

Welcome Everyone! 🥳

We are so excited that you start your dbt journey with us! Shall we start?

Presentation link: bit.ly/bdf-dbt-slides

dbt Cloud link: <https://cloud.getdbt.com/signup/>

💛 Get in touch: gergely.foldi@infinitelambda.com

Create your own Colab environment!

1. Click on File on the top left corner
2. Save copy in Drive

If you dont have Drive:

1. Click on File on the top left corner
2. Download .ipynb
3. Click on File and Open notebook
4. Upload the previously downloaded file

▸ Setup your dbt Cloud environment

Database connection details

Database connection below used during the tutorial. Please feel free to use any cloud database connection you wish!

Type: snowflake

Account: gezjpyc-datafissiongroup1

Role: ROLE_DEVELOPER_BDF

Database: BDF_SANDBOX

Warehouse: BP_DATA_FORUM_WH_XS

User: ***

Password: ***

Schema: dbt_{your.name}

Target name: default

Threads: 4

Repository details

Choose Managed repository.

Give a custom name. Example: My BDF repo

Almost there. Go to the hamburger menu, *Develop* tab. And initiate repository. You are ready 🐱

🔍 ↪ 4 cells hidden

Have a look at your 1st dbt project

Repository structure

Default folders: analyses, macros, models, seeds

Important: `dbt_project.yml`

Try running the examples!

🧠 Tutorial use case summary

Our goal is to build a 🍷 DWH with dbt.

Sources:

- 1 Snowflake DB `BDF_RAW` with 3 tables (`customer`, `nation`, `orders`)
- 1 csv file

DWH design:

- 2 layers
- `PSA` : persistent staging area
- `ANALYTICS` : business layer with star schema

▶ Build our PSA layer

Reference: <https://docs.getdbt.com/docs/building-a-dbt-project/using-sources>

1. Create a new folder `psa` under `models`
2. Add new file `00_psa.yml`. Insert the following:

```
version: 2

sources:
  - name: tpch_sf1
    database: BDF_RAW
    schema: tpch_sf1
    tables:
      - name: customer
```

3. Add a new file `psa_tpch_customer.sql` Insert the following:

```
select * from {{ source('tpch_sf1', 'customer') }}
```

4. Hit `dbt compile`. Check your `\target` folder.
5. Now hit `dbt run`
6. Reference `nation` table in the `schema.yml` as a source

```
tables:
  - name: customer
  - name: nation
```

7. Add a new file `psa_tpch_nation.sql` Insert the following:

```
select * from {{ source('tpch_sf1', 'nation') }}
```

8. Hit `dbt run`
9. 📁 Exercise01: Add `psa_tpch_orders` model into your PSA layer. Then `dbt run`

[] ↪ 3 cells hidden

▼ Build our ANALYTICS layer

Reference: <https://docs.getdbt.com/reference/dbt-jinja-functions/ref>

1. Create a new folder `models/analytics`
2. Add a new model `dim_customer.sql`

```
select
  a.c_custkey      as customer_key
```

```
,a.c_name          as customer_name
,b.n_name          as nation_name
from {{ref('psa_tpch_customer')}} a
```


```
left join {{ref('psa_tpch_nation')}} b
  on a.c_nationkey = b.n_nationkey
```

3. Try `dbt compile` first. Check your `\target` folder again.

4. Now hit `dbt run`

5. Add a new model **fact__orders.sql**.

```
select
  o_orderkey      as order_key
 ,o_custkey       as customer_key
 ,o_totalprice    as total_price
```

5.  Exercise02: reference your orders model from the PSA layer

6. Hit `dbt run`

7. Wow, you have built a little DWH baby with 2 layers 🍌

8. Please check your model **lineage**. How cool is that!!!

```
%sql select * from BDF_SANDBOX.DBT_GERGELYFOLDI.DIM__CUSTOMER limit 10
```

```
%sql select * from BDF_SANDBOX.DBT_GERGELYFOLDI.FACT__ORDERS limit 10
```

Advance your dbt commands

Reference: <https://docs.getdbt.com/reference/node-selection/syntax>

So far we used `dbt run` and `dbt compile`. That's great but it is just the surface!

Try these commands:

```
dbt run --select dim__customer
```

```
dbt run --select +dim__customer
```

```
dbt run --select psa
```

Insert from csv file

Reference: <https://docs.getdbt.com/docs/building-a-dbt-project/seeds>

1. Create a new file `seed__sales_segment_customer.csv` under the `seeds` folder. Insert:

```
C_CUSTKEY,SALES_SEGMENT
1,HIGH
2,LOW
3,MID
4,HIGH
5,middle
6,HIGH
7,LOW
8,MID
9,LOW
10,HIGH
11,LOW
12,MID
13,middle
14,MID
15,HIGH
```

```
16,LOW
17,low
18,MID
19,HIGH
20,HIGH
```

2. Run `dbt seed`
3. Check your target schema. (New table should have been added.)
4. 🔄 Exercise03: Join your new seed with the `dim__customer` model and add the `sales_segment` column. (Hint: `ref()` works with seeds)

▸ Organise your dbt project

Reference:

https://docs.getdbt.com/reference/dbt_project.yml

<https://docs.getdbt.com/docs/building-a-dbt-project/building-models/materializations>

Change the location of your models in the database

1. Add the following configuration into your `dbt_project.yml`

```
models:
  my_new_project:
    psa:
      database: BDF_PSA_DEV
    analytics:
      database: BDF_ANALYTICS_DEV
```

2. Change the database location of your seeds too.

```
seeds:
  my_new_project:
    database: BDF_PSA_DEV
```

Change materialization of your models

1. 🔄 Exercise04: Change the materialization of your models in PSA to view and in ANALYTICS to table. (Hint: you can use the `materialized` property)

😄 If you are ready, hit `dbt run`. Check what has been changed!

[] ↪ 2 cells hidden


Documentation

Reference: <https://docs.getdbt.com/docs/building-a-dbt-project/documentation>

1. Add `00_analytics.yml` file under the analytics folder. Insert:

```
version: 2

models:
  - name: dim__customer
    columns:
      - name: customer_key
        description: 'Unique id of the customer'
      - name: customer_name
        description: 'Full name of the customer'
      - name: nation_name
        description: 'Home country of the customer referenced from the nation table'
      - name: sales_segment
        description: 'Sales segment provided by the data science team'
```

2. Run `dbt docs generate` in the command panel. Now you are able to *view docs*
3. Have a look at your beautiful documentation page!
 - You can see your project **structure**. But you can also check how it has been materialized **in your database**.
 - Look at the `dim__customer` model. Many useful information are available such as dependencies, code and other meta data.
 - But most importantly, you can find your crispy **column description** there.
4.  Exercise04: Document `fact__orders` table.

Testing

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

Built-in dbt tests

1. In your `00_analytics.yml` file add some additional configuration under the `dim__customer\customer_key` column. Like this:

```
models:
- name: dim__customer
  columns:
  - name: customer_key
    description: 'Unique id of the customer'
    tests:
      - unique
      - not_null
```

2. Run `dbt test -s dim__customer`. Check the results details.
3. Let's try the accepted vaules test. Change your `00_analytics.yml` file again but now the `dim__customer\sales_segment` column. Like this:

```
- name: sales_segment
  description: 'Sales segment provided by the data science team'
  tests:
  - accepted_values:
    values: ['LOW', 'MID', 'HIGH']
```

4. Run `dbt test -s dim__customer` again. 😊 Hmm,we have a warning
5. Look at your log details and debug the problem.
6. The 4th type of built-in dbt test is the referential integrity test. Let's test if all the `o_custkey` in the `orders` table are existing in the `customer` table. To do that change our `00_analytics.yml` on `fact__orders`:

```
- name: fact__orders
  columns:
  - name: order_key
    description: 'Unique id of the order'
  - name: customer_key
    description: 'Unique id of the customer'
  tests:
  - relationships:
    to: ref('psa_tpch_customer')
    field: c_custkey
```

7. Let's run `dbt test`

Custom singular test

8. Add new file `max_order_total_price.sql` into your `\tests` folder

```
select order_key from {{ref('fact__orders')}} where total_price > 10000
```

9. Check if our assertion is correct: `dbt test -s max_order_total_price`


Custom generic test

10. Create a new folder `tests\generic`
11. Add a new file `customer_key_format.sql`

```
{% test customer_key_format(model, column_name) %}

select *
from {{ model }}
where {{ column_name }}::numeric > 100

{% endtest %}
```

12.  Exercise05: We want to test the `customer_key` in our `dim__customer` and `fact__orders` models with our brand new `customer_key_format` test. How would you do that? (Hint: `not_null`, `unique` are generic test by default)

▸ Macros

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

1. Add a new file `cents_to_dollars.sql` into your `\macros` folder. Insert:

```
{% macro cents_to_dollars(column_name, decimal_places=2) -%}
    round( 1.0 * {{ column_name }} / 100, {{ decimal_places }})
{% endmacro %}
```

2. Change your `fact__orders` model by adding a new column using the macro.

```
select
    o_orderkey      as order_key
  ,o_custkey       as customer_key
  ,o_totalprice    as total_price
  ,{{ cents_to_dollars('o_totalprice', 4) }} as total_price_to_dollar
from {{ref('psa_tpch_orders')}}
```

3. Save all. Try the compile first.
4. Ready? Let's run `dbt run -s fact__orders`

[] ↪ 2 cells hidden