

EBOOK DE FÁCIL APRENDIZADO

PRIMEIROS PASSOS COM DBT - DATA BUILD TOOL



Coffee and Tips

Tech Tutorials



O DBT tem sido utilizado por muitas empresas na área de Dados e acredito que podemos extrair bons insights neste post sobre ele. Esse vai ser um post prático mostrando como o DBT funciona e espero que vocês gostem.

O que é DBT?

DBT significa Data Build Tool e permite que equipes transformem os dados já carregados em seu warehouse através de operações de DML como um simples Select. DBT representa o T no processo de ELT, ou seja, ele não trabalha para extrair e carregar dados mas sim, para transformá-los.

Passo 1: Criando o projeto DBT

Agora, assumimos que o DBT já esteja instalado, mas se não estiver, recomendo consultar este <https://docs.getdbt.com/docs/core/installation> para mais informações.

Após a instalado, você pode criar um novo projeto usando CLI ou pode clonar este projeto do repositório DBT no Github no link a seguir <https://github.com/dbt-labs/dbt-starter-project>

Aqui para este ebook, vamos usar o modo CLI para criar nosso projeto e também para concluir as próximas etapas. Para criar um novo projeto, execute o comando abaixo no seu terminal.

dbt init

Coffee and Tips
Tech Tutorials

Depois de executar o comando acima, você precisa digitar o nome do projeto e qual warehouse ou banco de dados você vai usar conforme a imagem abaixo.

Neste ebook, vamos usar o adaptador do postgres. É muito importante que você tenha o banco de dados postgres já instalado ou você pode criar uma imagem postgres usando o docker.

```
Enter a name for your project (letters, digits, underscore): dbt_blog
Which database would you like to use?
[1] bigquery
[2] snowflake
[3] redshift
[4] postgres

(Don't see the one you want? https://docs.getdbt.com/docs/available-adapters)

Enter a number: 4
20:59:01
Your new dbt project "dbt_blog" was created!

For more information on how to configure the profiles.yml file,
please consult the dbt documentation here:

https://docs.getdbt.com/docs/configure-your-profile

One more thing:

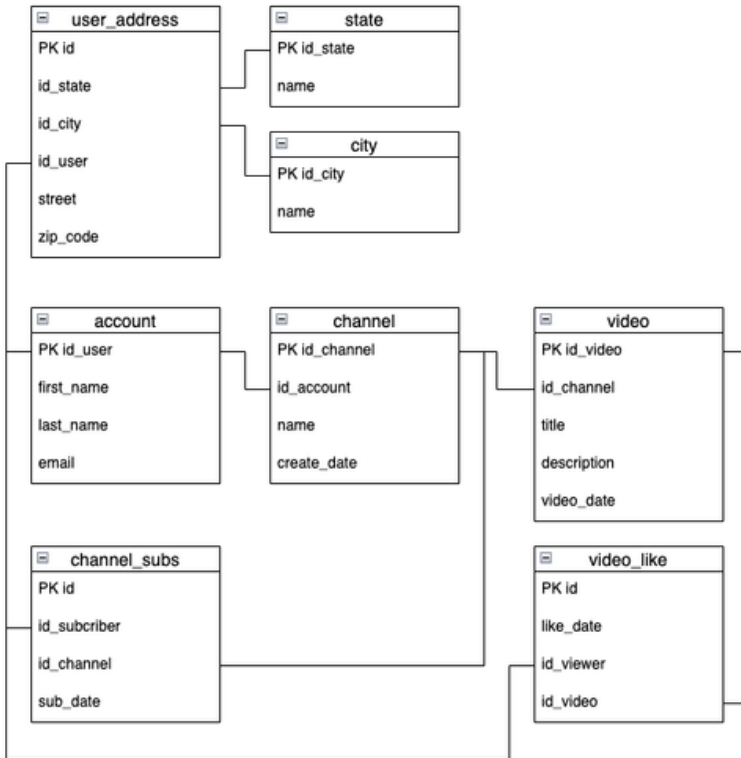
Need help? Don't hesitate to reach out to us via GitHub issues or on Slack:

https://community.getdbt.com/

Happy modeling!
```

Sobre os adaptadores, o DBT suporta vários deles e você pode conferir no link a seguir <https://docs.getdbt.com/docs/supported-data-platforms>.

Criei uma estrutura de tabela e também carreguei os dados simulando dados de uma plataforma de vídeo chamada **wetube** e vamos utilizá-los para entender como o DBT funciona. Acompanhe a estrutura:



Passo 2: Estrutura e mais sobre DBT

Após executar o comando `dbt init` para criar o projeto, uma estrutura de pastas e arquivos abaixo será criada.

Não vou falar sobre todos os diretórios do projeto, mas gostaria de focar em dois deles.

```
dbt_blog
├── .gitignore
├── .idea
│   ├── .gitignore
│   ├── aws.xml
│   ├── codeStyles
│   │   ├── Project.xml
│   │   └── codeStyleConfig.xml
│   ├── dbt_blog.iml
│   ├── misc.xml
│   ├── modules.xml
│   ├── runConfigurations.xml
│   └── workspace.xml
├── README.md
├── analyses
│   └── .gitkeep
├── dbt_packages
├── dbt_project.yml
├── logs
│   ├── dbt.log
│   └── dbt.log.legacy
├── macros
│   └── .gitkeep
├── models
│   └── example
│       ├── my_first_dbt_model.sql
│       ├── my_second_dbt_model.sql
│       └── schema.yml
├── seeds
│   └── .gitkeep
├── snapshots
│   └── .gitkeep
├── target
│   ├── graph.gpickle
│   ├── manifest.json
│   ├── partial_parse.msgpack
│   └── run_results.json
├── tests
│   └── .gitkeep
```

Sources

Antes de falarmos sobre os dois diretórios, vamos falar sobre os Sources, são basicamente os dados já carregados em seu warehouse. No processo DBT, as fontes têm o mesmo significado de dados brutos.

Não há pastas que representem dados Sources para este projeto, mas você precisa saber sobre este termo pois vamos configurar tabelas já criadas como Sources para as próximas seções.

Seeds

Seeds é um diretório que oferece um mecanismo interessante e útil para carregar dados estáticos em seu warehouse por meio de arquivos CSV. Se você deseja carregar esses dados, você precisa criar um arquivo CSV neste diretório e executar o comando abaixo.

dbt seed

Para cada campo no arquivo CSV, o DBT irá inferir os tipos e criará tabelas e suas colunas no warehouse ou banco de dados.

Models

O DBT funciona com o paradigma de Model, a ideia principal é que você pode criar modelos através da transformações utilizando instruções SQL baseadas em fontes de tabelas ou modelos existentes

Cada arquivo SQL localizado na pasta de model criará um modelo em seu warehouse ou banco de dados quando o comando abaixo for executado.

dbt run

Lembre-se que um modelo pode ser criado através de uma fonte ou outro modelo e não se preocupe com isso, vou mostrar mais detalhes sobre isso.

Passo 3: Configurando a configuração com o banco de dados

Com o projeto já criado, precisamos configurar a conexão com o banco de dados e aqui neste ebook vamos usar o postgres como banco de dados.

Depois de inicializar o projeto, vários arquivos são criados e um deles é chamado de **profiles.yml**.

profiles.yml é o arquivo responsável por controlar os diferentes perfis/profiles para as diferentes conexões com os bancos de dados, como ambiente de desenvolvimento e produção. Se você notou, não podemos ver este arquivo na imagem acima porque este arquivo é criado fora do projeto para evitar credenciais que sejam confidenciais. Você pode encontrar esse arquivo no diretório `~/dbt/`.

Se você observar, temos um perfil chamado **dbt_blog** e um destino chamado **dev**, por padrão, o destino refere-se a **dev** com as configurações de conexão do banco de dados. Além disso, é possível criar um ou mais perfis e alvos(**target**), permitindo trabalhar com diferentes ambientes.

```
dbt_blog:
  outputs:

  dev:
    type: postgres
    threads: 1
    host: localhost
    port: 5432
    user: [dev_username]
    password: [dev_password]
    dbname: get_dbt
    schema: wetube

  target: dev
```

Outro detalhe importante é que o perfil **dbt_blog** deve ser especificado no arquivo `dbt_project.yml` como um perfil padrão. Nas próximas seções, discutiremos o que é e como o arquivo **dbt_project.yml** funciona.

Passo 4: Criando o arquivo dbt_project.yml

Todo projeto DBT possui um arquivo dbt_project.yml, você pode configurar informações como nome do projeto, diretórios, perfis e tipo de materialização.

```
name: 'dbt_blog'
version: '1.0.0'
config-version: 2
profile: 'dbt_blog'

model-paths: ["models"]
analysis-paths: ["analyses"]
test-paths: ["tests"]
seed-paths: ["seeds"]
macro-paths: ["macros"]
snapshot-paths: ["snapshots"]

target-path: "target" # directory which will store compiled SQL
files
clean-targets:        # directories to be removed by `dbt clean`
  - "target"
  - "dbt_packages"

models:
  dbt_blog:
    # Config indicated by + and applies to all files under
    models/example/
  mart:
    +materialized: table
```

Observe que o campo de **profile** foi configurado como o mesmo **profile** especificado no arquivo **profiles.yml** e outro detalhe importante é sobre o campo **materialized**. Aqui foi configurado como um valor **table**, mas por padrão, é uma **view**.

O campo **materialized** permite que você crie modelos como uma tabela ou view em cada execução. Existem outros tipos de materialização, mas não vamos discutir aqui e eu recomendo ver a documentação do dbt.

Passo 5: Criando nosso primeiro modelo

Criando os primeiros arquivos

Vamos mudar um pouco e vamos criar uma subpasta no diretório do **model** chamada **mart** e dentro desta pasta vamos criar nossos arquivos **.SQL** e também outro arquivo importante que ainda não discutimos chamado **schema.yml**.

Coffee and Tips
Tech Tutorials

Criando o arquivo schema

Os arquivos de schema são usados para mapear fontes e documentar modelos como o nome do modelo, colunas e muito mais. Agora você pode criar um arquivo chamado `schema.yml` e preencher com as informações abaixo.

```
version: 2

sources:
  - name: wetube
    tables:
      - name: account
      - name: city
      - name: state
      - name: channel
      - name: channel_subs
      - name: video
      - name: video_like
      - name: user_address

models:
  - name: number_of_subs_by_channel
    description: "Number of subscribers by channel"
    columns:
      - name: id_channel
        description: "Channel's ID"
        tests:
          - not_null
      - name: channel
        description: "Channel's Name"
        tests:
          - not_null
      - name: num_of_subs
        description: "Number of Subs"
        tests:
          - not_null
```

Sources: No campo source você pode incluir tabelas do seu warehouse ou banco de dados que serão utilizadas na criação do modelo.

models: No campo models você pode incluir o nome do modelo, colunas e suas descrições



Coffee and Tips

Tech Tutorials

Criando um modelo

Esta parte é onde podemos criar scripts .SQL que resultarão em nosso primeiro modelo.

Para o primeiro modelo, vamos criar uma instrução SQL para representar um modelo que podemos ver os números de inscritos do canal. Vamos criar um arquivo chamado **number_of_subs_by_channel.sql** e preenchê-lo com os scripts abaixo.

```
with source_channel as (  
    select * from  
        {{ source('wetube', 'channel') }}  
),  
  
source_channel_subs as (  
    select * from  
        {{ source('wetube','channel_subs') }}  
),  
  
number_of_subs_by_channel as (  
    select  
        source_channel.id_channel,  
        source_channel.name,  
        count(source_channel_subs.id_subscriber) num_subs  
    from source_channel_subs  
        inner join source_channel using (id_channel)  
    group by 1, 2  
)  
  
select * from number_of_subs_by_channel
```

Entendendo o modelo

- Observe que temos vários scripts separados por expressão de tabela comum (CTE) que se torna útil para entender o código.
- O DBT permite usar o template Jinja `{{ }}` trazendo uma maior flexibilidade ao nosso código.
- O uso da palavra-chave **source** dentro do modelo Jinja significa que estamos nos referindo a tabelas de origem. Para referenciar um modelo, você precisa usar a palavra-chave **ref**.
- A última instrução `SELECT` baseada nas tabelas de origem (`source`) irá gerar o modelo (`model`) como tabela no banco de dados.

Executando o nosso primeiro modelo

Execute o comando abaixo para criar nosso primeiro modelo baseado nos arquivos anteriores.

dbt run

Saída

```
Concurrency: 1 threads (target='dev')

1 of 3 START table model wetube.my_first_dbt_model ..... [RUN]
1 of 3 OK created table model wetube.my_first_dbt_model ..... [SELECT 2 in 0.22s]
2 of 3 START table model wetube.number_of_subs_by_channel ..... [RUN]
2 of 3 OK created table model wetube.number_of_subs_by_channel ..... [SELECT 3 in 0.08s]
3 of 3 START view model wetube.my_second_dbt_model ..... [RUN]
3 of 3 OK created view model wetube.my_second_dbt_model ..... [CREATE VIEW in 0.08s]

Finished running 2 table models, 1 view model in 0.65s.

Completed successfully
```

Coffee and Tips
Tech Tutorials

Criando um novo modelo

Imagine que precisamos criar um novo modelo contendo as informações da conta e seus canais. Vamos voltar ao arquivo **schema.yml** para adicionar esse novo modelo.

```
- name: account_information
  description: "Model containing account information and it's
  channels"
  columns:
    - name: id_account
      description: "Account ID"
      tests:
        - not_null
    - name: first_name
      description: "First name of user's account"
      tests:
        - not_null
    - name: last_name
      description: "Last name of user's account"
      tests:
        - not_null
    - name: email
      description: "Account's email"
      tests:
        - not_null
    - name: city_name
      description: "city's name"
      tests:
        - not_null
    - name: state_name
      description: "state's name"
      tests:
        - not_null
    - name: id_channel
      description: "channel's Id"
      tests:
        - not_null
    - name: channel_name
      description: "channel's name"
      tests:
        - not_null
    - name: channel_creation
      description: "Date of creation name"
      tests:
        - not_null
```

Agora, vamos criar um novo arquivo SQL e nomeá-lo como **account_information.sql** e adicionar os scripts abaixo:

```
with source_channel as (  
    select * from  
        {{ source('wetube', 'channel') }}  
),  
  
source_city as (  
    select * from  
        {{ source('wetube','city') }}  
),  
  
source_state as (  
    select * from  
        {{ source('wetube','state') }}  
),  
  
source_user_address as (  
    select * from  
        {{ source('wetube','user_address') }}  
),  
  
source_account as (  
    select * from  
        {{ source('wetube','account') }}  
),  
account_info as (  
    select  
        account.id_user as id_account,  
        account.first_name,  
        account.last_name,  
        account.email,  
        city.name as city_name,  
        state.name as state_name,  
        channel.id_channel,  
        channel.name as channel,  
        channel.creation_date as channel_creation  
    FROM source_account account  
        inner join source_channel channel on (channel.id_account =  
account.id_user)  
        inner join source_user_address user_address using  
(id_user)  
        inner join source_state state using (id_state)  
        inner join source_city city using (id_city)  
)  
select * from account_info
```

Criando nosso último modelo

Para o nosso último modelo, vamos criar um modelo sobre quantas curtidas tem um vídeo. Vamos alterar novamente o **schema.yml** para descrever e documentar nosso futuro e último modelo.

```
- name: total_likes_by_video
  description: "Model containing total of likes by video"
  columns:
    - name: id_channel
      description: "Channel's Id"
      tests:
        - not_null
    - name: channel
      description: "Channel's name"
      tests:
        - not_null
    - name: id_video
      description: "Video's Id"
      tests:
        - not_null
    - name: title
      description: "Video's Title"
      tests:
        - not_null
    - name: total_likes
      description: "Total of likes"
      tests:
        - not_null
```

Crie um arquivo chamado **total_likes_by_video.sql** e coloque o código abaixo:

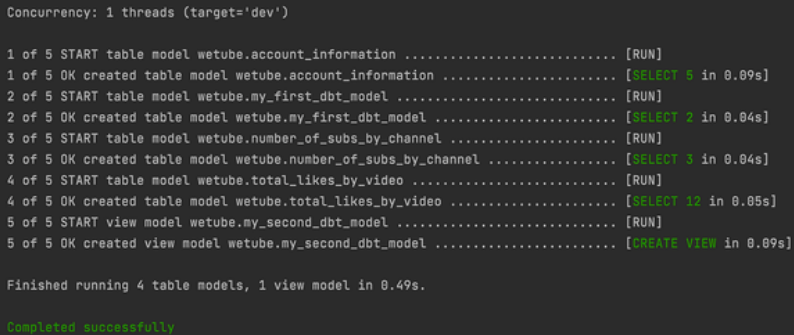
```
with source_video as (  
    select * from  
        {{ source('wetube','video') }}  
),  
source_video_like as (  
    select * from  
        {{ source('wetube','video_like') }}  
),  
source_account_info as (  
    select * from  
        {{ ref('account_information') }}  
),  
source_total_like_by_video as (  
  
    select source_account_info.id_channel,  
source_account_info.channel,  
        source_video.id_video, source_video.title, count(*) as  
total_likes  
    FROM source_video_like  
        inner join source_video using (id_video)  
        inner join source_account_info using (id_channel)  
    GROUP BY source_account_info.id_channel,  
        source_account_info.channel,  
        source_video.id_video,  
        source_video.title  
    ORDER BY total_likes DESC  
)  
  
select * from source_total_like_by_video
```

Executando novamente

Após a criação dos arquivos, vamos executar DBT novamente para criar os novos modelos.

dbt run

Saída



```

Concurrency: 1 threads (target='dev')














1 of 5 START table model wetube.account_information ..... [RUN]
1 of 5 OK created table model wetube.account_information ..... [SELECT 5 in 0.09s]
2 of 5 START table model wetube.my_first_dbt_model ..... [RUN]
2 of 5 OK created table model wetube.my_first_dbt_model ..... [SELECT 2 in 0.04s]
3 of 5 START table model wetube.number_of_subs_by_channel ..... [RUN]
3 of 5 OK created table model wetube.number_of_subs_by_channel ..... [SELECT 3 in 0.04s]
4 of 5 START table model wetube.total_likes_by_video ..... [RUN]
4 of 5 OK created table model wetube.total_likes_by_video ..... [SELECT 12 in 0.05s]
5 of 5 START view model wetube.my_second_dbt_model ..... [RUN]
5 of 5 OK created view model wetube.my_second_dbt_model ..... [CREATE VIEW in 0.09s]

Finished running 4 table models, 1 view model in 0.49s.

Completed successfully

```

Os modelos foram criados no banco de dados e você pode executar instruções select diretamente em seu banco de dados para verificá-lo. Perceba que além dos modelos criados, você pode notar as demais tabelas que foram mapeadas no arquivo **schema.yml** e que já existiam na estrutura do banco inicial. Lembre-se do mecanismo de criar tabelas estáticas através do diretório **Seeds**, pode ser uma boa escolha para uma carga inicial.

- ▼  Tables (12)
 - >  account
 - >  account_information
 - >  channel
 - >  channel_subs
 - >  city
 - >  my_first_dbt_model
 - >  number_of_subs_by_channel
 - >  state
 - >  total_likes_by_video
 - >  user_address
 - >  video
 - >  video_like

Coffee and Tips

Tech Tutorials

Modelo: account_information

Data Output		Explain	Messages	Notifications				
id_account bigint	first_name character (50)	last_name character (50)	email character (100)	city_name character (50)	state_name character (50)	id_channel bigint	channel character (100)	channel_creation date
1	Joao	Sanches	Sanches@dbt.com	New York	New York	1	Yes Theory	2010-02-03
2	Jorge	Jeff	Jeff@dbt.com	San Francisco	California	3	Casey Neistat	2019-02-19
3	Sheldon	Santos	Santos@dbt.com	New York	New York	4	James Corden	2018-02-21
4	Joe	Rogan	joe@dbt.com	San Francisco	California	5	Joe Rogan	2010-12-12
5	Jimmy	Donaldson	donaldson@dbt.com	New York	New York	2	Mr Beast	2010-02-02

Modelo: number_of_subs_by_channel

Data Output		Explain	Messages	Notifications
id_channel bigint	name character (100)	num_subs bigint		
1	Casey Neistat	1		
2	Mr Beast	4		
3	Yes Theory	1		

Modelo: total_likes_by_video

Data Output		Explain	Messages	Notifications
id_channel bigint	channel character (100)	id_video bigint	title character (250)	total_likes bigint
1	Mr Beast	1	I Built Willy Wonka's Chocolate Factory!	5
2	Mr Beast	4	Giving 10,000 Presents To Kids For Christmas	3
3	James Corden	11	James in LA	3
4	Joe Rogan	6	Joe Rogan Experience #1153 - Macaulay Culkin	2
5	Mr Beast	2	\$10,000 Every Day You Survive Prison	2
6	Casey Neistat	15	i miss being a YouTuber	2
7	Mr Beast	3	Would You Swim With Sharks For \$100,000?	2
8	Casey Neistat	16	Make It Count	2
9	Yes Theory	19	I Spent 24 Hours in Korea with No Money	2
10	Yes Theory	20	I Bought a \$150,000 Passport that can Travel the World ...	2
11	Joe Rogan	10	Joe Rogan Experience #1411 - Robert Downey Jr	1
12	Mr Beast	5	I Won Every Prize At A Theme Park	1

Passo 6: DBT Docs

Documentação

Depois de gerados nossos modelos, agora vamos gerar documentos com base nestes. O DBT gera uma documentação completa sobre modelos (models), sources e suas colunas, através de uma página da web.

Gerando as docs

dbt docs generate

Disponibilizando as docs no servidor Web

Após a geração dos documentos, você pode executar o comando abaixo no seu terminal para iniciar um servidor da Web na porta 8080 e consultar a documentação localmente utilizando o seu navegador. Caso o navegador não abra automaticamente, digite o seguinte endereço localhost:8080 no seu navegador.

dbt docs serve

dbt Search for models...

Overview Project Database

Sources

- wetube
 - account
 - channel
 - channel_sub
 - city
 - state
 - user_address
 - video
 - video_like

Projects

- dbt_blog
 - models
 - example
 - main
 - account_information
 - number_of_subs_by_channel
 - total_likes_by_video

wetube source
Details Description Sources

Details

LOOKER	OWNER	DATABASE	SCHEMA	TABLES
	postgres	get_dbt	wetube	8

Description

This is not currently documented

Source Tables

SOURCE	TABLE	DESCRIPTION	LINK	MODEL
wetube	account		View docs	
wetube	channel		View docs	
wetube	channel_sub		View docs	
wetube	city		View docs	
wetube	state		View docs	
wetube	user_address		View docs	
wetube	video		View docs	
wetube	video_like		View docs	

dbt Search for models...

Overview Project Database

Sources

- wetube

Projects

- dbt_blog
 - models
 - example
 - main
 - account_information
 - number_of_subs_by_channel
 - total_likes_by_video

account_information table
Details Description Columns Referenced By Depends On SQL

Details

TAGS	OWNER	TYPE	PACKAGE	RELATION
untagged	postgres	table	dbt_blog	get_dbt.wetube.account_information

Description

Model containing account information and it's channels

Columns

COLUMN	TYPE	DESCRIPTION	TESTS	MODEL
id_account	bigint	Account ID	N	✓
first_name	character(50)	First name of user's account	N	✓
last_name	character(50)	Last name of user's account	N	✓
email	character(100)	Account's email	N	✓
city_name	character(50)	city's name	N	✓
state_name	character(50)	state's name	N	✓
id_channel	bigint	channel's id	N	✓
channel	character(100)			
channel_creation	date	Date of creation name	N	✓
channel_name		channel's name	N	✓

dbt Search for models...

Overview Project Database

Sources

- wetube

Projects

- dbt_blog
 - models
 - example
 - main
 - account_information
 - number_of_subs_by_channel
 - total_likes_by_video

number_of_subs_by_channel table
Details Description Columns Referenced By Depends On SQL

Details

TAGS	OWNER	TYPE	PACKAGE	RELATION
untagged	postgres	table	dbt_blog	get_dbt.wetube.number_of_subs_by_channel

Description

Number of subscribers by channel

Columns

COLUMN	TYPE	DESCRIPTION	TESTS	MODEL
id_channel	bigint	Channel's ID	N	✓
name	character(100)			
num_subs	bigint			
channel		Channel's Name	N	✓
num_of_subs		Number of Subs	N	✓

The screenshot shows the dbt interface for the 'total_likes_by_video' table. The table is located in the 'models' directory of the 'dbt_blog' project. The table is untagged and is a PostgreSQL table. The description is 'Model containing total of likes by video'. The columns are listed in a table below.

TABLE	OWNER	TYPE	POSTGRES	SOLUTION	
untagged		postgres	table	dbt_blog	git_dbt_wetube_total_likes_by_video

COLUMNS	TYPE	DESCRIPTION	TEXTS	MARKET
id_channel	bigint	Channel's id	N	>
channel	character(100)	Channel's name	N	>
id_video	bigint	Video's id	N	>
title	character(250)	Video's Title	N	>
total_likes	bigint	Total of likes	N	>

Lineage

Outro detalhe sobre a documentação é que você pode ver através de um Lineage os modelos e suas dependências.





Código no Github

Você pode conferir esse código no nosso repositório do Github <https://github.com/coffeeandtips-tech/first-steps-with-dbt>

Curtiu? Eu espero que tenha gostado!

CURTIU? ACESSE MAIS TUTORIAIS COMO ESSE EM COFFEEANDTIPS.COM

E NOS SIGA NO [INSTAGRAM](#)



Coffee and Tips

Tech Tutorials