**Title: Apache Iceberg Views and Materialized View Behavior Using dbt (Easy English)**

**Slide 1: What is an Iceberg View?**

* A view in Iceberg is **not real data**.
* It stores **only the SQL query** and **output schema**.
* It is a **metadata file** (JSON format).
* Used to simplify complex queries and reuse logic.
* Query definition is executed whenever the view is queried.

**Slide 2: What is Stored in an Iceberg View?**

* SQL query (as a string)
* Output schema (column names and data types)
* Default catalog and namespace
* View version and view UUID
* Optional properties (comments, tags)

**Slide 3: How Iceberg View Works When Queried**

1. Query engine (like Trino or Spark) loads the view metadata.
2. It reads and parses the SQL stored inside.
3. Resolves table names using the catalog.
4. Executes the query live on the latest table versions.

**Slide 4: Properties of Iceberg Views**

* **Dynamic**: Executes fresh SQL every time.
* **Does not store data or snapshots**.
* Uses latest version of tables at runtime.
* Stored in JSON files with version history.

**Slide 5: Engine Interoperability and Dialect**

* Views include a dialect field (e.g., spark, trino).
* Spark cannot read views with dialect: trino and vice versa.
* Spec supports multi-dialect, but engines currently overwrite each other’s dialects.
* To make it engine-friendly:
  + Recreate view in both engines.
  + Or materialize as a shared Iceberg table.

**Slide 6: Based on Iceberg View Spec (**[**https://iceberg.apache.org/view-spec/**](https://iceberg.apache.org/view-spec/)**)**

* Iceberg view spec is **engine-agnostic by design**.
* It allows storing **multiple SQL dialects** (e.g., spark, trino, presto, flink, sql-standard) using representations.
* Each representation holds:
  + SQL string
  + Dialect name
  + Schema and version metadata
* However, **current engines like Spark and Trino only store their own dialect**, overwriting others.
* True multi-dialect support is possible but **not fully implemented yet** in engines.

**Slide 7: Example Iceberg View Metadata with Multiple Dialects and Execution Flow**

{  
 "view-uuid": "fa6...",  
 "format-version": 1,  
 "current-version-id": 2,  
 "versions": [...],  
 "schemas": [...],  
 "representations": [  
 {  
 "type": "sql",  
 "sql": "SELECT COUNT(1), CAST(event\_ts AS DATE)...",  
 "dialect": "spark"  
 },  
 {  
 "type": "sql",  
 "sql": "SELECT COUNT(1), CAST(event\_ts AS DATE)...",  
 "dialect": "trino"  
 },  
 {  
 "type": "sql",  
 "sql": "SELECT COUNT(1), CAST(event\_ts AS DATE)...",  
 "dialect": "sql-standard"  
 }  
 ]  
}

**Execution Flow in Trino:**

1. User queries the view: SELECT \* FROM analytics.daily\_events
2. Trino finds the Iceberg view metadata.
3. Parses the sql under dialect: trino.
4. Resolves all table names using Trino catalog (e.g., iceberg.analytics.events).
5. Builds a logical plan and executes the query.
6. Returns the result set to the user.

**Execution Flow in Spark:**

1. Spark session queries the view.
2. Finds and loads dialect: spark representation.
3. Resolves tables and compiles the query.
4. Executes the physical plan on Spark engine.

**Slide 8: Materialized Views - What & Why?**

* Stores **actual query results** on disk.
* Faster query performance.
* Must be refreshed (manually or automatically).
* Not natively supported in Iceberg (yet).
* Can be simulated using dbt.

**Slide 9: Simulating Materialized Views in dbt (with Iceberg)**

1. ``
   * Full rebuild each run (like CTAS).
2. ``
   * Only new/updated data is merged (efficient).

**Slide 10: dbt Incremental Example**

{{ config(  
 materialized='incremental',  
 table\_format='iceberg',  
 incremental\_strategy='merge',  
 unique\_key='event\_date'  
) }}  
  
SELECT CAST(event\_ts AS DATE) AS event\_date,  
 COUNT(\*) AS event\_count  
FROM {{ ref('events') }}  
{% if is\_incremental() %}  
 WHERE event\_ts > (SELECT MAX(event\_date) FROM {{ this }})  
{% endif %}  
GROUP BY CAST(event\_ts AS DATE);

**Slide 11: Summary**

* Iceberg views are **SQL + schema only** — no data.
* Can be queried across engines if dialect is understood.
* dbt can simulate materialized views:
  + table: full refresh.
  + incremental: efficient updates.
* Materialized views help for performance, caching, and BI reporting.

**Slide 12: Best Practices**

* Use Iceberg views for logic reuse.
* Use dbt table or incremental to simulate materialized views.
* Use CTAS for cross-engine compatibility.
* Monitor dialects when using multiple engines.

**Slide 13: Thank You**

* Questions?
* Try creating a view in Spark + dbt now!