**✅ dbt + Snowflake Setup: Full Flow (Secure & Reproducible)**

**🟦 1. Snowflake Admin Setup (run by ACCOUNTADMIN)**

-- Create compute and data objects

CREATE WAREHOUSE dbt\_wh;

CREATE DATABASE dbt\_db;

-- Create role and user

CREATE ROLE dbt\_role;

CREATE USER ani;

-- Grant necessary permissions

GRANT USAGE ON WAREHOUSE dbt\_wh TO ROLE dbt\_role;

GRANT ALL ON DATABASE dbt\_db TO ROLE dbt\_role;

GRANT ROLE dbt\_role TO USER ani;

-- (Optional) Set user password securely via Snowflake UI or SSO

**🟦 2. Create and Activate Python Virtual Environment**

python -m venv venv

.\venv\Scripts\activate # Windows

**🟦 3. Install dbt + Snowflake Adapter**

pip install dbt-core dbt-snowflake

**🟦 4. Initialize the Project (Auto-creates profiles.yml)**

dbt init dbt\_snowflake\_pipeline

You’ll be prompted for:

* Adapter: snowflake
* Account ID (e.g., abcd-xy123)
* User: ani
* Role: dbt\_role
* Warehouse: dbt\_wh
* Database: dbt\_db
* Schema: dbt\_schema

This writes a ~/.dbt/profiles.yml file with the details **you just entered**.

**🟦 5. Update dbt\_project.yml for Folder Structure**

In dbt\_snowflake\_pipeline/dbt\_project.yml:

name: 'dbt\_snowflake\_pipeline'

version: '1.0.0'

profile: 'dbt\_snowflake\_pipeline'

model-paths: ["models"]

analysis-paths: ["analyses"]

test-paths: ["tests"]

seed-paths: ["seeds"]

macro-paths: ["macros"]

snapshot-paths: ["snapshots"]

clean-targets:

- "target"

- "dbt\_packages"

models:

staging:

+materialized: view

snowflake\_warehouse: dbt\_wh

facts:

+materialized: table

snowflake\_warehouse: dbt\_wh

dimensions:

+materialized: table

snowflake\_warehouse: dbt\_wh

marts:

+materialized: table

snowflake\_warehouse: dbt\_wh

Perfect — you're now documenting the **6th step: the staging layer**, and you're doing it exactly how a production dbt project should.

Let me help you formalize and clean it up with:

**✅ Step 6: Staging Layer (Models + Source YAML)**

This layer prepares raw data from Snowflake’s SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF1 schema into clean, standardized fields used by downstream models.

**📄 models/staging/tpch\_staging.yaml**

version: 2

sources:

- name: tpch

database: SNOWFLAKE\_SAMPLE\_DATA

schema: tpch\_sf1

tables:

- name: orders

columns:

- name: o\_orderkey

tests:

- unique

- not\_null

- name: lineitem

columns:

- name: l\_orderkey

tests:

- relationships:

to: source('tpch', 'orders')

field: o\_orderkey

✅ These are **generic tests** to validate source integrity — retained as requested.

**📄 models/staging/stg\_tpch\_orders.sql**

{{ config(materialized='view') }}

select

o\_orderkey as order\_key,

o\_custkey as customer\_key,

o\_orderstatus as status\_code,

o\_orderdate as order\_date

from

{{ source('tpch', 'orders') }}

**📄 models/staging/stg\_tpch\_line\_items.sql**

{{ config(materialized='view') }}

select

{{ dbt\_utils.generate\_surrogate\_key(['l\_orderkey', 'l\_linenumber']) }} as order\_item\_key,

l\_orderkey as order\_key,

l\_partkey as part\_key,

l\_linenumber as line\_number,

l\_quantity as quantity,

l\_extendedprice as extended\_price,

l\_discount as discount\_percentage,

l\_tax as tax\_rate

from

{{ source('tpch', 'lineitem') }}

**✅ Step 7: Dimension Layer**

**📄 models/dimensions/dim\_line\_items.sql**

{{ config(materialized='table') }}

select

order\_item\_key,

part\_key,

line\_number

from

{{ ref('stg\_tpch\_line\_items') }}

**📄 models/dimensions/dim\_orders.sql**

{{ config(materialized='table') }}

select

order\_key,

customer\_key,

status\_code,

order\_date

from

{{ ref('stg\_tpch\_orders') }}

📌 These are **dimension tables**, materialized as full-refresh tables, used for joins and lookups in marts or facts.

**✅ Step 8: Fact Layer**

**📄 models/facts/fact\_orders.sql**

{{ config(materialized='table') }}

select

o.order\_key,

l.order\_item\_key,

l.quantity,

l.extended\_price,

l.discount\_percentage,

l.tax\_rate,

{{ discounted\_price('extended\_price', 'discount\_percentage') }} as discounted\_price,

{{ price\_after\_tax(discounted\_price('extended\_price', 'discount\_percentage'), 'tax\_rate') }} as price\_after\_tax

from

{{ ref('stg\_tpch\_orders') }} as o

inner join

{{ ref('stg\_tpch\_line\_items') }} as l

on l.order\_key = o.order\_key

📌 This is your **fact table** — it joins staging views, calculates prices using macros, and includes grain at the order item level.

**✅ Step 8.5: Macros Used**

**📄 macros/pricing\_macros.sql**

{% macro discounted\_price(extended\_price, discount) %}

({{ extended\_price }} \* (1 - {{ discount }}))

{% endmacro %}

{% macro price\_after\_tax(price, tax\_rate) %}

({{ price }} \* (1 + {{ tax\_rate }}))

{% endmacro %}

📌 These **reusable macros** encapsulate business logic and keep your fact models clean.

**✅ Step 9: Mart Layer**

**📄 models/marts/mart\_orders\_profit.sql**

{{ config(materialized='table') }}

select

order\_key,

sum(extended\_price) as actual\_order\_value,

sum(price\_after\_tax) as sale\_order\_value,

sum(price\_after\_tax) - sum(extended\_price) as profit

from

{{ ref('fact\_orders') }}

group by order\_key

📌 This is your **reporting mart**, summarizing key business metrics by order\_key:

* Actual value before discount/tax
* Sale value after tax
* Profit (value added due to pricing)

**✅ Step 10: Testing + Core Concepts in dbt**

**🧪 Difference Between Generic and Singular Tests**

**🔷 Generic Tests (Reusable, YAML-based)**

Defined in schema.yml, used for **column-level validations**. These are pre-built or custom reusable macros.

**📄 Example (Generic test in dimensions/schema.yml):**

models:

- name: dim\_orders

columns:

- name: order\_key

tests:

- not\_null

- unique

- name: dim\_line\_items

columns:

- name: order\_item\_key

tests:

- not\_null

- unique

**✅ Behavior:**

* Fails if null/duplicate values are found
* Easy to apply across many columns

**🔶 Singular Tests (Custom SQL)**

Defined as **.sql files inside tests/**, these are full SQL queries that **fail if they return any rows**.

**📄 Example (tests/test\_discount\_percentage\_limit.sql):**

select \*

from

{{ ref('fact\_orders') }}

where

discount\_percentage > 1

**✅ Behavior:**

* Fails if any row violates business logic
* Flexible for complex rules (e.g., date ranges, joins, aggregations)

**🛠️ What is ref() in dbt?**

{{ ref('model\_name') }}

* ref() tells dbt to:
  + Track **dependencies** between models
  + Determine **build order**
  + Replace with the actual **schema.table** name in SQL

**Example:**

select \* from {{ ref('stg\_tpch\_orders') }}

✅ Ensures that if stg\_tpch\_orders changes, dbt automatically rebuilds dependent models like fact\_orders.

**🔌 What is dbt\_utils?**

dbt\_utils is a popular **community package** providing reusable macros for:

* Surrogate keys
* Safe cast
* Union, Pivot, Fuzzy matching
* Expression-based tests

**🔧 Install it by adding to packages.yml:**

packages:

- package: dbt-labs/dbt\_utils

version: 1.1.1

Then run:

dbt deps

✅ Now you can use:

{{ dbt\_utils.generate\_surrogate\_key(['col1', 'col2']) }}

**📦 Why Define source for Staging?**

Defining source() in tpch\_staging.yaml tells dbt:

* What **external raw tables** you're pulling from (e.g., tpch.orders)
* To treat them as **source-of-truth inputs**
* To allow **testing, documentation, and lineage tracking** from staging models

**📄 Example:**

sources:

- name: tpch

database: SNOWFLAKE\_SAMPLE\_DATA

schema: tpch\_sf1

tables:

- name: orders

columns:

- name: o\_orderkey

tests: [not\_null, unique]

Used in SQL as:

{{ source('tpch', 'orders') }}

✅ Improves **data lineage**, **testing**, and **documentation** for source inputs.

**📌 Summary Table**

| **Concept** | **Purpose** | **How to Use** |
| --- | --- | --- |
| ref() | Model dependency & table name resolution | {{ ref('model\_name') }} |
| source() | Reference external tables + enable testing | {{ source('source\_name', 'table') }} |
| dbt\_utils | Reusable macros + generic test tools | generate\_surrogate\_key, etc. |
| Generic Test | YAML-based, reusable tests (e.g., not\_null) | In schema.yml |
| Singular Test | Custom logic in .sql files (returns rows) | In tests/ folder |