**DBT project creation  
In this project,**

* **The dbt project name is dbt-etl-adw (defined in dbt\_project.yml).**
* **The unique profile for this project (e.g., dbt\_etl\_adw) defined in profiles.yml.**

**What are these:**

1. **dbt\_project.yml: This file defines your dbt project settings, including its name, where models are located, and which profile the project will use. The name dbt\_project.yml is hardcoded in dbt as the main configuration file for every project created.**
2. **Profile: Profiles are defined in the profiles.yml file (usually in ~/.dbt/profiles.yml). They specify the connection to your database (e.g., hostname, username, schema). Each project must reference a unique profile to avoid overlap.**

**Steps to Create a New Isolated dbt Project**

**1. Use a Different Folder for the New Project**

When creating the new project, ensure you use a separate directory (you've already specified C:\Users\Shai\Documents\GitHub\aws-1st-DE-Project), which is fine. Just make sure the new project directory does not overlap with your existing one.

cmd

CopyEdit

cd C:\Users\Shai\Documents\GitHub\aws-1st-DE-Project

dbt init dbt-etl-adw

This will create a **new folder** dbt-etl-adw inside C:\Users\Shai\Documents\GitHub\aws-1st-DE-Project.

**2. Use a Separate Profile in profiles.yml**

dbt projects use the **profiles.yml** file to define database connections. To isolate the new project:

1. **Open profiles.yml**:
   * The file is typically located in ~/.dbt/profiles.yml (Windows equivalent: %USERPROFILE%\.dbt\profiles.yml).
2. **Add a New Profile for the New Project**:
   * Create a unique profile name (e.g., dbt\_etl\_adw).
   * Ensure the dbt\_project.yml file in your new project uses this unique profile.

Example of profiles.yml:

yaml

CopyEdit

dbt\_etl\_adw:

outputs:

dev:

type: postgres

host: <your-db-host>

user: <your-db-user>

password: <your-db-password>

port: 5432

dbname: <your-database-name>

schema: <your-schema>

target: dev

1. **Link the Profile in the New Project**:
   * In the dbt\_project.yml file of the new project, update the profile key to the unique profile name:

yaml

CopyEdit

name: dbt-etl-adw

profile: dbt\_etl\_adw

**3. Activate a Separate Virtual Environment**

If your existing project uses a virtual environment, create a **new virtual environment** for the new project to avoid dependency conflicts:

cmd

CopyEdit

cd C:\Users\Shai\Documents\GitHub\aws-1st-DE-Project\dbt-etl-adw

python -m venv venv

.\venv\Scripts\activate

pip install dbt-postgres # Or the adapter you're using

**4. Use Separate Database Schemas**

When configuring the new profile, ensure that it points to a different schema or database from the existing project. For example, in profiles.yml:

yaml

CopyEdit

schema: dbt\_etl\_adw # A unique schema for the new project

**5. Validate Project Independence**

To confirm the new project does not interfere with the existing one:

* Run dbt debug in the new project directory and ensure it uses the correct profile.
* Verify that models created in the new project write to the specified schema/database without affecting the existing project's schema.

**1. Understand Your Data**

* **SalesHeader**: Likely contains overall sales information, such as OrderID, CustomerID, OrderDate, TotalAmount, etc.
* **SalesOrderDetail**: Likely contains detailed line items for each order, such as OrderDetailID, OrderID, ProductID, Quantity, UnitPrice, etc.

**2. Define the Project Scope**

Examples of ETL goals for these sources:

1. **Data Cleaning**:
   * Ensure all columns are properly formatted (e.g., dates, numeric values).
   * Remove invalid records (e.g., missing CustomerID or ProductID).
2. **Transformations**:
   * Aggregate sales data (e.g., total sales per customer, monthly sales trends).
   * Calculate derived metrics (e.g., total revenue, average order value).
3. **Modeling**:
   * Create **fact tables** (e.g., fact\_sales).
   * Create **dimension tables** (e.g., dim\_customers, dim\_products).

**3. Setup dbt Sources**

**Define Your Sources:**

1. Open your dbt\_project.yml file and set up a folder structure:

yaml

CopyEdit

name: sales\_etl\_project

profile: dbt\_etl\_adw

model-paths: ["models"]

1. Create a models/sources.yml file:

yaml

CopyEdit

version: 2

sources:

- name: adw

tables:

- name: sales\_header

- name: sales\_order\_detail

1. Test the sources:

yaml

CopyEdit

version: 2

tables:

- name: sales\_header

description: "Header data for sales orders."

columns:

- name: order\_id

tests:

- unique

- not\_null

- name: customer\_id

tests:

- not\_null

- name: sales\_order\_detail

description: "Detailed line items for sales orders."

columns:

- name: order\_id

tests:

- not\_null

- name: product\_id

tests:

- not\_null

**4. Transformations**

**Step 1: Staging Models**

Create models for cleaning and preparing the raw data. Place these in models/staging/.

1. **Staging SalesHeader (stg\_sales\_header.sql)**:

sql

CopyEdit

with sales\_header as (

select

order\_id,

customer\_id,

order\_date,

total\_amount

from {{ source('adw', 'sales\_header') }}

)

select \* from sales\_header;

1. **Staging SalesOrderDetail (stg\_sales\_order\_detail.sql)**:

sql

CopyEdit

with sales\_order\_detail as (

select

order\_detail\_id,

order\_id,

product\_id,

quantity,

unit\_price,

quantity \* unit\_price as total\_price

from {{ source('adw', 'sales\_order\_detail') }}

)

select \* from sales\_order\_detail;

**Step 2: Intermediate Models**

Create models for combining and enriching the data. Place these in models/transform/.

1. **Combine Header and Details (trans\_sales\_orders.sql)**:

sql

CopyEdit

with combined as (

select

sh.order\_id,

sh.customer\_id,

sh.order\_date,

sh.total\_amount,

sd.product\_id,

sd.quantity,

sd.unit\_price,

sd.total\_price

from {{ ref('stg\_sales\_header') }} sh

join {{ ref('stg\_sales\_order\_detail') }} sd

on sh.order\_id = sd.order\_id

)

select \* from combined;

1. **Aggregate Data (agg\_sales\_summary.sql)**:
   * Monthly Sales Summary:

sql

CopyEdit

with monthly\_sales as (

select

date\_trunc('month', order\_date) as month,

sum(total\_price) as total\_revenue,

count(distinct order\_id) as total\_orders

from {{ ref('trans\_sales\_orders') }}

group by 1

)

select \* from monthly\_sales;

**Step 3: Fact and Dimension Models**

1. **Fact Table (fact\_sales.sql)**:

sql

CopyEdit

with fact\_sales as (

select

order\_id,

customer\_id,

sum(total\_price) as order\_total

from {{ ref('trans\_sales\_orders') }}

group by order\_id, customer\_id

)

select \* from fact\_sales;

1. **Dimension Tables**:
   * **Products (dim\_products.sql)**:

sql

CopyEdit

select distinct product\_id

from {{ ref('stg\_sales\_order\_detail') }};

**5. Next Steps**

1. **Test Your Models**:
   * Add tests for critical columns to ensure data integrity (e.g., not\_null, unique).

yaml

CopyEdit

version: 2

models:

- name: trans\_sales\_orders

columns:

- name: order\_id

tests:

- not\_null

- unique

1. **Run dbt**:
   * Compile and test your models:

cmd

CopyEdit

dbt run

dbt test

1. **Validate Output**:
   * Ensure the transformed tables have correct aggregations and calculations.
2. **Document the Project**:
   * Use dbt's auto-documentation feature to generate docs for your models and sources:

cmd

CopyEdit

dbt docs generate

dbt docs serve