



Dremio Software

Evaluating Reflection Usage

Introduction

The primary goal of this document is to familiarize users with the concept of reflections in Dremio and equip them with best practices for defining and managing reflections tailored to their specific workloads. Additionally, the document aims to raise awareness about potential pitfalls associated with overusing reflections.

What are Reflections?

Reflections are an optimized materialization of either source data or a query derived from an existing table or view. Reflections can accelerate a query against tables or views by using one or more reflections to partially or entirely satisfy the query. Further details can be found [here](#).

Purpose Of Reflections

The primary purpose of reflections is to accelerate the execution of end-user queries. Dremio uses reflections in the background and end users are not required to have explicit knowledge of reflections. Dremio uses them at runtime and intelligently leverages them during query planning time.

Reflection Types

Reflections come in different types, each serving specific purposes in optimizing query performance. The two primary types are Raw and Aggregate, and there is also a distinct category known as External Reflections.

1. Raw Reflections
 - a. Raw reflections contain detailed, row-level information from the source data. They include one or more fields, offering a granular representation of the original data.
 - b. Raw reflections can be partitioned based on one or more columns. Partitioning helps organize the data for more efficient retrieval and analysis.
2. Aggregate Reflections
 - a. Aggregate reflections, as the name suggests, aggregate data on a table/view based on the selected dimensions and specific aggregate functions for each measure.
 - b. Aggregate reflections are designed to provide summarized data views, optimizing queries that require aggregated information.
3. External Reflections
 - a. External reflections share similarities with Raw and Aggregate reflections but with a critical distinction - they are computed outside the Dremio environment. The materializations of external reflections are computed external to Dremio and then registered within Dremio as reflections.
 - b. This type of reflection allows for flexibility in computation, enabling external tools or systems to generate materializations that integrate into Dremio for query acceleration.

Review this [additional information](#) for a deeper understanding of the characteristics, implementation, and use cases for these reflection types.

Reflection System Tables

Reflection system tables play a crucial role in managing and organizing metadata associated with reflections. These tables serve as repositories for critical information, including details about reflections, their dependencies, materializations, and the refresh processes.

Table Name	Description
sys.reflections	Contains information such as the status, structure, display fields, partition fields, and other attributes that define the characteristics of each reflection.
sys.reflection_dependencies	Information about dependencies between different data sets and reflections is stored in this table. This helps in understanding the relationships and interdependencies within the system. It also helps to understand the order of refreshes.
sys.materializations	This table tracks every materialization job for a reflection. Details about the creation time, expiration times, and other attributes of reflection materializations are stored in these system tables. Materializations represent the actual precomputed results of reflections.
sys.refreshes	The table maintains information on the reflections' refresh process, including incremental refreshes, refresh job IDs, timestamps, the number of records affected, size, and other relevant metrics.

Materialization Storage

It's essential to note that while reflection system tables store metadata about reflections, the actual materializations - meaning the pre-computed results or aggregated data - are not stored within these tables. Instead, the materializations themselves are stored in the Dremio distributed store.

Inclusion in Backup and Restore Operations

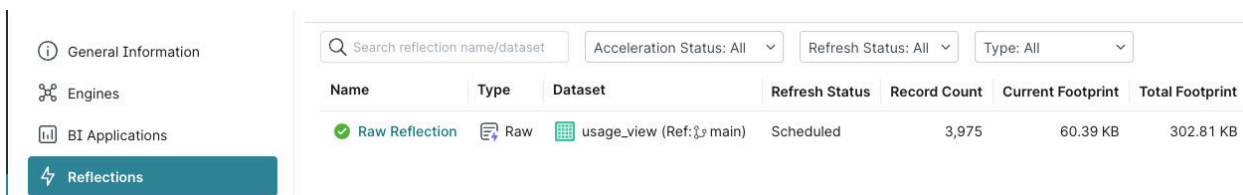
The reflection system tables are integral to Dremio's [backup and restore](#) operations. This inclusion ensures that the critical metadata related to reflections is preserved during these processes, allowing for seamless recovery and continuity in case of system failures, migrations, or other operational requirements.

Evaluate Reflection Usage

It is essential to evaluate the reflection usage strategies monthly. Users and Administrators are responsible for judiciously adding reflections when appropriate, assessing their impact, and removing unused reflections.

Evaluate Existing Reflections

Performing a thorough analysis of reflections and optimizing their usage is essential for improving query performance and resource efficiency. Leverage the information available on the reflection page to analyze the reflections. You can go to this page by clicking Settings (⚙️) and then Reflections. The page lists all the reflections. You can filter them based on Acceleration Status, Refresh Status and Type.



Name	Type	Dataset	Refresh Status	Record Count	Current Footprint	Total Footprint
Raw Reflection	Raw	usage_view (Ref: ⚡ main)	Scheduled	3,975	60.39 KB	302.81 KB

Specific key metrics play a crucial role when evaluating the effectiveness of reflections. Considered Count, Matched Count, and Accelerated Count provide valuable insights into how often reflections are employed in query acceleration. If these fields are not displayed by default, click the 'Manage Columns' button in the top-right of the reflections screen and select these columns. More details about them can be found [here](#).

Available Until	Considered Count	Matched Count	Accelerated Count	Refresh Job Histo
02/2023 13:40:28	9	4	4	History
04/3023 12:19:18	0	0	0	History

Dremio considers using a reflection if the queried view/table, parents, or children have reflections defined on them. Review Considered Count and Matched Count metrics for insights into why reflections aren't matched for specific queries.

One valid reason could be that additional fields/measures are included in the view that are not included in the reflection definition. Consider adding missing fields/measures to reflections for better matching. Explore consolidation opportunities for datasets with a high Considered to Matched ratio.

Assess Matched Count and Accelerated Count to identify scenarios where multiple reflections are eligible but not used based on query cost. Reviewing them allows one to identify duplicate or high-cost reflections.

The Reflections page provides extensive information about the reflections, including size, cardinality, last refresh timestamp, last refresh method, last refresh duration, expiration time etc.

You can use this information to identify the unmatched reflections for a query. For example, you ran a query and expected Dremio to use reflection, but it didn't. You can start with the reflection page and verify if it has been considered and matched. If not, you can go to the acceleration tab in the raw profile, which provides enough information to see why it wasn't matched. You can find the least used reflections that have a higher refresh time. You can review them and re-consider if reflection is necessary or not.

If you have requirements like integrating reflections summary with other tools like Splunk, you can extract the same information using Reflection Summary [APIs](#).

Evaluate Adding Reflections

When developing use cases in Dremio's semantic layer, it's often best to build out the use case iteratively without any reflections. Then, as you complete iterations, you can run the queries and analyze the data in the query history to deduce which ones take the longest to execute and whether there are any common factors between a set of slow queries contributing to the slowness.

For example, there may be a set of five slow queries, each derived from a view containing a join between two relatively large tables. In this situation, you might find that putting a raw reflection on the view performing the join helps speed up all five queries because a materialized representation of the results of the join will be created in an Apache Iceberg table format that can be automatically used to accelerate views derived from the join. This allows you to get the query planning and performance benefits of Apache Iceberg, and you can even partition the reflection to accelerate queries the underlying data wasn't initially optimized for.

This is a critical pattern since it means you can leverage a small number of reflections to speed up potentially many workloads.

Raw reflections can be helpful in cases where you have large volumes of JSON or CSV data. Whenever this type of data is queried, the entire data set must be processed, which can be inefficient. Adding a raw reflection over the JSON or CSV data again allows for an Apache

Iceberg representation of that data to be created, opening up all of the planning and performance benefits that come with it.

You can even consider creating multiple reflections on a dataset with different partitions. For example, two sets of users are querying the same employee table. Both are running heavy workloads; one uses the department field as a filter, and the other uses the grade field as a filter. You can create two reflections on the employee table with different partition schemes, one partitioning by department and the other by grade.

Similar to the JSON/CSV situation described above, another use of raw reflections is simply to offload heavy queries from an operational data store. Often, DBAs do not want their operational data stores (e.g. OLTP databases) being overloaded with analytical queries while they are busy processing billions of transactions, so in this situation, you can leverage Dremio raw reflections again to create that Apache Iceberg representation of the operational table and when a query comes in that needs the data it will read the reflection data as opposed to going back to the source.

Another critical use case often requiring raw reflections is joining on-premises data to cloud data. In this situation, you will typically find that the retrieval of the on-premises data becomes the bottleneck for queries due to the latency in retrieving the data from the source system; therefore, leveraging a raw reflection on the view where the data is joined together can almost always yield significant performance gains.

If you have connected Dremio to client tools and those client tools are issuing different sets of group-by queries against a view, if those group-by statements are taking too long to process compared to the desired SLAs, then you might want to consider adding an aggregate reflection to the view to satisfy the combinations of dimensions and measures being submitted from the client tool.

Dremio offers a powerful optimization feature called starflake reflections for queries involving multiple datasets and joins. This optimization is particularly beneficial when dealing with record-preserving joins, where each original record can be identified in the resulting dataset. For example, by creating a reflection on the joined datasets A, B, C, and D, Dremio can accelerate queries that include subsets of these joins. The starflake reflection eliminates the need to manually create reflections on combinations of joins, such as A,B,C; A,B; A,C; A,D; B,C, etc. Instead, a single reflection covers the commonality among these joins, making query execution more efficient. Starflake reflections are not limited to join operations; they can also apply to aggregate queries. This simplifies the management of reflections, as you don't need to create reflections for every combination.

For further best practices when considering how and where to apply reflections, visit [this page](#).

For detailed instructions on creating and updating reflections, visit [this page](#).

Failure to use Dremio reflection means you could miss out on significant performance enhancements for some of your poorest-performing queries. However, creating too many reflections can also have a negative impact on the system as a whole. The misconception is often that more reflections must be better, but when you consider the overhead in maintaining and refreshing them at intervals, the reflection refreshes can end up stealing valuable resources from your everyday workloads.

Where possible, organize your queries by pattern. The idea here is that you try to create as few reflections as possible to service as many of our queries as possible, so finding those points in our semantic tree where a lot of queries go through can help us accelerate a larger number of queries; the more reflections you have that may be able to accelerate the same query patterns the longer the planner will need to take evaluating which reflection will be best suited for accelerating the query being planned.

Evaluate Unused Reflections

The primary purpose of reflections is to accelerate queries. If reflections are unused in queries, they need to be reviewed. Unused reflections can be identified on the Reflections page with Accelerated Count = 0. Review the unused reflections and disable or delete them if they continue to be refreshed. If there's a possibility of future usage, consider disabling them; otherwise, delete them.

Checking for and removing unused reflections monthly is good practice because it can reduce clutter in the reflection configuration and often free up many hours of cluster execution cycles that can be used for more critical workloads.

Reflections Maintenance

Once created, reflections need to be maintained. For example, if the underlying table update rate changes or the business requires data no longer than 6 hours old when it used to be 24 hours, then the reflection settings may need to be adjusted accordingly.

Setup Isolated Engine

To refresh reflections, Dremio strongly recommends setting up a dedicated engine. Setting a dedicated engine isolates the reflection refreshes from other workloads and gives the flexibility to scale up or down based on the need.

Adjusting Frequency Refresh and Method

Reflections must be refreshed when their underlying table is loaded with fresh data and according to SLA requirements. When a reflection is active, Dremio considers using it even if it is not refreshed with new data. This may create obsolete reports (based on the SLA), so it is critical to refresh the reflections based on data updates and SLA requirements.

Reflections can be refreshed fully or incrementally. You can find further information on the reflection refreshes [here](#).

On-Demand Refresh

Reflections can be refreshed on demand. This policy is recommended in production environments. It gives the flexibility to refresh the reflection based on an event, such as when new data arrives for a dataset. As an example, your ETL job may run based on an event.

On-demand refresh can be accomplished using REST API calls, as described [here](#). This reflects all the reflections defined on the table. It is advised to run this command using the REST API and include it as a last step in your ETL pipeline. The on-demand refresh can also be performed in the Dremio UI on a specific reflection. If you are using this policy, make sure that you set the 'Never Refresh' option.

Scheduled Refresh

Reflections can be scheduled to refresh using Dremio UI. You can set the refresh frequency to every n hours/day(s)/week(s). For example, if you set the refresh to every 3 hours, the reflection will refresh 3 hours after the previous refresh attempt completes.

Use caution when using this policy in production environments. Instead, the recommendation is to use on-demand refresh. You can set the refresh frequency at the data source level and overwrite for a specific dataset.

Refresh Dependencies

When the table is refreshed (using Catalog API), Dremio refreshes all reflections dependent on it in hierarchical lineage order so that data is consistent.

Consider an example where you have a table and two views, and their hierarchical lineage is like Table1 → View1 → View2. If you have a reflection on Table1, View1 and View2, the order of the reflections refresh is Table1, View1, and then View2. This is to preserve the data consistency. Dremio automatically refreshes them in the correct order.

Reflections Expiration

Dremio provides an option to expire a reflection by time or manually. Dremio doesn't consider expired reflections in query acceleration. Setting the expiration time based on the data freshness SLAs is essential. For example, if a dataset gets new data every hour and the business data freshness requirement is at most three hours, set the expiration time to three hours.

Pitfalls of Reflections

Reflections are a powerful tool for accelerating queries. Use them wisely. Over-using them could cause wasted cluster resources and planning times may increase.

Cluster Usage

Review the reflections and their refresh frequencies compared to the frequency of the dataset updates. If your dataset changes once a day and you refresh the reflection defined on that dataset multiple times daily, that wastes cluster resources.

Planning Time

Dremio substitutes datasets with appropriate reflections based on the cost. As part of the substitution process, it considers all the reflections defined on the underlying datasets, generates alternate plans, and calculates the cost for the matched reflections. As the number of reflections increases, Dremio has to spend more time in reflection substitutions, generating alternate plans and cost calculations. This can increase the planning time.

Reflection Hints

Reflection Hints are designed to help you reduce the query planning time by providing functions to override the query planner choices in using reflections for query acceleration. If you can't avoid having many reflections, query planning time will increase. You can use hints to force the planner to consider/exclude particular reflections or to use no reflections. You can find more details [here](#).

As an example, you have tables that get updated frequently and create reflections on them with a frequency that meets most of your queries SLA. However, you have a few queries that need fresh data and would like to go to the tables directly (i.e. you don't want to use reflections due to freshness requirement). In such a scenario, for those few queries, you can use the `no_reflections` hint to prevent using reflections.

Summary

This document provides a comprehensive guide to understanding and effectively managing reflections. It covers various aspects, including the purpose of reflections, types, system tables, evaluation of reflection usage, adding and maintaining reflections, pitfalls, and hints for optimizing performance.