Okay, here's a summary of the important points about dbt's incremental materialization, especially for your Trino/Iceberg/Nessie setup, including the example in bullet points:

* **Purpose:**
  + Use incremental models for very large tables where rebuilding them completely on every dbt run is too slow or costly.
  + They allow dbt to process only **new or changed data** after the initial run.
* **How dbt Handles It:**
  + **First Run:** Behaves like materialized: 'table'; builds the entire table from scratch.
  + **Subsequent Runs:**
    - The is\_incremental() Jinja macro returns true.
    - You use {% if is\_incremental() %} in your SQL to add a WHERE clause. This clause filters your source data to select only the new/updated records.
    - The {{ this }} variable in your SQL refers to your existing target table, often used in the WHERE clause to determine what's new (e.g., WHERE event\_timestamp > (SELECT MAX(event\_timestamp) FROM {{ this }})).
* **Best Strategy for Iceberg/Trino:** merge  
  + Configure with incremental\_strategy='merge'.
  + This tells dbt to generate a Trino MERGE INTO ... statement.
  + Apache Iceberg is highly optimized for MERGE operations, making inserts and updates (upserts) very efficient and transactionally safe.
* **Example:** fct\_daily\_user\_activity.sql  
  + **Model Configuration (at the top of your SQL file):**
  + SQL

{{

config(

materialized='incremental',

incremental\_strategy='merge',

unique\_key=['user\_id', 'activity\_date'] -- Columns to match records for updates/inserts

)

}}

* + **SQL Logic to Filter New Data (simplified concept):**
  + SQL

-- (Your CTEs to get daily source activity here)

select

das.user\_id,

das.activity\_date,

das.number\_of\_events,

das.last\_event\_timestamp

from daily\_activity\_source das -- This is your CTE with the latest daily data

{% if is\_incremental() %}

-- This is the crucial part for subsequent runs:

-- Only process records from the source that are newer

-- than what's already in the target table for that user and day.

-- (The actual logic might involve comparing against max(last\_event\_timestamp)

-- from {{ this }} for that user\_id and activity\_date)

where das.last\_event\_timestamp > (

select coalesce(max(target\_table.last\_event\_timestamp), '1900-01-01')

from {{ this }} target\_table

where target\_table.user\_id = das.user\_id

and target\_table.activity\_date = das.activity\_date

)

-- OR handle completely new user/date combinations

or not exists (

select 1

from {{ this }} target\_table

where target\_table.user\_id = das.user\_id

and target\_table.activity\_date = das.activity\_date

)

{% endif %}

* + *(Note: The exact WHERE clause for incremental logic can vary based on your data and business rules. The key is to filter the incoming data based on what's already in {{ this }}.)*
* **How it Runs (Example Flow):**
  + **Run 1:** Full table fct\_daily\_user\_activity is created.
  + **New Day, New Data:** New user activity occurs.
  + **Run 2:** The {% if is\_incremental() %} block is active. Only data for new or updated user\_id/activity\_date combinations is processed. dbt generates a MERGE statement, and Trino efficiently updates the Iceberg table.
* **Benefit with Nessie:**
  + Each successful incremental dbt run that modifies data will create a new, distinct commit in your Nessie catalog. This gives you a full version history of how your large tables have evolved.

This incremental approach is a key technique for building scalable and maintainable data pipelines. I hope this summary helps as you wrap up your Monday night in Chennai!