easily. Testing can be much harder and more verbose.

Roughly how much more effective did you feel?

Modeling: multiples faster. It nailed the tedious parts—regex, casting, pass-throughs—so staging/intermediate layers flew. In marts/semantic metrics, the benefit was brainstorming. It helped me think of metrics I wouldn't have.

Did the dbt Fusion engine help?

Yes. Fusion shows lineage and whether a column is pass-through. For example, if a column is pass-through with no transforms, don't add another `not_null` or `unique` if there's one upstream. I bounced between the IDE to check this and codified it as a testing strategy. That's already top-10% testing hygiene.

Any MCP feature requests surface?

The more context and tools the agent has, the more it can do. In the fourth post, for root cause analysis, we used the SYNQ MCP. We collect all your Git commits and have history, so the agent could correlate recent code changes with incidents. Requests depend on the job at hand.

Let's move to testing—why was it the most additive?

Testing is hard; many teams don't know how to do it and alert fatigue is common. A huge share of tests we see are `not_null`/`unique`, which doesn't reflect real data risks. First thing I did in Cursor for testing was provide our internal testing philosophy as guidelines: test heavily at the source, don't retest pass-through columns, focus on business and metric anomalies in marts. That worked really well. For sources and staging, it generated relevant tests. Then for marts, I asked for unit tests and gave it a thousand sample rows from Snowflake. It wrote very relevant unit tests I'd otherwise spend a lot of time on.

Examples?

Simple ones like: when you pass a string value in the date column, does it transform correctly to datetime and match the expected format? These just worked. Then at the metric level, it looked at raw data and proposed assumptions—like square-meter price should be between X and Y—sometimes segmenting by postcode. Very thoughtful, though I'd replace static thresholds with anomaly monitors so they don't go stale as prices move.

So at least 5× on testing?

At least. Apart from swapping static thresholds for anomaly detection, it nailed testing and did so in a lineage-aware, layer-appropriate way.

Tell me about the BI layer.

Many teams start at the BI layer with a chat interface. I think that's risky because it's used by business users and you only get so many chances before trust drops. I moved into Omni. You create a “topic” (a data model you can join with others) and then specify an AI context: instructions for how the LLM should behave. For example: if a user asks about price, always return square-meter price; never make up fields not present in the mart; if asked about provenance, mention the source. Writing AI context is a new skill for our industry.

Were you using Omni’s AI assistant to create assets faster, or to let users self-serve?

The latter—so users could ask questions instead of going to a dashboard. It could have been any BI tool with similar functionality; we just use Omni internally.

And how was the experience as a consumer?

Amazing when it works, but I'd hesitate to give my VP of Marketing access. It gets things wrong maybe one in five times, and it's not obvious why if you're not a data person. For analysts doing exploratory work, it's great—they can inspect and dig in. I wouldn't replace company-wide dashboards with a chat bot yet. Omni does log freeform queries and feedback, so there's a path to iterate the AI context over time.

The last thing you did was use AI plus SYNQ to monitor production infrastructure. What does observability look like in the future? Historically it's looked like dashboards—Datadog for data pipelines. Is it just more effective monitors, or fundamentally different?

Fundamentally different. We’re heading to a place where observability tools can tell you what's wrong at the right time, with just the right context, delivered to the right person—inside or outside the data team. Done well, there may be few dashboards; instead you get an LLM-summarized root cause delivered from a monitor that might be auto-created. Less “active tool you poke at,” more “proactive explanation.”

Still technical observability (pipelines/data issues), or business observability?

More the former. Teams at the edges—Sales Ops managing Salesforce, engineering teams creating web events—often need to be notified about data issues. Business KPI movements require a different experience for marketers, etc.

Automated remediation?

Gradual. You can imagine an issue occurs without a dedicated test; the system proposes a new test. But 80% of issues come from root systems elsewhere (someone typing in Salesforce), and closing that loop is still hard. In the article's fourth part, we had a data issue and I asked the SYNQ MCP through Claude Desktop to do root cause analysis. It walked the same steps a data person would: inspect the model, check errors, examine lineage and upstreams, review recent commits, and documented each step to the root cause. That works now.

At the beginning you said there's no good reason not to use these tools today. What reasons do you hear for not trying?

People are busy. But if you look at a risk curve, lowest risk is modeling and testing—you're in the driver's seat and it boosts productivity. Higher risk is replacing your BI tool with a chat bot; higher still is customer-facing experiences. The first two are hard to argue against.

Enterprise IT approvals might be one blocker—approved models, data access, etc.

True. For example, our MCP can query raw data to detect if an issue happens in a segment, and enterprises might hesitate there. Also, "MCP" as a term can be confusing. But it's actually simple and explainable, not a black box. Setting up dbt-MCP can still feel hacky in enterprises; if it lived natively in cloud environments, it'd be easier to adopt.

You can set it up locally—no permissions/procurement—and just play. We also shipped the MCP server as a remote MCP in cloud, though that introduces auth/permissions considerations.

If I had to pick a persona, it's the analyst. Analysts have had a tough decade: more tools, harder workflows, less time to tinker. MCPs and AI workflows are a turning point. At Monzo, we had a philosophy that you should be able to have an idea on your commute and have it implemented by midday. As we grew to 10,000 dbt models and long CI checks, that faded. I can see a world where this returns. MCPs can help. I'm excited.

I love that. Analytics engineers think “infrastructure, correctness.” Analysts think “idea to validation fast.” Excel was always the analyst’s best friend because it's fast and flexible. MCPs make it easy to plug tools together and get answers quickly again.

One company we work with—Voi, a scooter company out of Sweden—has a strong data leader, Magnus, who is very bought into metrics. Their data team doesn't produce dashboards; they produce metrics. In an AI world with MCPs, flows, and curves, that's a clear decision.

I believe there's no such thing as the wrong BI tool—different tools have different trade-offs. Probably true for models/IDEs too: Claude Desktop vs. Claude Code vs. Cursor—no single “right answer” as long as the underlying context and metric definitions are shared.

Agreed. What really matters across workflows: consistent metric definitions, documentation for columns and fields, and high-quality data. Those foundations matter even more when an LLM is in the loop; you may not have a human sanity-checking every result.

Chapters

- **07:17** — Tooling: Cursor + dbt MCP server + Snowflake + Omni
- **09:38** — Sampling real data into the agent's context
- **13:14** — Modeling workflow: one model at a time
- **15:14** — Where agents help most: testing > modeling
- **18:10** — dbt Fusion engine: lineage-aware checks, fewer redundant tests
- **19:50** — Feature requests and root-cause via commit history
- **20:57** — Testing philosophy: source-heavy, pass-through aware, metric-level
- **22:49** — Unit tests from samples; thresholds vs anomaly monitors
- **25:10** — BI agents: great for analysts, risky for broad rollout
- **31:54** — The future of observability: explain first, dashboards second
- **36:10** — Adoption curve: safe places to start
- **40:49** — Analyst superpowers return
- **42:04** — Metrics over dashboards

This newsletter is sponsored by dbt Labs. Discover why more than 60,000 companies use dbt to accelerate their data development.

Book a demo