**Lab Exercise 8– Using CTEs vs Ephemeral Models in dbt**

**Objective:**

This lab will help you understand the differences between **CTEs** and **ephemeral models** in dbt. You’ll create two versions of a dbt model, one using CTEs and one using ephemeral models, and observe the differences in performance and usage.

**Prerequisites**

* You have a dbt project set up and connected to a data warehouse (e.g., Snowflake).
* You have basic familiarity with dbt model structure and the ref function.

**Scenario**

You’re tasked with creating a simple report on customer order metrics for a fictional jaffel\_shop schema. You’ll calculate metrics like total\_order\_amount and order\_count for each customer using both **CTEs** and **ephemeral models**.

**Step 1: Set Up Source Tables in Your Data Warehouse**

Before we start with dbt, create source tables in your data warehouse.

**SQL to Create Source Tables**

Run these commands in your data warehouse to create the necessary tables and insert sample data.

-- Create the customers table

CREATE OR REPLACE TABLE jaffle\_shop.hks\_customers (

customer\_id INT PRIMARY KEY,

customer\_name STRING,

email STRING,

created\_at TIMESTAMP

);

-- Insert sample data into the customers table

INSERT INTO jaffle\_shop.hks\_customers (customer\_id, customer\_name, email, created\_at)

VALUES

(1, 'Alice Johnson', 'alice@example.com', '2021-05-01 10:00:00'),

(2, 'Bob Smith', 'bob@example.com', '2021-06-15 14:30:00'),

(3, 'Charlie Brown', 'charlie@example.com', '2021-07-20 09:45:00');

-- Create the orders table

CREATE OR REPLACE TABLE jaffle\_shop.hks\_orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

amount DECIMAL(10, 2),

order\_date TIMESTAMP,

status STRING

);

-- Insert sample data into the orders table

INSERT INTO jaffle\_shop.hks\_orders (order\_id, customer\_id, amount, order\_date, status)

VALUES

(101, 1, 250.00, '2021-08-05 15:00:00', 'completed'),

(102, 2, 150.00, '2021-08-06 12:00:00', 'completed'),

(103, 3, 200.00, '2021-08-07 18:00:00', 'pending'),

(104, 1, 300.00, '2021-08-08 10:00:00', 'completed');

**Step 2: Model Using CTEs**

In this step, you’ll create a model that uses CTEs to compute customer order metrics.

**File Location: models/fct\_customer\_order\_metrics\_cte.sql**

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

WITH customer\_orders AS (

    SELECT

        customer\_id,

        COUNT(order\_id) AS order\_count,

        SUM(amount) AS total\_order\_amount

    FROM raw.jaffle\_shop.hks\_orders

    GROUP BY customer\_id

),

customers AS (

    SELECT

        customer\_id,

        customer\_name,

        email

    FROM raw.jaffle\_shop.hks\_customers

)

SELECT

    c.customer\_id,

    c.customer\_name,

    c.email,

    co.order\_count,

    co.total\_order\_amount

FROM customers AS c

LEFT JOIN customer\_orders AS co

ON c.customer\_id = co.customer\_id

**Explanation:**

* **CTEs**: We use customer\_orders and customers CTEs to calculate the order metrics and retrieve customer details.
* **Materialization**: The model is materialized as a table.

**Step 3: Model Using Ephemeral Models**

Now, let’s create a similar model using ephemeral models instead of CTEs.

1. **Create Ephemeral Models for Customer Orders and Customer Details**

**File Location: models/stg\_customer\_orders.sql**

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

SELECT

customer\_id,

COUNT(order\_id) AS order\_count,

SUM(amount) AS total\_order\_amount

FROM raw.jaffle\_shop.hks\_orders

GROUP BY customer\_id

**File Location: models/stg\_customers.sql**

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

SELECT

customer\_id,

customer\_name,

email

FROM raw.jaffle\_shop.hks\_customers

1. **Use Ephemeral Models in the Final Model**

**File Location: models/ fct\_customer\_order\_metrics\_ephemeral.sql**

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

SELECT

c.customer\_id,

c.customer\_name,

c.email,

co.order\_count,

co.total\_order\_amount

FROM {{ ref('stg\_customers') }} AS c

LEFT JOIN {{ ref('stg\_customer\_orders') }} AS co

ON c.customer\_id = co.customer\_id

**Explanation:**

* **Ephemeral Models**: The stg\_customer\_orders and stg\_customers models are ephemeral, so dbt will inline them as subqueries when fct\_customer\_order\_metrics\_ephemeral.sql runs.
* **Reusable Transformations**: These ephemeral models can be referenced in other models as well.

**Step 5: Run and Test the Models**

1. **Run Both Models**:

dbt run

This will:

* + Execute both the CTE-based model and the ephemeral model.
  + Create the final tables for fct\_customer\_order\_metrics\_cte and fct\_customer\_order\_metrics\_ephemeral in the data warehouse.

1. **Compare Query Plans** (optional):
   * You can inspect the query plans for each model to observe any performance differences, especially with larger datasets.

**Step 6: Review Results**

1. **Query Results**:
   * Check that both models (fct\_customer\_order\_metrics\_cte and fct\_customer\_order\_metrics\_ephemeral) produce the same results in terms of output.
2. **Performance and Maintainability**:
   * **CTEs**: Suitable for isolated, single-use transformations in one model. Easier to troubleshoot within a single model.
   * **Ephemeral Models**: Ideal for reusable transformations across multiple models but can create nested subqueries that may impact performance if overused.

**Summary**

In this lab exercise, you explored:

1. Creating models using both **CTEs** and **ephemeral models** in dbt.
2. Understanding when to use each approach based on reusability and performance.
3. Observing how ephemeral models are inlined as subqueries while CTEs are specific to a single model.

This exercise highlights the trade-offs between using CTEs and ephemeral models for complex transformations in dbt.