DBT Concepts Overview

Diogo Landau

2022-09-15

Contents

DBT Project

> dbt docs projects:
> https://docs.getdbt.com/docs/building-a-dbt-project/projects

A dbt project isa directoryo of .sql and .yml files, which dbt uses to transform the data. At a minimum a dbt project must contain:

- A project file (dbt_project.yml): This file indicates that a particular directory is a dbt project, and also contains configurations for the project.
- Models

DBT Resources

A dbt project can have multiple resources. These can be:

- Models
- Tests
- Sources
- Macros
- Seeds
- Snapshots

DBT Profile

> dbt docs configure your profile:
> https://docs.getdbt.com/dbt-cli/configure-your-profile
> dbt docs profile:
> https://docs.getdbt.com/reference/project-configs/profile

the profile parameter specified in the dbt_project.yml is the profile dbt should use to connect to the datawarehouse.

profiles.yml

In the `profiles.yml` file, you can store as many profiles as you need. Typically, you would have one profile for each warehouse used. Most organizatinos only have one profile.

A profile consists of targets, and a specified default target. Eacch target specifies the type of warehouse you are connecting to, the credentials to connect to the warehouse, and some dbt-specific configurations.

Setting up Profiles

To setup a profile, copy the correct sample profile for the datawarehouse into the profiles.yml file and update the details as follows:

- Profile name: replcae the name of the profile with a sensible name - it's often a good idea to use the name of the organization. Make sure that this is the same name as the profile indicates in your dbt_project.yml file.
- target: This is the default target your dbt project will use. It must be one of the targets you define in your profile. Commonly it is set to dev.
- Popultaing your target:
  - type: The type of data waerhouse
  - warehouse credentials
  - schema: the default schema (bigquery dataset) that dbt will build objets in.
  - threads: The number of threads the dbt project will run on.

DBT Model

A model is a single .sql file. Each model contains a single select statement that either transforms raw data into a dataset that is ready for analytics, or more often, is an intermediate step in such a transformation.

When running dbt run, dbt will build the model in the data warehouse by wrapping it in a create view as or create table as statement.

Configuring Models

Configurations are "model settings" that can be set in the dbt_project.yml file, and in the model file using a config block.

> Ex:
>
> - Change a model's materialization: View, Table, Incremental, Ephemeral.

Model Materialization

View: When using the view materialization, the model is rebuilt as a view on each run, via a `create view as` statement.

Table: When creating the `table` materializatoin, the model is rebuilt as a table on each run, via a create table as statement.

Incremental: The `incremental` model allows dbt to insert or update recors into a table since the lat time that dbt was run.

Ephemeral: `ephemeral` models are not directly built into the database. Instead, dbt will interpolate the code from this model into dependent models as a common table espression (CTE).

Incremental Materialization

Incremental models are built as tables in the data warehouse. The first time a model is run, the table is built by transforming all rows of source data. On subsequent runs, dbt transforms only the rows in the source data, posteriorly inserting / updating them into the already existing table.

Using Incremental Materializations

Like other materializations, incremental models are defined with `select` statements, with the materialization type defined in the models configuration properties.

To use incremental models, dbt also needs to know:
* How to filter the rows on an incremental run. * The uniqueness constraint of the model (if any)

Filtering Rows on an Incremental Run

To tell dbt which rows it should transform on an incremental run, wrap valid SQL that filters for these rows in the `is_incremental()` macro.

The `is_incremental()` macro returns True if: * The destination table already exists in the database * dbt is not running in full-refresh mode * the running model is configured with materialized='incremental'

How do Incremental Models Work?

dbt's incremental materializatoin works differently on different databases. Where supported, a merge statement is used to insert new records and update existing records.

On warehouses that don't support merge statements, a merge is implemented by first using a delete statement to delete records in the target table that are to be updated, and then an insert statement.

DBT Source

Sources make it possible to name and describe the data loaded into the warehouse by the EL tools. By declaring these tables as sources in dbt, it is possible to then:

- Select from source tables in the models using the `{{ source() }}` function.
- test the assumptions about the data source.
- calculate the freshness of the source data.

DBT Tests

> dbt docs tests:
> https://docs.getdbt.com/docs/building-a-dbt-project/tests

Tests are assertions made about the models and other resources in the dbt project. When the command `dbt test` is executed, dbt will indicate if each test in the project passes or fails.

As with most things in dbt, tests are select statements that seek to grab failingrecords, ones that disprove the assertions. As an example:

- If it is asserted that a column is never null, then the test is searching for the columns that are null.
- If the test asserts that a column is unique, then the statement select rows where the column is not unique.

As such, if the statement returns 0 rows, then it passed, otherwise, it is considered to have failed.

There are 2 types of tests, Singular and Generic. Generic tests have to be instantiated

DBT Target

> dbt docs target:
> https://docs.getdbt.com/reference/dbt-jinja-functions/target

target contains information about a connection to the datawarehouse.

dbt supposts multiple target within one profile to encourage the use of seperate development and production environments.

A typical profile using dbt locally will have a target named dev, and have this set as the default. There may also exist a production target within the same profile.

using the `--target` option allows dbt to specify which target is to be used.

DBT Macros

> dbt docs macros:
> https://docs.getdbt.com/docs/building-a-dbt-project/jinja-macros#macros

Macros in Jinja are pieces of code that can be reused multiple times. They are defined in .sql files, typically in your macros directory.

DBT Configs and Properties

> Configs, properties, what are they?
> https://docs.getdbt.com/reference/configs-and-properties

A project's resources - models, snapshots, seeds, tests, etc... - can have a number of declared properties. Resources can also define configurations, which are a special kind of property that bring extra abilities.

Properties: Declared for resources one-by-one in .yml files. Properties delcare things about the resource.

Configs: Configs tell dbt how to build those resources in the data warehouse. Configs can be defined for the resource within the .yml file in a nested `config` property. For some resources they can also be set all at once in `dbt_project.yml` . Because configs can be defined in multiple places, they are also applied hierarchically. An individual resource might *inherit* or *override* configs set elsewhere.

> **Ex.**
> Properties can be used to:
> - Describe models, snapshots, etc…
> - Assert truths about a model, in the form of tests;
> - Define pointeres to existing tables that contain raw data, in the form of sources, and assert the freshness of the raw data.
> - Define officail downstream uses of the data models, in the form of exposures.
>
> **Configurations** can be used to:
> - Change how a model is to be materialized
> - Declare where a seed will be created in the database
> - Declare whether a resource should persist its descriptions as comments in the database
> - Apply tags and meta properties

DBT Packages

> DBT docs packages: https://docs.getdbt.com/docs/building-a-dbt-project/package-management

dbt packages are in fact standalone dbt projects, with models and macros that tackle a specific problem area. As a dbt user, by adding a package to your project, the package's models and macros will become part of the project.

This means:

- Models in the package will be materialized when `dbt run` is executed.
- `ref` can be used in the hosting project to refer to the models in the package.
- Macros from the package can be used in the host project.

How to Add a Package?

1. Add a `packages.yml` file to the dbt project. This should be at the same level as the `dbt_project.yml` file.
2. Specify the package(s) to be integrated in the host project:

```yaml
# packages.yml

packages:
  - package: dbt-labs/snowplow
    version: 0.7.0

  - git: "https://github.com/dbt-labs/dbt-utils.git"
    revision: 0.1.21

  - local: /opt/dbt/redshift
```