**CTEs vs Subqueries**

**CTEs (WITH Clauses)**

A **CTE** is a named temporary result set, defined at the start of a query, which you can reference multiple times in the main query. CTEs can improve readability by organizing complex logic into named sections.

**Syntax**:

WITH cte\_name AS (

SELECT ...

)

SELECT ...

FROM cte\_name

**Advantages of CTEs**:

* **Readability**: CTEs make the SQL easier to read and maintain, especially for complex transformations.
* **Reusability**: You can reuse the CTE multiple times within the same query, reducing redundant code.
* **Modularity**: dbt relies on modular transformations. CTEs allow for breaking down complex transformations in a way that aligns with dbt's structure.

**Disadvantages of CTEs in Snowflake**:

* **Materialization and Performance**: Snowflake’s execution engine might re-evaluate CTEs each time they’re referenced in the query, which can slow performance in some cases. Snowflake treats CTEs as inline views, so they are not always optimized like temporary tables.

**Subqueries**

A **subquery** is a nested query within a larger query, typically in the SELECT, FROM, or WHERE clauses. Unlike CTEs, subqueries don’t have named references but execute in line with the main query.

**Syntax**:

SELECT ...

FROM (

SELECT ...

FROM ...

) AS subquery\_name

**Advantages of Subqueries**:

* **In-place Execution**: Subqueries are often evaluated immediately within their place in the main query, which can sometimes lead to performance gains compared to multiple references of a CTE.
* **Simplicity for Small Transformations**: Subqueries are useful for simpler queries that don’t require reuse of the result.

**Disadvantages of Subqueries**:

* **Readability**: Nested subqueries can make SQL hard to read, especially if there are multiple levels of nesting.
* **Reusability**: Subqueries can’t be referenced multiple times. You would need to rewrite the same query if it’s used more than once, which can lead to redundancy.