Open in app ↗ Sign up Sign in

Medium Search Write

dbt Tests

Jimmy Pang · Follow
Published in Data Panda · 6 min read · Nov 11, 2023

38 1

A necessary assertion to ensure the reliablity of the building blocks

The dbt logo.

Table of Content

38 1

- Types of Tests in dbt — Generic Tests & Singular Tests
- Additional Advance Tests
- How to run tests
- Test “severity”
- Test Implementation Tactics
- Appendix

Overview

From dbt official documentation (Add tests to your DAG | dbt Developer Hub):

> *Tests are assertions you make about your models and other resources in your dbt project (e.g. sources, seeds and snapshots). When you run `dbt test`, dbt will tell you if each test in your project passes or fails.*

In short, there are 2 types of in dbt tests:

- Generic tests, which are set up by macros, and some of them are already available out of the box.
- Singular tests, which are `.sql` scripts that you expect to return zero rows, otherwise it failed.

Generic Tests

Generic Tests are put in place by calling a macro. They are defined in a `.yaml` file within the folder where the model is.

There are 4 Out-of-the-box tests provided by dbt:

1. not_null : check a column to validate there are no null values.
2. unique: check that values in a column don't contain duplicates.
3. accepted_values: define a list of expected values a column can have.
4. relationships: defines a column on the current table that references another column in another table, i.e. asserting Referential Integrity.

Under the hood, dbt constructs a `select` query for each test, using the parameterized query from the generic test block. These queries return the rows where your assertion is *not* true; if the test returns zero rows, your assertion passes.

not_null

Asserting if there is any NULL value under the selected column.

By default, this test will fail even if just 1 NULL record is found. This behavior can be adjusted in Severity.

unique

Asserting if the column is unique across the whole table.

accepted_values

Asserting if the value in the column is within the expected values.

```
version: 2
```

```yaml
models:
  - name: orders
    columns:
      - name: status
        tests:
          - accepted_values:
              values: ['placed', 'shipped', 'completed', 'returned']

      - name: status_id
        tests:
          - accepted_values:
              values: [1, 2, 3, 4]
              quote: false
```

relationships

It asserts the Relationship between the 2 models. The Relationship is based on the Entity Relationship (ER) Diagram. A typical example would be the id of the Buyer in the Orders table should always be found in the Customer dimension table.

The following example tests that every order's `customer_id` maps back to a valid `customer`:

```yaml
version: 2

models:
  - name: orders
    columns:
      - name: customer_id
        tests:
          - relationships:
              to: ref('customers')
              field: id
```

Singular Tests

Singular Tests are custom sql scripts we can build to test specific values.

- You can think of it as a query from which you expect zero results: if there are returned rows, then the test will fail.
- These are defined in `.sql` files, typically in your `tests` directory.
- You can use Jinja (including `ref` and `source`) in the test definition, just like you can when creating models.
- Each `.sql` file contains one `select` statement, and it defines one test.
- For reference, you can check the tests folder in the main root of the project. And the folder structure should be mirroring the `model` folder as well

Additional Advance Tests

More additional Advance Tests are available after implementing dbt-utils and Elementary, more advance Tests are available.

dbt-utils

The following tests come from the dbt-utils package.

- recency
- expression_is_true
- accepted_range
- not_constant
- not_empty_string
- not_null_proportion

- not_accepted_values
- relationships_where — The advanced version of the `relationship`.

Elementary — Data Anomaly Detection

Elementary also adds extra dbt Tests for anomaly detection, and it comes in 3 different levels:

- Table level monitors
- Column level monitors
- Dimension monitors

How to run tests

in dbt, tests are run via the `dbt tests` command (About dbt test command | dbt Developer Hub). Examples:

Run tests on one specific model:

```
dbt test --select hourly_tmp_dim_vc_usr_customer
```

Run tests on a model and all its dependencies:

```
dbt test --select hourly_tmp_dim_vc_usr_customer+
```

Run tests on a model and all upstream models:

```
dbt test --models +hourly_fct_vc_usr_client_stats
```

Run tests on a list of models (separated by whitespace only):

```
dbt test --models hourly_fct_vc_usr_client_stats hourly_fct_vc_usr_client_stats
```

When it comes to model select, the choices in dbt is actually very flexible — please see Methods | dbt Developer Hub under Node selection section.

Test "severity"

Configuring test `severity` | dbt Developer Hub

dbt support 2 kinds of severity for failed dbt test: `error` and `warn`.

The relevant configs are:

- `severity`: `error` or `warn` (default: `error`)
- `error_if`: conditional expression (default: `!=0`)
- `warn_if`: conditional expression (default: `!=0`)

For example,

```yaml
version: 2

models:
  - name: large_table
    columns:
      - name: slightly_unreliable_column
        tests:
          - unique:
              config:
                severity: error
                error_if: ">1000"
                warn_if: ">10"
```

This `yml` file is saying a couple of things:

1. for the test of the column `slightly_unreliable_column`, make sure there is no duplication (`unique` test)
2. the severity of this test is error (default value), BUT:
3. It would start generating `warn` messages in the logs (CLI) if there are more than 10duplicated records; AND,
4. The test will fail if there are more than 1000 duplicated records found, which makes the run of `dbt test` failed

Severity could also be configured in various locations in the dbt repo, please see here (Configuring test `severity` | dbt Developer Hub) for details.

Note

- Only use Error for business-critical bases, as data quality issue like duplication of those would have big impact of a lot of tables downstream

- Most issues are minor and shouldn't break a whole pipeline, so Warning is more than enough
- It would also be nice if there are different layers of Warning to help assessing the impact

Test Implementation Tactics

Primary Key

Even though Snowflake and the other DWH solutions do not enforce Primary Key, it is still very vital to set the Primary Key to all tables.

To assert the Primary Key of the table, it is suggested to use both of the following Test:

- not_null
- unique

Surrogate Key

What Is a Surrogate Key?

The ingested tables do not always come with a clear key. By working with stakeholders at Product, Tech, and Business, Surrogate Key can be generated with the relevant domain knowledge (perfect example here).

The assertion of Surrogate Key is similar to Primary Key:

- not_null
- unique

It is also recommended to generate the Surrogate Key by using Macro from dbt-utils for the sake of standardization. BUT note that it is very critical to conduct enough investigation to find out the hidden granularity in the table before the implementation.

NULLs sometimes

During data development, there is a lot of times that the BI team doesn't have in-depth domain knowledge yet.

It is always useful to start with this one for new column:

- not_null

If the `not_null` test doesn't pass, we could fall back to make use of where config to make a selective test, or use `not_null_proportion`. The rule of thumb is always to start very strictly, then fall back step by step.

Example:

```
version: 2

models:
  - name: large_table
    columns:
      - name: sometimes_null_column
        tests:
          - not_null:
              config:
                where: "category = 'category that we know they shouldn't be NULL
```

Dimension columns with categorical values

Suggested to use these 2 together:

- `not_null`
- `accepted_value`

Metrics columns with continuous values

Suggested to use:

- `not_null`
- `accepted_range`

Future TIMESTAMPs

In general, future TIMESTAMPs are not a very critical data quality issue.

Suggested to use:

- `not_null`
- `accepted_range` OR `expression_is_true`

Note:

- Also while comparing TIMESTAMPs, it is very vital to keep in mind the timezone of different TIMESTAMP values.

Referential Integrity issues

What is Referential Integrity?

A typical example of a referential integrity issue is the `customer_id` in the Sales table cannot be found as `customer_id` in the Customer table table.

Suggested to use:

- `relationship`
- (If the previous one fails, fall back to) `relationships_where`

Appendix

Compile

There is an additional check that can be run In dbt: Compile

```
dbt compile
```

According to dbt, it would:

> dbt compile generates executable SQL from source model, test, and analysis files. You can find these compiled SQL files in the target/ directory of your dbt project.

If there are files in the repository that cannot be generated into executable SQL, this test would fail.

Further Reads

- dbt Models — Marts layer (i.e. Data Marts)
- dbt Models — Staging layer

- dbt — An overview of the supernova in data & analytics engineering
- dbt Lab — Add tests to your DAG

Dbt | Analytics Engineering | Data Engineering | Data Modeling

Written by Jimmy Pang

Follow

142 Followers · Editor for Data Panda

Data Leader, Evangelist and Educator, dedicated to the data journey. Interested in tech and classy stuffs: art, whiskey, coffee, tea, spirituality, history etc.

More from Jimmy Pang and Data Panda

Jimmy Pang in Data Panda

dbt Models—Marts layer (i.e. Data Marts)

Business facing entities, ready for reporting, ad-hoc analysis, Machine Learning and...

10 min read · Nov 11, 2023

267 1

Jimmy Pang in Data Panda

dbt Models—Staging layer

The basic building blocks of a dbt project.

6 min read · Nov 11, 2023

19

Jimmy Pang in Data Panda

Dashboarding SOP (Standard Operating Procedure)—How to...

Dashboards—what it is and why it is still relevant

5 min read · Jan 9, 2023

62 1

Jimmy Pang in Data Panda

How Agile could work in data teams—Scrum & Kanban

One of the possible ways to organize data team workflow

9 min read · Sep 3, 2023

135

See all from Jimmy Pang

See all from Data Panda

Recommended from Medium

Blosher Brar

Why DBT could be the future of data engineering?

In case you missed the memo, there is a new data warehousing sharif in town, and...

5 min read · May 9, 2024

271 3

Jack C in Better Programming

dbt v1.5—the 3 Big New Things

Data contracts, model versions, and model access

6 min read · Apr 28, 2023

338 5

Lists

data science and AI
40 stories · 169 saves

Natural Language Processing
1476 stories · 988 saves

Staff Picks
651 stories · 997 saves

Ryan Eakman in Towards Dev

SQLFrame: Turning PySpark into a Universal DataFrame API

Have your SQL/Python Cake and Eat it Too

5 min read · May 21, 2024

160 3

Todd Perry in The Finatext Tech Blog

DBT Incremental Strategy and Idempotency

Background

8 min read · May 21, 2024

246

Dave Flynn in In the Pipeline

Speed up pull request review for SQLMesh data projects

SQLMesh is a powerful ELT platform with features to help you iterate data pipelines wi...

5 min read · May 9, 2024

9

Mahdi Karabiben in Towards Data Science

A Simple (Yet Effective) Approach to Implementing Unit Tests for db...

Unit testing dbt models has always been one of the most critical missing pieces of the dbt...

9 min read · Aug 18, 2023

240 2

See more recommendations