**Incremental Materialization in dbt for Snowflake**

Incremental materialization is used to load only new or modified data into your target table in dbt. It is highly useful for large datasets as it minimizes the data processing time by only loading the necessary changes. In this context, I'll show you how to implement incremental materialization in dbt for a Snowflake model.

**Steps for Incremental Materialization in dbt for Snowflake:**

1. **Create a Source Table**: Make sure you have a source table that you can incrementally process.
2. **Define the Incremental Model**: Create a model that will be loaded incrementally.
3. **Write the Incremental Logic**: Use the is\_incremental() macro to define the incremental behavior.
4. **Use unique\_key**: Specify the unique key that will allow dbt to identify changes in the source data.

Here’s a step-by-step guide with an example.

**Step 1: Create a Source Table in Snowflake**

Assume you have a source table in Snowflake called raw.orders which contains order data. Here’s how you would define it in Snowflake:

CREATE OR REPLACE TABLE raw.orders (

order\_id INT,

order\_date DATE,

customer\_id INT,

total\_amount DECIMAL(10,2),

created\_at TIMESTAMP,

updated\_at TIMESTAMP

);

This orders table contains order details such as order\_id, order\_date, customer\_id, total\_amount, and created\_at/updated\_at timestamps.

**Step 2: Define the Incremental Model in dbt**

In dbt, you'll define a model that performs incremental loading of this data. Let's assume you want to load only the records that have been updated or added after the last dbt run.

Create a new model file models/incremental\_orders.sql and add the following SQL:

-- models/incremental\_orders.sql

{{ config(

materialized='incremental', -- Specifies that this is an incremental model

unique\_key='order\_id' -- Specifies the unique key for the model

) }}

WITH new\_orders AS (

SELECT

order\_id,

order\_date,

customer\_id,

total\_amount,

created\_at,

updated\_at

FROM {{ source('raw', 'orders') }} -- Referencing the source table

{% if is\_incremental() %}

-- Only fetch records that are new or changed

WHERE updated\_at > (SELECT MAX(updated\_at) FROM {{ this }})

{% endif %}

)

SELECT \*

FROM new\_orders

**Explanation:**

* **materialized='incremental'**: This config setting ensures that the model is built incrementally rather than fully replacing the target table each time.
* **unique\_key='order\_id'**: This defines the order\_id column as the unique key for the model, ensuring that dbt knows how to uniquely identify each record.
* **is\_incremental()**: This macro is used to differentiate between a full refresh (when dbt runs the model for the first time) and subsequent incremental runs. When it's an incremental run, only records that have been updated since the last run (based on the updated\_at field) are included.
* **WHERE updated\_at > (SELECT MAX(updated\_at) FROM {{ this }})**: This condition ensures that only records where the updated\_at timestamp is greater than the maximum updated\_at in the target table (the model) are selected, thus fetching only new or updated data.

**Step 3: Run the Model**

Run the model to load the data into Snowflake. First, ensure that your Snowflake connection is set up in profiles.yml as explained earlier.

Run the following command to execute the incremental model:

dbt run --models incremental\_orders

* On the **first run**, this will load all data into the target table because there is no existing data.
* On **subsequent runs**, it will only load the new or updated records (based on the updated\_at timestamp) since the last dbt run.

**Step 5: Verify the Data**

After running the model, you can verify the data in the target table (incremental\_orders) by querying it in Snowflake:

SELECT

order\_id,

order\_date,

customer\_id,

total\_amount,

created\_at,

updated\_at

FROM your\_database.your\_schema.incremental\_orders;

**Step 6: Handling the First Time Run**

The first time you run an incremental model, dbt will perform a full load (it essentially performs a full refresh because there’s no previous data). On subsequent runs, only new or modified data will be added.

You can force a full-refresh (if needed) with the following command:

dbt run --models incremental\_orders --full-refresh

This will drop and recreate the target table, which can be useful when you want to fully reload the data.

**Summary**

* **Incremental Materialization**: The incremental materialization type only processes new or modified data, which is more efficient for large datasets.
* **is\_incremental() Macro**: This is used to apply conditional logic that fetches only new or changed data during incremental runs.
* **Unique Key**: The unique\_key setting ensures that dbt knows how to uniquely identify records in the table to handle updates or inserts properly.
* **Run dbt**: Use dbt run to apply your incremental model and only update the new or modified data in your Snowflake target table.

By following these steps, you can implement incremental loading of data into your Snowflake tables using dbt, which is both efficient and scalable for large datasets.

The ***is\_incremental()*** macro in dbt is an internal macro provided by dbt itself. It is automatically available for use in your dbt models to determine if the model is being run in an incremental mode or a full-refresh mode. You do not need to define this macro manually; dbt handles it for you.

**How is\_incremental() Works:**

* **During a Full Refresh**: When dbt is running the model and the --full-refresh flag is used (or the model hasn't been run before), is\_incremental() will return False, and the model will perform a full reload of the data.
* **During Incremental Runs**: When dbt is running the model incrementally (i.e., without --full-refresh), is\_incremental() will return True, and you can use it to specify that only new or changed records should be processed.

**Where to Use is\_incremental():**

You can use is\_incremental() in your SQL logic to modify the query behavior depending on whether dbt is doing a full refresh or an incremental update.

**Explanation of the Example:**

* The model is configured with materialized='incremental' and uses order\_id as the unique\_key.
* The is\_incremental() macro is used to check if the model is running in incremental mode.
  + If is\_incremental() is True, the model will only fetch records from the source where the updated\_at is greater than the maximum updated\_at value already present in the target table.
  + If is\_incremental() is False (i.e., on the first run or if --full-refresh is used), the query will include all records from the source table.

**Key Points:**

* **No need to define is\_incremental()**: This macro is automatically available in any dbt model, so you don't have to define or import it.
* **Use it in the model’s SQL logic**: The macro helps you write conditional logic depending on whether dbt is performing a full refresh or an incremental run.

**Full-Refresh Run:**

If you want to force a full-refresh of your model (i.e., reload the entire dataset), you can use the --full-refresh flag when running dbt run. In this case, is\_incremental() will return False, and the entire dataset will be reloaded.

dbt run --models incremental\_orders --full-refresh

In summary, ***is\_incremental()*** is a built-in macro in dbt that helps you define behavior for incremental models, making it easier to handle both full refreshes and incremental runs without needing to manually check or define the logic.