What does dbt do?

The talk of the (fish)town

Justin
Sep 27, 2021

The TL;DR

dbt (no capitals) is a tool for **transforming and organizing data in your warehouse**. It helps data teams get raw data ready for analysis and impact.

- **Data models** are the core of effective data teams – they map business concepts onto cleaned, organized data
- dbt helps data teams use SQL to **build useful, documented data models** that the rest of the company can benefit from
- Core concepts in dbt: **models**, **docs**, **seeds**, and **runs**
- dbt isn't quite like anything on the market, and they've **partnered with tools across the spectrum**

The open source dbt product has seen almost fanatical levels of support from the engineering and data community; they also recently raised a $150M Series C.

What's a data model exactly?

> 🔮 **Dependencies** 🔮
>
> Understanding dbt will be *a lot* easier if you get comfortable with what ETL is and what teams use for it. You'll also want to be familiar with the concept of a data warehouse.
>
> 🔮 **Dependencies** 🔮

dbt is quite simply a tool for building data models. If you're on a data team, you probably know what that means. But alas, my dear audience, if you're not, it may be unfamiliar. You've heard of machine learning models, but what's a data model?

→ The age of the cheap warehouse

First, the fundamentals. In a previous post, we talked about data integration:

> *Every company has this idealized vision of a data science and analytics team, with full visibility into how the business is doing, how the product gets used, how experiments are performing, super good looking and funny people, etc. The problem with getting there (and this is part of why data teams don't get hired until later in the company lifecycle) is that the actual, cold hard data that you need to answer important questions typically lies all over the place. And it needs cleaning.*

The process and discipline of gathering data from original sources, cleaning it, and getting into a warehouse is a tedious, ongoing process, and it's a lot (most?) of what early data teams spend their time on.

Traditionally, there was a three step process for getting that done: you'd first **extract** the data from the source, then **transform** it in flight to clean and ready it for analysis, then **load** it into your data warehouse. Transformation was done in flight, because putting raw source data into the warehouse *first* wasn't financially or technically feasible.

Today, though, as data warehouses have gotten easier to use, cheaper, and we've separated storage from compute, the paradigm is changing – companies are just funneling source data directly into their warehouses, and then working with it there. This is called **ELT** (because the transformation is happening after the load). And this is *very important*, because it makes data transformation as simple as writing SQL in your warehouse.

→ "Analysis" ready data

Source data – or in other words, what your production database, events, or even Stripe customers look like – is usually very different than the format you'd want for analysis. You might be capturing event data that looks like this:

user_id	event_type	resource_id	timestamp
1	resource_created	1	'2020-01-01'
1	resource_edited	1	'2020-01-02'
1	resource_destroyed	1	'2020-01-03'
2	resource_created	2	'2020-02-01'

How do you get from *this* to answering a question like "what's our most popular product?" or "what do our retention numbers look like?"

The answer is **transformation** – data teams write intermediate SQL queries that format this data in a way that makes it easier to answer those high level questions. A few examples of those kinds of transformations:

- Aggregating all of the different types of product events into one table
- Re-formatting your Stripe data to show MRR by month
- Grouping website visits into "sessions" for easier analysis

Mature data teams will have hundreds of these kinds of transformations. When you group all of these together, you've got what we call a **data model**. Having a well thought out, accurate, documented data model is the eternal struggle for today's data teams. And it's very hard! Data constantly changes, the business constantly changes, and analysis needs do too. You can think of a model as consisting of a few things:

A mapping from the business domain to data. Businesses have concepts like users and revenue. But what do those actually *mean*? Is a user someone that signed up for the product, or someone that paid for the product? Does revenue include the startup credits that your marketing team gives out?

SQL queries that transform and clean data. Data will often be dirty from the source, and need cleaning. That in addition to the queries that organize and name things into their business context will make up the bulk of your model SQL.

Documentation on what things mean. How do I find how many users we have? What does "address_redacted" mean? Why are there two tables that have data on our organizations? A good set of documentation helps analysts know where their queries should be pulling from and how.

→ **Data team structure**

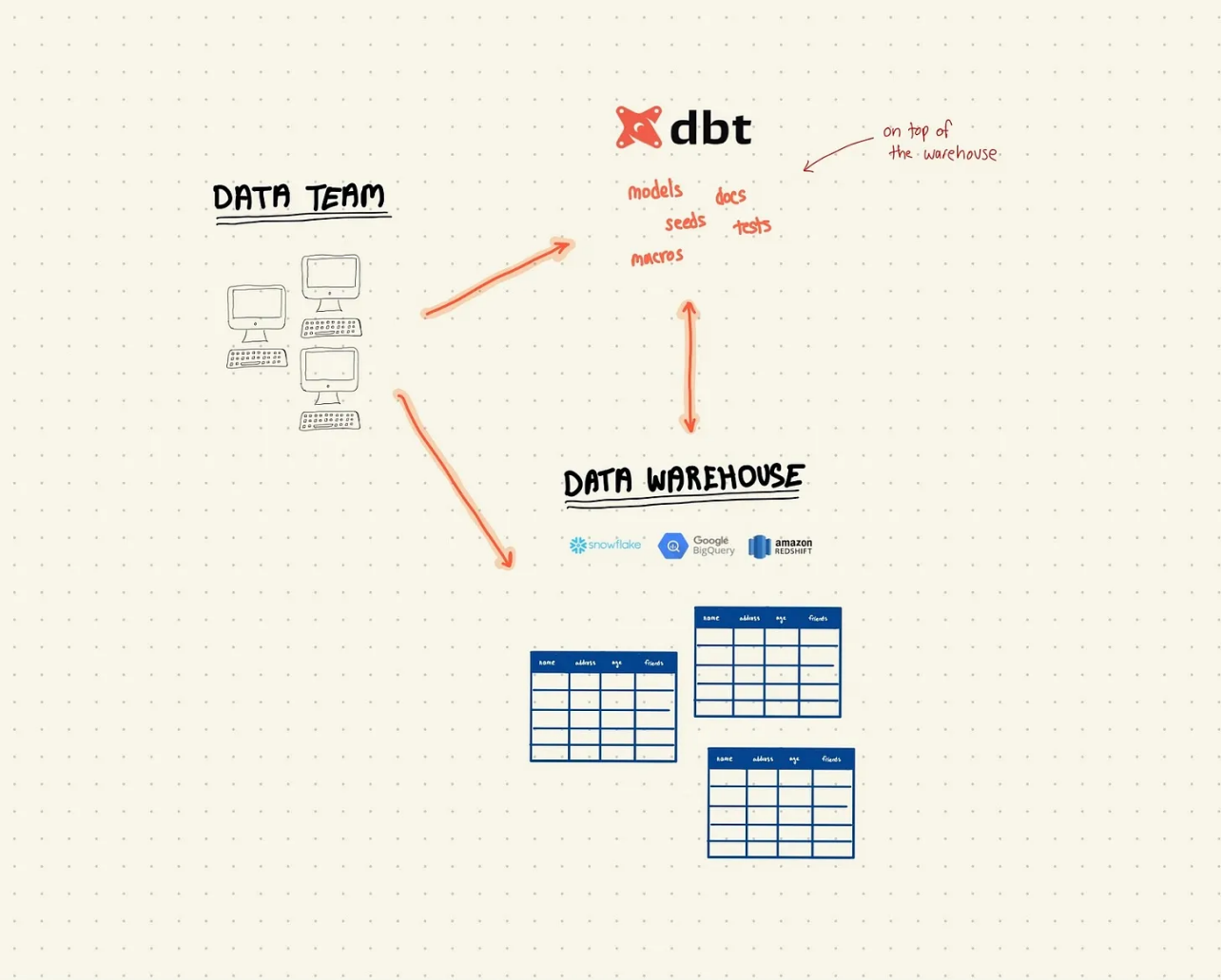
Before diving into the dbt product, a final note on data team structure and who is actually using these models. Data teams are fluid, and there's no one size fits all on how they're structured. But here are a few profiles that might exist:

- **Data engineer** – build complex data pipelines, connect databases, etc.
- **Data scientist** – build predictive machine learning models
- **Data analyst** – SQL based analysis, deeper questions
- **Analytics engineer** – build data models
- **Business analyst** – build dashboards

Model-wise, it's usually the analytics engineer and the data analyst working on building models, while (also) data analysts and business analysts might consume that model. You can think of the data model as a **defined interface** (or API) built by the data team for consumption by the rest of the company.

The dbt product: core concepts

dbt is a toolset for building data models. It lets data teams run SQL to build models, add documentation to each table, run tests to ensure data quality, and many other nice things. The major *output* of a dbt project is a command you can run (literally, `dbt run`) that runs your SQL and builds or updates tables in your warehouse.



→ The model

If there's one thing to remember about dbt, it's the model. A model is just a SQL file, usually for one big SQL query. You might have one model to format your Stripe data into monthly increments, and one to aggregate your marketing site events into a more readable format. Models can also build on top of each other – you might have a "core" model to do some lower level cleaning, and then another model that uses it to build the table you need.

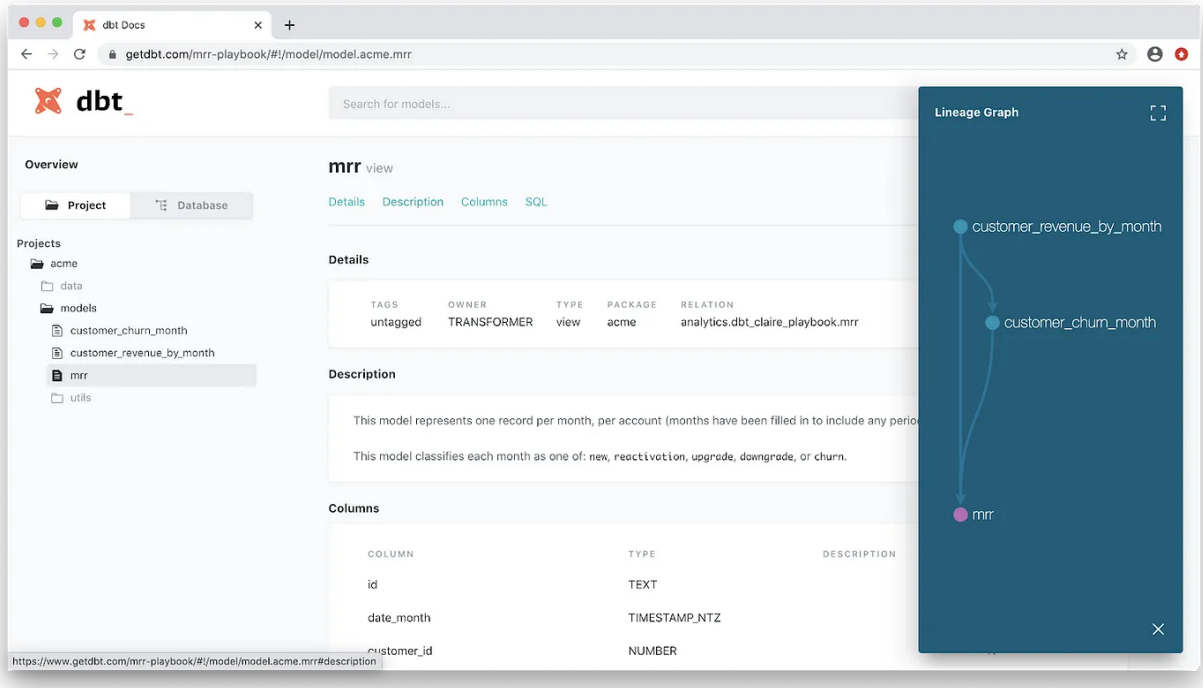
Each dbt model has to end up creating something: either **a table** in your warehouse, or **materialized view**. If you're not familiar with the latter, think of it as a saved query. Most teams will create a new project or schema in their warehouse, and have all of their dbt models result in new tables or views in that schema. Your "stripe_monthly" model could be configured to create a table called analytics.stripe_monthly.

> **🔍 Deeper Look 🔍**

> Even if you *could* run this SQL elsewhere and store views directly in your warehouse, one of the benefits of dbt is having **all** of your data model code together in one place, versioned via git.
>
> **🔍 Deeper Look 🔍**

→ Docs

Your data model isn't worth anything if people can't use it. dbt gives you the ability to document each table, and even each column within each table, with names, descriptions, and data types. The output of [dbt's docs command] is a big JSON blob, but if you pay for dbt's cloud product, they give you a nice frontend to view and interact with those docs. You can also [serve them locally] on the open source version.



Writing SQL in dbt is great, because they provide some really convenient utilities. One example is a **[macro]**: you can use macros to templatize repetitive SQL, just like you would in code. Another one: dbt provides a magical little utility called a **reference**. If any of your models want to [reference other models], you can write ref('some_table_name') and dbt will automatically figure out which order to run things in (see below).

→ Tests

Like software, data needs to be unit tested to make sure everything is as expected, nothing is missing, there aren't duplicates, etc. dbt lets you write column by column tests to check for null values, specify only certain allowed values, and validate unique primary keys.

→ Seeds

If you have external data that you need to bring into your warehouse – like a CSV that maps addresses to zip codes – you can do that with dbt's seed command. This is a pretty easy way to basically upload CSVs into your warehouse as tables.

→ Runs

Everything in dbt culminates in a **run**, which is basically just a fancy way of running all of your SQL. When you type dbt run, it calculates that cool dependency graph we talked about above, and automatically figures out which SQL queries need to run in which order.

In a more general purpose ETL tool like Airflow, you'd need to build this ordering yourself.

Generally, teams will schedule dbt runs (i.e. run every model) to happen twice a day or so. You can do that with cron, or if you're paying for dbt cloud, they'll do it for you.

The dbt ecosystem and cloud model

One of the confusing things about dbt is where it fits into the broader data ecosystem. It's not quite an ETL tool, it's not a warehouse, so what is it? Who do they compete with? Let's start with what dbt is *not*.

dbt is not a data warehouse. dbt doesn't store any data, and it is not a database. It interacts with your data warehouse, and pulls data from it to build new tables and views.

dbt is not a full service ETL tool. dbt is SQL only – you cannot write or schedule Python, Scala, etc. You can't run Spark jobs. You don't have granular control over the dependency graph. You can use dbt together with Airflow quite nicely.

dbt is not Looker or Tableau. dbt has no visualization capabilities, nor does it have a native language for exposing attributes to end users. Looker and dbt have partnered on a bunch of content.

dbt is not (yet) a full service testing framework. While dbt does let you run tests against your data, it's not a fully featured unit testing or data diff tool. Datafold recently wrote about a dbt integration. Having said that, they're working on improving it, and a community member recently built a package that integrates with Great Expectations.

Having said all of that, I doubt that this will all remain true forever. The nature of data tools has been to expand horizontally. Snowflake has been explicit in their goal to grow into a platform (not just a data warehouse), and my hunch is this will continue to be the norm – and so dbt, too, will likely end up needing to grow the product suite to match.

Finally, two interesting tidbits for your amusement:

dbt is open source. They have a paid cloud product that gives you a few nice features – a frontend for docs, scheduled runs, etc. – for a fee. They're still

experimenting with it, and it's definitely a little janky to use.

dbt is not actually a company. dbt is actually made by a company called Fishtown Analytics, a data consultancy that developed dbt internally while they were, well, consulting. *Edit: since I wrote this, they renamed the company "dbt labs" and it is no longer a consultancy.*

Comments

Write a comment...