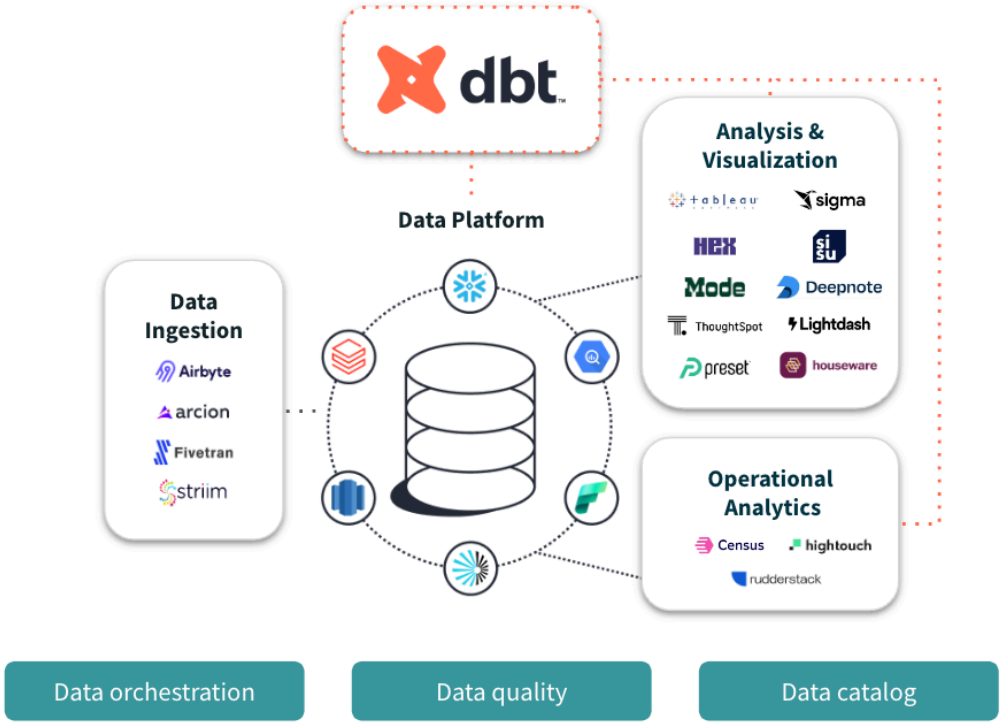
What is DBT and is it Right for My Team?



How dbt connects to your data tools. Image courtesy of getdbt.com

Introduction: what dbt is

DBT (Data Build Tool, stylized *dbt*) is a flexible tool for transforming raw data into a ready-use format for analytics (think aggregations, transformations, data cleaning, etc.). Its parent company has earned a $4.2 billion valuation. It is often described as “a tool that incorporates software engineering best practices into data pipelines,” which is true. Despite the hype, though, I feel the dbt community could better explain *what it is* to the world. Calling something a “tool” or “platform” does little to describe its use when you work with it everyday.

So I phrase it like this: Imagine if your entire data pipeline (logging, tests/validation, transformations, refreshes, extracts, queries) was one piece of software, organized neatly into easily-read files and folders instead of sprawled across several data platforms and analytics tools. It’s a low-code software that **demystifies the backend as much as possible.** When the backend is demystified, your workflow runs

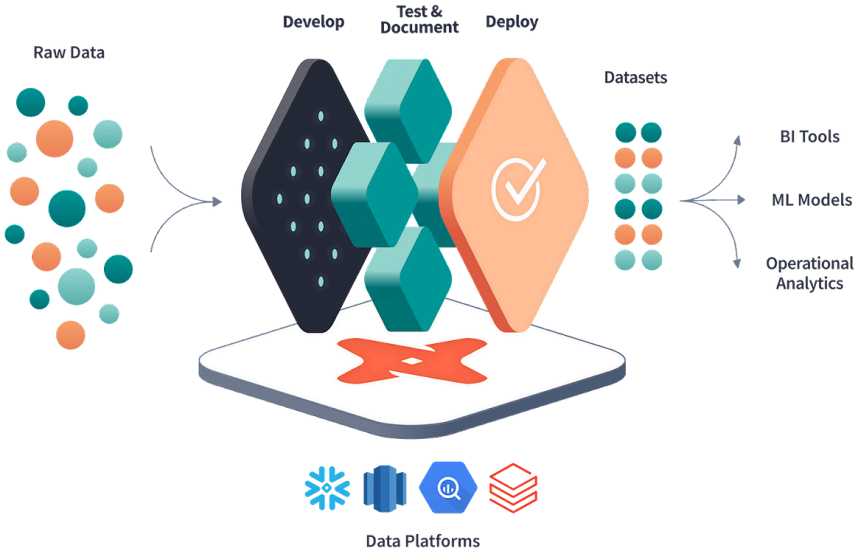
smooth and error-free. There are less questionable graphs presented to executives, and engineers spend less hours hunting down bugs in ancient code.

dbt is not a shiny tech tool offering a data panacea; it is a simple, open-source software which weaves your tech tools together seamlessly and traceably. It does so by saving SQL scripts into modular, reusable files, generating tests and logs, and visualizing data lineage. It simplifies the most laborious aspects of data management, which saves time and reduces the potential for human error.

In fact, much of dbt will feel familiar to engineers with experience managing data pipelines, and they will remember the work covered by dbt as particularly tedious. Logging, testing, and validation are the thankless realms of data science often handed to engineers known for their resilience. Oddly, much of that work is also handed to new data analysts with a limited knowledge of their business *and* data engineering principles, which can lead to erroneous results presented to executives. dbt allows analysts and engineers to collaborate and present quality data to management.

I've found that learning dbt has improved my skills in cloud computing and command line tools by tying them all together so I'm eager to share it with the larger data science world.

"But what does it *do*?"*:* Essential Features



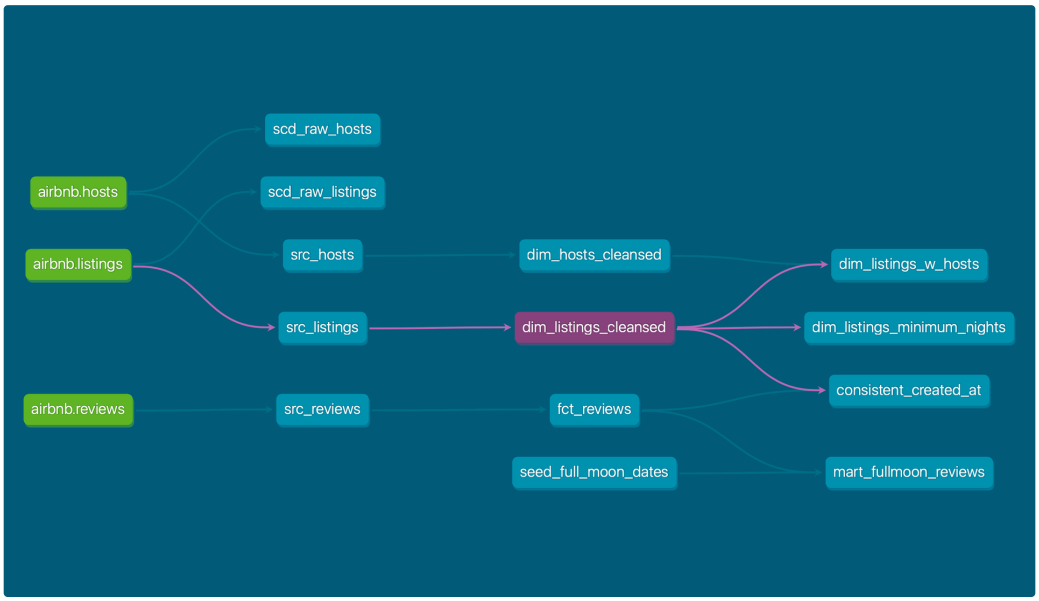
Where dbt sits within your data ecosystem. Image courtesy of getdbt.com

Before we dive into using dbt and whom it's best for, here's a run-down of its major features:

- tests your data pipeline with built-in tests (also allows for third-party test packages and custom tests) to validate data quality
- visualizes your data pipeline and data lineage
- tracks dependencies between models, sources, and macros
- integrates with Git for version control of your pipeline
- automatically documents data flows and generates documentation (which can be customized)
- generates logs
- reduces the amount of total code and repeated work needed for your data team (via DRY)
- centralizes your logic for transformations and calculated fields in dashboards (so that if logic is changed in one place, it updates everywhere automatically)
- By using Jinja templates, dbt users can **define variables and write loops in SQL** to make SQL more flexible and reusable
- Provides a web-based interface to explore and understand the data transformation process.
- Optimizes performance and reduces compute costs with optimized SQL under the hood

- Offers incremental processing of large tables and workloads to reduce compute costs

My favorite feature is dbt's built-in tests, but it allows for custom tests (and third-party extensions) as well. There is also a plug-in for Great Expectations, a renowned data testing package with pre-programmed data quality tests.



Interactive lineage graphs are easily created with the **dbt docs generate** command, and viewed online with **dbt docs serve**.

Nuts and Bolts: How Does it Work?

dbt runs on "models," which are SQL files with a SELECT statement (though they now offer .py files, for PySpark). At its core, dbt transforms data by running these models against your data warehouse.

SQL statements are programmed into Jinja "macros" which look like this:

```
-- macros/transform.sql
{% macro transform_column(column_name) %}
  CASE
    WHEN {{ column_name }} IS NULL THEN 'Unknown'
    ELSE {{ column_name }}
  END
{% endmacro %}

```

And called by model files like so:

```
-- models/example.sql
SELECT
  id,
  {{ transform_column('name') }} AS name_transformed
FROM {{ source('schema', 'table') }};
```

Jinja is a special templating language built in Python. Because essential pieces of code are saved in macros, they can easily be reused, which facilitates collaboration. And whilst employees are collaborating, dbt integrates with Git to allow version control.

And finally, your project's configurations are stored into YAML files which are pretty straightforward (and used by engineers outside of dbt). Each project has a few essential YAML files referenced by the models and terminal commands, which indicate tests to run, documentation of the tables and models, or staging/dev/prod environments to use.

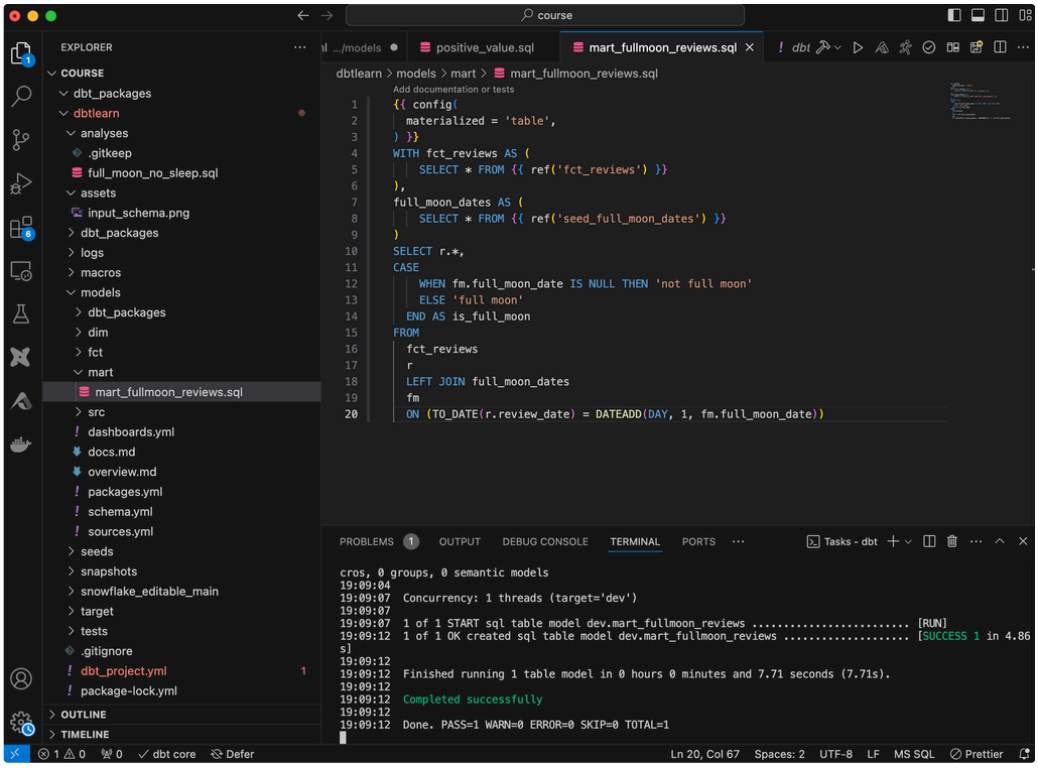
For examples and more information on dbt's use of YAML, check their site here

The open-source version of dbt is run by simple terminal commands like dbt run, dbt test, dbt debug etc.

And that is the entirety of dbt's core functions. Under the hood, it automatically manages dependencies between models and optimizes when to query them according to their position in the pipeline. When one model references another, dbt ensures the dependent model is built first. To me, its power lies in its simplicity; because it doesn't attempt other things, it allows your cloud data services to do the low-level engineering jobs they were optimized for (e.g. cache results in Snowflake) while dbt specializes in managing your pipeline. Its simplicity also makes it accessible to Data Analysts who know SQL and would like more control over their data sources.

What is Using dbt Like?

Essentially, you look at a collection of files and folders in your IDE (VS Code works well, since you can see the directory, code, and terminal together), and run operations from the command line. This view allows you to *holistically* see your data pipeline as a single piece of software.



What working in dbt looks like. The project here is from the brilliant dbt Zero to Hero course by Zoltan Toth.
On the sidebar is also the dbt Power User extension!

A dbt project is a collection of models orchestrated by a .YML file, which configures the models. Logs and documentation are generated according to the YAML instructions, and the CLI commands allow you to run pipelines, tests, refreshes, etc.

Managing your pipelines in this way lets you see all your data transformations together, instead of switching between multiple browser tabs and/or programs!

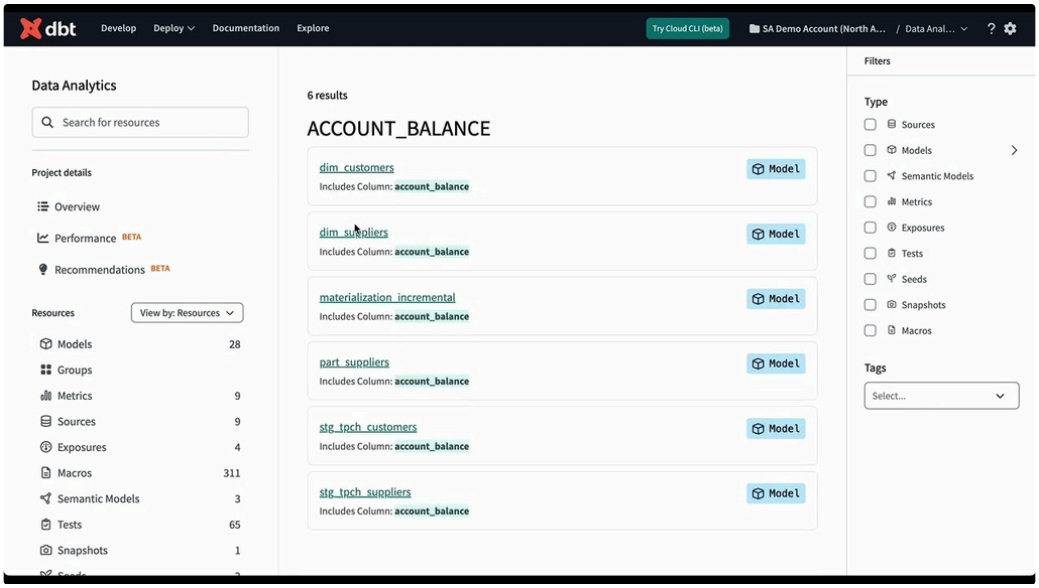
An example workflow using dbt would look like

1. Create project using

```
dbt init my_project
cd my_project
```

2. Configure database connections in **profiles.yml** and data sources in **sources.yml**
3. Write SQL Models- first staging models which reference raw data, then final models which reference the staging models
4. Add tests to YAML, run tests with **dbt test**
5. Generate documentation with **dbt docs generate**
6. Push to Git and release version-controlled updates:

```
git add models/new_model.sql
git commit -m "Add new model"
git push origin feature/new-model
```



Working in dbt Explorer, a feature of the paid dbt Cloud service.

dbt Cloud incorporates many of the CLI commands as features in a GUI for simplicity, and dbt Explorer integrates more easily with other data platforms. If using dbt Cloud, you may define calculated variables for dashboards right in the Semantic Layer.

The Brilliance of DRY

It's often said that dbt allows data teams to finally execute software engineering best practices in SQL. DRY (Don't Repeat Yourself!) is perhaps the most important of those practices.

Not just because it saves time, but because it ensures data cleanliness and reliability; each transformation is defined once and reused whenever needed, ensuring consistency and reducing the risk of error. With Jinja macros, common SQL logic such as complex calculations, filtering, or transformations only needs to be written once- a single source of truth. DRY actually means writing as little code as possible.

A DRY codebase also makes it easier for team members to collaborate: since logic is centralized, team members can work on different parts of the project without worrying about conflicting changes or duplicated efforts.

And, finally, dbt's modular SQL development allows easier maintenance and scalability. Business logic within pipelines can be centrally maintained, making it easier to implement (and test!) updates without modifying the entire codebase. And when a piece of logic needs to be changed, you only do so in one place. As projects grow, new features and enhancements can be added without introducing significant complexity.

dbt vs SQL Stored Procedures

A data veteran might ask "why can't I just run a Stored Procedure on my SQL tables? It runs within the database engine itself so it's blazing fast."

The obvious answer is that if you're using cloud-based tools (like Snowflake), you can't use SQL stored procedures. dbt is tailored for working with modern cloud data warehouses (like Snowflake, AWS and DataBricks) to weave them together. And, unlike stored procedures, dbt integrates with Git for version control, which eases collaboration, and automatically documents data flows.

But even for data on SQL Server, the company themselves phrase it:

> For one, stored procedures don't lend themselves well to documenting data flow, as the intermediate steps are a black box. Secondly, this also means that your stored procedures aren't very testable. Finally, we often see logic from intermediate steps in one stored procedure copied almost line-for-line to others!
>
> Migrating from Stored Procedures to dbt | dbt Developer Blog

To be testable in general, SQL requires third-party applications and/or is a several-step process which involves writing new scripts or creating (and loading) a separate test database.

dbt offers detailed data lineage features, which are harder to implement in stored procedures. Even third-party extensions to SQL have a hard time correctly identifying data lineage.

Lastly, a long SQL stored procedure runs in one script, risking all the hazards of human error. A dbt pipeline is a series of modular, testable and documented pieces of code with no mysteries. Via Jinja, dbt allows users to expand SQL's functionality to include loops and variables as well.

dbt will (efficiently) automate the more tedious parts of SQL code for you. Running raw SQL against the database engine will always be fastest, but dbt's automation ensures performant, clean code. Validationg data is simply a much easier job with dbt than SQL Server because dbt is a purpose-built tool for data quality.

Cons and Caveats

YAML is not conceptually hard, but formatting must be precise. A wrong indentation level is not obvious, but will cause your code to break. Furthermore, the debug statements in your terminal will not mention anything about an indentation error. If indentations are not exact, your dbt commands will error out. YAML is not like many programming languages where indentation is arbitrary.

Another point is resource dependency: dbt relies heavily on the underlying data warehouse's performance. Inefficient SQL can lead to resource contention and increased costs. Also, for small teams or simple projects, the overhead of setting up dbt may outweigh its benefits.

And finally, you have to acknowledge that it's a limited software out-of-the-box; it is designed to be minimal. Therein lies its strength, but it only specializes in one function (the "T" in ETL/ELT- Transformation). If you need more workflow orchestration, dbt integrates well with Airflow or Dagster.

"Why dbt?": Who Benefits the Most from it

dbt delivers the most value if

- you have a single project using multiple data sources
  - those data sources feed multiple outputs (i.e. ML models or dashboards)
- you need Data Analysts collaborating with Data Engineers
- your company has multiple legacy data systems and
  - you want to make sense of how they all connect (including tracking every table/column/calculated field
  - you want to connect them more efficiently
- a data team which is particularly large or cross-functional, in a large company

Or any combination of the above.

Conveniently, your team only needs one worker skilled in dbt to implement it; the rest of the team can go on dashboarding and writing SQL queries.

As data volumes and complexity grow, dbt's modular and scalable approach becomes increasingly valuable. Its integration with version control and automated documentation fosters collaboration and transparency, which is a bonus for larger teams and regulated industries.

Simplifying Your Data Ecosystem

dbt centralizes your data systems (and transformations) into one place, letting you see and test their interactions. This approach improves maintainability and provides a clear view of the entire data pipeline.

It was designed from square one to integrate with modern cloud data warehouses like Snowflake, BigQuery, Redshift, and Databricks. When other platforms further integrate with dbt, that makes it even simpler to use (Tableau is improving its integrations with dbt to make up for Tableau's lack of version control and pipelining features). dbt offers a variety of free courses to guide integration with your company's data systems. They even have explainer courses for integrating with Excel.

When we centralize our transformation logic, we reduce data movement and minimize latency, then we can leverage the powers of cloud data warehouses for efficient processing. When we streamline our data workflows, we reduce complexity and accelerate time-to-insight.

A Closing Note

> *"It takes 20 years to build a reputation, then it takes five minutes to lose it."*
>
> Warren Buffett

Building data pipelines may be one of those jobs people only notice when it goes wrong. Perhaps dbt's meteoric rise is due to it automating the most thankless and tedious aspects of data science.

With dbt, you can immediately spot an error in your pipeline by running

```
dbt test
```

and it will point out which file and line contains the bug. A raw Python pipeline, even with homemade logging and testing, still needs to be run (eating compute resources), and will (probably) only point out an error if it's in one of the tests you wrote.

Assuming it goes unnoticed, you will either

- Break your pipeline and spend hours combing through various files line by line, searching for a bug which doesn't throw an error
- Compile the pipeline and generate a report or dashboard which doesn't make sense, then spend hours combing through various files line by line, searching for a bug which doesn't throw an error. Management trusts the data team less, and there is tension between data employees because it's not clear who is at fault.

In other words, the benefit of doing it right isn't immediately obvious, but the cost of doing it wrong is enormous. Don't let that happen to you. Ensuring you always deliver clean, robust code will build you a great personal brand as a data scientist or consultant.

With a proper pipelining tool, many crucial tasks are automated, which means teams can focus on drawing valuable insights from data.

At Data Surge, responsible data handling is our top priority. Data Surge is a consultancy with experienced engineers certified in dbt, and a passion for clean, presentable data, ready to demystify your company's data systems so you can focus on what you do best.