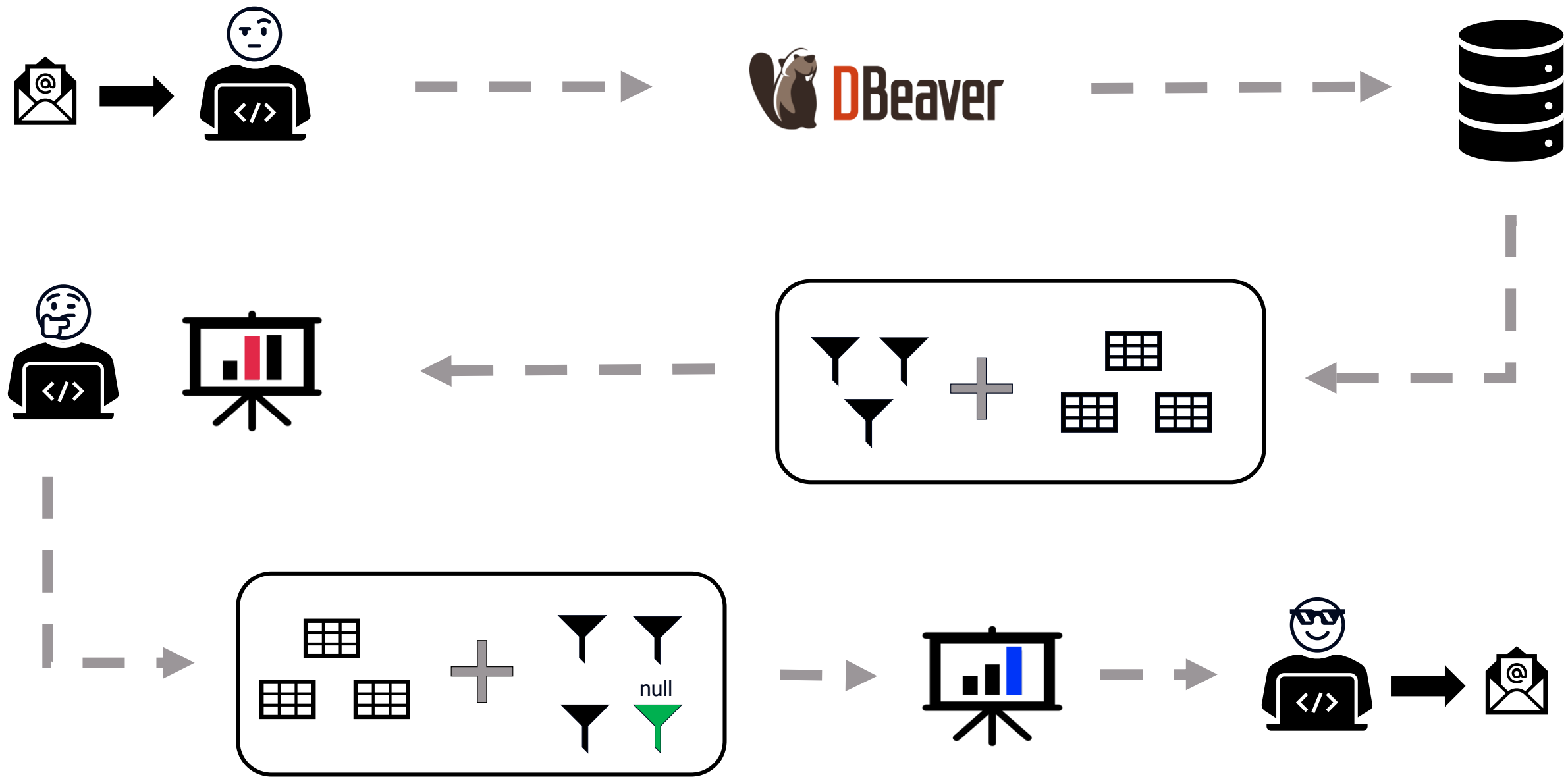




Introduction to data build tool (dbt)

By Mina Sonbol

A typical work scenario for a data analyst



Common shortfalls of standard approach

Reusable code	✗
Version control	✗
Data quality	✗
Documentation	✗

dbt: What it is and how it addresses these shortfalls?

- **Transforming large volumes of data efficiently**
- **Modular & reusable code**
- **Automating data quality testing**
- **Version control**
- **Documentation**

This makes dbt a transformative tool that can improve efficiency and productivity, enhance data quality and trust, and allow for better collaboration and transparency, providing a framework for **a better analytics process.**

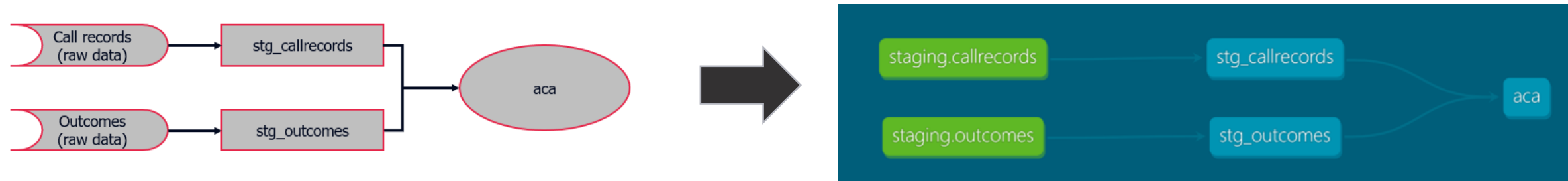
[source](#)

dbt in action 1: Matching query simplified

We will create several models to imitate the matching query. The models will be split into 3 phases:

- Staging: This is where we type cast the raw data into appropriate data types
- Core: This is where we join the data from the staging phase to create a source of truth table
- Reporting: This is where we create data artifacts that can be used for building dashboards

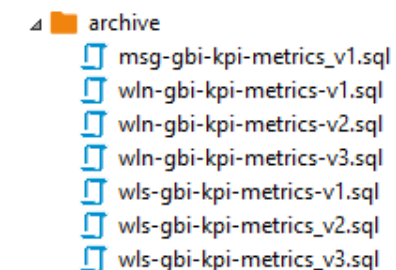
By splitting the processing into phases, each with its own directory, we create **a modular simplified matching query that is easier to understand and debug**. dbt also allows us to generate data lineage charts like this:



[Source, demo link](#)

dbt in action 2: Documentation, testing, and version control

- **Documentation:** We add documentation to the schema.yml file, including descriptions and data types. That way our data dictionaries sit where the data is.
- **Tests:** Tests are also defined in the schema.yml file. dbt provides generic and bespoke tests.
 - *Generic tests* – Tests that can be performed out-of-the-box. There are 4 types:
 - Unique
 - Not null
 - Accepted values
 - Relationships
 - *Bespoke tests* - These are custom queries created under the test directory. The test succeeds if the query does not return any rows.
- **Version control:** Having the tests and documentation in the same file makes it easy to version control them, along with our sql models, using tools such as git. No need to track changes like this →



```
archive
├── msg-gbi-kpi-metrics_v1.sql
├── wln-gbi-kpi-metrics-v1.sql
├── wln-gbi-kpi-metrics-v2.sql
├── wln-gbi-kpi-metrics-v3.sql
├── wls-gbi-kpi-metrics-v1.sql
├── wls-gbi-kpi-metrics_v2.sql
└── wls-gbi-kpi-metrics_v3.sql
```

[source](#)

Appendix: dbt configurations and commands

Main Configuration Files

1. `profiles.yml`: Contains the profile settings, including database connection settings (host, user, password). Can accept multiple profiles. Is typically stored outside the dbt project directory.
2. `dbt_project.yml`: Defines the dbt project configuration, including the project directories and model configurations (ex. materialization settings).
3. `schema.yml`: Contains the definitions of the sources, models, and tests that dbt will run when compiling the models.

dbt commands

- `dbt init` → Initiates a project and creates / modifies the `profiles.yml` configuration file
- `dbt debug` → Tests database connections
- `dbt build` → Builds the dbt models
- `dbt test` → Runs tests defined on the data to ensure its integrity
- `dbt docs generate/serve` → Generates documentation for the project
- `dbt deps` → Downloads and installs package dependencies



afniti®
PAIR BETTER®

Thank you