**https://github.com/trinodb/trino/issues/17768**

**Nessie Catalogs Don't Support Views:** Your research on the Trino GitHub issue confirmed that the createView is not supported error is expected. Nessie is designed to version physical **Iceberg tables**, not logical SQL VIEWs, which is a known limitation of this architecture.

**The ephemeral Materialization is the Solution:** Because views are not supported by the catalog, the correct dbt strategy is to change your staging models' materialization from view to **ephemeral**. This solves the error because an ephemeral model is never created in the lakehouse; its code is simply injected as a subquery into downstream models.

**incremental Models are Powerful on Iceberg:** Your reading of the dbt documentation highlighted that for very large tables, the incremental materialization using the merge strategy is a state-of-the-art combination with Trino and Iceberg. Iceberg is highly optimized for these MERGE operations, making it very efficient.

Trino offers two incremental data loading strategies: **append**, which quickly adds new rows but doesn't handle updates, and **merge**, a more powerful strategy using Trino's MERGE command for atomic inserts and updates ("upserts").

{{

config(

materialized='incremental',

incremental\_strategy='merge',

-- We need a unique key to match records between the old and new data

unique\_key='signup\_month'

)

}}

select

date\_trunc('month', signup\_date) as signup\_month,

count(user\_id) as number\_of\_signups

from {{ ref('stg\_users') }}

{% if is\_incremental() %}

-- This WHERE clause is the key to making the run fast.

-- It tells dbt to only look at source records from the last 2 days.

-- This is because we only need to update the count for the current month.

where signup\_date >= date\_add('day', -2, current\_date)

{% endif %}

group by 1

https://docs.getdbt.com/reference/resource-configs/trino-configs#incremental

**A Global "Safety Switch" for Views:** You correctly identified the **views\_enabled: false** flag in the dbt documentation. Adding this to your dbt\_project.yml is an excellent additional step that tells dbt to never attempt creating views for any reason, making your project fully compatible with the lakehouse's limitations.

# In your dbt\_project.yml file

flags: views\_enabled: false

Of course. You've done an incredible job digging through some very dense, technical developer discussions. That text is the key to understanding the entire situation.

Here is a summary of what those developer comments mean, in simple points.

### **Key Points from Your Research**

* **The "Native" Nessie Connector is Problematic:** The first part of your text is a developer explaining that the way the iceberg.catalog.type = 'nessie' connector was originally built in Trino makes adding new features like view support very difficult. It would require a lot of complex work and code duplication. This is why the problem has existed for a long time.
* **The Future is the "REST Catalog" Mode:** The second part of the text reveals the solution the community is moving towards. They recommend that users should **stop using iceberg.catalog.type = 'nessie'**. Instead, you should connect to your Nessie server by setting **iceberg.catalog.type = 'rest'**. Nessie is fully compatible with this more standard protocol.
* **Trino is Likely "Abandoning" the Old Connector:** The developer comments suggest that the old, specialized "native" Nessie connector will likely be abandoned over time in favor of the more standard rest connector. Using type = 'rest' is the modern and future-proof way to connect Trino to Nessie.
* **The Catch (This Doesn't Solve Our Immediate Problem):** This is the most important point. Even though connecting via type = 'rest' is the better way, it **still does not magically enable view support**. As we discovered from the Nessie GitHub issue, the **Nessie server itself** does not yet have a way to store view definitions. This is a "two-sided problem": even if Trino's connector allowed views, the Nessie catalog has nowhere to put them.

**In our current Trino-Nessie setup, you also cannot create a MATERIALIZED VIEW.**

### **The Reason Why**

The reason is the same as for regular views, but even more fundamental.

A MATERIALIZED VIEW is an even more complex object than a standard VIEW. It requires the system to understand:

1. The SQL query to run.
2. Where to store the **physical data** that results from the query.
3. The special rules for **how and when to REFRESH** this data.

The Trino-Nessie connector is highly specialized. It is expertly built to manage the lifecycle of one specific object type: the **Apache Iceberg Table**. It does not have the built-in logic to handle the special commands (CREATE MATERIALIZED VIEW, REFRESH MATERIALIZED VIEW) or to store the extra metadata associated with a materialized view.

### 

### **The dbt Solution: materialized: 'table'**

This limitation is actually not a problem for us, because dbt provides its own, more flexible way to achieve the exact same result.

**In the dbt world, a model with materialized: 'table' *is* your materialized view.**

**in the dbt world, a model with materialized: 'table' *is* your materialized view.**

Think about the comparison:

| **Traditional Database** | **dbt Workflow** |
| --- | --- |
| 1 You run CREATE MATERIALIZED VIEW ... one time. | 1 You create a model .sql file with a SELECT statement. |
| 2. The database physically stores the data. | 2. You configure it with materialized: 'table'. |
| 3. You run REFRESH MATERIALIZED VIEW on a schedule to update the data. | 3\. You run **dbt run** on a schedule to update the data. |

The dbt run command becomes your refresh mechanism. By scheduling it to run every hour or every day, you are rebuilding your tables with the latest data, effectively doing the same job as a materialized view, but with the full power and control of dbt's testing and dependency graphing.