# Setting Up My dbt Project: dbt-etl-adw

# 1. Creating a New Isolated dbt Project

**Initializing the Project**

To create a new dbt project, I start by choosing a separate folder for it. In my case, I’m placing it inside my existing directory:

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

Then, I initialize the dbt project:

dbt init dbt-etl-adw

This creates a new folder named dbt-etl-adw within my project directory.

# 2. Configuring a Separate Profile

Each dbt project requires a unique profile to define database connections. To isolate this new project, I update my profiles.yml file, which is typically found in:

* Windows: %USERPROFILE%\.dbt\profiles.yml

I add a new profile for my project:

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

Then, I make sure my dbt\_project.yml file in the new project correctly references this profile:

profile: dbt\_etl\_adw

If I encounter issues, I check whether an existing dbt project is interfering and remove any old references.

# 3. Setting Up a Virtual Environment

To avoid dependency conflicts, I create a separate virtual environment for this project:

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

python -m venv venv

I activate it:

.\venv\Scripts\activate

Then, I install the required dbt adapter:

pip install dbt-postgres # Or the adapter I'm using

# 4. Using Separate Database Schemas (not followed)

To ensure my dbt project remains isolated, I configure my profile to use a unique schema:

schema: dbt\_etl\_adw

# 5. Validating the Project Setup

To confirm my project is correctly configured, I run:

dbt debug

This verifies that my project is using the correct profile and schema.

# Understanding My Data

I’m working with two key tables:

* **SalesHeader**: Contains overall order details (OrderID, CustomerID, OrderDate, TotalAmount).
* **SalesOrderDetail**: Contains line item details (OrderDetailID, OrderID, ProductID, Quantity, UnitPrice).

**Defining My ETL Goals**

1. **Data Cleaning**
   * Ensure proper formatting for dates and numeric values.
   * Remove invalid records (e.g., missing CustomerID or ProductID).
2. **Transformations**
   * Aggregate sales data (e.g., total sales per customer, monthly trends).
   * Calculate derived metrics (e.g., total revenue, average order value).
3. **Modeling**
   * Create fact tables (fact\_sales).

# 6. Setting Up dbt Sources

**Defining Sources**

I configure my sources in models/sources.yml:

version: 2

sources:

- name: adw

tables:

- name: sales\_header

- name: sales\_order\_detail

I add tests for these tables:

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

# 7. Creating Transformations

**Step 1: Staging Models**

I create staging models in models/staging/:

-- models/staging/stg\_sales\_header.sql

with sales\_header as (

select

order\_id,

customer\_id,

order\_date,

total\_amount

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

)

select \* from sales\_header;

-- models/staging/stg\_sales\_order\_detail.sql

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**

-- models/transform/trans\_sales\_orders.sql

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;

-- models/transform/agg\_sales\_summary.sql

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**

-- models/marts/fact\_sales.sql

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;

-- models/marts/dim\_products.sql

select distinct product\_id

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

# 8. Running and Testing dbt

I ensure my models are working correctly:

dbt run

dbt test

I also document my project:

dbt docs generate

dbt docs serve

# Final Notes

Always make sure your column names and syntax align with your chosen RDS to avoid errors when running your models