**CTEs vs ephemeral models**

In dbt, **CTEs (Common Table Expressions)** and **ephemeral models** are two approaches for creating intermediate data transformations within a SQL query. They serve similar purposes but have different implementations and use cases. Here’s a comparison of each approach and when to use them.

**1. CTEs in dbt**

**CTEs** are temporary result sets within a SQL query, defined using the WITH keyword. In dbt, CTEs are typically used within models to structure complex transformations into logical, named steps.

**Example of a CTE**

WITH customer\_orders AS (

SELECT

customer\_id,

SUM(amount) AS total\_amount,

COUNT(order\_id) AS order\_count

FROM {{ ref('orders') }}

GROUP BY customer\_id

)

SELECT \* FROM customer\_orders

**Characteristics:**

* **Scope**: CTEs exist only within a single SQL query or dbt model. They are temporary by design and do not persist in the database.
* **Performance**: They are re-executed each time the model runs, which may not be optimal for large datasets or highly complex queries.
* **Usage**: Use CTEs when you need intermediate steps within a model but do not need to reuse them across other models.

**2. Ephemeral Models in dbt**

**Ephemeral models** are a special type of dbt model with the materialized configuration set to ephemeral. dbt inlines the SQL from ephemeral models directly into the parent model’s SQL code as a subquery.

**Example of an Ephemeral Model**

Define an ephemeral model named stg\_customer\_totals.sql:

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

SELECT

customer\_id,

SUM(amount) AS total\_amount,

COUNT(order\_id) AS order\_count

FROM {{ ref('orders') }}

GROUP BY customer\_id

Then, reference it in another model:

SELECT \*

FROM {{ ref('stg\_customer\_totals') }}

**Characteristics:**

* **Scope**: Ephemeral models do not create tables or views in the database. Instead, their SQL is included as a subquery in any model that references them.
* **Performance**: They reduce the need for additional database objects (tables/views), keeping the database cleaner and simpler. However, they may increase query complexity if inlined into multiple parent models, especially with nested queries.
* **Usage**: Ideal for reusable transformations that you want to use in multiple models without creating tables or views in the database.

**Key Differences Between CTEs and Ephemeral Models**

| **Aspect** | **CTEs** | **Ephemeral Models** |
| --- | --- | --- |
| **Definition** | Defined within a single SQL query | Defined as standalone models in dbt |
| **Persistence** | Temporary within a query | Inlined as subqueries, not persistent |
| **Reusability** | Limited to the current model | Can be referenced across multiple models |
| **Performance Impact** | May affect performance if re-used | Reduces database objects, can impact query performance if over-nested |
| **Best Use Cases** | Single-model transformations | Reusable transformations across models |

**When to Use Each Approach**

* **Use CTEs** when:
  + You only need the transformations within a single model.
  + The transformations are specific to a single SQL query or model.
  + You want to keep all transformations in one place for readability.
* **Use Ephemeral Models** when:
  + You need to share the same transformation logic across multiple models.
  + You want to avoid cluttering the database with intermediate tables/views.
  + You prefer dbt to manage inlining transformations, especially if your transformations are reusable across multiple queries.

**Example Scenario**

Suppose you’re building a customer analytics pipeline with multiple stages:

1. **CTE**: Use a CTE if you're only calculating a summary metric (e.g., total sales by customer) within a single model that won’t be reused elsewhere.
2. **Ephemeral Model**: If you calculate customer-level metrics (e.g., order count and average order value) that need to be accessed across multiple downstream models, an ephemeral model is more efficient, allowing you to keep transformations consistent without persisting tables.

Both CTEs and ephemeral models offer ways to organize and structure complex queries in dbt, and choosing between them depends on the requirements for reusability, performance, and database simplicity.