**Lab Exercise 5 -Jinja Template with for loop in dbt and Snowflake**

Here’s a more **complex lab exercise** that uses a Jinja **loop** in dbt to dynamically build columns or filters in a model. This example will show how to use a Jinja for loop to generate a SQL statement that aggregates data across multiple columns.

**Exercise Overview**

In this exercise, we’ll:

1. Define multiple columns to aggregate.
2. Use a Jinja loop to dynamically generate SQL for each column.
3. Apply the loop to produce a complex aggregation table.

**Scenario**

Assume you have a sales table with monthly sales data in Snowflake. You want to calculate the total and average sales for each month and generate these calculations dynamically using a Jinja loop in dbt.

**Step 1: Set Up the Source Table**

If not already available, create a sales table in Snowflake with columns for store\_id, sales\_month, and monthly sales columns (sales\_jan, sales\_feb, ..., sales\_dec).

CREATE OR REPLACE TABLE raw\_db.raw\_data.sales (

store\_id INT,

sales\_month STRING,

sales\_jan FLOAT,

sales\_feb FLOAT,

sales\_mar FLOAT,

sales\_apr FLOAT,

sales\_may FLOAT,

sales\_jun FLOAT,

sales\_jul FLOAT,

sales\_aug FLOAT,

sales\_sep FLOAT,

sales\_oct FLOAT,

sales\_nov FLOAT,

sales\_dec FLOAT

);

INSERT INTO raw\_db.raw\_data.sales VALUES

(1, '2024-01', 1000, 1200, 1100, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000, 2100),

(2, '2024-01', 4000, 1500, 1100, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000, 2100),

(1, '2024-01', 5000, 1200, 1100, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000, 2100),

(2, '2024-01', 800, 1200, 1100, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000, 2100),

(1, '2024-01', 7000, 1200, 1100, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000, 2100),

(2, '2024-01', 900, 1100, 1050, 1250, 1350, 1450, 1550, 1650, 1750, 1850, 1950, 2050);

**Step 2: Create the Model File with Jinja Loop**

Let’s create a model file called ***monthly\_sales\_summary.sql*** in the models directory. In this file:

1. Define a list of columns (one for each month).
2. Use a Jinja for loop to dynamically generate SQL statements that calculate the total and average sales for each month.

-- models/monthly\_sales\_summary.sql

{{ config(

materialized = 'table'

) }}

{% set months = [

'jan', 'feb', 'mar', 'apr', 'may', 'jun',

'jul', 'aug', 'sep', 'oct', 'nov', 'dec'

] %}

WITH base AS (

SELECT

store\_id,

sales\_month,

-- Use a Jinja loop to dynamically calculate total and average sales for each month

{% for month in months %}

SUM(sales\_{{ month }}) AS total\_sales\_{{ month }},

AVG(sales\_{{ month }}) AS avg\_sales\_{{ month }}

{% if not loop.last %}, {% endif %}

{% endfor %}

FROM raw\_db.raw\_data.sales -- Reference the source table directly

GROUP BY store\_id, sales\_month

)

SELECT

store\_id,

sales\_month,

{% for month in months %}

total\_sales\_{{ month }},

avg\_sales\_{{ month }}

{% if not loop.last %}, {% endif %}

{% endfor %}

FROM base

**Explanation of the Code**

* {% set months = [...] %}: This line creates a list of month abbreviations (jan, feb, mar, etc.) to loop through.
* {% for month in months %}: This Jinja loop iterates through each month abbreviation in the months list.
* SUM(sales\_{{ month }}) AS total\_sales\_{{ month }}: Within the loop, each month is dynamically inserted to create SUM and AVG calculations for each sales\_month column.
* GROUP BY store\_id, sales\_month: Groups the data by store\_id and sales\_month, calculating aggregated values per month.

The loop dynamically builds SQL statements for each month’s total and average sales, avoiding the need to write each line individually.

**Step 3: Run the Model in dbt Cloud**

1. **Run the Model**: In dbt Cloud, go to **Deploy** -> **Run** or, if running dbt locally, use:

dbt run --select monthly\_sales\_summary

1. **Check Results**: Look at the monthly\_sales\_summary table in Snowflake. The resulting table will contain columns like total\_sales\_jan, avg\_sales\_jan, total\_sales\_feb, avg\_sales\_feb, etc., with total and average sales for each month.

**Resulting Table Structure**

Your resulting table should look like this:

| **store\_id** | **sales\_month** | **total\_sales\_jan** | **avg\_sales\_jan** | **total\_sales\_feb** | **avg\_sales\_feb** | **...** | **total\_sales\_dec** | **avg\_sales\_dec** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2024-01 | 1000 | 1000 | 1200 | 1200 | ... | 2100 | 2100 |
| 2 | 2024-01 | 900 | 900 | 1100 | 1100 | ... | 2050 | 2050 |

**Summary**

In this exercise, you:

* Defined a list of column names (months) for looping.
* Used a Jinja for loop to dynamically build SQL statements for each month.
* Generated total and average sales calculations for each month, making the SQL logic more flexible and scalable.

This technique is useful in scenarios where you need to apply the same calculations to multiple columns, improving readability and maintainability in your dbt models.