Creating dbt descriptions and tests

CASE STUDY: BUILDING E-COMMERCE DATA MODELS WITH DBT

Susan Sun
Freelance Data Scientist

docs: dbt user-defined descriptions

- User-defined descriptions in yaml files
- Used to document: dbt **models, sources, seeds, data tests**, etc.
- Naming convention:

```
_<data source>__<asset>.yml
```

- Store in the same directory as the assets

```
looker_ecommerce/
  macros/
  models/
    _looker__models.yml     <------
    _looker__sources.yml    <------
    stg_looker__distribution_center.sql
    stg_looker__orders.sql
  seeds/
    looker__distribution_center.csv
```

docs: dbt model yaml

Sample `_looker__models.yml`:

```yaml
version: 2

models:
  - name: model_name
    description: This is a table
    columns:
      - name: column_name
        description: This is a column
      - name: column_name
        description: This is a column
```

Note:

- `version: 2` is the schema configuration format used by dbt
- `models` defines what asset this user defined is documenting
- 2 spaces before table name
- 4 spaces before column name

dbt data tests: not null and unique

- Four default data tests live in yaml files:
  - unique
  - not_null
  - accepted_values
  - relationship

1. `unique` : each row value is unique

```
- name: table_name
  columns:
    - name: column_name
      data_tests:
        - not_null
```

2. `not_null` : no row can have a null value

```
- name: table_name
  columns:
    - name: column_name
      data_tests:
        - unique
        - not_null
    - name: column_name
      data_tests:
        - unique
```

dbt data tests: accepted values

3. `accepted_values` : only values in list are accepted

```
- name: table_name
  columns:
    - name: column_name
      data_tests:
        - accepted_values:
            values: ['value_a', 'value_b', 'value_c', NULL]
```

dbt data tests: relationships

4. `relationships` : referential integrity (foreign key) between tables

```
- name: table_1
  columns:
    - name: column_1
      data_tests:
        - relationships:
            to: ref('table_2')
            field: column_2
```

Let's practice!

CASE STUDY: BUILDING E-COMMERCE DATA MODELS WITH DBT

Building dbt data marts and snapshot models

CASE STUDY: BUILDING E-COMMERCE DATA MODELS WITH DBT

Susan Sun
Freelance Data Scientist

Introducing dbt data mart models

Data marts are clean and accessible data models at the end of the pipeline.

Use cases

- Feature stores for the data science team
- Aggregated KPIs for the finance team
- Latency metric for the engineering team
- Reduces repetition by storing the SQL as code

Looker e-commerce data marts

Customers

Answers the questions:

> Who are our customers?

> What are their purchase behaviors?

Data grain:

> One row per customer

Sample columns:

> Customer name, amount customers spent

Products

Answers the questions:

> What is our revenue, cost, and profit?

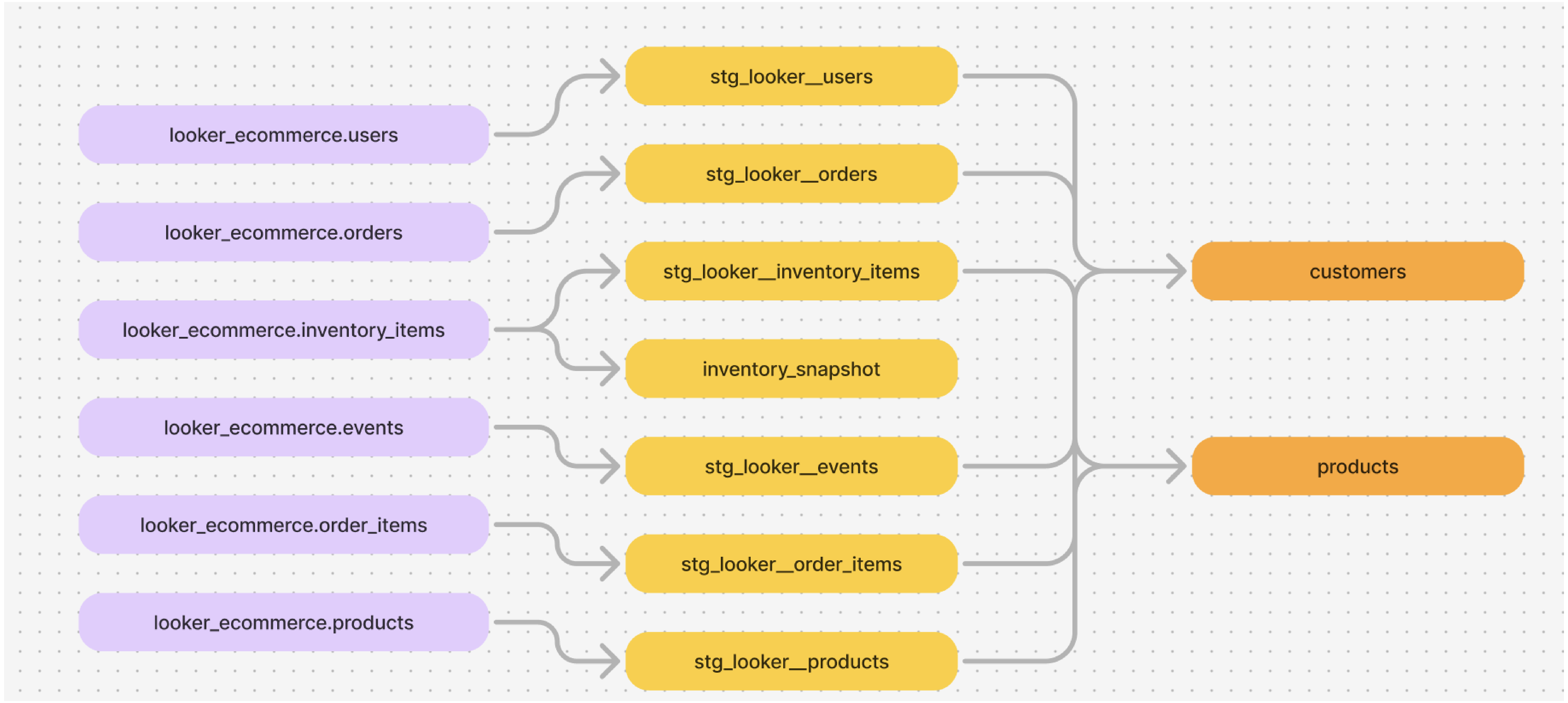
Data grain:

> One row per product

Sample columns:

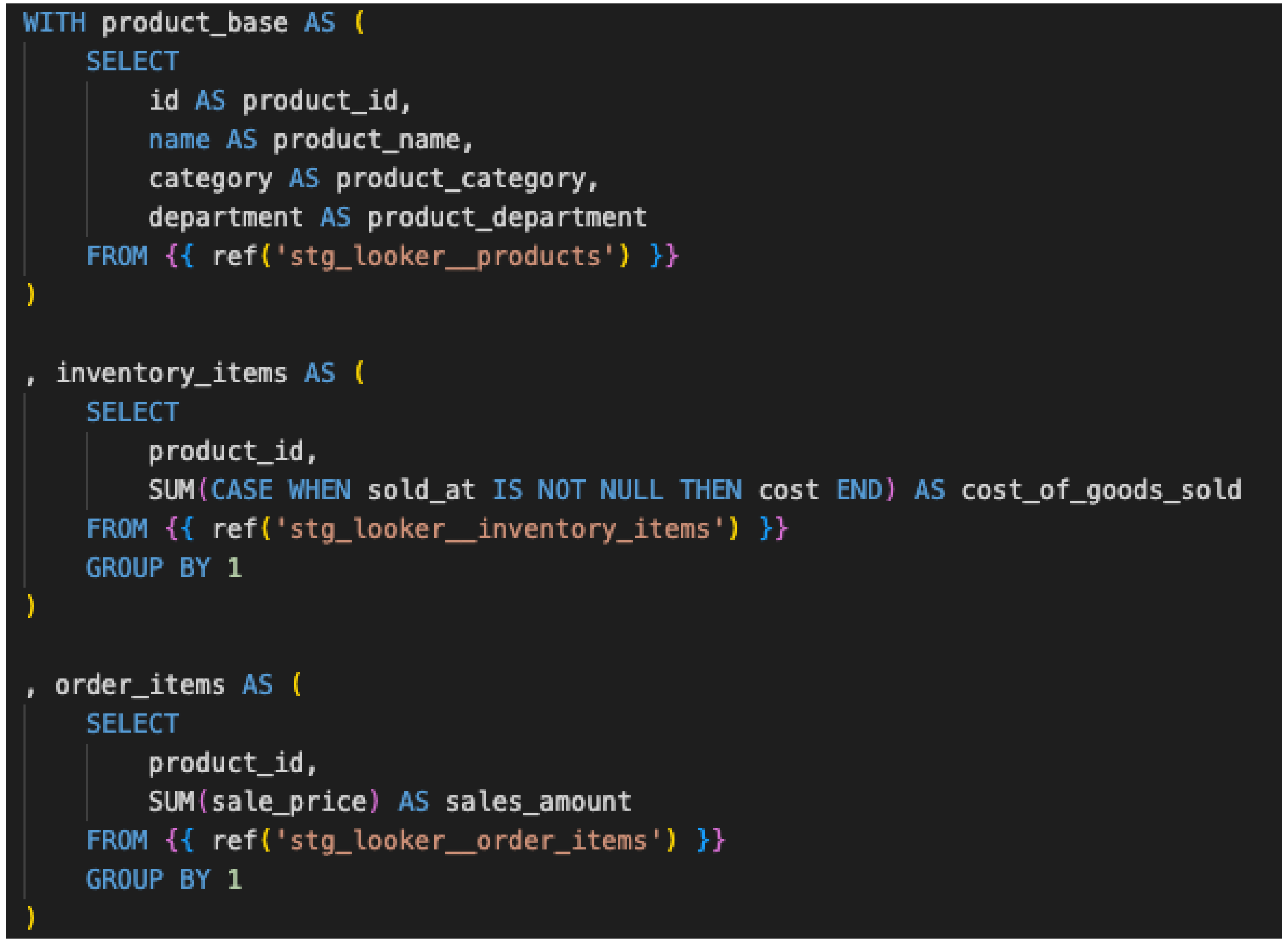
> Product category, revenue, profit

Looker e-commerce data marts



Building step-by-step

Step 1: Refine SQL logic outside of dbt, then replace references with dbt syntax.

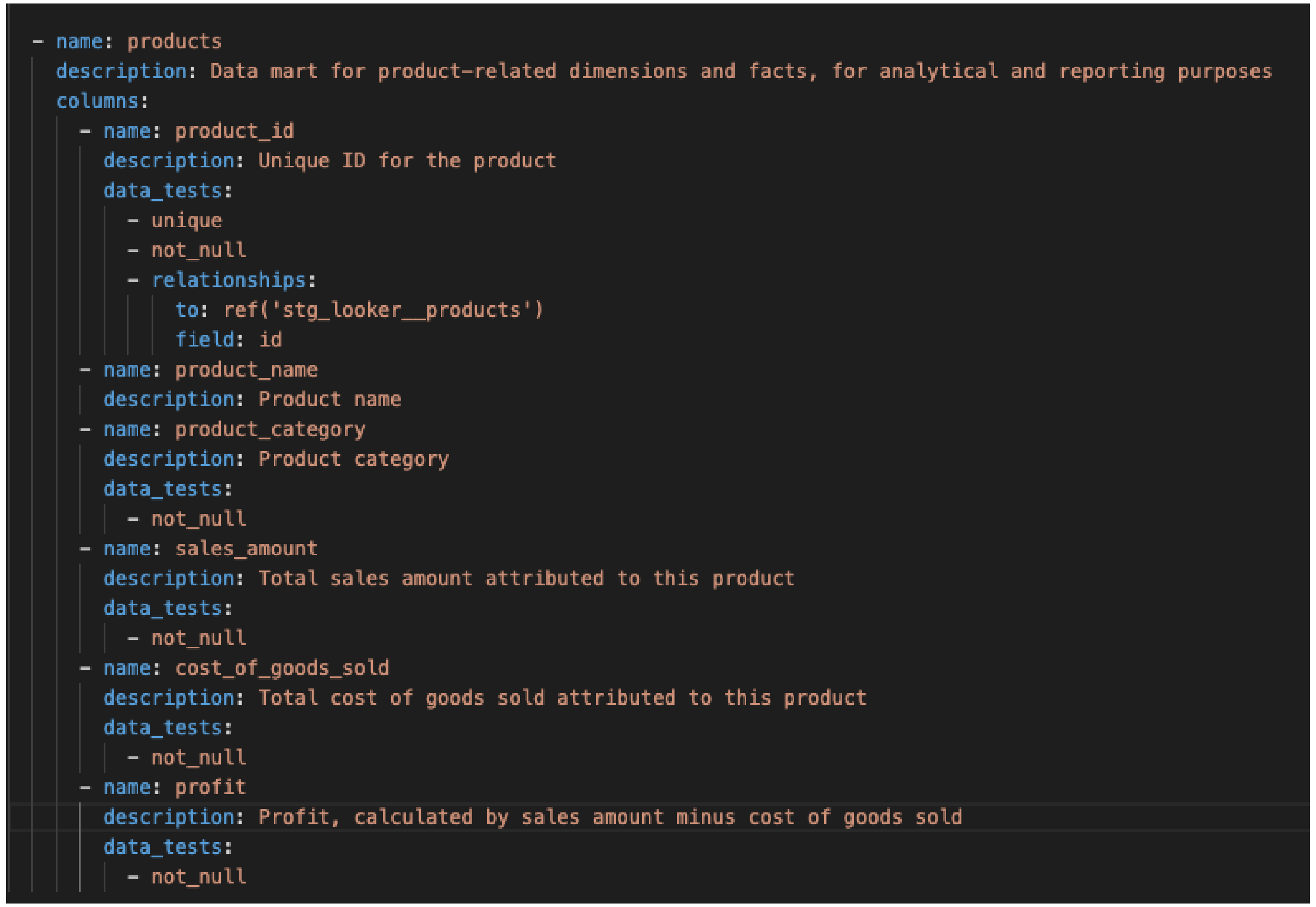


```sql
WITH product_base AS (
    SELECT
        id AS product_id,
        name AS product_name,
        category AS product_category,
        department AS product_department
    FROM {{ ref('stg_looker__products') }}
)

, inventory_items AS (
    SELECT
        product_id,
        SUM(CASE WHEN sold_at IS NOT NULL THEN cost END) AS cost_of_goods_sold
    FROM {{ ref('stg_looker__inventory_items') }}
    GROUP BY 1
)

, order_items AS (
    SELECT
        product_id,
        SUM(sale_price) AS sales_amount
    FROM {{ ref('stg_looker__order_items') }}
    GROUP BY 1
)
```

Step 2: Build out data tests and docs



```yaml
- name: products
  description: Data mart for product-related dimensions and facts, for analytical and reporting purposes
  columns:
    - name: product_id
      description: Unique ID for the product
      data_tests:
        - unique
        - not_null
        - relationships:
            to: ref('stg_looker__products')
            field: id
    - name: product_name
      description: Product name
    - name: product_category
      description: Product category
      data_tests:
        - not_null
    - name: sales_amount
      description: Total sales amount attributed to this product
      data_tests:
        - not_null
    - name: cost_of_goods_sold
      description: Total cost of goods sold attributed to this product
      data_tests:
        - not_null
    - name: profit
      description: Profit, calculated by sales amount minus cost of goods sold
      data_tests:
        - not_null
```

Step 3: Test the build! (e.g. `dbt build`)

Introducing dbt snapshot models

Five order status:

`Processing` , `Shipped` , `Complete` , `Cancelled` , and `Returned`

Data sample:

```
order_id,user_id,status,gender,created_at,returned_at,shipped_at,delivered_at,num_of_items

88616,70663,Returned,F,2024-01-15 18:12:00+00:00,2024-01-21 08:54:00+00:00,2024-01-18
05:01:00+00:00,2024-01-18 23:54:00+00:00,2

88641,70659,Returned,F,2021-03-18 17:45:00+00:00,2021-03-27 02:56:00+00:00,2021-03-21
17:08:00+00:00,2021-03-25 03:07:00+00:00,4


....
```

Orders: snapshot status change

- Create file `orders_snapshot.sql`

```
{% snapshot orders_snapshot %}
{{
    config(target_schema='main',
      unique_key='order_id',
      strategy='timestamp',
      updated_at='created_at')
}}

SELECT *
FROM
{{source('looker_ecommerce', 'orders')}}
```

[1] https://docs.getdbt.com/docs/build/snapshots

- Run a specific snapshot model

```
dbt snapshot -s orders_snapshot.sql
```

- Run all snapshot models

```
dbt snapshot
```

- Run all models (including snapshots)

```
dbt build
```

Let's practice!