

Introdução ao DBT (Data Build Tool)

O que é o DBT?

DBT (Data Build Tool) é uma ferramenta que permite aplicar princípios de engenharia de software à transformação de dados no ambiente de dados analíticos.

Propósito:

Transformar dados já carregados em um banco de dados (ex: DuckDB, BigQuery, Redshift) usando **SQL modular, versionado e testado**.



Como funciona o DBT?

- Você **cria models** em arquivos SQL, organizados por camadas (ex.: `staging/`, `marts/`).
- DBT converte esses arquivos em SQL executável no warehouse, respeitando dependências (via `ref()`).
- Gera documentação visual, realiza testes de qualidade, cria lineage (linhagem) e suporta versionamento com Git.
- Não é ferramenta de ingestão ou visualização – foca na transformação (o **T** de ELT).
- Você **não move dados com o DBT**, mas transforma dados **já existentes no banco**.

Dbt Cloud vs Dbt Core

Item	DBT Cloud	DBT Core (open source)
Interface	Web, com UI e agendador	Linha de comando (CLI)
Infraestrutura	DBT executa na nuvem	Você executa localmente
Preço	Gratuito (limitado) / Pago	Gratuito
Recursos extras	Scheduler, logs, alertas, UI	Totalmente manual
Ideal para	Equipes e ambientes gerenciados	Aprendizado e pequenos projetos

Organização das camadas

O DBT incentiva uma estrutura em camadas, baseada em boas práticas de modelagem de dados analíticos:

- **1. Staging (`models/staging/`)**: Prepara os dados "atomizados" vindos das fontes (raw). Renomeia colunas, padroniza tipos. Única referência às tabelas de origem.
- **2. Intermediate (`models/intermediate/`)**: Agrega lógica de negócio intermediária. Organiza os dados por grupo de negócio (ex.: `finance/`).
- **3. Marts (`models/marts/`)**: Dados finais preparados para análise e relatórios. Combina dados processados para formar as entidades úteis para o negócio.
- **Outras:** `raw` (dados brutos, sem alterações) e `utilities` (modelos utilitários, ex: tabela de datas).

O que o DBT faz vs. O que não faz

O que o DBT faz

- Transforma dados com SQL
- Controla dependências entre modelos
- Roda transformações com versionamento (Git)
- Valida com testes automatizados
- Documenta todo o projeto com interface
- Gera lineage (linhagem) de dados

O que o DBT não faz

- Não extrai dados de fontes (não faz ingestão)
- Não carrega dados para o banco
- Não cria dashboards ou relatórios
- Não armazena dados em si
- Não é ferramenta de Machine Learning

Documentação com DBT

Com um único comando (`dbt docs generate`), o DBT cria uma documentação HTML com:

- Definição de cada modelo
- Descrição das colunas
- Relações entre tabelas (lineage)
- Testes aplicados
- Seeds, snapshots, macros

É possível acessar via navegador (`dbt docs serve`) ou publicar no DBT Cloud.

The screenshot shows a web browser window titled "Overview - DBT Docs" at the URL "localhost:8080/index.html#/model/fishtown_internal_analytics.fct_subscription_transactions". The interface has a sidebar on the left labeled "Project" which lists various schema and model folders. The main content area displays the details for the "analytics.fct_subscription_transactions" table. It includes sections for "Description" (which states it's a Fact Table for Subscription Transactions), "Data Sources" (listing Trello boards, Quickbooks, and Stripe), and "Columns" (a table with columns for TESTS, COLUMN, and TYPE). To the right of the main content is a dark panel titled "Lineage Graph" containing a diagram where a blue node labeled "subscription_transactions_typed" is connected by a line to a pink node labeled "fct_subscription_transactions".

Testes no DBT

DBT possui testes **integrados e customizados**, aplicados diretamente no SQL.

Testes integrados (genéricos):

Definidos em arquivos `*.yml` para garantir a qualidade dos dados.

```
models:  
  - name: vendas  
    columns:  
      - name: id_venda  
        tests:  
          - unique  
          - not_null
```

DBT Avançado

Macros, Snapshots e
Seeds

Avançado: Macros

O que são?

Macros são funções reutilizáveis escritas em Jinja (um template engine para Python), usadas para gerar SQL dinamicamente dentro do DBT.

Servem para:

- Reduzir repetição de código SQL
- Criar lógica condicional ou parametrizada
- Padronizar transformações

Avançado: Exemplo de Macro

macros/formatar_datas.sql

```
{% macro formatar_data(coluna) %}  
    date_trunc('day', {{ coluna }})  
{% endmacro %}
```

models/meu_modelo.sql

```
SELECT  
    {{ formatar_data("data_pedido") }} AS data_formatada  
FROM {{ ref('stg_pedidos') }}
```

Avançado: Snapshots

O que são?

Snapshots são usados para capturar mudanças históricas em dados dimensionais, como SCDs (Slowly Changing Dimensions). Eles permitem rastrear como um registro mudou ao longo do tempo.

Servem para:

- Registrar cada vez que o endereço de um cliente mudar, mantendo o histórico.
- Auditar mudanças em tabelas de configuração.

Avançado: Exemplo de Snapshot

snapshots/clientes_snapshot.sql

```
{% snapshot clientes_snapshot %}
{{ config(
    target_schema='snapshots',
    unique_key='id_cliente',
    strategy='check',
    check_cols=['nome', 'email', 'endereco']
)
}}
SELECT * FROM {{ source('vendas', 'clientes') }}
{% endsnapshot %}
```

dbt_project.yml

```
# Configuração de snapshots
snapshots:
  my_dbt_project:
    +target_schema: snapshots
    +strategy: check
```

Avançado: Seeds

O que são?

Seeds são arquivos CSV usados como tabelas estáticas no projeto DBT. Ideal para dados pequenos e constantes como tabelas de referência, códigos de UF, calendários etc.

Como funciona:

- Salve um CSV na pasta /seeds/ (ex: seeds/calendario.csv)
- Execute dbt seed
- O DBT converte o CSV em uma tabela no banco de dados.

Estrutura de um Projeto DBT

Estrutura de Pastas Principal

Um projeto DBT é organizado para promover modularidade, reutilização e testabilidade.

Abaixo, a estrutura de pastas detalhada.

```
meu_projeto_dbt/
├── models/      # Modelos de transformação (SQL)
├── seeds/       # Arquivos CSV com dados estáticos
├── tests/        # Testes de dados personalizados
├── macros/      # Funções reutilizáveis (Jinja)
├── snapshots/   # Configuração para capturar mudanças
└── dbt_project.yml # Configuração principal
```

Arquivo de Configuração: `dbt_project.yml`

É o coração do projeto. Este arquivo define:

- Nome do projeto e versão.
- Perfil de conexão a ser usado (do `profiles.yml`).
- Caminhos dos recursos (`model-paths`, etc.).
- Configurações globais (ex: materialização).

```
name: 'meu_projeto_dbt'  
version: '1.0.0'  
profile: 'default'  
  
model-paths: ["models"]  
test-paths: ["tests"]  
seed-paths: ["seeds"]  
  
models:  
  meu_projeto_dbt:  
    staging:  
      materialized: view  
    marts:  
      materialized: table
```

A Camada de Transformação (`models/`)

Esta é a pasta mais importante, onde toda a lógica de transformação reside.

- **Organização em Camadas:**
 - `models/staging/`: Limpeza e padronização inicial.
 - `models/intermediate/`: Lógica de negócio intermediária.
 - `models/marts/`: Modelos finais, prontos para consumo.
- **Arquivos de Modelo (.sql):** Usam `{{ ref(...) }}` para criar dependências e `{{ source(...) }}` para referenciar dados brutos.
- **Arquivos de Propriedades (.yml):** Usados para documentar, adicionar testes e definir `sources`.

Exemplo: Modelo `sql`

A maioria dos modelos são arquivos SQL.

Neles, você usa as funções `{{ source(...) }}` para referenciar dados brutos e `{{ ref(...) }}` para criar dependências entre modelos.

models/staging/stg_pedidos.sql

```
SELECT
    id AS id_pedido,
    id_cliente,
    status,
    valor AS valor_total
FROM {{ source('loja', 'pedidos') }}
```

Exemplo: Propriedades `*.yml`

Arquivos `*.yml` são usados para documentar, adicionar testes e declarar `sources` (fontes de dados brutos).

Isso alimenta o site de documentação e o sistema de testes.

models/staging/stg_loja.yml

```
version: 2
sources:
  - name: loja
    database: raw
    tables:
      - name: pedidos

models:
  - name: stg_pedidos
    columns:
      - name: id_pedido
        tests:
          - unique
          - not_null
```

Outras Pastas Essenciais

- `seeds/`: Armazena arquivos `.csv` com dados estáticos (ex: feriados, UFs). Carregados com `dbt seed` e referenciados com `{{ ref('nome_do_arquivo') }}`.
- `tests/`: Contém testes de dados personalizados (singulares). São consultas SQL que devem retornar zero linhas para o teste passar.

Outras Pastas Essenciais (Cont.)

- `macros/`: Define macros em Jinja (SQL reutilizável). Útil para evitar repetição e padronizar lógica (ex: `{{ formatar_moeda('valor_centavos') }}`).
- `snapshots/`: Permite capturar o histórico de mudanças em uma tabela (SCD Tipo 2). Executado com `dbt snapshot`.

Documentação no DBT

Importância da Documentação

A documentação é um pilar do dbt, escrita em arquivos ` .yml` junto com o código.

- **Fonte Única da Verdade (SSOT):** Centraliza o conhecimento sobre os dados, evitando documentos desatualizados.
- **Data Discovery:** Facilita a descoberta de quais dados estão disponíveis e como podem ser usados.
- **Confiança e Governança:** Aumenta a confiança ao expor a lógica, os testes e a linhagem de ponta a ponta.
- **Colaboração:** Permite que analistas, engenheiros e negócio falem a mesma língua.

Documentação: Exemplo Completo

Este exemplo documenta uma fonte (`source`) e um modelo (`model`), incluindo descrições e testes de relacionamento e valores aceitos.

models/staging/stg_ecommerce.yml

```
version: 2
sources:
  - name: ecommerce_raw
    tables:
      - name: pedidos
models:
  - name: stg_pedidos
    description: "Limpa e padroniza dados de pedidos."
    columns:
      - name: id_pedido
        tests:
          - unique
          - not_null
      - name: status_pedido
        tests:
          - accepted_values:
              values: ['proc', 'env', 'ent', 'canc']
      - name: id_cliente
        tests:
```

Comandos Mais Frequentes

- **dbt run:** Executa todos os modelos.
- **dbt test:** Executa todos os testes.
- **dbt build:** Atalho para `run`, `test`, `seed`, `snapshot`.
- **dbt compile:** Compila Jinja para SQL (para depuração).
- **dbt seed:** Carrega arquivos CSV da pasta `seeds`.
- **dbt snapshot:** Executa os snapshots.
- **dbt docs generate:** Gera o site de documentação.
- **dbt docs serve:** Inicia o servidor web da documentação.
- **dbt deps:** Baixa pacotes externos.
- **dbt clean:** Remove artefatos de compilação.

Perguntas?